«Қостанай жоғары политехникалық колледжі» КМҚК
КГКП «Костанайский политехнический высший колледж»

Учебно-методический комплекс

ПМ03 «ПРОГРАММИРОВАНИЕ МОДУЛЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ»

ОГЛАВЛЕНИЕ

1 ТЕОРЕТИЧЕСКИЕ ЗАНЯТИЯ
РАЗДЕЛ 1. ОСНОВНЫЕ ПРИНЦИПЫ ПРОГРАММИРОВАНИЯ

Лекция 1. Жизненный цикл программы

В начале развития компьютерной индустрии проблема написания программ относилась к области искусства и состояла в том, чтобы получить "работающую" (доходящую до своего завершения) программу. Лишь затем программисты добивались от неё необходимых результатов путём длительной отладки и доработки. При этом было принято считать, что ни одна большая программа не может быть свободной от ошибок. Однако со временем появилось множество больших программ, которые должны были работать и работали годами, не выявляя никаких отклонений от требований. Но проходили годы, изменялась окружающая среда и изменялись требования. "Хорошие" работающие программы требовали модификации.

Важным толчком к совершенствованию методов разработки программ стало разделение пользователей на два класса: тех, кто пишет программы, и тех, кто решает с помощью этих программ прикладные задачи. Не будем забывать в первой группе так называемых системных программистов, которые пишут инструментальные программы (трансляторы, редакторы, операционные системы и т.п.) для других программистов. Таким образом программы стали отчуждаться от своих разработчиков. Они начали становиться самостоятельным продуктом.

Существенный качественный скачок стал возможен в семидесятых годах прошлого столетия. В 1969 году Эдсгер Дейкстра опубликовал статью "Структурное программирование". Это событие ознаменовало начало десятилетия напряжённого труда над инженерными методами, которые в корне изменили представления о подходах и тех возможностях, что открыты человеку в сфере разработки программного обеспечения (ПО). Благодаря структурному программированию и правильно поставленному процессу пошаговой разработки стало возможным создание действительно больших программных систем.

Расширение области применения вычислительной техники приводило к росту сложности ПО, ужесточению требования к надежности программ и, соответственно, росту времени построения программной системы и численности команды разработчиков. Возникла самостоятельная задача управления программными проектами, которая была разрешена путём построения методологии разработки программного обеспечения, включающей в себя описание процесса, организации работ, методов и инструментальных средств.

Общее описание жизненного цикла. Проектирование любой программной системы включает в себя несколько этапов или процессов, и чем лучше поставлено управление проектом, тем ярче они выражены. Методология проектирования программных систем описывает процесс создания и

сопровождения программного обеспечения в виде жизненного цикла (ЖЦ) разработки, представляя его как некоторую последовательность стадий и выполняемых на них процессов.

Для каждого этапа определяются состав и последовательность выполняемых работ, критерии начала и завершения этапа, получаемые результаты, методы и средства, необходимые для выполнения работ, роли и ответственность участников и т.д. Такое формальное описание ЖЦ разработки позволяет спланировать и организовать процесс коллективной работы, а также обеспечить управление этим процессом.

Понятия жизненного цикла и модели жизненного цикла несколько отличаются, однако термин "жизненный цикл" часто используется в смысле "модели жизненного цикла". Модель жизненного цикла разработки ПО - обобщенная структура, содержащая процессы, действия и задачи, которые осуществляются в ходе разработки, функционирования и сопровождения типового программного продукта в течение всей жизни системы, т.е. от определения требований до завершения ее использования.

На данный момент можно выделить следующие часто упоминаемые и используемые модели жизненного цикла:

- каскадная модель;
- процессная или поэтапная модель с промежуточным контролем;
- итерационная модель.

Каскадная модель (или, как ее еще называют, водопадная) (рис. 1) рассматривает последовательное выполнение всех этапов проекта в строго фиксированном порядке. Эта модель происходит от структуры диаграммы Ганта для поэтапного процесса. Переход на следующий этап означает полное завершение работ на предыдущем этапе. Модель подчеркивает, что результаты, полученные в ходе выполнения одного этапа, используются для выполнения следующего этапа.

Рис. 1. Каскадная (водопадная) модель ЖЦ разработки

Марри Кантор отмечает ряд важных аспектов, характерных для каскадной модели. Каскадная схема включает несколько важных операций, применимых ко всем проектам:

- составление плана действий по разработке системы;
- планирование работ, связанных с каждым действием;
- применение контрольных этапов отслеживания хода выполнения действий.

При разработке относительно простых программных систем каждое приложение представляло собой единый, функционально и информационно независимый блок. Для разработки такого типа приложений эффективным оказался каскадный способ. Каждый этап завершался после полного выполнения и документального оформления всех предусмотренных работ.

В результате можно выделить следующие положительные стороны применения каскадного подхода:

- результат каждого этапа - законченный документ, отвечающий критериям полноты и непротиворечивости;
- заранее заданная последовательность этапов упрощает задачу планирования и позволяет вести контроль сроков завершения каждого этапа.

Каскадный подход хорошо зарекомендовал себя при построении простых систем, когда в самом начале разработки можно достаточно точно и полно сформулировать все требования к системе. Основным недостатком этого подхода является то, что, как показывает практика, реальный процесс создания сложной системы никогда полностью не укладывается в такую жесткую схему из-за большой динамики корректировок и уточнений, которые могут поступать как в результате дальнейшего развития проекта, так и извне в качестве новых требований и уточнений.

"Основное заблуждение каскадной модели состоит в предположениях, что проект проходит через весь процесс один раз, архитектура хороша и проста в использовании, проект осуществления разумен, а ошибки в реализации устраняются по мере тестирования. Иными словами, каскадная модель исходит из того, что все ошибки будут сосредоточены в реализации, а потому их устранение происходит во время тестирования компонентов и системы".

Порой для улучшения характеристик каскадной модели в нее включают возможность возвратов на ранее выполненные этапы (рис. 2). Межэтапные корректировки позволяют учитывать реально существующее взаимовлияние результатов разработки на различных стадиях разработки. Даже на уровне модели видно, что самыми трудоемкими являются возвраты на уровень уточнения исходных требований.

Одна из проблем такой модели ЖЦ разработки заключается в отсутствии возможности оценить конечный результат до завершения всего процесса разработки ПО. Как показывает практика, будущие пользователи охотнее вносят изменения и уточнения в ходе оценки какого-либо прототипа системы (которого в данной модели ЖЦ не строится).

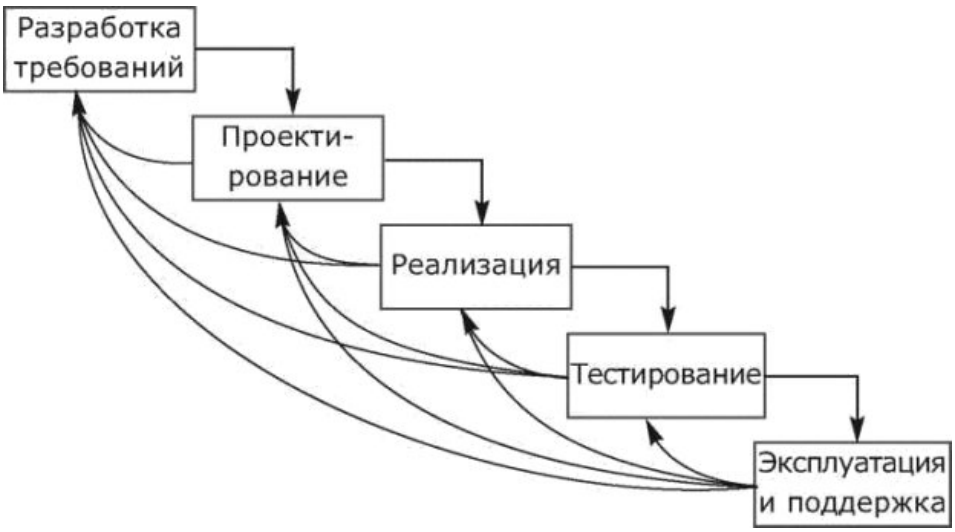


Рис. 2. Поэтапная модель с возвратами

Одним интересным вариантом развития поэтапной модели является V-образный жизненный цикл (рис. 3). В этой модели акцент делается на работы, связанные с верификацией документов разработки. Нисходящая ветвь модели описывает собственно разработку программной документации: требования к ПО, функции программных элементов, архитектуру - связи программных функций, программный код. Восходящая ветвь - этапы верификации.

Применение V-образной модели жизненного цикла позволяет сконцентрировать внимание на проверке результатов разработки, точно спланировать, какие свойства программного обеспечения будут исследоваться, на каких этапах и на основании какой документации. Обычно обращение к данному виду модели связано с повышенными требованиями к качеству результатов разработки. Именно такой подход к выбору модели жизненного цикла разработки характерен для бортовых авиационных систем, систем управления космическими аппаратами, разработки встроенного ПО.

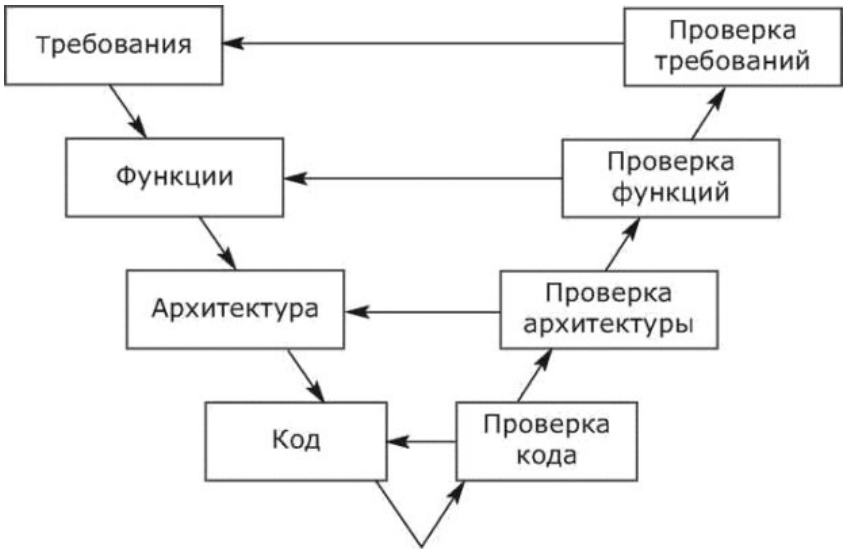


Рис. 3. V-образная модель жизненного цикла

Спиральная модель. Основная идея спиральной модели (рис. 4) заключается в том, что на этапах анализа и проектирования реализуемость технических решений и степень удовлетворения потребностей заказчика проверяется путём создания прототипов. Каждый виток спирали соответствует созданию работоспособного фрагмента или версии системы, в процессе анализа которых проводится конкретизация деталей проекта. В результате выбирается обоснованный вариант, который действительно удовлетворяет требованиям заказчика.

Итеративная разработка отражает объективно существующий спиральный цикл создания сложных систем. Она позволяет переходить на следующий этап, не дожидаясь полного завершения работы за счёт частичной реализации функциональности программного продукта. Тем самым активизируется процесс уточнения и дополнения требований со стороны пользователей системы.

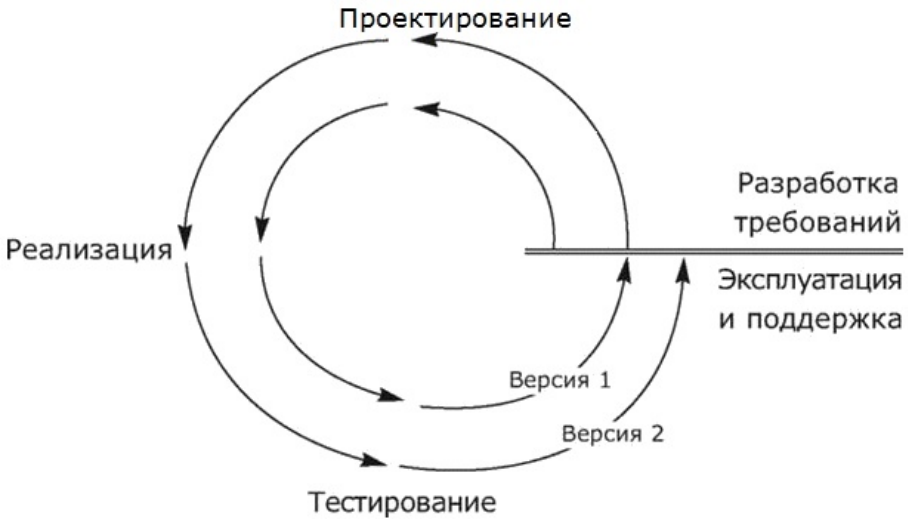


Рис. 4. Спиральная модель ЖЦ разработки

Процессная модель. Стоит различать понятие модели жизненного цикла и жизненный цикл конкретного программного продукта. Конкретный программный проект обычно использует различные модели ЖЦ для различных компонент программного продукта. Некоторые компоненты могут быть заимствованы из предшествующих проектов или приобретены на рынке ПО, другие проходят итерационный спиральный или каскадный ЖЦ. На практике мы имеем дело со множеством параллельно протекающих процессов разработки, каждый из которых может находиться в состоянии, отличном от других (рис. 5).

Так, первая компонента разрабатывается по классическому жизненному циклу: требования, проектирование, реализация, интеграция. Вторая компонента - по сокращенному жизненному циклу. Третья относится к разряду приобретаемых или заимствованных компонент и потому проходит интеграционные испытания сразу после определения к ней требований.

Разработка четвертой компоненты явно определяется спиральным ЖЦ,

в рамках которого последовательно реализуется два прототипа. На основании интеграционных исследований прототипов каждый раз производится уточнение требований. На третьем витке спирали, наконец, производится полномасштабное проектирование (создание документов описания архитектуры системы) и реализация окончательного варианта программной компоненты.

Применение процессной модели позволяет более точно отразить ход программной разработки, подчеркивая тот факт, что программный комплекс не однороден и содержит части, степень сложности которых сильно различается. Какие-то функции системы реализовались уже не один раз и могут использоваться практически без изменений в новых проектах. Другие требуют проведения исследовательских работ, многократного моделирования, оценки различных вариантов их реализации.

В общем случае следует отличать ЖЦ разработки ПО, который описывает именно процессы разработки, и ЖЦ самого ПО. В жизненном цикле проекта (разработки) присутствуют такие этапы, как формирование и обучение коллектива, закупка оборудования, создание стендов отладки и тестирования. ЖЦ программного продукта определяет, в свою очередь, процессы, связанные с функционированием самой программы, такие как:

- установка/настройка;
- эксплуатация;
- обновление;
- резервное копирование;
- временный останов;
- вывод из обращения.

Таким образом, можно и нужно говорить и о таком понятии, как ЖЦ проекта разработки ПО, который обычно на фазе инициализации включает в себя и подписание контракта с заказчиком, приобретение лицензий на инструментальное ПО, формирование среды информационной поддержки проекта, подбор и обучение персонала и т.п.

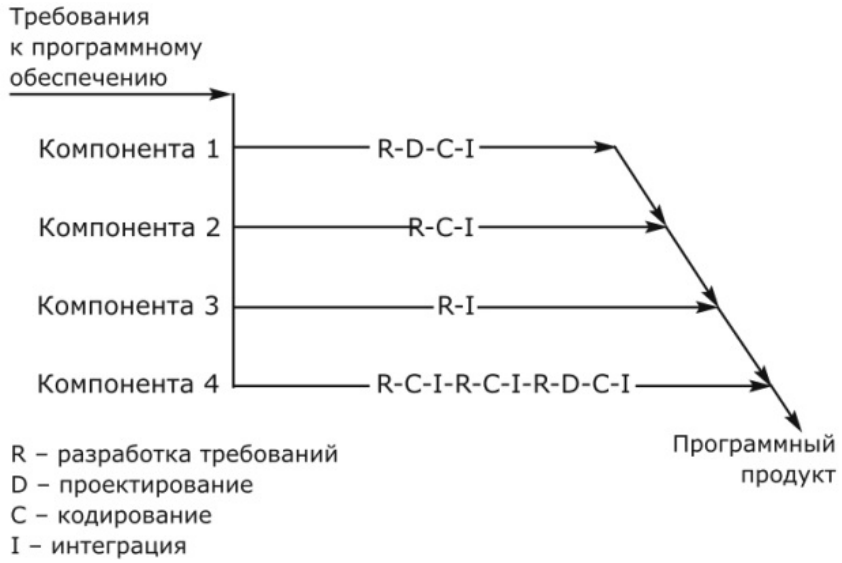


Рис. 5. Процессная модель разработки программного обеспечения

Формальные преобразования. Рассматривая разные подходы к процессу создания программного обеспечения, нельзя не упомянуть о таком, как метод формальных преобразований. Он заключается в автоматическом построении программы на основе ее формального описания, что гарантирует получение кода, который будет абсолютно точно соответствовать исходной спецификации.

Формальные преобразования — это набор методов, позволяющих математически корректно трансформировать исходные спецификации в код целевой программной системы. Т.к. переход от требований к коду происходит математически корректно, то проблема тестирования и доказательства корректности конечной программы по отношению к спецификации исчезает. Но верификация спецификаций по отношению к требованиям к системе остается.

Очевидно, что этот метод требует наличия формальных спецификаций, составление и доказательство корректности которых является нетривиальной задачей. Именно поэтому метод формальных преобразований редко используется для всего программного комплекса. В большинстве случаев он применяется для той части сложной системы и реализации тех алгоритмов, где исходные требования хорошо формализуемы.

Вопросы и задачи для самостоятельного решения

1. Что включает в себя понятие ЖЦ разработки?
2. Что такое ЖЦ программы?
3. Имеет ли преимущества поэтапная модель ЖЦ разработки по сравнению со спиральной? Если да, то какие?
4. Чем отличается процессная модель от поэтапной?
5. Какие основные этапы разработки программных систем вам известны?

Лекция 2. Простые и составные управляющие структуры

Основные структуры алгоритмов (ОСА) – это определенный набор блоков и стандартных способов их соединения для выполнения типичных последовательностей действий.

Любой алгоритм может быть записан с помощью трёх алгоритмических конструкций: последовательного выполнения команд (линейных алгоритмов), условных операторов и циклов.

Важно знать, что в языке Python синтаксис обладает следующей особенностью: дело в том, что в коде нет операторных скобок (begin..end или {...}); вместо них отступы указывают, какие операторы выполнять внутри той или иной конструкции.

Последовательность действий (линейные алгоритмы)

Алгоритм называется линейным, если он содержит N шагов, и все шаги выполняются последовательно друг за другом от начала до конца.

Для реализации алгоритмов линейной структуры используются следующие операторы:

1) оператор input() - осуществляет ввод данных;

2) оператор print() - осуществляет вывод данных;

3) оператор присваивания (=) - устанавливает связь между данными и переменными.

Последовательные действия описываются последовательными строками программы. Все операторы, входящие в последовательность действий, должны иметь один и тот же отступ. Например:

```
a = 1
b = 2
a = a + b
b = a - b
a = a - b
print (a, b)
```

Оператор условия и выбора (алгоритмы ветвления)

Алгоритм называется разветвляющимся, если последовательность выполнения шагов алгоритма изменяется в зависимости от выполнения некоторых условий. Условие — это логическое выражение, которое может принимать одно из двух значений: True - если условие верно (истинно), и False - если условие неверно (ложно).

В условиях используют знаки отношений: < (меньше), > (больше), <= (меньше или равно), >= (больше или равно), == (равно) и != (не равно).

В качестве условия в условном операторе можно указать любое логическое выражение, в том числе сложное условие, составленное из простых отношений с помощью логических операций (связок) «И», «ИЛИ» и «НЕ» (and, or и not).

Операторы сравнения в Python можно объединять в цепочки (в отличие от большинства других языков программирования, где для этого нужно использовать логические связки), например, a == b == c или 1 <= x <= 10.

Разветвляющийся алгоритм можно реализовать в программах с помощью простого, сокращенного, составного операторов, а также конструкции многозначных ветвлений.

Условная инструкция в Питоне имеет следующий синтаксис:

```
if Условие:
 Блок инструкций 1
else:
 Блок инструкций 2
```

Блок инструкций 1 будет выполнен, если Условие истинно. Если Условие ложно, будет выполнен Блок инструкций 2.

Обратите внимание, что слова if и else начинаются на одном уровне, а все команды внутренних блоков сдвинуты относительно этого уровня вправо на одно и то же расстояние (4 отступа).

В условной инструкции может отсутствовать слово else и последующий блок. Такая инструкция называется неполным ветвлением.

Внутри условного оператора могут находиться любые операторы, в том числе и другие условные операторы. Получаем вложенное ветвление – после одной развилки в ходе исполнения программы появляется другая развилка. При этом вложенные блоки имеют больший размер отступа (например, 8 пробелов).

```
if Условие 1:
 Блок инструкций 1
else:
 if Условие 2:
 Блок инструкций 2
else:
 Блок инструкций 3
```

Если нужно последовательно проверить несколько условий, используется форма с дополнительным оператором elif (сокращение от else if) - оператор выбора:

```
if условие 1:
 Блок инструкций 1
elif условие 2:
 Блок инструкций 2
else:
 Блок инструкций 3
```

Дополнительных условий и связанных с ними блоков elif может быть сколько угодно, но важно отметить, что в такой сложной конструкции будет выполнен всегда только один блок кода. Другими словами, как только некоторое условие оказалось истинным, соответствующий блок кода выполняется, и дальнейшие условия не проверяются,

Циклы

Циклы — это инструкции, выполняющие одну и ту же последовательность действий многократно.

В Python имеются два вида циклов: цикл ПОКА (выполняется некоторое условие) и цикл ДЛЯ (всех значений последовательности).

Цикл с условием (while)

Цикл while (“пока”) позволяет выполнить одну и ту же последовательность действий, пока проверяемое условие истинно.

Синтаксис цикла while в простейшем случае выглядит так:

while условие:
блок инструкции (тело цикла)

При выполнении цикла while сначала проверяется условие. Если оно ложно, то выполнение цикла прекращается и управление передается на следующую инструкцию после тела цикла while. Если условие истинно, то выполняется инструкция, после чего условие проверяется снова и снова выполняется инструкция.

Так продолжается до тех пор, пока условие будет истинно. Как только условие станет ложно, работа цикла завершится и управление передастся следующей инструкции после цикла. В языке Python тело цикла выделяется отступом.

Один шаг цикла (выполнение тела цикла) называют итерацией. Используют цикл while всегда, когда какая-то часть кода должна выполниться несколько раз, причем невозможно заранее сказать, сколько именно.

Цикл с переменной (for)

Цикл for, также называемый циклом с параметром, представляет собой цикл обхода заданного множества элементов и выполнения в своем теле различных операций над ними. Множество значений может быть задано списком, кортежем, строкой или диапазоном.

Как правило, циклы for используются либо для повторения какой-либо последовательности действий заданное число раз, либо для изменения значения переменной в цикле от некоторого начального значения до некоторого конечного.

Для повторения цикла некоторое заданное число раз можно использовать цикл for вместе с функцией range, синтаксис:

```
for ... in range(...):
 блок кода (тело цикла)
```

Range означает «диапазон», то есть for i in range(n) читается как «для (всех) i в диапазоне от 0 (включительно) до n (не включительно)...». Цикл выполняется n раз.

В скобках после слова range можно записать не одно, а два или три числа. Эти числа будут интерпретироваться как начальное значение итератора, конечное и его шаг (может быть отрицательным).

Если для range задано одно число, то итератор идет от 0 до заданного значения (не включая его).

Если задано два числа, то это начальное значение итератора и конечное, причем указанное конечное значение не входит в диапазон.

Если задано три числа, то это не только начальное и конечное значение итератора, но и шаг итератора. Например:

```
for i in range(4): # равносильно инструкции for i in 0, 1, 2, 3:
 print(i) # выведет на отдельных строчках числа от 0до 3
for i in range(1, 11):
 print(i) # выведет на отдельных строчках числа от 1до 10
for i in range(1, 11, 2):
 print(i) # выведет на отдельных строчках 1, 3, 5, 7, 9
for i in range(10, 0, -1):
 print(i) # выведет на отдельных строчках 10, 9, ..., 1
```

Если телом цикла является циклическая структура, то такие циклы называются вложенными. Цикл, содержащий в себе другой цикл, называют внешним, а цикл, содержащийся в теле другого цикла, называют внутренним.

Синтаксис операторов сложного цикла:

```
for i in range(Ni, N2): # внешний цикл
 for j in range(Mi, M2): # внутренний цикл
 тело цикла
```

Досрочное завершение цикла

Ходом выполнения цикла можно управлять, для этого применяются операторы break (прервать) и continue (продолжить) (рис. 6).

Оператор break прерывает выполнение цикла, управление передается операторам, следующим за оператором цикла.

Break - завершает выполнение цикла на определенном участке кода.

```
for i in 'a b v':
 if i=='b':
 break
```

*print(i*3)*
Результат:
aaa

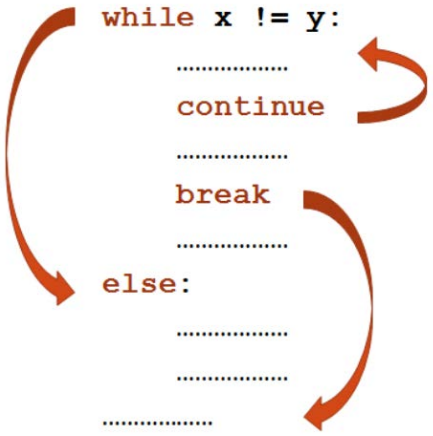


Рис. 6 - Операторы break и continue

Оператор continue прерывает выполнение очередного шага цикла и возвращает управление в начало цикла, начиная следующий шаг.

Continue - пропускает определенный участок кода.

for i in 'a b v':
if i=='b':
continue
*print(i*3)*
Результат:
aaa
vvv

Цикл, который начинается с заголовка while True будет выполняться бесконечно, потому что условие True всегда истинно. Выйти из такого цикла можно только с помощью оператора break.

while True:
print ("Введите положительное число:")
n = int (input())
if n > 0:
break

В данном случае он сработает тогда, когда станет истинным условие n > 0, то есть тогда, когда пользователь введет допустимое значение.

Исключения. Обработка исключений

Исключения (exceptions) - ещё один тип данных в Python. Часто в работе программы возникают ошибки, препятствующие её дальнейшему выполнению.

Для обработки особых ситуаций (таких как деление на ноль или попытка чтения из несуществующего файла) применяется механизм исключений. Исключения можно обрабатывать, для чего используется конструкция try–except.

Лучше всего пояснить синтаксис оператора try-except следующим примером:

```
try:
 res = int(open('a.txt').read()) / int(open('c.txt').read())
 print res
except IOError:
 print "Ошибка ввода-вывода"
except ZeroDivisionError:
 print "Деление на 0"
except KeyboardInterrupt:
 print "Прерывание с клавиатуры"
except:
 print "Ошибка"
```

В этом примере берутся числа из двух файлов и делятся одно на другое. В результате этих действий может возникнуть несколько исключительных ситуаций, некоторые из них отмечены в частях except (здесь использованы стандартные встроенные исключения Python). Последняя часть except в этом примере улавливает все другие исключения, которые не были пойманы выше. Например, если хотя бы в одном из файлов находится нечисловое значение, функция int() возбудит исключение ValueError. Его-то и сможет отловить последняя часть except.

Разумеется, выполнение части try в случае возникновения ошибки уже не продолжается после выполнения одной из частей except.

Отладка программы

Процесс написания программы состоит из двух этапов: кодирование и отладки.

В процессе отладки программы возможны различные ошибки.

Все ошибки можно условно разделить на следующие три категории (рис. 7):

1) ошибки, выявляемые препроцессором

В интерпретатор Python встроена специальная программа - препроцессор. У препроцессора несколько функций, в частности, он переводит текст программы в специальный байт-код, понятный для интерпретатора. В процессе перевода текста в байт-код препроцессор вынужден анализировать синтаксис вашей программы, для чего используется синтаксический анализатор,

проверяющий ваш текст с целью понять, похож ли он на текст программы на Python по ряду формальных признаков.

Если препроцессор не может понять смысл тех или иных символов в вашем тексте, он чаще всего указывает вам на ошибку типа (SyntaxError).

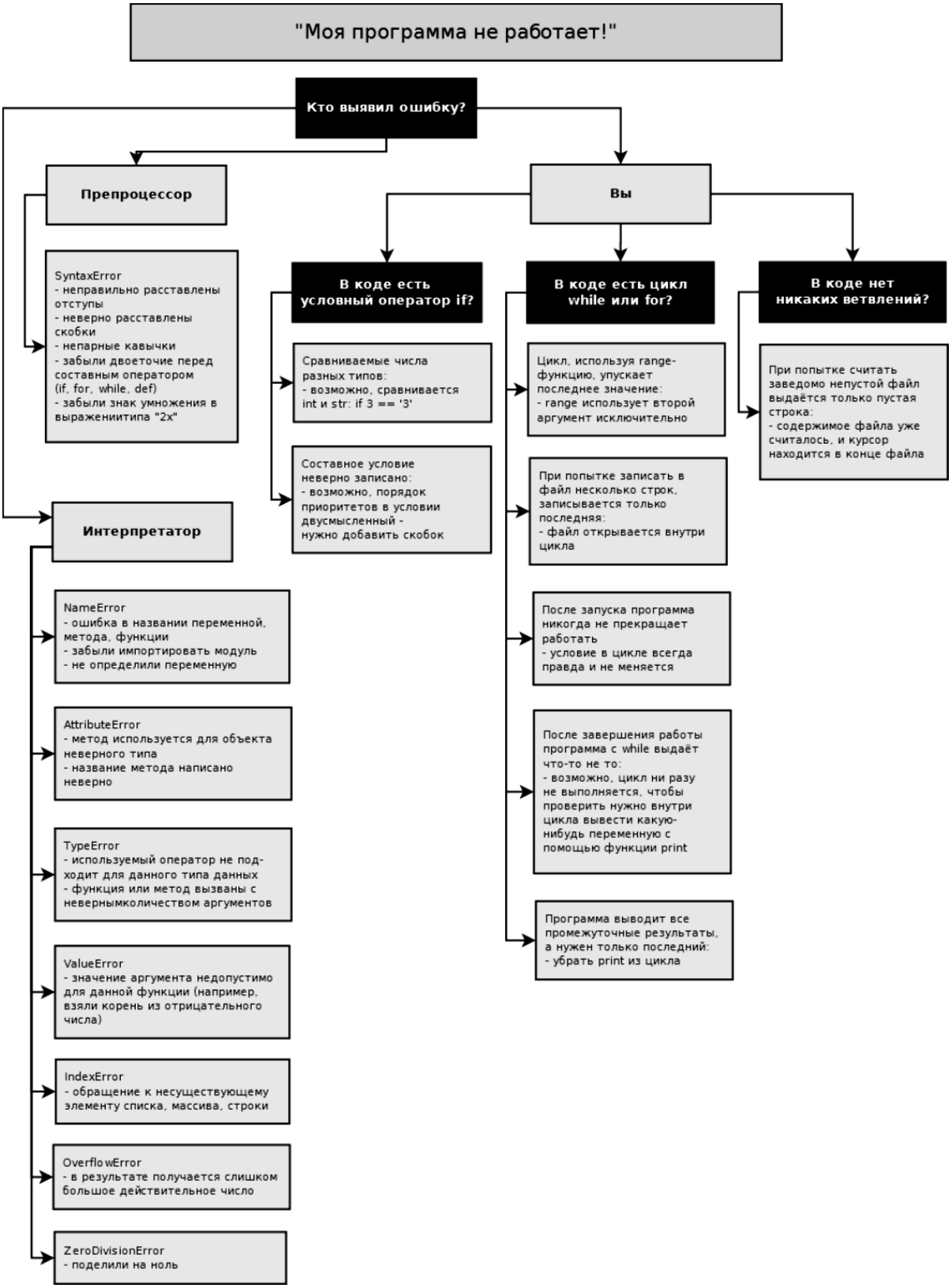


Рис. 7 - Возможные источники ошибок и алгоритм их нахождения

2) ошибки, выявляемые интерпретатором

Интерпретатор выявит ошибки во время исполнения программы и напишет, что это за ошибка и в какой она строке кода.

3) ошибки, выявляемые разработчиком

Их ещё можно назвать логическими. Это такие ошибки, когда ваша программа работает, но выдаёт что-то не то. Это наиболее сложный тип ошибок, потому что их нужно не только устранять, но и выявлять самостоятельно.

Лекция 3. Автоматизированные средства проектирования программного обеспечения

По определению, САПР — это организационно-техническая система, состоящая из совокупности комплекса средств автоматизации проектирования и коллектива специалистов подразделений проектной организации, выполняющая автоматизированное проектирование объекта, которое является результатом деятельности проектной организации.

Из этого определения следует, что САПР — это не средство автоматизации, а система деятельности людей по проектированию объектов. Поэтому автоматизация проектирования как научно-техническая дисциплина отличается от обычного использования ЭВМ в процессах проектирования тем, что в ней рассматриваются вопросы построения системы, а не совокупность отдельных задач. Эта дисциплина является методологической, поскольку она обобщает черты, являющиеся общими для разных конкретных приложений.

Идеальная схема функционирования САПР представлена на рис. 8.

Рис. 8. Схема функционирования САПР; КСА — комплекс технических средств

Эта схема идеальна в смысле полного соответствия формулировке согласно существующим стандартам и несоответствия реально действующим системам, в которых далеко не все проектные работы выполняются с помощью средств автоматизации, и не все проектировщики пользуются этими средствами.

Проектировщики, как следует из определения, относятся к САПР. Это утверждение вполне правомерно, т. к. САПР — это система автоматизированного, а не автоматического проектирования. Это значит, что часть операций проектирования может и всегда будет выполняться человеком. При этом в более совершенных системах доля работ, выполняемых человеком, будет меньше, но содержание этих работ будет более творческим, а роль человека в большинстве случаев — более ответственной.

Из определения САПР следует, что целью ее функционирования является проектирование. Как уже было сказано, проектирование — это процесс переработки информации, приводящий в конечном счете к получению полного представления о проектируемом объекте и способах его изготовления.

В практике неавтоматизированного проектирования полное описание проектируемого объекта и способов его изготовления содержит проект изделия и техническую документацию. Для условия автоматизированного проектирования еще не узаконено названия конечного продукта проектирования,

содержащего данные об объекте, и технологии его создания. На практике его называют по-прежнему "проектом".

Проектирование — это один из наиболее сложных видов интеллектуальной работы, выполняемой человеком. Более того, процесс проектирования сложных объектов не под силу одному человеку и выполняется творческим коллективом. Это, в свою очередь, делает процесс проектирования еще более сложным и трудно поддающимся формализации. Для автоматизации такого процесса необходимо четко знать, что в действительности он собой представляет и как выполняется разработчиками. Опыт свидетельствует, что изучение процессов проектирования и их формализация давались специалистам с большим трудом, поэтому автоматизация проектирования всюду осуществлялась поэтапно, охватывая последовательно все новые проектные операции. Соответственно, поэтапно создавались новые и совершенствовались старые системы. Чем на большее число частей разбита система, тем труднее правильно сформулировать исходные данные для каждой части, но тем легче провести оптимизацию.

Объектом автоматизации проектирования являются работы, действия человека, которые он выполняет в процессе проектирования. А то, что проектируют, называют объектом проектирования.

Человек может проектировать дом, машину, технологический процесс, промышленное изделие. Такие же объекты призвана проектировать САПР. При этом разделяют САПР изделия (САПР И) и САПР технологических процессов (САПР ТП).

Следовательно, объекты проектирования не являются объектами автоматизации проектирования. В производственной практике объектом автоматизации проектирования является вся совокупность действий проектировщиков, разрабатывающих изделие или технологический процесс, или то и другое, и оформляющих результаты разработок в виде конструкторской, технологической и эксплуатационной документаций.

Разделив весь процесс проектирования на этапы и операции, можно описать их с помощью определенных математических методов и определить инструментальные средства для их автоматизации. Затем необходимо рассмотреть выделенные проектные операции и средства автоматизации в комплексе и найти способы сопряжения их в единую систему, отвечающую поставленным целям.

При проектировании сложного объекта различные проектные операции многократно повторяются. Это связано с тем, что проектирование представляет собой закономерно развивающийся процесс. Начинается он с выработки общей концепции проектируемого объекта, на ее основе - эскизного проекта. Далее приближенные решения (прикидки) эскизного проекта уточняются на всех последующих стадиях проектирования. В целом такой процесс можно представить в виде спирали. На нижнем витке спирали находится концепция проектируемого объекта, на верхнем — окончательные данные о спроектированном объекте. На каждом витке спирали выполняют, с точки зрения

технологии обработки информации, идентичные операции, но в увеличивающемся объеме. Следовательно, инструментальные средства автоматизации повторяющихся операций могут быть одни и те же.

Практически решить в полном объеме задачу формализации всего процесса проектирования очень сложно, однако если будет автоматизирована хотя бы часть проектных операций, это себя все равно оправдает, т. к. позволит в дальнейшем развивать созданную САПР на основе более совершенных технических решений и с меньшими затратами ресурсов.

В целом для всех этапов проектирования изделий и технологии их изготовления можно выделить следующие основные виды типовых операций обработки информации:

- поиск и выбор из всевозможных источников нужной информации;
- анализ выбранной информации;
- выполнение расчетов;
- принятие проектных решений;
- оформление проектных решений в виде, удобном для дальнейшего использования (на последующих стадиях проектирования, при изготовлении или эксплуатации изделия).

Автоматизация перечисленных операций обработки информации и процессов управления использованием информации на всех стадиях проектирования составляет сущность функционирования современных САПР.

Каковы основные черты систем автоматизированного проектирования и их принципиальные отличия от "позадачных" методов автоматизации?

Первой характерной особенностью является возможность комплексного решения общей задачи проектирования, установления тесной связи между частными задачами, т. е. возможность интенсивного обмена информацией и взаимодействие не только отдельных процедур, но и этапов проектирования. Например, применительно к техническому (конструкторскому) этапу проектирования САПР РЭС позволяет решать задачи компоновки, размещения и трассировки в тесной взаимосвязи, которая должна быть заложена в технических и программных средствах системы.

Применительно к системам более высокого уровня можно говорить об установлении тесной информационной связи между схемотехническим и техническим этапами проектирования. Такие системы позволяют создавать радиоэлектронные средства, более эффективные с точки зрения комплекса функциональных и конструкторско-технологических требований.

Вторым отличием САПР РЭС является интерактивный режим проектирования, при котором осуществляется непрерывный процесс диалога "человек-машина". Сколь ни сложны и изощренны формальные методы проектирования, сколь ни велика мощность вычислительных средств, невозможно создать сложную аппаратуру без творческого участия человека. Системы автоматизации проектирования по своему замыслу должны не заменять конструктора, а выступать мощным инструментом его творческой деятельности.

Третья особенность САПР РЭС заключается в возможности имитационного моделирования радиоэлектронных систем в условиях работы, близких к реальным. Имитационное моделирование дает возможность предвидеть реакцию проектируемого объекта на самые различные возмущения, позволяет конструктору "видеть" плоды своего труда в действии без макетирования. Ценность этой особенности САПР заключается в том, что в большинстве случаев крайне трудно сформулировать системный критерий эффективности РЭС. Эффективность связана с большим числом требований различного характера и зависит от большого числа параметров РЭС и внешних факторов. Поэтому в сложных задачах проектирования практически невозможно формализовать процедуру поиска оптимального по критерию комплексной эффективности решения. Имитационное моделирование позволяет провести испытания различных вариантов решения и выбрать лучший, причем сделать это быстро и учесть всевозможные факторы и возмущения.

Четвертая особенность заключается в значительном усложнении программного и информационного обеспечения проектирования. Речь идет не только о количественном, объемном увеличении, но и об идеологическом усложнении, которое связано с необходимостью создания языков общения проектировщика и ЭВМ, развитых банков данных, программ информационного обмена между составными частями системы, программ проектирования. В результате проектирования создаются новые, более совершенные РЭС, отличающиеся от своих аналогов и прототипов более высокой эффективностью за счет использования новых физических явлений и принципов функционирования, более совершенной элементной базы и структуры, улучшенных конструкций и прогрессивных технологических процессов.

Принципы создания систем автоматизированного проектирования конструкции и технологии

При создании САПР руководствуются следующими общесистемными принципами:

1. Принцип включения состоит в том, что требования к созданию, функционированию и развитию САПР определяются со стороны более сложной системы, включающей в себя САПР в качестве подсистемы. Такой сложной системой может быть, например, комплексная система АСНИ — САПР — АСУТП предприятия, САПР отрасли и т. п.
2. Принцип системного единства предусматривает обеспечение целостности САПР за счет связи между ее подсистемами и функционирования подсистемы управления САПР.
3. Принцип комплексности требует связности проектирования отдельных элементов и всего объекта в целом на всех стадиях проектирования.
4. Принцип информационного единства предопределяет информационную согласованность отдельных подсистем и компонентов

САПР. Это означает, что в средствах обеспечения компонентов САПР должны использоваться единые термины, символы, условные обозначения, проблемно-ориентированные языки программирования и способы представления информации, которые обычно устанавливаются соответствующими нормативными документами. Принцип информационного единства предусматривает, в частности, размещение всех файлов, используемых многократно при проектировании различных объектов, в банках данных. За счет информационного единства результаты решения одной задачи в САПР без какой-либо перекомпоновки или переработки полученных массивов данных могут быть использованы в качестве исходной информации для других задач проектирования.

5. Принцип совместимости состоит в том, что языки, коды, информационные и технические характеристики структурных связей между подсистемами и компонентами САПР должны быть согласованы так, чтобы обеспечить совместное функционирование всех подсистем и сохранить открытую структуру САПР в целом. Так, введение каких-либо новых технических или программных средств в САПР не должно приводить к каким-либо изменениям уже эксплуатируемых средств.
6. Принцип инвариантности предусматривает, что подсистемы и компоненты САПР должны быть по возможности универсальными или типовыми, т. е. инвариантными к проектируемым объектам и отраслевой специфике. Применительно ко всем компонентам САПР это, конечно, невозможно. Однако многие компоненты, например программы оптимизации, обработки массивов данных и другие, могут быть сделаны одинаковыми для разных технических объектов.
7. Принцип развития требует, чтобы в САПР предусматривалось наращивание и совершенствование компонентов и связей между ними. При модернизации подсистемы САПР допускается частичная замена компонентов, входящих в подсистему, с изданием соответствующей документации.

Приведенные общесистемные принципы являются чрезвычайно важными на этапе разработки САПР. Контроль над их соблюдением обычно осуществляет специальная служба САПР предприятия.

Сущность процесса проектирования РЭС заключается в разработке конструкций и технологических процессов производства новых радиоэлектронных средств, которые должны с минимальными затратами и максимальной эффективностью выполнять предписанные им функции в требуемых условиях.

В результате проектирования создаются новые, более совершенные РЭС, отличающиеся от своих аналогов и прототипов более высокой эффективностью за счет использования новых физических явлений и принципов.

Контрольные вопросы и упражнения

1. Дайте определение САПР.
2. Что является целью функционирования САПР?
3. Что включает полный комплект документации при неавтоматизированном проектировании?
4. Что включает полный комплект документации при автоматизированном проектировании?
5. Что является объектом проектирования?
6. Что является объектом автоматизации проектирования?
7. В чем заключается сущность функционирования САПР?
8. Каковы основные черты современных САПР?
9. Какие преимущества дает имитационное моделирование?
10. Перечислите принципы создания САПР.
11. В чем заключается принцип информационного единства САПР?
12. В чем заключается принцип совместимости САПР?
13. Что значит "открытая структура САПР"?
14. Что означает "принцип инвариантности САПР"?
15. Что включает в себя понятие "Жизненный цикл промышленных изделий"?
16. Перечислите разновидности САПР.

РАЗДЕЛ 2. ЭЛЕМЕНТЫ И ПРИЕМЫ ПРОГРАММИРОВАНИЯ НА АППАРАТНОМ УРОВНЕ

Лекция 4. Классификация программного обеспечения. Основные показатели качества программного обеспечения

Программное обеспечение (ПО)– неотъемлемая часть компьютерной системы. Оно является логическим продолжением технических средств.

Уровни ПО (снизу вверх):

1. Базовое ПО – базовый уровень
2. Системное ПО – системный уровень
3. Служебное (сервисное) ПО
4. Прикладное ПО

Каждый вышележащий уровень повышает функциональность всей системы.

Структура ПО.

Базовое ПО – самый низкий уровень ПО. Базовое ПО отвечает за взаимодействие с базовыми аппаратными средствами. Как правило, базовые программные средства непосредственно входят в состав базового оборудования и хранятся в специальных микросхемах, называемых постоянными запоминающими устройствами (ПЗУ).

Базовое ПО в архитектуре компьютера занимает особое положение. С одной стороны, его можно рассматривать как составную часть аппаратных средств, с другой стороны, оно является одним из программных модулей операционной системы.

Базовое ПО, или BIOS, представляет программа, которая отвечает за управление всеми компонентами, установленными на материнской плате. Фактически BIOS является неотъемлемой составляющей системной платы и поэтому может быть отнесена к особой категории компьютерных компонентов, занимающих промежуточное положение между аппаратурой и программным обеспечением.

Функцией базового программного обеспечения является проверка состава и работоспособности вычислительной системы.

Совокупность программ системного уровня образуют ядро операционной системы (ОС) компьютера. Эти программы обеспечивают взаимодействие всех программ с программами базового уровня и непосредственно с аппаратным обеспечением, отвечают за взаимодействие с пользователем.

Служебное (сервисное) ПО

Основное назначение служебных программ (утилит) состоит в автоматизации работ по проверке, наладке и настройке компьютерной системы. Некоторые служебные программы (как правило, это программы обслуживания) изначально включаются в состав ОС, но большинство служебных программ являются для ОС внешними и служат для расширения и ее функций.

Это различные сервисные программы, используемые при работе или техническом обслуживании компьютера, — редакторы, отладчики, диагностические программы, архиваторы, программы для борьбы с вирусами и другие вспомогательные программы. Данные программы облегчают пользователю взаимодействие с компьютером. К ним примыкают программы, обеспечивающие работу компьютеров в сети. Они реализуют сетевые протоколы обмена информацией между машинами, работу с распределенными базами данных, телеобработку информации.

Классификация служебных программных средств

1. Средства диагностики Предназначены для автоматизации процесса диагностики аппаратного и программного обеспечения. Используются не только для устранения неполадок, но и для оптимизации работы компьютерной системы. Например, Утилита «Дефрагментация диска» позволяет данные, принадлежащие одному файлу, объединить в одной непрерывной области данных

2. Средства сжатия данных (архиваторы)

Предназначены для создания архивов. Архивирование данных упрощает их хранение за счет того, что большая группа файлов и каталогов сводятся в один архивный файл

Наиболее известными архиваторами являются WinZip, WinRAR, WinAce.

3. Средства обеспечения компьютерной безопасности

Это средства пассивной и активной защиты данных от повреждения, а также средства от несанкционированного доступа, просмотра и изменения данных.

Средства пассивной защиты – служебные программы, предназначенные для резервного копирования (нередко они обладают базовыми свойствами архиваторов).

Средства активной защиты – антивирусное программное обеспечение.

Для защиты данных от несанкционированного доступа, их просмотра и изменения служат специальные системы, основанные на криптографии.

4. Средства контроля (мониторинга)

Они позволяют следить за процессами. происходящими в компьютерной системе.

5. Диспетчеры файлов

Программы для выполнения большинства операций, связанных с обслуживанием файловой системы: копирование, перемещение и переименование файлов, создание каталогов (папок), удаление файлов и каталогов, поиск файлов, навигация в файловой структуре.

Наиболее популярными являются Total Commander (бывший Windows Commander) и FAR Manager.

6. Мониторы установки

Предназначены для контроля над установкой ПО.

7. Средства коммуникаций.

Они позволяют устанавливать соединение с удаленными компьютерами, обслуживают передачу сообщений электронной почты, работу с телеконференциями и др.

Прикладное ПО

Это комплекс прикладных программ, с помощью которых на данном рабочем месте выполняются конкретные задания. Это программы конечного пользователя, общего и специализированного назначения. Они предназначены для решения задач в конкретной предметной области

Классификация прикладных программных средств

1. Текстовые редакторы
2. Текстовые процессоры
3. Графические редакторы
4. Системы управления базами данных
5. Электронные таблицы
6. Системы автоматизированного проектирования
7. Настольные издательские системы
8. Экспертные системы
9. WEB-редакторы
10. Браузеры
11. Бухгалтерские системы

12. Геоинформационные системы
13. Интегрированные системы делопроизводства
14. Финансовые аналитические системы
15. Системы видеомонтажа

Инструментальное программное обеспечение представляет комплексы программ для создания других программ. Программы инструментального программного обеспечения управляются системными программами, поэтому они относятся к более высокому уровню. К инструментальному ПО относятся: компиляторы, редакторы связей, отладчики, интегрированные системы разработки ПО, например, интегрированная система Delphi.

Понятие системного программного обеспечения. Операционные системы.

Основу системного программного обеспечения составляют программы, входящие в операционные системы компьютеров.

ОС представляет собой комплекс системных и служебных программных средств.

С одной стороны она опирается на базовое ПО компьютера, входящее в состав BIOS (базовая система ввода-вывода), с другой стороны, она сама является опорой для ПО более высоких уровней – прикладных и большинства служебных приложений.

Все ОС обеспечивают свой автоматический запуск.

После включения компьютера производится самотестирование компьютера и затем загрузка операционной системы с системного диска в оперативную память. Загрузка должна выполняться в соответствии с программой загрузки. Однако для того чтобы компьютер выполнял какую-нибудь программу, эта программа должна уже находиться в оперативной памяти, а в момент включения компьютера в его оперативной памяти нет ничего, поскольку оперативная память не может ничего хранить без подзарядки ячеек. Разрешение этого противоречия состоит в последовательной, поэтапной загрузке операционной системы.

Самотестирование компьютера. В состав компьютера входит постоянное запоминающее устройство (ПЗУ), содержащее программы тестирования компьютера и первого этапа загрузки операционной системы, – это BIOS (Basic Input/Output System – базовая система ввода/вывода).

После включения питания компьютера процессор начинает выполнение программы самотестирования компьютера POST (Power-ON Self Test). Производится тестирование работоспособности процессора, памяти и других аппаратных средств компьютера.

Загрузка операционной системы. После проведения самотестирования специальная программа, содержащаяся в BIOS, начинает поиск загрузчика операционной системы. Происходит поочередное обращение к имеющимся в компьютере дискам (гибким, жестким, CD-ROM) и поиск на определенном

месте (в первом, так называемом загрузочном, секторе диска) наличия специальной программы Master Boot (загрузчика операционной системы).

Если установлен системный диск и программа-загрузчик оказывается на месте, то она загружается в оперативную память и ей передается управление работой компьютера. Программа ищет файлы операционной системы на системном диске и загружает их в оперативную память в качестве программных модулей. Если системные диски в компьютере отсутствуют, на экране монитора появляется сообщение «Non system disk», и компьютер «зависает», то есть загрузка операционной системы прекращается и компьютер остается неработоспособным. После окончания загрузки операционной системы управление передается командному процессору.

ОС предназначены для обеспечения нескольких видов интерфейса:

- интерфейса между пользователем и программно-аппаратными средствами компьютера (интерфейс пользователя);
- интерфейса между программным и аппаратным обеспечением (аппаратно-программный интерфейс);
- интерфейса между разными видами программного обеспечения (программный интерфейс).

Основные функции ОС:

- выполнение по запросу программ тех достаточно элементарных (низкоуровневых) действий, которые являются общими для большинства программ и часто встречаются почти во всех программах (ввод и вывод данных, запуск и остановка других программ, выделение и освобождение дополнительной памяти и др.);
- загрузка программ в оперативную память и их выполнение;
- стандартизованный доступ к периферийным устройствам (устройства ввода-вывода);
- управление оперативной памятью (распределение между процессами, организация виртуальной памяти);
- управление доступом к данным на энергонезависимых носителях (таких как жесткий диск, оптические диски и др.), организованным в той или иной файловой системе;
- обеспечение пользовательского интерфейса;
- сетевые операции, поддержка стека сетевых протоколов.

Дополнительные функции ОС:

- параллельное или псевдопараллельное выполнение задач (многозадачность);
- эффективное распределение ресурсов вычислительной системы между процессами;
- разграничение доступа различных процессов к ресурсам;
- организация надежных вычислений (невозможности одного вычислительного процесса намеренно или по ошибке повлиять на вычисления в другом процессе), основанная на разграничении доступа к ресурсам;

- взаимодействие между процессами: обмен данными, взаимная синхронизация;
- защита самой системы, а также пользовательских данных и программ от действий пользователей (злонамеренных или по незнанию) или приложений;
- многопользовательский режим работы и разграничение прав.

Современные ОС:

1. ОС семейства Windows - продукт корпорации Microsoft

Свою «родословную» Windows начинают от операционной системы DOS и первоначально представляли собой надстраиваемые над ней оболочки (Windows запускался из-под DOS), увеличивающие возможности DOS и облегчающие неподготовленному пользователю работу с компьютером. Уже более поздние версии (начиная с Windows NT) представляли собой полноценные операционные системы. Преимуществом Windows считается дружественный для пользователя интерфейс. Из недостатков отмечают ненадежность системы.

2. Unix-подобные ОС

Операционная система UNIX оказала большое влияние на развитие мира операционных систем, заложив основы работы современных ОС. Изначально UNIX был системой для разработки ПО. Несмотря на то, что Unix-подобные системы уступают по популярности Windows, они работают на больших типах компьютеров.

- Linux - представляет собой множество Unix-подобных операционных систем (дистрибутивов), которые чаще всего являются свободно распространяемыми
- MAC OS - также создавалась на основе ядра UNIX. Является продукт компании Apple для ее же компьютеров Macintosh. Считается надежной и удобной. Но в отличие от Windows не так популярна

Качество программного обеспечения

Можно определить качество ПО, как его пригодность и удобство для решения тех задач, для которых оно создано (так же, как и качество любого инструмента). Однако у программных систем есть две особенности, отличающие их, если не от всех, то от многих других инструментов, используемых человеком в своей деятельности.

- Во-первых, цели создания программной системы чаще всего сложны и включают множество аспектов. Программное обеспечение, за редким исключением, не разрабатывается для решения ровно одной задачи. Гораздо чаще это целый набор связанных задач из некоторой области. Кроме того, в него входят и экономическая эффективность использования данной системы, и удобство работы с ней для того персонала, который имеется у организации-заказчика.

- • Во-вторых, очень часто программы используются для решения несколько не тех задач, для которых они предназначались при создании. Набор целей, для достижения которых применяется данная система, изменяется со временем, что отражается и в постоянном добавлении новых возможностей и функций в новые версии программ.

Поэтому целостное и четкое понятие качества ПО определить очень нелегко. Вместо этого используют различные модели качества, систематизирующие набор аспектов, характеристик и метрик качества, рассмотрение которых необходимо для адекватной оценки качества разнообразных программ. Такие модели могут изменяться со временем, поскольку изменяются потребности индустрии производства ПО, появляются новые группы возможностей или внимание разработчиков привлекается к новым аспектам качества, ранее считавшимся несущественными.

Наиболее широко на данный момент используется модель качества ПО, зафиксированная в наборе стандартов ISO 9126. В несколько упрощенном виде (при рассмотрении так называемого внутреннего качества) эта модель определяет 6 основных характеристик качества программного обеспечения. Каждая характеристика уточняется при помощи некоторого набора более детальных атрибутов.

Рис. 9. Характеристики и атрибуты качества ПО по ISO 9126.

Функциональность. Эта характеристика обозначает способность ПО решать определенный круг задач. Функциональность определяет, что именно делает данная программа. Атрибуты функциональности, следующие: функциональная пригодность — способность решать нужный набор задач; точность выдаваемых результатов; защищенность — способность предотвращать доступ к функциям и данным ПО людям или другим системам, у которых нет прав на это; способность к взаимодействию с другими системами; и др.

Надежность. Это способность ПО поддерживать определенный уровень работоспособности в заданных условиях. Надежность является вероятностной характеристикой работоспособности ПО. Атрибуты ее таковы: зрелость — обратная величина к частоте отказов ПО; устойчивость к отказам, способность выполнять определенные задачи и придерживаться некоторых ограничений даже в случае отказов и сбоев; способность к восстановлению после отказов и среднее время такого восстановления; и др.

Удобство использования. Удобство использования показывает, насколько ПО привлекательно, удобно в обучении работе с ним и при выполнении самой работы. К атрибутам удобства использования относятся: понятность — показатель, обратный к усилиям, затрачиваемым пользователями на понимание основных понятий и способов работы ПО и их применимости для решения нужных им задач; удобство обучения, обратное к усилиям на обучение работе с системой; удобство работы, обратное к усилиям на выполнение определенного круга задач; привлекательность, способность привлекать новых пользователей; и др.

Производительность. Это способность ПО обеспечивать необходимую работоспособность по отношению к выделяемым для этого ресурсам. В соответствии с затратами ресурсов разного вида — времени, памяти, пропускной способности сетевых соединений — выделяются и различные атрибуты производительности.

Переносимость. Эта характеристика показывает сохранение работоспособности ПО при изменении его окружения. Ее атрибутами являются, например, возможность развертывания или установки ПО в различных окружениях и его адаптируемость — способность приспосабливаться к работе в различных окружениях при помощи действий, зафиксированных в документации.

Удобство сопровождения. Удобство сопровождения определяет трудоемкость анализа, исправления ошибок и внесения изменений в ПО. Его атрибутами являются, в частности, удобство проведения тестирования, удобство внесения изменений и риск возникновения неожиданных эффектов при изменениях.

Требования к программному обеспечению

Перечисленные характеристики качества ПО представляют собой одну из систематик различных видов требований к программным системам. Такая систематика полезна, если необходимо выделить и проанализировать требования для сложной системы. Она позволяет организовать анализ, разбивая

возможные требования и ограничения на разные группы и позволяя последовательно рассматривать различные аспекты системы. Сами требования как раз и говорят о том, какие конкретные свойства и характеристики должна иметь система, чтобы с ее помощью можно было успешно решать заданный набор задач.

Требования определяют, какое поведение системы является правильным и желаемым, а какое должно рассматриваться как некорректное. Поэтому они играют первостепенную роль при тестировании — именно они и проверяются с его помощью, а ошибки связаны именно с их нарушениями.

Сами требования определяются при анализе задач, стоящих перед рассматриваемой программной системой и ее окружения. Они могут извлекаться из различных источников.

- При аккуратном следовании стандартам на процессы разработки ПО требования обычно фиксируются в документе, создаваемом при решении о разработке системы и называемом техническим заданием (а по-английски — requirements specification, спецификация требований). Иногда этот документ создается представителями заказчика, но часто его приходится писать самим разработчикам на основе информации, полученной из других источников.
- Важным источником требований являются стандарты, регламентирующие характеристики, функции и состав систем, работающих в определенной предметной области. Такие стандарты создаются на основе опыта, накопленного в большом количестве проектов, в ходе которых такие системы создавались или развивались, и поэтому содержат ценную информацию о желательных характеристиках таких систем. Часто работа системы связана с выполнением каких-то действий, регулируемых существующим законодательством и нормами, действующими в данной области. Нормы и законы тоже являются разновидностью действующих стандартов, хотя обычно они формулируются менее четко и однозначно, чем технические стандарты и правила.
- Разработчики сами могут осознать, что что-то должно быть сделано определенным образом — так обычно возникают внутренние ограничения задач, не соблюдая которые, невозможно решить их правильно. Чаще всего ограничения такого рода можно усмотреть при детальном анализе решаемой задачи.
- Иногда ряд требований к новой системе можно сформулировать на основе анализа уже существующих систем для решения схожих задач.
- Большая часть требований формулируется на основе явно высказываемых пожеланий пользователей системы, их руководителей, заказчиков ее разработки и других заинтересованных лиц.
- Наконец, наиболее нечетким, но, тем не менее, достаточно важным источником требований являются невысказанные явно

потребности и нужды пользователей создаваемой системы, которые, несмотря на это, все же часто поддаются анализу.

При извлечении требований из перечисленных выше источников их четкость и согласованность возрастают при движении от потребностей и пожеланий к стандартам и техническому заданию.

Тестирование проверяет соответствие требованиям, и поэтому чем более точно и ясно они сформулированы, тем аккуратнее и полнее можно провести тестирование. Если какие-то требования определены нечетко, проверка их тоже может быть выполнена лишь с определенной степенью точностью, и системы, ведущие себя сильно по-разному, могут быть признаны удовлетворяющими ему в одинаковой мере. Получаемые при этом оценки качества таких систем будут во многом субъективными.

В некоторых случаях тестирование кажется возможным и в отсутствии всяких требований, по крайней мере, документально зафиксированных. В этих случаях, однако, используются какие-то свойства, которые считаются само собой разумеющимися и, соответственно, представляющие собой неявные требования. Пример такого свойства — отсутствие сбоев при работе программы. Если такие неявные требования действительно должны быть выполнены, можно проводить тестирование на их основе. Если же это не так, то есть иногда сбой может рассматриваться как корректное поведение системы (если, скажем, вводятся совсем некорректные исходные данные), то подобное тестирование может привести к неправильным выводам о наличии ошибок в системе.

Чтобы требования к программному обеспечению можно было уверенно использовать при его разработке и тестировании, они должны обладать рядом характеристик, которые зафиксированы в стандартах, регламентирующих разработку программного обеспечения.

Два таких стандарта — IEEE 830 [6] и IEEE 1233 [7] — определяют следующие характеристики правильно составленных требований к ПО.

- Адекватность, соответствие реальным потребностям пользователей ПО.
- Однозначность, отсутствие двусмысленностей и возможностей разного толкования.
- Полнота — отражение в требованиях всех существенных потребностей и всех ситуаций, в которых система должна будет функционировать.
- Непротиворечивость или согласованность между разными элементами требований.
- Систематичность представления — требования должны быть описаны в рамках некоторой системы с четким указанием места каждого требования среди остальных, с определением связей и зависимостей между ними и приоритетности для различных заинтересованных лиц.
- Прослеживаемость — требования должны иметь четко определенные связи с модулями разрабатываемой системы, частями

проектной документации и тестами, чтобы всегда можно было определить, для выполнения или проверки каких требований создан каждый из этих элементов и насколько он им соответствует.

- Проверяемость или возможность для каждого требования однозначно установить при помощи некоторых действий, выполнено это требование или нет.
- Модифицируемость или возможность внесения изменений в набор требований с максимально быстрым отслеживанием последствий такой модификации и исправлением всех возникающих при этом дефектов с точки зрения других характеристик.

В ходе работы над создаваемой программной системой или переработки уже существующей всегда, в том или ином виде, проводится анализ требований, цель которого — подготовить представление требований к ПО, имеющее все указанные характеристики. Анализ требований обычно включает следующие виды деятельности.

- Выделение требований. Его задача — определить полный список требований, уточнить недостаточно четко сформулированные и определить возможные компромиссы в тех случаях, когда различные заинтересованные лица высказывают противоречащие друг другу требования. Выделение требований включает определение доступных источников требований, извлечение требований из них, в ходе чего, собственно, и фиксируются отдельные требования, и согласование требований, полученных от разных источников, при возникновении необходимости в этом. При извлечении требований может применяться широкий диапазон различных техник — от простого анализа задач, анализа имеющихся документов, до проведения интервью, опросов и семинаров, с использованием специальных методов для фиксации как высказываемых пожеланий и формулировок, так и эмоционального состояния опрашиваемых, чтобы в дальнейшем оценить правдивость и полноту предоставленных сведений.
- Систематизация и описание требований, их сведение в единую систему и составление представляющих ее моделей, отражающих различные аспекты собранных требований. При систематизации особое внимание уделяется полному отражению всех извлеченных сведений в требованиях, определению связей и зависимостей между требованиями, идентификации требований, необходимой, чтобы иметь возможность ссылаться на них из различных проектных документов.
- Валидация и верификация требований. Задача этих видов деятельности — проверка необходимых свойств требований к ПО. Валидация представляет собой проверку адекватности и полноты требований, то есть проверяет, что зафиксированные в требованиях ограничения действительно представляют потребности

пользователей, заказчиков и других заинтересованных лиц, а также, что все их существенные потребности нашли соответствующее отражение в требованиях. Верификация проверяет внутреннюю согласованность, непротиворечивость и однозначность требований, а также их проверяемость и возможность проследить связи требований друг с другом, с кодом, тестами и другими проектными документами.

РАЗДЕЛ 3. РАЗРАБОТКА СПЕЦИФИКАЦИЙ ДЛЯ КОМПОНЕНТОВ ПРОГРАММНОГО ПРОДУКТА

Лекция 5. Объектно-ориентированное программирование. Основные понятия, принципы объектно-ориентированного программирования

Python проектировался как объектно-ориентированный язык программирования. Это означает, что он построен с учетом следующих принципов:

1. Все данные в нем представляются объектами.
2. Программу можно составить как набор взаимодействующих объектов, посылающих друг другу сообщения.
3. Каждый объект имеет собственную часть памяти и может состоять из других объектов.
4. Каждый объект имеет тип.
5. Все объекты одного типа могут принимать одни и те же сообщения (и выполнять одни и те же действия).

Основные понятия

При процедурном программировании программа разбивается на части в соответствии с алгоритмом: каждая часть (подпрограмма, функция, процедура) является составной частью алгоритма.

При объектно-ориентированном программировании программа строится как совокупность взаимодействующих объектов.

С точки зрения объектно-ориентированного подхода, объект — это нечто, обладающее значением (состоянием), типом (поведением) и индивидуальностью. Когда программист выделяет объекты в предметной области, он обычно абстрагируется (отвлекается) от большинства их свойств, концентрируясь на существенных для задачи свойствах. Над объектами можно производить операции (посылая им сообщения). В языке Python все данные представлены в виде объектов.

Взаимодействие объектов заключается в вызове методов одних объектов другими. Иногда говорят, что объекты посылают друг другу сообщения. Сообщения — это запросы к объекту выполнить некоторые действия. (Сообщения, методы, операции, функции-члены являются синонимами).

Каждый объект хранит свое состояние (для этого у него есть атрибуты) и имеет определенный набор методов. (Синонимы: атрибут, поле, слот, объект-член, переменная экземпляра). Методы определяют поведение объекта. Объекты класса имеют общее поведение.

Объекты описываются не индивидуально, а с помощью классов. Класс - объект, являющийся шаблоном объекта. Объект, созданный на основе некоторого класса, называется экземпляром класса. Все объекты определенных пользователем классов являются экземплярами класса. Тем не менее, объекты даже с одним и тем же состоянием могут быть разными объектами. Говорят, что они имеют разную индивидуальность.

В языке Python для определения класса используется оператор class:

class имя_класса(класс1, класс2, ...):
 # определения методов

Класс определяет тип объекта, то есть его возможные состояния и набор операций.

Абстракция и декомпозиция

Абстракция в ООП позволяет составить из данных и алгоритмов обработки этих данных объекты, отвлекаясь от несущественных (на некотором уровне) с точки зрения составленной информационной модели деталей. Таким образом, программа подвергается декомпозиции на части "дозированной" сложности. Отдельный объект, даже вместе с совокупностью его связей с другими объектами, человеком воспринимается легче (именно так он привык оперировать в реальном мире), чем что-то неструктурированное и монотонное.

Перед тем как начать написание даже самой простенькой объектно-ориентированной программы, необходимо провести анализ предметной области, для того чтобы выявить в ней классы объектов.

При выделении объектов необходимо абстрагироваться (отвлечься) от большинства присущих им свойств и сконцентрироваться на свойствах, значимых для задачи.

Выделяемые объекты необязательно должны походить на физические объекты — ведь это абстракции, за которыми скрываются процессы, взаимодействия, отношения.

Удачная декомпозиция стоит многого. От нее зависят не только количественные характеристики кода (быстродействие, занимаемая память), но и трудоемкость дальнейшего развития и сопровождения. При отсутствии соответствующего опыта лучше не загадывать будущих путей развития программы, а делать ее как можно проще, под конкретную задачу.

Даже если просто перечислить все существительные, встретившиеся в описании задачи (явно или неявно), получится неплохой список кандидатов в классы.

При процедурном подходе тоже используется декомпозиция, но при объектно-ориентированном подходе производится декомпозиция не самого алгоритма на более мелкие части, а предметной области на классы объектов.

Объекты

До этой лекции объекты Python встречались много раз: ведь каждое число, строка, функция, модуль и т.п. — это объекты. Некоторые встроенные объекты имеют в Python синтаксическую поддержку (для задания литералов). Таковы числа, строки, списки, кортежи и некоторые другие типы.

Теперь следует посмотреть на них в свете только что приведенных определений. Пример:

```
a = 3
b = 4.0
c = a + b
```

Здесь происходит следующее. Сначала имя " a " связывается в локальном пространстве имен с объектом-числом 3 (целое число). Затем " b " связывается с объектом-числом 4.0 (число с плавающей точкой). После этого над объектами 3 и 4.0 выполняется операция сложения, и имя " c " связывается с получившимся объектом. Кстати, операциями, в основном, будут называться методы, которые имеют в Python синтаксическую поддержку, в данном случае - инфиксную запись. То же самое можно записать как:

```
c = a.__add__(b)
```

Здесь __add__() - метод объекта a, который реализует операцию + между этим объектом и другим объектом.

Узнать набор методов некоторого объекта можно с помощью встроенной функции dir():

```
>>> dir(a)
['__abs__', '__add__', '__and__', '__class__', '__cmp__', '__coerce__',
'__delattr__', '__div__', '__divmod__', '__doc__', '__float__',
'__floordiv__', '__getattribute__', '__getnewargs__', '__hash__',
'__hex__', '__init__', '__int__', '__invert__', '__long__',
'__lshift__', '__mod__', '__mul__', '__neg__', '__new__',
'__nonzero__', '__oct__', '__or__', '__pos__', '__pow__',
'__radd__', '__rand__', '__rdiv__', '__rdivmod__', '__reduce__',
'__reduce_ex__', '__repr__', '__rfloordiv__', '__rlshift__',
'__rmod__', '__rmul__', '__ror__', '__rpow__', '__rrshift__',
'__rshift__', '__rsub__', '__rtruediv__', '__rxor__',
'__setattr__', '__str__', '__sub__', '__truediv__', '__xor__']
```

Здесь стоит указать на еще одну особенность Python. Не только инфиксные операции, но и встроенные функции ожидают наличия некоторых методов у объекта. Например, можно записать:

```
abs(c)
```

А функция abs() на самом деле использует метод переданного ей объекта:

```
c.__abs__()
```

Объекты появляются в результате вызова функций-фабрик или конструкторов классов (об этом ниже), а заканчивают свое существование при

удалении последней ссылки на объект. Оператор del удаляет имя (а значит, и одну ссылку на объект) из пространства имен:

```
a = 1
# ...
del a
# имени a больше нет
```

Типы и классы

Тип определяет область допустимых значений объекта и набор операций над ним. В ООП тип тесно связан с поведением - действиями объекта, состоящими в изменении внутреннего состояния и вызовами методов других объектов.

Ранее в языке Python встроенные типы данных не являлись экземплярами класса, поэтому считалось, что это были просто объекты определенного типа. Теперь ситуация изменилась, и объекты встроенных типов имеют классы, к которым они принадлежат. Таким образом, тип и класс в Python становятся синонимами.

Интерпретатор языка Python всегда может сказать, к какому типу относится объект. Однако с точки зрения применимости объекта в операции его принадлежность к классу не играет решающей роли: гораздо важнее, какие методы поддерживает объект.

Экземпляры классов могут появляться в программе не только из литералов или в результате операций. Обычно для получения объекта класса достаточно вызвать конструктор этого класса с некоторыми параметрами. Объект-класс, как и объект-функция, может быть вызван. Это и будет вызовом конструктора:

```
>>> import sets
>>> s = sets.Set([1, 2, 3])
```

В этом примере модуль sets содержит определение класса Set. Вызывается конструктор этого класса с параметром [1, 2, 3]. В результате с именем s будет связан объект-множество из трех элементов 1, 2, 3.

Следует заметить, что, кроме конструктора, определенные классы имеют и деструктор - метод, который вызывается при уничтожении объекта. В языке Python объект уничтожается в случае удаления последней ссылки на него либо в результате сборки мусора, если объект оказался в неиспользуемом цикле ссылок. Так как Python сам управляет распределением памяти, деструкторы в нем нужны очень редко. Обычно в том случае, когда объект управляет ресурсом, который нужно корректно вернуть в определенное состояние.

Еще один способ получить объект некоторого типа - использование функций-фабрик. По синтаксису вызов функции-фабрики не отличается от вызова конструктора класса.

Определение класса

Пусть в ходе анализа данной предметной области необходимо определить класс Граф. Граф — это множество вершин и набор ребер, попарно соединяющий эти вершины. Над графом можно проделывать операции, такие как добавление вершины, ребра, проверка наличия ребра в графе и т.п. На языке Python определение класса может выглядеть так:

```
from sets import Set as set # тип для множества
class G:
def __init__(self, V, E):
self.vertices = set(V)
self.edges = set(E)
def add_vertex(self, v):
self.vertices.add(v)
def add_edge(self, (v1, v2)):
self.vertices.add(v1)
self.vertices.add(v2)
self.edges.add((v1, v2))
def has_edge(self, (v1, v2)):
return (v1, v2) in self.edges
def __str__(self):
return "%s; %s" % (self.vertices, self.edges)
Использовать класс можно следующим образом:
g = G([1, 2, 3, 4], [(1, 2), (2, 3), (2, 4)])
print g
g.add_vertex(5)
g.add_edge((5,6))
print g.has_edge((1,6))
print g
что даст в результате
Set([1, 2, 3, 4]); Set([(2, 4), (1, 2), (2, 3)])
False
Set([1, 2, 3, 4, 5, 6]); Set([(2, 4), (1, 2), (5, 6), (2, 3)])
```

Как видно из предыдущего примера, определить класс не так уж сложно. Конструктор класса имеет специальное имя __init__. (Деструктор здесь не нужен, но он бы имел имя __del__.) Методы класса определяются в пространстве имен класса. В качестве первого формального аргумента метода принято использовать self. Кроме методов в объекте класса имеются два атрибута: vertices (вершины) и edges (ребра). Для представления объекта G в виде строки используется специальный метод __str__().

Принадлежность классу можно выяснить с помощью встроенной функции isinstance():

print isinstance(g, G)

Инкапсуляция

Обычно считается, что без инкапсуляции невозможно представить себе ООП, что это ключевое понятие. История развития методологий программирования движима борьбой со сложностью разработки программного обеспечения. Сложность больших программных систем, в создании которых участвует сразу большое количество разработчиков, уменьшается, если на верхнем уровне не видно деталей реализации нижних уровней. Собственно, процедурный подход был первым шагом на этом пути. Под инкапсуляцией (encapsulation, что можно перевести по-разному, но на нужные ассоциации хорошо наводит слово "обволакивание") понимается сокрытие информации о внутреннем устройстве объекта, при котором работа с объектом может вестись только через его общедоступный (public) интерфейс. Таким образом, другие объекты не должны вмешиваться в "дела" объекта, кроме как используя вызовы методов.

В языке Python инкапсуляции не придается принципиального значения: ее соблюдение зависит от дисциплинированности программиста. В других языках программирования имеются определенные градации доступности методов объекта.

Доступ к свойствам

В языке Python не считается зазорным получить доступ к некоторому атрибуту (не методу) напрямую, если, конечно, этот атрибут описан в документации как часть интерфейса класса. Такие атрибуты называются свойствами (properties). В других языках программирования принято для доступа к свойствам создавать специальные методы (вместо того чтобы напрямую обращаться к общедоступным членам-данным). В Python достаточно использовать ссылку на атрибут, если свойство ни на что в объекте не влияет (то есть другие объекты могут его произвольно менять). Если же свойство менее тривиально и требует каких-то действий в самом объекте, его можно описать как свойство (пример взят из документации к Python):

```
class C(object):
def getx(self): return self.__x
def setx(self, value): self.__x = value
def delx(self): del self.__x
x = property(getx, setx, delx, "I'm the 'x' property.")
```

Синтаксически доступ к свойству x будет обычной ссылкой на атрибут:

```
>>> c = C()
>>> c.x = 1
>>> print c.x
```

```
1
>>> del c.x
```

А на самом деле будут вызываться соответствующие методы: setx(), getx(), delx().

Следующий небольшой пример демонстрирует все перечисленные моменты. В этом примере из словаря создается объект, именами атрибутов которого будут ключи словаря, а значениями - значения из словаря по заданным ключам:

```
class AttDict(object):
def __init__(self, dict=None):
object.__setattr__(self, '_selfdict', dict or {})
def __getattr__(self, name):
if self._selfdict.has_key(name):
return self._selfdict[name]
else:
raise AttributeError
def __setattr__(self, name, value):
if name[0] != '_':
self._selfdict[name] = value
else:
raise AttributeError
def __delattr__(self, name):
if name[0] != '_' and self._selfdict.has_key(name):
del self._selfdict[name]
ad = AttDict({'a': 1, 'b': 10, 'c': '123'})
print ad.a, ad.b, ad.c
ad.d = 512
print ad.d
```

Сокрытие данных

Подчеркивание (" _ ") в начале имени атрибута указывает на то, что он не входит в общедоступный интерфейс. Обычно применяется одиночное подчеркивание, которое в языке не играет особой роли, но как бы говорит программисту: "этот метод только для внутреннего использования". Двойное подчеркивание работает как указание на то, что атрибут - приватный. При этом атрибут все же доступен, но уже под другим именем, что и иллюстрируется ниже:

```
>>> class X:
... x = 0
... _x = 0
... __x = 0
```

...
>>> dir(X)
['_X__x', '__doc__', '__module__', '_x', 'x']

Полиморфизм

В переводе с греческого полиморфизм означает "многоформие". Так в информатике называют возможность использования одного имени для выполнения различных действий.

Можно встретить множество определений полиморфизма (также есть несколько видов полиморфизма) в зависимости от языка программирования. Как правило, в качестве примера проявления полиморфизма приводят переопределение методов в подклассах. При этом можно создать функцию, требующую формального аргумента - экземпляра базового класса, а в качестве фактического аргумента давать экземпляр подкласса. Функция будет вызывать метод объекта с именем, а за именем будут скрываться различные действия. В связи с этим полиморфизм обычно связывают с иерархией наследования.

В Python полиморфизм связан не с наследованием, а с набором и смыслом доступных методов в экземпляре класса. Ниже будет показано, что, имея определенные методы, можно воссоздать класс для строки или любого другого встроенного типа. Для этого необходимо определить свойственный типу набор методов. Конечно, нужный набор методов можно получить и с помощью наследования, но в Python это не только не обязательно, но иногда и противоречит здравому смыслу.

При написании функции в Python обычно не проверяется, к какому типу (классу) относится тот или иной аргумент: некоторые методы просто применяются к переданному объекту. Тем самым функции получаются максимально обобщенными: они не требуют от объектов-параметров большего, чем наличие методов с определенным именем, набором аргументов и семантикой.

Следующий пример показывает полиморфизм в том виде, в котором он свойственен Python:

```
def get_last(x):
return x[-1]
print get_last([1, 2, 3])
print get_last("abcd")
```

Описанной функции будет подходить в качестве аргумента все, от чего можно взять индекс -1 (последний элемент). Однако семантика "взятие последнего элемента" выполняется только для последовательностей. Функция будет работать и для словарей, но смысл при этом будет немного другой.

Имитация типов

Для иллюстрации понятия полиморфизма можно построить собственный тип, похожий на встроенный тип "функция". Построить класс, объекты

которого вызываются подобно методам или функциям, можно так:

```
class CountArgs(object):
def __call__(self, *args, **kwargs):
return len(args) + len(kwargs)
cc = CountArgs()
print cc(1, 3, 4)
```

Как видно из этого примера, экземпляры класса CountArgs можно вызывать подобно функциям (в результате будет возвращено количество переданных параметров). При попытке вызова экземпляра на самом деле будет вызван метод __call__() со всеми аргументами.

Следующий пример показывает, что сравнением экземпляров класса тоже можно управлять:

```
class Point:
def __init__(self, x, y):
self.coord = (x, y)
def __nonzero__(self):
return self.coord[0] != 0 or self.coord[1] != 0
def __cmp__(self, p):
return cmp(self.coord, p.coord)
for x in range(-3, 4):
for y in range(-3, 4):
if Point(x, y) < Point(y, x):
print "*",
elif Point(x, y):
print ".",
else:
print "o",
print
Программа выведет:
. * * * * * *
. . * * * * *
. . . * * * *
. . . o * * *
. . . . . * *
. . . . . . *
. . . . . .
```

В данной программе класс Point (Точка) имеет метод __nonzero__(), который определяет истинностное значение объекта класса. Истину будут давать только точки, отличные от (0, 0). Другой метод - __cmp__() - вызывается при необходимости сравнить точку и другой объект (имеющий как и точка атрибут

coord, который содержит кортеж как минимум из двух элементов). Нужно заметить, что вместо __cmp__ можно определить отдельные методы для операций сравнения: __lt__, __le__, __ne__, __eq__, __ge__, __gt__ (для <, <=, !=, ==, >=, > соответственно).

Достаточно легко имитировать и числовые типы. Класс, который пользуется удобством синтаксиса инфиксного +, можно определить так:

```
class Plussable:
def __add__(self, x):
...
def __radd__(self, x):
...
def __iadd__(self, x):
...
```

Здесь метод __add__() вызывается, когда экземпляр класса Plussable стоит слева от сложения, __radd__() - если справа от сложения и метод слева от него не имеет метода __add__(). Метод __iadd__() нужен для реализации +=.

Отношения между классами

Наследование

На практике часто возникает ситуация, когда в предметной области выделены очень близкие, но вместе с тем неодинаковые классы. Одним из способов сокращения описания классов за счет использования их сходства является выстраивание классов в иерархию. В корне этой иерархии стоит базовый класс, от которого нижележащие классы иерархии наследуют свои атрибуты, уточняя и расширяя поведение вышележащего класса. Обычно принципом построения классификации является отношение "IS-A" ("ЕСТЬ"). Например, класс Окружность в программе - графическом редакторе может быть унаследован от класса Геометрическая Фигура. При этом Окружность будет являться подклассом (или субклассом) для класса Геометрическая Фигура, а Геометрическая Фигура - надклассом (или суперклассом) для класса Окружность.

В языке Python во главе иерархии ("новых") классов стоит класс object. Для ориентации в иерархии существуют некоторые встроенные функции, которые будут рассмотрены ниже. Функция issubclass(x, y) может сказать, является ли класс x подклассом класса y:

```
>>> class A(object): pass
...
>>> class B(A): pass
...
>>> issubclass(A, object)
True
>>> issubclass(B, A)
```

True
>>> issubclass(B, object)
True
>>> issubclass(A, str)
False
>>> issubclass(A, A) # класс является подклассом самого себя
True

В основе построения классификации всегда стоит принцип, играющий наиболее важную роль в анализируемой и моделируемой системе. Следует заметить, что одним из "перегибов" при использовании ОО методологии является искусственное выстраивание иерархии классов. Например, не стоит наследовать класс Машина от класса Колесо (внимательные заметят, что здесь отношение другое: колесо является частью машины).

Класс называется абстрактным, если он предназначен только для наследования. Экземпляры абстрактного класса обычно не имеют большого смысла. Классы с рабочими экземплярами называются конкретными.

В Python примером абстрактного класса является встроенный тип basestring, у которого есть конкретные подклассы str и unicode.

Множественное наследование

В отличие, например, от Java, в языке Python можно наследовать класс от нескольких классов. Такая ситуация называется множественным наследованием (multiple inheritance).

Класс, получаемый при множественном наследовании, объединяет поведение своих надклассов, комбинируя стоящие за ними абстракции.

Использовать множественное наследование следует очень осторожно, а необходимость в нем возникает реже одиночного.

- Множественное наследование можно применить для получения класса с заданными общедоступными методами, причем методы задает один родительский класс, а реализуются они на основе методов второго класса. Первый класс может быть полностью абстрактным.
- Множественное наследование применяется для добавления примесей (mixins). Примесь - специально сконструированный класс, добавляющий в некоторый класс какую-либо черту поведения (привнесением атрибутов). Примеси обычно являются абстрактными классами.
- Изредка множественное наследование применяется в своем основном смысле, когда объекты класса, получающегося в результате множественного наследования, предназначаются для использования в качестве объектов всех родительских классов.

В случае с Python наследование можно считать одним из способов собрать нужные комбинации методов в серии классов:

```
class A:
def a(self): return 'a'
class B:
def b(self): return 'b'
class C:
def c(self): return 'c'
class AB(A, B):
pass
class BC(B, C):
pass
class ABC(A, B, C):
pass
```

Впрочем, собрать нужные методы можно и по-другому, без использования наследования:

```
def ma(self): return 'a'
def mb(self): return 'b'
def mc(self): return 'c'
class AB:
a = ma
b = mb
class BC:
b = mb
c = mc
class ABC:
a = ma
b = mb
c = mc
```

Порядок разрешения методов

В случае, когда надклассы имеют одинаковые методы, использование того или иного метода определяется порядком разрешения методов (method resolution order). Для "новых" классов узнать этот порядок очень просто с помощью атрибута __mro__:

```
>>> str.__mro__
(<type 'str'>, <type 'basestring'>, <type 'object'>)
```

Это означает, что сначала методы ищутся в классе str, затем в basestring, а уже потом - в object.

Для "классических" классов порядок несколько отличается от порядка разрешения методов в "новых" классах. Нужно стараться избегать множественного наследования или применять его очень аккуратно.

Агрегация

Контейнеры

Под контейнером обычно понимают объект, основным назначением которого является хранение и обеспечение доступа к другим объектам. Контейнеры реализуют отношение "HAS-A" ("ИМЕЕТ") между объектами. Встроенные типы, список и словарь -- яркие примеры контейнеров. Можно построить собственные типы контейнеров, которые будут иметь свою логику доступа к хранимым объектам. В контейнере хранятся не сами объекты, а ссылки на них.

Для практических нужд в Python обычно хватает встроенных контейнеров (словаря и списка), но если это необходимо, можно создать и другие. Ниже приведен класс Стек, реализованный на базе списка:

```
class Stack:
def __init__(self):
"""Инициализация стека"""
self._stack = []
def top(self):
"""Возвратить вершину стека (не снимая)"""
return self._stack[-1]
def pop(self):
"""Снять со стека элемент"""
return self._stack.pop()
def push(self, x):
"""Поместить элемент на стек"""
self._stack.append(x)
def __len__(self):
"""Количество элементов в стеке"""
return len(self._stack)
def __str__(self):
"""Представление в виде строки"""
return " : ".join(["%s" % e for e in self._stack])
Использование:
>>> s = Stack()
>>> s.push(1)
>>> s.push(2)
>>> s.push("abc")
>>> print s.pop()
abc
>>> print len(s)
2
>>> print s
1 : 2
```

Таким образом, контейнеры позволяют управлять набором (любых) других объектов в соответствии со структурой их хранения, не вмешиваясь во внутренние дела объектов. Узнав интерфейс класса Stack, можно и не догадаться, что он реализован на основе списка, и каким именно образом он реализован с помощью него. Но для использования стека это не важно.

Ассоциация

Если в случае агрегации имеется довольно четкое отношение "ИМЕЕТ" (HAS-A) или "СОДЕРЖИТСЯ-В", которое даже отражено в синтаксисе Python:

```
lst = [1, 2, 3]
if 1 in lst:
...
```

то в случае ассоциации ссылка на экземпляр другого класса используется без отношения включения одного в другой или принадлежности. О таком отношении между классами говорят как об отношении USE-A ("ИСПОЛЬЗУЕТ"). Это достаточно общее отношение зависимости между классами.

В языке Python границы между агрегацией и ассоциацией несколько размыты, так как объекты при агрегации обычно не хранятся в области памяти, выделенной под контейнер (хранятся только ссылки).

Объекты могут также ссылаться друг на друга. В этом случае возникают циклические ссылки, которые при неаккуратном использовании могут привести (в старых версиях Python) к утечкам памяти. В новых версиях Python для циклических ссылок работает сборщик мусора.

Разумеется, при "чистой" агрегации циклических ссылок не возникает.

Например, при представлении дерева ссылки могут идти от родителей к детям и обратно от каждого дочернего узла к родительскому.

Слабые ссылки

Для обеспечения ассоциаций объектов без свойственных ссылкам проблем с возможностью образования циклических ссылок, в Python для сложных структур данных и других видов использования, при которых ссылки не должны мешать удалению объекта, предлагается механизм слабых ссылок. Такая ссылка не учитывается при подсчете ссылок на объект, а значит, объект удаляется с исчезновением последней "сильной" ссылки.

Для работы со слабыми ссылками применяется модуль weakref. Основные принципы его работы станут понятны из следующего примера:

```
>>> import weakref
>>>
>>> class MyClass(object):
... def __str__(self):
... return "MyClass"
```

```
...
>>>
>>> s = MyClass() # создается экземпляр класса
>>> print s
MyClass
>>> s1 = weakref.proxy(s) # создается прокси-объект
>>> print s1 # прокси-объект работает как исходный
MyClass
>>> ss = weakref.ref(s) # создается слабая ссылка на него
>>> print ss() # вызовом ссылки получается исходный объект
MyClass
>>> del s # удаляется единственная сильная ссылка на объект
>>> print ss() # теперь исходного объекта не существует
None
>>> print s1
Traceback (most recent call last):
File "<stdin>", line 1, in ?
ReferenceError: weakly-referenced object no longer exists
```

К сожалению, поведение прокси-объекта не совсем такое, как у исходного: он не может быть ключом словаря, так как является нехэшируемым.

Статический метод

Иногда необходимо использовать метод, принадлежащий классу, а не его экземпляру. В этом случае можно описать статический метод. До появления декораторов (до Python 2.4) определять статический метод приходилось следующим образом:

```
class A(object):
def name():
return A.__name__
name = staticmethod(name)
print A.name()
a = A()
print a.name()
```

Статическому методу не передается параметр с экземпляром класса. Он ему попросту не нужен.

В Python 2.4 для применения описателей (descriptors) был придуман новый синтаксис - декораторы:

```
class A(object):
@staticmethod
def name():
```

return A.__name__

Смысл декоратора в том, что он "пропускает" определяемую функцию (или метод) через заданную в нем функцию. Теперь писать name три раза не потребовалось. Декораторов может быть несколько, и применяются они в обратном порядке.

Метод класса

Если статический метод имеет свои аналоги в C++ и Java, то метод класса основан на том, что в Python классы являются объектами. В отличие от статического метода, в метод класса первым параметром передается объект-класс. Вместо self для подчеркивания принадлежности метода к методам класса принято использовать cls.

Пример использования метода класса можно найти в модуле tree пакета nltk (Natural Language ToolKit, набор инструментов для естественного языка). Ниже приведен лишь фрагмент определения класса Tree (базового класса для других подклассов). Метод convert класса Tree определяет процедуру преобразования дерева одного типа в дерево другого типа. Эта процедура абстрагируется от деталей реализации конкретных типов, описывая обобщенный алгоритм преобразования:

class Tree:
 # ...
def convert(cls, val):
if isinstance(val, Tree):
children = [cls.convert(child) for child in val]
return cls(val.node, children)
else:
return val
convert = classmethod(convert)
Пример использования (взят из строки документации метода convert()):
>>> # Преобразовать tree в экземпляр класса Tree
>>> tree = Tree.convert(tree)
>>> # " " " " " ParentedTree
>>> tree = ParentedTree.convert(tree)
>>> # " " " " " MultiParentedTree
>>> tree = MultiParentedTree.convert(tree)

Метод класса позволяет более естественно описывать действия, которые связаны в основном с классами, а не с методами экземпляра класса.

Метаклассы

Еще одним отношением между классами является отношение класс-метакласс. Метакласс можно считать "высшим пилотажем" объектно-ориентированного программирования, но, к счастью, в Python можно создавать собственные метаклассы.

В Python класс тоже является объектом, поэтому ничего не мешает написать класс, назначением которого будет создание других классов динамически, во время выполнения программы.

Пример, в котором класс порождается динамически в функции-фабрике классов:

```
def cls_factory_f(func):
class X(object):
pass
setattr(X, func.__name__, func)
return X
Использование будет выглядеть так:
def my_method(self):
print "self:", self
My_Class = cls_factory_f(my_method)
my_object = My_Class()
my_object.my_method()
```

В этом примере функция cls_factory_f() возвращает класс с единственным методом, в качестве которого используется функция, переданная ей как аргумент. От этого класса можно получить экземпляры, а затем у экземпляров - вызвать метод my_method.

Теперь можно задаться целью построить класс, экземплярами которого будут классы. Такой класс, от которого порождаются классы, и называется метаклассом.

В Python имеется класс type, который на деле является метаклассом. Вот как с помощью его конструктора можно создать класс:

```
def my_method(self):
print "self:", self
My_Class = type('My_Class', (object,), {'my_method': my_method})
```

В качестве первого параметра type передается имя класса, второй параметр - базовые классы для данного класса, третий - атрибуты.

В результате получится класс, эквивалентный следующему:

```
class My_Class(object):
def my_method(self):
print "self:", self
```

Но самое интересное начинается при попытке составить собственный метакласс. Проще всего наследовать метакласс от метакласса type (пример взят из статьи Дэвида Мертца):

```
>>> class My_Type(type):
... def __new__(cls, name, bases, dict):
... print "Выделение памяти под класс", name
... return type.__new__(cls, name, bases, dict)
... def __init__(cls, name, bases, dict):
... print "Инициализация класса", name
... return super(My_Type, cls).__init__(cls, name, bases, dict)
...
>>> my = My_Type("X", (), {})
Выделение памяти под класс X
Инициализация класса X
```

В этом примере не происходит вмешательство в создание класса. Но в __new__() и __init__() имеется полный программный контроль над создаваемым классом в период выполнения.

Мультиметоды

Некоторые объектно-ориентированные "штучки" не входят в стандартный Python или стандартную библиотеку. Ниже будут рассмотрены мультиметоды - методы, сочетающие объекты сразу нескольких различных классов. Например, сложение двух чисел различных типов фактически требует использования мультиметода. Если "одиночный" метод достаточно задать для каждого класса, то мультиметод требует задания для каждого сочетания классов, которые он обслуживает:

```
>>> import operator
>>> operator.add(1, 2)
3
>>> operator.add(1.0, 2)
3.0
>>> operator.add(1, 2.0)
3.0
>>> operator.add(1, 1+2j)
(2+2j)
>>> operator.add(1+2j, 1)
(2+2j)
```

В этом примере operator.add ведет себя как мультиметод, выполняя разные действия для различных комбинаций параметров.

Для организации собственных мультиметодов можно воспользоваться модулем Multimethod (автор Neel Krishnaswami), который легко найти в Интернете. Следующий пример, адаптированный из документации модуля, показывает построение собственного мультиметода:

```
from Multimethod import Method, Generic, AmbiguousMethodError
# классы, для которых будет определен мультиметод
class A: pass
class B(A): pass
# функции мультиметода
def m1(a, b): return 'AA'
def m2(a, b): return 'AB'
def m3(a, b): return 'BA'
# определение мультиметода (без одной функции)
g = Generic()
g.add_method(Method((A, A), m1))
g.add_method(Method((A, B), m2))
g.add_method(Method((B, A), m3))
# применение мультиметода
try:
print 'Типы аргументов:', 'Результат'
print 'A, A:', g(A(), A())
print 'A, B:', g(A(), B())
print 'B, A:', g(B(), A())
print 'B, B:', g(B(), B())
except AmbiguousMethodError:
print 'Неоднозначный выбор метода'
```

Даже достаточно неформальное введение в ООП потребовало определения большого количества терминов. В лекции была сделана попытка с помощью примеров передать не столько букву, сколько дух терминологии ООП. Были рассмотрены все базовые понятия: объект, тип, класс и виды отношений между объектами (IS-A, HAS-A, USE-A). Слушатели получили представление о том, что такое инкапсуляция и полиморфизм в стиле ООП, а также наследование, продление времени жизни объекта за рамками исполняющейся программы, известное как устойчивость объекта (object persistence). Были указаны недостатки ООП, но при этом весь предыдущий материал объективно свидетельствовал о достоинствах этого подхода.

Лекция 6. Стиль программирования. Структурное, модульное программирование

Стиль программирования - дополнительные ограничения, накладываемые на структуру и вид программного кода группой совместно работающих программистов с целью получения удобных для применения, легко читаемых и эффективных программ. Основные ограничения на вид программы дает синтаксис языка программирования, и его нарушения вызывают синтаксические ошибки. Нарушение стиля не приводит к синтаксическим ошибкам, однако как отдельные программисты, так и целые коллективы сознательно ограничивают себя в средствах выражения ради упрощения совместной разработки, отладки и сопровождения программного продукта.

Стиль программирования затрагивает практически все аспекты написания кода:

- именование объектов в зависимости от типа, назначения, области видимости;
- оформление функций, методов, классов, модулей и их документирование в коде программы;
- декомпозиция программы на модули с определенными характеристиками;
- способ включения отладочной информации;
- применение тех или иных функций (методов) в зависимости от предполагаемого уровня совместимости разрабатываемой программы с различными компьютерными платформами;
- ограничение используемых функций из соображений безопасности.

Для языка Python Гвидо ван Россум разработал официальный стиль. С оригинальным текстом "Python Style Guide" можно ознакомиться по адресу http://www.python.org/doc/essays/styleguide.html.

Наиболее существенные положения этого стиля перечислены ниже. В случае сомнений хорошим образцом стиля являются модули стандартной библиотеки.

- Рекомендуется использовать отступы в 4 пробела.
- Длина физической строки не должна превышать 79 символов.
- Длинные логические строки лучше разбивать неявно (внутри скобок), но и явные методы вполне уместны. Отступы строк продолжения рекомендуется выравнивать по скобкам или по первому операнду в предыдущей строке. Текстовый редактор Emacs в режиме python-mode и некоторые интегрированные оболочки (IDE) автоматически делают необходимые отступы в Python-программах:

```
def draw(figure, color="White", border_color="Black",
```

```
size=5):
if color == border_color or \
size == 0:
raise "Bad figure"
else:
draw(size, size, (color,
 border_color))
```

- Не рекомендуется ставить пробелы сразу после открывающей скобки или перед закрывающей, перед запятой, точкой с запятой, перед открывающей скобкой при записи вызова функции или индексного выражения. Также не рекомендуется ставить более одного пробела вокруг знака равенства в присваиваниях. Пробелы вокруг знака равенства не ставятся в случае, когда он применяется для указания значения по умолчанию в определении параметров функции или при задании именованных аргументов.
- Также рекомендуется применение одиночных пробелов вокруг низкоприоритетных операций сравнения и оператора присваивания. Пробелы вокруг более приоритетных операций ставятся в равном количестве слева и справа от знака операции. Несколько рекомендаций касаются написания комментариев.
- Комментарии должны точно отражать актуальное состояние кода. (Поддержание актуальных комментариев должно быть приоритетной задачей!) После коротких комментариев можно не ставить точку, тогда как длинные лучше писать по правилам написания текста. Автор Python обращается к неанглоязычным программистам с просьбой писать комментарии на английском, если есть хотя бы небольшая вероятность того, что код будут читать специалисты, говорящие на других языках.
- Комментарии к фрагменту кода следует писать с тем же отступом, что и комментируемый код. После "#" должен идти одиночный пробел. Абзацы можно отделять строкой с "#" на том же уровне. Блочный комментарий можно отделить пустыми строками от окружающего кода.
- Комментарии, относящиеся к конкретной строке, не следует использовать часто. Символ "#" должен отстоять от комментируемого оператора как минимум на два пробела.
- Хороший комментарий не перефразирует программу, а содержит дополнительную информацию о действии программы в терминах предметной области.

Все модули, классы, функции и методы, предназначенные для использования за пределами модуля, должны иметь строки документации, описывающие способ их применения, входные и выходные параметры.

- Строка документации для отдельной программы должна объяснять используемые ею ключи, назначение аргументов и переменных среды и другую подобную информацию.
- Для строк документации рекомендуется везде использовать утроенные кавычки (""").
- Однострочная документация пишется в императиве, как команда: "делай это", "возвращай то".
- Многострочная документация содержит расширенное описание модуля, функции, класса. Она будет смотреться лучше, если текст будет написан с тем же отступом, что и начало строки документации.
- Документация для модуля должна перечислять экспортируемые функции, классы, исключения и другие объекты, по одной строке на объект.
- Строка документации для функции или метода должна кратко описывать действия функции, ее входные параметры и возвращаемое значение, побочные эффекты и возможные исключения (если таковые есть). Должны быть обозначены необязательные аргументы и аргументы, не являющиеся частью интерфейса.
- Документация для класса должна перечислять общедоступные методы и атрибуты, содержать рекомендации по применению класса в качестве базового для других классов. Если класс является подклассом, необходимо указать, какие методы полностью заменяют, перегружают, а какие используют, но расширяют соответствующие методы надкласса. Необходимо указать и другие изменения по сравнению с надклассом.
- Контроль версий повышает качество процесса создания программного обеспечения. Для этих целей часто используются RCS или CVS. "Python Style Guide" рекомендует записывать $Revision: 1.31 $ в переменную с именем __version__, а другие данные заключать в комментарии "#".

Сегодня сосуществуют несколько более или менее широко распространенных правил именования объектов. Программисты вольны выбрать тот, который принят в их организации или конкретном проекте. Автор Python рекомендует придерживаться нижеследующих правил для именования различных объектов, с тем чтобы это было понятно любому программисту, использующему Python.

- Имена модулей лучше давать строчными буквами, например, shelve, string, либо делать первые буквы слов прописными, StringIO, UserDict. Имена написанных на C модулей расширения обычно начинаются с подчеркивания "_", а соответствующие им высокоуровневые обертки - с прописных букв: _tkinter и Tkinter.

- Ключевые слова нельзя использовать в качестве имен, однако, если все-таки необходимо воспользоваться этим именем, стоит добавить одиночное подчеркивание в конце имени. Например: class_.
- Классы обычно называют, выделяя первые буквы слов прописными, как в Tag или HTTPServer.
- Имена исключений обычно содержат в своем составе слово "error" (или "warning"). Встроенные модули пишут это слово со строчной буквы (как os.error) (но могут писать и с прописной): distutils.DistutilsModuleError.
- Функции, экспортируемые модулем, могут именоваться по-разному. Можно давать с прописных букв имена наиболее важных функций, а вспомогательные писать строчными.
- Имена глобальных переменных (если таковые используются) лучше начинать с подчеркивания, чтобы они не импортировались из модуля оператором from-import со звездочкой.
- Имена методов записываются по тем же правилам, что и имена функций.
- Имена констант (имен, которые не должны переопределяться) лучше записывать прописными буквами, например: RED, GREEN, BLUE.
- При работе с языком Python необходимо учитывать, что интерпретатор считает некоторые классы имен специальными (обычно такие имена начинаются с подчеркивания).

Структурное программирование — парадигма программирования, в основе которой лежит представление программы в виде иерархической структуры блоков.

По своей сути оно воплощает принципы системного подхода в процессе создания и эксплуатации программного обеспечения ЭВМ.

Структурное программирование воплощает принципы системного подхода в процессе создания и эксплуатации программного обеспечения ЭВМ. В основу структурного программирования положены следующие достаточно простые положения:

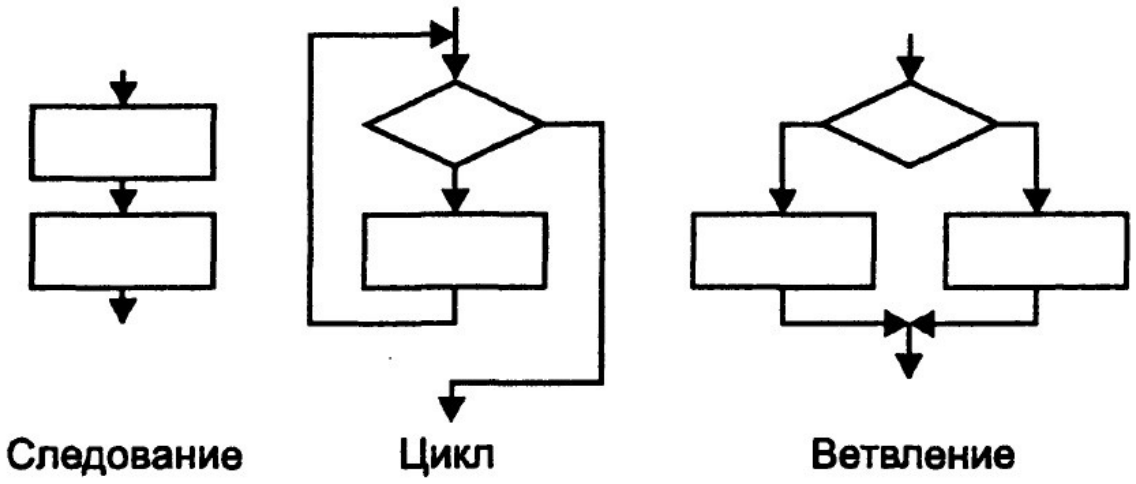
- алгоритм и программа должны составляться поэтапно (по шагам).
- сложная задача должна разбиваться на достаточно простые части, каждая из которых имеет один вход и один выход.
- логика алгоритма и программы должна опираться на минимальное число достаточно простых базовых управляющих структур.

Структурное программирование иногда называют еще «программированием без go to». Рекомендуется избегать употребления оператора перехода всюду, где это возможно, но чтобы это не приводило к слишком громоздким структурированным программам.

goto (перейти на) — оператор безусловного перехода (перехода к определённой точке программы, обозначенной номером строки либо меткой) в некоторых языках программирования. В некоторых языках оператор безусловного перехода может иметь другое имя (например, jmp в языках ассемблера).

Фундаментом структурного программирования является теорема о структурировании. Эта теорема устанавливает, что, как бы сложна ни была задача, схема соответствующей программы всегда может быть представлена с использованием ограниченного числа элементарных управляющих структур.

Базовыми элементарными структурами являются структуры: следование, ветвление и повторение (цикл), любой алгоритм может быть реализован в виде композиции этих трех конструкций.



Базовые конструкции структурного программирования

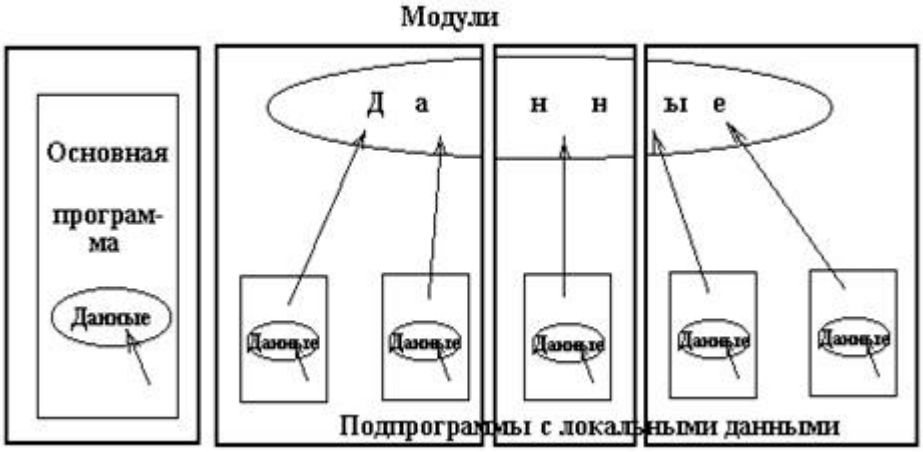
Достоинства структурного программирования:

- повышается надежность программ (благодаря хорошему структурированию при проектировании, программа легко поддается тестированию и не создает проблем при отладке);
- повышается эффективность программ (структурирование программы позволяет легко находить и корректировать ошибки, а отдельные подпрограммы можно переделывать (модифицировать) независимо от других);
- уменьшается время и стоимость программной разработки;
- улучшается читабельность программ.

Модульное программирование является естественным следствием проектирования сверху вниз и заключается в том, что программа разбивается на части – модули, разрабатываемые по отдельности.

В программировании под модулем понимается отдельная подпрограмма, а подпрограммы часто называются процедурами или процедурами-

функциями. Поэтому модульное программирование еще называется процедурным.



Модуль должен обладать следующими свойствами:

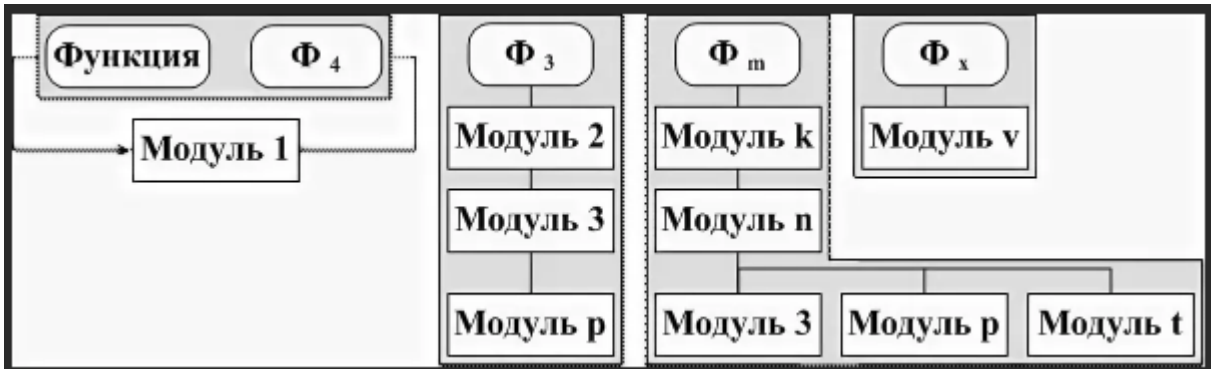
- один вход и один выход – на входе программный модуль получает определенный набор исходных данных, выполняет содержательную обработку и возвращает один набор результатных данных, т.е. реализуется стандартный принцип IPO (Input — Process — Output — вход-процесс-выход);
- функциональная завершенность – модуль выполняет перечень регламентированных операций для реализации каждой отдельной функции в полном составе, достаточных для завершения начатой обработки;
- логическая независимость – результат работы программного модуля зависит только от исходных данных, но не зависит от работы других модулей;
- слабые информационные связи с другими программными модулями – обмен информацией между модулями должен быть по возможности минимизирован;
- обозримый по размеру и сложности программный код.

Модули содержат определение доступных для обработки данных, операции обработки данных, схемы взаимосвязи с другими модулями.

Каждый модуль состоит из спецификации и тела. Спецификации определяют правила использования модуля, а тело – способ реализации процесса обработки.

Принципы модульного программирования программных продуктов во многом сходны с принципами нисходящего проектирования: сначала определяются состав и подчиненность функций, а затем — набор программных модулей, реализующих эти функции.

Функционально-модульная структура приложения



Лекция 7. Принципы межмодульного взаимодействия. Принципы мультипрограммирования

Встроенные в язык программирования функции доступны сразу. Чтобы их вызвать, не надо выполнять никаких дополнительных действий. Однако за время существования любого популярного языка на нем было написано столько функций и классов, которые оказались востребованными множеством программистов и в разных областях, что включить весь этот объем кода в сам язык если возможно, то нецелесообразно.

Чтобы разрешить проблему доступа к дополнительным возможностям языка, в программировании стало общепринятой практикой использовать так называемые модули, пакеты и библиотеки. Каждый модуль содержит коллекцию функций и классов, предназначенных для решения задач из определенной области. Так в модуле math языка Python содержатся математические функции, модуль random позволяет генерировать псевдослучайные числа, в модуле datetime содержатся классы для работы с датами и временем, модуль sys предоставляет доступ к системным переменным и т. д.

Количество модулей для языка Python огромно, что связано с популярностью языка. Часть модулей собрана в так называемую стандартную библиотеку. Стандартная она потому, что поставляется вместе с установочным пакетом. Однако существуют сторонние библиотеки. Они скачиваются и устанавливаются отдельно.

Для доступа к функционалу модуля, его надо импортировать в программу. После импорта интерпретатор "знает" о существовании дополнительных классов и функций и позволяет ими пользоваться.

В Питоне импорт осуществляется командой import. При этом существует несколько способов импорта. Рассмотрим работу с модулем на примере math. Итак,

```
>>> import math
```

Ничего не произошло. Однако в глобальной области видимости появилось имя math. Если до импорта вы упомянули бы имя math, то возникла бы ошибка NameError. Теперь же

```
>>> math
<module 'math' (built-in)>
```

В программе завелся объект math, относящийся к классу module.

Чтобы увидеть перечень функций, входящих в этот модуль, воспользуемся встроенной в Python функцией dir(), передав ей в качестве аргумента имя модуля:

```
>>> dir(math)
['__doc__', '__loader__', '__name__',
'__package__', '__spec__', 'acos', 'acosh',
'asin', 'asinh', 'atan', 'atan2', 'atanh',
'ceil', 'copysign', 'cos', 'cosh',
'degrees', 'e', 'erf', 'erfc', 'exp',
'expm1', 'fabs', 'factorial', 'floor',
'fmod', 'frexp', 'fsum', 'gamma', 'gcd',
 'hypot', 'inf', 'isclose', 'isfinite',
'isinf', 'isnan', 'ldexp', 'lgamma', 'log',
'log10', 'log1p', 'log2', 'modf',
'nan', 'pi', 'pow', 'radians', 'sin', 'sinh',
'sqrt', 'tan', 'tanh', 'trunc']
```

Проигнорируем имена с двойными подчеркиваниями. Все остальное – имена функций и констант (переменных, которые не меняют своих значений), включенных в модуль math. Чтобы вызвать функцию из модуля, надо впереди написать имя модуля, поставить точку, далее указать имя функции, после чего в скобках передать аргументы, если они требуются. Например, чтобы вызвать функцию pow из math, надо написать так:

```
>>> math.pow(2, 2)
4.0
```

Обратите внимание, эта другая функция pow(), не та, что встроена в сам язык. "Обычная" функция pow() возвращает целое, если аргументы целые числа:

```
>>> pow(2, 2)
4
```

Для обращения к константе скобки не нужны:

```
>>> math.pi
3.141592653589793
```

Если мы не знаем, что делает та или иная функция, то можем получить справочную информацию о ней с помощью встроенной в язык Python функции help():

```
>>> help(math.gcd)
Help on built-in function gcd in module math:

gcd(...)
```

gcd(x, y) -> int
greatest common divisor of x and y

Для выхода из интерактивной справки надо нажать клавишу q. В данном случае сообщается, что функция возвращает целое число, и это наибольший общий делитель для чисел x и y. Описание модулей и их содержания также можно посмотреть в официальной документации на сайте python.org.

Второй способ импорта – это когда импортируется не сам модуль, а только необходимые функции из него.

```
>>> from math import gcd, sqrt, hypot
```

Перевести можно как "из модуля math импортировать функции gcd, sqrt и hypot".

В таком случае при их вызове не надо перед именем функции указывать имя модуля:

```
>>> gcd(100, 150)
50
>>> sqrt(16)
4.0
>>> hypot(3, 4)
5.0
```

Чтобы импортировать сразу все функции из модуля:

```
>>> from math import *
```

Импорт через from не лишен недостатка. В программе уже может быть идентификатор с таким же именем, как имя одной из импортируемых функций или констант. Ошибки не будет, но одно из них окажется "затерто":

```
>>> pi = 3.14
>>> from math import pi
>>> pi
3.141592653589793
```

Здесь исчезает значение 3.14, присвоенное переменной pi. Это имя теперь указывает на число из модуля math. Если импорт сделать раньше, чем присвоение значения pi, то будет все наоборот:

```
>>> from math import pi
>>> pi = 3.14
```

```
>>> pi
3.14
```

В этой связи более опасен именно импорт всех функций. Так как в этом случае очень легко не заметить подмены значений идентификаторов.

Однако можно изменить имя идентификатора из модуля на какое угодно:

```
>>> from math import pi as P
>>> P
3.141592653589793
>>> pi
3.14
```

В данном случае константа pi из модуля импортируется под именем P. Смысл подобных импортов в сокращении имен, так как есть модули с длинными именами, а имена функций и классов в них еще длиннее. Если в программу импортируется всего пара сущностей, и они используются в ней часто, то имеет смысл переименовать их на более короткий вариант. Сравните:

```
>>> import calendar
>>> calendar.weekheader(2)
'Mo Tu We Th Fr Sa Su'
u

>>> from calendar import weekheader as week
>>> week(3)
'Mon Tue Wed Thu Fri Sat Sun'
```

Во всех остальных случаях лучше оставлять идентификаторы содержимого модуля в пространстве имен самого модуля и получать доступ к ним через имя модуля, то есть выполнять импорт командой import имя_модуля, а вызывать, например, функции через имя_модуля.имя_функции().

Создание собственного модуля

Программист на Python всегда может создать собственный модуль, чтобы использовать его в нескольких своих программах или даже предоставить в пользование всему миру. В качестве тренировки создадим модуль с функциями для вычисления площадей прямоугольника, треугольника и круга:

```
from math import pi, pow

def rectangle(a, b):
    return round(a * b, 2)
```

```
def triangle(a, h):
    return round(0.5 * a * h, 2)

def circle(r):
    return round(pi * pow(r, 2), 2)
```

Здесь также иллюстрируется принцип, что один модуль может импортировать другие. В данном случае импортируются функции из модуля math.

Поместите данный код в отдельный файл square.py. Однако куда поместить сам файл?

Когда интерпретатор Питона встречает команду импорта, то просматривает на наличие файла-модуля определенные каталоги. Их перечень можно увидеть по содержимому sys.path:

```
>>> import sys
>>> sys.path
['', '/usr/lib/python35.zip',
'/usr/lib/python3.5',
'/usr/lib/python3.5/plat-x86_64-linux-gnu',
'/usr/lib/python3.5/lib-dynload',
'/home/pl/.local/lib/python3.5/site-packages',
'/usr/local/lib/python3.5/dist-packages',
'/usr/lib/python3/dist-packages']
```

Это список адресов в Linux. В Windows он будет несколько другим. Первый элемент – пустая строка, что обозначает текущий каталог, то есть то место, где сохранена сама программа, импортирующая модуль. Если вы сохраните файл-модуль и файл-программу в одном каталоге, то интерпретатор без труда найдет модуль.

Также модуль можно положить в любой другой из указанных в списке каталогов. Тогда он будет доступен для всех программ на Python, а также его можно будет импортировать в интерактивном режиме.

Можно добавить в sys.path свой каталог. Однако в этом случае либо код программы должен содержать команды изменения значения sys.path, либо надо править конфигурационный файл операционной системы. В большинстве случаев лучше так не делать.

Поместите файл square.py в тот же каталог, где будет исполняемая программа. Ее код должен включать инструкцию импорта модуля square (при импорте расширение файла не указывается) и вызов той функции и с теми параметрами, которые ввел пользователь. То есть у пользователя надо спросить, площадь какой фигуры он хочет вычислить. Далее запросить у него аргументы для соответствующей функции. Передать их в функцию из модуля square, а полученный оттуда результат вывести на экран.

Примечание. Исполнение модуля как самостоятельного скрипта, а также создание строк документации, которые отображает встроенная в Python функция help(), будут рассмотрены в курсе объектно-ориентированного программирования.

РАЗДЕЛ 4. РАЗРАБОТКА КОДА ПРОГРАММНОГО ПРОДУКТА НА УРОВНЕ МОДУЛЯ

Лекция 8. Элементы и приемы программирования на аппаратном уровне

Язык ассемблера

Первые ЭВМ появились в 40-х годах XX века. Эти устройства занимали несколько больших комнат, по вычислительной мощности были сопоставимы с микроЭВМ, встраеваемых в современные сотовые телефоны и программировались путем непосредственного указания числовых кодов операций и номеров ячеек памяти. Вскоре появились более совершенные ЭВМ, “понимающие” десятки команд, программы усложнились и их размер превысил сотню команд. Для упрощения труда программистов был разработан язык ассемблера (от англ. assembler – сборщик), позволяющий заменить числовые коды команд символьными сокращениями или аббревиатурами, а номера ячеек памяти мнемониками имен переменных.

Транслятор ассемблера при генерации машинного кода заменяет эти аббревиатуры и мнемонические обозначения численными кодами операций и номерами (адресами) ячеек памяти. Однозначность такой трансляции символьных обозначений в коды допускает обратную операцию – дизассемблирование – преобразование машинных кодов в мнемонические обозначения команд. Понятно, что при таком подходе программа оказывается привязанной к системе команд конкретной ЭВМ или процессора. Однако, несмотря на громоздкий исходный текст с множеством различных команд и плохую переносимость между различными семействами ЭВМ, ассемблер оставался практически единственным языком программирования до 60-х годов XX века. Такую живучесть обеспечили прямота и четкость этого языка, и эффективность результирующего машинного кода. В конце 50-х годов размер типичной программы перерос 1000 строк, кроме того, появилось большое количество семейств ЭВМ, и разработчики предложили процедурно-ориентированные языки программирования – языки “высокого уровня”, наиболее известные из которых Fortran, Algol, Pascal и C. Эти языки обеспечивали существенно лучшую переносимость программ и предоставили ряд удобств программистам, однако, их преимущества были достигнуты ценой снижения эффективности машинного кода и потери возможности однозначной трансляции дизассемблирования программ.

В настоящее время, скорости разработки прикладных программ уделяется большое внимание. Поэтому, основными инструментами разработки стали языки высокого уровня и даже интерпретируемые языки (C#, Python и др.). Однако, “низкоуровневое программирование” на языке ассемблера до сих пор применяется для решения некоторых типов задач, и в обозримом будущем полный отказ от использования ассемблера не ожидается.

Задачами, требующими использования ассемблера являются:

- разработка системного программного обеспечения (операционных систем и их компонент);
- разработка программных драйверов периферийных устройств;
- анализ и коррекция кода программы при недоступном исходном тексте (типичные задачи хакинга, изучение компьютерных вирусов, полицейские мероприятия и т.п.);
- анализ и математическая обработка сигналов в реальном времени и др.

Таким образом, ассемблер часто используют для решения задач в областях биомедицинской инженерии, промышленной автоматизации и др.

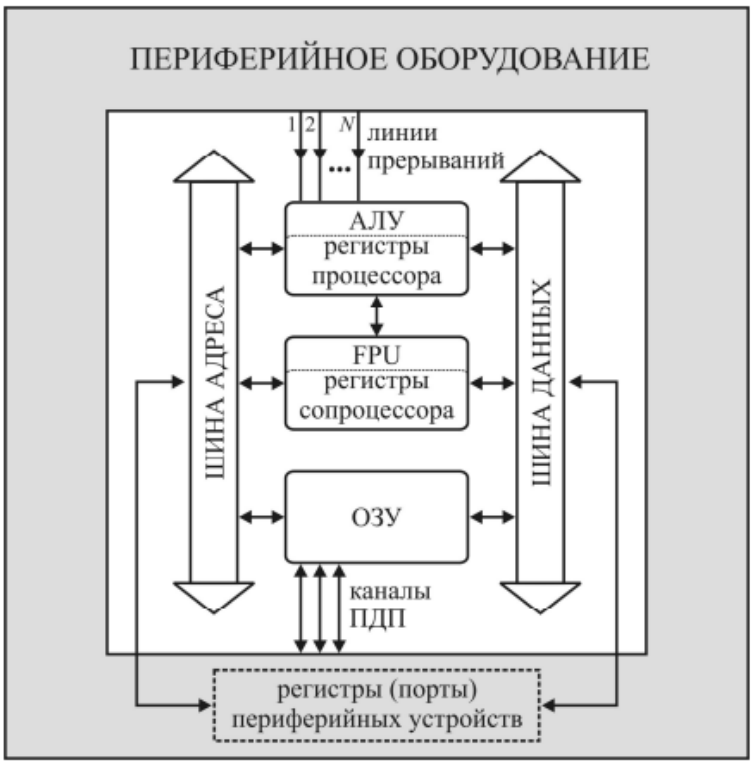
Программы, написанные на языке ассемблера полностью, сейчас редкость даже в области программирования микроконтроллеров и сигнальных процессоров. Современные языки программирования, например, C, позволяют использовать отдельные функции, написанные на ассемблере и даже ассемблерные вставки, как часть кода функции, написанной на языке высокого уровня. Такой подход позволяет сочетать простоту разработки и отладки языка высокого уровня с высокой эффективностью наиболее критичных участков кода, написанных на ассемблере.

Образно, использование языка ассемблера можно сравнить с попыткой наладить деловое общение с китайцем. Вы можете общаться с китайцем на универсальном английском – это программирование на C, можете говорить через переводчика – эквивалент использования интерпретируемого языка, например, Python, но наиболее эффективная коммуникация получиться, в случае, если вы освоите сложный китайский язык и заговорите с китайцем на родном для него языке.

Еще одной сильной стороной ассемблера являются его методические качества. Это родной язык ЭВМ, поэтому, понимание ассемблера существенно помогает в профессиональном освоении любого языка высокого уровня, позволяя понять внутренние механизмы функционирования этого языка.

Структура ЭВМ

Одной из основных сложностей, с которой сталкивается начинающий программист на ассемблере, является необходимость понимания некоторых особенностей устройства ЭВМ на аппаратном уровне. Базовые сведения, необходимые для понимания основных принципов организации работы любой ЭВМ сведены в этом разделе. Нужно понимать, что приведенные здесь сведения очерчивают собирательный образ, и не являются универсальными для всех ЭВМ. Например, далеко не все современные процессоры имеют каналы прямого доступа к памяти, ряд современных процессоров имеют шину для быстрого доступа к ОЗУ встроенную непосредственно в кристалл и т.п. Блок схема, содержащая, с учетом сделанных оговорок, основные структурные элементы ЭВМ приведена на рис.



- Центральным элементом ЭВМ является арифметико-логическое устройство (АЛУ, англ. arithmetic and logic unit, ALU), обеспечивающее выполнение программы и осуществляющее управление остальными блоками. Современные процессоры могут иметь несколько блоков АЛУ, работающих параллельно.
- Основной обмен данными между АЛУ, ОЗУ, математическим сопроцессором и периферийными устройствами осуществляется через шину адреса (англ. address bus) и шину данных (англ. data bus). Шина предназначена для обмена цифровой информацией между несколькими устройствами. Физически, она представляет собой набор линий (англ. lines) – проводников и контролирующую логику. К настоящему моменту разработаны и используются в различных системах десятки стандартов шин, как последовательных, так и параллельных (передающих несколько битов в один момент времени). Некоторые из этих стандартов подразумевают передачу и данных и адреса по одним и тем же линиям. Многие ЭВМ, например, современные ПК могут иметь несколько шин, используемых для обмена данными между различными блоками и элементами.
- Оперативное запоминающее устройство (ОЗУ, англ. random-access memory, RAM) предназначено для временного хранения данных, используемых программой. Основными характеристиками ОЗУ является его объем и скорость с которой оно может обмениваться данными с АЛУ.

- Обычно, АЛУ может выполнять инструкции только целочисленной арифметики. Работу с вещественными числами оно эмулирует (англ. emulate), т.е. симулирует. Эмуляция существенно замедляет вычисления, поэтому, уже в середине 80-х годов XX века, были разработаны в виде отдельных устройств т.н. математические сопроцессоры (англ. mathematical coprocessor), позволяющие быстро осуществлять арифметические действия с вещественными числами. Вещественные числа представляются в ЭВМ, как числа с плавающей точкой (англ. floating point), поэтому, по-английски такие устройства называют Floating Point Unit (FPU). Сейчас многие процессоры имеют по нескольку блоков FPU, размещенных на одном кристалле с АЛУ. Однако, микроконтроллеры и сигнальные процессоры обычно имеют только блоки для целочисленной арифметики, а FPU не имеют.
- Регистры процессора представляют из себя быстродействующие ячейки памяти, расположенные непосредственно в АЛУ. Регистры используются для хранения операндов, параметров, промежуточных и окончательных результатов арифметических и логических операций, для обмена данными с ОЗУ, для управления работой процессора и контроля его состояния. В зависимости от архитектуры процессора и назначения регистра, регистры могут иметь разрядность 8, 16, 32, 64 или 80 бит. С точки зрения программиста, программирование процессора фактически сводится к программированию его регистров. В различных системах программный доступ к регистрам может осуществляться по имени регистра, по его номеру (в этом случае, говорят: “по адресу регистра в адресном пространстве ввода-вывода”), адресное пространство регистров может встраиваться в адресное пространство ОЗУ (в этом случае, к регистру обращаются, как к ячейке ОЗУ с фиксированным адресом). Набор регистров конкретного процессора строго определен и подробно описан в технической документации процессора, часто этот набор называют регистровым файлом (англ. register file) данного процессора. Современные процессоры имеют более сотни регистров различного назначения.
- Регистры (порты) периферийного устройства представляют из себя ячейки памяти, расположенные в микросхемах управляющей логики этого устройства. Их назначение аналогично назначению регистров процессора: управление и контроль состояния устройства, обмен данными с АЛУ. Список регистров устройства, правила их адресации и использования подробно описываются в технической документации (англ. manual, developer’s manual, programmers’s guide) на данное устройство.
- Механизм обработки прерываний (англ. interrupt) широко используется на практике для взаимодействия с периферийным

оборудованием. ЭВМ, обычно, имеет несколько десятков линий прерываний, представляющих собой отдельные проводники, соединяющие периферийное устройство с контроллером прерываний АЛУ. При посылке сигнала запроса на прерывание (англ. InteRrupt reQuest, IRQ) периферийным устройством контроллеру прерываний по линии k, АЛУ приостанавливает выполнение текущей программы, запоминает адрес следующей команды NI текущей программы и немедленно начинает выполнение процедуры обработчика прерывания номер k (говорят: осуществляется переход на вектор прерывания k). Естественно, адрес процедуры обработчика прерывания k, должен быть предварительно загружен в специальную системную область памяти – таблицу векторов прерываний программистом, пишущим прикладную программу. После выполнения функции-обработчика, контроллер прерываний продолжает выполнять задачу, прерванную возбуждением прерывания (англ. interrupt execution), начиная с запомненного адреса возврата NI. Взаимодействие с периферийными устройствами посредством механизма обработки прерываний используется в случаях, требующих немедленной реакции процессора на событие периферийного устройства, например, в системах контроля, при синхронном сборе данных с помощью внешнего АЦП и т.п. Прерывания могут возбуждаться самим АЛУ в случае возникновения исключительных ситуаций (англ. exception), например, при ошибках арифметических операций типа деления на 0. Прерывания также могут возбуждаться программно по команде программиста. Программный вызов прерываний широко используется, например, в прикладных программах MS-DOS для организации взаимодействия с операционной системой.

- Каналы прямого доступа к памяти – ПДП предоставляют возможность периферийным устройствам писать данные в ОЗУ и читать из него информацию в обход АЛУ. Это позволяет осуществлять обработку непрерывно поступающих от периферийного устройства данных без потери и искажения информации даже если АЛУ работает в многозадачном режиме и его загрузка меняется во времени непредсказуемо. Типовой задачей, требующей использования ПДП, является ввод/вывод данных через АЦП/ЦАП для процессора, работающего в многозадачном режиме. Для ПЭВМ это, например, запись звука и видео, синтез звука и т.п.
- Центральным процессором – CPU называют микросхему, содержащую АЛУ. При этом современные процессоры могут иметь на своем кристалле несколько параллельно работающих АЛУ (многоядерный процессор), несколько FPU, контроллеры шин, и даже графический контроллер.

РАЗДЕЛ 5. РАЗРАБОТКА МОДУЛЕЙ СИСТЕМНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Лекция 9. Разработка модулей системного программного обеспечения

Порядок разработки программного модуля.

При разработке программного модуля целесообразно придерживаться следующего порядка:

- изучение и проверка спецификации модуля, выбор языка программирования;
- выбор алгоритма и структуры данных;
- программирование (кодирование) модуля;
- шлифовка текста модуля;
- проверка модуля;
- компиляция модуля.

Первый шаг разработки программного модуля в значительной степени представляет собой смежный контроль структуры программы снизу: изучая спецификацию модуля, разработчик должен убедиться, что она ему понятна и достаточна для разработки этого модуля. В завершении этого шага выбирается язык программирования: хотя язык программирования может быть уже предопределен для всего ПС, все же в ряде случаев (если система программирования это допускает) может быть выбран другой язык, более подходящий для реализации данного модуля (например, язык ассемблера).

На втором шаге разработки программного модуля необходимо выяснить, не известны ли уже какие-либо алгоритмы для решения поставленной и или близкой к ней задачи. И если найдется подходящий алгоритм, то целесообразно им воспользоваться. Выбор подходящих структур данных, которые будут использоваться при выполнении модулем своих функций, в значительной степени предопределяет логику и качественные показатели разрабатываемого модуля, поэтому его следует рассматривать как весьма ответственное решение.

На третьем шаге осуществляется построение текста модуля на выбранном языке программирования. Обилие всевозможных деталей, которые должны быть учтены при реализации функций, указанных в спецификации модуля, легко могут привести к созданию весьма запутанного текста, содержащего массу ошибок и неточностей. Искать ошибки в таком модуле и вносить в него требуемые изменения может оказаться весьма трудоемкой задачей. Поэтому весьма важно для построения текста модуля пользоваться технологически обоснованной и практически проверенной дисциплиной программирования. Впервые на это обратил внимание Дейкстра, сформулировав и обосновав основные принципы структурного программирования. На этих принципах базируются многие дисциплины программирования, широко применяемые на

практике. Наиболее распространенной является дисциплина пошаговой детализации.

Следующий шаг разработки модуля связан с приведением текста модуля к завершенному виду в соответствии со спецификацией качества ПС. При программировании модуля разработчик основное внимание уделяет правильности реализации функций модуля, оставляя недоработанными комментарии и допуская некоторые нарушения требований к стилю программы. При шлифовке текста модуля он должен отредактировать имеющиеся в тексте комментарии и, возможно, включить в него дополнительные комментарии с целью обеспечить требуемые примитивы качества. С этой же целью производится редактирование текста программы для выполнения стилистических требований.

Шаг проверки модуля представляет собой ручную проверку внутренней логики модуля до начала его отладки (использующей выполнение его на компьютере), реализует общий принцип, сформулированный для обсуждаемой технологии программирования, о необходимости контроля принимаемых решений на каждом этапе разработки ПС.

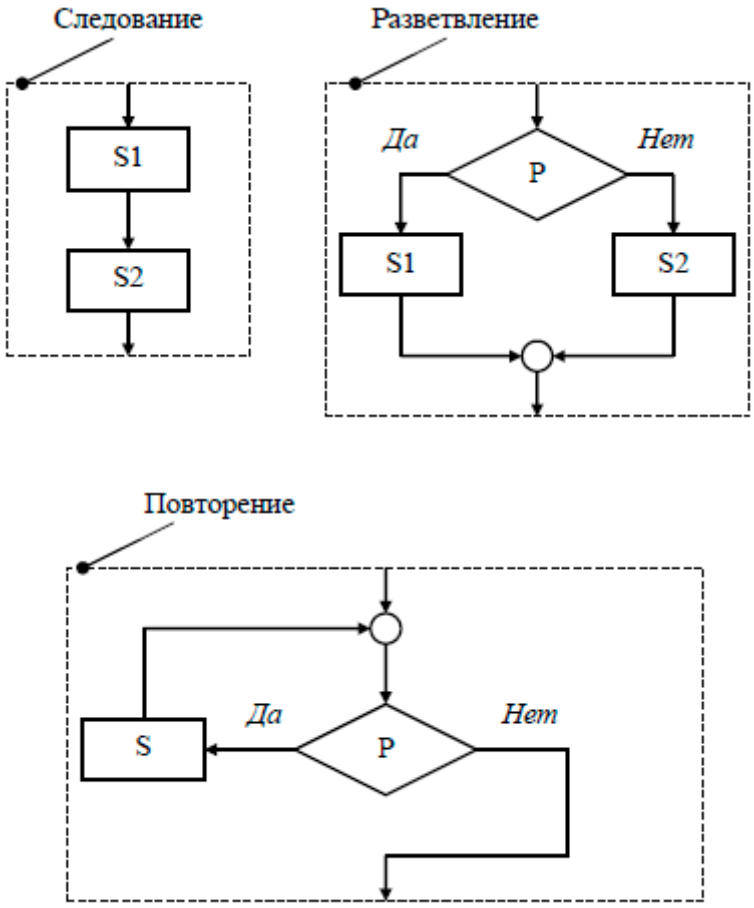
И, наконец, последний шаг разработки модуля означает завершение проверки модуля (с помощью компилятора) и переход к процессу отладки модуля.

Структурное программирование.

При программировании модуля следует иметь в виду, что программа должна быть понятной не только компьютеру, но и человеку: и разработчик модуля, и лица, проверяющие модуль, и тестовики, готовящие тесты для отладки модуля, и сопроводители ПС, осуществляющие требуемые изменения модуля, вынуждены будут многократно разбирать логику работы модуля. В современных языках программирования достаточно средств, чтобы запутать эту логику сколь угодно сильно, тем самым, сделать модуль трудно понимаемым для человека и, как следствие этого, сделать его ненадежным или трудно сопровождаемым. Поэтому необходимо принимать меры для выбора подходящих языковых средств и следовать определенной дисциплине программирования. В связи с этим Дейкстра и предложил строить программу как композицию из нескольких типов управляющих конструкций (структур), которые позволяют сильно повысить понимаемость логики работы программы. Программирование с использованием только таких конструкций назвали структурным.

Основными конструкциями структурного программирования являются: следование, разветвление и повторение. Компонентами этих конструкций являются обобщенные операторы (узлы обработки) S, S1, S2 и условие (предикат) P. В качестве обобщенного оператора может быть либо простой оператор используемого языка программирования (операторы присваивания, ввода, вывода, обращения к процедуре), либо фрагмент программы, являющийся композицией основных управляющих конструкций структурного программирования. Существенно, что каждая из этих конструкций имеет по управлению только один вход и один выход. Тем самым, и обобщенный оператор имеет только один вход и один выход.

Весьма важно также, что эти конструкции являются уже математическими объектами (что, по существу, и объясняет причину успеха структурного программирования). Доказано, что для каждой неструктурированной программы можно построить функционально эквивалентную (т.е. решающую ту же задачу) структурированную программу. Для структурированных программ можно математически доказывать некоторые свойства, что позволяет обнаруживать в программе некоторые ошибки. Этому вопросу будет посвящена отдельная лекция.



Основные управляющие конструкции структурного программирования.

Структурное программирование иногда называют еще «программированием без GO TO». Однако дело здесь не в операторе GO TO, а в его беспорядочном использовании. Очень часто при воплощении структурного программирования на некоторых языках программирования (например, на ФОРТРАНе) оператор перехода (GO TO) используется для реализации структурных конструкций, что не нарушает принципов структурного программирования. Запутывают программу как раз «неструктурные» операторы перехода, особенно переход к оператору, расположенному в тексте модуля выше (раньше) выполняемого оператора перехода. Тем не менее, попытка избежать оператора перехода в некоторых простых случаях может привести к слишком громоздким структурированным программам, что не улучшает их ясность и содержит опасность появления в тексте модуля дополнительных ошибок. Поэтому

можно рекомендовать избегать употребления оператора перехода всюду, где это возможно, но не ценой ясности программы.

К полезным случаям использования оператора перехода можно отнести выход из цикла или процедуры по особому условию, «до-срочно» прекращающего работу данного цикла или данной процедуры, т.е. завершающего работу некоторой структурной единицы (обобщенного оператора) и тем самым лишь локально нарушающего структурированность программы. Большие трудности (и усложнение структуры) вызывает структурная реализация реакции на возникающие исключительные (часто ошибочные) ситуации, так как при этом требуется не только осуществить досрочный выход из структурной единицы, но и произвести необходимую обработку (исключение) этой ситуации (например, выдачу подходящей диагностической информации). Обработчик исключительной ситуации может находиться на любом уровне структуры программы, а обращение к нему может производиться с разных нижних уровней. Вполне приемлемой с технологической точки зрения является следующая «не-структурная» реализация реакции на исключительные ситуации. Обработчики исключительных ситуаций помещаются в конце той или иной структурной единицы, и каждый такой обработчик программируется таким образом, что после окончания своей работы производит выход из той структурной единицы, в конце которой он помещен. Обращение к такому обработчику производится оператором перехода из данной структурной единицы (включая любую вложенную в нее структурную единицу).

Пошаговая детализация и понятие о псевдокоде.

Структурное программирование дает рекомендации о том, каким должен быть текст модуля. Возникает вопрос, как должен действовать программист, чтобы построить такой текст. Часто программирование модуля начинают с построения его блок-схемы, описывающей в общих чертах логику его работы. Однако современная технология программирования не рекомендует этого делать без подходящей компьютерной поддержки. Хотя блок-схемы позволяют весьма наглядно представить логику работы модуля, при их ручном кодировании на языке программирования возникает весьма специфический источник ошибок: отображение существенно двумерных структур, какими являются блок-схемы, на линейный текст, представляющий модуль, содержит опасность искажения логики работы модуля, тем более что психологически довольно трудно сохранить высокий уровень внимания при повторном ее рассмотрении. Исключением может быть случай, когда для построения блок-схем используется графический редактор и они формализованы настолько, что по ним автоматически генерируется текст на языке программирования (как, например, это делается в Р-технологии).

В качестве основного метода построения текста модуля современная технология программирования рекомендует пошаговую детализацию. Сущность этого метода заключается в разбиении процесса разработки текста модуля на ряд шагов. На первом шаге описывается общая схема работы модуля

в обозримой линейной текстовой форме (т.е. с использованием очень крупных понятий), причем это описание не является полностью формализованным и ориентировано на восприятие его человеком. На каждом следующем шаге производится уточнение и детализация одного из понятий (будем называть его уточняемым), в каком-либо описании, разработанном на одном из предыдущих шагов. В результате такого шага создается описание выбранного уточняемого понятия либо в терминах базового языка программирования (т.е. выбранного для представления модуля), либо в такой же форме, что и на первом шаге с использованием новых уточняемых понятий. Этот процесс завершается, когда все уточняемые понятия будут уточнения (т.е. в конечном счете будут выражены на базовом языке программирования). Последним шагом является получение текста модуля на базовом языке программирования путем замены всех вхождений уточняемых понятий заданными их описаниями и выражение всех вхождений конструкций структурного программирования средствами этого языка программирования.

Пошаговая детализация связана с использованием частично формализованного языка для представления указанных описаний, который получил название псевдокода. Этот язык позволяет использовать все конструкции структурного программирования, которые оформляются формализовано, вместе с неформальными фрагментами на естественном языке для представления обобщенных операторов и условий. В качестве обобщенных операторов и условий могут задаваться и соответствующие фрагменты на базовом языке программирования.

Головным описанием на псевдокоде можно считать внешнее оформление модуля на базовом языке программирования, которое

должно содержать:

- начало модуля на базовом языке, т.е. первое предложение или заголовок (спецификацию) этого модуля;
- раздел (совокупность) описаний на базовом языке, причем вместо описаний процедур и функций только их внешнее оформление;
- неформальное обозначение последовательности операторов тела модуля как одного обобщенного оператора (см. ниже), а также неформальное обозначение тела каждого описания процедуры или функции как одного обобщенного оператора;
- последнее предложение (конец) модуля на базовом языке.

Внешнее оформление описания процедуры или функции представляется аналогично. Впрочем, если следовать Дейкстре, раздел описаний лучше также представить здесь неформальным обозначением, произведя его детализацию в виде отдельного описания.

Неформальное обозначение обобщенного оператора на псевдокоде производится на естественном языке произвольным предложением, раскрывающим в общих чертах его содержание. Единственным формальным требованием к оформлению такого обозначения является следующее: это предложение должно занимать целиком одно или несколько графических (печатных)

строк и завершаться точкой (или каким-либо другим знаком, специально выделенным для этого).

Для каждого неформального обобщенного оператора должно быть создано отдельное описание, выражающее логику его работы (детализирующее его содержание) с помощью композиции основных конструкций структурного программирования и других обобщенных операторов. В качестве заголовка такого описания должно быть неформальное обозначение детализируемого обобщенного оператора. Основные конструкции структурного программирования могут быть представлены в следующем виде. Здесь условие может быть либо явно задано на базовом языке программирования в качестве булевского выражения, либо неформально пред ставлено на естественном языке некоторым фрагментом, раскрывающим в общих чертах смысл этого условия. В последнем случае должно быть создано отдельное описание, детализирующее это условие, с указанием в качестве заголовка обозначения этого условия (фрагмента на естественном языке).

В качестве обобщенного оператора на псевдокоде можно использовать указанные выше частные случаи оператора перехода. Последовательность обработчиков исключительных ситуаций (исключений) задается в конце модуля или описания процедуры (функции). Каждый такой обработчик имеет вид:

ИСКЛЮЧЕНИЕ имя_исключения
обобщенный_оператор
ВСЕ ИСКЛЮЧЕНИЕ

Отличие обработчика исключительной ситуации от процедуры без параметров заключается в следующем: после выполнения процедуры управление возвращается к оператору, следующему за обращением к ней, а после выполнения исключения управление возвращается к оператору, следующему за обращением к модулю или процедуре (функции), в конце которого (которой) помещено данное исключение.

Рекомендуется на каждом шаге детализации создавать достаточно содержательное описание, но легко обозримое (наглядное), так чтобы оно размещалось на одной странице текста. Как правило, это означает, что такое описание должно быть композицией пяти-шести конструкций структурного программирования. Рекомендуется также вложенные конструкции располагать со смещением вправо на несколько позиций. В результате можно получить описание логики работы по наглядности вполне конкурентное с блок-схемами, но обладающее существенным преимуществом сохраняется линейность описания.

Идею пошаговой детализации приписывают иногда Дейкстре. Однако Дейкстра предлагал принципиально отличающийся метод построения текста модуля, который нам представляется более глубоким и перспективным. Во-первых, вместе с уточнением операторов он предлагал постепенно (по шагам) уточнять (детализировать) и используемые структуры данных. Во-вторых, на

каждом шаге он предлагал создавать некоторую виртуальную машину для детализации и в ее терминах производить детализацию всех уточняемых понятий, для которых эта машина позволяет это сделать. Таким образом, Дейкстра предлагал, по существу, детализировать по горизонтальным слоям, что является перенесением его идеи о слоистых системах на уровень разработки модуля. Такой метод разработки модуля поддерживается в настоящее время пакетами языка АДА и средствами объектно-ориентированного программирования.

Контроль программного модуля.

Применяются следующие методы контроля программного модуля:

- статическая проверка текста модуля;
- сквозное прослеживание;
- доказательство свойств программного модуля.

При статической проверке текста модуля этот текст просматривается с начала до конца с целью найти ошибки в модуле. Обычно для такой проверки привлекают, кроме разработчика модуля, еще одного или даже нескольких программистов. Рекомендуется ошибки, обнаруживаемые при такой проверке исправлять не сразу, а по завершению чтения текста модуля.

Сквозное прослеживание представляет собой один из видов динамического контроля модуля. В нем также участвуют несколько программистов, которые вручную прокручивают выполнение модуля (оператор за оператором в той последовательности, какая вытекает из логики работы модуля) на некотором наборе тестов.

Доказательству свойств программ посвящена следующая лекция. Здесь следует лишь отметить, что этот метод применяется пока очень редко.

Лекция 10. Модульное тестирование

Уровни процесса верификации

Как уже было сказано ранее, процесс верификации активен в течение практически всего жизненного цикла системы и работает параллельно с процессом разработки. Разработка системы, как правило, идет на различных уровнях: вначале разрабатывается концепция системы, системные требования, затем архитектура системы, ее разбиение на модули, затем разрабатываются отдельные модули. Последовательность этих уровней зависит от типа жизненного цикла, но их состав практически всегда одинаков. Процесс верификации также разбивается на отдельные уровни:

- системное тестирование, в ходе которого тестируется система в целом;
- интеграционное тестирование, в ходе которого тестируются группы взаимодействующих модулей и компонент системы;
- модульное тестирование, в ходе которого тестируются отдельные компоненты.

Технические аспекты методов разработки и проведения тестирования на каждом из трех уровней были рассмотрены в предыдущих лекциях - на каждом из уровней разрабатываются тестовое окружение, автоматизированные тесты, проводятся формальные инспекции. Однако, каждый из этих трех уровней имеет свои организационные особенности, рассмотрению которых будут посвящены следующие три лекции, начиная с данной.

Задачи и цели модульного тестирования

Каждая сложная программная система состоит из отдельных частей - модулей, выполняющих ту или иную функцию в составе системы. Для того, чтобы удостовериться в корректной работе всей системы, необходимо вначале протестировать каждый модуль системы по отдельности. В случае возникновения проблем при тестировании системы в целом это позволяет проще выявить модули, вызвавшие проблему, и устранить соответствующие дефекты в них. Такое тестирование модулей по отдельности получило называние модульного тестирования (unit testing).

Для каждого модуля, подвергаемого тестированию, разрабатывается тестовое окружение, включающее в себя драйвер и заглушки, готовятся тест-требования и тест-планы, описывающие конкретные тестовые примеры.

Основная цель модульного тестирования - удостовериться в соответствии требованиям каждого отдельного модуля системы перед тем, как будет произведена его интеграция в состав системы.

При этом в ходе модульного тестирования решаются следующие основные задачи:

- Поиск и документирование несоответствий требованиям
- Поддержка разработки и рефакторинга низкоуровневой архитектуры системы и межмодульного взаимодействия

- Поддержка рефакторинга модулей
- Поддержка устранения дефектов и отладки

Первая задача - классическая задача тестирования, включающая в себя не только разработку тестового окружения и тестовых примеров, но и выполнение тестов, протоколирование результатов выполнения, составление отчетов о проблемах.

Вторая задача больше свойственна "легким" методологиям типа XP, где применяется принцип тестирования перед разработкой (Test-driven development), при котором основным источником требований для программного модуля является тест, написанный до реализации самого модуля. Однако, даже при классической схеме тестирования модульные тесты могут выявить проблемы в дизайне системы и нелогичные или запутанные механизмы работы с модулем.

Третья задача связана с поддержкой процесса изменения системы. Достаточно часто в ходе разработки требуется проводить рефакторинг модулей или их групп - оптимизацию или полную переделку программного кода с целью повышения его сопровождаемости, скорости работы или надежности. Модульные тесты при этом являются мощным инструментом для проверки того, что новый вариант программного кода выполняет те же функции, что и старый.

Последняя, четвертая, задача сопряжена с обратной связью, которую получают разработчики от тестировщиков в виде отчетов о проблемах. Подробные отчеты о проблемах, составленные на этапе модульного тестирования, позволяют локализовать и устранить многие дефекты в программной системе на ранних стадиях ее разработки или разработки ее новой функциональности.

В силу того, что модули, подвергаемые тестированию, обычно невелики по размеру, модульное тестирование считается наиболее простым (хотя и достаточно трудоемким) этапом тестирования системы. Однако, несмотря на внешнюю простоту, с модульным тестированием сопряжены две проблемы.

Первая из них связана с тем, что не существует единых принципов определения того, что в точности является отдельным модулем.

Вторая заключается в различиях в трактовке самого понятия модульного тестирования - понимается ли под ним обособленное тестирование модуля, работа которого поддерживается только тестовым окружением, или речь идет о проверке корректности работы модуля в составе уже разработанной системы. В последнее время термин "модульное тестирование" чаще используется во втором смысле, хотя в этом случае речь скорее идет об интеграционном тестировании.

Эти две проблемы рассмотрены в двух следующих разделах.

Понятие модуля и его границ. Тестирование классов

Традиционное определение модуля с точки зрения его тестирования: "модуль — это компонент минимального размера, который может быть независимо протестирован в ходе верификации программной системы". В

реальности часто возникают проблемы с тем, что считать модулем. Существует несколько подходов к данному вопросу:

- модуль — это часть программного кода, выполняющая одну функцию с точки зрения функциональных требований;
- модуль — это программный модуль, т.е. минимальный компилируемый элемент программной системы;
- модуль — это задача в списке задач проекта (с точки зрения его менеджера);
- модуль — это участок кода, который может уместиться на одном экране или одном листе бумаги;
- модуль — это один класс или их множество с единым интерфейсом.

Обычно за тестируемый модуль принимается либо программный модуль (единица компиляции) в случае, если система разрабатывается на процедурном языке программирования, или класс, если система разрабатывается на объектно-ориентированном языке.

В случае систем, написанных на процедурных языках, процесс тестирования модуля происходит так, как это было рассмотрено в предыдущих лекциях: для каждого модуля разрабатывается тестовый драйвер, вызывающий функции модуля и собирающий результаты их работы, и набор заглушек - они имитируют поведение функций, которые содержатся в других модулях, не попадающих под тестирование данного модуля. При тестировании объектно-ориентированных систем существует ряд особенностей, прежде всего вызванных инкапсуляцией данных и методов в классах.

В случае объектно-ориентированных систем более мелкое деление классов и использование отдельных методов в качестве тестируемых модулей нецелесообразно, поскольку для тестирования каждого метода потребуется разработка тестового окружения, сравнимого по сложности с уже написанным программным кодом класса. Кроме того, декомпозиция класса нарушает принцип инкапсуляции, согласно которому объекты каждого класса должны вести себя как единое целое с точки зрения других объектов.

Процесс тестирования классов как модулей иногда называют компонентным тестированием. В ходе такого тестирования проверяется взаимодействие методов внутри класса и правильность доступа методов к внутренним данным класса. Здесь возможно обнаружение не только стандартных дефектов, связанных с выходами за границы диапазона или неверно реализованными требованиями, но и специфических дефектов объектно-ориентированного программного обеспечения:

- дефектов инкапсуляции, в результате которых, например, сокрытые данные класса оказываются недоступными для соответствующих публичных методов;
- дефектов наследования, при наличии которых схема наследования блокирует важные данные или методы от классов-потомков;

- дефектов полиморфизма, при которых полиморфное поведение класса оказывается распространенным не на все возможные классы;
- дефектов инстанцирования, при которых во вновь создаваемых объектах класса не устанавливаются корректные значения по умолчанию параметров и внутренних данных класса.

Однако, согласно, выбор класса в качестве тестируемого модуля имеет и ряд сопряженных проблем.

Определение степени полноты тестирования класса. В том случае, если в качестве тестируемого модуля выбран класс, не совсем ясно, как определять степень полноты его тестирования. С одной стороны, можно использовать классический критерий полноты покрытия программного кода тестами: если полностью выполнены все структурные элементы всех методов, как публичных, так и скрытых, то тесты можно считать полными.

Однако существует альтернативный подход к тестированию класса, в котором все публичные методы должны предоставлять пользователю данного класса согласованную схему работы - тогда достаточно проверить типичные корректные и некорректные сценарии работы с данным классом. Т.е., например, в классе, объекты которого представляют записи в телефонной книжке, одним из типичных сценариев работы будет "Создать запись \to искать запись и найти ее \to удалить запись \to искать запись вторично и получить сообщение об ошибке".

Различия в этих двух методах напоминают различия между тестированием черного и белого ящиков, но, на самом деле, второй подход отличается от черного ящика тем, что функциональные требования на систему могут быть составлены на уровне более высоком, чем отдельные классы, и установление адекватности тестовых сценариев требованиям остается на откуп тестировщику.

Протоколирование состояний объектов и их изменений. Некоторые методы класса предназначены не для выдачи информации пользователю, а для изменения внутренних данных объекта класса. Значение внутренних данных объекта определяет его состояние в каждый отдельный момент времени, а вызов методов, изменяющих данные, изменяет и состояние объекта. При тестировании классов необходимо проверять, что класс адекватно реагирует на внешние вызовы в любом из состояний. Однако, зачастую из-за инкапсуляции данных невозможно определить внутреннее состояние класса программными способами внутри драйвера.

В этом случае может помочь составление схемы поведения объекта как конечного автомата с определенным набором состояний. Такая схема может входить в низкоуровневую проектную документацию (например, в составе описания архитектуры системы), а может составляться тестировщиком или разработчиком на основе функциональных требований к системе. В последнем случае для определения всех возможных состояний может потребоваться ручной анализ программного кода и определение его соответствия требованиям.

Автоматизированное тестирование в этом случае может лишь определить, по всем ли выявленным состояниям осуществлялись переходы и все ли возможные реакции проверялись.

Тестирование изменений. Как уже упоминалось выше, модульные тесты - мощный инструмент проверки корректности изменений, внесенных в исходный код при рефакторинге. Однако, в результате рефакторинга только одного класса, как правило, не меняется его внешний интерфейс с другими классами (интерфейсы меняются при рефакторинге сразу нескольких классов). В результате обычных эволюционных изменений системы у класса может меняться внешний интерфейс, причем как по формальным (изменяются имена и состав методов, их параметры), так и по функциональным (при сохранении внешнего интерфейса меняется логика работы методов) признакам. Для проведения модульного тестирования класса после таких изменений потребуется изменение драйвера и, возможно, заглушек. Но только модульного тестирования в данном случае недостаточно, необходимо также проводить и интеграционное тестирование данного класса вместе со всеми классами, которые связаны с ним по данным или по управлению.

Вне зависимости от того, на какие модули, подвергаемые тестированию, разбивается система, рекомендуется изложить принципы выделения тестируемых модулей в плане и стратегии тестирования, а также составить на базе структурной схемы архитектуры системы новую структурную схему, на которой нужно отметить все тестируемые модули. Это позволит спрогнозировать состав и сложность драйверов и заглушек, требуемых для модульного тестирования системы. Такая схема также может использоваться позже на этапе модульного тестирования для выделения укрупненных групп модулей, подвергаемых интеграции.

Подходы к проектированию тестового окружения

Вне зависимости от того, какая минимальная единица исходных кодов системы принята за минимальный тестируемый модуль, существует еще одно различие в подходах к модульному тестированию.

Первый подход к модульному тестированию основывается на предположении, что функциональность каждого вновь разработанного модуля должна проверяться в автономном режиме без его интеграции с системой. При таком подходе для каждого вновь разрабатываемого модуля создаются тестовый драйвер и заглушки, с помощью которых выполняется набор тестов. Только после устранения всех дефектов в автономном режиме производится интеграция модуля в систему и проводится тестирование на следующем уровне. Достоинством данного подхода является более простая локализация ошибок в модуле, поскольку при автономном тестировании исключается влияние остальных частей системы, которое может вызывать маскировку дефектов (эффект четного числа ошибок). Основной недостаток данного метода - повышенная трудоемкость написания драйверов и заглушек, поскольку заглушки должны адекватно моделировать поведение системы в различных ситуациях,

а драйвер должен не только создавать тестовое окружение, но и имитировать внутреннее состояние системы, в составе которой будет функционировать модуль.

Второй подход построен на предположении, что модуль все равно работает в составе системы и если модули интегрировать в систему по одному, то можно протестировать поведение модуля в составе всей системы. Этот подход свойственен большинству современных "облегченных" методологий разработки, в том числе и XP.

В результате применения такого подхода резко сокращаются трудозатраты на разработку заглушек и драйверов - в роли заглушек выступает уже оттестированная часть системы, а драйвер выполняет только функции передачи и приема данных, не моделируя внутреннее состояние системы.

Тем не менее, при использовании данного метода возрастает сложность написания тестовых примеров - для приведения системы в нужное состояние системы заглушек, как правило, требуется только установить значения тестовых переменных, а для приведения в нужное состояние части реальной системы необходимо выполнить сценарий, приводящий в это состояние. Каждый тестовый пример в этом случае должен содержать такой сценарий.

Кроме того, при этом подходе не всегда удается локализовать ошибки, скрытые внутри модуля, которые могут проявиться при интеграции следующих модулей.

Организация модульного тестирования

Модульное тестирование, с точки зрения тестировщика, — это комплекс работ по выявлению дефектов в тестируемых модулях. В эти работы включается анализ требований, разработка тест-требований и тест-планов, разработка тестового окружения, выполнение тестов, сбор информации об их прохождении.

Однако, с точки зрения руководителя группы тестирования (или с точки зрения руководителя проекта, если в нем не выделена отдельная группа тестирования), модульное тестирование является более широким понятием. Для того, чтобы процесс модульного тестирования мог функционировать совместно с другими процессами разработки, он должен включать в себя несколько фаз: планирование процесса, разработку тестов, выполнение тестов, сбор статистики, управление отчетами о выявленных дефектах.

Согласно стандарту IEEE 1008 процесс модульного тестирования состоит из трех фаз, в состав которых входит 8 видов деятельности (этапов).

Фаза планирования тестирования

- Этап планирования основных подходов к тестированию, ресурсное планирование и календарное планирование
- Этап определения свойств, подлежащих тестированию
- Этап уточнения основного плана, сформированного на этапе (1)

Фаза получения набора тестов

- Этап разработки набора тестов
- Этап реализации уточненного плана

Фаза измерений тестируемого модуля

- Этап выполнения тестовых процедур
- Этап определения достаточности тестирования
- Этап оценки результатов тестирования и тестируемого модуля.

Во время этапа планирования основных подходов в качестве входных данных используется общий план проекта (модульное тестирование, как часть проектных работ, должно укладываться в общий график) и требования к системе (для оценки трудоемкости работ и любого планирования необходимо проводить анализ сложности системы на основании требований к ней).

Основные задачи, решаемые в ходе этапа планирования, включают в себя:

- определение общего подхода к тестированию модулей - определяются риски и на их основе - степень полноты и охвата тестирования системы. Определяются источники входных и выходных данных. Определяются технологии проверки результатов тестирования и форматы записи данных о проведенном тестировании. Описывается внешний интерфейс тестируемых модулей и их информационное окружение;
- определение требований к полноте тестирования - определяется необходимая степень покрытия программного кода различных участков тестируемого модуля, определяется подходы к классам эквивалентности (требуется ли тестирование за границами диапазона);
- определение требований к завершению тестирования - определяются условия, проверка которых позволяет утверждать, что тестирование модуля завершено, и условия, при которых дальнейшее тестирование модуля считается невозможным до его изменения и доработки. Примером таких условий может служить достижение определенного уровня покрытия исходного кода тестами и невозможность компиляции модуля соответственно;
- определение требований к ресурсам - для разработки и выполнения тестов, а также для анализа результатов тестирования необходимы ресурсы - как технические (компьютеры и программное обеспечение), так и людские (тестировщики). При решении этой задачи необходимо указывать требования к программному и аппаратному обеспечению, требования к необходимой квалификации людей, а также должно определяться необходимое для проведения количество ресурсов и время их занятости;
- определение общего плана-графика работ - на основании общего плана проекта составляется план работ по модульному тестированию. Основной критерий начала работ по тестированию - готовность модулей, т.е. общий план работ по тестированию

согласуется по датам начала работ с датами окончания работ общего плана разработки.

После завершения этапа планирования начинается этап определения свойств системы, подлежащих тестированию.

Основные задачи, которые решаются в ходе деятельности по определению свойств системы, подлежащих тестированию, включают в себя:

- изучение функциональных требований - определение тестопригодности требований, при необходимости запрашивается уточнение требований;
- определение дополнительных требований и связанных процедур - определение требований, которые не попадают под функциональные требования, но могут быть протестированы на уровне модульного тестирования (например, это могут быть требования к производительности системы, входящие в состав системных требований);
- определение состояний тестируемого модуля - если тестируемый модуль может быть представлен в виде конечного автомата с определенным набором состояний, то каждое состояние должно быть идентифицировано, а также должны быть выделены все требования, относящиеся к этому состоянию;
- определение характеристик входных и выходных данных - для всех данных, которые поступают в модуль, а также выходят из него, должны быть определены форматы, частота поступления, допустимые значения и т.п.;
- выбор элементов, подвергаемых тестированию - в случае, когда не может применяться полное тестирование, необходимо выбрать элементы тестируемого модуля, которые будут подвергаться тестированию. Основной источник информации здесь - данные о рисках, проанализированные на уровне структуры исходного кода тестируемого модуля. Для тестирования в первую очередь должны отбираться элементы с максимальной степенью риска.

И, наконец, в завершение фазы планирования производится уточнение основного плана - уточняется общий подход к тестированию, формулируются специальные и дополнительные требования к ресурсам, составляется детальный план-график работ.

По завершению этих этапов фаза планирования считается оконченной и начинается фаза разработки тестов. При этом процесс разработки тестов подчиняется тем планам и требованиям, которые были созданы на предыдущем этапе. Таким образом, если на первом этапе основную роль выполнял руководитель группы тестирования, то на втором этапе основную роль начинает играть тестировщик, действующий в согласии с указаниями руководителя.

Фаза разработки тестов начинается с собственно разработки набора тестов, который будет использован для тестирования модуля. Основные документы, которые используются на этом этапе: функциональные требования к

модулю, архитектура модуля, список элементов, подвергаемых тестированию, план-график работ, определения тестовых примеров от предыдущей версии модуля (если они существовали) и результаты тестирования прошлой версии (если они существовали).

В ходе этого этапа должны быть решены следующие задачи:

- разработка архитектуры тестового набора - под тестовым набором здесь понимается не набор конкретных тестовых примеров, а общая структура системы тестов для проверки функциональности тестируемого модуля. Организация тестов в такой системе как правило отражает структуру функциональных требований и зачастую представляет собой иерархию, на каждом уровне которой определяется свой набор тестов;
- разработка явных тестовых процедур (тест-требований) - в случае достаточно подробных функциональных требований и четко прописанной концепции разработки тестов явные тестовые процедуры могут и не разрабатываться. Однако, при наличии необходимых ресурсов, разработка тест-требований позволит более четко интерпретировать подвергаемые тестированию функциональные требования - тест-требования снижают риск неоднозначной трактовки функциональных требований;
- разработка тестовых примеров - тестовые примеры должны соответствовать требованиям к полноте тестирования и составляться либо на базе тест-требований, либо на основании функциональных требований. Данный вид деятельности наиболее продолжителен во времени;
- разработка тестовых примеров, основанных на архитектуре (в случае необходимости) - некоторые тестовые примеры основываются не на функциональных требованиях, а на особенностях архитектуры тестируемого модуля. Для разработки тестовых примеров, основанных на архитектуре, необходимо использовать подход стеклянного ящика. Эти тестовые примеры пишутся либо на основе низкоуровневых требований к системе, либо на основе низкоуровневых тест-требований, если они разрабатывались на одном из предыдущих этапов;
- составление спецификации тестовых примеров - результатом деятельности тестировщика в ходе данного этапа составляется документ Test Design Specification (формат которого описан в стандарте IEEE 829 [15]).

На следующем этапе проводится реализация тестов (например, в виде тестового окружения и формализованных описаний тестовых примеров). В ходе этого этапа формируются тестовые наборы данных, которые используются в тестовых примерах, создается тестовое окружение, и также осуществляется интеграция тестового окружения с тестируемым модулем.

После того, как все тесты реализованы, они выполняются на тестовом стенде в ручном или автоматическом режиме. Вне зависимости от вида тестирования в ходе этого этапа решаются две задачи: выполнение тестовых примеров, и сбор и анализ результатов тестирования.

Сбору подлежит следующая информация:

- результат выполнения каждого тестового примера (прошел/не прошел);
- информация об информационном окружении системы в случае, если тест не прошел;
- информация о ресурсах, которые потребовались для выполнения тестового примера.

По результатам анализа этой информации составляются запросы на изменение проектной документации, программного кода тестируемого модуля или тестового окружения.

Этапы разработки (доработки), реализации и выполнения тестов продолжаются до тех пор, пока не будет достигнут критерий завершения модульного тестирования. Примером такого критерия может служить отсутствие не прошедших тестовых примеров при 75% покрытии строк исходного кода.

После прекращения тестирования выполняются работы по оценке проведенного тестирования, в ходе которых:

- описываются отличия реального процесса тестирования от запланированного;
- отличия поведения тестируемого модуля от описанного в требованиях (с целью дальнейшей коррекции требований);
- составляется общий отчет о прохождении тестов, включающий в себя и информацию о покрытии.

В завершение модульного тестирования необходимо проверить, что все созданные в его ходе артефакты - документы, программный код, файлы отчетов и данных - помещены в базу данных проекта, которая хранит все данные, используемые и создаваемые в процессе разработки программной системы.

Лекция 11. Содержание технической документации

Документация является органической, составной частью программного продукта для ЭВМ и требуются значительные ресурсы для ее создания и применения. Тексты и объектный код программ для ЭВМ могут стать программным продуктом только в совокупности с комплексом документов, полностью соответствующих их содержанию и достаточных для его освоения, применения и изменения. Для этого документы должны быть корректными, строго адекватными текстам программ и содержанию баз данных – систематически, структурировано и понятно изложены, для возможности их успешного освоения и использования достаточно квалифицированными специалистами различных рангов и назначения. Качество и полнота отображения в документах процессов и продуктов в жизненном цикле программных средств должны полностью определять достоверность информации для взаимодействия заказчиков, пользователей и разработчиков, а тем самым, корректность функций и достигаемое качество программных продуктов и соответствующих систем. Посредством документов (электронных или бумажных) специалисты взаимодействуют с программными средствами и данными в реализующих их вычислительных машинах, а также между собой.

Существует большая разница между тем, чтобы просто написать и запрограммировать некоторую функцию для индивидуального использования её разработчиком, и тем, чтобы изготовить её как качественный программный продукт, отчуждаемый от разработчиков, поставляемый заказчику и пользователям. Создание программного продукта требует значительных организационных усилий, ибо её документация – это сложный живой организм, подверженный постоянным изменениям, которые могут вноситься многими специалистами. Управление документацией должно непрерывно поддерживать её полноту, корректность и согласованность с программным продуктом. Необходимо обеспечивать возможность достоверного, формально точного общения всех участников проекта ПС между собой, с создаваемым продуктом и с документами для гарантии поступательного развития, совершенствования и применения комплекса программ. Адекватность документации требованиям, состоянию текстов и объектных кодов программ должна инспектироваться и удостоверяться (подписываться) ответственными руководителями и заказчиками проекта. Ошибки и дефекты документов не менее опасны для применения ПС, чем ошибки в структуре, интерфейсах, файлах текстов программ и в содержании данных. Поэтому к разработке, полноте, корректности и качеству документации необходимо столь же тщательное отношение, как к разработке и изменениям текстов программ и данных.

Реализация документов ПС в значительной сте-пени определяет достигаемое качество сложных прог-раммных продуктов, трудоемкость и длительность их создания. Для этого должна формироваться и использоваться регламентированная стратегия, стандарты, распределение ресурсов и планы создания, изменения и применения документов на программы и данные сложных

систем. В общем случае должны быть выделены руководители и коллектив специалистов, которые будут планировать, описывать, утверждать, выпускать, распрост-ранять и сопровождать комплекты документов. Они должны стимули-ровать разработчиков ПС осуществлять непрерыв-ное, полноценное документирование процессов и результатов своей деятельности, а также контролировать полноту и качество исходных, результирующих и отчетных документов ЖЦ ПС. Официальная, описанная и утвержденная стратегия документирования должна устанавливать дисциплину, необходимую для эффективного создания высо-кокачественных документов на продукты и процессы в жизненном цикле ПС.

Назначение технической документации

Техническая документация является составляющей проекта по созданию, внедрению, сопровождению, модернизации и ликвидации информационной системы на всем протяжении жизненного цикла.

Комплекс технических документов, который регламентирует деятельность разработчиков, называется нормативно-методическим обеспечением (НМО). В данный комплекс входят:

- стандарты;
- руководящие документы;
- методики и положения;
- инструкции и т. д.

НМО регламентирует порядок разработки, общие требования к составу и качеству программного обеспечения (ПО), связям между компонентами, определяет содержание проектной и программной документации.

Основным назначением технической документации является обеспечение эффективных процедур разработки и использования информационной системы как программного продукта, а также организация обмена между разработчиками и пользователями ИС.

Таким образом, можно выделить следующие функции технической документации:

- дает описание возможностей системы;
- обеспечивает фиксацию принятых и реализованных проектных решений;
- определяет условия функционирования ИС;
- предоставляет информацию об эксплуатации и обслуживании ИС;
- регламентирует процедуру защиты информации, регулирует права различных групп пользователей;
- определяет возможности модернизации системы.

Перед составлением технической документации необходимо
иметь ответы на следующие вопросы:

- что и зачем должно быть документировано?
- для кого предназначен тот или иной документ?

- какие ошибки может допустить пользователь и что нужно сделать для их устранения?
- как и в каких условиях будет использоваться документ?
- каковы сроки разработки документа?
- как будет обновляться и поддерживаться документация, каковы механизмы и сроки внесения изменений и пересмотра документов и кто ответственен за реализацию этих действий, а также за хранение, неизменность и контроль за исполнением?
- кто будет оценивать документ и как он соотносится с отраслевыми или ведомственными требованиями на сертификацию разработки?

Ответы на эти вопросы должны быть получены на ранних стадиях разработки информационной системы и входить в состав разрабатываемой в рамках проекта документации.

Требования к технической документации

Как правило, к технической документации предъявляются следующие основные требования:

- документы должны быть точными, полными и, по возможности, краткими, иметь четкое и однозначное толкование;
- документация должна создаваться параллельно с разработкой самой информационной системы;
- обязанности по документированию системы лежат на ее разработчике;
- для повышения эффективности работы с документами должны использоваться стандарты, регламентирующие форму и содержание документов.

Исходя из последнего требования к документации, необходимо рассмотреть основные стандарты, которые используются в области информационных систем.

Вопросы для самоконтроля:

1. Что такое нормативно-методическое обеспечение?
2. Каково основной назначение технической документации информационных систем?
3. Каковы функции технической документации?
4. В чьи обязанности входят работы по документированию информационной системы?

РАЗДЕЛ 6. МОДУЛЬНОЕ ТЕСТИРОВАНИЕ

Лекция 12. Методы разработки технической документации

Методы и средства документирования каждой процедуры в стандартах обычно не раскрываются и адресуются к специальным нормативным документам различного уровня. Быстро оснащающиеся различными ме-тодами и инструментальными средствами этапы системного анализа, моделирования и проектирования ПС различных классов и назначения затрудняют стандартизацию этих процессов, достаточную для полной формализации структуры и содержания документов на программы и данные на уровне международных стандартов. Поэтому для этих этапов создаются нормативные документы – шаблоны на уровне стандартов де-факто, использующие, адаптирующие и дополняющие компоненты стандартов де-юре в разумной степени. Такие нормативные документы содержат выделенные фрагменты стандартов ЖЦ ПС и других стандартов, регламентирующих шаблоны программных документов на различных этапах проекта. В результате создаются и применяются проблемно-ориентиро-ванные совокупности методических руководств, отражающие наиболее современные методы, формы и фрагменты документов, для документальной поддержки этапов и процессов жизненного цикла ПС, определенного класса или функционального назначения.

При создании и применении сложных ПС и обеспе-чении их жизненного цикла документами, целесообразно применять выборку из совокупности стандартов, детализирующих частные процессы, работы и документы. В результате на начальном этапе проектирования следует выделять и формировать целесообразный комплект шаблонов документов, обеспечивающих регла-ментирование всех этапов, процессов и документов при создании опре-деленных проблемно-ориентированных проектов ПС. Для создания и реализации положений этих документов должны быть выбраны инструментальные средства, совместно образующие взаимосвязанный комплекс технологической поддержки и автоматизации ЖЦ и не противоречащие, предварительно скомпонованному набору нормативных документов.

Процессы документирования программ и данных входят в весь жизненный цикл сложных систем и ПС. Поэтому организация и реализация работ по созданию документов должны распределяться между специа-листами, ведущими непосредственное и преимущественное создание проектов комплексов программ и специалистами осуществляющими, в основном, разработ-ку, контроль и издание документов. При создании особо сложных систем целесообразно выделение специального кол-лектива, обеспечивающего организацию и реализацию основных сис-темных работ по документообороту ПС. Совокупные затраты на документирование крупных программных продуктов могут достигать 20 – 30% от общей трудоемкости проекта и необходимого числа (десятки) специалистов в жизненном цикле проекта ПС. В более простых случаях, ор-ганизация работ может быть упрощена, затраты на документирование

снижаются приблизительно до 10%, однако всегда целесообразно выделять специалистов, непосредственно ответственных за создание, адекватность и контроль полноценного комплекта документов на программный продукт. Состав и общий объем документов широко варьируется в зависимости от класса и характеристик объекта раз-работки, а также в зависимости от используемой технологии. Наиболее сложному случаю разработки критических ПС реального времени высокого качест-ва соответствует самая широкая номенклатура документов. Такой перечень документов может быть использован как базовый для формирования на его основе состава и шаблонов документов в ос-тальных более простых проектах.

Создание и применение ПС сложных систем сопровождается документированием этих объектов и процес-сов их жизненного цикла для обеспечения возможности освоения и развития функций программных средств и баз данных на любых этапах проекта ПС, а также для обеспечения интерфейса между разработчиками и с пользователями. По своему назначению и ориентации на определенные задачи и группы пользователей, документацию ПС можно разделить на:

- технологическую документацию процессов разработки и обеспечения всего жизненного цикла, включаю-щую подробные технические описания, и подготавливаемую для спе-циалистов, ведущих проектирование, разработку и сопровождение комплексов программ, обеспечивающую возможность отчуждения, детального освоения, развития и корректировки ими программ и данных на всем жизненном цикле ПС;
- эксплуатационную документацию программного продукта – объекта и результатов разработки, создаваемую для конечных пользователей ПС и позволяющую им осваивать и квалифицированно применять эти средства для решения конкретных функциональных задач систем.

Технологическая документация, непосредственно и в наибольшей степени должна отражать процессы жизненного цикла комплексов программ и данных и требования к этим документам. Стандар-ты и нормативные документы, входящие в жизненный цикл проекта ПС, должны регламентировать структуру, состав этапов, работ и доку-ментов ЖЦ ПС. Они должны: формализовать выполнение и документирование конкретных работ при проектировании, разработке и сопровождении ПС; обеспечивать адаптацию документов к характеристикам среды разработки, внешней и операционной системы; регламентировать процессы обеспечения качества ПС и его компонентов, методы и средства их достижения, реальные значения достигнутых показателей качества. Для контроля возможных изменений целесообразно предусматривать и согласовывать с заказчиком специальный документ, регламентирующий правила применения и корректировки номенклатуры, а также состава и содержания документации, поддерживающей ЖЦ ПС.

Эксплуатационная документация должна обеспечивать отчуждаемость программного продукта от первичных поставщиков – разработчиков и возможность освоения и эффективного применения комплексов программ достаточно квалифицированными специалистами – пользователями. Эксплуатационные документы должны исключать возможность некорректного использования ПС за пределами условий эксплуатации, при которых документами гарантируются требуемые показатели качества функционирования ПС. Основная ее задача состоит в фиксировании, полноценном использовании и обобщении результатов функционировании объектов и процессов всего жизненного цикла ПС и системы.

Базой эффективного управления проектом ПС и его документированием должен быть План, в котором задачи исполнителей частных работ согласованы с выделяемыми для них ресурсами, а также между собой по результатам и срокам их достижения. План проекта должен отражать рациональное сочетание целей, стратегий действий, конкретных процедур, доступных ресурсов и других компонентов, необходимых для достижения основной цели с заданным качеством. Планирование реализации проектов и их документирования должно обеспечивать компромисс между характеристиками создаваемой системы и ресурсами, необходимыми на её разработку и применение. Реализация плана зависит от результатов выполнения частных работ и документов и может требовать оперативной корректировки плана. Контроль обеспечивает исходные данные для координации элементов данной организации в соответствии с планом конкретной задачи. Для этого необходимо следить за ходом проекта и документирования на всем протяжении жизненного цикла и сравнивать запланиро-ванные и фактические результаты работ и документы. Контроль яв-ляется органической функцией управления и должен иметь ряд средств регулирования поведения отдельных специалистов и коллектива разработчиков документов в целом.

При подготовке этих планов целесообразно по возможности разделять их цели и функции. План управления разработкой следует ориентировать на организацию специалистов, непосредственно создающих компоненты и ПС в целом, на эффективное распределение и использование ими ресурсов и средств автоматизации. В Плане управления документированием и обеспечением качества ПС внимание специалистов должно акцентироваться на анализе достигнутых результатов разработки, методах и средствах достижения заданных заказчиком характеристик ПС. При планировании и разработке комплекс документации должен проверяться и аттестовываться на полноту в условиях ограниченных ресурсов, на корректность, адекватность и непротиворечивость отдельных документов.

Различия в организационных службах и процедурах, методах и стратегиях приобретения ПС, масштабах и сложности проектов, требованиях систем и методах их разработки влияют на способы разработки, применения и сопровождения документов. Во многих предприятиях используются собственные нормативные шаблоны документов на процессы и продукты ЖЦ ПС,

адаптированные к конкретным условиям разработки и характеристикам создаваемых ПС. В них выделяются состав и шаблоны документов, наиболее важные для проекта и качества создаваемого ПС, а также для конкретных заказчиков и пользователей. Соответствен-но определяются технологические и эксплуатационные документы, для которых регламентируются структура и основное содержание шаблонов документов.

Одна из важнейших задач документирования состоит в том, чтобы увязать четкими эконо-мическими категориями взаимодействие разных специалистов в типовой производственной цепочке: заказчик - разработчик - изготовитель - пользователь документации. Для этого объект потребления - программный продукт, его документация и все процессы взаимодействия в цепочке должны быть связаны системой экономических и технических характеристик, в той или иной степени, использующих основные экономические показатели - реальные затраты ресурсов: финансов, труда и времени специалистов на конечный программный продукт и документы. Сложность документирования, количество и полнота содержания комплекса документов в первую очередь зависят от масштаба – размера проекта ПС, что целесообразно оценивать в начале его ЖЦ. Для решения этой задачи необходимо детально учитывать требуемые ресурсы современных процессов создания, документирования и использования программ различных классов и назначения - встроенных, коммерческих, административных, учебных, уникальных.

Особое внимание в последнее время уделяется совершенствованию и детализации документов, обеспечивающих высокое качество создаваемых ПС, а также возможности их эффективного итерационного развития длительное время в многочисленных версиях. Соответственно должны изменяться документы, отражающие состояние процессов и компонентов проектов. Для этого организация процессов документирования должна обеспечивать гибкое и точное изменение документов – сопровождение и конфигурационное управление версиями и редакциями каждого документа. Эти процессы и поддерживающие их средства автоматизации должны быть адекватными тем, которые применяются при сопровождении непосредственных объектов разработки – комплекса программ и данных. Они должны быть поддержаны организацией контроля, регистрации и утверждения версии каждого документа, в той степени и на том уровне, которые необходимы в данном проекте для определенного документа.

Для хранения, тиражирования и распространения документов, сложных ПС высокого качества, следует выделять группу специалистов, ответственных за контроль, обеспечение и гарантированное сохранение документации. Для критических, важных систем, документация на программы и данные должна храниться и дублироваться на различных типах носителей и эпизодически выводиться на бумажные носители. При определении схемы обеспечения сохранности документации разного содержания, следует учитывать её важность, трудоемкость хранения и возможность аварийного восстановления. Кроме того, должна быть организована служба нормативного контроля, ответственная за соблюдение стандартов, нормативных и руководящих документов при

подготовке документации всеми специа-листами, участвующими в крупном проекте. Эта служба обязана обеспечить унификацию и высокое качество содержания, структуры и оформления шаблонов документов.

Для обеспечения достоверных данных об объектах и процессах управления документами ПС, необходима автоматизированная база данных - информационная система обеспечения и хранения документов проекта. Такая информационная система документооборота представляет собой комплекс формальных и неформальных каналов поэтапного обмена информацией и документами между участниками проекта. Сте-пень формализации документооборота в коллективе специалистов может варьироваться от утверждаемых руководи-телями планов, технических заданий и документов на компоненты и версии ПС до личных бесед между разработчиками.

Формирование требований к документации сложных программных средств

Масштаб проектов комплексов и компонентов программ является одним из важнейших факторов, влияющим на формирование, структуры и содержание документации, поддерживающей весь жизненный цикл ПС. Оценки масштаба проекта ПС должны быть проанализированы и скорректированы для установления в договоре между заказчиками и разработчиками исходного компромиссного масштаба, допустимого для разработки первичных требований к комплексу документации. Некоторые требования могут потребовать изменения (обычно увеличения) масштаба, и соответственно ресурсов на этапах предварительного и детального проектирования для обеспечения возможности реализации требуемой функциональной пригодности. Таким образом, требования к функциональной пригодности, к конструктивным характеристикам ПС, к форматам и структуре документации должны быть согласованы с масштабом проекта и доступными ресурсами для реализации, на ранних этапах его жизненного цикла. Для этого необходимо использовать адекватные методы и единицы измерения масштаба проектов ПС.

Формированию требований к комплексу программ должно сопутствовать создание требований, отражающих его документооборот, вследствие чего эти процессы во многом подобны. Каждый из разработчиков в той или иной степени должен привлекаться к управлению требованиями, как к проекту, так и его документации. Разработчикам необходимо выработать профессиональные приемы для понимания и изложения в документах потребностей заказчиков и пользователей, управления масштабом проекта, построением системы и документации, которые включают:

- команду разработчиков ПС - получает представление о сложности и размере создаваемого продукта и составе его документации;
- менеджеров проекта - базу для расчета содержания спецификаций, графиков, затрат и ресурсов;

- группы тестирования, - планы тестирования, варианты испытаний и процедуры проверок;
- специалистов по сопровождению и поддержке - получают представление о функциональности каждой составной части продукта;
- клиентов отдела маркетинга и специалистов по продажам - должны иметь представление о конечном программном продукте;
- составителей документации, создающих шаблоны документов, руководства для пользователей и справки на основании спецификации требований к ПС - получат проект пользовательского интерфейса;
- специалистов, ответственные за обучение персонала - получат спецификации требований к ПС и документацию для пользователей, а также для разработки обучающих материалов;
- персонал, занимающийся юридической стороной проекта - проверяет, соответствуют ли требования к продукту существующим законам и постановлениям.

Размер или масштаб комплексов программ в настоящее время приводится в публикациях в различных единицах, что может изменять их численные значения для одних и тех же программ в несколько раз. Влияние единиц измерения размера программ на оценку рационального объема документации можно значительно изменяться, если учесть их принципиальные особенности и, прежде всего, выделить две группы единиц измерения масштаба проектов ПС:

- группу, характеризующую размер исходных текстов программ, которые разрабатываются и анализируются специалистами – человеком, отражающую, сложность, трудоемкость и длительность создания ПС, его компонентов и основных документов;
- группу, отражающую размер программ и данных, размещаемых в реализующей (объектной) ЭВМ, и характеризующую объем памяти и производительность ЭВМ, необходимые для рабочего функционирования и исполнения комплекса программ в соответствие с его назначением.

Эти две группы единиц отражают размер программ и документов, с разных позиций, и должны использоваться в зависимости от целей анализа и применения значений масштаба проекта. Для разработки набора документов после оценки масштаба комплекса программ, необходимо выполнить цикл поэтапного определения и формирования совокупности спецификаций требований к компонентам и документации проекта ПС. Первым этапом является создание Концепции проекта ПС и комплекса первичных требований спецификаций к иерархическому набору функций, на которые могут быть разбиты предполагаемые фактические компоненты ПС. В дальнейшем разбиение может структурироваться и детализироваться, формируя упрощенный или более

точный уровень абстракции и взаимодействия компонентов. Цель документа – Концепция, состоит в сборе, анализе и определении высокоуровневых потребностей пользователей, функций и документов программного продукта. Основное внимание должно уделяться возможностям и функциям документов, в которых нуждаются будущие разработчики и пользователи.

После первичной оценки масштаба проекта ПС итерационно выполняется поэтапная разработка спецификаций требований проекта и документов ПС. Ограничения при прогнозировании требований к документам, определяются, прежде всего, имеющимися данными, которые могут быть ис-пользованы в качестве исходных аналогов или обобщенных характеристик. Будучи конечным хранилищем комплекса требований к продукту и документов, спецификация требований к ПС должна быть ясной и понятной, дабы у разработчиков и заказчиков не оставалось ни малейших возможностей для разночтения. Не обязательно составлять спецификацию на документацию для всего продукта до начала разработки, но необходимо зафиксировать требования для каждой составляющей перед ее созданием. При работе над любым проектом необходимо достичь согласованности решений по каждому набору требований и документов до начала их реализации разработчиками. Согласованность решений представляет собой процесс изучения и одобрения документов к ПС. В общем случае для оценки, прогнозирования и обоснования спецификаций требований нового комплекса документов необходимы исходные данные:

- обобщенные характеристики использованных ресурсов для документирования и технико-экономические показатели завершенных разработок - прототипов ПС, а также оценки влияния на них функций, различных факторов объектов и среды разработки;
- реализованные планы документирования и обобщенные перечни выполненных работ, реальные графики проведенных ранее оценок и разработок, различных ПС и документов;
- цели и содержание частных работ в процессе создания предыдущих, сложных комплексов программ и набора различных документов для обеспечения необходимого качества ПС в целом;
- структура и содержание полного комплекта документов, являвшегося результатом выполнения отдельных работ конкретного проекта.

Составлять спецификацию требований к документации ПС следует таким образом, чтобы все заинтересованные в проекте лица смогли в ней разобраться и применять:

- разделы, подразделы и отдельные требования должны быть названы согласованно;

- полезно использовать средства визуального выделения (такие, как полужирное начертание, подчеркивание, курсив и различные шрифты) последовательно, иерархически и в разумных пределах;
- создавайте оглавление, а также алфавитный указатель, чтобы облегчить пользователям поиск необходимой информации;
- нумеровать все рисунки и таблицы, ссылки на них, используя присвоенные номера.

Для устранения или снижения ошибок состава, содержания и рисков документации до допустимых пределов может быть необходимо изменение требований к функциональной пригодности и/или к конструктивным характеристикам и доступным ресурсам проекта ПС. Поэтому на этапах проектирования последовательно должны определяться:

- при проектировании концепции и первичной оценке масштаба проекта - предварительные требования к назначению, функциональной пригодности, к составу и номенклатуре необходимых конструктивных характеристик качества и к первичному перечню набора документов ПС;
- при предварительном проектировании - уточненная оценка масштаба проекта, требования к функциональным и конструктивным характеристикам качества, к ориентировочной структуре и содержанию набора шаблонов документов в жизненном цикле ПС с учетом общих ограничений ресурсов;
- при детальном проектировании - подробные спецификации требований к функциональным и конструктивным характеристикам качества с детальным учетом и распределением реальных ограниченных ресурсов, а также полный состав и содержание шаблонов документов в жизненном цикле ПС.

Чем крупнее и сложней проект ПС и соответственно выше его стоимость, тем тщательнее следует разрабатывать требования к его характеристикам, к составу и содержанию документов, а также распределять ресурсы на их реализацию. Поэтому при средней и относительно невысокой сложности ПС во многих случаях можно удовлетвориться подготовкой требований к комплексу программ с подробностью анализа, соответствующего предварительному проектированию. Для крупных и особо сложных проектов необходим более детальный анализ факторов при разработке набора документов.

В зависимости от сложности проекта окончательным результатом работ должны быть детализированные и утвержденные документы спецификаций к номенклатуре, свойствам, значениям качества и документации проекта ПС, которые достаточны для его полноценного рабочего проектирования и последующей, эффективной эксплуатации. Эти требования к документам закрепляются в контракте и техническом задании, по которым разработчик

впоследствии должен отчитываться перед заказчиком при завершении проекта. Однако на последующих этапах жизненного цикла и при конфигурационном управлении, документы могут изменяться по согласованию между заказчиком и разработчиком, которые чаще всего приурочиваются к подготовке новой версии программного продукта. Для этого необходим мониторинг масштаба проекта, комплекта документов, спецификаций требований и реализаций характеристик в течение всего ЖЦ ПС.

Требования к функциональным характеристикам, качеству, составу и содержанию документов, утвержденные после предварительного проектирования, могут быть закреплены в техническом задании как обязательные для детального и рабочего проектирования. Эти данные могут использоваться при последующем оценивании качества документации при их сопоставлении с требованиями в процессе квалификационных испытаний или сертификации программного продукта. Однако для крупномасштабных и дорогих проектов может потребоваться уточнение требований к качеству документации при детальном проектировании с позиции улучшения соотношения значений качества и затрат ресурсов, которые необходимы или допустимы для их реализации в ЖЦ ПС.

Для заказчика и пользователей может иметь значение не только определение функциональной пригодности, но и оценка потенциального спроса на рынке конкретного программного продукта, а также его конкурентоспособности с другими аналогичными по функциям ПС с учетом его качества и стоимости. Это обстоятельство может определять необходимость уточнения требований к отдельным характеристикам и документам не только для их реализации разработчиками в ЖЦ ПС, но также для оценивания интегрального качества и документации готового программного продукта, поставляемого на рынок.

Обычно заказчики и разработчики первоначально устанавливают требования к каждой характеристике и к номенклатуре документов ПС без учета относительных затрат на их достижение, а также без детального анализа их совместного влияния на полную функциональную пригодность у потребителей. Это может приводить к значительным перекосам и к несбалансированным значениям требований к отдельным, взаимосвязанным характеристикам и документам, на которые не рационально используются ограниченные ресурсы ЖЦ ПС, или к не адекватно низким их значениям. В проектах крупных ПС это может угрожать значительным повышением стоимости и/или снижением конкурентоспособности создаваемого программного продукта из-за недостаточного уровня отдельных показателей качества и документов.

Атрибуты качества ПС и полезность документов имеют различные меры, вследствие чего они в большинстве своем непосредственно не сопоставимы между собой. Они предварительно выбираются и согласовываются с заказчиком при последовательном, почти независимом анализе каждого документа, для использования в контракте и техническом задании. Для обобщенного оценивания качества ПС необходим учет относительного влияния

каждой конструктивной характеристики и документа, на функциональную пригодность. При этом не всегда учитываются ресурсы для их реализации в конкретном ПС. Это часто приводит к не рациональным требованиям и документам, которые значительно отличаются: либо по степени влияния на функциональную пригодность, либо по величине ресурсов, необходимых для их реализации как полноценных документов.

Для эффективного управления документацией сложного ПС при детальном проектировании целесообразно учитывать и обобщать степень полезности разнородных характеристик конкретных документов в некоторый интегральный показатель, отражающий их совокупное влияние на его функциональную пригодность. Таким образом, при разработке требований к характеристикам документов проекта выявилась проблема анализа системной эффективности документации и обобщения их характеристик, а также оценивания совместного влияния различных документов на функциональную пригодность ПС с учетом затрат на их реализацию.

Для управления и сопоставительного оценивания выбранных характеристик качества документов целесообразно каждому из них присваивать коэффициент или приоритет влияния на функциональную пригодность. Аналогично, экспертами целесообразно оценивать относительные ресурсы, которые следует затрачивать на реализацию каждого документа. Для каждого вида документов отношение коэффициента влияния на функциональную пригодность к относительным затратам на его достижение можно рассматривать как обобщенный уровень приоритета реализации этого документа. Для конкретного проекта ПС состав и значения приоритетов документов следует поэтапно адаптировать и уточнять с учетом их назначения и функций. Если затраты на разработку документации ПС можно оценивать и прогнозировать с некоторой достоверностью, то эффективность применения и особенно будущий спрос на конкретные документы комплекса программ со стороны различных пользователей априори оценить трудно. Такие оценки могут проводиться на основе специальных маркетинговых исследований и опыта эксплуатации аналогичных комплексов программ или достаточно близких их прототипов.

Планирование документирования проектов сложных программных средств

Общее руководство процессом документирования комплексов программ можно разделить на два уровня:

- адаптация состава и содержания документов к данной деловой, проблемно-ориентированной об-ласти, например, авиационной, медицинской, военной, финансовой или административной;
- адаптация номенклатуры, структуры и содержания документов для каждого специфического проекта, контракта или предп-рия-тия.

В соответствии со стандартами план документирования в виде совокупности руководящих, промежуточных и отчетных документов должен разрабатываться системными аналитиками и утверждаться менеджером проекта вместе со спецификацией требований к ПС. В спецификации формали-зуются требования к результатам документирования, а в плане – методы и средства их достижения. Тем самым характеристики ПС не только декларируются в виде требований, но и сопровождаются совокупностью рекомендуе-мых мероприятий и документов по их обеспечению и реализации.

Достоверность прогнозов требующихся ресурсов для документирования зависит, прежде всего, от точности оценки исходных данных. Такие оценки зависят от компетенции и объективности экспертов, их оптимистичности, пессимистичности и знания существенных особенностей проекта. Оценивая масштаб, функции и требования к документации, заказчик и разработчики должны представлять тот объем затрат и физический размер комплекса документации, который придется создать в процессе всего жизненного цикла ПС, а также для обеспечения его эффективного применения.

Может представлять интерес оценка ориентировочного физического объема документации (например, в стандартных страницах А4 или эквивалентных объемов файлов) для проектов комплексов программ. В качестве гипотетического примера выделим два масштаба проектов: малый – 50 тысяч строк и крупный – один миллион строк, и выделим оценки на технологическую и на эксплуатационную документацию. В эксплуатационной документации обычно не оформляются и не приводятся спецификации компонентов, тексты программ с комментариями, тесты и результаты тестирования, что резко сокращает номенклатуру документов до трех – семи видов. Каждое описание, руководство или инструкция может содержать до 100 страниц текста, что в совокупности дает до тысячи страниц эксплуатационных документов. Для крупного программного продукта несколько возрастает номенклатура документов, но главное, пропорционально увеличению сложности и масштаба комплекса программ до 10 6 строк, объем эксплуатационной документации может увеличиться до 10 тысяч страниц.

Номенклатура технологических документов в жизненном цикле крупного ПС может доходить до 50 видов, среди которых наибольшее влияние на объем документации оказывают: спецификации программ и данных, тексты программ с комментариями, тестовые сценарии и результаты тестирования компонентов и модулей. Для отражения совокупности этих документов, в среднем на каждую строку текста программы может требоваться от 10% до полной страницы документации. Остальные документы в основном являются интегральными для проекта и вряд ли займут более 10% общего объема от перечисленных категорий документов. Таким образом, технологическая документация для жизненного цикла ПС размером 10 6 строк может составить около ста тысяч страниц или ста томов по тысяче страниц.

Вряд ли целесообразно изготавливать такой объем твердых копий документов на бумаге. Большая часть этих документов может оставаться в файлах (сотни мегабайт), однако каждый документ должен быть оформлен в соответствии со стандартами и шаблонами, и скреплен подписями разработчиков и, где нужно, заказчика. Изменение этих документов должно санкционироваться, также как твердых копий. Для относительно малого проекта ПС (50 тысяч строк) с минимальной технологической документацией может потребоваться около 5 тысяч страниц или эквивалентных файлов.

Приведенные оценки следует рассматривать только как ориентиры, которые в реальных проектах могут изменяться на порядок в ту или иную сторону, в зависимости от требований заказчика и характеристик проекта. Оценки объема предстоящей разработки технологической и эксплуатационной документации целесообразно проводить на этапе детального проектирования с учетом реальных характеристик проекта ПС, что позволит избежать неприятных сюрпризов вызванных превышением затрат на реализацию документов.

Менеджер проекта для оценок объема и содержания документации должен подготовить план выполнения документирования в жизненном цикле ПС. Этот план должен содержать описания соответствующих работ и задач и обозначения создаваемых программных продуктов и документов. Он должен охватывать следующие задачи:

- установление графиков и сроков своевременного решения задач документирования;
- оценку необходимых трудозатрат на создание каждого документа и всего комплекса;
- определение времени, необходимого для выполнения конкретных задач документирования;
- распределение задач документирования по исполнителям;
- определение обязанностей исполнителей по созданию содержания документов;
- выделение критических ситуаций, связанных с задачами или самим процессом документирования;
- установление критериев управления и обеспечение качества документов;
- обеспечение внешних условий и определение инфраструктуры проекта системы для выполнения процесса документирования.

В проекте ПС должен быть определен базовый график выполнения работ в жизненном цикле, а графики создания отдельных документов должны быть связаны и согласованы с этим базовым графиком. Менеджер каждого программного проекта должен стремиться по возможности использовать существующую организационную инфраструктуру документирования на предприятии. Должен быть определен механизм для разрешения или преодоления конфликтных ситуаций между менеджером всего проекта ПС и

администратором процесса документирования на соответствующем уровне их полномочий по организационному управлению. Это положение является общим для всех контрольных этапов, связанных с выполнением договорных обязательств по вспомогательным процессам, поэтому необходимы синхронизация соответствующих планов и своевременное уведомление менеджера ПС о всех затруднениях, возникающих при выполнении соответствующих задач процессов документирования.

В интересах сокращения стоимости и улучшения качества, стандарты и регламентируемый ЖЦ ПС рекомендуется адаптировать к характеристикам конкретного проекта. Соответственно сформированному жизненному циклу следует адаптировать состав документов конкретного проекта ПС. Для адаптации состава и содержания документации, должны быть определены характеристики окружения проекта, которые могут воздействовать на адаптацию документирования: процессы жизненного цикла создаваемой системы; требования к системе и программному средству; организационные процедуры и стратегии документирования; размер, критичность и функции основных компонентов системы; количество задействованного в проекте персона-ла и сторон.

В плане управления документированием каждого этапа жизнен-ного цикла ПС целесообразно фиксировать и документально оформлять:

- исходные данные (шаблоны), требующиеся для успешного выполнения дан-ного этапа документирования проекта или компонента ПС;
- контролируемые и документируемые данные о состоянии объекта и процесса разработки, регистрируемые после завершения этапа;
- содержание процедур контроля состояния проекта и документов в процессе выполнения работ этапа;
- критерии оценки результатов выполненных работ и качества отчетных документов при завершении этапа;
- состав и содержание отчетных документов (шаблонов), представляемых для оценки состояния проекта, результатов завершенного этапа и работ и для использования на следующем этапе или при завершении проекта ПС.

Менеджер программного проекта должен отвечать за проведение оценок программных продуктов, документов и планов: на соответствие принципам, методологии и технологии управления проектом и документированием; в части удовлетворения их установленным требованиям договора с заказчиком. Необходимыми компонентами успешной реализации программных проектов являются периодические анализы и оценки хода выполнения и завершения этапов и заданий на документирование.

Планирование качества доку-ментов в ряде стандартов принято отделять от планов непосредственного управления процессом создания комплекса программ. Для реализации планов качественного документирования должны быть созданы регламентирующие документы, охватывающие:

- процессы создания документов, отражающих качество программного продукта;
- обязанности и ответственность специалистов за качество конкретных документов;
- используемые ресурсы, обеспечивающие создание документов высокого качества;
- требования к качеству конкретных документов и способы его контроля.

Реальные ограничения ресурсов, используемых в процессе разработки, квали-фикация специалистов, изменения внешней среды и требований заказчика объективно приводят к отклонениям реализации плана документирования от предполагавшегося. Величина таких отклонений в значительной степени зависит от принятой технологии разработки, от уровня и характеристик средств разработки, а также от средств автоматизации создания программ. Для своевременного обнаружения отклонений от плана документирования необходимо регулярно регистрировать результаты выполненных работ и их характеристики качества. Для реализации таких измерений целесообразно предусмотреть и согласовать с заказчиком специальный документ, регламентирующий правила корректировки плана обеспечения качества ПС, а также состава и содержания поддержива-ющей его документации.

Адаптация номенклатуры и содержания документов ПС к особенностям системы и пользователей может базироваться на выборе подходящих шаблонов из набора документов. В процессе адаптации состава и содержания документов должны быть учтены особенности пользователей, поддерживающего персонала, руководи-телей контракта, потенциальных покупателей. Процессы, работы, шаблоны и задачи ЖЦ должны селектироваться и включать в себя перечень документов, которые обязательно нужно разработать и информацию о персональной ответственности за них. Выбранные процессы, работы и задачи, не обеспечиваемые конкретными документами, следует оговаривать в самом контракте и утвержденном ЖЦ ПС. При адаптации документиро-вания ПС необходимо также учитывать особенности проекта: стоимость, планирование, производительность специалис-тов, размеры проекта и интерфейс с человеком-пользователем. Все исходные дан-ные и решения по адаптации номенклатуры, структуры и содержания документов должны быть документированы и утверждены руководством проекта вместе с обоснованием их целесообразности.

Для реализации на практике требований и планов документирования в жизненном цикле программных средств необходимы организационные мероприятия, гарантирующие участникам проектов определенную культуру, дисциплину разработки и применения документов. Такая организационная система должна обеспечивать специалистам разной квалификации и специализации, возможность взаимодействия при решении требуемых комплексных задач, для накопления, хранения и обмена упорядоченной информацией о состоянии и изменениях компонентов проекта. Формализованная в документах информация должна повышать ответственность специалистов за корректность её содержания, оперативность формирования и изменения, качество процессов трансформации и реализации в жизненном цикле ПС.

Для этого в каждом проекте комплекса программ должен быть организован регламентированный процесс документооборота, обеспечивающий для коллектива специалистов единую среду разработки, изменения и утверждения документов, адекватных реальному содержанию объектного кода программ и текстовых данных файлов ПС. Процесс организации и технологического обеспечения документооборота должен быть ориентирован на слаженную, коллективную работу различных профессионалов, объединенных единой целью создания требуемого заказчиком комплекса программ с заданными функциями и высоким качеством документации. Каждый участник проекта в соответствии со своими функциональными обязанностями должен иметь доступ к необходимой для него корректной информации, и ограничен возможностями обращения к несанкционированным для него данным. Должны быть упорядочены деловые коммуникации между специалистами разных категорий, управление динамическими процессами изменений и транспортировки документов между подсистемами документооборота.

Первоначально должен быть разработан проект архитектуры системы документооборота и руководство по её применению, настроена выбранная СУБД на управление взаимодействующими подсистемами документооборота, с соответствующими комплектами выбранных шаблонов документов, с учетом класса и масштаба предполагаемого ЖЦ ПС. Шаблоны подсистем документооборота должны поэтапно заполняться реальными данными проекта от заказчика и разработчиков соответствующих квалификаций, и контролироваться менеджерами проекта. При этом следует управлять динамикой процессов реализации процедур документооборота, регистрировать реальное использование ресурсов специалистов, текущее время выполнения процедур процессов ЖЦ ПС и оформления документов в подсистемах. В реальных проектах ПС возможны отклонения от такой линейной схемы двух типов:

- прерывание процессов документооборота и прекращения всей разработки после промежуточных этапов системного, предварительного или детального проектирования, вследствие, недостаточности ресурсов для его полной реализации или вследствие отказа

заказчика продолжать данный проект при появлении альтернативного варианта программного продукта;

- итерационный возврат на предшествующие подсистемы документооборота, а также на их компоненты и шаблоны для корректировки и уточнения, обусловленные изменениями компонентов ПС.

При наличии адекватных прототипов и ряда детальных спецификаций требований, для создания новой базовой версии программного продукта может отсутствовать необходимость её системного, предварительного или детального проектирования, а разработка и тестирование новых программных компонентов выполняться в процессе расширения функций предшествовавшей базовой версии. Тем самым процессы и руководства документооборота при развитии и совершенствовании версий ПС могут формализоваться, начиная с некоторых промежуточных этапов жизненного цикла комплекса программ.

В состав базовых версий программного продукта входит седьмая подсистема документооборота и комплект документов пользователей. Этот комплект и содержание документов также следует адаптировать в соответствии с требованиями и характеристиками проекта. В системах реального времени в ряде случаев могут отсутствовать документы административного управления, а также сокращены в шаблонах требования и функции оперативных пользователей. В комплексах программ административных систем может доминировать документооборот администраторов, действующих совместно с оперативными пользователями при реализации основных функций программного продукта. В некоторых автоматизированных системах реального времени программный продукт является органическим компонентом системы управления и внешней среды, и эксплуатационная документация должна отражать в основном контрольные функции, инсталляцию и оценки целостности функционирования программного продукта в системе.

РАЗДЕЛ 7. ДОКУМЕНТИРОВАНИЕ

Лекция 13. Средства разработки технической документации

Одним из главных отличий программы от программного продукта является наличие разнообразной, хорошо подготовленной документации. Составление программной документации - важный процесс. На каждый программный продукт должна разрабатываться документация двух типов: для пользователей различных групп и для разработчиков. Отсутствие документации любого типа для конкретного ПО не допустимо. При подготовке документации не следует забывать, что она разрабатывается для того, чтобы ее использовали, и потому она должна содержать все необходимые сведения.

К программным относят документы, содержащие сведения, необходимые для разработки, сопровождения и эксплуатации программного обеспечения. Документирование программного обеспечения осуществляется в соответствии с Единой системой программной документации (ГОСТ 19.XXX). Так ГОСТ 19.101-77 устанавливает виды программных документов для ПО различных типов. К основным программным документам по этому стандарту относятся: спецификация, ведомость держателей подлинников, текст программы, описание программы, ведомость эксплуатационных документов, формуляр, описание применения, руководство системного программиста, руководство программиста, руководство оператора, описание языка, руководство по техническому обслуживанию, программа и методика испытаний, пояснительная записка. Допускается объединять отдельные виды эксплуатационных документов. Необходимость объединения указывается в техническом задании, а имя берут у одного из объединяемых документов. При оформлении текстовых и графических материалов, входящих в программную документацию также следует придерживаться действующих стандартов. Рассмотрим подробнее некоторые из перечисленных документов.

Самым главным документом для разработчиков является техническое задание (ТЗ), в котором описываются цели и задачи работы, заказчик и исполнители, технические требования, сроки и этапы, требования секретности, форс-мажорные обстоятельства и правила предъявления результатов. ТЗ должно быть составлено таким образом, чтобы исключить возможные разночтения, все требования должны быть сформулированы так, чтобы их можно было проверить однозначным образом.

Следующим по важности документом является программа и методика испытаний (ПМИ). Структурно она подобна ТЗ – практически для каждого пункта ТЗ в ПМИ говорится, как этот пункт будет проверяться. Способы проверки могут быть самыми разными – от пропуска специального теста до изучения исходных текстов программы, но они должны быть предусмотрены заранее, а не придумываться в момент испытаний. Новички приступают к составлению ПМИ непосредственно перед завершением работ, а опытные руководители составляют и согласовывают с заказчиками одновременно с ТЗ. Хорошо составленная ПМИ является гарантией успешной сдачи работ.

Руководство системного программиста, на современном языке называется руководством по инсталляции. В нем описывается порядок установки системы, как проверить корректность поставленной системы, как вносить изменения и т.п. Обычно это простой короткий документ.

Руководство оператора (пользователя) – это основной документ, описывающий, как пользоваться системой. В хорошем руководстве сначала описывается идея системы, основные функции и как ими пользоваться, а уже потом идет описание всех клавиш и меню.

Руководство программиста — это самый объемный документ, описывающий внутреннюю организацию программы. Обычно этот документ идет в паре с документом "текст программы" – одностраничным документом с оглавлением дискеты или CD. Руководство программиста дает заказчику возможность дописать новые фрагменты программы или переделать старые. В современной литературе этот документ называется SDK (Software Development Kit). Продукт, снабженный SDK, может стоить на порядок дороже, чем такой же продукт без него. Сейчас не принято продавать исходные тексты программ – проблемы с интеллектуальной собственностью, даже при наличии SDK трудно "влезть" в чужую программу – как говорится, себе дороже. Поэтому большое распространение получили API (Application Program Interface). Программа передается только в виде DLL (библиотека двоичных кодов), но известно, как обратиться к каждой функции из других программ, т.е. известно имя точки входа, количество, типы и значения параметров. Наличие множества API, конечно, хуже, чем наличие исходных текстов (например, нельзя переделать что-то в середине функции), зато много проще в использовании. С другой стороны, все большую популярность приобретает FSF (Free Software Foundation). Основателем этого движения был Ричард Столман, который забил тревогу по поводу попыток крупных фирм запатентовать многие основные алгоритмы и программы: "Дойдет до того, что они запатентуют понятия "цикл" и "подпрограмма", что мы будем тогда делать?" FSF представляет собой собрание программ в исходных текстах; любой программист может свободно использовать их в своих целях, но все добавления и улучшения, которые он сделал, тоже следует положить в FSF. Таким образом, FSF представляет собой одно из самых больших доступных хранилищ программ.

Основные вопросы при разработке программных средств

Когда программист-разработчик получает в той или иной форме задание на программирование, перед ним, перед руководителем проекта и перед всей проектной группой встают вопросы:

- что должно быть сделано, кроме собственно программы?
- что и как должно быть оформлено в виде документации?
- что передавать пользователям, а что службе сопровождения?
- как управлять всем этим процессом?
- что должно входить в само задание на программирование?

Кроме упомянутых вопросов есть и другие.

На эти и массу других вопросов когда-то отвечали государственные стандарты на программную документацию комплекс стандартов 19-й серии ГОСТ ЕСПД. Но уже тогда у программистов была масса претензий к этим стандартам. Что-то требовалось дублировать в документации много раз (как, казалось - неоправданно), а многое не было предусмотрено, как, например, отражение специфики документирования программ, работающих с интегрированной базой данных.

В настоящее время остается актуальным вопрос о наличии системы, регламентирующей документирование программных средств (ПС).

Общая характеристика состояния

Основу отечественной нормативной базы в области документирования ПС составляет комплекс стандартов Единой системы программной документации (ЕСПД). Основная и большая часть комплекса ЕСПД была разработана в 70-е и 80-е годы. Сейчас этот комплекс представляет собой систему межгосударственных стандартов стран СНГ (ГОСТ), действующих на территории Российской Федерации на основе межгосударственного соглашения по стандартизации.

Говоря о состоянии ЕСПД в целом, можно констатировать, что большая часть стандартов ЕСПД морально устарела.

К числу основных недостатков ЕСПД можно отнести:

- ориентацию на единственную, "каскадную" модель жизненного цикла (ЖЦ) ПС;
- отсутствие четких рекомендаций по документированию характеристик качества ПС;
- отсутствие системной увязки с другими действующими отечественными системами стандартов по ЖЦ и документированию продукции в целом, например, ЕСКД;
- нечетко выраженный подход к документированию ПС как товарной продукции;
- отсутствие рекомендаций по самодокументированию ПС, например, в виде экранных меню и средств оперативной помощи пользователю ("хелпов");
- отсутствие рекомендаций по составу, содержанию и оформлению перспективных документов на ПС, согласованных с рекомендациями международных и региональных стандартов.

Итак, ЕСПД нуждается в полном пересмотре на основе стандарта ИСО/МЭК 12207-95 на процессы жизненного цикла ПС об этом стандарте далее будет сказано подробнее).

Надо сказать, что наряду с комплексом ЕСПД официальная нормативная база РФ в области документирования ПС и в смежных областях включает ряд перспективных стандартов (отечественного, межгосударственного и международного уровней).

Международный стандарт ISO/IEC 12207: 1995-08-01 на организацию ЖЦ продуктов программного обеспечения (ПО) - казалось бы весьма неконкретный, но вполне новый и отчасти "модный" стандарт.

Краткое представление стандартов ЕСПД

Тем не менее, до пересмотра всего комплекса, многие стандарты ЕСПД могут с пользой применяться в практике документирования ПС. Эта позиция основана на следующем:

- стандарты ЕСПД вносят элемент упорядочения в процесс документирования ПС;
- предусмотренный стандартами ЕСПД состав программных документов вовсе не такой "жесткий", как некоторым кажется: стандарты позволяют вносить в комплект документации на ПС дополнительные виды
- стандарты ЕСПД позволяют вдобавок мобильно изменять структуры и содержание установленных видов ПД исходя из требований заказчика и пользователя.

При этом стиль применения стандартов может соответствовать современному общему стилю адаптации стандартов к специфике проекта: заказчик и руководитель проекта выбирают уместное в проекте подмножество стандартов и ПД, дополняют выбранные ПД нужными разделами и исключают ненужные, привязывают создание этих документов к той схеме ЖЦ, которая используется в проекте.

Лекция 14. Автоматизированные средства оформления документации

Процессы разработки (сопровождения и т.д.) технической документации всегда и везде идут плечом к плечу с процессами разработки изделий, программных изделий, создания автоматизированных систем. В крупных организациях в процессе разработки техдокументации задействовано, как правило, значительное число специалистов различных подразделений. В мелких и средних компаниях техническая документация «составляется» «узким кругом ограниченных лиц», именующих себя техническими писателями (техписами) - представителями профессии, по сей день не нашедшей себе достойного места в общероссийском классификаторе.

Как бы то ни было, для большинства компаний разного калибра процесс разработки (сопровождения и т.д.) техдокументации остается занятием в немалой степени рутинным, трудоемким и ресурсоемким и, как правило, неблагодарным по отношению к непосредственным исполнителям.

Речь пойдет об автоматизации разработки (сопровождения и т.д.) технической документации для автоматизированных систем. Итак, целями автоматизации разработки являются:

- повышение управляемости жизненного цикла техдокументации;
- снижение трудоемкости и ресурсоемкости процессов жизненного цикла технической документации.
- Задачи:
- показать взаимосвязь документов (а также разделов, подразделов, пунктов, подпунктов и т.д.), входящих в состав техдокументации, разрабатываемой на стадиях и этапах создания автоматизированных систем;
- дать практические рекомендации по автоматизации процессов жизненного цикла технической документации.

Техническая документация: ее назначение (и отношение к ней)

В компаниях, чья продукция может повлечь за собой гибель людей или нанесение серьезного материального ущерба, к составу и содержанию техдокументации относятся более или менее ответственно, помня о прокуроре. Таким образом, техдокументация в указанных компаниях, разрабатывается, как минимум:

- во избежание ответственности перед законом - на авиажаргоне такой подход называется «прикрытием задней полусферы»;
- для исключения возможных формальных претензий со стороны заказчика.

Указанный подход обеспечивает все предпосылки к разработке технической документации высочайшего качества. Ясно, что нести судебную ответственность и выплачивать денежную компенсацию за гибель животного американке, выстиравшей свою кошку в стиральной машине и высушившей

любимицу в микроволновой печи, - дураков нет. Требуются лишь усилия разработчиков-профессионалов, помноженные на реализацию ими требований государственных стандартов.

Классикой жанра можно считать образцы эксплуатационной документации, разработанной советской оборонкой для Военно-Воздушных Сил. Точность, согласованность и уровень детализации документов был таков, что гарному хлопцу из отдаленного горного аула, призванному на срочную службу и едва читающему по-русски, можно было доверить выполнение ряда довольно ответственных операций при проведении регламентных работ на авиационной технике.

Но, как это ни печально, до сих пор многие г-да «эффективные менеджеры» (в погоне за длинным баксом) считают расходы на подразделения, занятые разработкой (сопровождением и т.д.) техдокументации, бессмысленными издержками. Г-да руководители - карающий меч прокуратуры снесет, в первую очередь, ваши головы, поскольку именно вы ставите на документах утверждающую подпись. А разработчик ни за что не отвечает.

В компаниях, выпускающих продукцию для «дома и офиса», состав, содержание и содержимое техдокументации мало кому интересны. Приоритеты иные. Для таких компаний главное, чтобы техдокументация была оформлена ярко и затейливо, выделялась на общем фоне и привлекала к себе внимание любителей фантиков. А еще лучше - в виде комиксов, поскольку все идет к тому, что публика скоро окончательно разучится читать.

Примеры, по мнению автора, приводить бессмысленно. Многие неоднократно сталкивались и сталкиваются с совершенно бестолковой, размазанной, как манная каша по тарелке, писаниной. Писаниной, изданной в красочной суперобложке, на глянцевой бумаге. С глоссарием. С индексом. Сверстанной по «законам восприятия». И бесполезной в плане содержания. Usability в вульгарном, а не ГОСТовском смысле.

Тем не менее и компании-акулы, и мелкие любители национальной рыбалки в мутной водичке отечественного бизнеса едины в своем отношении к техдокументации, как минимум:

- вынужденным осознанием факта, что техническая документация должна быть;
- отсутствием интереса к процессу разработки (и т.д.) техдокументации.

С отношением и назначением покончено, теперь коротко о составе технической документации.

Техническая документация: состав

В серьезных компаниях, работающих на рынке автоматизированных систем, поставляемых солидным заказчикам, в состав техдокументации входят:

- техническая документация на автоматизированные системы;
- техническая документация на изделия;

- техдокументация на программные изделия - программная документация.

Следует отметить, что разработка технической документации на автоматизированные системы в современных условиях неумолимо влечет за собой разработку техдокументации на изделия и программные изделия, за исключением покупных.

С другой стороны, все, что производится в современых условиях, проявляет все больше и больше признаков автоматизированных систем. Мобильный телефон, к примеру, трудно назвать изделием в классическом понимании. Вот ведро или лом - это точно изделия, и ничего больше. Тот же FineReader™ можно назвать программным изделием только на дистрибутивном носителе. После инсталляции FineReader начинает автоматически распознавать тексты - становится (в совокупности с программным и техническим обеспечением (средствами ПЭВМ) полноценной автоматизированной системой или системой обработки информации.

Техническая документация: жизненный цикл

Как бы то ни было, кто бы ни занимался разработкой техдокументации, каков бы ни был ее состав, обладает техническая документация, как и все в этом мире, собственным жизненным циклом. До слез понятно, даже без ссылок на всевозможные стандарты, что жизненный цикл техдокументации включает в себя, как минимум:

- процесс разработки технической документации;
- процесс публикации техдокументации как на бумажных носителях, так и в электронном виде;
- процессы учета и хранения технической документации;
- процессы модификации, отслеживания изменений техдокументации - сопровождения;
- процесс обмена технической документацией между подразделениями компании;
- процесс передачи техдокументации заказчику (или конечному пользователю).

При разработке и публикации техдокументации применяются различные текстовые и графические редакторы различных производителей и версий. Пишут все. Сохраняют файлы все. Печатают все. Никакими санкциями, никакими корпоративными стандартами преодолеть склонность россиянина к анархии невозможно. Каждый пользуется тем, что привычней. Как следствие, форматы файлов, стили оформления технической документации трудно назвать единообразными и соответствующими требованиям стандартов. У каждого индивидуума свои предпочтения (в шрифтах, отступах и прочей разметке).

Еще одна общая беда - электронная техническая документация не структурируется должным образом. Для разбиения электронной техдокументации

на элементы данных - разделы, подразделы, пункты, подпункты - применяются:

- стили заголовков с многоуровневой нумерацией - особо продвинутыми пользователями (крайне редко);
- нумерованные списки – ворд достаточно «умен» и при попытках ручного ввода нумерации разделов иногда пытается автоматически разбить сплошной текст на разделы и подразделы в виде нумерованных списков;
- выделение строки абзацного текста жирным и ручной ввод номера раздела – лишь бы на бумаге все выглядело «изячно», если угодно - «элегантно».

Так ведь и претензии предъявить нельзя! У того же проджект-менеджера основная задача - не владение вордом на уровне продвинутого пользователя, а получение подписи заказчика на Акте приемки-сдачи работ. Мотивация, пардон, отсутствует.

О хранении - электронная техдокументация хранится в структуре каталогов пользовательских компьютеров и серверов. Естественно, абы как. Российский менталитет-с. Структура пользовательских каталогов хаотична. Где что лежит - в лучшем случае вспомнит лишь хозяин. Названия файлов электронной технической документации также не отличаются единообразием и удобочитаемостью. Резервное копирование файлов электронной техдокументации, как правило, не производится, что приводит к утрате документов.

Об учете - попытки поставить классический учет бумажных документов терпит фиаско. Уследить за всеми участниками процесса разработки нереально. Время заказ-нарядов на машинописные работы давным-давно прошло. Быстро не «только кошки плодятся», но и документация. И также «промискуитетно».

О сопровождении техдокументации - при больших объемах своевременное внесение изменений в ряд отдельных документов, отслеживание актуальных версий, архивирование старых версий технической документации невозможны. Утрачивается согласованность (непротиворечивость) техдокументации.

Об обмене технической документацией внутри компании. Затребовали - отправил документ электронной почтой или показал, в каком каталоге сетевого диска документ хранится. Позднее внес в документ изменения. Не затребовали - не передал. Сбросить на расшаренный сетевой диск или на svn поленился, решил отложить на завтра - и благополучно обо всем забыл.

А документ, тем временем, отправляется затребовавшим подразделением в типографию, для тиражирования в количестве двух тысяч экземпляров и незамедлительной отправки заказчику.

Техническая документация: предпосылки к автоматизации процессов жизненного цикла

Каковы же предпосылки к автоматизации процессов жизненного цикла технической документации? Вот они:

- доступность специализированных средств разработки текстовых документов, построенных на основе концепции единого источника (исходника - single source) с возможностью многократного повторного использования текстов, графики, мультимедиа, гиперссылок и т.д.;
- замечательные особенности советской нормативно-технической документации.

Специализированные средства разработки технической документации

Специализированных средств разработки технической (да и любой документации) навалом. Производятся таковые как отечественными компаниями, так и буржуйскими. Средства известные, на слуху, подробно останавливаться и приводить сравнительный анализ особого смысла нет. Предпочтения автора сводятся к применению при разработке (сопровождении и т.д.) техдокументации программы AuthorIT от AuthorIT Software Corporation Ltd.

РАЗДЕЛ 8. СРЕДСТВА РАЗРАБОТКИ ТЕХНИЧЕСКОЙ ДОКУМЕНТАЦИИ

Лекция 15. Создание многодокументного приложения

Обзор графических библиотек

Строить графический интерфейс пользователя (GUI, G raphical U ser I nterface) для программ на языке Python можно при помощи соответствующих библиотек компонентов графического интерфейса или, используя кальку с английского, библиотек виджетов.

Следующий список далеко не полон, но отражает многообразие существующих решений:

1. Tkinter Многоплатформенный пакет имеет хорошее управление расположением компонентов. Интерфейс выглядит одинаково на различных платформах (Unix, Windows, Macintosh). Входит в стандартную поставку Python. В качестве документации можно использовать руководство "An Introduction to Tkinter" ("Введение в Tkinter"), написанное Фредриком Лундом: http://www.pythonware.com/library/tkinter/introduction/

2. wxPython Построен на многоплатформной библиотеке wxWidgets (раньше называлась wxWindows). Выглядит родным для всех платформ, активно совершенствуется, осуществлена поддержка GL. Имеется для всех основных платформ. Возможно, займет место Tkinter в будущих версиях Python. Сайт: http://www.wxpython.org/

3. PyGTK Набор визуальных компонентов для GTK+ и Gnome. Только для платформы GTK.

4. PyQT/PyKDE Хорошие пакеты для тех, кто использует Qt (под UNIX или Windows) или KDE.

5. Pythonwin Построен вокруг MFC, поставляется вместе с оболочкой в пакете win32all; только для Windows.

6. pyFLTK Аналог Xforms, поддержка OpenGL. Имеется для платформ Windows и Unix. Сайт: http://pyfltk.sourceforge.net/

7. AWT, JFC, Swing поставляется вместе с Jython, а для Jython доступны средства, которые использует Java. Поддерживает платформу Java.

8. anygui Независимый от нижележащей платформы пакет для построения графического интерфейса для программ на Python. Сайт: http://anygui.sourceforge.net/

9. PythonCard Построитель графического интерфейса, сходный по идеологии с HyperCard/MetaCard. Разработан на базе wxPython. Сайт: http://pythoncard.sourceforge.net/

Список актуальных ссылок на различные графические библиотеки, доступные из Python, можно найти по следующему адресу: http://phaseit.net/claird/comp.lang.python/python_GUI.html

Библиотеки могут быть многоуровневыми. Например, PythonCard использует wxPython, который, скажем, на платформе Linux базируется на

многоплатформной GUI-библиотеке wxWindows, которая, в свою очередь, базируется на GTK+ или на Motif, а те - тоже используют для вывода X Window. Кстати, для Motif в Python имеются свои привязки.

О графическом интерфейсе

Почти все современные графические интерфейсы общего назначения строятся по модели WIMP - Window, Icon, Menu, Pointer (окно, иконка, меню, указатель). Внутри окон рисуются элементы графического интерфейса, которые для краткости будут называться виджетами (widget - штучка). Меню могут располагаться в различных частях окна, но их поведение достаточно однотипно: они служат для выбора действия из набора предопределенных действий. Пользователь графического интерфейса "объясняет" компьютерной программе требуемые действия с помощью указателя. Обычно указателем служит курсор мыши или джойстика, однако есть и другие "указательные" устройства. С помощью иконок графический интерфейс приобретает независимость от языка и в некоторых случаях позволяет быстрее ориентироваться в интерфейсе.

Основной задачей графического интерфейса является упрощение коммуникации между пользователем и компьютером. Об этом следует постоянно помнить при проектировании интерфейса. Применение имеющихся в наличии у программиста (или дизайнера) средств при создании графического интерфейса нужно свести до минимума, выбирая наиболее удобные пользователю виджеты в каждом конкретном случае. Кроме того, полезно следовать принципу наименьшего удивления: из формы интерфейса должно быть понятно его поведение. Плохо продуманный интерфейс портит ощущения пользователя от программы, даже если за фасадом интерфейса скрывается эффективный алгоритм. Интерфейс должен быть удобен для типичных действий пользователя. Для многих приложений такие действия выделены в отдельные серии экранов, называемые "мастерами" (wizards). Однако если приложение - скорее конструктор, из которого пользователь может строить нужные ему решения, типичным действием является именно построение решения. Определить типичные действия не всегда легко, поэтому компромиссом может быть гибрид, в котором есть "мастера" и хорошие возможности для собственных построений. Тем не менее, графический интерфейс не является самым эффективным интерфейсом во всех случаях. Для многих предметных областей решение проще выразить с помощью деклараций на некотором формальном языке или алгоритма на сценарном языке.

Основы Tk

Основная черта любой программы с графическим интерфейсом - интерактивность. Программа не просто что-то считает (в пакетном режиме) от начала своего запуска до конца: ее действия зависят от вмешательства пользователя. Фактически, графическое приложение выполняет бесконечный цикл обработки событий. Программа, реализующая графический интерфейс,

событийно-ориентирована. Она ждет от интерфейса событий, которые и обрабатывает сообразно своему внутреннему состоянию.

Эти события возникают в элементах графического интерфейса (виджетах) и обрабатываются прикрепленными к этим виджетам обработчиками. Сами виджеты имеют многочисленные свойства (цвет, размер, расположение), выстраиваются в иерархию принадлежности (один виджет может быть хозяином другого), имеют методы для доступа к своему состоянию.

Расположением виджетов (внутри других виджетов) ведают так называемые менеджеры расположения. Виджет устанавливается на место по правилам менеджера расположения. Эти правила могут определять не только координаты виджета, но и его размеры. В Tk имеются три типа менеджеров расположения: простой упаковщик (pack), сетка (grid) и произвольное расположение (place).

Но этого для работы графической программы недостаточно. Дело в том, что некоторые виджеты в графической программе должны быть взаимосвязаны определенным образом. Например, полоска прокрутки может быть взаимосвязана с текстовым виджетом: при использовании полоски текст в виджете должен двигаться, и наоборот, при перемещении по тексту полоска должна показывать текущее положение. Для связи между виджетами в Tk используются переменные, через которые виджеты и передают друг другу параметры.

Классы виджетов

Для построения графического интерфейса в библиотеке Tk отобраны следующие классы виджетов (в алфавитном порядке):

1. Button (Кнопка) Простая кнопка для вызова некоторых действий (выполнения определенной команды).

2. Canvas (Рисунок) Основа для вывода графических примитивов.

3. Checkbutton (Флажок) Кнопка, которая умеет переключаться между двумя состояниями при нажатии на нее.

4. Entry (Поле ввода) Горизонтальное поле, в которое можно ввести строку текста.

5. Frame (Рамка) Виджет, который содержит в себе другие визуальные компоненты.

6. Label (Надпись) Виджет может показывать текст или графическое изображение.

7. Listbox (Список) Прямоугольная рамка со списком, из которого пользователь может выделить один или несколько элементов.

8. Menu (Меню) Элемент, с помощью которого можно создавать всплывающие (popup) и ниспадающие (pulldown) меню.

9. Menubutton (Кнопка-меню) Кнопка с ниспадающим меню.

10. Message (Сообщение) Аналогично надписи, но позволяет заворачивать длинные строки и менять размер по требованию менеджера расположения.

11. Radiobutton (Селекторная кнопка) Кнопка для представления одного из альтернативных значений. Такие кнопки, как правило, действует в группе. При нажатии на одну из них кнопка группы, выбранная ранее, "отскакивает".

12. Scale (Шкала) Служит для задания числового значения путем перемещения движка в определенном диапазоне.

13. Scrollbar (Полоса прокрутки) Полоса прокрутки служит для отображения величины прокрутки в других виджетах. Может быть как вертикальной, так и горизонтальной.

14. Text (Форматированный текст) Этот прямоугольный виджет позволяет редактировать и форматировать текст с использованием различных стилей, внедрять в текст рисунки и даже окна.

15. Toplevel (Окно верхнего уровня) Показывается как отдельное окно и содержит внутри другие виджеты.

Все эти классы не имеют отношений наследования друг с другом - они равноправны. Этот набор достаточен для построения интерфейса в большинстве случаев.

События

В системе современного графического интерфейса имеется возможность отслеживать различные события, связанные с клавиатурой и мышью, и происходящие на "территории" того или иного виджета. В Tk события описываются в виде текстовой строки - шаблона события, состоящего из трех элементов (модификаторы, тип события и детализация события).

Тип события	**Содержание события**
Activate	Активизация окна
ButtonPress	Нажатие кнопки мыши
ButtonRelease	Отжатие кнопки мыши
Deactivate	Деактивация окна
Destroy	Закрытие окна
Enter	Вхождение курсора в пределы виджета
FocusIn	Получение фокуса окном
FocusOut	Потеря фокуса окном
KeyPress	Нажатие клавиши на клавиатуре
KeyRelease	Отжатие клавиши на клавиатуре
Leave	Выход курсора за пределы виджета
Motion	Движение мыши в пределах виджета
MouseWheel	Прокрутка колесика мыши
Reparent	Изменение родителя окна
Visibility	Изменение видимости окна

Примеры описаний событий строками и некоторые названия клавиш приведены ниже:

"<ButtonPress-3>" или просто "<3>" - щелчок правой кнопки мыши (то есть, третьей, если считать на трехкнопочной мыши слева-направо). "<Shift-Double-Button-1>" - двойной щелчок мышью (левой кнопкой) с нажатой кнопкой Shift. В качестве модификаторов могут быть использованы следующие (список неполный):

Control, Shift, Lock,
Button1-Button5 или B1-B5,
Meta, Alt, Double, Triple.

Просто символ обозначает событие - нажатие клавиши. Например, "k" - тоже, что "<KeyPress-k>". Для неалфавитно-цифровых клавиш есть специальные названия:

Cancel, BackSpace, Tab, Return, Shift_L, Control_L, Alt_L,
Pause, Caps_Lock, Escape, Prior, Next, End, Home, Left,
Up, Right, Down, Print, Insert, Delete, F1, F2, F3, F4, F5, F6, F7,
F8, F9, F10, F11, F12, Num_Lock, Scroll_Lock, space, less

Здесь <space> обозначает пробел, а <less> - знак меньше. <Left>, <Right>, <Up>, <Down> - стрелки. <Prior>, <Next> — это PageUp и PageDown. Остальные клавиши более или менее соответствуют надписям на стандартной клавиатуре.

Создание и конфигурирование виджета

Создание виджета происходит вызовом конструктора соответствующего класса. Вызов конструктора имеет следующий синтаксис:

Widget([master[, option=value, ...]])

Здесь Widget - класс виджета, master - виджет-хозяин, option и value - конфигурационная опция и ее значение (таких пар может быть несколько).

Каждый виджет имеет свойства, которые можно устанавливать (конфигурировать) с помощью методов config() (или configure()) и читать с помощью методов, подобных методам работы со словарями. Ниже приведен возможный синтаксис для работы со свойствами:

widget.config(option=value, ...)
widget["option"] = value
value = widget["option"]
widget.keys()

В случае, когда имя свойства совпадает с ключевым словом языка Python, принято использовать после имени одиночное подчеркивание. Так, свойство class нужно задавать как class_, а to как to_.

Изменять конфигурацию виджета можно в любой момент. Это изменение прорисуется на экране по возвращении в цикл обработки событий или при явном вызове update_idletasks().

Следующий пример показывает окно с двумя виджетами внутри - полем ввода и надписью. С помощью переменной надпись напрямую связана с полем ввода. Этот пример нарочно использует очень много свойств, чтобы продемонстрировать возможности по конфигурированию:

```
from Tkinter import *
tk = Tk()
tv = StringVar()
Label(tk,
textvariable=tv,
relief="groove",
borderwidth=3,
font=("Courier", 20, "bold"),
justify=LEFT,
width=50,
padx=10,
pady=20,
takefocus=False,
).pack()
Entry(tk,
textvariable=tv,
takefocus=True,
).pack()
tv.set("123")
tk.mainloop()
```

В результате на экране можно увидеть:

Виджеты конфигурируются прямо при создании. Более того, виджеты не связываются с именами, их только располагают внутри виджета-окна. В данном примере использованы свойства textvariable (текстовая переменная), relief (рельеф), borderwidth (ширина границы), justify (выравнивание), width (ширина, в знакоместах), padx и pady (прослойка в пикселях между

содержимым и границами виджета), takefocus (возможность принять фокус при нажатии клавиши Tab), font (шрифт, один из способов его задания). Эти свойства достаточно типичны для многих виджетов, хотя иногда единицы измерения могут отличаться, например, для виджета Canvas ширина задается в пикселях, а не в знакоместах.

В следующем примере демонстрируются возможности по назначению цветов фону, переднему плану (тексту), выделению виджета (подсветка границы) в активном состоянии и при отсутствии фокуса:

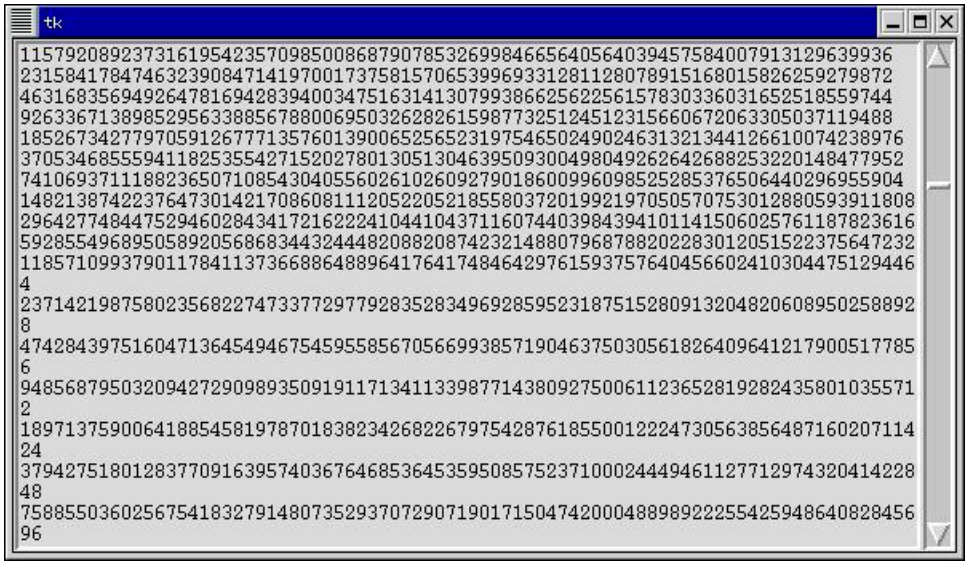
```
from Tkinter import *
tk = Tk()
tv = StringVar()
Entry(tk,
textvariable=tv,
takefocus=True,
borderwidth=10,
).pack()
mycolor1 = "#%02X%02X%02X" % (200, 200, 20)
Entry(tk,
textvariable=tv,
takefocus=True,
borderwidth=10,
foreground=mycolor1, # fg, текст виджета
background="#0000FF", # bg, фон виджета
highlightcolor='green', # подсветка при фокусе
highlightbackground='red', # подсветка без фокуса
).pack()
tv.set("123")
tk.mainloop()
```

При желании можно задать стилевые опции для всех виджетов сразу: с помощью метода tk_setPalette(). Помимо использованных выше свойств в этом методе можно использовать selectForeground и selectBackground (передний план и фон выделения), selectColor (цвет в выбранном состоянии, например, у Checkbutton), insertBackground (цвет точки вставки) и некоторые другие. Виджет форматированного текста

Для того чтобы показать работу с нетривиальным виджетом, можно взять виджет ScrolledText из одноименного модуля Python. Этот виджет

аналогичен рамке с форматированным текстом и вертикальной полосой прокрутки:

```
from Tkinter import *
from ScrolledText import ScrolledText
tk = Tk() # окно верхнего уровня
txt = ScrolledText(tk) # виджет текста с прокруткой
txt.pack() # виджет размещается
for x in range(1, 1024): # виджет наполняется текстовым содержимым
txt.insert(END, str(2L**x)+"\n")
tk.mainloop()
```



Теперь следует рассмотреть методы и свойства виджета с форматированным текстом более подробно.

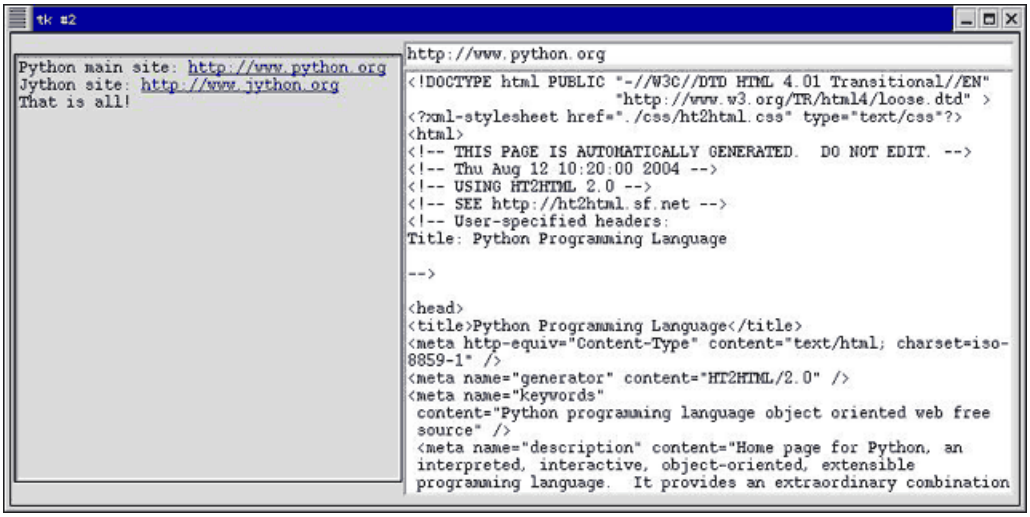
Для навигации в тексте в Tk предусмотрены специальные индексы. Индексы вроде 1.0 и END уже встречались — это начало текста (первая строка, нулевой символ) и его конец. (В Tk строки нумеруются с единицы, а символы строки - с нуля). Более полный список индексов:

1. L.C Здесь L - номер строки, а C - номер символа в строке.
2. INSERT Точка вставки.
3. CURRENT Символ, ближайший к курсору мыши.
4. END Позиция сразу за последним символом в тексте
5. M.first, M.last Индексы начала и конца помеченного тегом M участка текста.
6. SEL_FIRST, SEL_LAST Индексы начала и конца выделенного текста.
7. M Пользователь может определять свои именованные позиции в тексте (аналогично END, INSERT или CURRENT). При редактировании текста маркеры будут сдвигаться с заданными для них правилами.
8. @x,y Символ текста, ближайший к точке с координатами x, y.

Следующий пример показывает, как снабдить форматированный текст гипертекстовыми возможностями:

```
rom Tkinter import *
import urllib
tk = Tk()
txt = Text(tk, width=64) # поле с текстом
txt.grid(row=0, column=0, rowspan=2)
addr=Text(tk, background="White", width=64, height=1) # поле адреса
addr.grid(row=0, column=1)
page=Text(tk, background="White", width=64) # поле с html-кодом
page.grid(row=1, column=1)
def fetch_url(event):
click_point = "@%s,%s" % (event.x, event.y)
trs = txt.tag_ranges("href") # список областей текста, отмеченных как
href
url = ""
# определяется, на какой участок пришелся щелчок мыши, и берется
# соответствующий ему URL
for i in range(0, len(trs), 2):
if txt.compare(trs[i], "<=", click_point) and \
txt.compare(click_point, "<=", trs[i+1]):
url = txt.get(trs[i], trs[i+1])
html_doc = urllib.urlopen(url).read()
addr.delete("1.0", END)
addr.insert("1.0", url) # URL помещается в поле адреса
page.delete("1.0", END)
page.insert("1.0", html_doc) # показывается HTML-документ
textfrags = ["Python main site: ", "http://www.python.org",
"\nJython site: ", "http://www.jython.org",
"\nThat is all!"]
for frag in textfrags:
if frag.startswith("http:"):
txt.insert(END, frag, "href") # URL помещается в текст с меткой href
else:
txt.insert(END, frag) # фрагмент помещается в текст
# ссылки отмечаются подчеркиванием и синим цветом
txt.tag_config("href", foreground="Blue", underline=1)
# при щелчке мыши на тексте, отмеченном как "href",
# следует вызывать fetch_url()
txt.tag_bind("href", "<1>", fetch_url)
tk.mainloop() # запускается цикл событий
```

В результате (после нажатия на гиперссылку) можно увидеть примерно следующее:



Для придания некоторым участкам текста особых свойств необходимо их отметить тегом. В данном случае URL отмечается тегом href. Позднее с помощью метода tag_config() задаются свойства отображения текста, отмеченного таким тегом. Методом tag_bind() привязывается некоторое событие (щелчок мыши) с вызовом заданной функции (fetch_url()).

В самой функции fetch_url() нужно в начале определить, на какой именно участок текста пришелся щелчок мыши. Для этого с помощью метода tag_ranges() получаются все интервалы, которые отмечены как href. Для определения конкретного URL проводятся сравнения (методом compare()) точки щелчка мышью с каждым из интервалов. Так находится интервал, на который попал щелчок, и с помощью метода get() получается текстовое значение найденного интервала. Найдя URL, его в поле записываются адреса, и получается HTML-код, соответствующий URL.

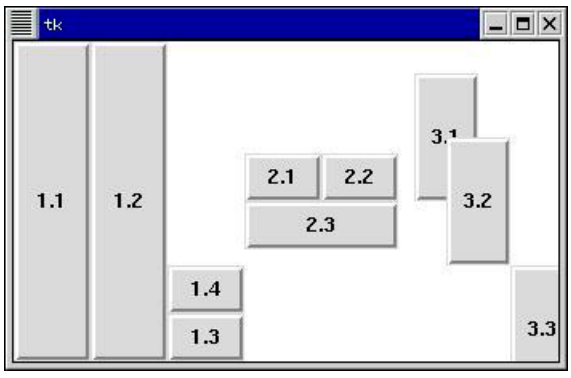
Менеджеры расположения

Следующий пример достаточно нагляден, чтобы понять принципы работы менеджеров расположения, имеющихся в Tk. В трех рамках можно применить различные менеджеры: pack, grid и place:

```
from Tkinter import *
tk = Tk()
# Создаем три рамки
frames = {}
b = {}
for fn in 1, 2, 3:
f = Frame(tk, width=100, height=200, bg="White")
f.pack(side=LEFT, fill=BOTH)
frames[fn] = f
for bn in 1, 2, 3, 4: # Создаются кнопки для каждой из рамок
```

```
b[fn, bn] = Button(frames[fn], text="%s.%s" % (fn, bn))
# Первая рамка:
# Сначала две кнопки прикрепляются к левому краю
b[1, 1].pack(side=LEFT, fill=BOTH, expand=1)
b[1, 2].pack(side=LEFT, fill=BOTH, expand=1)
# Еще две - к нижнему
b[1, 3].pack(side=BOTTOM, fill=Y)
b[1, 4].pack(side=BOTTOM, fill=BOTH)
# Вторая рамка:
# Две кнопки сверху
b[2, 1].grid(row=0, column=0, sticky=NW+SE)
b[2, 2].grid(row=0, column=1, sticky=NW+SE)
# и одна на две колонки в низу
b[2, 3].grid(row=1, column=0, columnspan=2, sticky=NW+SE)
# Третья рамка:
# Кнопки высотой и шириной в 40% рамки, якорь в левом верхнем углу.
# Координаты якоря 1/10 от ширины и высоты рамки
b[3, 1].place(relx=0.1, rely=0.1, relwidth=0.4, relheight=0.4, anchor=NW)
# Кнопка строго по центру. Якорь в центре кнопки
b[3, 2].place(relx=0.5, rely=0.5, relwidth=0.4, relheight=0.4, anchor=CEN-
TER)
# Якорь по центру кнопки. Координаты якоря 9/10 от ширины и высоты
рамки
b[3, 3].place(relx=0.9, rely=0.9, relwidth=0.4, relheight=0.4, anchor=CEN-
TER)
tk.mainloop()
```

Результат, следующий:



Менеджер pack просто заполняет внутреннее пространство на основании предпочтения того или иного края, необходимости заполнить все измерение. В некоторых случаях ему приходится менять размеры подчиненных виджетов. Этот менеджер стоит использовать только для достаточно простых схем расположения виджетов.

Менеджер grid помещает виджеты в клетки сетки (это очень похоже на способ верстки таблиц в HTML). Каждому располагаемому виджету даются координаты в одной из ячеек сетки (row - строка, column - столбец), а также, если нужно, столько последующих ячеек (в строках ниже или в столбцах правее) сколько он может занять (свойства rowspan или columnspan). Это самый гибкий из всех менеджеров.

Менеджер place позволяет располагать виджеты по произвольным координатам и с произвольными размерами подчиненных виджетов. Размеры и координаты могут быть заданы в долях от размера виджета-хозяина.

Непосредственно внутри одного виджета нельзя использовать более одного менеджера расположения: менеджеры могут наложить противоречащие ограничения на вложенные виджеты и внутренние виджеты просто не смогут быть расположены.

2 ЛАБОРАТОРНО-ПРАКТИЧЕСКИЕ ЗАНЯТИЯ
РАЗДЕЛ 1. ОСНОВНЫЕ ПРИНЦИПЫ ПРОГРАММИРОВАНИЯ

Практическая работа 1. Выделение структурных единиц и разработка типовых алгоритмом

Цель работы: научиться вводить и выводить данные, создавать переменные и выполнять арифметические операторы, использовать операторы сравнения.

Теоретический материал

Теорию для выполнения лабораторной работы смотрите на странице tk.ulstu.ru/video.php#video в следующих видеоматериалах:

- #1. Первое знакомство с Python Установка на компьютер
- #2. Варианты исполнения команд. Переходим в PyCharm
- #3. Переменные, оператор присваивания, функции type и id
- #4. Числовые типы, арифметические операции
- #5. Математические функции и работа с модулем math
- #6. Функции print() и input(). Преобразование строк в числа int() и float()
- #7. Логический тип bool. Операторы сравнения и операторы and, or, not

Задания на лабораторную работу

1. Установить интерпретатор языка Python на свой рабочий компьютер.
2. Установить интегрированную среду PyCharm на свой рабочий компьютер.
3. Написать программы в среде PyCharm в соответствии с номером своего варианта.

Номер	Задание №1	Задание №2
1	Напишите программу, которая запрашивает у пользователя два вещественных значения a, b и вычисляет площадь прямоугольника. Вывести на экран вычисленное значение.	Напишите программу, которая запрашивает у пользователя три целых числовых значения a, b, c. Используя операторы сравнения, определить, можно ли из чисел a, b, c составить треугольник. (Правило: каждая сторона треугольника должна быть меньше суммы двух других). Вывести на экран

2	Напишите программу, которая запрашивает у пользователя два целых числовых значения a, b – катеты прямоугольного треугольника, и вычисляет значение гипотенузы. Вывести на экран вычисленное значение.	Напишите программу, которая запрашивает у пользователя вещественное значение a и проверяет вхождение этого числа в диапазон [-5; 10]. Вывести на экран полученное булево значение (True или False).
3	Напишите программу, которая запрашивает у пользователя два вещественных значения a, b – высота и основание треугольника, и вычисляет его площадь. Вывести на экран вычисленное значение.	Напишите программу, которая запрашивает у пользователя вещественное значение x и делает проверку на не вхождение этого числа в диапазон [-3; 4]. Вывести на экран полученное булево значение (True или False).
4	Напишите программу, которая запрашивает у пользователя два целых значения a, b и возводит первое число в степень второго. Вывести на экран вычисленное значение.	Напишите программу, которая запрашивает у пользователя целое числовое значение x и делает проверку на кратность x числу 2 или 3. Вывести на экран полученное булево значение (True или False).
5	Вводится вещественное значение y. Необходимо вычислить значение x уравнения: $2x + 5 = y$. Вывести на экран вычисленное значение.	Вводится два вещественных числа a и b. Необходимо определить, является ли квадрат числа a меньше квадрата числа b. Вывести на экран полученное бу-
6	Вводится радиус круга. Необходимо вычислить его длину, используя константу pi модуля math. Вывести на экран вычисленное значение.	Вводятся два вещественных числа: R – радиус круга; a – длина стороны квадрата. Определить у какой фигуры больше площадь. Вывести на экран полученное булево значение
7	Вводится вещественное значение x. Выполнить вычисление функции: $y = 7x^2 - 2x + 5$ Результат (значение y) вывести на экран.	Вводятся три натуральных числа m, n, q. Определить, кратно ли число m числу n и числу q одновременно. Вывести на экран полученное булево

8	Вводится вещественное значение x. Выполнить вычисление функции: $y = 3 \cdot \lvert x \rvert - x$ Результат (значение y) вывести на экран.	Вводится натуральное число x. Определить, кратно ли оно четырем и оканчивается ли на цифру 8 (одновременно). Вывести на экран полученное булево значение (True или False).
9	Вводятся два целых значения a, b – катеты прямоугольного треугольника. Необходимо найти его периметр. Результат вывести на экран.	Вводятся два вещественных числа x, y. Определить, что их модуль разности меньше пяти или модуль суммы меньше десяти. Вывести на экран полученное булево значение
10	Вводятся два вещественных значения a, b. Найти их среднее арифметическое $(\lvert a \rvert + \lvert b \rvert) : 2$ и среднее геометрическое $\lvert a \rvert \cdot \lvert b \rvert$ их модулей. Результаты вычислений вывести на экран.	Вводится натуральное число x. Определить, что оно не кратно трем и не кратно пяти (одновременно). Вывести на экран полученное булево значение (True или False).

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Тексты программ.
3. Результаты работы программ.

Практическая работа 2. Определение компонентов программного обеспечения

Цель работы: научиться работать со строками и списками на языке Python.

Теоретический материал

Теорию для выполнения лабораторной работы смотрите на странице tk.ulstu.ru/video.php#video в следующих видеоматериалах:

- #8. Введение в строки. Базовые операции над строками
- #9. Знакомство с индексами и срезами строк
- #10. Основные методы строк
- #11. Спецсимволы, экранирование символов, row-строки
- #12. Форматирование строк: метод format и F-строки
- #13. Списки - операторы и функции работы с ними
- #14. Срезы списков и сравнение списков
- #15. Основные методы списков
- #16. Вложенные списки, многомерные списки

Задания на лабораторную работу

1. Написать программы в среде PyCharm в соответствии с номером своего варианта.

Номер	Задание №1	Задание №2
1	Вводится строка. Найти все индексы начала фрагмента «ра» во введенной строке. Результат вывести на экран.	Вводится список целых чисел в одну строчку через пробел. Необходимо представить его в виде списка целых чисел, а затем, вычислить сумму этих
2	Вводится строка с одним арифметическим действием (сложением) для двух целых чисел. Например, «5+3» или «7 + 2». Обратите внимание на возможные пробелы до и после оператора +. Вычислить, указанное в строке арифметическое действие и результат вывести на экран.	Вводится список вещественных чисел в одну строчку через пробел. Необходимо представить его в виде списка вещественных чисел, а затем, найти минимальное среди этих чисел и вывести его на экран.

3	Вводится слово. Необходимо определить, является ли это слово палиндромом (читается одинаково и вперед и назад, например, слово «Анна» – палиндром). Вывести True, если слово палиндром и False – в противном случае. Программу реализовывать без использования условного оператора if.	Вводится список оценок студента в одну строчку через пробел. Необходимо преобразовать эту строку в список из чисел и подсчитать количество двоек в нем. Результат (число двоек) вывести на экран. Программу реализовать без использования операторов циклов for или while.
4	Вводится пароль. Убедиться, что пароль введен корректно по следующим критериям: - число символов не менее 8; - имеются цифровые и буквенные символы; - присутствует хотя бы один из символов «$#!?-_». Вывести True, если пароль корректен и False – в противном случае. Программу реализовывать без использования условного оператора if.	Вводятся слова в одну строчку через пробел. Преобразовать их в список из слов. Проверить, повторяется ли в этом списке последнее введенное слово. Вывести True, если это так и False – в противном случае. Программу реализовывать без использования условного оператора if.
5	Вводится фрагмент URL-адреса латинскими символами с дефисами, например: python-info--data--study Необходимо все два подряд идущих дефиса (--) заменить на один дефис (-). Дополнительно подсчитать число таких замен. Результат (преобразованную строку и число замен) вывести на экран.	В одну строку вводится информация по студенту в формате (без кавычек): «Имя возраст группа список оценок» Список оценок – это набор целых чисел от 2 до 5. Их может быть разное количество. Преобразовать эту информацию в список в формате: ['Имя', возраст, 'группа', [3, 5, 4, 2, ...]] То есть, сформировать вложенный список из оценок. Результат (список) вывести на экран.

6	Вводится строка из нескольких слов, слова разделены пробелом. Убедиться, что первое и последнее слово во введенной строке не совпадают. Вывести True, если это так и False – в противном случае. Программу реализовать без использования условного оператора if.	Вводится список из четырех предметов в формате: название_1=вес_1 название_2=вес_2 название_3=вес_3 название_4=вес_4 Необходимо преобразовать введенные данные в двумерный список вида: [['название_1', вес_1], ['название_2', вес_2], ...] Вес – это целое число. Результат (список) вывести на экран.
7	Вводится вещественное значение x – аргумент функции: $y(x) = x^2 - 2$. С помощью F-строки вывести на экран сообщение (без кавычек): «Значение функции y(x) = число_1, при значении x = число_2» Величины (число_1 и число_2) выводить с точностью до сотых.	Вводится список по четырем смартфонам в формате: название_1:цена_1 … название_4:цена_4 Здесь цена – это вещественное число. Преобразовать эту информацию в список вида: [['название_1', цена_1], ['название_2', цена_2], ...] Результат (список) вывести на экран.
8	Вводятся два целых числа a, b – катеты прямоугольного треугольника. С помощью F-строки вывести на экран сообщение (без кавычек): «Периметр прямоугольного треугольника со сторонами a, b, c, равен: <число>». Все вещественные значения выводить с точностью до десятых.	Вводится список городов в одну строчку через пробел. Необходимо преобразовать его в список и вывести названия через один. Например: Ввод: Москва Омск Уфа Тверь Вывод: Москва Уфа Также вывести общее число введенных городов.

9	С каждой новой строки вводится следующая информация: - ФИО студента - возраст (целое число) - вес (вещественное число) - название группы необходимо объединить все эти данные в единую строку с помощью оператора +, записанные через пробел. Результат вывести на экран.	Вводятся названия стран через пробел в одну строчку. Сформировать на основе этой строки список из стран. Поменять местами значения первого и последнего элемента этого списка. Вывести на экран полученный список, а также значение True, если в списке есть страна «Россия», иначе – значение False. Программу реализовать без
10	Вводится строка. Определить в ней все индексы вхождения	Вводится строка из вещественных чисел, записанных через пробел.

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Тексты программ.
3. Результаты работы программ.

Практическая работа 3. Разработка алгоритмов и спецификаций структурных единиц. Реализация алгоритмов средствами автоматизированного проектирования

Цель работы: научиться работать с условными операторами на языке Python.

Теоретический материал

Теорию для выполнения лабораторной работы смотрите на странице tk.ulstu.ru/video.php#video в следующих видеоматериалах:

- #17. Условный оператор if. Конструкция if-else
- #18. Вложенные условия и множественный выбор. Конструкция if-elif-else
- #19. Тернарный условный оператор. Вложенное тернарное условие

Задания на лабораторную работу

1. Написать программы в среде PyCharm в соответствии с номером своего варианта.

Номер	Задание №1	Задание №2
1	Вводятся три целых числа a, b, c. С помощью условного оператора if найти минимальное значение и вывести его на экран.	Вводится аргумент x функции: $y(x) = \begin{cases} \lvert x\rvert + 2x, & x < 0 \\ x^2 - x + 2, & x \geq 0 \end{cases}$ Вычислить значение функции для введенного аргумента x и вывести результат на экран.
2	Вводятся коэффициенты a, b, c квадратного уравнения вида: $ax^2 + bx + c = 0$. Необходимо определить, имеет ли уравнение хотя бы один корень. Вывести «ДА», если решения есть и «НЕТ» - в противном случае.	Вводятся два целых значения m, n. Необходимо вывести их частное (m/n), если m нацело делится на n (и n не равно нулю). Иначе вывести их произведение. Реализовать программу с использованием тернарного условного оператора.
3	Вводится натуральное четырехзначное число. Если оно кратно трем, то вычислить сумму цифр этого числа, иначе – произведение	Имеется меню: 1. Кафедра ТК 2. Факультет ФИСТ 3. Кафедра Радиотехника 4. Выход из программы

	цифр. Результат вывести на экран.	Представить это меню в виде многострочной строки и вывести ее на экран. Запросить у пользователя ввести пункт меню и с помощью операторов if-elif-else реализовать выбор введенного пункта меню. На экране отобразить: «Выбран … пункт меню». Если указано недопустимое значение, то «Такой пункт не найден».
4	Вводится натуральное число. Определить, является ли оно трехзначным. Если это так, то вывести «Трехзначное число», иначе – «Не трехзначное число». Программу реализовать без использования операторов циклов.	Вводятся два натуральных числа: a – время в секундах; b – время в минутах. С помощью тернарного условного оператора определить наибольшее время из двух введенных и отобразить результат на экране.
5	Вводятся вещественные положительные числа a, b, c, d. Выяснить, можно ли прямоугольник со сторонами a, b уместить внутри прямоугольника со сторонами c, d так, чтобы каждая из сторон одного прямоугольника была параллельна или перпендикулярна каждой стороне второго прямоугольника. Вывести на экран «Входит», если входит, а иначе - «Не входит».	Вводятся оценки студента в виде целых чисел в одну строчку через пробел. Необходимо преобразовать эту строку в список целых чисел и если в нем окажется более одной двойки, то вывести сообщение «Отчислен», а иначе – «Учится». Программу реализовать без использования операторов циклов.

6	Вася пытается высунуть голову в форточку размерами a и b см (вводятся с клавиатуры). Приняв условно, что его голова – круглая диаметром d см (вводится с клавиатуры), определить, сможет ли Вася сделать это. Для прохождения головы в форточку необходим зазор в 1 см с каждой стороны. Вывести «ДА», если голова пролезает и «НЕТ» - в противном случае.	Вводятся названия городов в одну строчку через пробел. Преобразовать строку в список из названий и если в этом списке отсутствует город Москва, то добавить это значение в конец списка. Программу реализовать без использования операторов циклов.
7	Вводятся два вещественных числа a, b. Если хотя бы одно из этих чисел отрицательное, то вычислить их среднее арифметическое $(a+b):2$, иначе – среднее геометрическое: $a \cdot b$. Результат вывести на экран.	Вводятся оценки студента в одну строку через пробел. Преобразовать эту строку в список целых чисел и удалить из него максимум три четверки (или меньшее количество, если четверок меньше трех). Результирующий список вывести на экран.
8	Вводятся два натуральных числа a, b. Если они оба кратны 3, то вычислить значение $(a+b):3$, иначе вычислить значение $(a+b)\cdot 3$. Результат вывести на экран. Программу реализовать с использованием тернарного условного оператора.	Вводятся два списка городов, каждый список с новой строки. В списке по три города, записанных через пробел. Если все города в этих списках разные, то объединить их в один список. Иначе удалить из первого списка города, присутствующие во втором списке. Результат вывести на экран. Программу реализовать без использования операторов циклов.

9	Вводится email-адрес. Необходимо проверить корректность его написания по следующим критериям: - используются только символы латинского алфавита, цифры и символ подчеркивания ‘_’; - символ ‘@’ стоит до символа ‘.’. Вывести «ДА», если email записан корректно и «НЕТ» - в противном случае.	Вводятся целые числа в одну строчку. Они могут быть введены через пробел или через запятую. Необходимо преобразовать их в список целых чисел. Затем, вычислить среднее арифметическое значений полученного списка и вывести результат на экран. Программу реализовать без использования операторов циклов.
10	Вводится строка с одним арифметическим действием (сложением или вычитанием) для двух целых чисел. Например, «5+3» или «7 - 2». Обратите внимание на возможные пробелы до и после операторов + и -. Вычислить, указанное в строке арифметическое действие и результат вывести на экран.	Вводится аргумент x функции: $$y(x)=\begin{cases}1, & x\geq 4\\ x^3-x^2, & 2\leq x<4\\ x^2+x, & 1\leq x<2\\ -1, & x<1\end{cases}$$ Вычислить значение функции для введенного аргумента x и вывести результат на экран.

РАЗДЕЛ 2. ЭЛЕМЕНТЫ И ПРИЕМЫ ПРОГРАММИРОВАНИЯ НА АППАРАТНОМ УРОВНЕ

Практическая работа 4. Управление памятью. Способы выделения памяти в программах. Программно-доступные ресурсы процессора

Цель работы: научиться использовать операторы циклов и итерируемые объекты при программировании различных практических задач.

Теоретический материал

Теорию для выполнения лабораторной работы смотрите на странице tk.ulstu.ru/video.php#video в следующих видеоматериалах:

- #20. Оператор цикла while
- #21. Операторы циклов break, continue и else
- #22. Оператор цикла for. Функция range()
- #23. Примеры работы оператора цикла for. Функция enumerate()
- #24. Итератор и итерируемые объекты. Функции iter() и next()
- #25. Вложенные циклы. Примеры задач с вложенными циклами
- #26. Треугольник Паскаля как пример работы вложенных циклов

Задания на лабораторную работу

1. Написать программы в среде PyCharm в соответствии с номером своего варианта.

Номер	Задание №1	Задание №2
1	Вводится натуральное число. Найти все делители этого числа и вывести на экран. Программу реализовать с использованием цикла while.	Вводится фрагмент URL-адреса в виде строки (пример): python----cikly-for--and---while Необходимо все подряд идущие дефисы заменить одним дефисом (-). Результат вывести на экран.
2	Вводится натуральное число. Необходимо вычислить сумму цифр этого числа. Результат вывести на экран. Программу реализовать с использованием цикла while.	Вводится список городов в одну строчку через пробел. Преобразовать эту строку в список слов и, затем, через итератор выбрать первые два значения этого списка и вывести их на экран.

3	Вводится список названий городов в одну строчку через пробел. Определить, что в этом списке все города имеют длину более 5 символов. Реализовать программу с использованием операторов while и break. Вывести ДА, если условие выполняется, и НЕТ – в противном случае.	Задается двумерный список чисел размерностью 5 х 5 элементов, состоящий из нулей и, в некоторых позициях, единиц (формируется самостоятельно). Требуется проверить, не касаются ли единицы друг друга по горизонтали, вертикали и диагонали. То есть, вокруг каждой единицы должны быть нули. Если проверка проходит вывести ДА, иначе – НЕТ.
4	Вводится список имен студентов в одну строчку через пробел. Определить, что хотя бы одно имя в этом списке начинается и заканчивается на ту же самую букву (без учета регистра). Реализовать программу с использованием операторов while и break. Вывести ДА, если условие выполняется, и НЕТ – в противном случае.	В некоторой стране используются денежные купюры достоинством в 1, 2, 4, 8, 16, 32 и 64. Вводится натуральное число n. Как наименьшим количеством таких денежных купюр можно выплатить сумму n? Вывести на экран список купюр для формирования суммы n. Предполагается, что имеется достаточно большое количество купюр всех достоинств.
5	Вводится натуральное число n. Вывести первое найденное натуральное число (то есть, перебирать числа, начиная с 1), квадрат которого больше значения n. Реализовать программу с использованием операторов while и break.	Задается список с вложенностью,равной 3. Например, такой: lst = [1, 2, [True, False, ["a", "ra"]], 3] Необходимо с помощью вложенных циклов for создать новый одномерный список, который бы содержал все значения заданного списка lst. Результат вывести на экран.

6	Вводятся целые числа в одну строчку через пробел. Необходимо преобразовать эти данные в список целых чисел. Затем, перебрать этот список в цикле for и просуммировать все нечетные значения. Результат вывести на экран.	Задано игровое поле для игры «Крестики-нолики» в виде двумерного списка. Например: P = [['x', 'x', 'o'], ['o', 'x', 'x'], ['#', 'x', '#']] Здесь 'x' – крестик; 'o' – нолик; '#' – свободная клетка. Необходимо проверить, есть ли в поле P выигрышная ситуация для крестиков. Вывести «ДА», если крестики победили и «НЕТ» - в противном случае.
7	Вводится список названий городов в одну строчку через пробел. Перебрать все эти названия с помощью цикла for и определить, начинается ли название следующего города на последнюю букву предыдущего города в списке. Если последними встречаются буквы 'ь', 'ъ', 'ы', то берется следующая с конца буква слова. Вывести на экран ДА, если последовательность удовлетворяет правилу, и НЕТ – в противном случае.	Имеется список предметов с указанием их веса (в гр.): карандаш 20 зеркальце 100 зонт 500 рубашка 300 молоток 600 пила 400 удочка 1200 Их следует представить в виде вложенного (двумерного) списка. Вводится натуральное число N – суммарный вес, который можно положить в рюкзак. Предметы кладутся в порядке убывания их веса. Напишите программу, которая для введенного N определит наибольшее число предметов, положенных в рюкзак. На экране отобразить список этих предметов. (Каждый предмет в единственном экземпляре).

8	Вводится строка с номером телефона. Ожидается формат ввода: +7(xxx)xxx-xx-xx где x - это цифра. Необходимо проверить, что введенная строка соответствует этому формату. Вывести ДА, если соответствует, и НЕТ – в противном случае.	Вводится четырехзначное целое положительное число. Подумайте, как можно определить итератор для перебора его цифр. Выведите цифры этого введенного числа с помощью итератора.
9	Вводится список в виде целых чисел в одну строку через пробел. Необходимо сначала сформировать список на основе введенной строки, а затем, каждое значение этого списка изменить, возведя в квадрат. Отобразить результат на экране. Программу следует реализовать с использованием функции enumerate.	Иван Иванович 1 марта открыл счет в банке, положив 1000 руб. Через каждый месяц размер вклада увеличивается на 2 % от имеющейся суммы. Определить, через сколько месяцев размер вклада превысит 1200 руб. Результат (число месяцев) вывести на экран.
10	Вводится список в виде вещественных чисел в одну строку через пробел. С помощью цикла for необходимо найти наименьшее четное значение в этом списке. Полученный результат вывести на экран. Если четного значения нет, то вывести слово "None" (без кавычек). Реализовать программу без использования функции min.	Вводится натуральное число. Необходимо проверить, есть ли в этом числе какие-либо две одинаковые цифры, стоящие рядом. Вывести эти цифры, если такие есть, иначе вывести «НЕТ».

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Тексты программ.
3. Результаты работы программ.

Практическая работа 5. Использование пользовательских регистров для сохранения данных в памяти ЭВМ

Цель работы: научиться использовать генераторы списков при программировании различных практических задач.

Теоретический материал

Теорию для выполнения лабораторной работы смотрите на странице tk.ulstu.ru/video.php#video в следующих видеоматериалах:

· #27. Генераторы списков (List comprehensions)
· #28. Вложенные генераторы списков

Задания на лабораторную работу

1. Написать программы в среде PyCharm в соответствии с номером своего варианта.

Номер	Задание №1	Задание №2
1	Вводятся вещественные числа в строку через пробел. Необходимо на их основе сформировать список с помощью list comprehension (генератора списков) из модулей введенных чисел (в списке должны храниться именно числа, а не строки). Результат вывести на экран.	Задается двумерный (вложенный) список, представляющий таблицу целых чисел. Необходимо с помощью list comprehension преобразовать его в одномерный так, чтобы значения элементов шли в обратном порядке. Результат преобразования отобразить на экране.
2	Вводится семизначное целое положительное число. С помощью list comprehension сформировать список, содержащий цифры этого числа (в списке должны быть записаны числа, а не строки). Результат вывести на экран в одну строку через пробел.	Вводится список целых чисел в строку через пробел. Количество чисел равно N^2. С помощью list comprehension сформировать из них двумерный (вложенный) список размером N x N (квадратную таблицу чисел). Гарантируется, что из набора введенных чисел можно сфор-

3	Вводится натуральное число N. С помощью list comprehension сформировать двумерный список размером N x N, состоящий из нулей, а по главной диагонали - единицы. (Главная диагональ – это элементы, имеющие одинаковые индексы, например, a[1][1], a[2][2], …). Результат вывести на экран.	Имеется список из строк: t = ["– Скажи-ка, дядя, ведь не даром", "Я Python выучил с каналом", "Наместников что раздавал?"] Необходимо преобразовать его в двумерный (вложенный) список, где каждая строка представляется списком из слов (слова разделяются пробелом). При этом сохранять слова только длиной более трех символов. Решить данную задачу с использованием list comprehension. Результат отобразить на экране.
4	Вводятся названия городов в строку через пробел. Необходимо сформировать список с помощью list comprehension, содержащий названия городов длиной более пяти символов. Результат вывести на экран.	Вводятся строки из целых чисел через пробел, пока пользователь не введет пустую строку. Необходимо все введенные строки вначале сохранить в список. Затем, на основе этого списка, используя list comprehension, сформировать двумерный список, где каждый элемент будет представлять одно отдельное число.
5	Вводится натуральное число n. Необходимо сформировать список с помощью list comprehension, состоящий из делителей числа n (включая и само число n). Результат вывести на экран.	Используя вложенный list comprehension, сформируйте двумерный список, представляющий следующую квадратную таблицу чисел размером 4 x 4: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 Результат выведите на экран.

6	Вводится натуральное число N. Необходимо сгенерировать вложенный список с помощью list comprehension, размером N x N, где первая строка содержала бы все нули, вторая - все единицы, третья - все двойки и так до N-й строки. Результат вывести на экран.	Вводятся два вещественных значения a, b (a < b). С помощью list comprehension сформируйте список со значениями синусов от аргументов в диапазоне [a; b] с шагом 0.1. Результат выведите на экран в виде списка чисел с точностью до сотых.
7	Вводится список вещественных чисел. С помощью list comprehension сформировать список, состоящий из элементов введенного списка, имеющих четные индексы (то есть, выбрать все элементы с четными индексами). Результат вывести на экран.	Вводятся названия три строки: первая строка содержит названия городов, вторая – названия стран, а третья – названия рек. Все названия следуют в строке через пробел. С помощью list comprehension сформируйте единый список из слов, длины которых больше пяти. Результат выведите на экран.
8	Вводятся два списка целых чисел одинаковой длины каждый с новой строки. С помощью list comprehension сформировать третий список, состоящий из суммы соответствующих пар чисел введенных списков. Результат вывести на экран.	Имеется двумерный список чисел. Например: d = [[1, 2, 3], [4, 5, 6], [7, 8, 9], [8, 7, 6]] С помощью list comprehension необходимо сформировать новый список, в котором строки идут в обратном порядке. Результат выведите на экран.
9	Вводятся названия стран в одну строчку через пробел. С помощью list comprehension сформировать список, состоящий из названий стран, в которых присутствует фрагмент «ро» (без учёта регистра). Результат вывести на экран.	Имеется трехмерный список. Например: t = [[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [9, 8, 7]], [[0, 1, 2], [-1, -2]]] С помощью list comprehension необходимо сформировать новый одномерный список, состоящий из значений элементов списка t. Результат выведите на экран.

10	Вводятся два натуральных числа a, b (a < b). С помощью list comprehension сформировать список чисел в диапазоне [a; b] с шагом 0.1. Результат вывести на экран.	Вводится строка с координатами точек в формате (пример): 5;4 -3;2 7;56 -4;-10 … То есть, пары координат разделены пробелом, а сами координаты – точкой с запятой. При этом все числовые значения – целые числа. Необходимо с помощью list comprehension преобразовать эту строку в двумерный список вида (пример): [[5, 4], [-3, 2], [7, 56], …] Результат вывести на экран.

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Тексты программ.
3. Результаты работы программ.

РАЗДЕЛ 3. РАЗРАБОТКА СПЕЦИФИКАЦИЙ ДЛЯ КОМПОНЕНТОВ ПРОГРАММНОГО ПРОДУКТА

Практическая работа 6. Управление видеоадаптером. Особенности функционирования видеосистемы

Цель работы: научиться использовать словари, кортежи и множества при программировании различных практических задач.

Теоретический материал

Теорию для выполнения лабораторной работы смотрите на странице tk.ulstu.ru/video.php#video в следующих видеоматериалах:

- #29. Введение в словари (dict). Базовые операции над словарями
- #30. Методы словаря. Перебор его элементов в цикле
- #31. Кортежи (tuple) и их методы
- #32. Множества (set) и их методы
- #33. Операции над множествами. Сравнение множеств
- #34. Генераторы множеств и словарей

Задания на лабораторную работу

1. Написать программы в среде PyCharm в соответствии с номером своего варианта.

Номер	Задание №1	Задание №2
1	Вводится строка из русских букв. Необходимо ее закодировать, а затем, раскодировать азбукой Морзе. После каждой закодированной буквы должен стоять пробел (символ окончания кода буквы). После последнего кода пробела быть не должно (в конце строки). Словарь с кодами азбуки Морзе приведен после этой таблицы. Закодированную строку вывести на экран.	Вводятся названия городов в одну строку через пробел. На их основе формируется кортеж. Если в этом кортеже нет города "Москва", то следует его добавить в конец кортежа. Результат вывести на экран.

2	Вводится строка с русскими и латинскими буквами. Например: «Занятие по языку Python» Необходимо с помощью словаря t (приведен в приложении после этой таблицы) перевести введенную строку в латиницу. Кроме того, символы " ?!;:" заменять на символ дефиса (-). Замены делать без учета регистра (строку перевести в нижний регистр – малые буквы). Результат вывести на экран.	Вводятся имена студентов в одну строчку через пробел. На их основе формируется кортеж. Отобразите на экране все имена из этого кортежа, которые содержат фрагмент "ва". Имена выводятся в одну строку через пробел.
3	С клавиатуры вводятся данные в формате: Имя_1:номер телефона_1 … Имя_N:номер телефона_N пока пользователь не введет пустую строку. Необходимо на основе введенных данных формировать словарь, в котором ключами будут имена, а значениями – список номеров телефонов. Имена в этом списке могут повторяться. Тогда одному имени в словаре будет соответствовать несколько номеров. Результат вывести на экран.	Вводится строка, содержащая латинские символы, пробелы и цифры. Необходимо выделить из нее все неповторяющиеся цифры (символы от 0 до 9). Вывести на экран найденный цифры. Если цифр нет, то вывести слово НЕТ.

4	Задается словарь для перевода с английского на русский язык в виде: d = {'house': 'дом', 'river': 'река', 'car': 'машина', ...} Вводится строка с английскими словами, записанными через пробел. Необходимо заменить в строке все английские слова на русские, присутствующие в словаре d. Результат вывести на экран.	В ночном клубе фиксируется список гостей. Причем гости могут выходить из помещения, а затем, снова заходить. Тогда их имена фиксируются повторно. На вход программы поступает такой список (каждое имя записано с новой строки). Например: Сергей Мария Наталья Евгений Сергей Мария … пока не будет введена пустая строка. Требуется подсчитать общее число гостей, которые посетили ночной клуб. Полагается, что гости имеют уникальные имена. На экран вывести общее число гостей клуба.
5	Имеется словарь с описанием предметов: things = {'карандаш': 20, 'зеркальце': 100, 'зонт': 500, 'рубкашка': 300} Затем, в программе вводятся данные в формате: предмет_1=вес_1 … предмет_2=вес_2 пока не будет введена пустая строка. Необходимо введенные данные сохранить в отдельном словаре d, а затем, в словарь things добавить содержимое словаря d. Результат вывести на экран.	В аккаунте социальной сети Сергея прокомментировали фотографию. Некоторые посетители оставляли несколько комментариев. Требуется по списку комментариев определить уникальное число комментаторов. Комментарии поступают на вход программы в формате: имя 1: комментарий 1 имя 2: комментарий 2 ... имя N: комментарий N пока не будет введена пустая строка. Также полагается, что имена у разных комментаторов

6	Вводится информация по книгам в формате: автор_1: название_1 … автор_N: название_N пока не будет введена пустая строка. Авторы могут повторяться. Необходимо сформировать словарь только из уникальных авторов с первым встретившимся значением. Результат вывести на экран.	Вводятся названия городов в одну строку через пробел. На их основе формируется кортеж. Если в этом кортеже присутствует город "Самара", то следует его удалить. Результат вывести на экран в виде строки с названиями городов через пробел. Обратите внимание, что город Самара может быть записан несколько раз. Тогда нужно удалить все его упоминания.
7	Вводятся оценки студента в одну строчку через пробел. Необходимо сформировать словарь, в котором ключами являются оценки (числа), а значениями – количество этих оценок во введенном списке. Результат вывести на экран.	Вводятся два списка целых чисел каждый с новой строки (в строке наборы чисел через пробел). Необходимо выбрать и отобразить на экране числа, присутствующие в первом списке, но отсутствующие во втором. Результат выведите на экран в виде строки чисел, записанных через пробел. Задачу решить с использованием множеств.

8	Вводятся числа в одну строчку в формате: 1;3 4 5;2 4 3 4;1;6 8 10; … То есть, здесь группы чисел, разделенных точкой с запятой. Одно число – это длина отрезка, три числа – это треугольник, четыре – четырехугольник и т.п. На основе введенных данных необходимо сформировать словарь с ключами: отрезок, треугольник, четырехугольник и т.п., а значениями должен быть двумерный (вложенный) кортеж, содержащий соответствующие группы чисел.	Вводятся два списка городов каждый с новой строки (в каждой строке названия через пробел). Необходимо сравнить их между собой на равенство по уникальным (не повторяющимся) городам. Если списки содержат одни и те же уникальные города, то вывести «ДА», иначе – «НЕТ». Задачу решить с использованием множеств.
9	Вводятся номера телефонов в формате: +7xxxxxxxxxx +6xxxxxxxxxx +7xxxxxxxxxx +5xxxxxxxxxx … пока не будет введена пустая строка. Здесь x – это цифры. Необходимо сформировать словарь, в котором ключами будут коды стран: «+7», «+5», «+6», «+8», …, а значениями – список соответствующих номеров телефонов.	Вводятся два списка городов каждый с новой строки (в строке названия через пробел), которые объехал Сергей в 1-й и 2-й годы своего путешествия по России. Требуется определить, включал ли его маршрут во 2-й год все города 1-го года путешествия? Если это так, то вывести «ДА», иначе – «НЕТ». Задачу решить с использованием множеств.

10	В программе в каждой новой строке вводятся целые числа, пока не будет введено число 0. Необходимо для каждого введенного числа вычислить косинус и вывести результат с точностью до тысячных. Если числовое значение вводится повторно, то заново его вычислять не нужно, а брать ранее вычисленное значение из словаря.	Вводятся два списка целых чисел каждый с новой строки (в строке наборы чисел через пробел). Необходимо выбрать и отобразить на экране числа, присутствующие и в первом и во втором списках. Результат выведите на экран в виде строки чисел, записанных через пробел. Задачу решить с использованием множеств.

Приложение

1.Словарь для кодирования русских букв и символа пробела азбукой Морзе:
morze = {'а': '.-', 'б': '-...', 'в': '.--', 'г': '--.', 'д': '-..', 'е': '.', 'ё': '.', 'ж': '...-', 'з': '--..', 'и': '..', 'й': '.---', 'к': '-.-', 'л': '.-..', 'м': '--', 'н': '-.', 'о': '---', 'п': '.--.', 'р': '.-.', 'с': '...', 'т': '-', 'у': '..-', 'ф': '..- .', 'х': '....', 'ц': '-.-.', 'ч': '---.', 'ш': '----', 'щ': '--.-', 'ъ': '--.--', 'ы': '-.--', 'ь': '-..-', 'э': '..-..', 'ю': '..--', 'я': '.-.-', ' ': '-···-'}

2.Словарь для замены русских букв на соответствующие латинские сочетания букв:
t = {'ё': 'yo', 'а': 'a', 'б': 'b', 'в': 'v', 'г': 'g', 'д': 'd', 'е': 'e', 'ж': 'zh',
'з': 'z', 'и': 'i', 'й': 'y', 'к': 'k', 'л': 'l', 'м': 'm', 'н': 'n', 'о': 'o', 'п': 'p',
'р': 'r', 'с': 's', 'т': 't', 'у': 'u', 'ф': 'f', 'х': 'h', 'ц': 'c', 'ч': 'ch', 'ш': 'sh',
'щ': 'shch', 'ъ': '', 'ы': 'y', 'ь': '', 'э': 'e', 'ю': 'yu', 'я': 'ya'}

Практическая работа 7. Управление программами в объектно-ориентированной среде

Цель работы: научиться основам работы с функциями, в том числе с рекурсивными и анонимными.

Теоретический материал

Теорию для выполнения лабораторной работы смотрите на странице сайтatk.ulstu.ru/video.php#video в следующих видеоматериалах:

- #35. Что такое функции. Их объявление и вызов
- #36. Оператор return
- #37. Алгоритм Евклида для нахождения НОД
- #38. Именованные аргументы. Фактические и формальные параметры
- #39. Функции с произвольным числом параметров
- #40. Операторы упаковки и распаковки коллекций
- #41. Рекурсивные функции
- #42. Анонимные (lambda) функции

Задания на лабораторную работу

Вариант №1

Задание №1. Объявите функцию для проверки числа на нечетность (возвращается True, если переданное число нечетное и False, если число четное).
После объявления функции прочитайте (с помощью функции input) список целых значений, записанных в одну строку через пробел. И, используя генератор списков и объявленную функцию, сформируйте список из нечетных значений на основе введенного исходного списка. Результат отобразите на экране.

Задание № 2. Вводится натуральное число n. Необходимо с помощью рекурсивной функции сформировать последовательность чисел Фибоначчи по правилу: первые два числа равны 1 и 1, а каждое следующее значение равно сумме двух предыдущих. Пример такой последовательности для первых 7 чисел: 1, 1, 2, 3, 5, 8, 13, ... Функция должна возвращать список сформированной последовательности длиной n.

Вариант №2

Задание №1. Вводится слово. Если это слово RECT, то следует объявить функцию с именем get_sq, двумя параметрами, вычисляющую площадь прямоугольника и возвращающую вычисленное значение. Если же введенное слово не RECT (любое другое), то объявляется функция с тем же именем get_sq, с двумя параметрами для вычисления периметра прямоугольника. Вычисленное значение возвращается функцией.
В конце программы вызовите функцию с двумя аргументами и выведите результат на экран.

Задание № 2. Вводится натуральное число n. Необходимо с помощью рекурсивной функции вычислить факториал числа n. Напомню, что факториал числа, равен: n! = 1 * 2 * 3 *...* n. Функция должна возвращать вычисленное значение.

Вызовите эту функцию и отобразите на экране результат ее работы.

Вариант №3

Задание №1. Объявите функцию, которая принимает строку (в качестве аргумента) и возвращает False, если длина строки меньше трех символов. Иначе возвращается значение True.
После объявления функции прочитайте (с помощью функции input) список названий городов, записанных в одну строку через пробел. Затем, используя генератор списка и созданную функцию, сформируйте список из названий городов длиной не менее трех символов на основе введенного исходного списка. Результат отобразите на экране.

Задание № 2. Имеется следующий многомерный список:

d = [1, 2, [True, False], ["Москва", "Уфа", [100, 101], [True, [1, -1]]], 7.89]

С помощью рекурсивной функции создать на его основе одномерный список из значений элементов списка d. Функция должна возвращать новый созданный одномерный список.

Вызовите эту функцию и отобразите на экране результат ее работы.

Вариант №4

Задание №1. Объявите функцию, которая принимает строку (в качестве аргумента) и возвращает два значения в виде кортежа: переданная строка и ее длина.

После объявления функции прочитайте (с помощью функции input) список названий городов, записанных в одну строку через пробел. Затем, используя генератор словарей и созданную функцию, сформируйте словарь d в формате:

d = {<город 1>: <число символов>, ..., <город N>: <число символов>}

Задание № 2. Объявите анонимную (лямбда) функцию для вычисления модуля числа (то есть, отрицательные числа нужно делать положительными).

Вызовите эту функцию для введенного с клавиатуры вещественного числа и отобразите результат на экране.

Вариант №5

Задание №1. Вводится список целых чисел в одну строчку через пробел. Необходимо задать функцию с именем get_sq, которая принимает два аргумента (максимальное и минимальное значения из списка) и возвращает их произведение. Вызовите эту функцию и отобразите на экране полученное числовое значение.

Подсказка: для передачи аргументов функции используйте функции max и min для введенного списка чисел.

Задание № 2. Объявите обычную функцию, которая первым параметром принимает список, а вторым – ссылку на функцию. Она должна возвращать отфильтрованный список. Делается это по следующему алгоритму. В цикле переберите элементы списка, для каждого вызовите, переданную вторым параметром, функцию и если она возвращает True, то элемент должен быть добавлен в возвращаемый основной функцией список.

Вызовите объявленную функцию, первым параметром передайте список, а вторым запишите лямбда-функцию для отбора элементов из передаваемого списка. Результат работы функции выведите на экран.

Вариант №8

Задание №1. Напишите функцию, которая проверяет корректность переданного ей email-адреса в виде строки. Будем полагать, что адрес верен, если он обязательно содержит символы '@' и '.', а все остальные символы могут принимать значения: 'a-z', 'A-Z', '0-9' и '_'. Если email верен, то функция возвращает True, иначе – False.

После объявления функции прочитайте (с помощью функции input) строку с email-адресом и вызовите функцию с этим аргументом. Результат отобразите на экране.

Задание № 2. Объявите анонимную (лямбда) функцию для определения вхождения в строку фрагмента "ra". То есть, функция должна возвращать True, если такой фрагмент присутствует в строке и False – в противном случае.

Вызовите эту функцию для введенной строки и отобразите результат ее работы на экране.

Вариант №9

Задание №1. Объявите функцию, которая принимает список, находит максимальное, минимальное и сумму значений этого списка и возвращает все вычисленные результаты в виде кортежа.

Вызовите объявленную функцию со списком чисел, введенными с клавиатуры и отобразите результат работы функции на экране.

Задание № 2. С помощью рекурсивной функции найдите минимальное значение элементов, переданного ей кортежа. Функция должна возвращать найденное минимальное значение.

Вызовите эту функцию и отобразите результат ее работы на экране.

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Тексты программ.
3. Результаты работы программ.

Практическая работа 8. Использование пользовательских регистров для обработки данных

Цель работы: познакомиться с механизмом замыканий функций и декораторами функций.

Теоретический материал

Теорию для выполнения лабораторной работы смотрите на странице tk.ulstu.ru/video.php#video в следующих видеоматериалах:

- #43. Области видимости. Ключевые слова global и nonlocal
- #44. Замыкания в Python. Вложенные функции
- #45. Декораторы функций
- #46. Передача аргументов декораторам

Задания на лабораторную работу

Вариант №1

Задание №1. Используя замыкания функций, объявите внутреннюю функцию, которая заключает строку s (s – строка, параметр внутренней функции) в произвольный тег, содержащийся в переменной tag – параметре внешней функции.

Далее, на вход программы поступает две строки: первая с тегом, вторая с некоторым содержимым. Вторую строку нужно поместить в тег из первой строки с помощью реализованного замыкания. Результат выведите на экран.

Задание №2. На вход программы поступает строка из целых чисел, записанных через пробел. Напишите функцию get_list, которая преобразовывает эту строку в список из целых чисел и возвращает его.

Определите декоратор для этой функции, который сортирует список чисел, полученный из вызываемой в нем функции. Результат сортировки должен возвращаться при вызове декоратора.

Вариант №2

Задание №1. Используя замыкания функций, объявите внутреннюю функцию, которая преобразует строку из списка целых чисел, записанных через пробел, либо в список, либо в кортеж. Тип коллекции определяется параметром type внешней функции. Если type = 'list', то используется список, иначе – кортеж.

Далее, на вход программы поступает две строки: первая – это значение для параметра type; вторая – список целых чисел, записанных через пробел. С помощью реализованного замыкания преобразовать эту строку в соответствующую коллекцию. Результат работы замыкания выведите на экран.

Задание № 2. Вводятся два списка (каждый с новой строки) из слов, записанных через пробел. Имеется функция, которая преобразовывает эти две строки в два списка слов и возвращает эти списки.

Определите декоратор для этой функции, который из этих двух списков формирует словарь, в котором ключами являются слова из первого списка, а значениями – соответствующие элементы из второго списка. Полученный словарь должен возвращаться при вызове декоратора.

Примените декоратор к первой функции и вызовите ее. Результат (словарь) отобразите на экране.

Вариант №3

Задание №1. Используя замыкания функций, объявите внутреннюю функцию, которая из переданного ей списка строк формирует многострочную строку вида:

строка_1
…
строка_N

и возвращает ее. Где строка_1, строка_2, … — это строки из переданного функции списка. Вызовите внутреннюю функцию замыкания и отобразите на экране результат ее работы.

Задание № 2. Объявите функцию с именем to_lat, которая принимает строку на кириллице и преобразовывает ее в латиницу, используя следующий словарь для замены русских букв на соответствующее латинское написание:
t = {'ё': 'yo', 'а': 'a', 'б': 'b', 'в': 'v', 'г': 'g', 'д': 'd', 'е': 'e', 'ж': 'zh',
'з': 'z', 'и': 'i', 'й': 'y', 'к': 'k', 'л': 'l', 'м': 'm', 'н': 'n', 'о': 'o', 'п': 'p',
'р': 'r', 'с': 's', 'т': 't', 'у': 'u', 'ф': 'f', 'х': 'h', 'ц': 'c', 'ч': 'ch', 'ш': 'sh',
'щ': 'shch', 'ъ': '', 'ы': 'y', 'ь': '', 'э': 'e', 'ю': 'yu', 'я': 'ya'}
Функция должна возвращать преобразованную строку. Замены делать без учета регистра (исходную строку перевести в нижний регистр – малые буквы). Все небуквенные символы "! ?:;.,_" превращать в символ '-' (дефиса). Определите декоратор для этой функции, который несколько подряд идущих дефисов, превращает в один дефис. Полученная строка должна возвращаться при вызове декоратора.
Примените декоратор к функции to_lat и вызовите ее Результат работы декорированной функции отобразите на экране.

Вариант №4

Задание №1. Используя замыкания функций, объявите внутреннюю функцию, которая принимает в качестве параметров фамилию и имя, а затем, заносит в шаблон эти данные. Сам шаблон – это строка, которая передается внешней функции и, например, может иметь такой вид:
«Уважаемый %F%, %N%! Вы делаете работу по замыканиям функций.»
Здесь %F% - это фрагмент куда нужно подставить фамилию, а %N% - фрагмент, куда нужно подставить имя. (Шаблон может быть и другим, вы это определяете сами). Здесь важно, чтобы внутренняя функция умела подставлять данные в шаблон, формировать новую строку и возвращать результат. Вызовите внутреннюю функцию замыкания и отобразите на экране результат ее работы.

Задание № 2. Вводится строка целых чисел через пробел. Напишите функцию, которая преобразовывает эту строку в список чисел и возвращает их сумму.
Определите декоратор для этой функции, который имеет один параметр start – начальное значение суммы. Примените декоратор со значением start=5 к функции и вызовите декорированную функцию. Результат отобразите на экране.

Вариант №5

Задание №1. Используя замыкания функций, объявите внутреннюю функцию, которая бы все повторяющиеся символы заменяла одним другим указанным символов. Какие повторяющиеся символы искать и на что заменять, определяются параметрами внешней функции. Внутренней функции передается только строка для преобразования. Преобразованная (сформированная) строка должна возвращаться внутренней функцией.

Вызовите внутреннюю функцию замыкания и отобразите на экране результат ее работы.

Задание № 2. Объявите функцию, которая возвращает переданную ей строку в нижнем регистре (с малыми буквами).

Определите декоратор для этой функции, который имеет один параметр tag, определяющий строку с названием тега (начальное значение параметра tag равно "h1"). Этот декоратор должен заключать возвращенную функцией строку в тег tag и возвращать результат.

Пример заключения строки "python" в тег h1: <h1>python</h1>

Примените декоратор со значением tag="div" к функции и вызовите декорированную функцию. Результат отобразите на экране.

Вариант №6

Задание №1. Используя замыкания функций, объявите внутреннюю функцию, которая на основе двух параметров вычисляет площадь фигуры. Какой именно фигуры: треугольника или прямоугольника, определяется параметром type внешней функции. Если type принимает значение 0, то вычисляется площадь треугольника, а иначе – прямоугольника. По умолчанию параметр type должен быть равен 0. Вычисленное значение должно возвращаться внутренней функцией.

Вызовите внутреннюю функцию замыкания и отобразите на экране результат ее работы.

Задание № 2. Объявите функцию, которая вычисляет периметр многоугольника и возвращает вычисленное значение. Длины сторон многоугольника передаются в виде коллекции (списка или кортежа).

Определите декоратор для этой функции, который выводит на экран сообщение:

«Периметр фигуры равен = <число>»

Вариант №7

Задание №1. Используя замыкания функций, объявите внутреннюю функцию, которая принимает два параметра a, b, а затем, возвращает строку в формате:

«Для значений a, b функция f(a,b) = <число>»

где число – это вычисленное значение функции f. Ссылка на f передается как аргумент внешней функции.

Вызовите внутреннюю функцию замыкания и отобразите на экране результат ее работы. Функцию f придумайте самостоятельно (она должна что то делать с двумя параметрами a, b и возвращать результат).

Задание № 2. Объявите функцию, которая вычисляет площадь круга и возвращает вычисленное значение. В качестве аргумента ей передается значение радиуса.

Определите декоратор для этой функции, который выводит на экран сообщение:

«Площадь круга равна = <число>»

В строке выведите числовое значение с точностью до сотых. Примените декоратор к функции и вызовите декорированную функцию.

Вариант №8

Задание №1. Используя замыкания функций, объявите внутреннюю функцию, которая принимает в качестве аргумента коллекцию (список или кортеж) и возвращает или минимальное значение, или максимальное, в зависимости от значения параметра type внешней функции. Если type равен «max», то возвращается максимальное значение, иначе – минимальное. По умолчанию type должно принимать значение «max».

Вызовите внутреннюю функцию замыкания и отобразите на экране результат ее работы.

Задание № 2. Объявите функцию, которая принимает строку на кириллице и преобразовывает ее в латиницу, используя следующий словарь для замены русских букв на соответствующее латинское написание:

```
t = {'ё': 'yo', 'а': 'a', 'б': 'b', 'в': 'v', 'г': 'g', 'д': 'd', 'е': 'e', 'ж': 'zh',
    'з': 'z', 'и': 'i', 'й': 'y', 'к': 'k', 'л': 'l', 'м': 'm', 'н': 'n', 'о': 'o', 'п': 'p',
    'р': 'r', 'с': 's', 'т': 't', 'у': 'u', 'ф': 'f', 'х': 'h', 'ц': 'c', 'ч': 'ch', 'ш': 'sh',
    'щ': 'shch', 'ъ': '', 'ы': 'y', 'ь': '', 'э': 'e', 'ю': 'yu', 'я': 'ya'}
```

Функция должна возвращать преобразованную строку. Замены делать без учета регистра (исходную строку перевести в нижний регистр – малые буквы).

Определите декоратор с параметром chars и начальным значением " !?", который данные символы преобразует в символ "-" и, кроме того, все подряд идущие дефисы (например, "--" или "---") приводит к одному дефису. Полученный р езультат должен возвращаться в виде строки.

Примените декоратор со значением chars="?!:;,. " к функции и вызовите декорированную функцию. Результат отобразите на экране.

Вариант №9

Задание №1. Используя замыкания функций, объявите внутреннюю функцию, которая принимает в качестве аргумента список целых чисел и удаляет из него все четные или нечетные значения в зависимости от значения параметра type. Если type равен «even», то удаляются четные значения, иначе – нечетные. По умолчанию type должно принимать значение «even».

Вызовите внутреннюю функцию замыкания и отобразите на экране результат ее работы.

Задание №2. Объявите функцию, которая принимает строку, удаляет из нее все подряд идущие пробелы и переводит ее в нижний регистр – малые буквы. Результат (строка) возвращается функцией.

Определите декоратор, который строку, возвращенную функцией, переводит в азбуку Морзе, используя следующий словарь для замены русских букв и символа пробела на соответствующие последовательности из точек и тире:

morze = {'а': '.-', 'б': '-...', 'в': '.--', 'г': '--.', 'д': '-..', 'е': '.', 'ё': '.', 'ж': '...-', 'з': '--..', 'и': '..', 'й': '.---', 'к': '-.-', 'л': '.-..', 'м': '--', 'н': '-.', 'о': '---', 'п': '.--.', 'р': '.-.', 'с': '...', 'т': '-', 'у': '..-', 'ф': '..- .', 'х': '....', 'ц': '-.-.', 'ч': '---.', 'ш': '----', 'щ': '--.-', 'ъ': '--.--', 'ы': '-.--', 'ь': '-..-', 'э': '..-..', 'ю': '..--', 'я': '.-.-', ' ': '-···-'}

Преобразованная строка возвращается декоратором.

Примените декоратор к функции и вызовите декорированную функцию. Результат работы отобразите на экране.

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Тексты программ.
3. Результаты работы программ.

РАЗДЕЛ 4. РАЗРАБОТКА КОДА ПРОГРАММНОГО ПРОДУКТА НА УРОВНЕ МОДУЛЯ

Практическая работа 9. Обработка числовых данных при вводе и выводе

Цель работы: научиться выполнять импорт модулей в программах на языке Python и работать с файлами.

Теоретический материал

Теорию для выполнения лабораторной работы смотрите на странице tk.ulstu.ru/video.php#video в следующих видеоматериалах:

- #47. Импорт стандартных модулей. Команды import и from
- #48. Импорт собственных модулей
- #49. Установка сторонних модулей. Пакетная установка
- #50. Пакеты (package) в Python
- #51. Функция open. Чтение данных из файла
- #52. Обработка исключения FileNotFoundError и менеджер контекста
- #53. Запись данных в файл

Задания на лабораторную работу

Вариант №1

Задание №1. Импортируйте в программу стандартный модуль time и, используя одноименную функцию time из этого модуля, определите время работы программы, которая сначала сохраняет в файл с именем out.txt строку:

«Работа №3. Импорт модулей и пакетов. Работа с файлами»

а, затем, читает из этого файла его содержимое и отображает результат (чтения из файла) на экран. Также на экран нужно вывести время работы программы.

Вариант №2

Задание №1. Импортируйте в программу стандартный модуль math и, используя функцию log2, вычислите логарифм по основанию 2 для целых чисел от 1 до 10. Результат представьте в виде списка, который сформируйте с помощью генератора списков.

Полученный список сохраните в файл в бинарном режиме доступа. А, затем, прочитайте его содержимое в другой список. Выведите на экран прочитанные
данные (список).

Вариант №3

Задание №1. Импортируйте в программу стандартный модуль random и, используя функцию randint, сгенерируйте 100 случайных чисел в диапазоне [-2; 5]. Все эти числа следует представить в виде списка, сформированного с помощью генератора списков.

Сохраните полученные данные в файл в текстовом режиме доступа. Затем, прочитайте из него данные и поместите в другой список. Выведите на экран полученный список.

Вариант №4

Задание №1. Импортируйте в программу стандартный модуль math и, используя функцию log10, вычислите логарифм по основанию 10 для целых чисел от 1 до
10. Результат представьте в виде множества, который сформируйте с помощью генератора множеств.

Полученный список сохраните в файл в текстовом режиме доступа в виде строки вещественных чисел с точностью до тысячных, записанных через пробел. Затем, прочитайте содержимое этого файла и поместите в другое множество. Выведите на экран прочитанные данные (множество).

Вариант №5

Задание №1. Импортируйте в программу стандартный модуль time и, используя одноименную функцию time из этого модуля, определите время работы программы, которая сначала сохраняет в файл с именем out.dat кортеж (в бинарном режиме доступа):

books = (('Пушкин А. С.', 'Пикова дама'), ('Булгаков М. А.', 'Мастер и Маргарита'), ('Конан Дойль', 'Собака Баскервилей'))

а, затем, читает из этого файла (также в бинарном режиме) его содержимое и отображает прочитанный кортеж на экран. Также на экран нужно вывести время работы программы.

Вариант №6

Задание №1. Импортируйте в программу стандартный модуль random и, используя одноименную функцию random, сгенерируйте 100 случайных чисел. Все эти числа следует представить в виде списка, сформированного с помощью генератора списков.

Сохраните полученные данные в файл в бинарном режиме доступа. Затем, прочитайте из него данные и поместите в другой список. Выведите на экран
полученный список.

Вариант №7

Задание №1. Импортируйте в программу стандартный модуль math и, используя функцию factorial, вычислите факториал для целых чисел от 1 до 10. Результат представьте в виде словаря, ключами которого являются аргументы функции (числа от 1 до 10), а значениями – вычисленный факториал. Сформируйте словарь с помощью генератора словарей.

Полученный словарь сохраните в файл в текстовом режиме доступа (формат данных придумайте сами). Затем, прочитайте содержимое этого файла и поместите в другой словарь. Выведите на экран прочитанные данные (словарь).

Вариант №8

Задание №1. Импортируйте в программу стандартный модуль random и, используя функцию randint, сформируйте двумерный список, размером 5x5 элементов, со случайными значениями в диапазоне [-5; 5]. Двумерный список следует сформировать с помощью вложенных генераторов списков.

Сохраните полученные данные в файл в бинарном режиме доступа. Затем, прочитайте из него данные и поместите в другой список. Выведите на экран полученный двумерный список.

Вариант №9

Задание №1. Импортируйте в программу стандартный модуль math и, используя функцию sin, вычислите синус для значений от -π до π с шагом 0.1. Результат представьте в виде списка. Сформируйте список с помощью генератора списков.

Полученный список сохраните в файл в бинарном режиме доступа. Затем, прочитайте содержимое этого файла и поместите в другой список. Выведите на экран прочитанные данные (список).

Вариант №10

Задание №1. Импортируйте в программу стандартный модуль time, используя одноименную функцию time из этого модуля, определите время работы программы, которая сохраняет в файл таблицу умножения для целых чисел от 1 до 10 (каждое умножается от 1 до 10). Затем, открывает этот же файл на чтение и формирует из прочитанных данных двумерный кортеж в формате:

((2, 1, 2), (2, 2, 4), (2, 3, 6), …,)

Выведите на экран кортеж и время работы программы.

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Тексты программ.
3. Результаты работы программ.

Практическая работа 10. Проверка состава оборудования. Механизмы взаимодействия с аппаратными ресурсами

Цель работы: научиться использовать генераторы и функции-генераторы в программах на языке Python.

Теоретический материал

Теорию для выполнения лабораторной работы смотрите на странице tk.ulstu.ru/video.php#video в следующих видеоматериалах:

- #54. Выражения-генераторы
- #55. Функция-генератор. Оператор yield

Задания на лабораторную работу

Вариант №1

Задание №1. Используя символы малых букв латинского алфавита (строка ascii_lowercase):

from string import ascii_lowercase

запишите генератор, который бы возвращал все сочетания из двух букв латинского алфавита. Выведите первые 50 сочетаний на экран.

Задание №2. Имеется функция $f(x) = 0{,}1x^2 + 5x - 2$. Необходимо записать функцию-генератор, которая бы выдавала значения этой функции в диапазоне [a; b] с шагом 0,01. Параметры a, b – это параметры, определенные у функции- генератора. Вызовите эту функцию со значениями a=-5, b = 7 и выведите на экран первые 20 значений, которые она возвратит. Какие-либо коллекции в программе не использовать.

Вариант №2

Задание №1. Вводится два целых числа a, b (a < b). Необходимо определить генератор, который бы возвращал модули чисел, в диапазоне [a; b]. Вывести первые четыре значения (или меньше, если значений меньше четырех) этого генератора на экран. (Программа должна быть универсальной и не меняться для разного количества генерируемых данных).

Задание № 2. Задайте функцию-генератор, которая на основе символов строки chars:

```
from string import ascii_lowercase, ascii_uppercase
chars = ascii_lowercase + ascii_uppercase + "0123456789!?@#$*"
```

формирует и выдает случайно сгенерированные пароли длиной 12 символов. Количество выдаваемых паролей функцией должно быть неограниченным. Случайный выбор символа из последовательности chars можно реализовать с помощью функции choice модуля random.

Выведите первые пять сгенерированных паролей на экран.

Вариант №3

Задание №1. Используя модуль random и его функцию randint(a, b) для генерации целых чисел в диапазоне [a; b], а также символы малых букв латинского алфавита (строка ascii_lowercase):

```
from string import ascii_lowercase
```

Запишите генератор, который бы возвращал 1 000 000 строк из случайно выбранных символов. Длина каждой строки должна быть 10 символов. Выведите на экран первые пять сгенерированных строк.

Задание № 2. Написать функцию-генератор, которая должна выдавать значения арифметической последовательности: $a_n = a_{n-1} + d$. Значения a_0, d - указываются как параметры этой функции. Вызовите эту функцию со значениями аргументов $a_0 = -0{,}5$, $d = 0.1$ и выведите на экран первые 50 значений, возвращенных этой функцией. Какие-либо коллекции в программе не использовать.

Вариант №4

Задание №1. Имеется список из названий городов:
cities = ["Москва", "Ульяновск", "Самара", "Уфа", "Омск", "Тула"]

Необходимо записать генератор, который бы используя этот список, выдавал 1 000 000 наименований городов по циклу. То есть, дойдя до конца списка, возвращался в начало и повторял перебор. И так, для выдачи миллиона названий. Вывести на экран первые 20 наименований городов с помощью генератора.

Задание № 2. Запишите функцию-генератор, которая бы выдавала кортежи длиной n, состоящие из случайных целых чисел в диапазоне [a; b]. Величины n, a, b должны определяться как параметры функции-генератора. Сгенерировать случайные целые величины в диапазоне [a; b] можно функцией randint(a, b) модуля random.

Используя эту функцию-генератор, сформируйте с помощью генератора списков двумерную коллекцию размером 20x20 элементов, состоящую из случайных чисел в диапазоне [-5; 5]. Вывести полученную двумерную коллекцию на экран.

Вариант №5

Задание №1. Напишите генератор, который бы возвращал площади кругов с радиусами в диапазоне от 10 до 10 000. Вывести на экран первые пять значений с точностью до сотых.

Задание № 2. Задайте функцию-генератор, которая на основе символов строки to_emails:

from string import ascii_lowercase, ascii_uppercase
to_emails = ascii_lowercase + ascii_uppercase +

"0123456789_" формирует и выдает случайно

сгенерированные email-адреса вида: xxxxxxxx@mail.ru

где x – случайный символ из строки to_emails. Количество выдаваемых email- адресов функцией должно быть неограниченным. Случайный выбор символа из последовательности to_emails можно реализовать с помощью функции choice модуля random.

Выведите первые семь сгенерированных адресов на экран.

Вариант №7

Задание №1. Имеется строка, содержащая

цифры: d = "0123456789"

Необходимо записать генератор, который бы формировал и выдавал все сочетания из трех цифр. Выдавать он должен именно целые числа, а не строки. Вывести первые 50 сгенерированных значений на экран.

Задание № 2. Определите функцию-генератор, которая бы возвращала значения
некоторой функции f(x) для значений $x \in [a;b]$ с шагом 0,01. Величины a, b, a
также функцию f, определить как параметры фукнции-генератора.

Вызовите функцию-генератор со значениями $a = -20;\ b = 100$ для функции $f\ (x) = -1,5x + 2$, которую определите как лямбда-функцию непосредственно в аргументе функции-генератора. Выведите на экран первые 20 значений, возвращенные функцией-генератором. Какие-либо коллекции в программе не использовать.

Вариант №8

Задание №1. Напишите генератор, который бы возвращал площади прямоугольников с длиной, изменяющейся в диапазоне [200; 10 000], и шириной – в диапазоне [500; 100 000]. Вывести на экран первые 100 значений.

Задание № 2. Запишите функцию-генератор, которая бы выдавала списки длиной n, состоящие из случайно выбранных нот из списка p:

p = ['до', 'ре', 'ми', 'фа', 'соль', 'ля', 'си']

Величины n, a, b должны определяться как параметры функции-генератора. Выбрать случайное значение из списка p можно функцией choice модуля random.

Используя эту функцию-генератор, сформируйте с помощью генератора списков двумерную коллекцию размером 10x10 элементов. Выведите полученный двумерный список на экран.

Вариант №9

Задание №1. Имеется график линейной функции $f(x) = 0{,}5x - 2$. Записать генератор, который бы выдавал значения этой функции в диапазоне $x \in [0;100]$ с шагом 0,01 (для x). Вывести первые 50 сгенерированных значений.

Задание № 2. Определите функцию-генератор, которая бы перебирала все комбинации из трех целых положительных чисел, каждое в диапазоне [2; 20]. Для каждой тройки чисел она должна определять, могут ли они образовывать длины сторон треугольника. (Критерий: длина любой стороны треугольника должна быть меньше суммы двух других). Функция должна возвращать текущие длины сторон и булево значение True – если числа образуют треугольник и False – в противном случае.

Выведите с помощью этой функции-генератора первые 20 значений. Какие-либо коллекции в программе не использовать.

Вариант №10

Задание №1. Имеется список из нот:
p = ['до', 'ре', 'ми', 'фа', 'соль', 'ля', 'си']

Необходимо записать генератор, который бы случайным образом выдавал ноты из этого списка 1 000 000 раз. Для случайного выбора нот, используйте функцию choice модуля random. Выведите на экран первые 20 значений этого генератора.

Задание №2. Определите функцию-генератор, которая бы возвращала целые числа кратные 3, начиная поиск от значения a, которое задается через параметр функции. Вызовите эту функции со значением аргумента a=-100 и выведите первые 20 чисел, возвращенных этой функцией-генератором. Какие-либо коллекции в программе не использовать.

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Тексты программ.
3. Результаты работы программ.

РАЗДЕЛ 5. РАЗРАБОТКА МОДУЛЕЙ СИСТЕМНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Практическая работа 11. Реализация механизмов взаимодействия с аппаратными устройствами через порты ввода-вывода

Цель работы: научиться работать со встроенными функциями: map, filter, zip, all и any. А также выполнять сортировку по ключу методом sort и функцией sorted.

Теоретический материал

Теорию для выполнения лабораторной работы смотрите на странице tk.ulstu.ru/video.php#video в следующих видеоматериалах:

- #56. Функция map
- #57. Функция filter
- #58. Функция zip
- #59. Сортировка с помощью sort и sorted
- #60. Аргумент key для сортировки по ключу
- #61. Функция isinstance для проверки типов данных
- #62. Функции all и any

Задания на лабораторную работу

Вариант №1 **Задание №1.** На вход поступает список из вещественных чисел, записанных в строку через пробел. С помощью функции map преобразовать числа в строке в их вещественное представление и отобразить первые три числа. (Полагается, что минимум три вещественных числа имеются). Реализовать извлечение чисел через функцию next.
Вариант №2. **Задание №1.** Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию filter, которая бы возвращала только названия длиной более 5 символов. Извлеките первые три значения на выходе этой функции с помощью next и отобразите их на экране (Полагается, что минимум три значения имеются).

Вариант №3

Задание №1. Вводятся два списка целых чисел. Необходимо попарно перебрать
их элементы и перемножить между собой. При реализации программы используйте функции zip и map. Выведите на экран первые три значения, используя функцию next. (Полагается, что три выходных значения всегда будут присутствовать).

Вариант №4

Задание №1. Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию map, которая бы возвращала названия городов только длиной более 5 символов. Вместо остальных названий - строку с дефисом ("-"). Сформировать список из полученных значений и отобразить его на экране.

Вариант №5

Задание №1. Вводится строка из слов, записанных через пробел. Необходимо на их основе составить прямоугольную таблицу из трех столбцов и N строк (число строк столько, сколько получится). Лишнее (выходящее) слово - отбросить. Реализовать эту программу с использованием функции zip. Результат отобразить на экране в виде прямоугольной таблицы из слов, записанных через пробел (в каждой строчке).

Вариант №6

Задание №1. Вводится список целых чисел в одну строчку через пробел. Необходимо оставить в нем только двузначные числа. Реализовать программу с использованием функции filter. Результат отобразить на экране в виде последовательности оставшихся чисел.

Вариант №7

Задание №1. На вход функции с именем get_sort поступает словарь, например, такой:

d = {'cat': 'кот', 'horse': 'лошадь', 'tree': 'дерево', 'dog': 'собака', 'book': 'книга'}

Необходимо отсортировать словарь d по убыванию ключей (лексикографическая сортировка строк) и возвратить список из соответствующих значений ключей словаря. Например, для указанного словаря d, результатом должен быть список:

['дерево', 'лошадь', 'собака', 'кот', 'книга']

Вызовите функцию get_sort для заданного в программе произвольного словаря dct и выведите возвращенный ею список на экран.

Вариант №8

Задание №1. Саша и Галя коллекционируют монетки. Каждый из них решил записать номиналы монеток из своей коллекции. Получилось два списка. Эти списки поступают на вход программы в виде двух строк из целых чисел, (числа записаны через пробел). Необходимо выделить значения, присутствующие в
обоих списках и оставить среди них только четные. Результат вывести на экран.

При реализации программы используйте функцию filter и кое-что еще (для упрощения программы), подумайте что.

Вариант №9

Задание №1. На вход программы поступает список целых чисел, записанных в одну строчку через пробел. Необходимо выбрать из них четыре наибольших уникальных значения. Результат вывести на экран.

Вариант №10

Задание №1. Вводится список email-адресов в одну строчку через пробел. Среди них нужно оставить только корректно записанные адреса. Будем полагать, что к таким относятся те, что используют латинские буквы, цифры и символ подчеркивания. А также в адресе сначала должен идти символ "@", а затем, точка "." (между ними, конечно же, могут быть и другие символы).

Отобразить результат отбора корректных адресов на экране.

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Тексты программ.
3. Результаты работы программ.

Практическая работа 12. Использование системных ресурсов через обработку прерывания

Цель работы: научиться работать со встроенными функциями: map, filter, zip, all и any. А также выполнять сортировку по ключу методом sort и функцией sorted.

Теоретический материал

Теорию для выполнения лабораторной работы смотрите на странице tk.ulstu.ru/video.php#video в следующих видеоматериалах:

- #56. Функция map
- #57. Функция filter
- #58. Функция zip
- #59. Сортировка с помощью sort и sorted
- #60. Аргумент key для сортировки по ключу
- #61. Функция isinstance для проверки типов данных
- #62. Функции all и any

Задания на лабораторную работу

Вариант №1 **Задание №1.** На вход поступает список из вещественных чисел, записанных в строку через пробел. С помощью функции map преобразовать числа в строке в их вещественное представление и отобразить первые три числа. (Полагается, что минимум три вещественных числа имеются). Реализовать извлечение чисел через функцию next.
Вариант №2. **Задание №1.** Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию filter, которая бы возвращала только названия длиной более 5 символов. Извлеките первые три значения на выходе этой функции с помощью next и отобразите их на экране (Полагается, что минимум три значения имеются).
Вариант №3 **Задание №1.** Вводятся два списка целых чисел. Необходимо попарно перебрать их элементы и перемножить между собой. При реализации программы используйте функции zip и map. Выведите на экран первые три значения, используя функцию next. (Полагается, что три выходных значения всегда будут присутствовать).

Вариант №4

Задание №1. Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию map, которая бы возвращала названия городов только длиной более 5 символов. Вместо остальных названий - строку с дефисом ("-"). Сформировать список из полученных значений и отобразить его на экране.

Вариант №5

Задание №1. Вводится строка из слов, записанных через пробел. Необходимо на их основе составить прямоугольную таблицу из трех столбцов и N строк (число строк столько, сколько получится). Лишнее (выходящее) слово - отбросить. Реализовать эту программу с использованием функции zip. Результат отобразить на экране в виде прямоугольной таблицы из слов, записанных через пробел (в каждой строчке).

Вариант №6

Задание №1. Вводится список целых чисел в одну строчку через пробел. Необходимо оставить в нем только двузначные числа. Реализовать программу с использованием функции filter. Результат отобразить на экране в виде последовательности оставшихся чисел.

Вариант №7

Задание №1. На вход функции с именем get_sort поступает словарь, например, такой:

d = {'cat': 'кот', 'horse': 'лошадь', 'tree': 'дерево', 'dog': 'собака', 'book': 'книга'}

Необходимо отсортировать словарь d по убыванию ключей (лексикографическая сортировка строк) и возвратить список из соответствующих значений ключей словаря. Например, для указанного словаря d, результатом должен быть список:

['дерево', 'лошадь', 'собака', 'кот', 'книга']

Вызовите функцию get_sort для заданного в программе произвольного словаря dct и выведите возвращенный ею список на экран.

Вариант №8

Задание №1. Саша и Галя коллекционируют монетки. Каждый из них решил записать номиналы монеток из своей коллекции. Получилось два списка. Эти списки поступают на вход программы в виде двух строк из целых чисел, (числа записаны через пробел). Необходимо выделить значения, присутствующие в
обоих списках и оставить среди них только четные. Результат вывести на экран.

При реализации программы используйте функцию filter и кое-что еще (для упрощения программы), подумайте что.

Вариант №9

Задание №1. На вход программы поступает список целых чисел, записанных в одну строчку через пробел. Необходимо выбрать из них четыре наибольших уникальных значения. Результат вывести на экран.

Вариант №10

Задание №1. Вводится список email-адресов в одну строчку через пробел. Среди них нужно оставить только корректно записанные адреса. Будем полагать, что к таким относятся те, что используют латинские буквы, цифры и символ подчеркивания. А также в адресе сначала должен идти символ "@", а затем, точка "." (между ними, конечно же, могут быть и другие символы).

Отобразить результат отбора корректных адресов на экране.

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Тексты программ.
3. Результаты работы программ.

Практическая работа 13. Создание программы по разработанному сценарию как отдельного модуля

Цель работы: научиться работать со встроенными функциями: map, filter, zip, all и any. А также выполнять сортировку по ключу методом sort и функцией sorted.

Теоретический материал

Теорию для выполнения лабораторной работы смотрите на странице tk.ulstu.ru/video.php#video в следующих видеоматериалах:

- #56. Функция map
- #57. Функция filter
- #58. Функция zip
- #59. Сортировка с помощью sort и sorted
- #60. Аргумент key для сортировки по ключу
- #61. Функция isinstance для проверки типов данных
- #62. Функции all и any

Задания на лабораторную работу

Вариант №1 **Задание №1.** На вход поступает список из вещественных чисел, записанных в строку через пробел. С помощью функции map преобразовать числа в строке в их вещественное представление и отобразить первые три числа. (Полагается, что минимум три вещественных числа имеются). Реализовать извлечение чисел через функцию next.
Вариант №2. **Задание №1.** Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию filter, которая бы возвращала только названия длиной более 5 символов. Извлеките первые три значения на выходе этой функции с помощью next и отобразите их на экране (Полагается, что минимум три значения имеются).
Вариант №3 **Задание №1.** Вводятся два списка целых чисел. Необходимо попарно перебрать их элементы и перемножить между собой. При реализации программы используйте функции zip и map. Выведите на экран первые три значения, используя функцию next. (Полагается, что три выходных значения всегда будут присутствовать).

Вариант №4

Задание №1. Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию map, которая бы возвращала названия городов только длиной более 5 символов. Вместо остальных названий - строку с дефисом ("-"). Сформировать список из полученных значений и отобразить его на экране.

Вариант №5

Задание №1. Вводится строка из слов, записанных через пробел. Необходимо на их основе составить прямоугольную таблицу из трех столбцов и N строк (число строк столько, сколько получится). Лишнее (выходящее) слово - отбросить. Реализовать эту программу с использованием функции zip. Результат отобразить на экране в виде прямоугольной таблицы из слов, записанных через пробел (в каждой строчке).

Вариант №6

Задание №1. Вводится список целых чисел в одну строчку через пробел. Необходимо оставить в нем только двузначные числа. Реализовать программу с использованием функции filter. Результат отобразить на экране в виде последовательности оставшихся чисел.

Вариант №7

Задание №1. На вход функции с именем get_sort поступает словарь, например, такой:

d = {'cat': 'кот', 'horse': 'лошадь', 'tree': 'дерево', 'dog': 'собака', 'book': 'книга'}

Необходимо отсортировать словарь d по убыванию ключей (лексикографическая сортировка строк) и возвратить список из соответствующих значений ключей словаря. Например, для указанного словаря d, результатом должен быть список:

['дерево', 'лошадь', 'собака', 'кот', 'книга']

Вызовите функцию get_sort для заданного в программе произвольного словаря dct и выведите возвращенный ею список на экран.

Вариант №8

Задание №1. Саша и Галя коллекционируют монетки. Каждый из них решил записать номиналы монеток из своей коллекции. Получилось два списка. Эти списки поступают на вход программы в виде двух строк из целых чисел, (числа записаны через пробел). Необходимо выделить значения, присутствующие в
обоих списках и оставить среди них только четные. Результат вывести на экран.

При реализации программы используйте функцию filter и кое-что еще (для упрощения программы), подумайте что.

Вариант №9

Задание №1. На вход программы поступает список целых чисел, записанных в одну строчку через пробел. Необходимо выбрать из них четыре наибольших уникальных значения. Результат вывести на экран.

Вариант №10

Задание №1. Вводится список email-адресов в одну строчку через пробел. Среди них нужно оставить только корректно записанные адреса. Будем полагать, что к таким относятся те, что используют латинские буквы, цифры и символ подчеркивания. А также в адресе сначала должен идти символ "@", а затем, точка "." (между ними, конечно же, могут быть и другие символы).

Отобразить результат отбора корректных адресов на экране.

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Тексты программ.
3. Результаты работы программ.

РАЗДЕЛ 6. МОДУЛЬНОЕ ТЕСТИРОВАНИЕ

Практическая работа 14. Разработка системы тестов. Тестирование на основе потока управления

Цель работы: научиться работать со встроенными функциями: map, filter, zip, all и any. А также выполнять сортировку по ключу методом sort и функцией sorted.

Теоретический материал

Теорию для выполнения лабораторной работы смотрите на странице tk.ulstu.ru/video.php#video в следующих видеоматериалах:

- #56. Функция map
- #57. Функция filter
- #58. Функция zip
- #59. Сортировка с помощью sort и sorted
- #60. Аргумент key для сортировки по ключу
- #61. Функция isinstance для проверки типов данных
- #62. Функции all и any

Задания на лабораторную работу

Вариант №1 **Задание №1.** На вход поступает список из вещественных чисел, записанных в строку через пробел. С помощью функции map преобразовать числа в строке в их вещественное представление и отобразить первые три числа. (Полагается, что минимум три вещественных числа имеются). Реализовать извлечение чисел через функцию next.
Вариант №2. **Задание №1.** Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию filter, которая бы возвращала только названия длиной более 5 символов. Извлеките первые три значения на выходе этой функции с помощью next и отобразите их на экране (Полагается, что минимум три значения имеются).
Вариант №3 **Задание №1.** Вводятся два списка целых чисел. Необходимо попарно перебрать их элементы и перемножить между собой. При реализации программы используйте функции zip и map. Выведите на экран первые три значения, используя функцию next. (Полагается, что три выходных значения всегда

будут присутствовать).

Вариант №4

Задание №1. Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию map, которая бы возвращала названия городов только длиной более 5 символов. Вместо остальных названий - строку с дефисом ("-"). Сформировать список из полученных значений и отобразить его на экране.

Вариант №5

Задание №1. Вводится строка из слов, записанных через пробел. Необходимо на их основе составить прямоугольную таблицу из трех столбцов и N строк (число строк столько, сколько получится). Лишнее (выходящее) слово - отбросить. Реализовать эту программу с использованием функции zip. Результат отобразить на экране в виде прямоугольной таблицы из слов, записанных через пробел (в каждой строчке).

Вариант №6

Задание №1. Вводится список целых чисел в одну строчку через пробел. Необходимо оставить в нем только двузначные числа. Реализовать программу с использованием функции filter. Результат отобразить на экране в виде последовательности оставшихся чисел.

Вариант №7

Задание №1. На вход функции с именем get_sort поступает словарь, например, такой:

d = {'cat': 'кот', 'horse': 'лошадь', 'tree': 'дерево', 'dog': 'собака', 'book': 'книга'}

Необходимо отсортировать словарь d по убыванию ключей (лексикографическая сортировка строк) и возвратить список из соответствующих значений ключей словаря. Например, для указанного словаря d, результатом должен быть список:

['дерево', 'лошадь', 'собака', 'кот', 'книга']

Вызовите функцию get_sort для заданного в программе произвольного словаря dct и выведите возвращенный ею список на экран.

Вариант №8

Задание №1. Саша и Галя коллекционируют монетки. Каждый из них решил записать номиналы монеток из своей коллекции. Получилось два списка. Эти списки поступают на вход программы в виде двух строк из целых чисел, (числа записаны через пробел). Необходимо выделить значения, присутствующие в
обоих списках и оставить среди них только четные. Результат вывести на экран.

При реализации программы используйте функцию filter и кое-что еще (для упрощения программы), подумайте что.

Вариант №9

Задание №1. На вход программы поступает список целых чисел, записанных в одну строчку через пробел. Необходимо выбрать из них четыре наибольших уникальных значения. Результат вывести на экран.

Вариант №10

Задание №1. Вводится список email-адресов в одну строчку через пробел. Среди них нужно оставить только корректно записанные адреса. Будем полагать, что к таким относятся те, что используют латинские буквы, цифры и символ подчеркивания. А также в адресе сначала должен идти символ "@", а затем, точка "." (между ними, конечно же, могут быть и другие символы).

Отобразить результат отбора корректных адресов на экране.

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Тексты программ.
3. Результаты работы программ.

Практическая работа 15. Тестирование на основе потока данных

Цель работы: научиться работать со встроенными функциями: map, filter, zip, all и any. А также выполнять сортировку по ключу методом sort и функцией sorted.

Теоретический материал

Теорию для выполнения лабораторной работы смотрите на странице tk.ulstu.ru/video.php#video в следующих видеоматериалах:

- #56. Функция map
- #57. Функция filter
- #58. Функция zip
- #59. Сортировка с помощью sort и sorted
- #60. Аргумент key для сортировки по ключу
- #61. Функция isinstance для проверки типов данных
- #62. Функции all и any

Задания на лабораторную работу

Вариант №1 **Задание №1.** На вход поступает список из вещественных чисел, записанных в строку через пробел. С помощью функции map преобразовать числа в строке в их вещественное представление и отобразить первые три числа. (Полагается, что минимум три вещественных числа имеются). Реализовать извлечение чисел через функцию next.
Вариант №2. **Задание №1.** Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию filter, которая бы возвращала только названия длиной более 5 символов. Извлеките первые три значения на выходе этой функции с помощью next и отобразите их на экране (Полагается, что минимум три значения имеются).
Вариант №3 **Задание №1.** Вводятся два списка целых чисел. Необходимо попарно перебрать их элементы и перемножить между собой. При реализации программы используйте функции zip и map. Выведите на экран первые три значения, используя функцию next. (Полагается, что три выходных значения всегда будут присутствовать).

Вариант №4

Задание №1. Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию map, которая бы возвращала названия городов только длиной более 5 символов. Вместо остальных названий - строку с дефисом ("-"). Сформировать список из полученных значений и отобразить его на экране.

Вариант №5

Задание №1. Вводится строка из слов, записанных через пробел. Необходимо на их основе составить прямоугольную таблицу из трех столбцов и N строк (число строк столько, сколько получится). Лишнее (выходящее) слово - отбросить. Реализовать эту программу с использованием функции zip. Результат отобразить на экране в виде прямоугольной таблицы из слов, записанных через пробел (в каждой строчке).

Вариант №6

Задание №1. Вводится список целых чисел в одну строчку через пробел. Необходимо оставить в нем только двузначные числа. Реализовать программу с использованием функции filter. Результат отобразить на экране в виде последовательности оставшихся чисел.

Вариант №7

Задание №1. На вход функции с именем get_sort поступает словарь, например, такой:

d = {'cat': 'кот', 'horse': 'лошадь', 'tree': 'дерево', 'dog': 'собака', 'book': 'книга'}

Необходимо отсортировать словарь d по убыванию ключей (лексикографическая сортировка строк) и возвратить список из соответствующих значений ключей словаря. Например, для указанного словаря d, результатом должен быть список:

['дерево', 'лошадь', 'собака', 'кот', 'книга']

Вызовите функцию get_sort для заданного в программе произвольного словаря dct и выведите возвращенный ею список на экран.

Вариант №8

Задание №1. Саша и Галя коллекционируют монетки. Каждый из них решил записать номиналы монеток из своей коллекции. Получилось два списка. Эти списки поступают на вход программы в виде двух строк из целых чисел, (числа записаны через пробел). Необходимо выделить значения, присутствующие в
обоих списках и оставить среди них только четные. Результат вывести на экран.

При реализации программы используйте функцию filter и кое-что еще (для упрощения программы), подумайте что.

Вариант №9

Задание №1. На вход программы поступает список целых чисел, записанных в одну строчку через пробел. Необходимо выбрать из них четыре наибольших уникальных значения. Результат вывести на экран.

Вариант №10

Задание №1. Вводится список email-адресов в одну строчку через пробел. Среди них нужно оставить только корректно записанные адреса. Будем полагать, что к таким относятся те, что используют латинские буквы, цифры и символ подчеркивания. А также в адресе сначала должен идти символ "@", а затем, точка "." (между ними, конечно же, могут быть и другие символы).

Отобразить результат отбора корректных адресов на экране.

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Тексты программ.
3. Результаты работы программ.

РАЗДЕЛ 7. ДОКУМЕНТИРОВАНИЕ

Практическая работа 16. Тестирование программного модуля по определенному сценарию

Цель работы: научиться работать со встроенными функциями: map, filter, zip, all и any. А также выполнять сортировку по ключу методом sort и функцией sorted.

Теоретический материал

Теорию для выполнения лабораторной работы смотрите на странице tk.ulstu.ru/video.php#video в следующих видеоматериалах:

- #56. Функция map
- #57. Функция filter
- #58. Функция zip
- #59. Сортировка с помощью sort и sorted
- #60. Аргумент key для сортировки по ключу
- #61. Функция isinstance для проверки типов данных
- #62. Функции all и any

Задания на лабораторную работу

Вариант №1 **Задание №1.** На вход поступает список из вещественных чисел, записанных в строку через пробел. С помощью функции map преобразовать числа в строке в их вещественное представление и отобразить первые три числа. (Полагается, что минимум три вещественных числа имеются). Реализовать извлечение чисел через функцию next.
Вариант №2. **Задание №1.** Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию filter, которая бы возвращала только названия длиной более 5 символов. Извлеките первые три значения на выходе этой функции с помощью next и отобразите их на экране (Полагается, что минимум три значения имеются).
Вариант №3 **Задание №1.** Вводятся два списка целых чисел. Необходимо попарно перебрать их элементы и перемножить между собой. При реализации программы используйте функции zip и map. Выведите на экран первые три значения, используя функцию next. (Полагается, что три выходных значения всегда

будут присутствовать).

Вариант №4

Задание №1. Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию map, которая бы возвращала названия городов только длиной более 5 символов. Вместо остальных названий - строку с дефисом ("-"). Сформировать список из полученных значений и отобразить его на экране.

Вариант №5

Задание №1. Вводится строка из слов, записанных через пробел. Необходимо на их основе составить прямоугольную таблицу из трех столбцов и N строк (число строк столько, сколько получится). Лишнее (выходящее) слово - отбросить. Реализовать эту программу с использованием функции zip. Результат отобразить на экране в виде прямоугольной таблицы из слов, записанных через пробел (в каждой строчке).

Вариант №6

Задание №1. Вводится список целых чисел в одну строчку через пробел. Необходимо оставить в нем только двузначные числа. Реализовать программу с использованием функции filter. Результат отобразить на экране в виде последовательности оставшихся чисел.

Вариант №7

Задание №1. На вход функции с именем get_sort поступает словарь, например, такой:

d = {'cat': 'кот', 'horse': 'лошадь', 'tree': 'дерево', 'dog': 'собака', 'book': 'книга'}

Необходимо отсортировать словарь d по убыванию ключей (лексикографическая сортировка строк) и возвратить список из соответствующих значений ключей словаря. Например, для указанного словаря d, результатом должен быть список:

['дерево', 'лошадь', 'собака', 'кот', 'книга']

Вызовите функцию get_sort для заданного в программе произвольного словаря dct и выведите возвращенный ею список на экран.

Вариант №8

Задание №1. Саша и Галя коллекционируют монетки. Каждый из них решил записать номиналы монеток из своей коллекции. Получилось два списка. Эти списки поступают на вход программы в виде двух строк из целых чисел, (числа записаны через пробел). Необходимо выделить значения, присутствующие в
обоих списках и оставить среди них только четные. Результат вывести на экран.

При реализации программы используйте функцию filter и кое-что еще (для упрощения программы), подумайте что.

Вариант №9

Задание №1. На вход программы поступает список целых чисел, записанных в одну строчку через пробел. Необходимо выбрать из них четыре наибольших уникальных значения. Результат вывести на экран.

Вариант №10

Задание №1. Вводится список email-адресов в одну строчку через пробел. Среди них нужно оставить только корректно записанные адреса. Будем полагать, что к таким относятся те, что используют латинские буквы, цифры и символ подчеркивания. А также в адресе сначала должен идти символ "@", а затем, точка "." (между ними, конечно же, могут быть и другие символы).

Отобразить результат отбора корректных адресов на экране.

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Тексты программ.
3. Результаты работы программ.

Практическая работа 17. Оформление документации на программные средства с использованием инструментальных средств

Цель работы: научиться работать со встроенными функциями: map, filter, zip, all и any. А также выполнять сортировку по ключу методом sort и функцией sorted.

Теоретический материал

Теорию для выполнения лабораторной работы смотрите на странице tk.ulstu.ru/video.php#video в следующих видеоматериалах:

- #56. Функция map
- #57. Функция filter
- #58. Функция zip
- #59. Сортировка с помощью sort и sorted
- #60. Аргумент key для сортировки по ключу
- #61. Функция isinstance для проверки типов данных
- #62. Функции all и any

Задания на лабораторную работу

Вариант №1 **Задание №1.** На вход поступает список из вещественных чисел, записанных в строку через пробел. С помощью функции map преобразовать числа в строке в их вещественное представление и отобразить первые три числа. (Полагается, что минимум три вещественных числа имеются). Реализовать извлечение чисел через функцию next.
Вариант №2. **Задание №1.** Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию filter, которая бы возвращала только названия длиной более 5 символов. Извлеките первые три значения на выходе этой функции с помощью next и отобразите их на экране (Полагается, что минимум три значения имеются).
Вариант №3 **Задание №1.** Вводятся два списка целых чисел. Необходимо попарно перебрать их элементы и перемножить между собой. При реализации программы используйте функции zip и map. Выведите на экран первые три значения, используя функцию next. (Полагается, что три выходных значения всегда будут присутствовать).

Вариант №4

Задание №1. Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию map, которая бы возвращала названия городов только длиной более 5 символов. Вместо остальных названий - строку с дефисом ("-"). Сформировать список из полученных значений и отобразить его на экране.

Вариант №5

Задание №1. Вводится строка из слов, записанных через пробел. Необходимо на их основе составить прямоугольную таблицу из трех столбцов и N строк (число строк столько, сколько получится). Лишнее (выходящее) слово - отбросить. Реализовать эту программу с использованием функции zip. Результат отобразить на экране в виде прямоугольной таблицы из слов, записанных через пробел (в каждой строчке).

Вариант №6

Задание №1. Вводится список целых чисел в одну строчку через пробел. Необходимо оставить в нем только двузначные числа. Реализовать программу с использованием функции filter. Результат отобразить на экране в виде последовательности оставшихся чисел.

Вариант №7

Задание №1. На вход функции с именем get_sort поступает словарь, например, такой:

d = {'cat': 'кот', 'horse': 'лошадь', 'tree': 'дерево', 'dog': 'собака', 'book': 'книга'}

Необходимо отсортировать словарь d по убыванию ключей (лексикографическая сортировка строк) и возвратить список из соответствующих значений ключей словаря. Например, для указанного словаря d, результатом должен быть список:

['дерево', 'лошадь', 'собака', 'кот', 'книга']

Вызовите функцию get_sort для заданного в программе произвольного словаря dct и выведите возвращенный ею список на экран.

Вариант №8

Задание №1. Саша и Галя коллекционируют монетки. Каждый из них решил записать номиналы монеток из своей коллекции. Получилось два списка. Эти списки поступают на вход программы в виде двух строк из целых чисел, (числа записаны через пробел). Необходимо выделить значения, присутствующие в
обоих списках и оставить среди них только четные. Результат вывести на экран.

При реализации программы используйте функцию filter и кое-что еще (для упрощения программы), подумайте что.

Вариант №9

Задание №1. На вход программы поступает список целых чисел, записанных в одну строчку через пробел. Необходимо выбрать из них четыре наибольших уникальных значения. Результат вывести на экран.

Вариант №10

Задание №1. Вводится список email-адресов в одну строчку через пробел. Среди них нужно оставить только корректно записанные адреса. Будем полагать, что к таким относятся те, что используют латинские буквы, цифры и символ подчеркивания. А также в адресе сначала должен идти символ "@", а затем, точка "." (между ними, конечно же, могут быть и другие символы).

Отобразить результат отбора корректных адресов на экране.

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Тексты программ.
3. Результаты работы программ.

Практическая работа 18. Моделирование потоков данных

Цель работы: научиться работать со встроенными функциями: map, filter, zip, all и any. А также выполнять сортировку по ключу методом sort и функцией sorted.

Теоретический материал

Теорию для выполнения лабораторной работы смотрите на странице tk.ulstu.ru/video.php#video в следующих видеоматериалах:

- #56. Функция map
- #57. Функция filter
- #58. Функция zip
- #59. Сортировка с помощью sort и sorted
- #60. Аргумент key для сортировки по ключу
- #61. Функция isinstance для проверки типов данных
- #62. Функции all и any

Задания на лабораторную работу

Вариант №1 **Задание №1.** На вход поступает список из вещественных чисел, записанных в строку через пробел. С помощью функции map преобразовать числа в строке в их вещественное представление и отобразить первые три числа. (Полагается, что минимум три вещественных числа имеются). Реализовать извлечение чисел через функцию next.
Вариант №2. **Задание №1.** Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию filter, которая бы возвращала только названия длиной более 5 символов. Извлеките первые три значения на выходе этой функции с помощью next и отобразите их на экране (Полагается, что минимум три значения имеются).
Вариант №3 **Задание №1.** Вводятся два списка целых чисел. Необходимо попарно перебрать их элементы и перемножить между собой. При реализации программы используйте функции zip и map. Выведите на экран первые три значения, используя функцию next. (Полагается, что три выходных значения всегда будут присутствовать).

Вариант №4

Задание №1. Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию map, которая бы возвращала названия городов только длиной более 5 символов. Вместо остальных названий - строку с дефисом ("-"). Сформировать список из полученных значений и отобразить его на экране.

Вариант №5

Задание №1. Вводится строка из слов, записанных через пробел. Необходимо на их основе составить прямоугольную таблицу из трех столбцов и N строк (число строк столько, сколько получится). Лишнее (выходящее) слово - отбросить. Реализовать эту программу с использованием функции zip. Результат отобразить на экране в виде прямоугольной таблицы из слов, записанных через пробел (в каждой строчке).

Вариант №6

Задание №1. Вводится список целых чисел в одну строчку через пробел. Необходимо оставить в нем только двузначные числа. Реализовать программу с использованием функции filter. Результат отобразить на экране в виде последовательности оставшихся чисел.

Вариант №7

Задание №1. На вход функции с именем get_sort поступает словарь, например, такой:

d = {'cat': 'кот', 'horse': 'лошадь', 'tree': 'дерево', 'dog': 'собака', 'book': 'книга'}

Необходимо отсортировать словарь d по убыванию ключей (лексикографическая сортировка строк) и возвратить список из соответствующих значений ключей словаря. Например, для указанного словаря d, результатом должен быть список:

['дерево', 'лошадь', 'собака', 'кот', 'книга']

Вызовите функцию get_sort для заданного в программе произвольного словаря dct и выведите возвращенный ею список на экран.

Вариант №8

Задание №1. Саша и Галя коллекционируют монетки. Каждый из них решил записать номиналы монеток из своей коллекции. Получилось два списка. Эти списки поступают на вход программы в виде двух строк из целых чисел, (числа записаны через пробел). Необходимо выделить значения, присутствующие в
обоих списках и оставить среди них только четные. Результат вывести на экран.

При реализации программы используйте функцию filter и кое-что еще (для упрощения программы), подумайте что.

Вариант №9

Задание №1. На вход программы поступает список целых чисел, записанных в одну строчку через пробел. Необходимо выбрать из них четыре наибольших уникальных значения. Результат вывести на экран.

Вариант №10

Задание №1. Вводится список email-адресов в одну строчку через пробел. Среди них нужно оставить только корректно записанные адреса. Будем полагать, что к таким относятся те, что используют латинские буквы, цифры и символ подчеркивания. А также в адресе сначала должен идти символ "@", а затем, точка "." (между ними, конечно же, могут быть и другие символы).

Отобразить результат отбора корректных адресов на экране.

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Тексты программ.
3. Результаты работы программ.

Практическая работа 19. Документирование программного обеспечения

Цель работы: научиться работать со встроенными функциями: map, filter, zip, all и any. А также выполнять сортировку по ключу методом sort и функцией sorted.

Теоретический материал

Теорию для выполнения лабораторной работы смотрите на странице tk.ulstu.ru/video.php#video в следующих видеоматериалах:

- #56. Функция map
- #57. Функция filter
- #58. Функция zip
- #59. Сортировка с помощью sort и sorted
- #60. Аргумент key для сортировки по ключу
- #61. Функция isinstance для проверки типов данных
- #62. Функции all и any

Задания на лабораторную работу

Вариант №1 **Задание №1.** На вход поступает список из вещественных чисел, записанных в строку через пробел. С помощью функции map преобразовать числа в строке в их вещественное представление и отобразить первые три числа. (Полагается, что минимум три вещественных числа имеются). Реализовать извлечение чисел через функцию next.
Вариант №2. **Задание №1.** Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию filter, которая бы возвращала только названия длиной более 5 символов. Извлеките первые три значения на выходе этой функции с помощью next и отобразите их на экране (Полагается, что минимум три значения имеются).
Вариант №3 **Задание №1.** Вводятся два списка целых чисел. Необходимо попарно перебрать их элементы и перемножить между собой. При реализации программы используйте функции zip и map. Выведите на экран первые три значения, используя функцию next. (Полагается, что три выходных значения всегда будут присутствовать).

Вариант №4

Задание №1. Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию map, которая бы возвращала названия городов только длиной более 5 символов. Вместо остальных названий - строку с дефисом ("-"). Сформировать список из полученных значений и отобразить его на экране.

Вариант №5

Задание №1. Вводится строка из слов, записанных через пробел. Необходимо на их основе составить прямоугольную таблицу из трех столбцов и N строк (число строк столько, сколько получится). Лишнее (выходящее) слово - отбросить. Реализовать эту программу с использованием функции zip. Результат отобразить на экране в виде прямоугольной таблицы из слов, записанных через пробел (в каждой строчке).

Вариант №6

Задание №1. Вводится список целых чисел в одну строчку через пробел. Необходимо оставить в нем только двузначные числа. Реализовать программу с использованием функции filter. Результат отобразить на экране в виде последовательности оставшихся чисел.

Вариант №7

Задание №1. На вход функции с именем get_sort поступает словарь, например, такой:

d = {'cat': 'кот', 'horse': 'лошадь', 'tree': 'дерево', 'dog': 'собака', 'book': 'книга'}

Необходимо отсортировать словарь d по убыванию ключей (лексикографическая сортировка строк) и возвратить список из соответствующих значений ключей словаря. Например, для указанного словаря d, результатом должен быть список:

['дерево', 'лошадь', 'собака', 'кот', 'книга']

Вызовите функцию get_sort для заданного в программе произвольного словаря dct и выведите возвращенный ею список на экран.

Вариант №8

Задание №1. Саша и Галя коллекционируют монетки. Каждый из них решил записать номиналы монеток из своей коллекции. Получилось два списка. Эти списки поступают на вход программы в виде двух строк из целых чисел, (числа записаны через пробел). Необходимо выделить значения, присутствующие в
обоих списках и оставить среди них только четные. Результат вывести на экран.

При реализации программы используйте функцию filter и кое-что еще (для упрощения программы), подумайте что.

Вариант №9

Задание №1. На вход программы поступает список целых чисел, записанных в одну строчку через пробел. Необходимо выбрать из них четыре наибольших уникальных значения. Результат вывести на экран.

Вариант №10

Задание №1. Вводится список email-адресов в одну строчку через пробел. Среди них нужно оставить только корректно записанные адреса. Будем полагать, что к таким относятся те, что используют латинские буквы, цифры и символ подчеркивания. А также в адресе сначала должен идти символ "@", а затем, точка "." (между ними, конечно же, могут быть и другие символы).

Отобразить результат отбора корректных адресов на экране.

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Тексты программ.
3. Результаты работы программ.

РАЗДЕЛ 8. СРЕДСТВА РАЗРАБОТКИ ТЕХНИЧЕСКОЙ ДОКУМЕНТАЦИИ

Практическая работа 20. Создание приложения с использованием диалоговых окон

Цель работы: научиться работать со встроенными функциями: map, filter, zip, all и any. А также выполнять сортировку по ключу методом sort и функцией sorted.

Теоретический материал

Теорию для выполнения лабораторной работы смотрите на странице tk.ulstu.ru/video.php#video в следующих видеоматериалах:

- #56. Функция map
- #57. Функция filter
- #58. Функция zip
- #59. Сортировка с помощью sort и sorted
- #60. Аргумент key для сортировки по ключу
- #61. Функция isinstance для проверки типов данных
- #62. Функции all и any

Задания на лабораторную работу

Вариант №1 **Задание №1.** На вход поступает список из вещественных чисел, записанных в строку через пробел. С помощью функции map преобразовать числа в строке в их вещественное представление и отобразить первые три числа. (Полагается, что минимум три вещественных числа имеются). Реализовать извлечение чисел через функцию next.
Вариант №2. **Задание №1.** Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию filter, которая бы возвращала только названия длиной более 5 символов. Извлеките первые три значения на выходе этой функции с помощью next и отобразите их на экране (Полагается, что минимум три значения имеются).

Вариант №3

Задание №1. Вводятся два списка целых чисел. Необходимо попарно перебрать
их элементы и перемножить между собой. При реализации программы используйте функции zip и map. Выведите на экран первые три значения, используя функцию next. (Полагается, что три выходных значения всегда будут присутствовать).

Вариант №4

Задание №1. Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию map, которая бы возвращала названия городов только длиной более 5 символов. Вместо остальных названий - строку с дефисом ("-"). Сформировать список из полученных значений и отобразить его на экране.

Вариант №5

Задание №1. Вводится строка из слов, записанных через пробел. Необходимо на их основе составить прямоугольную таблицу из трех столбцов и N строк (число строк столько, сколько получится). Лишнее (выходящее) слово - отбросить. Реализовать эту программу с использованием функции zip. Результат отобразить на экране в виде прямоугольной таблицы из слов, записанных через пробел (в каждой строчке).

Вариант №6

Задание №1. Вводится список целых чисел в одну строчку через пробел. Необходимо оставить в нем только двузначные числа. Реализовать программу с использованием функции filter. Результат отобразить на экране в виде последовательности оставшихся чисел.

Вариант №7

Задание №1. На вход функции с именем get_sort поступает словарь, например, такой:

d = {'cat': 'кот', 'horse': 'лошадь', 'tree': 'дерево', 'dog': 'собака', 'book': 'книга'}

Необходимо отсортировать словарь d по убыванию ключей (лексикографическая сортировка строк) и возвратить список из соответствующих значений ключей словаря. Например, для указанного словаря d, результатом должен быть список:

['дерево', 'лошадь', 'собака', 'кот', 'книга']

Вызовите функцию get_sort для заданного в программе произвольного словаря dct и выведите возвращенный ею список на экран.

Вариант №8

Задание №1. Саша и Галя коллекционируют монетки. Каждый из них решил записать номиналы монеток из своей коллекции. Получилось два списка. Эти списки поступают на вход программы в виде двух строк из целых чисел, (числа записаны через пробел). Необходимо выделить значения, присутствующие в
обоих списках и оставить среди них только четные. Результат вывести на экран.

При реализации программы используйте функцию filter и кое-что еще (для упрощения программы), подумайте что.

Вариант №9

Задание №1. На вход программы поступает список целых чисел, записанных в одну строчку через пробел. Необходимо выбрать из них четыре наибольших уникальных значения. Результат вывести на экран.

Вариант №10

Задание №1. Вводится список email-адресов в одну строчку через пробел. Среди них нужно оставить только корректно записанные адреса. Будем полагать, что к таким относятся те, что используют латинские буквы, цифры и символ подчеркивания. А также в адресе сначала должен идти символ "@", а затем, точка "." (между ними, конечно же, могут быть и другие символы).

Отобразить результат отбора корректных адресов на экране.

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Тексты программ.
3. Результаты работы программ.

Практическая работа 21. Создание приложений с использованием различных графических компонентов

Цель работы: научиться работать со встроенными функциями: map, filter, zip, all и any. А также выполнять сортировку по ключу методом sort и функцией sorted.

Теоретический материал

Теорию для выполнения лабораторной работы смотрите на странице tk.ulstu.ru/video.php#video в следующих видеоматериалах:

- #56. Функция map
- #57. Функция filter
- #58. Функция zip
- #59. Сортировка с помощью sort и sorted
- #60. Аргумент key для сортировки по ключу
- #61. Функция isinstance для проверки типов данных
- #62. Функции all и any

Задания на лабораторную работу

Вариант №1 **Задание №1.** На вход поступает список из вещественных чисел, записанных в строку через пробел. С помощью функции map преобразовать числа в строке в их вещественное представление и отобразить первые три числа. (Полагается, что минимум три вещественных числа имеются). Реализовать извлечение чисел через функцию next.
Вариант №2. **Задание №1.** Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию filter, которая бы возвращала только названия длиной более 5 символов. Извлеките первые три значения на выходе этой функции с помощью next и отобразите их на экране (Полагается, что минимум три значения имеются).
Вариант №3 **Задание №1.** Вводятся два списка целых чисел. Необходимо попарно перебрать их элементы и перемножить между собой. При реализации программы используйте функции zip и map. Выведите на экран первые три значения, используя функцию next. (Полагается, что три выходных значения всегда будут присутствовать).

Вариант №4

Задание №1. Вводятся названия городов в одну строчку через пробел. Необходимо определить функцию map, которая бы возвращала названия городов только длиной более 5 символов. Вместо остальных названий - строку с дефисом ("-"). Сформировать список из полученных значений и отобразить его на экране.

Вариант №5

Задание №1. Вводится строка из слов, записанных через пробел. Необходимо на их основе составить прямоугольную таблицу из трех столбцов и N строк (число строк столько, сколько получится). Лишнее (выходящее) слово - отбросить. Реализовать эту программу с использованием функции zip. Результат отобразить на экране в виде прямоугольной таблицы из слов, записанных через пробел (в каждой строчке).

Вариант №6

Задание №1. Вводится список целых чисел в одну строчку через пробел. Необходимо оставить в нем только двузначные числа. Реализовать программу с использованием функции filter. Результат отобразить на экране в виде последовательности оставшихся чисел.

Вариант №7

Задание №1. На вход функции с именем get_sort поступает словарь, например, такой:

d = {'cat': 'кот', 'horse': 'лошадь', 'tree': 'дерево', 'dog': 'собака', 'book': 'книга'}

Необходимо отсортировать словарь d по убыванию ключей (лексикографическая сортировка строк) и возвратить список из соответствующих значений ключей словаря. Например, для указанного словаря d, результатом должен быть список:

['дерево', 'лошадь', 'собака', 'кот', 'книга']

Вызовите функцию get_sort для заданного в программе произвольного словаря dct и выведите возвращенный ею список на экран.

Вариант №8

Задание №1. Саша и Галя коллекционируют монетки. Каждый из них решил записать номиналы монеток из своей коллекции. Получилось два списка. Эти списки поступают на вход программы в виде двух строк из целых чисел, (числа записаны через пробел). Необходимо выделить значения, присутствующие в
обоих списках и оставить среди них только четные. Результат вывести на экран.

При реализации программы используйте функцию filter и кое-что еще (для упрощения программы), подумайте что.

Вариант №9

Задание №1. На вход программы поступает список целых чисел, записанных в одну строчку через пробел. Необходимо выбрать из них четыре наибольших уникальных значения. Результат вывести на экран.

Вариант №10

Задание №1. Вводится список email-адресов в одну строчку через пробел. Среди них нужно оставить только корректно записанные адреса. Будем полагать, что к таким относятся те, что используют латинские буквы, цифры и символ подчеркивания. А также в адресе сначала должен идти символ "@", а затем, точка "." (между ними, конечно же, могут быть и другие символы).

Отобразить результат отбора корректных адресов на экране.

Содержание отчета

1. Титульный лист с названием лабораторной работы, номером своего варианта, фамилией студента и группы.
2. Тексты программ.
3. Результаты работы программ.

ТЕСТОВЫЕ ЗАДАНИЯ

1. Сұрақ (Вопрос)
К внешним факторам, влияющим на качество ПО относятся
A. корректность
B. модульность
C. расширяемость
D. язык разработки
E. нет верного ответа

2. Сұрақ (Вопрос)
Под расширяемостью ПО понимается?
A. возможность добавления новых функций
B. возможность сборки ПО из готовых компонентов
C. легкость адаптации ПО к изменениям спецификации
D. автоматическое развертывание кода модулей
E. нет верного ответа

3. Сұрақ (Вопрос)
Гибридный язык
A. язык, в основе которого лежат идеи нескольких языков
B. необъектный язык, дополненный ОО-механизмами
C. универсальный язык, обладающий преимуществами старых и новых языков
D. результат неудачного языкотворчества
E. нет верного ответа

4. Сұрақ (Вопрос)
Исключение
A. приводит к прерыванию нормального процесса вычислений
B. может служить причиной отказа
C. может привести в конечном счете к успеху
D. возникает всегда, когда в программе есть ошибка
E. нет верного ответа

5. Сұрақ (Вопрос)
Проектирование интерфейса пользователя
A. должно выполняться на самых ранних этапах ОО-проектирования

B. должно выполняться на поздних этапах ОО-проектирования
C. в процессе проектирования каждого класса
D. после завершения проектирования основных классов
E. нет верного ответа

6. Сұрақ (Вопрос)

Метод проектирования, который можно называть “Модульным”, должен удовлетворять основным требованиям:

A. декомпозиции, композиции
B. понятности
C. непрерывности
D. защищенности
E. все ответы верны

7. Сұрақ (Вопрос)

Тестирование представляет собой:

A. процесс выполнения программы для некоторого набора проверочных значений и сравнения полученных результатов с ожидаемыми
B. процесс понимания и исправления ошибок
C. процесс компиляции программы с целью исправления синтаксических и семантических ошибок
D. ответы 1 и 2
E. нет верного ответа

8. Сұрақ (Вопрос)

Расширяемость

A. это способность ПО соответствующим образом реагировать на аварийные ситуации
B. это способность ПО выполнять точные задачи так, как они определены их спецификацией
C. это легкость адаптации ПО к изменениям спецификации
D. это легкость сочетания одних элементов ПО с другими
E. нет верного ответа

9. Сұрақ (Вопрос)

Совместимость

A. это способность ПО соответствующим образом реагировать на аварийные ситуации

B. это способность ПО выполнять точные задачи так, как они определены их спецификацией
C. это легкость адаптации ПО к изменениям спецификации
D. это легкость сочетания одних элементов ПО с другими
E. нет верного ответа

10. Сұрақ (Вопрос)

Эффективность

A. это способность ПО как можно меньше зависеть от ресурсов оборудования: процессорного времени, пространства, занимаемого во внутренней и внешней памяти, пропускной способности, используемой в устройствах связи.
B. это способность ПО соответствующим образом реагировать на аварийные ситуации.
C. это способность ПО выполнять точные задачи так, как они определены их спецификацией.
D. это легкость адаптации ПО к изменениям спецификации.
E. это легкость сочетания одних элементов ПО с другими.

11. Сұрақ (Вопрос)

Переносимость

A. это легкость переноса ПО в различные программные и аппаратные среды.
B. это способность ПО соответствующим образом реагировать на аварийные ситуации.
C. это способность ПО выполнять точные задачи так, как они определены их спецификацией.
D. это легкость адаптации ПО к изменениям спецификации.
E. это легкость сочетания одних элементов ПО с другими.

12. Сұрақ (Вопрос)

Простота использования

A. это легкость адаптации ПО к изменениям спецификации.
B. это легкость сочетания одних элементов ПО с другими.
C. это легкость переноса ПО в различные программные и аппаратные среды.
D. это легкость, с которой люди с различными знаниями и квалификацией могут научиться использовать ПО и применять его для решения задач. Сюда также относится простота установки, работы и текущего контроля.

E. нет верного ответа

13. Сұрақ (Вопрос)

Функциональность

A. это степень возможностей, обеспечиваемых системой.
B. это способность ПО как можно меньше зависеть от ресурсов оборудования: процессорного времени, пространства, занимаемого во внутренней и внешней памяти, пропускной способности, используемой в устройствах связи.
C. это способность ПО соответствующим образом реагировать на аварийные ситуации.
D. это способность ПО выполнять точные задачи так, как они определены их спецификацией.
E. нет верного ответа

14. Сұрақ (Вопрос)

Алгоритм — это:

A. указание на выполнение действий
B. процесс выполнения вычислений, приводящих к решению задачи
C. система правил, описывающая последовательность действий, которые необходимо выполнить для решения задачи
D. все ответы правильные
E. нет правильного ответа

15. Сұрақ (Вопрос)

Свойствами алгоритма являются

A. информативность
B. массовость
C. оперативность
D. определенность
E. дискретность
F. цикличность
G. результативность

16. Сұрақ (Вопрос)

Алгоритм может быть задан следующими способами:

A. словесным
B. на алгоритмическом языке
C. графическим

D. формально-словесным
E. словесно-графическим
F. последовательностью байтов

17. Сұрақ (Вопрос)
Программа — это:
A. система правил, описывающая последовательность действий, которые необходимо выполнить для решения задачи
B. указание на выполнение действий из заданного набора
C. область внешней памяти для хранения текстовых, числовых данных и другой информации
D. последовательность команд, реализующая алгоритм решения задач
E. нет правильного ответа

18. Сұрақ (Вопрос)
Программа-интерпретатор выполняет:
A. поиск файлов на диске
B. пооператорное выполнение программы
C. полное выполнение программы
D. нет правильного ответа
E. все ответы правильные

19. Сұрақ (Вопрос)
Программа-компилятор выполняет:
A. переводит исходный текст в машинный код
B. записывает машинный код в форме загрузочного файла
C. формирует текстовый файл
D. нет правильного ответа
E. все ответы правильные

20. Сұрақ (Вопрос)
Числовые данные могут быть представлены как:
A. целые
B. с фиксированной запятой
C. в виде строк
D. с плавающей запятой
E. нет правильного ответа

21. Сұрақ (Вопрос)

Имя переменной — это:

A. любая последовательность любых символов
B. последовательность латинских букв, цифр, специальных знаков (кроме пробел)
C. последовательность любых символов, которая всегда должна начинаться с латинской буквы
D. последовательность русских, латинских букв, начинающихся с латинской буквы и из специальных знаков, допускающая знак подчеркивания
E. нет правильного ответа

22. Сұрақ (Вопрос)

Основными представлениями моделей являются:

A. текстовое описание
B. словесное описание
C. физическое описание
D. нет правильного ответа
E. все ответы правильные

23. Сұрақ (Вопрос)

Как называется первый этап процесса решения задачи с использованием готового ПО:

A. построение модели
B. постановка задачи
C. выбор готового ПО
D. нет правильного ответа
E. все ответы правильные

24. Сұрақ (Вопрос)

Какому критерию свойств алгоритмов относится решение целого класса однотипных задач?

A. конечность
B. однозначность
C. правильность
D. массовость
E. нет правильного ответа

25. Сұрақ (Вопрос)

Каким способом можно объявлять переменные в Python:

A. a=5
B. a=int (E.
C. int a=5
D. все ответы правильные
E. все ответы правильные

26. Сұрақ (Вопрос)
Какая функция отвечает за вывод на экран?
A. cout<<a
B. out (a)
C. print (a)
D. нет правильного ответа
E. все ответы правильные

27. Сұрақ (Вопрос)
Какая функция отвечает за открытие файла?
A. file()
B. open()
C. open_file()
D. нет правильного ответа
E. все ответы правильные

28. Сұрақ (Вопрос)
В каком из вариантов присутствует ошибка?
A. a=5
print ('a')
B. while True
print(a)
C. a=open("file.txt")
D. нет правильного ответа
E. все ответы правильные

29. Сұрақ (Вопрос)
Что делает команда import
A. импортирует файл модуля
B. создает функцию
C. удаляет файл
D. команда не существует

E. нет правильного ответа

30. Сұрақ (Вопрос)
Выберите вариант правильного удаления переменной a
A. del(a)
B. delete(a)
C. delete=a
D. нет правильного ответа
E. все ответы правильные

31. Сұрақ (Вопрос)
Какое значение 1//2 вернет выражение в среде IDLE?
A. 0
B. 0.5
C. 0.50
D. 0.05
E. нет правильного ответа

32. Сұрақ (Вопрос)
Как называется встроенный в языке Python тип данных неупорядоченной коллекции из нуля или более пар ключ-значение?
A. dict
B. set
C. list
D. frozenset
E. нет правильного ответа

33. Сұрақ (Вопрос)
Если предположить, что класс Mydict наследует класс dict, то каким класс dict является по отношению к классу Mydict?
A. дочерним
B. подклассом
C. базовым
D. родительским
E. нет правильного ответа

34. Сұрақ (Вопрос)
Что выведет данный код?
a = {1, 2}

```
b = {1}
b.union(a)
print(b)
```

A. {1}
B. {1, 2}
C. ошибку
D. ничего
E. нет правильного ответа

35. Сұрақ (Вопрос)

Что выведет данный код?

```
a = tuple({1, 4, 1})
print(a)
```

A. (1, 4, A.
B. (1, D.
C. [1, 4]
D. ошибку
E. нет правильного ответа

36. Сұрақ (Вопрос)

Что выведет данный код?

```
a = {1, 2, 1,}
b = a.copy()
print(a == b)
```

A. True
B. False
C. ошибку
D. ничего
E. нет правильного ответа

37. Сұрақ (Вопрос)

Что выведет данный код?

```
a = {}
b = a.add("py")
print(b)
```

A. {'py'}
B. {'p', 'y'}
C. ошибку
D. ничего

E. нет правильного ответа

38. Сұрақ (Вопрос)
Что выведет данный код?
```
a = set('qwerty')
b = frozenset('qwerty')
print(a == b)
```
A. True
B. False
C. ошибку
D. ничего
E. нет правильного ответа

39. Сұрақ (Вопрос)
Что выведет данный код?
```
a = set({})
b = set({})
print(type(a | b))
```
A. True
B. False
C. class 'set'
D. ошибку
E. нет правильного ответа

40. Сұрақ (Вопрос)
Что выведет данный код?
```
a = {1, 2, 5, 1.0}
a.discard(A.
print(a)
```
A. {2, 5}
B. {1, 2, 5}
C. {1.0, 2, 5}
D. ошибку
E. нет правильного ответа

41. Сұрақ (Вопрос)
Что выведет данный код?
```
a = {1, 2, 5, 1.0}
a = a.discard(A.
print(a)
```

A. {2, 5}
B. (1.0, A.
C. None
D. ошибку
E. нет правильного ответа

42. Сұрақ (Вопрос)
Что выведет данный код?
a = {1, 3}
b = {1.0}
d.union(a, a)
print(a)
A. {1.0, 3}
B. {1, 3}
C. ошибку
D. ничего
E. нет правильного ответа

43. Сұрақ (Вопрос)
Что выведет данный код?
a = set(1, B.
b = a.copy()
print(a == b)
A. {1.0, 3}
B. {1, 3}
C. ошибку
D. ничего
E. нет правильного ответа

44. Сұрақ (Вопрос)
Что выведет данный код?
a = list(1 + C.
print(a)
A. 4
B. [1, 3]
C. [4]
D. ошибку
E. нет правильного ответа

45. Сұрақ (Вопрос)

Что выведет данный код?

```
b = [1] + [2]
print(b)
```

A. [3]

B. [12]

C. [1, 2]

D. ошибку

E. нет правильного ответа

46. Сұрақ (Вопрос)

Что выведет данный код?

```
b = list.pop([1, 2, 3])
print(b)
```

A. 3

B. [1, 2]

C. ошибку

D. ничего

E. нет правильного ответа

47. Сұрақ (Вопрос)

Что выведет данный код?

```
a = str([1, 2])
print(int(a))
```

A. 12

B. 3

C. [1, 2]

D. ошибку

E. нет правильного ответа

48. Сұрақ (Вопрос)

Что выведет данный код?

```
a = [1, 2].insert(1, E.
print(a)
```

A. 1

B. 5

C. None

D. ошибку

E. нет правильного ответа

49. Сұрақ (Вопрос)

Что выведет данный код?

```
a = list({1: 2})
print(a)
```

A. [1]

B. [2]

C. [1, 2]

D. ошибку

E. нет правильного ответа

50. Сұрақ (Вопрос)

Что выведет данный код?

```
c = [c * 0 for c in 'li' if c != 'i']
print(c)
```

A. []

B. ['']

C. ошибку

D. ничего

E. нет правильного ответа

51. Сұрақ (Вопрос)

Что выведет данный код?

```
a = [1] + 3
print(a)
```

A. 4

B. 13

C. [1, 3]

D. ошибку

E. нет правильного ответа

52. Сұрақ (Вопрос)

Что выведет данный код?

```
a = str([1, 5]) + "3]"
print(a)
```

A. [1, 5]3]

B. [1, 3]

C. [1, 53]

D. ошибку

E. нет правильного ответа

53. Сұрақ (Вопрос)

Что выведет данный код?

```
a = list.clear([1])
print(a)
```

A. 1
B. None
C. ошибку
D. ничего
E. нет правильного ответа

54. Сұрақ (Вопрос)

Что выведет данный код?

```
a = 1
print(a.bit_length())
```

A. 1
B. 0
C. 3
D. ошибку
E. нет правильного ответа

55. Сұрақ (Вопрос)

Что выведет данный код?

```
x = (1-2j)
print(x.imag)
```

A. 1
B. -2
C. -2.0
D. ошибку
E. нет правильного ответа

56. Сұрақ (Вопрос)

Что выведет данный код?

```
c = 10 / 0.0
print(c)
```

A. Infinity
B. 10
C. ошибку
D. ничего
E. нет правильного ответа

57. Сұрақ (Вопрос)

Что выведет данный код?

```
a = 0b110
print(a)
```

A. 2
B. 20
C. 6
D. 3
E. нет правильного ответа

58. Сұрақ (Вопрос)

Что выведет данный код?

```
b = abs(-5)
print(b)
```

A. -5
B. 5
C. ошибку
D. ничего
E. нет правильного ответа

59. Сұрақ (Вопрос)

Что выведет данный код?

```
x = 0.10
print(x)
```

A. 1
B. 0.1
C. 0.10
D. ошибку
E. нет правильного ответа

60. Сұрақ (Вопрос)

Что выведет данный код?

```
a = 10 / 2
print(a)
```

A. 5
B. 4
C. 5.0
D. ошибка
E. нет правильного ответа

61. Сұрақ (Вопрос)

Что выведет данный код?

```
a = 3
print(b.bit_length())
```

A. 2
B. 3
C. 1
D. ошибку
E. нет правильного ответа

62. Сұрақ (Вопрос)

Что выведет данный код?

```
a = abs(~1.0)
print(a)
```

A. 2
B. 1
C. -1
D. ошибку
E. нет правильного ответа

63. Сұрақ (Вопрос)

Что выведет данный код?

```
a = 0b10 ** 2
print(a)
```

A. 4
B. 8
C. 2
D. 1
E. нет правильного ответа

64. Сұрақ (Вопрос)

Что выведет данный код?

```
if 1:
print(0)
```

A. 1
B. 0
C. ошибку
D. ничего
E. нет правильного ответа

65. Сұрақ (Вопрос)

Что выведет данный код?

```
x = 'one' if x == 1 else 'other'
print(x)
```

A. one

B. other

C. 1

D. ошибку

E. нет правильного ответа

66. Сұрақ (Вопрос)

Что выведет данный код?

```
a = 10 if 0 else 5
print(a)
```

A. 10

B. 0

C. 5

D. ошибку

E. нет правильного ответа

67. Сұрақ (Вопрос)

Что выведет данный код?

```
while 1:
print(6)
break
else:
print(0)
```

A. 6

B. 0

C. 60

D. ошибку

E. нет правильного ответа

68. Сұрақ (Вопрос)

Что выведет данный код?

```
a = [0]
if a:
print('a - t')
```

A. a - t

B. 0 - t

C. [0] - t
D. ошибку
E. нет правильного ответа

69. Сұрақ (Вопрос)
Что выведет данный код?
if 1:
print('true')
else:
print('false')
A. true
B. false
C. ошибку
D. ничего
E. нет правильного ответа

70. Сұрақ (Вопрос)
Что выведет данный код?
if 1:
print(A.
else:
print(B.
elif 1:
print(0)
A. 1
B. 2
C. 0
D. ошибку
E. нет правильного ответа

71. Сұрақ (Вопрос)
Что выведет данный код?
if 0:
print(1)
A. 0
B. 1
C. ошибку
D. ничего
E. нет правильного ответа

72. Сұрақ (Вопрос)

Что выведет данный код?

```
a = 't' if '0' else 'f'
print(a)
```

A. t
B. f
C. ошибку
D. ничего
E. нет правильного ответа

73. Сұрақ (Вопрос)

Что выведет данный код?

```
for i in [1]:
print(i, end="")
else:
print(0)
```

A. 10
B. 1
C. 1 0
D. ошибку
E. нет правильного ответа

74. Сұрақ (Вопрос)

Что выведет данный код?

```
if (a == a):
print(A.
else:
print(0)
```

A. 1
B. 0
C. ошибку
D. ничего
E. нет правильного ответа

75. Сұрақ (Вопрос)

Что выведет данный код?

```
a = 'False'
print(bool(a))
```

A. True
B. False

C. ошибку
D. ничего
E. нет правильного ответа

76. Сұрақ (Вопрос)
Что выведет данный код?
while 10 == 10:
print(1)
break
else:
print(0)
A. 1
B. 0
C. ошибку
D. ничего
E. нет правильного ответа

77. Сұрақ (Вопрос)
Что выведет данный код?
a = True == False
print(a)
A. True
B. False
C. ошибку
D. ничего
E. нет правильного ответа

78. Сұрақ (Вопрос)
Что выведет данный код?
a = True == 1
print(a)
A. True
B. False
C. ошибку
D. ничего
E. нет правильного ответа

79. Сұрақ (Вопрос)
Что выведет данный код?
a = False == 1

print(a)

A. True
B. False
C. ошибку
D. ничего
E. нет правильного ответа

80. Сұрақ (Вопрос)

Что выведет данный код?

```
a = True - ~False
print(a)
```

A. -1
B. 2
C. 0
D. ошибку
E. нет правильного ответа

81. Сұрақ (Вопрос)

Что выведет данный код?

```
for i in int(True):
print(i)
```

A. 1
B. 0 1
C. ошибку
D. ничего
E. нет правильного ответа

82. Сұрақ (Вопрос)

Что выведет данный код?

```
a = "-" + str(int(True))
print(int(a))
```

A. -1
B. 1
C. ошибку
D. ничего
E. нет правильного ответа

83. Сұрақ (Вопрос)

Что выведет данный код?

a = True > False
print(a - 10)

A. -9
B. -10
C. ошибку
D. ничего
E. нет правильного ответа

84. Сұрақ (Вопрос)

Что выведет данный код?
a = str("10")
print(a)

A. 10
B. "10"
C. -10
D. ошибку
E. нет правильного ответа

85. Сұрақ (Вопрос)

Что выведет данный код?
i = int("10.0")
print(i)

A. 10
B. 10.0
C. ошибку
D. ничего
E. нет правильного ответа

86. Сұрақ (Вопрос)

Что выведет данный код?
c = str(10, 5)
print(c)

A. 105
B. 15
C. 10, 5
D. ошибку
E. нет правильного ответа

87. Сұрақ (Вопрос)

Что выведет данный код?
e = len("/")
print(e * 3)

A. ///
B. 3
C. 0
D. ошибку
E. нет правильного ответа

88. Сұрақ (Вопрос)

Что выведет данный код?
r = [i for i in range("2")]
print(r)

A. 0 1
B. 1 2
C. 2 2
D. ошибку
E. нет правильного ответа

89. Сұрақ (Вопрос)

Что выведет данный код?
for i in range(len("py")):
print(i)

A. 0 1
B. p y
C. 1 2
D. ошибку
E. нет правильного ответа

90. Сұрақ (Вопрос)

Что выведет данный код?
a = "2" * 3
print(a)

A. 33
B. 222
C. 6
D. ошибку
E. нет правильного ответа

91. Сұрақ (Вопрос)

Что выведет данный код?

```
a = 2 - "3"
print(a)
```

A. -1
B. 2-3
C. 1
D. ошибку
E. нет правильного ответа

92. Сұрақ (Вопрос)

Что выведет данный код?

```
a = str(0.0)
print(len(a))
```

A. 3
B. 2
C. 1
D. ошибку
E. нет правильного ответа

93. Сұрақ (Вопрос)

Что выведет данный код?

```
b = int("-1")
print(b)
```

A. 1
B. -1
C. 11
D. ошибку
E. нет правильного ответа

94. Сұрақ (Вопрос)

Что выведет данный код?

```
a = set([1, 2])
b = set([1, 2])
print(a == b)
```

A. True
B. False
C. ошибку
D. ничего

E. нет правильного ответа

95. Сұрақ (Вопрос)

Что выведет данный код?

```
a = set([1, 2, 3, 2])
print(a)
```

A. {1, 2, 3}

B. {1, 2, 3, 2}

C. ошибку

D. ничего

E. Нет точного ответа

96. Сұрақ (Вопрос)

Что выведет данный код?

```
c = set("ру")
print(c)
```

A. {'р', 'у'}

B. {'у', 'р'}

C. {'ур'}

D. ошибку

E. нет точного ответа

97. Сұрақ (Вопрос)

Что выведет данный код?

```
a = {i ** 2 for i in range(1)}
print(a[0])
```

A. 0

B. 2

C. 1

D. 12

E. ошибку

98. Сұрақ (Вопрос)

Что выведет данный код?

```
a = {1, 2}
print(type(a))
```

A. class 'set'

B. class 'list'

C. class 'dict'

D. ошибку

E. нет правильного ответа

99. Сұрақ (Вопрос)
Что выведет данный код?
```
c = set("py")
cc = set("py")
print(c ==cc)
```
A. True
B. False
C. ошибку
D. нет точного ответа
E. ничего

100. Сұрақ (Вопрос)
Что выведет данный код?
```
a = {}
print(type(a))
```
A. class 'set'
B. class 'list'
C. class 'dict'
D. ошибку
E. нет правильного ответа

101. Сұрақ (Вопрос)
Что выведет данный код?
```
a = {1}
b = a.copy()
print(a == b)
```
A. True
B. False
C. ошибку

102. Сұрақ (Вопрос)
Что выведет данный код?
```
a = set([1, 5, 2, 4, 1])
b = set([1, 4])
print(a.difference_update(b))
```
A. {1, 4}
B. {1, 4, 1}
C. None

D. ошибку
E. нет правильного ответа

103. Сұрақ (Вопрос)
Что выведет данный код?
```
a = {}
b = a.copy()
print(a == b)
```
A. True
B. False
C. ошибку

ЛИТЕРАТУРА

1. Меняев,М.Ф. Информатика и основы программирования: учеб.пособие/ Меняев,М.Ф.. -3-е изд.,стер.. -М.: Омега-Л, 2007.
2. Вендров А.М. Проектирование программного обеспечения экономических информационных систем: Учебник. – М.: Финансы и статистика, 2006.
3. Вендров А.М. Практикум по проектированию программного обеспечения экономических информационных систем: Учеб. пособие. – М.: Финансы и статистика, 2006.
4. Орлов С.А. Программная инженерия. Технологии разработки программного обеспечения – СПб.: Питер, 2016. Программирование в Delphi для Windows. Версии 2006, 2007, Turbo Delphi – М.: ЗАО «Издательство БИНОМ», 2007.
5. Сысоева М. В., Сысоев И. В. Программирование для «нормальных» с нуля на языке Python: Учебник. В двух частях. Часть 1 / Ответственный
6. редактор: В. Л. Черный : — М.: Базальт СПО; МАКС Пресс, 2018. — 176 с. [+4 с. вкл]: ил. — (Библиотека ALT).