МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭКОНОМИЧЕСКИЙ УНИВЕРСИТЕТ»

КАФЕДРА ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ И ПРОГРАММИРОВАНИЯ

Д.Ю. ФЕДОРОВ

ЯЗЫКИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ

Учебное пособие

ИЗДАТЕЛЬСТВО САНКТ-ПЕТЕРБУРГСКОГО ГОСУДАРСТВЕННОГО ЭКОНОМИЧЕСКОГО УНИВЕРСИТЕТА
2019

ББК 32.973.2
Ф33

Федоров Д.Ю.

Ф33 Языки и методы программирования : учебное пособие / Д.Ю. Федоров. – СПб. : Изд-во СПбГЭУ, 2019. – 74 с.

ISBN 978-5-7310-4701-2

В учебном пособии раскрываются основные конструкции языка программирования Python, приводится практикум и демонстрируется пример интеграции языков программирования.

Предназначено для направлений подготовки бакалавров 10.03.01 – «Информационная безопасность», 01.03.02 – «Прикладная математика и информатика», 09.03.02 – «Информационные системы и технологии», 38.03.05 – «Бизнес-информатика»; может представлять интерес для преподавателей смежных дисциплин.

The tutorial covers the basic constructs of the Python programming language, provides a workshop and demonstrates an example of the integration of several programming languages.

The manual is intended for the bachelor degree program 10.03.01 – «Information Security», 01.03.02 – «Applied Mathematics and Informatics», 09.03.02 – «Information Systems and Technologies», 38.03.05 – «Business Informatics»; It may be of interest to teachers of related disciplines.

ББК 32.973.2

Рецензенты: канд. техн. наук, доцент **Г.М. Чернокнижный**
канд. экон. наук, доцент **И.Г. Гниденко**

ISBN 978-5-7310-4701-2

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Распоряжением Правительства Российской Федерации от 28.07.2017 № 1632-р была утверждена программа «Цифровая экономика Российской Федерации»[1], которая предполагает усовершенствование системы образования для подготовки грамотных кадров, обладающих базовыми компетенциями цифровой экономики. Одной из таких компетенций является знание языков программирования, которые изучаются в рамках дисциплины «Языки и методы программирования».

Существуют различные представления о методах преподавания данной дисциплины. Классический подход опирается на изучение языков C и C++, таким образом, обучающиеся переходят от процедурной к объектно-ориентированной парадигме программирования так, как это было исторически. В данном учебном пособии, исходя из собственного педагогического опыта [6], автором предложен иной подход: от объектно-ориентированной парадигмы языка Python к процедурной парадигме языка C.

Учебное пособие состоит из трех разделов.

В разделе 1 представлен обзор основных этапов развития языков программирования и основные синтаксические конструкции языка Python.

В разделе 2 приводятся примеры работы с файлами и собственными типами данных, рассматриваются свойства объектно-ориентированной парадигмы программирования.

Раздел 3 включает обзор практических задач, решаемых с помощью языка Python.

Учебное пособие предназначено для направлений подготовки бакалавров 10.03.01 – «Информационная безопасность», 01.03.02 – «Прикладная математика и информатика», 09.03.02 – «Информационные системы и технологии», 38.03.05 – «Бизнес-информатика»; может представлять интерес для преподавателей смежных дисциплин.

Учебное пособие направлено на формирование следующих компетенций:

- способность к разработке алгоритмических и программных решений в области системного и прикладного программирования, созданию прикладных баз данных, тестов и средств тестирования систем и средств на соответствие стандартам и исходным требованиям;
- способность решать стандартные задачи профессиональной деятельности с применением информационно-коммуникационных технологий и с учетом основных требований информационной безопасности.

[1] см. http://ac.gov.ru/files/content/14091/1632-r-pdf.pdf

ГЛАВА 1. ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ PYTHON

«*Есть два типа языков программирования –*
те, которые все ругают,
и те на которых никто не пишет».
(Б. Страуструп, разработчик языка программирования C++)

1.1. История создания языков программирования

У языков программирования интересная история. Они создавались для решения конкретных задач, стоящих на тот момент перед их разработчиками, отсюда становится понятной область применения того или иного языка программирования. На сегодняшний день существуют тысячи языков, но наибольшую роль сыграли лишь некоторые из них.

Началом работы с компьютером послужил машинный код – набор нулей и единиц. Затем в 50-ые годы двадцатого века появился низкоуровневый язык ассемблера, наиболее приближенный к машинному уровню. Он привязан к процессору, поэтому его изучение равносильно изучению архитектуры процессора. На языке ассемблера пишут программы и сегодня, он незаменим в случае небольших устройств (микроконтроллеров), обладающих ограниченными ресурсами памяти.

Следующий этап – появление языка Fortran, предназначавшегося для математических вычислений.

Со временем росла потребность в новых кадрах и необходимость в обучении программированию. Обучение на языках ассемблера или Fortran требовало много сил, поэтому в 60-70-ые годы появляется плеяда языков для обучения: Basic, Pascal. Язык Pascal до сих пор используется в школах в качестве основного языка обучения программированию.

В это же время ведутся исследования в области разработки операционных систем, что приводит к появлению системы UNIX. Первоначально эта операционная система была написана на языке ассемблера, что усложняло ее модификацию и изучение, тогда Деннис Ритчи разработал язык C для системного программирования и совместно с Брайаном Керниганом переписал систему UNIX на язык C. Впоследствии операционная система UNIX получила широкое распространение (в наши дни больше известны ее клоны GNU/Linux), а вместе с ней – появилось множество программистов, для которых язык C стал родным. Написание программ на этом языке требует хорошей квалификации от программиста, т.к. незамеченная ошибка способна привести к серьезным последствиям в работе программы. До сих пор язык C лидирует в качестве языка для системного программирования.

Следующий этап (80-ые годы) характеризуется появлением объектно-ориентированного программирования (ООП), которое должно было упростить создание крупных промышленных программ. Появляется ученый – Бьерн Страуструп, которому недостаточно было возможностей языка С, поэтому он расширяет этот язык путем добавления ООП. Новый язык получил название С++.

В 90-ые годы появляются персональные компьютеры и сеть Интернет, потому требуются новые технологии и языки программирования. В этот момент набирает популярность язык Java (наиболее популярный язык программирования в мире), который позволяет в кратчайшие сроки начать писать крупные приложения без опасений что-либо серьезно испортить в системе. Язык Java создавался с оглядкой на С++ и с перспективной развития сети Интернет. Данный язык характеризуется переносимостью своих программ, то есть написав Java-программу на персональном компьютере, можно запустить ее на кофемашине, если там присутствует виртуальная машина Java.

Примерно в одно время с Java появляется Python[2]. Разработчик языка – нидерландский математик Гвидо ван Россум (ныне работает в компании Dropbox) занимался долгое время разработкой языка ABC, предназначенного для обучения программированию. В одном из интервью он так ответил на вопрос о типе программистов, для которых Python был бы интересен: «Я представлял себе профессиональных программистов в UNIX или UNIX-подобной среде. Руководства для ранних версий Python возвещали что-то вроде «Python закрывает разрыв между Си и программированием оболочки», потому что именно это интересовало меня и моих ближайших коллег. Мне и в голову не приходило, что Python может стать хорошим языком для встраивания в приложения, пока меня не стали спрашивать об этом. То, что он оказался полезен для обучения началам программирования в школе или колледже, – счастливая случайность, обусловленная многими характеристиками ABC, которые я сохранил: ABC был специально предназначен для обучения программированию непрограммистов» [5]. На сегодняшний день Python занимает 4 место по популярности в рейтинге TIOBE[3] после Java, C и C++.

С ростом сети Интернет потребовалось создавать динамические сайты – появился серверный язык программирования PHP, который на сегодняшний день является лидером при разработке веб-сайтов.

В 2000-ые годы наблюдается тенденция объединения технологий вокруг крупных корпораций. В это время получает развитие язык C# на платформе .NET.

---

[2] Язык назван так в честь комедийного сериала Monty Python

[3] см. https://www.tiobe.com/tiobe-index/

В последнее время набирает популярность язык Go[4], который позиционируется как серверный язык программирования.

1.2. Этапы разработки программ

На первом шаге программист обладает набором «сырых» данных (рис. 1): разрозненными бухгалтерскими отчетами, статистикой и пр.

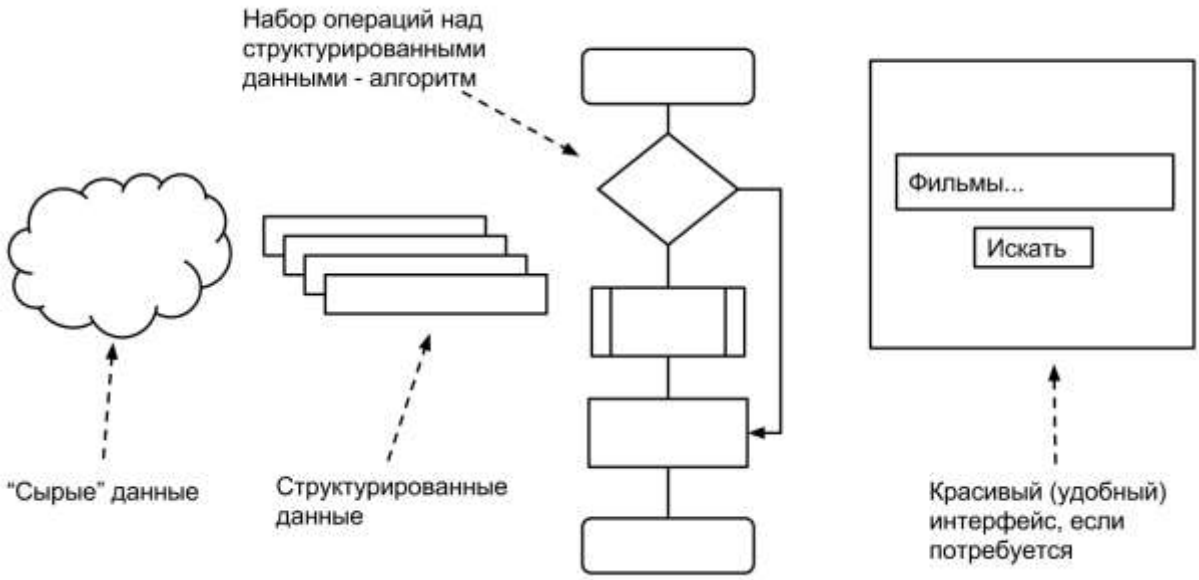


Рис. 1. Основные шаги при создании программы

Эти сведения необходимо структурировать и загрузить для обработки в компьютер. Далее, программист создает (выбирает) алгоритм, т.е. набор конечных действий для обработки структурированных данных, исходя из поставленной задачи. Отметим, что правильный выбор структуры данных влияет на создание (выбор) алгоритма. Сила языка программирования во многом заключена в структурах данных, которое он предоставляет для работы. После того, как алгоритм разработан, и программа работает (в результате ее работы получается корректный ответ), можно переходить к созданию интерфейса для диалога с пользователем.

1.3. Выполнение программ на языке Python и среда разработки JupyterLab

Выполнение программ осуществляется операционной системой (Microsoft Windows, GNU/Linux и пр.). В задачи операционной системы

[4] см. https://golang.org

(ОС) входит распределение ресурсов (оперативной памяти и пр.) для программы, запрет или разрешение на доступ к устройствам ввода/вывода и. т. д. Для запуска программ, написанных на языке программирования Python [7], понадобится программа-интерпретатор[5] – виртуальная машина Python[6] (рис. 2).

Рис. 2. Схема запуска программ

Виртуальная машина скрывает от Python-программиста особенности ОС, поэтому, написав программу в системе Windows, ее можно запустить, например, в GNU/Linux и получить схожий результат[7].

В качестве среды разработки будем использовать JupyterLab[8] – развитие проекта IPython[9], который в интерактивном режиме по средствам веб-интерфейса позволяет запускать программы на языке Python 3. JupyterLab (рис. 3) в отличие от IPython включает в себя не только интерпретатор языка Python, но и поддержку таких языков как Scala, Bash, Haskell, Julia, R, Ruby и пр. JupyterLab поддерживает отображение и редактирование множества форматов данных: изображений, CSV, JSON, Markdown, PDF, HTML, LaTeX и пр. Терминалы JupyterLab обеспечивают полную поддержку для системных оболочек (bash, tcsh и т. д.) на GNU/Linux и PowerShell на Windows[10].

---

[5] Python является интерпретируемым языком программирования (команды выполняются шаг за шагом).

[6] Далее рассматривается CPython – классическая реализация на языке C.

[7] При условии, что не производится обращение к специфическим возможностям отдельной операционной системы.

[8] Официальный сайт: http://jupyter.org.
Документация JupyterLab: https://jupyterlab.readthedocs.io/en/stable/index.html

[9] Проект, основанный в 2001 году, как усовершенствованный интерактивный интерпретатор языка Python.

[10] Обзор среды JupyterLab: https://proglib.io/p/jupyter/

Выполнить тестовый запуск полноценной версии JupyterLab можно на сайте: https://jupyter.org/try.

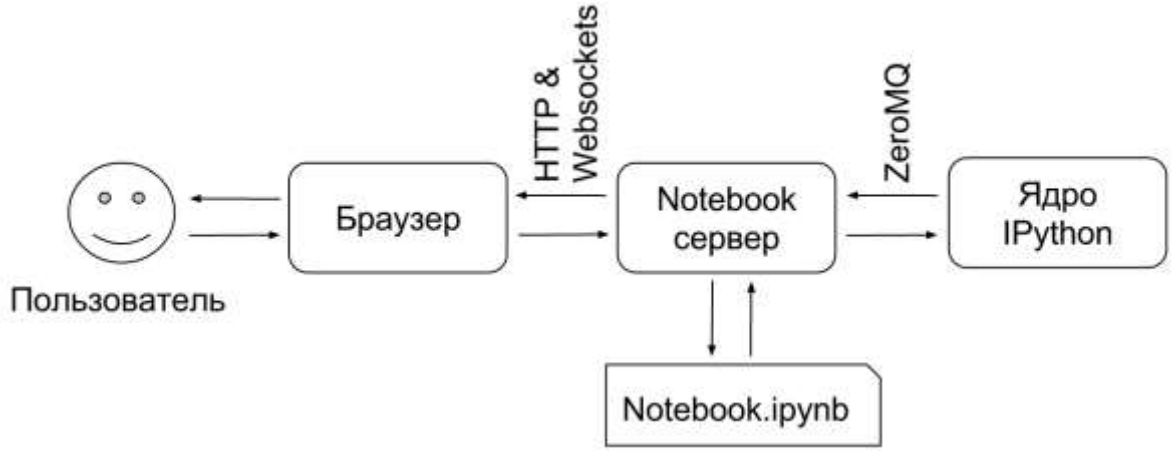


Рис. 3. Архитектура JupyterLab на основе ядра IPython

Для установки JupyterLab под ОС Windows[11] понадобится скачать и распаковать дистрибутив WinPython[12] версии 3.7 или выше. После установки запустите Jupyter Lab.exe: создастся локальный веб-сервер, прослушивающий сетевой порт с номером 8888, далее автоматически на странице http://localhost:8888/lab откроется веб-браузер. Создайте новый блокнот (Notebook) для дальнейшего запуска программ на языке Python 3: New → Notebook (Select Kernel: Python 3). Откроется веб-интерфейс с возможностью набора команд в строке In []. Переименуйте блокнот (File → Rename Notebook) в MyTest. Убедитесь, что в каталоге \notebooks\ появился файл с именем MyTest.ipynb[13].

1.4. Вычисление математических выражений на языке Python (простой калькулятор)

В самом начале обучения Python можно рассматривать как интерактивный калькулятор. Вычислим значения математических выражений. Для этого в ячейке In [] блокнота[14] JupyterLab наберите:

```
3.6 + 8
```

[11] Для пользователей ОС GNU/Linux: pip3 install jupyterlab и jupyter lab
[12] Официальный сайт: https://winpython.github.io. Еще один вариант установки – Anaconda: https://www.continuum.io
[13] Это файл в формате JSON.
[14] Пользователям программы Wolfram Mathematica знакомы пронумерованные строки.

Для выполнения ячейки и выделения следующей нажмите 

(сочетание клавиш <Shift> + <Enter> или в меню: Cell → Run Cells and Select Below). Результат отобразится в ячейке Out[15] [] с индексом 1 (рис. 4).

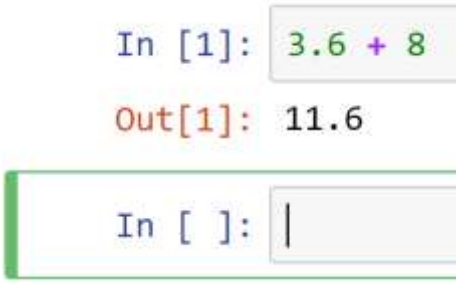


Рис. 4. Результат вычисления суммы чисел

Аналогично (каждое в отдельной ячейке) найдите значение следующих выражений:

```
4 + 9
1 - 5
_ + 6
```

Нижним подчеркиванием в предыдущем примере обозначается последний полученный результат. Если совершить ошибку при вводе команды:

```
In [5]:  a
```

интерпретатор сообщит:

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-5-60b725f10c9c> in <module>()
----> 1 a

NameError: name 'a' is not defined
```

В математических выражениях в качестве операндов могут использоваться как *целые числа*[16] (например, 1, 4, -5) так и *вещественные*[17] (*числа с*

[15] В новых версиях JupyterLab наименования ячеек OUT убрали.
[16] А также комплексные числа и логические значения: True, False.
[17] Объяснение правил хранения вещественных чисел в компьютере выходит за рамки пособия, сравните в следующем примере результаты вычислений:

плавающей точкой): 4.111, -9.3. Математические операции, доступные над числами в Python[18]:

Оператор	**Описание**
+	Сложение
-	Вычитание
/	Деление (в результате вещественное число)
//	Деление с округлением вниз
**	Возведение в степень
%	Остаток от деления

Таким образом, рассмотрели *числовой тип данных* (целочисленный тип int и вещественный тип float), т.е. множество числовых значений и множество математических операций, которые можно выполнять над данными значениям.

На языке Python вычислить значение `y = x + 3 * 6` при `x` равном 1 можно следующим образом:

```
In  [6]: x = 1
         y = x + 3*6
         y
Out [6]: 19
```

В выражении нельзя использовать переменную, если ранее ей не было присвоено значение с помощью *инструкции присваивания*. Для Python такие переменные не определены, и их использование приведет к ошибке (с подобной ошибкой уже столкнулись ранее).

Имена переменным задает программист, но есть несколько ограничений: нельзя начинать с цифры и использовать ключевые слова, которые для Python имеют определенный смысл (эти слова подсвечиваются в JupyterLab зеленым цветом):

and	as	assert	break	class	continue
def	del	elif	else	except	exec
finally	for	from	global	if	import
in	is	lambda	nonlocal	not	or
pass	raise	return	try	while	with
yield	True	False	None		

In [1]: 2/3 + 1
Out [1]: 1.666666666666666**5**
In [2]: 5/3
Out [2]: 1.666666666666666**7**

[18] В Python выражение (b * (a // b) + a % b) эквивалентно a.

Рассмотрим формулу перевода градусов по шкале Цельсия (T_C) в градусы по шкале Фаренгейта (T_F):

$$T_F = \frac{9}{5} T_C + 32$$

Определим значение T_F при T_C равном 26:

```
In  [7]: cel = 26
         cel
Out [7]: 26
In  [8]: 9/5 * cel + 32
Out [8]: 78.80000000000001
```

В момент выполнения инструкции присваивания cel = 26 в памяти компьютера создается *объект* (рис. 5), расположенный по некоторому *адресу* (условно обозначим его как *id*1), имеющий значение 26 целочисленного типа int. Затем создается переменная с именем cel, которой *присваивается адрес* объекта *id*1. Таким образом, переменные в Python содержат адреса объектов или *переменные ссылаются на объекты*. Тип переменной определяется типом объекта, на который он ссылается.

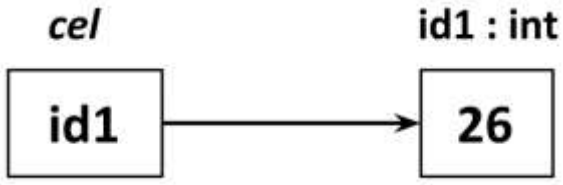


Рис. 5. Модель памяти для выражения cel = 26

JupyterLab позволят проводить *интроспекцию объекта*, т.е. выводить общую информацию о нем:

```
In  [9]: cel?
```

В дополнительном окне появится:

```
Type:        int
String form: 26
Docstring:
int(x=0) -> integer
int(x, base=10) -> integer
```

В Python числовые объекты являются *неизменяемыми*, поэтому при изменении содержимого переменной: создается новый числовой объект, а предыдущий удалятся[19] из памяти с помощью автоматической *сборки мусора*.

Функцией в программировании называется именованная последовательность инструкций. Рассмотрим встроенную функцию с именем abs, принимающую на вход один аргумент – объект числового типа и возвращающую абсолютное значение для этого объекта:

```
In  [10]: abs(-9)
Out [10]: 9
```

На практике при написании программ требуется преобразовывать типы объектов. Функция int принимает значение и преобразует его в целое число, если это возможно (возвращает 0, если аргумент не передан). Может преобразовать число с плавающей точкой в целое через отсечение дробной части:

```
In  [11]: int(5.6)
Out [11]: 5
```

Функция float возвращает число с плавающей точкой, построенное из числа или строки[20] (возвращает 0.0, если аргументы не переданы).

help отображает документацию[21] для функции, указанной в качестве аргумента:

```
In [13]: help(abs)

Help on built-in function abs in module builtins:

abs(x, /)
    Return the absolute value of the argument.
```

Вернемся к формуле перевода градусов по шкале Фаренгейта (T_F) в градусы по шкале Цельсия (T_C):

$$T_C = \frac{5}{9}(T_F - 32)$$

Создадим собственную функцию, выполняющую данные преобразования:

---

[19] Если на него не ссылаются другие переменные.

[20] О строках речь пойдет далее.

[21] Или воспользоваться возможностью интроспекции в JupyterLab: **abs?**

```
In  [19]:  def convert_co_cels(fahren):
               return (fahren-32)*5/9
```

Ключевое слово def означает, что далее последует определение функции. После def указывается имя функции convert_co_cels, затем в скобках указывается *параметр* (или параметры через запятую), которому будет присваиваться значение при вызове функции. Параметры функции – обычные переменные, которыми функция пользуется для внутренних вычислений. Переменные, объявленные внутри функции, называются *локальными* и не видны вне функции. После символа «:» начинается *тело функции* – блок команд, относящийся к функции. Тело функции может содержать любое количество инструкций. JupyterLab самостоятельно расставит отступы[22] от края ячейки, тем самым обозначив, где начинается тело функции. Выражение, стоящее после инструкции return будет возвращаться в качестве результата вызова функции. После того как функция создана, ее можно вызвать, подставив в скобках фактические аргументы:

```
In    [20]:  convert_co_cels(451)
Out [20]:  232.77777777777777
```

Имена функций в Python являются обычными переменными, содержащими адрес объекта[23] типа функция[24], поэтому этот адрес можно присвоить другой переменной и вызвать функцию уже с другим именем[25] (в отдельной ячейке JupyterLab):

```
def summa(x, y):
    return x + y
f = summa
v = f(10, 3)  # вызываем функцию с другим именем
```

При создании собственных функций помните совет Роберта С. Мартина: «Функции должны делать одну вещь и делать ее хорошо и делать только ее».

Рассмотрим пример функции, вычисляющей значение $x^4 + 4^x$.

```
In  [20]:  def add(x, y):
               return x + y
```

---

[22] Отступы играют важную роль в Python, отделяя блок команд тела функции, цикла и пр., см. правила оформления исходного кода на Python в PEP8: https://www.python.org/dev/peps/pep-0008/

[23] В Python все является объектами.

[24] Да, да, это еще один тип данных.

[25] Если по ошибке переопределили имя стандартной функции, то вернуть все обратно поможет инструкция del.

```
In  [21]:  def fun(x):
               return add(pow(x, 4), pow(4, x))

In  [22]:  fun(5)
Out [22]:  1649
```

Здесь для записи функции fun используется функциональная нотация (префиксная). Функцию add для сложения определили отдельно.

Как отмечает Дейкстра[26]: «Способ управления сложными системами был известен еще в древности – **divide et impera** (разделяй и властвуй)». При проектировании сложной программной системы необходимо разделять ее на все меньшие и меньшие подсистемы (функции), каждую из которых можно совершенствовать независимо.

1.5. Строки

Для работы с текстом в Python предусмотрен специальный строковый тип данных str. Строковые объекты создаются, если текст поместить в одиночные апострофы или двойные кавычки:

```
In  [1]:  'hello'
Out [1]:  'hello'
```

Для работы со строками предусмотрено большое количество встроенных функций. Например, функция len определяет длину строки, переданной ей в качестве аргумента:

```
In  [2]:  len('Привет!')
Out [2]:  7
```

С помощью операции *конкатенации* (оператор + для строк) Python позволяет объединить несколько строк в одну (также допускается располагать строки последовательно без каких-либо операторов):

```
In  [3]:  'Привет, ' + 'земляне!'
Out [3]:  'Привет, земляне!'
```

Попросим Python повторить (размножить) строку три раза:

```
In  [4]:  "СПАМ" * 3
Out [4]:  'СПАМСПАМСПАМ'
```

---

[26] Дейкстра Эдсгер – нидерландский ученый, один из разработчиков концепции структурного программирования.

Строки, по аналогии с числами, можно присваивать[27] переменным:

```
In   [5]:  s = "Python"
In   [6]:  s
Out [6]:  'Python'
In   [7]:  s*4
Out [7]:  'PythonPythonPythonPython'
```

Попросим пользователя ввести значение с клавиатуры (через вызов функции input):

```
In   [8]:  s = input()
```

После выполнения команды появится форма для ввода текста:

```
Земляне, мы прилетели с миром!
In    [9]:  s
Out  [9]:  'Земляне, мы прилетели с миром!'
In   [10]:  type(s)
Out [10]:  str      # функция input возвращает строку!
```

В примере была вызвана функция input, результат ее выполнения присвоен переменной s. Содержимое переменной s вывели на экран – отобразилась фраза, введенная пользователем. Затем вызвали функцию type, которая позволяет определить тип объекта, ссылка на который содержится в переменной s.

Каждый символ строки имеет собственный порядковый номер (*индекс*). Нумерация символов начинается с нуля. Python позволяет обратиться к заданному символу строки с помощью оператора скобок:

```
In   [11]:  s = 'Я люблю писать программы!'
In   [12]:  s[0]
Out [12]:  'Я'
In   [13]:  s[-1]
Out [13]:  '!'
```

В квадратных скобках указывается индекс символа. Строки, как и числовые объекты, являются *неизменяемыми*. Прежде чем поймем, как «модифицировать» строку, познакомимся со *срезами*:

```
In   [14]:  s = 'Питоны водятся в Африке'
In   [15]:  s[1:3]
Out [15]:  'ит'
```

---

[27] Напоминаем, что в переменной хранится адрес объекта (в данном случае строкового объекта).

s[1:3] – срез (часть) строки s, начиная с индекса 1, заканчивая индексом 3 (не включительно). Теперь «изменим» первый символ в строке через создание новой строки:

```
In  [16]:  s = 'Я люблю писать программы!'
In  [16]:  'J' + s[1:]    # формируется новая строка
Out [17]:  'J люблю писать программы!'
```

1.6. Операторы сравнения и инструкция if

В Python для сравнения объектов предусмотрено несколько операторов:

>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно
==	Равно
!=	Не равно

Произведем сравнение двух чисел:

```
In  [1]:  6 > 5
Out [1]:  True
In  [2]:  7 < 1
Out [2]:  False
In  [3]:  7 == 7      # не путайте == и = (присвоить)
Out [3]:  True
In  [4]:  7 != 7
Out [4]:  False
```

Python возвращает True[28] (истину[29]), когда сравнение верное и False (ложь) – в ином случае. True и False относятся к *логическому* (*булевому*) типу данных bool:

```
In  [1]:  type(True)
Out [1]:  bool
```

Для Python истинным или ложным может быть не только логическое высказывание, но и объект. Любое число, не равное нулю, или непустой

[28] Обязательно с большой буквы.
[29] True интерпретируется Python как число 1, а False как число 0.

объект интерпретируется как истина. Числа, равные нулю, пустые объекты и специальный объект None[30] интерпретируются как ложь.

Рассмотрим более подробно три логических оператора: and, or, not. Отрицание not (не) наиболее простой из них:

```
In  [1]:  y = 6 > 8
In  [2]:  y
Out [2]:  False
In  [3]:  not y
Out [3]:  True
```

В результате применения логического оператора and (и) получим True или False, если его операндами являются логические высказывания:

```
In  [1]:  2 > 4 and 45 > 3  # комбинация False and True вернет False
Out [1]:  False
```

Для вычисления результата применения оператора and Python вычисляет операнды слева направо и возвращает первый объект, имеющий ложное значение.

```
In  [1]:  0 and 3 # вернет первый ложный объект-операнд
Out [1]:  0
In  [2]:  5 and 4 # вернет крайний правый объект-операнд
Out [2]:  4
```

Если Python не удается найти ложный объект-операнд, то он возвращает крайний правый операнд.

Логический оператор or действует похожим образом, но для объектов-операндов Python возвращает первый объект, имеющий истинное значение. Python прекратит дальнейшие вычисления, как только будет найден первый объект, имеющий истинное значение:

```
In  [1]:  2 or 3     # вернет первый истинный объект-операнд
Out [1]:  2
In  [2]:  None or 5
Out [2]:  5
In  [3]:  None or 0   # вернет оставшийся крайний объект-операнд
Out [3]:  0
```

Следующий полезный логический оператор – in. Он принимает две строки и возвращает True, если первая строка является подстрокой второй строки:

---

[30] Имеет специальный тип NoneType.

```
In    [1]:  'a' in 'abc'
Out [1]:  True
```

Наиболее часто логические выражения используются внутри условной инструкции if. В следующем листинге представлен пример программы, использующей условные инструкции[31]:

```
# программа для определения водородного показателя pH
value = input("Введите pH: ")
if value:   # проверяем, что пользователь хоть что-нибудь ввел
   # переводим в вещественное число ввод пользователя:
   pH = float(value)
   if pH == 7.0:     # вложенный if
      print(pH, "Вода")
   elif 7.36 < pH < 7.44:
      print(pH, "Кровь")
   else:
      print("Что это?!")
else:
   print("Введите значение pH!")
```

Контрольные задания

1. Напишите программу, определяющую максимальное из двух введенных чисел. Реализовать в виде вызова собственной функции, возвращающей большее из двух переданных ей чисел.
2. Напишите программу, проверяющую целое число на четность. Реализовать в виде вызова собственной функции.
3. Напишите программу, которая по коду города и длительности переговоров вычисляет их стоимость. Результат выводится на экран. Тарифы: Екатеринбург – код 343, 15 руб/мин; Омск – код 381, 18 руб/мин; Воронеж – код 473, 13 руб/мин; Ярославль – код 485, 11 руб/мин.

1.7. Подключение модулей

Предположим, что имеется несколько полезных функций, которые часто используются в программах. Эти функции удобно поместить в отдельный файл и загружать (вызывать) оттуда. В Python такие файлы называются *модулями*. Для того чтобы воспользоваться функциями, находящимися в отдельном модуле, его необходимо *импортировать* с помощью инструкции import:

```
In [1]: import math
```

[31] При наборе исходного текста следите за отступами

Команда загружает в память стандартный модуль math[32] (содержит набор математических функций), теперь можем обращаться к функциям, находящимся внутри этого модуля следующим образом (например, для нахождения квадратного корня[33] из 9):

```
In  [2]: math.sqrt(9)
Out [2]: 3.0
```

Указываем имя модуля, точку и имя функции с аргументами[34]. В момент импортирования модуля math создается переменная с именем math, которая является объектом типа module. Можно импортировать отдельную функцию из модуля:

```
In  [3]: from math import sqrt
In  [4]: sqrt(9)
Out [4]: 3.0
```

В этом случае переменная math создана не будет, а в память загрузится только функция sqrt.

Ранее для нахождения модуля числа вызывали функцию abs. Эта функция находится в модуле __builtins__ (два нижних подчеркивания до и после имени модуля[35]), который загружается в память в момент начала работы интерпретатора.

Перечень встроенных модулей: https://docs.python.org/3/py-modindex.html
Хранилище сторонних модулей: https://pypi.org
Укажу несколько интересных сторонних модулей Python:
webbrowser – открытие в браузере определенной веб-страницы;
requests – загружает файлы и веб-страницы из сети Интернет;
Beautiful Soup – синтаксический анализ HTML;
Selenium – имитация щелчка мыши в браузере;
OpenPyXL – работа с Excel таблицами;
PyPDF2 – работа с PDF-документами;
Python-Docs – работа с DOC-файлами.

Для установки сторонних модулей в командной строке ОС Windows (в WinPython надо запустить WinPython Command Prompt.exe) необходимо выполнить:

```
pip install <имя модуля>
```

---

[32] см. help(math)
[33] см. help(math.sqrt)
[34] JupyterLab позволяет подсказывать пользователю список методов: math.<**Tab**>
[35] см. help(__builtins__) и dir(__builtins__)

При создании собственных модулей рекомендуется отделять код функций от кода, где эти функции исполняются с помощью проверки if __name__ == "__main__":

```
from operator import add
def func(x):
    return add(pow(x, 2), 7)

if __name_ _ == "__main__":
  x = int(input("Введите значение: "))
  print(func(x))
```

Таким образом, если модуль импортируется, то в памяти создается переменная func. При выполнении модуля (Run –> Run Module) создается переменная func и производится ввод с клавиатуры

1.8. Строковые методы

Тип данных в Python определяется *классом*. Для упрощения представим класс, как аналог модуля, т.е. набор функций и переменных, содержащихся внутри класса. Функции, которые находятся внутри класса, называются *методами*. Их главное отличие от вызова функций из модуля заключается в том, что в качестве первого аргумента метод принимает, например, строковый объект, если это метод строкового класса:

```
In   [1]:  str.capitalize('hello')
Out [1]:  'Hello'
```

По аналогии с вызовом функции из модуля указывается имя класса str, затем через точку – имя строкового метода capitalize, который принимает один строковый аргумент.

Форма вызова метода через обращение к его классу с помощью точки называется *полной формой*, но чаще используют *сокращенную форму вызова метода*:

```
In   [2]:  'hello'.capitalize()
Out [2]:  'Hello'
```

Первый аргумент метода поместили на место имени класса[36]. Получение справки для метода производится путем вызова функции help, но вместо имени модуля указывается имя класса: help(**str**.capitalize)

[36] Python всегда преобразует сокращенную форму в аналогичную ей полную форму.

Python содержит строковые методы, которые возвращают True или False. Например, строковый метод startswith проверяет, начинается ли строка с символа, переданного в качестве аргумента методу:

```
In  [3]:  'spec'.startswith('a')
Out [3]:  False
```

Предположим, что переменная s содержит некоторую строку, тогда к ней применимы следующие методы[37]:

```
s.upper() – возвращает строку в верхнем регистре
s.lower() – возвращает строку в нижнем регистре
s.title() – возвращает строку, первый символ которой в верхнем регистре
s.find('вет', 2, 3) – возвращает позицию подстроки в интервале либо -1
s.count('e', 1, 5) – возвращает количество подстрок в интервале либо -1
s.isalpha() – проверяет, состоит ли строка только из букв
s.isdigit() – проверяет, состоит ли строка только из чисел
s.isupper() – проверяет, написаны ли все символы в верхнем регистре
s.islower() – проверяет, написаны ли все символы в нижнем регистре
s.istitle() – проверяет, начинается ли строка с большой буквы
s.isspace() – проверяет, состоит ли строка только из пробелов
```

Отдельно рассмотрим метод format, который позволяет формировать[38] строку:

```
In  [4]:  "{0}, {1}, {2}".format(10, 12.3, "str")
Out [4]:  '10, 12.3, str'
```

В приведенном примере вместо {0}, {1}, {2} будут подставлены соответственно аргументы метода format, начиная с нулевого.

Следующий пример демонстрирует подстановку ключевых аргументов метода format:

```
In  [5]:  "{model}, {color}".format(color = 'red', model='lada')
Out [5]:  'lada, red'
```

Начиная с версии Python 3.6 появилась возможность использовать *f-строки* или *форматированные строковые литералы*[39] (Formatted String Literals):

---

[37] см. https://docs.python.org/3/library/stdtypes.html#string-methods

[38] Поведение метода похоже на функцию printf из языка C.

[39] см. https://docs.python.org/3/whatsnew/3.6.html#whatsnew36-pep498

```
In    [6]:  num = 2/3
In    [7]:  num
Out [7]:  0.6666666666666666
In    [8]:  f'{num:.3f}' # подстановка переменной происходит внутри строки
Out [8]:  '0.667'
```

f-строки обладают всеми возможностями метода format, но работают быстрее.

ГЛАВА 2. ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД В ПРОГРАММИРОВАНИИ НА ЯЗЫКЕ PYTHON

2.1. Списки

Списки (list) являются аналогом массива в других языках программирования, за исключением важной особенности – списки в качестве своих элементов могут содержать объекты различных типов. Список может быть присвоен переменной (переменной присваивается адрес объекта типа список):

```
In  [1]: e = [56.8060, 57.1578, 57.4093, 56.1843, 57.2207]
In  [2]: e
Out [2]: [56.806, 57.1578, 57.4093, 56.1843, 57.2207]
In  [3]: e?
Type:        list
String form: [56.806, 57.1578, 57.4093, 56.1843, 57.2207]
Length:      5
Docstring:
list() -> new empty list
list(iterable) -> new list initialized from iterable's items
```

Список позволяет хранить разнородные данные, обращаться к которым можно через имя списка (в данном случае переменную e). Рассмотрим схематично работу Python со списками (рис. 6).

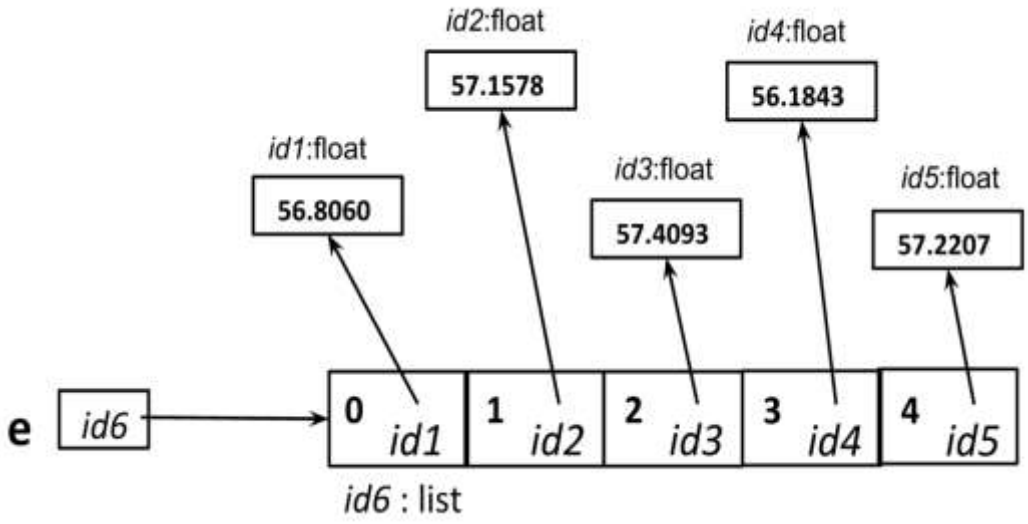


Рис. 6. Схема памяти Python при работе со списком

Переменная e содержит адрес списка (*id*6), каждый элемент списка содержит адрес соответствующего объекта.

Обращаться к отдельным элементам списка можно по их *индексу* (позиции), начиная с нуля по аналогии со строками:

```
In    [4]:  e = [56.8060, 57.1578, 57.4093, 56.1843, 57.2207]
In    [5]:  e[0]
Out [5]:  56.806
```

Список относится к *изменяемому типу данных*:

```
In    [6]: h = ['Hi', 27 , -8.1, [1,2]]   # создание списка
In    [7]: h[1] = 'hello' # изменение элемента списка в позиции 1
In    [8]: h                # результирующий список
Out [8]: ['Hi', 'hello', -8.1, [1, 2]]
```

Python содержит большое число встроенных функций, позволяющих легко и быстро обрабатывать списки:

len(L) – возвращает число элементов в списке L.
max(L) – возвращает максимальное значение в списке L.
min(L) – возвращает минимальное значение в списке L.
sum(L) – возвращает сумму значений в списке L.
sorted(L) – возвращает *копию* списка L, в котором элементы упорядочены по возрастанию.

Инструкция del удаляет элементы из списка по указанному индексу:

```
In    [9]:   h = ['Hi', 27 , -8.1, [1,2]]
In    [10]: del h[0]
In    [11]: h
Out [11]: ['hello', -8.1, [1, 2]]
```

Работа с методами списка[40] похожа на работу со строковыми методами[41]:

```
In    [1]: colors = ['red', 'orange', 'green']
In    [2]: colors.extend(['black','blue'])   # расширяет список
In    [3]: colors
Out [3]: ['red', 'orange', 'green', 'black', 'blue']
In    [4]: colors.append('purple')       # добавляет элемент в список
In    [5]: colors
Out [5]: ['red', 'orange', 'green', 'black', 'blue', 'purple']
```

[40] Подробнее: https://docs.python.org/3/tutorial/datastructures.html#more-on-lists
[41] Только вместо класса str в полной форме используется list, а в качестве первого аргумента метод принимает объект типа список.

На практике довольно часто возникает необходимость в изменении строк. Решением может стать преобразование строки в список, изменение списка и обратное преобразование списка в строку:

```
In  [1]: s = 'Строка для изменения'
In  [2]: list(s)  # функция list пытается преобразовать аргумент в список
Out [2]: ['С', 'т', 'р', 'о', 'к', 'а', ' ', 'д', 'л', 'я', ' ', 'и', 'з', 'м', 'е', 'н', 'е', 'н', 'и', 'я']
In  [3]: lst = list(s)
In  [4]: lst[0] = 'М'  # изменяем список, полученный из строки
In  [5]: lst
Out [5]: ['М', 'т', 'р', 'о', 'к', 'а', ' ', 'д', 'л', 'я', ' ', 'и', 'з', 'м', 'е', 'н', 'е', 'н', 'и', 'я']
In  [6]: s = ''.join(lst)  # преобразуем список в строку
In  [7]: s
Out [7]: 'Мтрока для изменения'
```

Строковый метод join принимает в качестве аргумента список, который необходимо преобразовать в строку, а в качестве строкового объекта указывается соединитель элементов списка.

Похожим образом можно преобразовать число к списку (на промежуточном этапе используется строка) и затем изменить полученный список:

```
In  [8]: n = 73485384753846538465
In  [9]: list(str(n)) # число преобразуем в строку, затем строку в список
Out [9]: ['7', '3', '4', '8', '5', '3', '8', '4', '7', '5', '3', '8', '4', '6', '5', '3', '8', '4', '6', '5']
```

Если строка содержит разделитель (пробел), то ее легко преобразовать к списку с помощью строкового метода split, который по умолчанию в качестве разделителя использует пробел:

```
In  [10]: s = 'd a dd dd gg rr tt yy rr ee'.split()
In  [11]: s
Out [11]: ['d', 'a', 'dd', 'dd', 'gg', 'rr', 'tt', 'yy', 'rr', 'ee']
```

При работе со списками необходимо внимательно производить присваивание:

```
In  [12]: h   # существует переменная, содержащая список
Out [12]: ['bonjour', 7, 'hola', -1.0, 'привіт']
In  [12]: p=h   # теперь p и h содержат указатель на один и тот же список
```

Если p и h содержат указатель на один и тот же список (являются псевдонимами), то изменение списка с помощью одной из этих переменных повлечет изменение в другой переменной. Выявить псевдонимы позволяет инструкция is:

```
In    [13]: x = y = [1, 2]
In    [14]: x is y
Out [14]: True # да, x и y - псевдонимы
```

В Python в целях повышения эффективности производится кэширование малых целых чисел:

```
In    [15]: x = 2; y = 2; z = 2  # объект 2 уже создан в памяти
In    [16]: x is y
Out [16]: True
In    [17]: y is z
Out [17]: True
```

Существует два способа копирования списков в Python: *поверхностное* (shallow) и *глубокое* (deep). Рассмотрим пример поверхностного копирования:

```
In    [1]: a = [4, 3, [2, 1]]
In    [2]: b = a[:] # производим копирование
In    [3]: b is a
Out [3]: False
In    [4]: b[2][0]=-100 # изменяем содержимое списка b
In    [5]: a # проверяем содержимое списка a
Out [5]: [4, 3, [-100, 1]] # список a изменился
```

Можно убедиться, что содержимое списка, на который указывает переменная a, изменится. Далее представлен пример глубокого копирования:

```
In    [6]: import copy # требуется подключить модуль copy
In    [7]: a = [4, 3, [2, 1]]
In    [8]: b = copy.deepcopy(a)
In    [9]: b[2][0] = -100
In    [10]: a
Out [10]: [4, 3, [2, 1]] # список a не изменился
```

2.2. Выполнение итераций на языке Python

Язык программирования Python позволяет в кратчайшие сроки создавать прототипы реальных приложений благодаря тому, что в него заложены конструкции для решения типовых задач. К примеру, вывести на экран каждый из элементов списка можно с помощью инструкции *цикла for:*

```
In [1]: num = [0.8, 7.0, 6.8, -6]
In [2]: for i in num:
            print(i, '- number')
```

```
0.8 - number
7.0 - number
6.8 - number
-6 - number
```

Инструкция цикла for позволяет обратиться к каждому из элементов указанного списка. Имя переменной, в которую на каждом шаге будет помещаться элемент списка, выбирает программист. На первом шаге переменной i присваивается первый элемент списка num, равный 0.8. Затем программа переходит в тело цикла for, отделенное отступами (четыре пробела). В теле цикла содержится вызов функции print, которой передается на вход переменная i. На следующем шаге (итерации цикла) переменной i присвоится второй элемент списка, равный 7.0. Произойдет вызов функции print для отображения текущего содержимого переменной i на экране и т.д. тело цикла выполняется до тех пор, пока не достигнет конца списка.

Инструкция for схожим образом работает для строк: происходит обращение к каждому из символов, пока не достигнет конца строки.

Достаточно часто на практике при разработке программ необходимо получить *диапазон* целых чисел, для этого предусмотрена функция range[42]. В качестве аргументов она принимает начальное значение диапазона (по умолчанию 0), конечное значение (*не включительно*) и шаг (по умолчанию 1). Если вызвать функцию range, то диапазон чисел не увидим:

```
In   [3]: range(0, 10, 1)
Out [3]: range(0, 10)
In   [4]: range(10)
Out [4]: range(0, 10)
```

Для создания диапазона (неизменяемой последовательности) необходимо использовать, например, инструкцию for:

```
In [5]: for i in range(2, 20, 2):
            print (i, end=' ')

        2 4 6 8 10 12 14 16 18
```

Таким образом, в переменную i на каждом шаге цикла (итерации) будет записываться значение из диапазона, который генерируется функцией range.

С помощью диапазона найдем сумму чисел на интервале от 1 до 100:

```
In   [6]:  sum(list(range(1, 101)))
Out [6]:  5050
```

[42] Подробнее: https://docs.python.org/3/library/functions.html#func-range

Рассмотрим пример создания списка с помощью метода append:

```
In   [1]:  a = []
In   [2]:  for i in range(1, 15):
               a.append(i)
In   [3]:  a
Out [3]:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

Инструкция for последовательно из диапазона от 1 до 14 выбирает числа и с помощью метода append, находящегося в теле цикла, добавляет их к списку a. Следующий пример – создание списка из диапазона с помощью функции list:

```
In   [1]:  a = list(range(1, 15))
In   [2]:  a
Out [2]:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

Далее показано использование «*спискового включения*» для создания списка:

```
In   [1]:  a = [i for i in range(1,15)]
In   [2]:  a
Out [2]:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

Правила работы со списковым включением представлены на рис. 7.

Рис. 7. Схема использования спискового включения

Функция map[43] позволяет создавать новый список на основе существующего:

```
In   [1]: def f(x):
              return x + 5

In   [2]: list(map(f, [1, 3, 4]))
Out [2]: [6, 8, 9]
```

[43] Подробнее: https://docs.python.org/3/library/functions.html#map

В примере функция map принимает два аргумента: имя функции f и список (можно передать строку). В процессе вызова функции map каждый элемент указанного списка (или строки) подается на вход функции f, и результат вызова функции f для каждого из элементов указанного списка «на лету» добавляется как элемент нового списка. Функция map возвращает объект типа map, поэтому получить итоговый список можно с помощью инструкции for либо функции list.

Функциональные возможности Python позволяют определять небольшие однострочные lambda-функции там, где требуется вызов функции:

```
In    [1]: list(map(lambda s: s*2, "hello"))
Out [1]: ['hh', 'ee', 'll', 'll', 'oo']
```

Рассмотрим генерацию списка, состоящего из случайных целых чисел:

```
In    [1]: from random import randint
In    [2]: A = [randint(1, 9) for i in range(5)]
In    [3]: A
Out [3]: [5, 6, 3, 3, 6]
```

В данном примере функция range выступает как счетчик количества элементов создаваемого списка. В результате пять раз будет произведен вызов функции randint[44], которая сгенерирует целое псевдослучайное[45] число из интервала от 1 до 9, и уже это число добавится в новый список.

Инструкция for используется, если заранее известно, сколько итераций необходимо выполнить (указывается через аргумент функции range или пока не закончится список/строка). В ином случае применяется инструкция цикла while. Тело цикла while выполняется до тех пор, пока выражение является истинным или не произошел выход из цикла с помощью инструкции break. Пример программы подсчета числа кроликов с использованием инструкции цикла while:

```
rabbits = 3
while rabbits > 0:
    print(rabbits)
    rabbits = rabbits - 1
```

Инструкция while выполняется до тех пор, пока число кроликов в условии положительное (rabbits > 0). На каждой итерации цикла перемен-

[44] Подробнее: https://docs.python.org/3/library/random.html#module-random
[45] т.е. генерируются на основе специального алгоритма «Вихрь Мерсенна»

ная rabbits уменьшается на 1, чтобы не получился бесконечный цикл, при котором условие всегда будет оставаться истинным. В начале работы программы инициализируется переменная rabbits (присваивается начальное значение 3), затем происходит переход в тело цикла while, т.к. условие rabbits>0 является истинным (вернет значение True). Далее в теле цикла вызывается функция print, которая отобразит на экране текущее значение переменной rabbits. После этого переменная rabbits уменьшится на 1 и снова произойдет проверка условия 2 > 0 (вернет True), перейдем в тело цикла и т.д., пока не дойдем до условия 0 > 0. В этом случае вернется логическое значение False и произойдет выход из инструкции while. В результате выполнения программы:

```
3
2
1
```

Бесконечный цикл, организованный с помощью инструкции while, позволяет производить обработку введенных с клавиатуры строк:

```
text = ""
while True:
text = input("Введите число или стоп для выхода: ")
    if text == "стоп":
        print("Выход из программы! До встречи!")
        break    # инструкция выхода из цикла
    elif text == '1':
        print("Число 1")
    else:
        print("Что это?!")
```

Результат выполнения программы:

```
Введите число или стоп для выхода: 4
Что это?!
Введите число или стоп для выхода: 1
Число 1
Введите число или стоп для выхода: стоп
Выход из программы! До встречи!
```

Тело цикла while в данном примере выполняется бесконечное число раз, т.к. True всегда является истиной. Выход из цикла осуществляется по инструкции break, если пользователь введет с клавиатуры строку "стоп".

Контрольные задания

1. Напишите программный код, который будет создавать новый список, содержащий в качестве элементов квадратные корни всех чисел из списка [2, 4, 9, 16, 25]:
   - на основе цикла for;
   - на основе функции map;
   - в виде генератора списка.
2. Напишите программу-игру: компьютер загадывает случайное число, пользователь пытается его угадать. Пользователь вводит число до тех пор, пока не угадает или не введет слово «Выход». Компьютер сообщает пользователю больше или меньше введенное число относительно загаданного.
3. Напишите программу. На вход программы поступает произвольный текст. Найдите номер первого самого длинного слова в нем.
4. Напишите программу. На вход программы поступает произвольный текст. Напечатайте все имеющиеся в нем цифры, определите их количество и сумму.

2.3. Дополнительные встроенные типы данных в Python

Множество (set) – неупорядоченная коллекция неизменяемых, уникальных элементов:

```
In  [1]: v = {'A', 'C', 4, '5', 'B'}
In  [2]: v
Out [2]: {'C', 'B', '5', 4, 'A'}
```

Повторяющиеся элементы, которые добавились при создании, удаляются из результирующего множества:

```
In  [3]: v = {'A', 'C', 4, '5', 'B', 4}
In  [4]: v
Out [4]: {'C', 'B', '5', 4, 'A'}
```

Создание множества из списка:

```
In  [5]:  set([3, 6, 3, 5])
Out [5]:  {3, 5, 6}
```

Обратите внимание, что в момент создания множества из списка удалены повторяющиеся элементы. Это легкий способ очистить список от повторов:

```
In  [6]:  list(set([3, 6, 3, 5]))
Out [6]:  [3, 5, 6]
```

Рассмотрим некоторые операции над множествами[46]:

```
In  [1]:  s1 = set(range(5))
In  [2]:  s2 = set(range(2))
In  [3]:  s1
Out [3]:  {0, 1, 2, 3, 4}
In  [4]:  s2
Out [4]:  {0, 1}
In  [5]:  s1.add('5')  # добавление элемента
In  [6]:  s1
Out [6]:  {0, 1, 2, 3, 4, '5'}
```

Кортеж (tuple) схож по своим свойствам со списком. Он используется, когда необходимо быть уверенным, что элементы структуры данных не будут изменены в процессе работы программы. Рассмотрим некоторые популярные операции над кортежами[47]:

```
In  [1]:  ()    # создание пустого кортежа
Out [1]:  ()
In  [2]:  (4)   # это не кортеж, а целочисленный объект!
Out [2]:  4
In  [3]:  (4,)  # а вот это – кортеж, состоящий из одного элемента!
Out [3]:  (4,)
In  [4]:  b = ('1', 2, '4')          # создание кортежа
In  [5]:  b
Out [5]:  ('1', 2, '4')
In  [6]:  len(b)  # определяем длину кортежа
Out [6]:  3
In  [7]:  t = tuple(range(10))   # создание кортежа
In  [8]:  t + b   # слияние кортежей
Out [8]:  (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, '1', 2, '4')
In  [9]:  r = tuple([1, 5, 6, 7, 8, '1'])   # создание кортежа из списка
In  [10]: r
Out [10]: (1, 5, 6, 7, 8, '1')
```

С помощью кортежей можно присваивать значения одновременно двум переменным:

```
In  [1]:  (x, y) = (10, 5)
In  [2]:  x
```

---

[46] см. https://docs.python.org/3/tutorial/datastructures.html#sets

[47] см. https://docs.python.org/3/tutorial/datastructures.html#tuples-and-sequences

```
Out [2]:  10
In   [3]:  y
Out [3]:  5
In   [4]:  x, y = 1, 3   # убрали круглые скобки
In   [5]:  x
Out [5]:  1
In   [6]:  y
Out [6]:  3
```

Словарь (dict) – неупорядоченная изменяемая коллекция или, проще говоря, «список» с произвольными ключами, неизменяемого типа.

Рассмотрим пример создания словаря, который каждому слову на английском языке (*множество ключей словаря*) ставит в соответствие слово на испанском языке (*набор значений словаря*). Схематично такой словарь представлен на рис. 8.

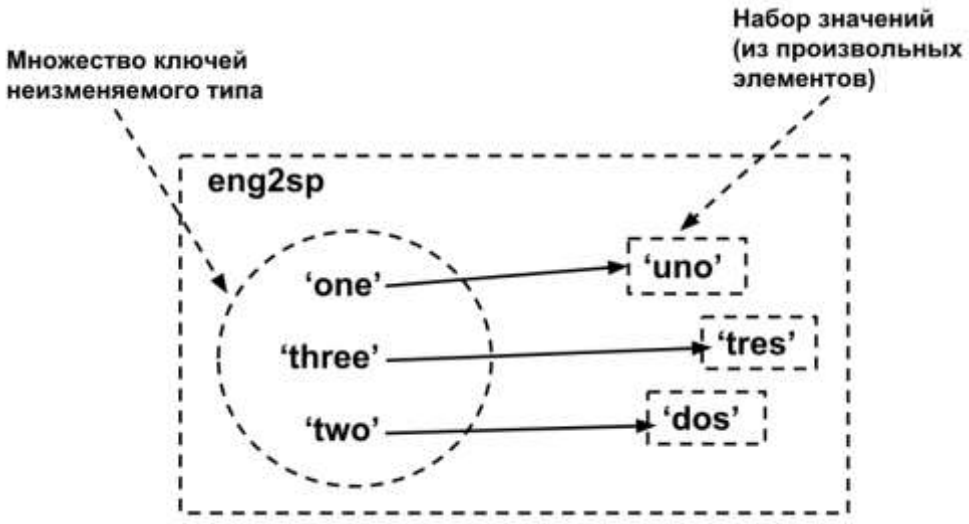


Рис. 8. Схема соответствия множеству ключей набор значений словаря

Создадим словарь-переводчик с английского языка на испанский язык:

```
In   [1]:  eng2sp = dict()   # создаем пустой словарь
In   [2]:  eng2sp
Out [2]:  {}
In   [3]:  eng2sp['one']='uno'
In   [4]:  eng2sp
Out [4]:  {'one': 'uno'}
In   [5]:  eng2sp['one']
Out [5]:  'uno'
In   [6]:  eng2sp['two'] = 'dos'
In   [7]:  eng2sp['three'] = 'tres'
In   [8]:  eng2sp
Out [8]:  {'three': 'tres', 'one': 'uno', 'two': 'dos'}
```

На основе словарей можно реализовать функцию для подсчета встречаемости элементов в последовательности:

```
# histogram.py
def histogram(s):
    d = dict()
    for c in s:
        if c not in d:
            d[c] = 1
        else:
            d[c] += 1
    return d

if __name__ == "__main__":
    print(histogram("sdfs"))
    print(histogram(['aaa', 'bbb', 'aaa', 'bbb', 'c']))
```

2.4. Обработка исключений

В Python реализован[48] перехват ошибок, основанный на обработке исключительных ситуаций:

```
try:
    x = int(input("Введите число: "))
    print(5/x)
except:
    print("Возникла ошибка деления на нуль")
```

Выполним программу и попытаемся разделить на нуль:

```
Введите число: 0
Возникла ошибка деления на нуль
```

В блок try помещается код, в котором может произойти ошибка. В случае возникновения ошибки (*исключительной ситуации*) управление передается в блок except. Повторно выполним программу и обнаружим, что при возникновении ошибки перевода буквы в число, снова попадаем в блок except:

```
Введите число:  к
Возникла ошибка деления на нуль
```

except без указания типа исключительной ситуации перехватывает все виды возникающих ошибок. Как нам отделить ошибку деления на нуль

[48] Другие объектно-ориентированные языки тоже поддерживают данный механизм.

от ошибки преобразования типов? Выполним несколько ошибочных команд, приводящих к исключениям[49]:

```
In [1]: 4/0

---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-39-6de94738d89d> in <module>()
----> 1 4/0

ZeroDivisionError: division by zero

In [2]: int("r")

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-40-991c836bf0cf> in <module>()
----> 1 int("r")

ValueError: invalid literal for int() with base 10: 'r'
```

При делении на нуль возникает ошибка ZeroDivisionError, при преобразовании типов – ValueError:

```
try:
    x = int(input("Введите число: "))
    print(5/x)
except ZeroDivisionError:  # указываем тип исключения
    print("Возникла ошибка деления на нуль")
except ValueError:
    print("Возникла ошибка преобразования типов")
```

Теперь срабатывают различные блоки except в зависимости от типа возникающего исключения.

2.5. Работа с текстовыми файлами

На практике данные для обработки часто поступают из внешних источников – файлов. Существуют различные форматы файлов, наиболее простой и универсальный – текстовый. Помимо текстовых есть другие типы файлов (музыкальные, видео, .doc, .ppt и пр.), которые открываются в специальных программах (музыкальных или видео проигрывателях, MS

[49] см. https://docs.python.org/3/library/exceptions.html#bltin-exceptions

Word и пр.). Остановимся на текстовых файлах, хотя возможности Python ими не ограничиваются.

Выполните следующие шаги.

1. Создайте директорию file_examples в notebooks;

2. Создайте в директории file_examples текстовый файл example_text.txt, содержащий следующий текст:

```
First line of text
Second line of text
Third line of text
```

3. В отдельной ячейке JupyterLab выполните код:

```
file = open('example_text.txt', 'r')
contents = file.read()
print(contents)
file.close()
```

В результате получится:

```
First line of text
Second line of text
Third line of text
```

Описание программы представлено на рис. 9.

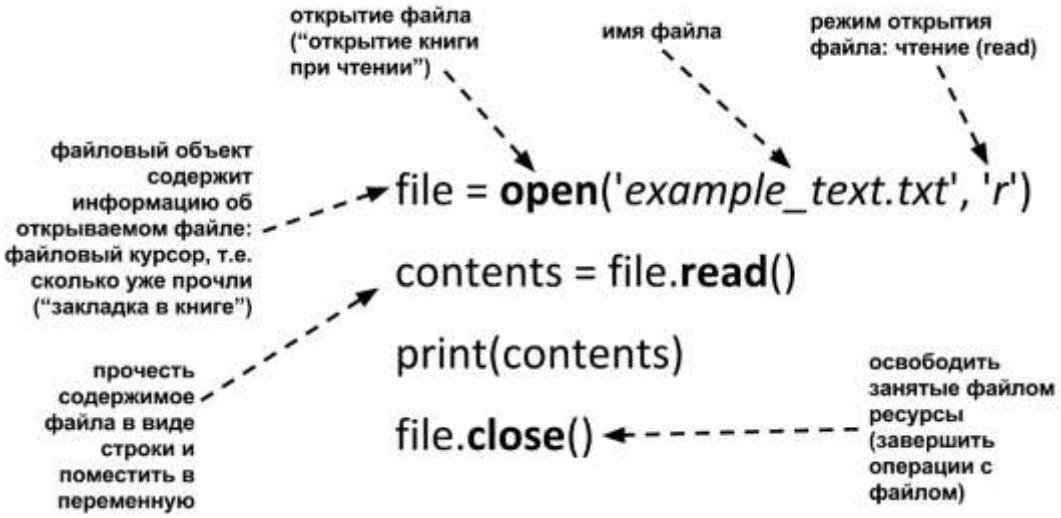


Рис. 9. Описание программы для работы с файлами в Python

Рассмотренный подход для работы с файлами[50] Python унаследовал из языка C. По умолчанию, если не указывать режим открытия, то использу-

[50] см. https://docs.python.org/3/library/os.html#os-file-dir

ется открытие на «чтение» (можно открыть на «запись» или «добавление»). В дальнейшем для работы с файлами будем использовать *менеджер контекста* (инструкцию with), который не требует ручного освобождения ресурсов, т.е. вызова метода close. В следующем примере представлен исходный код программы с использованием менеджера контекста:

```
try:
    # освобождение ресурсов происходит автоматически
    # внутри менеджера контекста:
    with open('1example_text.txt', 'r') as file:
        contents = file.read()
    print(contents)
except:
    print("Ошибка открытия файла")
```

Выполним программу:

```
Ошибка открытия файла
```

Далее представлен пример чтения содержимого всего файла, начиная с текущей позиции курсора (перемещает курсор в конец файла):

```
with open('example_text.txt', 'r') as file:
    contents = file.read()
print(contents)
```

Результат выполнения программы:

```
First line of text
Second line of text
Third line of text
```

В следующем примере производится запись строки (string) в файл (file). Если файла с указанным именем в рабочей директории нет, то он будет создан, если файл с таким именем существует, то он будет перезаписан:

```
def file_write(file, string):
    try:
        with open(file, 'w') as output_file:
            count = output_file.write(string)
        return count
    except Exception as e: # перехватываем все ошибки
        return e # возвращаем информацию об ошибке

if __name__ == '__main__': # отделяем запуск от импортирования
    print('записано символов {0}'.format(file_write("top.txt", "Hello!\n")))
```

JupyterLab обладает тесной интеграцией с оболочкой ОС. Существует набор магических функций[51], позволяющих выполнять команды, связанные с ОС:

```
In  [1]:  %pwd  # возвращает текущую рабочую директорию
Out [2]:  'C:\\WPy-3702\\notebooks'
```

Восклицательный знак ! в начале командной строки JupyterLab означает, что все следующее за ним необходимо выполнить в оболочке ОС.

Следующий исходный текст демонстрирует пример обращения к файлу, находящемуся в сети Интернет:

```
import urllib.request # модуль для работы с файлами в сети Интернет
def url_open(url):
    lst = list()
    try:
        # webpage – объект, который содержит информацию о файле:
        with urllib.request.urlopen(url) as webpage:
            for line in webpage:
                line = line.decode('utf-8') # декодируем строку
                line = line.strip()
                lst.append(line)
        return lst
    except Exception as e:
        return e

if __name__ == '__main__':
    print(url_open("http://dfedorov.spb.ru/python3/src/romeo.txt"))
```

Контрольные задания

1. Напишите программу, которая создает (генерирует) полноценный HTML-документ, содержащий текст, приведенный по ссылке: http://dfedorov.spb.ru/python/files/tutchev.txt и под текстом размещает картинку: http://dfedorov.spb.ru/python/files/tutchev.jpg. Пример итогового HTML-документа: http://dfedorov.spb.ru/python/files/p.html (обратите внимание на код страницы, содержащей HTML-теги). Выполните обработку ошибок. В момент чтения и записи используйте параметр функции open – encoding='utf-8'.

2. Определите частоту встречаемости слов для текста, расположенного в сети Интернет: http://dfedorov.spb.ru/python3/src/romeo.txt

[51] см. https://ipython.readthedocs.io/en/stable/interactive/magics.html

2.6. Работа с открытыми данными на языке Python

Открытые данные – это информация, которую кто угодно может свободно использовать и распространять. Допустимы лишь требования указывать источник данных и распространять их на тех же условиях, что и исходные[52].

Много полезного можно найти на портале открытых данных Российской Федерации: https://data.gov.ru

В качестве примера обратимся к открытым данным, предоставляемым «Сбербанком»[53]. Все данные находятся в файле формата csv – это текстовый файл, разделенный запятыми[54].

Рассмотрим программу, которая определяет за год среднюю зарплату в Санкт-Петербурге:

```
def money_reader(what, where, when):
    import csv # используем возможности модуля csv
    money = list()
    try:
        with open('opendata.csv', encoding='cp1251') as csvfile:
            money_reader = csv.reader(csvfile, delimiter=',')
            for row in money_reader:
                if row[0] == what:
                    if row[1] == where:
                        lst = row[2].split('-')
                        if lst[0] == when:
                            money.append(int(row[3]))
        return round(sum(money)/len(money), 2)
    except Exception as e:
        return e

if __name__ == '__main__': # отделяет запуск от импортирования
    for year in range(2015, 2019):
        print(str(year), money_reader('Средняя зарплата', 'Санкт-Петербург',
str(year)))
```

Контрольное задание

Определите регион с максимальным количеством заявок на потребительские кредиты в 2017 году.

---

[52] см. http://opendatahandbook.org/guide/ru/what-is-open-data/

[53] см. http://www.sberbank.com/ru/analytics/opendata

[54] см. http://sbrfdata.ru/opendata.zip

2.7. Работа с JSON

JSON (JavaScript Object Notation) – текстовый формат, предназначенный для обмена данными. JSON удобен тем, что позволяет сохранять в файле не отдельные строки (см. метод write), а типы данных, совместимые с Python. Например, преобразуем список в формат JSON с помощью функции dumps модуля json[55]:

```
In  [1]:  import json
In  [2]:  json.dumps(['foo', {'bar': ('baz', None, 1.0, 2)}])
Out [2]:  '["foo", {"bar": ["baz", null, 1.0, 2]}]'
```

Видим, что полученная строка в формате JSON содержит вместо объекта None языка Python тип null языка JavaScript. Произведем обратную операцию с помощью функции loads:

```
In  [3]:  json.loads('["foo", {"bar":["baz", null, 1.0, 2]}]')
Out [3]:  ['foo', {'bar': ['baz', None, 1.0, 2]}]
```

Получили обратно исходный список на языке Python.

Следующие две программы (number_writer.py и number_reader.py) демонстрируют соответственно запись и чтение списка языка Python в файл numbers.json в формате JSON:

```
# number_writer.py
def writer(filename, numbers):
    import json
    try:
        with open(filename, 'w') as f_obj:
            json.dump(numbers, f_obj)
    except Exception as e:
        print(e)

writer('numbers.json', [1, 3, 4, 5])

# number_reader.py
def reader(filename):
    import json
    try:
        with open(filename) as f_obj:
            numbers = json.load(f_obj)
        return numbers
```

[55] см. https://docs.python.org/3/library/json.html

```
    except Exception as e:
        return e

print(reader('numbers.json'))
```

2.8. Создание собственных типов данных

«В 1994 году Стив Джобс давал интервью журналу The Rolling Stone.

Джефф Гуделл: Объясните, пожалуйста, что такое объектно-ориентированный подход, используя простые термины.

Стив Джобс: Объекты похожи на людей. Они живые, дышащие сущности, которые обладают памятью и знаниями о том, как нужно действовать. И вместо того, чтобы взаимодействовать с ними на низком уровне, мы взаимодействуем с ними на очень высоком уровне абстракции.

Например, если для вас я объект-прачка, вы можете передать мне грязную одежду вместе с сообщением «постирай, пожалуйста, мою одежду». Я знаю, где в Сан-Франциско лучшая прачечная. Я говорю по-английски, в карманах у меня доллары. Я выхожу на улицу, ловлю такси и говорю таксисту, чтобы он отвез меня в нужное место. Я стираю вещи, снова сажусь в такси и возвращаюсь. Я отдаю вам чистую одежду с сообщением: «вот ваша чистая одежда».

Вы понятия не имеете, как и какие шаги я проделал. Вы не знаете о том, что существует прачечная. Возможно, вы говорите по-французски и не можете поймать такси – вам нечем за него заплатить, долларов у вас нет. Тем не менее, я знал, как все это сделать. А вам нет необходимости знать об этом. Вся комплексность была во мне запрятана, и мы смогли взаимодействовать на высоком уровне абстракции. В этом суть объектов – они сочетают в себе инкапсуляцию структурной сложности и высокоуровневый интерфейс»[56].

Собственные типы данных в Python определяются через создание классов. Например, создадим тип данных (класс) Address:

```
class Address:    # имя класса выбирает программист
    pass # пустая инструкция
```

Определили шаблон для хранения места проживания человека. Превратить шаблон в конкретный адрес можно через создание *объекта* (*экземпляра*)[57] класса Address[58]:

---

[56] Источник: http://eattheworldbook.com

[57] В Python классы являются объектами, но для упрощения будем считать, что это только шаблон для создания объектов.

[58] Вспомните о создании объекта класса int: a = int()

```
home = Address()
```

Заполним через точку поля объекта:

```
# заполняем поле name объекта home:
home.name = "Вася Пупкин"
home.line1 = "'Невский пр. 22"
home.city = "СПб"
```

Возможности классов и объектов не ограничиваются только хранением состояния объекта. Классы также позволяют задавать функции внутри себя (методы) для работы с полями класса, т.е. влиять на поведение объекта. Для демонстрации создадим класс Dog:

```
class Dog:
    # первым аргументом любого метода всегда является self,
    # т.е. сам объект:
    def bark(self): # функция внутри класса называется методом
        # self.name – обращение к имени текущего объекта-собаки
        print(self.name, " говорит гав")

# Создаем объект my_dog класса Dog:
my_dog = Dog()

# Присваиваем значения полям объекта my_dog:
my_dog.name = "Лайка" # имя собаки
my_dog.weight = 20       # вес собаки
my_dog.age = 1           # возраст собаки

# Вызываем метод bark объекта my_dog,
# т.е. попросим собаку подать голос:
my_dog.bark()

# Полная форма для вызова метода my_dog.bark()
# будет: Dog.bark(my_dog), где my_dog – сам объект (self)
```

Результат работы программы:

```
Лайка говорит гав
```

Данный пример демонстрирует *объектно-ориентированный подход в программировании*, когда создаются объекты, приближенные к реальной жизни. Между объектами происходит взаимодействие по средствам вызова методов. Поля объекта (переменные) фиксируют его состояние, а вызов метода приводит к реакции объекта и/или изменению его состояния (изменению переменных внутри объекта).

В предыдущем примере между созданием объекта my_dog класса Dog и присвоением ему имени (my_dog.name = "Лайка") прошло некоторое время. Может произойти так, что программист забудет указать имя и тогда собака останется безымянной. Избежать подобной ошибки позволяет специальный метод (*конструктор*), который вызывается автоматически в момент создания объекта заданного класса. Следующий пример демонстрирует присвоение имени собаке через вызов конструктора класса:

```
class Dog:
    # Конструктор
    # Вызывается на момент создания объекта этого класса
    def __init__ (self, new_name):
        self.name = new_name

# Создаем собаку и устанавливаем ее имя:
my_dog = Dog("Лайка")

# Выводим имя собаки:
print(my_dog.name)
```

Теперь имя собаки присваивается в момент ее создания («рождения»). В конструкторе указали self.name, т.к. в момент вызова конструктора вместо self подставится конкретный объект, т.е. my_dog. Результат выполнения программы:

```
Лайка
```

В предыдущем примере для обращения к имени собаки выводили на экран поле my_dog.name, т.е. залезали во «внутренности» собаки. Для более «гуманной» работы с объектом добавим в класс Dog два метода set_name и get_name:

```
class Dog:
    # Конструктор вызывается в момент создания объекта этого класса
    def __init__ (self, new_name):
        self.name = new_name
    # Можем в любой момент вызвать метод и изменить имя собаки
    def set_name(self, new_name):
        self.name = new_name
    # Можем в любой момент вызвать метод и узнать имя собаки
    def get_name(self):
        return self.name # возвращаем текущее имя объекта

# Создаем собаку с начальным именем:
my_dog = Dog("Лайка")
```

```
# Выводим имя собаки:
print(my_dog.get_name())

# Устанавливаем новое имя собаки:
my_dog.set_name("Шарик")

# Проверяем, что имя изменилось:
print(my_dog.get_name())
```

Выполним программу:

```
Лайка
Шарик
```

Переменные, которые объявляются в классе (вне методов), видны во всех экземплярах данного класса:

```
class Dog:
    # Атрибуты класса доступны из всех экземпляров данного класса
    count = 0
    legs = 4
    # Конструктор
    # Вызывается в момент создания объекта этого типа (класса)
    def __init__(self, new_name='Собака без имени'):
        self._name = new_name
        Dog.count += 1 # увеличиваем счетчик экземпляров класса Dog

    # Можем в любой момент вызвать этот метод и изменить имя:
    def set_name(self, new_name):
        self._name = new_name
    # Можем в любой момент вызвать этот метод и узнать имя:
    def get_name(self):
        return self._name

# Создаем конкретную собаку
my_dog = Dog()
# Изменяем имя собаки
my_dog.set_name("Шарик")
# Сколько ног у собаки (доступ к атрибуту класса):
print('У {0}а {1} ноги'.format(my_dog.get_name(), my_dog.legs))
# Создаем еще одну конкретную собаку
her_dog = Dog("Бобик")
print('У {0}а {1} ноги'.format(her_dog.get_name(), her_dog.legs))
# Сколько экземпляров класса Dog создано:
print('создано собак:', Dog.count)
```

Контрольные задания

1. Создайте класс StringVar для работы со строковым типом данных, содержащий методы set и get. Метод set служит для изменения содержимого строки, get – для получения содержимого строки. Создайте объект типа StringVar и протестируйте его методы.
2. Создайте класс точка Point, позволяющий работать с координатами (x, y). Добавьте необходимые методы класса.

2.9. Иерархия наследования в Python

В Python все создаваемые классы наследуются от класса object. Создадим класс Point, в котором определим (переопределим методы базового класса object) специальные методы __init__, __eq__, __str__:

```
class Point:
   def __init__(self, x=0, y=0): # устанавливаем координаты точки
      self.x = x
      self.y = y
   def __eq__(self, other): # выполняет сравнение двух точек
      return self.x == other.x and self.y == other.y
   def __str__(self): # выполняет строковый вывод информации
      return "({0.x}, {0.y})".format(self)

a = Point()    # создаем объект, по умолчанию координаты: x=0, y=0
print(str(a))   # вызывается метод __str__ класса Point,
                # полная форма вызова метода: Point.__str__(a)
b = Point(3, 4)
print(str(b))
b.x = -19
print(str(b))
print(a == b, a != b)   # вызывается метод __eq__
# полная форма для сравнения a == b имеет вид: Point.__eq__(a, b)
```

Результат выполнения программы:

```
(0, 0)
(3, 4)
(-19, 4)
False True
```

В Python за каждой операцией над объектами закреплен специальный метод[59], например в момент сложения (+) вызывается метод __add__. Заметим, что в предыдущем примере не переопределялся метод __ne__ для

[59] см. https://docs.python.org/3/reference/datamodel.html#emulating-numeric-types

неравенства a != b, но Python смог выполнить сравнение, т.к. принял его результат за обратный к равенству (вернул противоположный результат вызова метода __eq__).

Наследуем от класса Point класс Circle (рис. 10).

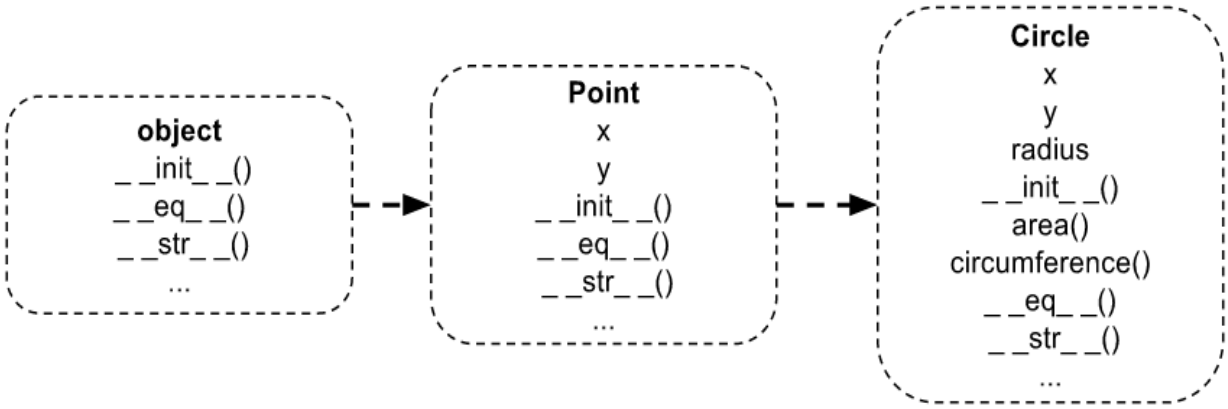


Рис. 10. Пример иерархии наследования классов

Исходный код класса Circle:

```
class Circle(Point):
    def __init__(self, radius, x=0, y=0):
        super().__init__(x, y) # вызов конструктора базового класса
        self.radius = radius
    def area(self): # вычисляет площадь окружности
        return math.pi * (self.radius ** 2)
    def circumference(self):  # вычисляет длину окружности
        return 2 * math.pi * self.radius
    def __eq__(self, other): # сравнивает две окружности
        return self.radius == other.radius and super().__eq__(other)
    def __str__(self):  # выводит информацию в виде строки
        return "({0.radius}, {0.x}, {0.y})".format(self)

circle = Circle(2) # создаем объект, radius=2, x=0, y=0
circle.radius = 3
circle.x = 12
a = Circle(4, 5, 6)
b = Circle(4, 5, 6)
print(str(a))    # вызывается специальный метод __str__
print(str(b))
print(a == b)  # вызывается специальный метод __eq__
# полная форма вызова метода для a == b: Circle. __eq__(a, b)
print(a == circle)
print(a != circle) # отрицание результата вызова метода __eq__
```

Результат выполнения программы:

```
(4, 5, 6)
(4, 5, 6)
True
False
True
```

2.10. Документирование и тестирование функций на языке Python

Вспомним, что для получения документации ранее вызывалась функция help. Данная функция считывает строку (многострочный комментарий в тройных кавычках), содержащуюся внутри тела функции:

```
def my_function():
    """Эта функция ничего не делает.
    """
    pass
help(my_function)
```

Результат выполнения программы:

```
Help on function my_function in module __main__:

my_function()
    Эта функция ничего не делает.
```

Помимо справочной информации, в комментариях к функции могут содержаться тесты для автоматизированной проверки правильности вычислений[60]:

```
def func_m(v1, v2, v3):
    """Вычисляет среднее арифметическое трех чисел.

    >>> func_m(20, 30, 70)
    40.0

    >>> func_m(1, 5, 8)
    4.667

    """
    return round((v1 + v2 + v3) / 3, 3)

import doctest
```

[60] см. https://docs.python.org/3/library/doctest.html

```
# автоматически проверяет тесты в документации к функции
doctest.testmod()
```

В результате запуска программы на экране отобразится:

```
TestResults(failed=0, attempted=2)
```

Если в первом тесте заменить аргумент 70 на 71, то произойдет ошибка:

```
**********************************************************
File "__main__", line 4, in __main__.func_m
Failed example:
   func_m(20, 30, 71)
Expected:
   40.0
Got:
   40.333
**********************************************************
```

ГЛАВА 3. ПРИМЕНЕНИЕ ВОЗМОЖНОСТЕЙ ЯЗЫКА PYTHON ДЛЯ РЕШЕНИЯ ЗАДАЧ

3.1. Сравнение времени работы алгоритмов поиска

Определим несколькими способами позиции двух наименьших элементов в неотсортированном списке, для этого на языке Python создадим три функции [1].

1. Поиск индекса минимального элемента в списке, удаление найденного элемента, повторный поиск минимального элемента, возвращение удаленного элемента в список:

```
def find_two_smallest_1(L):
    smallest = min(L)
    min1 = L.index(smallest)
    L.remove(smallest)   # удаляем первый минимальный элемент

    next_smallest = min(L)
    min2 = L.index(next_smallest)
    L.insert(min1, smallest) # возвращаем первый минимальный обратно
    if min1 <= min2: # проверяем индекс второго минимального
        min2 += 1     # min2 = min2 + 1

    return (min1, min2)  # возвращаем кортеж
```

2. Сортировка списка, поиск минимальных элементов, определение их индексов:

```
def find_two_smallest_2(L):
    temp_list = sorted(L)  # возвращаем копию отсортированного списка
    smallest = temp_list[0]
    next_smallest = temp_list[1]
    min1 = L.index(smallest)
    min2 = L.index(next_smallest)
    return (min1, min2)
```

3. Сравнение каждого элемента по порядку:

```
def find_two_smallest_3(L):
    if L[0] < L[1]:
        min1, min2 = 0, 1  # устанавливаем начальные значения
    else:
        min1, min2 = 1, 0
    for i in range(2, len(L)):
        if L[i] < L[min1]:
            min2 = min1
```

```
            min1 = i
        elif L[i] < L[min2]:
            min2 = i
    return (min1, min2)
```

Время работы алгоритмов измерим с помощью встроенной в JupyterLab команды:

```
In [1]: %time?

Docstring:
Time execution of a Python statement or expression.
…
```

Создадим три списка A1, A2 и A3, состоящих из случайных элементов:

```
In [2]: from random import randint
In [3]: A1 = [randint(1, 1000) for i in range(50000)]
In [4]: A2 = [randint(1, 1000) for i in range(100000)]
In [5]: A3 = [randint(1, 1000) for i in range(300000)]
```

Пример вычисления времени работы алгоритма find_two_smallest_1 для списка A1:

```
In   [6]: %time find_two_smallest_1(A1)
          Wall time: 4.01 ms
Out [6]: (250, 2158)
```

Результаты расчета времени представлены в табл. 1.

Таблица 1

Расчет времени работы алгоритмов поиска

	A1	**A2**	**A3**
find_two_smallest_1	4.01 ms	7.01 ms	22 ms
find_two_smallest_2	21 ms	41 ms	154 ms
find_two_smallest_3	15 ms	28 ms	89.1 ms

Контрольные задания

1. Напишите функцию, которая возвращает разность между наибольшим и наименьшим значениями из списка целых случайных чисел. Определите время работы алгоритма для различных входных данных.

2. Напишите программу, которая для целочисленного списка из 1000 случайных элементов определяет, сколько отрицательных элементов

располагается между его максимальным и минимальным элементами. Определите время работы алгоритма для различных входных данных.

3.2. Построение графиков с помощью matplotlib

Matplotlib[61] является пакетом для визуализации двумерной графики[62] на Python. Для демонстрации примера работы воспользуемся модулем pyplot[63], обеспечивающим MATLAB-подобный интерфейс [2, 4]:

```
# Подключение возможностей пакета NumPy
# для работы с однородными многомерными массивами:
import numpy as np
# Команда для вывода интерактивного графика в окне браузера:
%matplotlib notebook
# Подключение возможностей пакета matplotlib:
import matplotlib.pyplot as plt
# Генерация последовательности чисел от -10 до 10 с шагом 100,
# возвращает NumPy массив (значения по оси X):
x = np.linspace(-10, 10, 100)
# Создаем второй массив (значения по оси Y):
y = np.sin(x)
#  Создание линейного графика на основе двух массивов:
plt.plot(x, y)
```

Результат выполнения программы представлен на рис. 11.

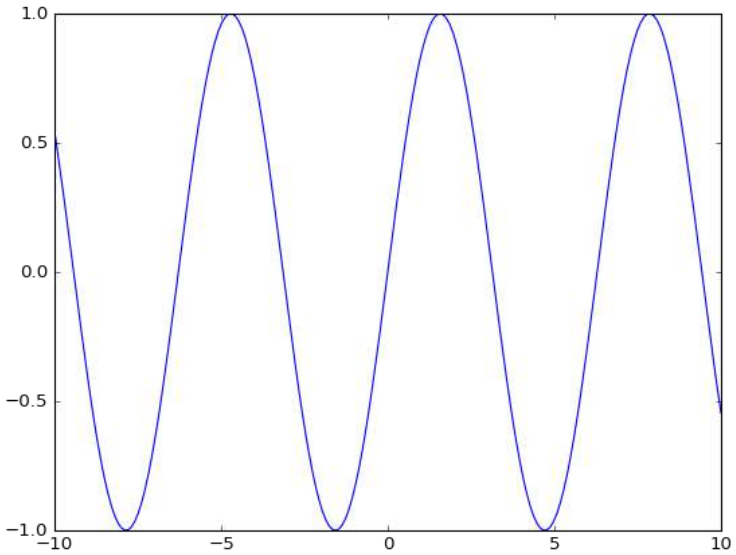


Рис. 11. График функции sin(x)

[61] см. http://matplotlib.org
[62] см. http://matplotlib.org/gallery.html
[63] см. http://matplotlib.org/users/pyplot_tutorial.html

Построим графики зависимости времени работы алгоритмов поиска (см. п. 2.15) от числа входных элементов[64]:

```
# Команда для вывода графика в окне браузера:
%matplotlib notebook
# Подключение возможностей пакета matplotlib:
import matplotlib.pyplot as plt
plt.plot([4.01, 7.01, 22], color="blue",
        linewidth=2.5, linestyle="-",
        label="find_two_smallest_1")
plt.plot([21, 41, 154], color="red",
        linewidth=2.5, linestyle="-",
        label="find_two_smallest_2")
plt.plot([15, 28, 89.1], color="green",
        linewidth=2.5, linestyle="-",
        label="find_two_smallest_3")
plt.legend(loc='upper left', frameon=False)
```

Результат выполнения программы представлен на рис. 12.

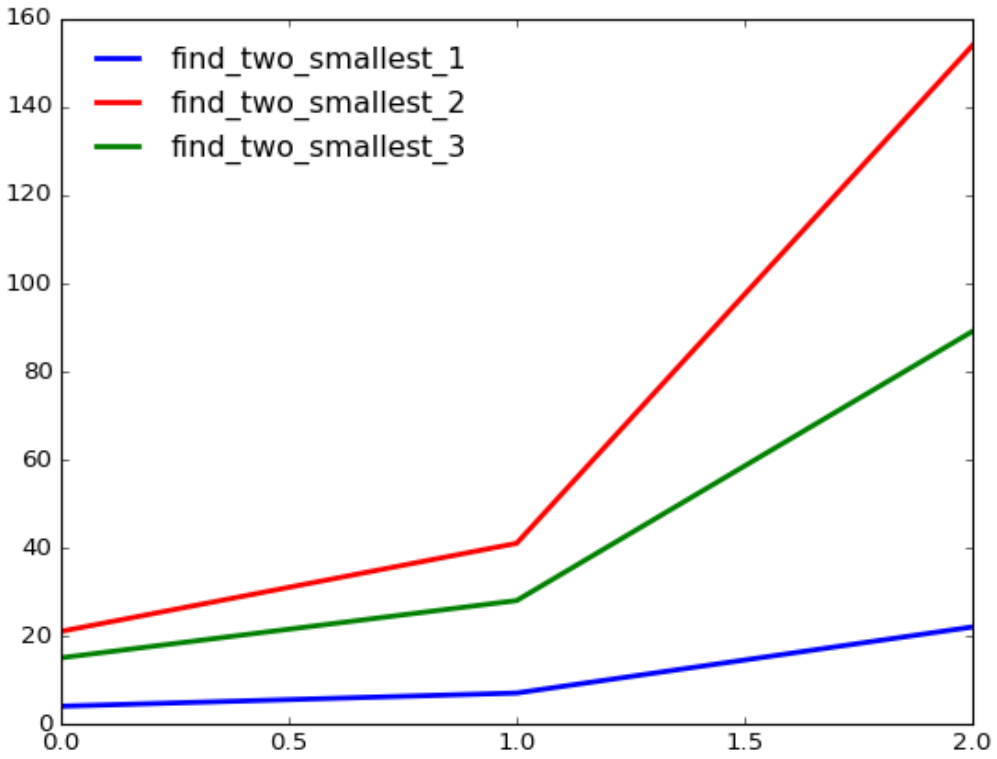


Рис. 12. Графики зависимости времени работы алгоритмов поиска от числа входных элементов

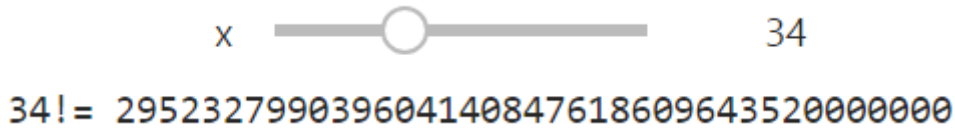
[64] см. http://matplotlib.org/api/pyplot_api.html

3.3. Интерактивные виджеты в Jupyter Lab

Выполним в Jupyter Lab следующий код:

```
import numpy as np # импортируем модуль numpy
from IPython.html.widgets import interact
def factorial(x) :
    f = np.math.factorial(x)
    print(str(x) + '!= ' + str(f))
i = interact(factorial , x=(0,100))
```

В блокноте отобразится интерактивный виджет[65]:



Он работает только при запущенном блокноте.

3.4. Создание графического интерфейса с помощью tkinter

Язык Python позволяет создавать приложения с графическим интерфейсом, для этого применяются различные графические библиотеки[66]. Остановимся на рассмотрении стандартной графической библиотеки tkinter[67].

Первым делом при работе с tkinter необходимо создать главное (корневое) окно (рис. 13), в котором размещаются остальные графические элементы – *виджеты*. Существует большой набор виджетов[68] на все случаи жизни: для ввода текста, вывода текста, выпадающие меню и т.д. Среди виджетов есть кнопка, при нажатии на которую происходит заданное событие. Некоторые виджеты (фреймы) используются для группировки других виджетов внутри себя.

Приведем пример простейшей программы для отображения главного окна:

```
# Подключаем модуль, содержащий методы для работы с графикой
import tkinter
# Создаем главное (корневое) окно,
```

[65] см. https://ipywidgets.readthedocs.io/en/latest/
[66] см. https://wiki.python.org/moin/GuiProgramming
[67] см. https://docs.python.org/3/library/tkinter.html
[68] см. http://effbot.org/tkinterbook/tkinter-index.htm

```
# в переменную window записываем ссылку на объект класса Tk
window = tkinter.Tk()
# Задаем обработчик событий для корневого окна
window.mainloop()
```

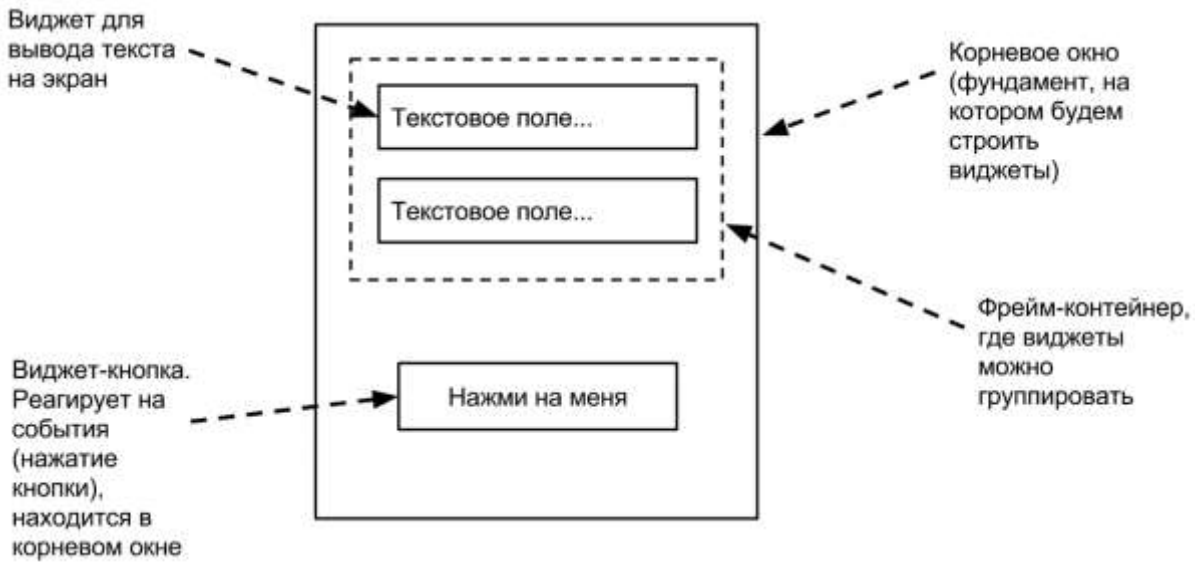
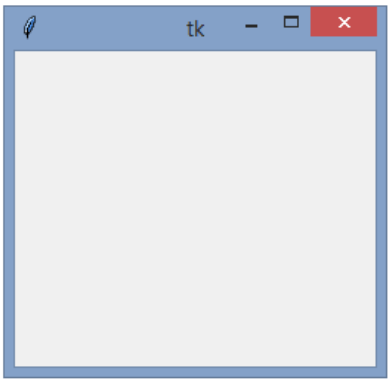


Рис. 13. Схема главного окна в tkinter

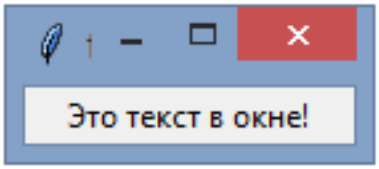
Результат выполнения программы:



Следующий пример демонстрирует создание виджета Label:

```
import tkinter
window = tkinter.Tk()
# Создаем объект-виджет класса Label в корневом окне window
# text – параметр для задания отображаемого текста
label = tkinter.Label(window, text = "Это текст в окне!")
# Отображаем виджет с помощью менеджера pack
label.pack()
window.mainloop()
```

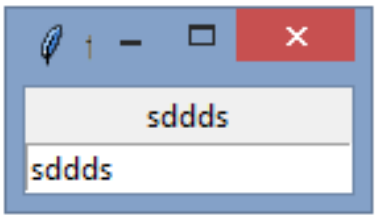
В результате работы программы отображается графическое окно с текстом внутри:



Следующий пример показывает, каким образом используется виджет Entry для ввода данных:

```
import tkinter
window = tkinter.Tk()
frame = tkinter.Frame(window)
frame.pack()
var = tkinter.StringVar()
# Обновление содержимого переменной в момент ввода текста
label = tkinter.Label(frame, textvariable=var)
label.pack()
# Пробуем набрать текст в появившемся поле для ввода
entry = tkinter.Entry(frame, textvariable=var)
entry.pack()
window.mainloop()
```

Запустим программу и попробуем набрать произвольный текст:



Видим, что текст, который мы набираем, сразу отображается в окне. Дело в том, что виджеты Label и Entry используют для вывода и ввода текста соответственно одну и ту же переменную data класса StringVar[69]. Подобная схема работы оконного приложения укладывается в универсальный шаблон (паттерн), который называется «Модель-вид-контроллер» (Model-View-Controller или MVC[70]). В общем виде под *моделью* (Model) понимают способ хранения данных, т.е. как данные хранятся (например, в переменной

[69] Tkinter поддерживает работу с переменными классов: BooleanVar, DoubleVar, IntVar, StringVar.

[70] Паттерн MVC получил широкое распространение при разработке веб-приложений.

какого класса). *Вид* (View) служит для отображения данных. *Контроллер* (Controller) отвечает за обработку данных (рис. 14).

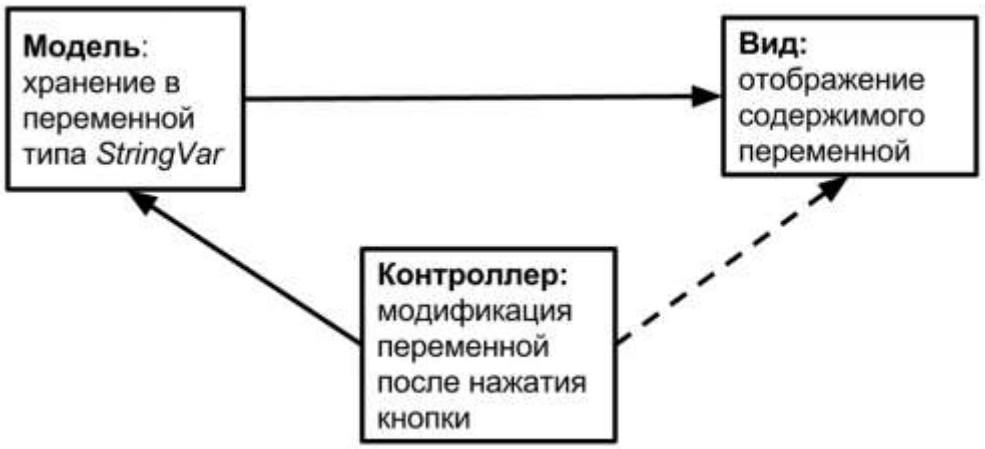


Рис. 14. Пример шаблона MVC

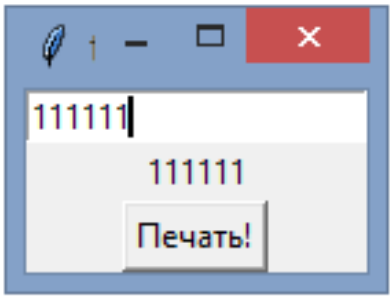
Интересная особенность MVC в том, что в случае изменения контроллером данных (как это было в предыдущем примере с изменением переменной var), «посылается сигнал» *виду* с просьбой обновить отображаемое содержимое (перерисовать окно), отсюда получается обновление текста в режиме реального времени.

В следующем примере введенный текст (виджет Entry) отображается на экране (виджет Label) только в момент нажатия кнопки (виджет Button):

```
import tkinter
# Вызывается в момент нажатия на кнопку:
def click():
    # Получаем строковое содержимое поля ввода и
    # с помощью config изменяем отображаемый текст
    label.config(text=entry.get())

window = tkinter.Tk()
frame = tkinter.Frame(window)
frame.pack()
entry = tkinter.Entry(frame)
entry.pack()
label = tkinter.Label(frame)
label.pack()
# Привязываем обработчик нажатия на кнопку к функции click
button = tkinter.Button(frame, text='Печать!', command=click)
button.pack()
window.mainloop()
```

Результат выполнения программы:



Контрольные задания

1. Напишите программу с графическим интерфейсом, которая позволяет произвольный текст, введенный с клавиатуры, сохранить в обычный текстовый файл либо в файл HTML-формата.
2. Напишите программу-генератор паролей с графическим интерфейсом. В качестве входных параметров генератора можно указать:
   - наличие цифр;
   - наличие прописных букв;
   - наличие строчных букв;
   - наличие специальных символов %, *,),?, @, #, $, ~;
   - длину пароля.

3.5. Клиент-серверное программирование на языке Python

Python содержит стандартные модули[71], позволяющие обратиться к удаленному веб-серверу по протоколу прикладного уровня HTTP[72]:

```
In  [1]: import urllib.request as ur
         url = "http://www.ya.ru:80"
         conn = ur.urlopen(url)
         print(conn)

Out [1]: <http.client.HTTPResponse object at 0x000000D156A03C18>

In  [2]: data = conn.read()
         print(data)
```

[71] см. https://docs.python.org/3/library/http.client.html
[72] Рекомендую видео о протоколе TCP/IP: https://www.youtube.com/watch?v=v-rIM2YvTfA

```
Out [2]: b'<!DOCTYPE html><html class="i-ua_js_no i-ua_css_standart i-
         ua_browser_ i-ua_browser_desktop i-ua_platform_other"
         lang="ru"><head xmlns:og="http://ogp.me/ns#"><meta http-
         equiv=Content-Type content="text/html;charset=UTF-8"><meta…..

In   [3]: print(conn.status)
Out [3]: 200
```

Переменная conn является объектом класса HTTPResponse, метод read предоставляет информацию о веб-странице, status содержит код статуса HTTP-ответа.

Предположим, что необходимо передать данные от персонального компьютера 1 (ПК1-клиента) к ПК (серверу), расположенным в одной сети.

Для идентификации ПК в сети применяются IP-адреса, например, 192.168.0.3. На ПК работает большое число сетевых приложений (Skype, Telegram и пр.), поэтому, чтобы ПК определить, для какого приложения поступили данные, необходимо каждому сетевому приложению присвоить уникальный номер – *сетевой порт* (например, Skype использует 80 и 443 порты). Связка «IP-адрес, сетевой порт» называется *сокетом* (*socket*). Сокеты предоставляют программный интерфейс для сетевого взаимодействия. Впервые они были реализованы на языке C в системе BSD. Python имеет встроенный модуль socket[73].

Сетевое взаимодействие происходит посредством клиент-серверного обмена данными, где клиент – запрашивает (отправляет) данные, сервер – обрабатывает данные, полученные от клиента. Например, веб-клиентом является браузер, а веб-сервером – удаленный ПК, способный обрабатывать HTTP-запросы, поступающие от браузера.

Рассмотрим пример серверного и клиентского приложений, написанных на языке Python: сначала запустить программу-сервер, а затем в другом ядре JupyterLab запустить программу-клиент. Клиент-серверное взаимодействие в нашем примере будет происходить на одном и том же ПК, поэтому в качестве IP-адрес указываем 127.0.0.1[74].

Исходный код сервера[75], который обрабатывает поступающие запросы от клиента:

```
import socket # подключаем модуль для взаимодействия по сети

HOST = '127.0.0.1' # IP-адрес для клиент-серверного обмена на одном ПК
PORT = 50007       # порт идентифицирует программу-сервер на данном ПК
```

---

[73] см. https://docs.python.org/3/howto/sockets.html
[74] см. https://en.wikipedia.org/wiki/Localhost
[75] https://github.com/dm-fedorov/python-lessons/blob/master/labs/lab3_server/easy_server.py

```
# создается программный сокет с гарантированной (SOCK_STREAM)
# доставкой данных (протокол TCP):
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# сокет привязывается (bind) к IP-адресу и сетевому порту для того,
# чтобы обрабатывать поступающие запросы:
s.bind((HOST, PORT))
# сервер слушает (listen), ожидает входные соединения от клиента:
s.listen(1)
# в момент, когда от клиента поступил запрос на соединение, вызывается
# метод accept, который приводит к созданию нового сокета
#(записывается в переменную conn). Данную операцию можно сравнить с
# поступлением телефонного звонка на коммутатор (listen), который
# перенаправляет звонок к конкретному оператору (accept) и снова
# переходит в режим ожидания:
conn, addr = s.accept() # в переменной addr IP-адрес клиента
print('Connected client')
while True:
   data = conn.recv(1024) # получение данных от клиента, 1024 байт
   if not data:
      break
   else:
      print('Received[2]: ', data)
   conn.send(data) # отправка данных клиенту
   print('Send[3]: ', data)
conn.close()  # закрытие соединения
```

Исходный код клиента[76] (необходимо выполнить в отдельном ядре JupyterLab Notebook):

```
import socket

HOST = '127.0.0.1' # IP-адрес сервера
PORT = 50007       # порт сервера
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# клиент устанавливает соединение с сервером:
s.connect((HOST, PORT))
data = 'Hello world'
# обмен по сети происходит в формате bytes, поэтому строку перед
# передачей ее серверу, преобразуем:
s.send(data.encode('utf-8'))
print('Send[1]: ', data)
# получение данных от сервера:
data = s.recv(1024)
s.close()
print('Received[4]: ', data)
```

[76] https://github.com/dm-fedorov/python-lessons/blob/master/labs/lab3_server/easy_client.py

Результат работы программы на стороне сервера:

```
Connected client
Received[2]:  b'Hello world'
Send[3]:  b'Hello world'
```

Результат работы программы на стороне клиента:

```
Send[1]:  Hello world
Received[4]:  b'Hello world'
```

Контрольное задание

Разработайте распределенную систему мониторинга удаленных ПК. Каждый ПК содержит программу-агент, который собирает информацию о текущем состоянии системы, например, контроль запуска определенных служб (выбираются на усмотрение разработчика, можно реализовать выбор службы для мониторинга через конфигурационный файл). Необходимо задействовать максимальные возможности Python для работы с операционными системами.

На ПК производится сбор основных действий агента и результатов мониторинга (время, состояние и пр.). Через определенные интервалы времени агенты отправляют информацию на центральный сервер мониторинга. Сервер мониторинга опрашивает агентов, в ответ получает информацию о текущем состоянии системы. На сервере мониторинга производится логирование (запись) основных действий и результатов сбора информации (IP-адрес хоста, время и пр.). Итоговый результат сбора информации представляется в виде таблицы и/или графика.

При реализации системы необходимо задействовать возможности модулей языка программирования Python (os, xmlrpclib и пр.). В качестве хранилища данных можно использовать текстовые файлы (JSON) или СУБД (MySQL, SQLite).

3.6. Использование возможностей языка Python для обработки естественного языка

Рассмотрим некоторые возможности пакета библиотек Natural Language Toolkit (nltk) для символьной и статистической обработки естественного языка. Для установки nltk потребуется выполнить следующую команду в Jupyter Lab:

```
!pip install nltk
```

Загрузить данные для конкретных задач можно следующим образом:

```
import nltk
nltk.download()
```

Полный пакет с данными для nltk занимает 2,2 Гб на жестком диске.

Рассмотрим простейший процесс выделения лексем (или сегментирование). Можно интерпретировать по-разному, но интересны слова и предложения:

```
text = "Меня беспокоит, что в этом будущем конгломерате, между людьми, которыми мы являемся, \
и теми существами, которыми мы станем через 4–5 поколений, будет мала человеческая доля. \
Поэтому единственный для нас способ — это не остановить прогресс, а возглавить его."
```

Самый простой вариант – разбить текст на лексемы с помощью метода split:

```
text = text.split()
print(text)
```

['Меня', 'беспокоит,', 'что', 'в', 'этом', 'будущем', 'конгломерате,', 'между', 'людьми,', 'которыми', 'мы', 'являемся,', 'и', 'теми', 'существами,', 'которыми', 'мы', 'станем', 'через', '4–5', 'поколений,', 'будет', 'мала', 'человеческая', 'доля.', 'Поэтому', 'единственный', 'для', 'нас', 'способ', '—', 'это', 'не', 'остановить', 'прогресс,', 'а', 'возглавить', 'его.']

Видим, что встроенный split не учитывает знаки пунктуации. nltk позволяет это исправить:

```
from nltk import word_tokenize
text = "Меня беспокоит, что в этом будущем конгломерате, между людьми, которыми мы являемся, \
и теми существами, которыми мы станем через 4–5 поколений, будет мала человеческая доля. \
Поэтому единственный для нас способ — это не остановить прогресс, а возглавить его."
words = word_tokenize(text)
print(words)
```

['Меня', 'беспокоит', ',', 'что', 'в', 'этом', 'будущем', 'конгломерате', ',', 'между', 'людьми', ',', 'которыми', 'мы', 'являемся', ',', 'и', 'теми', 'существами', ',', 'которыми', 'мы', 'станем', 'через', '4–5', 'поколений', ',', 'будет', 'мала', 'человеческая', 'доля', '.', 'Поэтому', 'единственный', 'для', 'нас', 'способ', '—', 'это', 'не', 'остановить', 'прогресс', ',', 'а', 'возглавить', 'его', '.']

nltk позволяет разбить текст на отдельные предложения:

```
from nltk import word_tokenize, sent_tokenize
text = "Меня беспокоит, что в этом будущем конгломерате, между людьми, которыми мы являемся, \
и теми существами, которыми мы станем через 4–5 поколений, будет мала человеческая доля. \
Поэтому единственный для нас способ — это не остановить прогресс, а возглавить его."

sentences = sent_tokenize(text)
print(sentences)
```

['Меня беспокоит, что в этом будущем конгломерате, между людьми, которыми мы являемся, и теми существами, которыми мы станем через 4–5 поколений, будет мала человеческая доля.', 'Поэтому единственный для нас способ — это не остановить прогресс, а возглавить его.']

nltk умеет отличать год от окончания предложения:

```
from nltk import word_tokenize, sent_tokenize
text = "Пилотируемый полет на Марс, вероятно, состоится в 2030-х г., \
после того как будет создана необходимая инфраструктура для такой экспедиции. \
Об этом заявил директор NASA Джим Брайденстайн в ходе брифинга по итогам посадки \
на Марсе автоматической станции Mars InSight."

sentences = sent_tokenize(text)
print(sentences)
```

['Пилотируемый полет на Марс, вероятно, состоится в 2030-х г., после того как будет создана необходимая инфраструктура для такой экспедиции.', 'Об этом заявил директор NASA Джим Брайденстайн в ходе брифинга по итогам посадки на Марсе автоматической станции Mars InSight.']

Обычно слова используются в определенной последовательности. Можем получить последовательности из n-смежных слов, которые называются n-граммами. При n=1 получим униграммы, далее – биграммы, триграммы и т.д.

```
from nltk.util import ngrams
text = "Меня беспокоит, что в этом будущем конгломерате, между людьми, которыми мы являемся, \
```

```
и теми существами, которыми мы станем через 4–5 поколений, будет мала человеческая доля. \
Поэтому единственный для нас способ — это не остановить прогресс, а возглавить его."

text = text.split()
print(text)
```

['Меня', 'беспокоит,', 'что', 'в', 'этом', 'будущем', 'конгломерате,', 'между', 'людьми,', 'которыми', 'мы', 'являемся,', 'и', 'теми', 'существами,', 'которыми', 'мы', 'станем', 'через', '4–5', 'поколений,', 'будет', 'мала', 'человеческая', 'доля.', 'Поэтому', 'единственный', 'для', 'нас', 'способ', '–', 'это', 'не', 'остановить', 'прогресс,', 'а', 'возглавить', 'его.']

```
print(list(ngrams(text, 3)))
```

[('Меня', 'беспокоит,', 'что'), ('беспокоит,', 'что', 'в'), ('что', 'в', 'этом'), ('в', 'этом', 'будущем'), ('этом', 'будущем', 'конгломерате,'), ('будущем', 'конгломерате,', 'между'), ('конгломерате,', 'между', 'людьми,'), ('между', 'людьми,', 'которыми'), ('людьми,', 'которыми', 'мы'), ('которыми', 'мы', 'являемся,'), ('мы', 'являемся,', 'и'), ('являемся,', 'и', 'теми'), …

Выделение основ слов (стемминг) – обычно отсечение суффикса:

```
from nltk import SnowballStemmer
stemmer = SnowballStemmer('russian')
print(stemmer.stem('рыбаки'))
```
'рыбак'

```
print(stemmer.stem('рыбаков'))
```
'рыбак'

```
print(stemmer.stem('рыбаками'))
```
'рыбак'

3.7. Использование возможностей Python для обработки изображений

Pillow[77] — библиотека языка Python (версии 3), предназначенная для работы с растровой графикой.

Открываем изображение для просмотра:

[77] см. https://pillow.readthedocs.io/en/stable/

```
from PIL import Image

image = Image.open('cat.jpg')
image.show()
```

Здесь импортируем модуль Image и просим его открыть наш файл. В ОС Unix метод open сохраняет изображения во временный файл PPM и открывает его с помощью утилиты xv (или ImageMagick). В Windows он сохранит изображение как временный BMP и откроет его в чем-то вроде Paint.

Вы можете получить много информации об изображении, используя Pillow. Давайте рассмотрим лишь несколько небольших примеров того, что можно извлечь:

```
print(image.split())
```

```
(<PIL.Image.Image image mode=L size=840x472 at 0x10838B6D8>, <PIL.Image.Image image mode=L size=840x472 at 0x1080E4CC0>, <PIL.Image.Image image mode=L size=840x472 at 0x10838B710>)
```

```
%matplotlib inline
import matplotlib.pyplot as plt

histogram = image.histogram()
plt.plot(histogram) # значения по Y
plt.show()
```

Можно обрезать изображения с помощью Pillow. Давайте попробуем обрезать нашу картинку:

```
cropped = image.crop((100, 150, 250, 250))
cropped.save('copy_cat.png')
```

Достаточно открыть изображение и затем вызвать его метод crop. Необходимо передать координаты x / y, которые вы хотите обрезать, то есть (x1, y1, x2, y2). В Pillow 0 – это верхний левый пиксель. При увеличении значения x, оно идет вправо. При увеличении значения y вы идете вниз по изображению.

Существуют различные фильтры, которые вы можете использовать в Pillow, чтобы применить к изображениям. Они содержатся в модуле ImageFilter. Давайте посмотрим на пару из них здесь:

```
from PIL import ImageFilter
fil = image.filter(ImageFilter.BLUR) # размывание
fil.save('blur_cat.png')
```

Вот один из способов повышения резкости (SHARPEN) изображения:

```
sh = image.filter(ImageFilter.SHARPEN) # резкость
sh.save('sh_cat.png')
```

3.8. Использование возможностей языка Python для решения задач анализа данных

На сегодняшний день язык Python входит в число лидеров по использованию в области анализа данных. Это стало возможным благодаря продуманной экосистеме, построенной на основе стандартных структур данных языка Python и модуля NumPy (тип данных array) [2, 4].

Рассмотрим с помощью среды Jupyter Lab некоторые возможности модуля pandas, который входит в экосистему анализа данных. Основная структура данных в этом модуле – кадр данных (Data Frame) представляет собой матрицу в памяти:

```
df = pd.DataFrame(grid)
print(df)
```

```
   0  1  2
0  1  2  3
1  4  5  6
2  7  8  9
```

Вместо номеров можно присвоить столбцам имена:

```
df = pd.DataFrame(grid, columns=["one", "two", "three"] )
print(df)
```

```
   one  two  three
0   1    2     3
1   4    5     6
2   7    8     9
```

Теперь можем обратиться по имени столбца:

```
print(df["two"])
```

```
0   2
1   5
2   8
Name: two, dtype: int64
```

Рассмотрим пример набора данных *Bank Marketing Data Set*[78]. Данные связаны с кампаниями прямого маркетинга (телефонными звонками) португальского банка. С помощью pandas можно напрямую читать файл в формате CSV (указывается разделитель) и преобразовать его к типу данных Data Frame:

```
temp = pd.read_csv('bank/bank-full.csv', delimiter=';')
temp.head()
```

	age	job	marital	education	default	balance
0	58	management.	married	tertiary	no	2143
1	44	technician	single	secondary	no	29
…						

Следующим шагом выведем на экран первые пять строк столбца job, для которых столбец marital содержит строку 'single':

```
temp[temp['marital'] == 'single'].job[:5]
```

```
1    technician
4    unknown
6    management
9    technician
11   admin.
Name: job, dtype: object
```

Далее узнаем все уникальные значения, которые содержит столбец job:

```
temp[temp['marital'] == 'single'].job.unique()
```

```
array(['technician', 'unknown', 'management', 'admin.', 'blue-collar',
       'entrepreneur', 'retired', 'unemployed', 'services',
       'self-employed', 'student', 'housemaid'], dtype=object)
```

3.9. Применение языка Python в области искусственного интеллекта

С помощью Python можно решать задачи в области ИИ. Приведем лишь перечень модулей, которые позволяют это делать.

Разработанная компанией Google, Tensor Flow – это надежная платформа с открытым исходным кодом, поддерживающая глубокое обучение.

Microsoft CNTK – это быстрый и универсальный фреймворк с открытым исходным кодом, основанный на нейронных сетях.

---

[78] см. https://archive.ics.uci.edu/ml/datasets/bank+marketing

3.10. Интеграция языков программирования на примере написания расширений для языка Python

На самом деле, Python – это спецификация того, как должен выглядеть язык программирования. Реализовывать эту спецификацию можно на любом другом языке программирования: C, Java, C#, Ruby и т.д. В пособии используется CPython – реализация на языке C, которая поддерживается официальными разработчиками языка. CPython предоставляет API (Application Programming Interface) для разработчиков[79], желающих написать собственный модуль на языке C [3].

3.11. Создание модуля Python на языке C

Рассмотрим процесс написания расширения (модуля) Python на языке программирования C[80].

Далее пример реализации модуля (plus.c), который содержит только одну функцию сложения двух чисел, переданных ей в качестве аргумента[81]:

```
#include <Python.h>

static PyObject *py_plus(PyObject *self, PyObject *args) {
  double x, y, result;

  if (!PyArg_ParseTuple(args,"dd", &x, &y)) {
    return NULL;
  }
  result = x + y;
  return Py_BuildValue("d", result);
}

static PyMethodDef ownmod_methods[] = {
{
        "plus",         // название функции внутри Python
         py_plus,      // функция C
         METH_VARARGS, /* макрос, поясняющий, что функция у нас с аргументами */
         "plus function" // документация для функции внутри Python
    },
```

---

[79] см. https://docs.python.org/3.6/extending/index.html#extending-index
см. https://docs.python.org/3.6/c-api/

[80] Данный раздел написан при участии обучающихся СПбГЭУ направления «Прикладная математика и информатика» Ю.С. Макарова и Д.Ю. Митюры.

[81] см. https://github.com/lvoursl/python-and-c-integration/blob/master/Functions-module/plus-function/plus.c

```
    { NULL, NULL, 0, NULL } /* так называемый sentiel. Сколько бы элементов структуры у вас не было, этот нулевой элемент должен быть всегда, и при этом быть последним */
};

static PyModuleDef simple_module = {
    /* Описываем наш модуль */
    PyModuleDef_HEAD_INIT, // обязательный макрос
    "my_plus", // my_plus.__name__
    "amazing documentation", // my_plus.__doc__
    -1,
    ownmod_methods // сюда передаем методы модуля
};

// В названии функции обязательно должен быть префикс PyInit
PyMODINIT_FUNC PyInit_my_plus(void) {
    PyObject* m;
    // создаем модуль
    m = PyModule_Create(&simple_module);
    // если все корректно, то эта проверка не проходит
    if (m == NULL)
        return NULL;
    return m;
}
```

В первой строке импортируется заголовочный файл Python.h. Он содержит API CPython. Его необходимо подключать перед другими модулями, т.к. он содержит в себе директивы препроцессора, которые могут оказать влияние на стандартные заголовочные файлы. Далее объявляется функция, складывающая два числа. Стоит обратить внимание на тип возвращаемого значения, а также аргументы функции. PyObject[82] – это C-структура, являющаяся базовой для всех остальных объектов. Все функции в будущем модуле, должны иметь данный тип возвращаемого значения.

self и args – обязательные аргументы функции. self – указатель на объект самого модуля, если данная функция принадлежит модулю, или указатель на объект, если это функция какого-либо класса. args – это кортеж аргументов функции, переданный прямиком Python. Например, если вызвать simple.plus(1, 3), то в этом случае args будет состоять из чисел (1, 3).

В теле функции объявляются три переменные, которые будут в дальнейшем использованы в функции PyArg_ParseTuple. Остановимся на ней подробнее. Она используется для обработки аргументов функции и позволяет трансформировать типы данных Python в типы данных C. В качестве первого аргумента она принимает PyObject, который, по сути, является кортежем переданных функции аргументов, вторым – типы переменных, в

[82] см. https://docs.python.org/3.6/c-api/structures.html#c.PyObject

которые мы хотим преобразовать наши аргументы. Далее идут переменные, в которые сохраняем преобразованные значения. Несколько примеров использования этой функции:

```
PyArg_ParseTuple(args, "s", &some_string) /* проверить, содержит ли массив args один элемент строкового типа, если да – сохранить это значение в переменную some_string */
PyArg_ParseTuple(args, "i", &some_int) // аналогично для числа
```

Далее вычисляем значение переменной result, и используем функцию Py_BuildValue[83], которая позволяет конвертировать переменные C в переменные соответствующих Python-типов. Примеры работы данной функции представлены в табл. 2.

Таблица 2

Соответствие между вызовами функции Py_BuildValue и возвращаемыми типами данных Python

Пример вызова функции	**Результат конвертирования**
Py_BuildValue("")	None
Py_BuildValue("i", 123)	123
Py_BuildValue("iii", 123, 456, 789)	(123, 456, 789)
Py_BuildValue("s", "hello")	'hello'
Py_BuildValue("y", "hello")	b'hello'
Py_BuildValue("ss", "hello", "world")	('hello', 'world')

Для того чтобы преобразовать C-программу в Python-модуль потребуется создать файл setup.py в той же директории, где лежит C-программа. Файл должен иметь следующее содержание[84]:

```
from distutils.core import setup, Extension

module1 = Extension(
        'my_plus',              # название модуля внутри Python
        sources = ['plus.c'] # исходные файлы модуля
)

setup(
        name = 'my_plus',       # название модуля (my_plus.__name__)
        version = '1.1',                # версия модуля
        description = 'Simple module',     # описание модуля
        ext_modules= [module1]     # объекты типа Extension
)
```

[83] см. https://docs.python.org/3.6/c-api/arg.html#c.Py_BuildValue

[84] см. https://github.com/lvoursl/python-and-c-integration/blob/master/Functions-module/plus-function/setup.py

Далее, необходимо последовательно в GNU/Linux выполнить несколько команд:

```
sudo python3 setup.py build
sudo python3 setup.py install
```

После этого можно запустить python3, импортировать написанный модуль, и вызвать метод plus сложения двух чисел:

```
>>> import my_plus
>>> my_plus.plus(1, 100)

101.0
```

3.12. Использование Cython для создания расширений языка Python

Компилятор Cython переводит программу, написанную на языках Python или Cython, в более эффективный код на C. Язык Cython является надстройкой Python, поддерживающей вызовы функций C и присвоение переменным типов данных языка C.
Код на Cython должен быть скомпилирован перед запуском.
Это происходит в два этапа:

1. Сначала Cython компилирует .pyx файл (содержащий нашу программу) в .c файл.
2. Затем C компилирует .c файл в модуль, который мы можем импортировать.

С помощью IPython Notebook можно писать и вызывать Cython код «на месте». Для этого нужно загрузить расширение Cython в IPython:

```
%load_ext Cython

%%cython
cdef int a = 0
for i in range(10):
    a += i
print(a)
```

ЗАКЛЮЧЕНИЕ

Область разработки программного обеспечения обновляется ежегодно: появляются новые технологии, новые языки программирования, новые среды и т.д. Невозможно угнаться за всеми обновлениями, поэтому в процессе обучение необходимо обращать внимание на элементы, которые сложились исторически и не подвержены существенным изменениям. В предложенном учебном пособии была сделана подобная попытка.

Несмотря на большой объем затронутых тем за рамками пособия остались: теория интерпретаторов, синтаксические конструкции языка C, особенности реализации языка Python и пр.

Исходя из практической направленности данной дисциплины, рекомендую в учебном процессе дополнительно задействовать бесплатные online-платформы[85], интерактивные задания[86], включающие игровые механизмы. Для развития командной деятельности можно создавать учебные проекты с последующей их защитой в конце семестра.

[85] см. https://stepik.org
[86] см. https://checkio.org

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Адитья Бхаргава. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. – СПб.: Питер, 2019. – 288 с.

2. Дж. Плас. Python для сложных задач. Наука о данных и машинное обучение. – СПб.: Питер, 2019. – 576 с.

3. Керниган Б., Ритчи Д. Язык программирования Си: Пер. с англ. – 3-е изд. – СПб.: Невский Диалект, 2001. – 288 с.

4. Маккинни Уэс. Python и анализ данных. – М.: ДМК Пресс, 2015. – 482 с.

5. Федерико Бьянкуцци, Шейн Уорден. Пионеры программирования. Диалоги с создателями наиболее популярных языков программирования, 2011. – 608 с.

6. Федоров, Д. Ю. Программирование на языке высокого уровня Python : учеб. пособие для прикладного бакалавриата / Д. Ю. Федоров. – 2-е изд., перераб. и доп. – М. : Издательство Юрайт, 2019. – 161 с.

7. Python 3.7.1 documentation [Электронный ресурс]. – Режим доступа: https://docs.python.org (дата обращения: 08.06.2019).

Учебное издание

Федоров Дмитрий Юрьевич

ЯЗЫКИ И МЕТОДЫ ПРОГРАММИРОВАНИЯ

Учебное пособие

Издано в авторской редакции

Подписано в печать 29.08.19. Формат 60×84 1/16.
Усл. печ. л. 4,75. Тираж 60 экз. Заказ 1155.

Издательство СПбГЭУ. 191023, Санкт-Петербург, Садовая ул., д. 21.

Отпечатано на полиграфической базе СПбГЭУ