**Майк МакГрат. Программирование на Python для начинающих**

Некоторое время тому назад я начал читать книгу Нейтана Яу. Искусство визуализации в бизнесе. Я думал еще более усовершенствовать свои знания в области визуализации с помощью Excel, но оказалось, что автор довольно невысокого мнения об Excel, и почти все его методы основаны на программировании. Почти сразу же я столкнулся с небольшой программой, написанной на языке Python, извлекающей данные из Интернета. Я установил на своем ПК свободно распространяющуюся версию программы, но код не заработал. Коллега подсказал, что код был написан в 2009 г., так что современная версия Python 3.5.1 его не поддерживает… И я решил немного больше узнать о языке программирования Python: либо после прочтения книги я смогу запустить этот код, либо обращусь к сообществу, и уже не буду полным профаном.

Майк МакГрат. Программирование на Python для начинающих. – М.: Эксмо, 2015. – 192 с.



Купить цифровую книгу в [ЛитРес](http://www.litres.ru/mayk-makgrat/python-programmirovanie-dlya-nachinauschih-14108932/?lfrom=13042861), бумажную книгу в [Ozon](http://www.ozon.ru/context/detail/id/34385287/?partner=baguzin) или [Лабиринте](http://www.labirint.ru/books/503717/?p=13320)

Примеры, используемые в книге, можно скачать с сайта [издательства](http://eksmo.ru/Python_examples.zip). Папки MyScripts и MyProjects разместите в ваш домашний каталог (например, в С:\); о папке htdocs см. ниже раздел *Обработка запросов*. Скачать программу можно с [python.org](https://www.python.org/). Я установил версию Python 3.5.1 с помощью [Windows x86 web-based installer](https://www.python.org/ftp/python/3.5.1/python-3.5.1.exe). Вместе с программой я установил среду разработки IDLE, которая облегчает набор и тестирование программ.

Различают три типа ошибок:

* *Синтаксическая* — происходит, когда интерпретатор обрабатывает код, который не соответствует правилам языка Python, например, отсутствие кавычек вокруг строковой переменной. Интерпретатор останавливается и сообщает об ошибке без дальнейшего выполнения программы.
* *Исполнения* — происходит во время исполнения программы. Например, когда переменная не может быть распознана из-за несоответствия типов. Интерпретатор запускает программу, останавливается на ошибке и сообщает о природе этой ошибки как об исключении.
* *Логическая* (смысловая) — происходит, когда программа ведет себя не так, как было задумано. Например, когда не определен порядок действий в вычисляемом выражении. Интерпретатор запускает программу и не сообщает об ошибке.

**Операции**

Помимо основных операторов, используются также: % – деление по модулю (делит одно число на другое и возвращает остаток от деления; он очень полезен для определения четности или нечетности числа); // – целочисленное деление (работает так же, как и обычное деление, /, но отбрасывает результат после десятичной точки), \*\* – возведение в степень (возводит первый операнд в степень второго операнда).

Помимо стандартного присваивания, =, в языке Python используется и более сложные выражения (рис. 1). В качестве операторов сравнения используются: равенство ==, неравенство !=, а также традиционные >, <, >=, <=. В языке используются обычные логические операторы: and, or, not.



Рис. 1. Операторы присваивания значений

*Проверка условий* использует в своем синтаксисе ключевые слова if и else:

*если-истина-возвращаем-это if (проверочное-выражение) else если-ложь-возвращаем-это*

Например, c = a if (a < b) else b. Выражение в скобках возвращает True, когда значение переменной a меньше, чем b — так что в этом случае меньшее значение присваивается переменной c. Еще одна распространенная область применения условного оператора включает использование оператора деления по модулю, %, для определения того, является число четным или нечетным:

*если-истина(нечетное)-выполняем-это if (var % 2 != 0 ) else если-ложь(четное)-выполняем-это*

Приоритеты выполнения операций приведены в таблице (рис. 2)

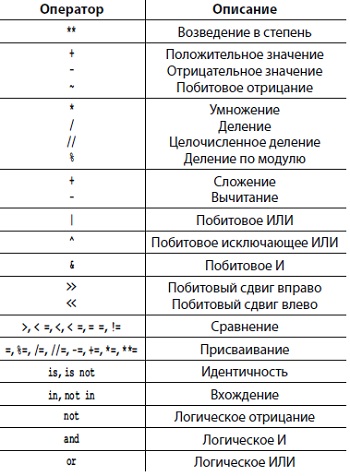


Рис. 2. Приоритет операторов

*Преобразование типов данных.* Переменные в языке Python могут хранить данные любого типа. Основные типы данных: строковые (str), целочисленные (int), с плавающей точкой (float). По умолчанию в переменной хранится строковый тип данных. Попытка сложить два строковых значения просто объединяет эти строки, а не использует операции над числами. Например, '8' + '4' = '84'. Данные, хранящиеся в переменных, могут быть легко преобразованы (приведены) к другим типам с помощью использования функций (рис. 3). Преобразование типа данных с плавающей точкой (float) в целочисленный тип данных (int) отбрасывает десятичную часть числа.

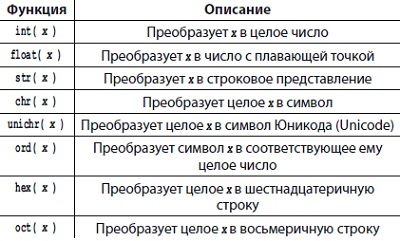


Рис. 3. Функции преобразования типов данных

Код на стр. 41 содержит неточность:

1. a = input( 'Enter A Number: ' )
2. b = input( 'Now Enter Another Number: ' )
3. sum = int( a ) + int( b )
4. print( 'Data Type sum :' , sum , type( sum ) )

Если в ответ на запрос числа (строки 1 и 2) ввести дробное число, то в строке 3 интерпретатор выдаст ошибку. Попытка преобразовать строку, являющуюся символьным представлением дробного числа, в целое число с помощью функции int() вызывает ошибку (см., например, [Что можно делать с помощью функции int](http://pythoner.name/int-function)). Следующее исправление кода спасет ситуацию:

1. sum = int(float( a )) + int(float( b ))

**Инструкции**

*Списки.* В отличие от обычных переменных, которые содержат единственный элемент данных, в Python существует так называемый список, в котором может храниться несколько элементов данных. Данные хранятся последовательно в «элементах» списка, которые индексируются числовыми значениями, начиная с нуля. Например, nums = [0 , 1 , 2 , 3 , 4 , 5]. Для работы со списками существует множество методов (рис. 4).



Рис. 4. Методы списка

Существует также два вида неизменяемых списков: кортеж и множество. *Кортеж*

colors-tuple = ( 'Red' , 'Green' , 'Red' , 'Blue' , 'Red' )

К отдельному элементу кортежа можно обращаться по имени кортежа и индекса в квадратных скобках. Элементы кортежа могут повторяться.

*Множество:* phonetic-set = { 'Alpha' , 'Bravo' , 'Charlie' }

Нельзя обратиться к отдельному элементу множества, используя имя множества и квадратные скобки, содержащие индекс. Элементы множества уникальны. Элементы хранятся внутри множество хаотически (выводятся не в том порядка, как введены).

*Словарем* называется некоторый контейнер, который может содержать несколько элементов данных в виде набора пар ключ:значение. В отличие от обычного списка, к элементам которого можно обратиться по их числовому индексу, к значениям в словаре можно обращаться по связанному с ним ключу. Ключ должен быть уникальным в пределах этого словаря и является, как правило, строкой, хотя числовые значения также иногда используются.

Метод keys() позволяет вернуть список всех ключей словаря в случайном порядке. Для сортировки используйте метод sorted(). С помощью оператора in можно определять, содержит ли словарь нужный ключ. Для этого используется синтаксис *ключ in словарь*. Поиск будет выдавать логическое значение True, если ключ найден в указанном словаре, в противном случае появится значение False.

*Ветвление с помощью условного оператора.* Синтаксис:

if проверочное-выражение:

выполняемая-инструкция-когда-проверочное-выражение-истинно

выполняемая-инструкция-когда-проверочное-выражение-истинно

else:

выполняемая-инструкция-когда-проверочное-выражение-ложно

выполняемая-инструкция-когда-проверочное-выражение-ложно

Отступы в коде очень важны, поскольку они определяют блоки кода для интерпретатора — в других языках программирования для этого используются фигурные скобки. После блока проверки if может использоваться еще ключевое слово elif (else if), которое осуществляет альтернативную проверку.

Цикл while не имеет конечного оператора. Например, цикл, генерирующий несколько первых чисел Фибоначчи (подробнее см. [Альфред Реньи. Числа Фибоначчи](http://baguzin.ru/wp/?p=12101)):

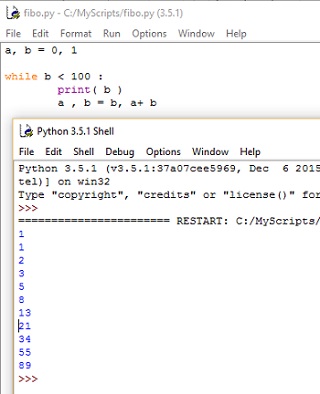


Рис. 5. Код, генерирующий числа Фибоначчи

Для перебора элементов списка используется оператор for in:

for элемент in имя списка :

выполняется-инструкция-на-каждой итерации

выполняется-инструкция-на-каждой итерации

С помощью цикла for in вы можете обходить элементы любого списка или символы строки в том порядке, в котором они появляются, но нельзя указать явно количество итераций цикла, условие остановки, либо размер шага итераций. Однако вы можете воспользоваться функцией языка range(), чтобы сгенерировать последовательность чисел, используемых для итераций. Эта функция генерирует последовательность, начинающуюся с нуля и кончающуюся числом в скобках, не включая его. Например, range (5) генерирует последовательность 0,1,2,3,4.

Вы можете указать в качестве параметров функции range() два числа, разделенных занятой, — начальную и конечную величину. Например, range (1,5) генерирует последовательность 1,2,3,4. Также можно использовать еще один вариант функции, которая принимает три параметра, разделенных занятой — начальная величина, конечная величина и шаг. Например, range (1,14,4) генерирует последовательность 1,5,9,13.

Существует полезная функция enumerate(), с помощью которой, указав в качестве параметра имя списка, вы можете вывести все индексы и связанные с ними значения. Допускается делать обход нескольких списков одновременно. Для этого вам нужно использовать специальную функцию zip(), указав в качестве параметров через запятую имена списков, и на выходе вы получите попарно значения элемента с одним и тем же индексом через запятую. При обходе элементов словаря вы можете вывести пары ключ:значение, используя метод словаря items() и указав после ключевого слова for два параметра — один для имени ключа, другой для его значения:

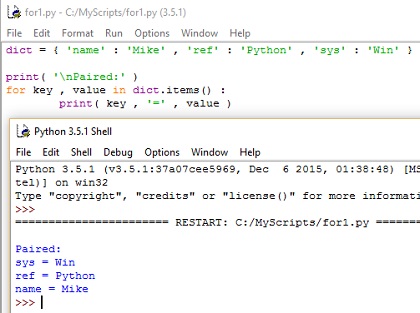


Рис. 6. Извлечение элементов словаря

Для того чтобы принудительно выйти из цикла, когда выполняется какое-то условие, используется ключевое слово break. Оно располагается внутри инструкции цикла после проверяемого выражения. Когда проверка возвращает значение True, цикл немедленно заканчивается. Иногда нужно пропустить одну из итераций цикла, если выполняется какое-либо условие. Для этого используется ключевое слово continue, которое располагается внутри блока инструкций цикла и предваряется проверочным выражением. Когда результат проверки возвращает значение True, данная итерация заканчивается, и программа переходит к следующей.

**Функции**

Пользовательскую функцию можно создать, используя ключевое слово def (definition), после которого следует выбранное вами имя функции и скобки. Строка с определением функции заканчивается двоеточием. Тело функции располагается на строках ниже с использованием отступа. Также, как и при задании цикла, определение функции не имеет закрывающей строки:

def имя-функции () :

исполняемое-выражение

исполняемое-выражение

К переменным, создаваемым вне функции, можно обращаться из инструкций внутри функций — они являются глобальными. К переменным, создаваемым внутри функций, нельзя обращаться извне — они имеют локальную область видимости. Если вы хотите, чтобы к локальной переменной был доступ из любого места, ее нужно сначала объявить с использованием ключевого слова global, после которого следует имя переменной. В тех случаях, когда две переменных – глобальная и локальная – имеют одно имя, функция будет использовать локальную версию.

При определении пользовательской функции можно указывать необязательный параметр (так называемый аргумент):

def echo( user ) :

print( 'User:' , user , 'Language:' , lang , 'Platform:' , sys )

При определении функции можно указать значение аргументов по умолчанию:

def echo( user , lang , sys = 'Linux' ) :

При вызове функции нужно сохранять последовательность аргументов, если только не передать их в явном виде:

echo( lang = 'Python' , sys = 'Mac OS' , user = 'Anne' )

Пользовательская функция также может возвращать значения тому оператору, который ее вызвал. Это выполняется с помощью ключевого слова return с указанием после него возвращаемого значения. Например, чтобы возвратить значение суммы двух складываемых аргументов, пишем:

def sum( a , b ) :

return a + b

total = sum( 8 , 4 )

В языке Python существует возможность создавать функции без имени (анонимные), используя ключевое слово lambda. Анонимная функция может содержать только одно выражение, которое должно всегда возвращать значение. Например, вместо стандартного синтаксиса:

def square( x ) :

return x \*\* 2

можно в более лаконичной форме записать так: square = lambda x : x \*\* 2.

Для того чтобы продолжить код на следующей строке, вы можете использовать символ \.

В языке Python существует ключевое слово pass, которое используется в качестве временной заглушки (заполнителя) и может быть добавлено в те места кода, куда впоследствии надо вставить какие-нибудь строки. Ключевое слово pass является «пустым», то есть не делает ничего. Данная возможность позволяет запускать еще не завершенные программы в целях отладки и исправления синтаксиса.

При работе с циклами следует различать ключевые слова pass и continue. Первый из них позволяет интерпретатору обрабатывать все следующие за ним инструкции в текущей итерации цикла, в то время как второй пропускает все последующие инструкции и передает управление в новую итерацию.

После завершения работы обычной функции управление передается вызвавшему ее оператору, а состояние функции при этом не сохраняется. При следующем вызове функции она будет обрабатывать те же инструкции от начала до конца еще раз. Существует специальная функция-генератор, которая возвращает объект, а не значение. При этом она сохраняет состояние своего последнего вызова и при следующем вызове продолжает работу с той же точки.

Функции-генераторы содержат инструкцию, начинающуюся с ключевого слова yield и определяет объект-генератор, который возвращается оператору, вызвавшему функцию. Когда генераторная инструкция исполняется, состояние объекта-генератора «замораживается» и сохраняется. Объект, возвращаемый генераторной инструкцией, может быть присвоен переменной. С помощью встроен-ной функции next() можно, передав ей имя этой переменной, продолжить выполнение функции с той самой точки заморозки.

Те части программы, в которых возможно появление ошибок, например, работающие с пользовательским вводом, разрешается заключать в специальные блоки try-except, с помощью которых можно обрабатывать «исключительные ситуации». Инструкции, которые способны вызвать ошибки при выполнении, группируются в блок try, а те команды, которым предстоит обрабатывать эти ошибки, — в последующий блок except. После него может стоять необязательный блок finally, где содержатся инструкции, выполняющиеся после того, как все исключения будут обработаны.

Используются следующие встроенные исключения: NameError (имя переменной не найдено), IndexError (попытке обращения к несуществующему индексу элемента списка), ValueError (во встроенную операцию или функцию передается аргумент, имеющий несоответствующее значение). Вы можете также создавать пользовательские типы исключений. Для этого используется ключевое слово raise, после которого указывается тип исключения и описательное сообщение, придуманное пользователем:

day = 32

try :

if day > 31 :

raise ValueError( 'Invalid Day Number' )

except ValueError as msg :

print( 'The Program Found An' , msg )

Для поиска ошибок в вашем коде при отладке программы полезно использовать такой прием, как комментирование, когда в начало строки добавляется символ #, и таким образом одна или несколько строк превращаются в комментарии. В результате интерпретатор Python опускает выполнение этих закомментированных строк, локализуя проблему в программе. Например,

# elest = elest / 2

Если после этого программа выполняется без ошибок, то очевидно, что проблема находилась в закомментированной строке. Еще одним полезным средством для отладки является использование инструкции assert. Она проверяет указанное тестовое выражение на предмет равенства значениям True или False и выдает ошибку AssertionError, если проверка не пройдена. Она может также включать описательное сообщение, имея следующий синтаксис:

assert проверочное-выражение, описательное-сообщение

На первый взгляд работа инструкции assert может быть похожа на обработку исключений, но важно помнить об их различиях.

* + Исключения предлагают способ для обработки ошибок, которые могут происходить на этапе исполнения.
  + Инструмент AssertionErrors обеспечивает программистов средствами для нахождения ошибок во время работы над программой.

Инструкции assert, как правило, удаляются из окончательных версий программы после того, как отладка завершена, в то время как инструкции except остаются в программе и обрабатывают ошибки исполнения.

**Импорт модулей**

Когда вы один раз определили какую-либо функцию, ее можно сохранить в одном или нескольких отдельных файлах (с расширением .py), а затем использовать в других программах без дополнительного копирования в каждую из них. Такой файл, хранящий определение функции, называется модулем, а именем модуля является имя соответствующего файла (без расширения .py). Функции, хранящиеся в модуле, можно сделать доступными в любой другой программе на Python с помощью импортирования модуля, используя ключевое слово import и стоящее после него имя нужного модуля. Обычно инструкции, содержащие импорт модулей, ставят в начало программы, хотя это и необязательно. Любую импортированную в программу функцию затем можно вызвать, используя суффиксную запись, а именно имя-модуля.имя-функции. Например, чтобы вызвать функцию steps из импортированного модуля с именем ineasy (имя файла ineasy.py), достаточно набрать ineasy.steps(). Когда хранящиеся в модуле функции включают в себя какие-либо аргументы, очень часто полезным будет назначать этим аргументам значения по умолчанию при определении функции. Это сделает функцию более универсальной, так как при вызове ее из какого-либо места программы указание аргументов становится необязательным.

Вы можете также импортировать отдельные имена функций с помощью инструкции from import. Имя модуля при этом указывается после ключевого слова from, а имена импортируемых функций — в виде разделенного запятой списка после ключевого слова import. Еще существует способ импортирования всех имен функций в таблицу символов программы. Для этого используется шаблон \* после ключевого слова import. В этом случае все импортированные функции можно будет вызывать без префикса имени модуля. Например:

from dog import bark , lick , nap  
from dog import \*

Здесь dog – имя модуля, а bark, lick и nap – имена функций, определенных в модуле dog.

В Python включены модули sys и keyword, которые применяются для организации доступа к некоторым переменным и функциям, взаимодействующим с самим интерпретатором Python. Модуль keyword содержит список всех ключевых слов языка, содержащихся в его атрибуте kwlist, а также обеспечивает метод iskeyword() для определения, является ли слово зарезервированным.

С помощью модуля sys можно просмотреть список каталогов, где интерпретатор осуществляет поиск. Первым элементом в списке каталогов поиска является ваш текущий каталог — таким образом, интерпретатор Python автоматически найдет любой файл внутри этого каталога либо в его подкаталогах.

Модуль math и его методы используются для работы с математическими операциями, а также модуль random для работы с псевдослучайными числами. Метод random.random генерирует одно число с плавающей точкой от нуля до 1,0, а метод random.sample – список элементов, случайно выбранных из последовательности. Следующая инструкция генерирует случайный список из шести уникальных чисел в диапазоне от 1 до 49:

nums = random.sample( range( 1, 49 ) , 6 )

При выполнении арифметических операций над числами с плавающими точками, используется модуль decimal. Для работы с датами и временем – модуль datatime.

Если знать время начала и конца какого-либо события, то можно вычислить его длительность по разнице двух значений времени. Для того чтобы использовать различные функции, связанные с системным временем, можно импортировать модуль time. Метод sleep() используется, чтобы организовывать паузы в выполнении программы. Аргумент этого метода определяет количество времени в секундах, на которое необходимо сделать задержку.

Существует средство для поиска и замены текста при помощи шаблонов — так называемые *регулярные выражения*. Для их использования необходимо импортировать модуль re. Шаблон регулярного выражения может состоять целиком из символьных литералов, описывающих строку символов, соответствующих какому-либо тексту. Например, регулярное выражение wind найдет соответствие в строке windows. В общем случае шаблон регулярного выражения состоит из набора символьных литералов, а также метасимволов (рис. 7).



Рис. 7. Таблица метасимволов

Комбинация литералов и метасимволов, образующая регулярное выражение, при помощи метода re.compile() компилируется в объект шаблона. У этого объекта существуют методы для различных операций, например при помощи метода match() можно проверять соответствие шаблону строки, переданной методу в качестве аргумента. Метод match() возвращает значение None, если совпадений с шаблоном не найдено, а в случае успешного поиска возвращается объект, содержащий информацию о соответствии строки шаблону. Полученный в результате объект содержит методы start() и end(), возвращающие соответственно позиции начала и конца совпадения, а также метод group(), который возвратит всю строку соответствия. На рис. 8. представлен код программы, проверяющей валидность введенного адреса e-mail

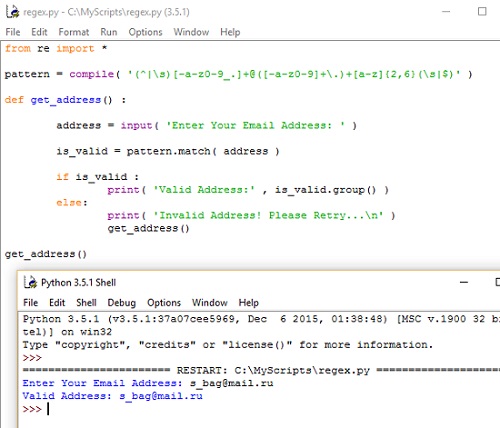


Рис. 8. Код программы, проверяющей валидность введенного адреса e-mail

**Строки и работа с файлами**

В программах на языке Python для работы с переменными строкового типа используются различные операторы (рис. 9).

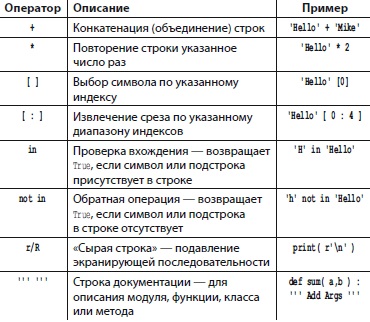


Рис. 9. Операторы для работы со строками

Оператор извлечения среза [:] возвращает строку до символа, чей порядковый номер указан последним в диапазоне, не включая его. Если при извлечении среза опущен начальный индекс диапазона, то предполагается, что он равен нулю, а при отсутствии конечного индекса берется значение длины строки. Оператор «сырая строка», r (или R), должен располагаться непосредственно перед открывающими кавычками для подавления управляющих символов в строке. Используется обычно в случаях, когда строка содержит символ \. Строка документации во время выполнения программы доступна в виде специального атрибута \_\_doc\_\_ объекта.

Функция dir() может быть использована для получения имен всех функций и переменных, определенных в модуле. При этом имя нужного модуля указывается в скобках в качестве параметра (рис. 10). Имена, которые начинаются и заканчиваются символом двойного подчеркивания, являются зарезервированными объектами языка Python, а все остальные — определенными программистом. При помощи функций dir() можно также получить список имен функций и переменных, определенных по умолчанию в модуле \_\_builtins\_\_ таких, например, как функция print() и объект str.

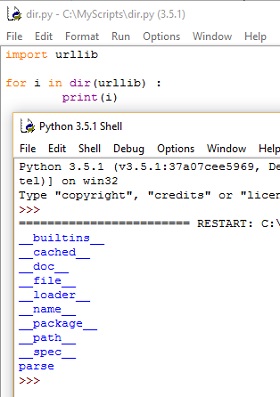


Рис. 10. Извлечение всех функций модуля urllib

Объект str определяет методы форматирования строк, включая метод format(), который производит подстановки. Строка, которую необходимо отформатировать методом format(), может содержать текстовые поля и поля для замены, куда будет подставляться текст из списка элементов, разделенных запятой. Каждое поле замены обозначается парой фигурных скобок {}. Внутри фигурных скобок может стоять порядковый номер заменяемого элемента, в соответствии с которым будут происходить подстановки из списка. Строки можно также форматировать с помощью оператора замены %s. Данный оператор будет помечать места в строке, куда будет вставляться текст из упорядоченного списка значений (рис. 11). Для других типов данных при форматировании используйте %d для целых чисел, %c — для символов, %f — для чисел с плавающей точкой.

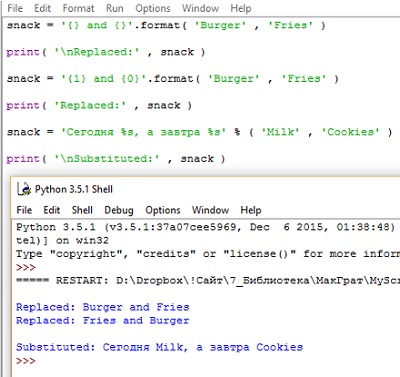


Рис. 11. Форматирование строк по шаблону

Объект str имеет очень много полезных методов для проверки содержимого строк, а также их модификации (рис. 12).

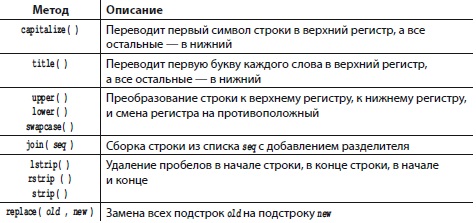


Рис. 12. Некоторые методы объекта str для модификации строк

Python поддерживает Unicode. Для преобразования символов в кодировку Unicode существует метод encode () объекта str, а для обратного действия — метод decode(). В модуле unicodedata существует метод name(), который отображает имя каждого символа в формате Unicode. Таким образом, нелатинские, а также символы с ударениями можно получить, используя их Unicode-имя, либо с помощью преобразования их шестнадцатеричного представления.

Если с помощью функции dir() проверить модуль \_\_builtins\_\_, то можно увидеть, что данный модуль содержит объект file, который определяет несколько методов для работы с файлами системы, включая такие методы, как оpеn(), read(), write() и close().ореп(). Метод оpеn() требует два строковых аргумента для указания имени и пути расположения файла, а также одного из спецификаторов режима открытия файла. Например,

file = open( 'example.txt' , 'w' )

говорит об открытии файла example.txt на запись. Создает новый файл, если он не существует, или открывает существующий файл и стирает все его содержимое. После того как вы открыли файл и у вас появился объект file, с помощью свойств последнего вы можете получить различные подробности, относящиеся к данному файлу (рис. 13).

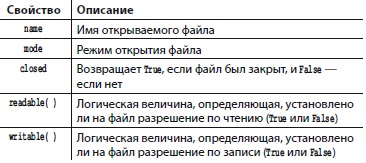


Рис. 13. Методы объекта файл

Вы можете читать содержимое файла, используя циклы и итерации по строкам. Операция записи в существующий файл автоматически перезапишет все его содержимое. Чтобы добавить новое содержимое в файл откройте его в режиме а: file = open( 'example.txt' , 'а' ).

Метод read() файлового объекта по умолчанию читает все содержимое файла, но можно в целочисленном аргументе указать, сколько символов нужно прочитать из файла. Метод seek() позволяет указать позицию, с которой следует начинать читать или записывать. Текущую позицию внутри файла можно получить в любое время с помощью метода tell(). Используйте ключевое слово with для группировки в блок инструкций, работающих с файлом. Это позволит, во-первых, убедиться в том, что файл корректно закрылся после выполнения с ним операций, даже если возникли исключения, а, во-вторых, синтаксис будет гораздо понятнее (рис. 14). Первая инструкция print действует внутри блока with, когда файл открыт. При выходе из блока фал закрывается, что видно при запуске второй инструкция print.

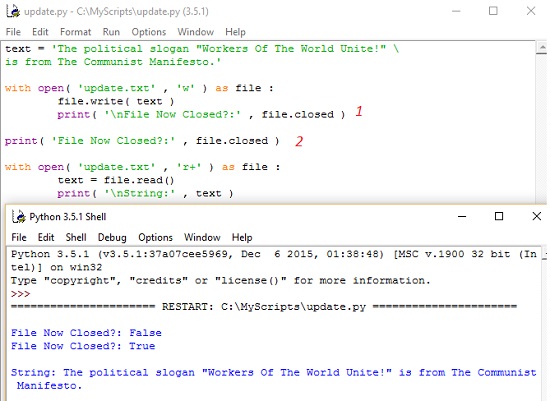


Рис. 14. Метод with при работе с файлом

Строковые данные легко могут быть сохранены в текстовые файлы. Другие типы данных, такие как числа, списки или словари можно также сохранять в текстовых файлах, но для этого требуется сначала преобразовать их в строки. А возвращение этих данных к их первоначальному типу будет требовать еще одного преобразования. Самый простой способ для достижения этого предлагает модуль pickle. Для сохранения объекта в файл используйте метод dump(), которому укажите в качестве аргументов объект и файл. Для последующего восстановления этого объекта из файла используйте метод load(), принимающий в качестве единственного аргумента имя файла.

Поскольку для хранения файлов не требуется, чтобы они были записаны в человеко-читаемом виде, то наиболее эффективно хранить файл в двоичном виде.

Если программе нужно будет проверить существование сохраняемого файла, то для этого в языке Python используется модуль os, предлагающий объект path, у которого есть метод isfile(), возвращающий значение True, если указанный ему в скобках файл найден. Если программе нужно будет проверить существование сохраняемого файла, то для этого в языке Python используется модуль os, предлагающий объект path, у которого есть метод isfile(), возвращающий значение True, если указанный ему в скобках файл найден.

**Объектное программирование**

*Класс* – это тип, описывающий набор свойств, которые характеризуют объект. Каждый класс имеет структуру данных, которая может содержать как функции, так и переменные, характеризующие объект. Членами класса могут быть функции, называемые *методами*, а также переменные (объявленные внутри структуры класса), называемые *атрибутами*. Ко всем свойствам класса можно обратиться локально, используя точечную запись с префиксом self. Кроме того, все определения методов должны содержать self в качестве первого аргумента, то есть метод с именем talk выглядит как talk(self). Объявление класса, в котором определяются его атрибуты и методы, является образцом, из которого могут быть произведены рабочие копии (экземпляры) класса.

Как правило, переменные экземпляра содержат данные, передаваемые вызывающим оператором во время создания экземпляра класса. Поскольку эти данные доступны только локально (как говорится «для внутреннего пользования») они фактически скрыты от остальной части программы. Данный прием, называемый инкапсуляцией данных, гарантирует, что данные надежно хранятся в структуре класса, и является первым принципом объектно ориентированного программирования.

При создании экземпляра класса вызывается специальный метод init(self). Если необходимо передать еще значения для инициализации его атрибутов, то в скобки могут быть добавлены последующие аргументы. Например,

class Critter :

''' Базовый класс для всех живых существ. '''

count = 0

def \_\_init\_\_( self , chat ) :

self.sound = chat

Critter.count += 1

def talk( self ) :

return self.sound

Здесь count – переменная класса, чье целочисленное значение доступно всем экземплярам данного класса. К ней можно обратиться с помощью записи Critter.count как внутри, так и извне класса. Первый из методов \_\_init\_\_() — метод инициализации, который вызывается автоматически во время создания экземпляра класса. В данном случае метод инициализирует переменную экземпляра sound значением, переданным из аргумента chat, и увеличивает значение переменной класса count при каждом создании экземпляра этого класса. Второй метод talk() объявлен как обычная функция, за исключением автоматически указываемого аргумента self — других значений для передачи из вызываемого оператора не требуется.

Экземпляр класса представляет собой объект, возвращаемый конструктором, и его можно присвоить переменной, используя синтаксис *имя-экземпляра = ИмяКласса( аргументы ).* Для того чтобы обратиться к методам и атрибутам созданного экземпляра, можно использовать точечную запись, например, *имя-экземпляра.имя-метода()* или *имя-экземпляра.имя-атрибута*. Обычно определяют некоторый базовый класс и его хранят в отдельном файле модуля Python. В дальнейшем его можно импортировать в другие программы и, таким образом, с легкостью создавать объекты-экземпляры из так называемого мастер-прототипа класса. Имена классов в Python принято начинать с прописной буквы, а имена объектов — со строчной (рис. 15).

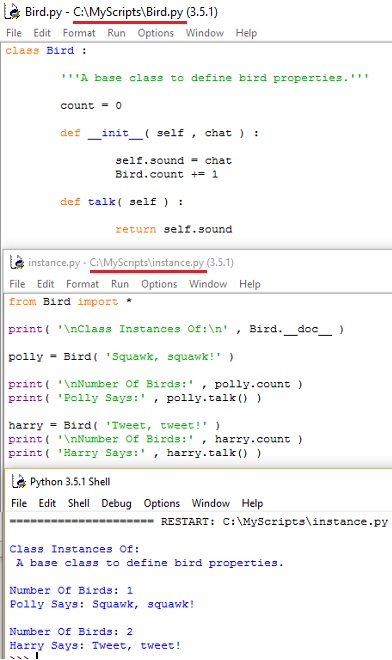


Рис. 15. Создание класса и двух экземпляров объектов

В модуле bird.py объявлен класс Bird. В модуле instance.py создано два экземпляра класса: polly и harry, издающие разные крики с помощью метода talk.

Имена атрибутов, автоматически предоставляемые интерпретатором Python, всегда содержат символ подчеркивания, чтобы обозначить «частный» характер этих атрибутов — их не следует изменять либо удалять. Таким же образом вы можете добавить свои собственные атрибуты, обозначив их как «частные», но не забывайте, что они, как и все другие, могут быть изменены.

Каждый класс автоматически создается с определенным набором встроенных атрибутов. Доступ к их значениям можно получить, используя точечную запись. Атрибуты класса наследуются экземпляром по умолчанию. Чтобы отобразить список всех встроенных атрибутов указанного класса, можно использовать функцию dir(), которой в качестве параметра указать имя класса (или экземпляра), а затем проверить, начинается ли имя атрибута с символа подчеркивания (рис. 16).

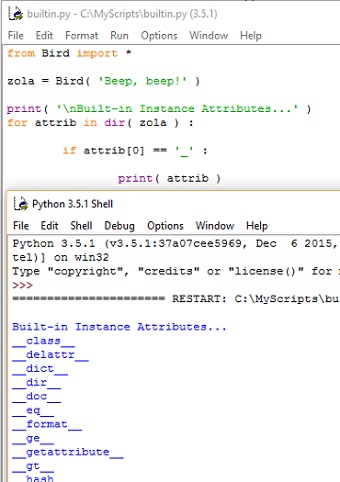


Рис. 16. Встроенные атрибуты

Когда создастся объект — экземпляр класса, под него выделяется уникальный адрес в памяти, который можно посмотреть с помощью встроенной функции id(). Python автоматически выполняет «сборку мусора», чтобы освободить память, периодически удаляя ненужные объекты, такие как экземпляры класса, — в результате их адреса в памяти становятся свободными. Для этого используется встроенный атрибут \_\_weakref\_\_.

Класс может быть создан либо как совершенно новый, либо как «производный» от существующего. Производный класс наследует члены родительского (базового) класса, от которого он произошел. Модель наследования является исключительно мощным инструментом, а также вторым основным принципом объектно ориентированного программирования. Чтобы объявить производный класс, нужно после его имени добавить скобки, в которых указать имя родительского класса.

Если в производном классе объявить метод, имеющий то же самое имя и то же число аргументов, что и какой-либо метод в родительском классе, то данный метод «переопределяется». При этом метод из базового класса становится как бы скрытым и недоступным до тех пор, пока не будет вызван явно с использованием имени базового класса.

Третье ключевое понятие объектно ориентированного программирования — принцип полиморфизма. Полиморфизм (от греческого, означает «много форм») описывает способность назначать элементу различные смысловые значения в зависимости от контекста, в котором он используется. Например, можно создать функцию, в которой берется объект любого типа, и вызываются методы этого объекта. Если объект в действительности имеет вызываемые методы, то они выполняются, в противном случае функция сигнализирует об ошибке выполнения.

**Обработка запросов**

Когда какой-либо пользователь Всемирной паутины просматривает в своем браузере веб-страницу, то на самом деле его браузер запрашивает страницу у веб-сервера и получает ее в ответ через протокол HTTP. Адрес запрашиваемой веб-страницы представляет собой документ HTML (обычно с расширением файла .html), и веб-сервер возвращает его браузеру, который может отразить содержимое этого файла на экране пользователя. Если на компьютере, где находится программное обеспечение веб-сервера, также установлен Python, то веб-сервер может быть сконфигурирован таким образом, чтобы распознавать программы на Python (обычно с расширением файла .py) и вызывать интерпретатор Python для обработки программного кода перед отправкой HTML-ответа веб-серверу, возвращающему данный ответ уже браузеру клиента.

Программа на Python, к которой обратился веб-браузер, может генерировать ответ в виде готового документа HTML, в первой строке кода которого указывается тип содержимого в виде

Content-tjpe:text/html\r\n\r\n

Веб-браузер анализирует содержимое файла разметки и отображает на экране пользователя. К веб-серверу можно обращаться из браузера по доменному имени localhost. Убедитесь, что веб-сервер запущен и сконфигурирован для использования программ на языке Python (подробнее см. [Настройка сервера под Windows 10 для Python](http://baguzin.ru/wp/?p=14293)). Если вы скачали примеры кода, используемые в книге, то самое время разместить содержимое папки htdocs в c:\inetpub\wwwroot – физической папке по умолчанию для доменного имени localhost. Вот, например, что происходит при вызове файла response.py в браузере:



Рис. 17. Интерпретация файла-скрипта Python в браузере (а), код страницы, соответствующий тексту скрипта (б), и, собственно, скрипт в окне IDLE (в)

Когда браузер производит HTTP-запрос, то из него разрешается передать какие-либо значения в программу на Python, находящуюся на веб-сервере. Эти значения могут быть использованы в программе, а также переданы в обратном ответе браузеру. Например, если в окне браузера выведено несколько гиперссылок, то кликая на каждую из них, можно передать параметр в программу Python, которая вернет связанное содержимое.

Для обработки данных, переданных из веб-браузера через HTTP-запрос в программу на Python, допустимо использовать модуль cgi. В нем есть конструктор FieldStorage(), который создает объект, хранящий переданные данные в виде словаря, содержащего пары ключ:значение. Любое значение можно будет получить с помощью метода getvalue() объекта FieldStorage, указав соответствующее имя ключа в виде параметра для данного метода.

Браузер может передавать данные в программу, используя метод GET, который просто добавляет к URL-адресу программы пару в виде ключ=значение. Эта пара следует после символа вопросительного знака, стоящего после имени файла. Таких пар может быть передано несколько, тогда они отделяются символом &, например script.py?key1=value1&key2=value2.

Существует альтернативный и более надежный способ, при помощи которого браузер может отправлять данные в программу без добавления какой-либо информации к URL- адресу — это использование метода POST, который пересылает информацию на веб-сервер в виде отдельного сообщения. Для того чтобы передать с веб-страницы на веб-сервер большое количество текстовых данных, введенных пользователем, в дополнение к формам и методу POST применяется HTML-элемент *textarea*.

В HTML-формах существует такой элемент графического интерфейса, как флажок (или флаговая кнопка). Пользователь может переключать его состояния между «включено» и «выключено», тем самым определяя, добавлять или нет связанные с флажком данные в отправку на веб-сервер. В программе на Python, которой предстоит обрабатывать данные из формы, можно определить установку каждого из флажков, просто проверив, получено ли значение от флажка с определенным именем. Также HTML-формы предоставляют инструмент – группа положений переключателя, с помощью которого пользователь может выбрать только один и передать связанные с этим переключателем данные на веб-сервер (рис. 18).

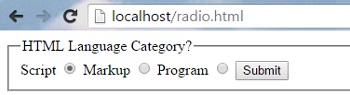


Рис. 18. Форма переключателя

HTML-формы предоставляют еще один инструмент — так называемый выпадающий список, содержащий возможные варианты, из которых пользователь может выбрать один для того чтобы передать связанные с этим элементом данные на сервер. После этого переданное значение можно получил, с помощью метода getvalue() объекта Fieldstorage, указав в качестве аргумента этому методу имя соответствующего списка.

HTML-формы предоставляют средство для выбора файла, которое вызывает стандартное Windows-диалоговое окно, позволяющее пользователю выбрать файл, просматривая каталоги локальной файловой системы. Для того чтобы задействовать данное средство, в HTML-элемент form нужно включить атрибут enctype и указать тип кодировки multipart/form-data. Полный путь к файлу (или полное имя файла), который выбран для загрузки, — это значение, сохраняющееся в объекте FieldStorage, и получить к нему доступ можно, используя соответствующее имя как ключ. С помощью метода path.basename() модуля os из полного пути разрешается выделить краткое имя файла.

**Разработка интерфейсов**

В языке Python модуль, который вы можете использовать для создания графических приложений, называется tkinter (toolkit to interface, набор инструментов для интерфейса). Чтобы обеспечить атрибуты и методы для создания оконного интерфейса, tkinter, как и другие модули, должен был, импортирован в программу на Python. При помощи метода geometry () этого объекта можно указать размеры окна, передав в качестве аргумента строку 'ширина***х***высота'. Можно создать заголовок окна, указав методу title()строковый аргумент 'заголовок'. Если размеры и заголовок не указаны, то будут использоваться значения, заданные по умолчанию.

Завершать каждую программу должен так цикл обработки событий окна, вызываемый метод mainloop(), который реагирует на действия пользователя с окном в течение исполнения программы (закрытие окна для выхода из программы или просто изменение его размеров).

Все элементы управления графического интерфейса, создаваемые с использованием модуля tkinter, такие как кнопки или флажки, называют виджетами. После создания любого виджета его нужно поместить на окно. Для этого в Python существуют специальные методы, называемые менеджерами размещений: pack(), place(), grid().

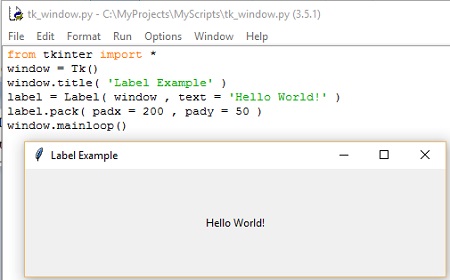


Рис. 19. Вывод оконной формы

Виджет Button создается с помощью указания конструктору Button() имени окна и опций в качестве аргументов. Каждая опция определяется как пара опция=значение. Определяющей является опция command, в которой указывается имя функции или метода, которые должны быть вызваны, когда пользователь нажимает на кнопку.

В программе на Python можно выводить сообщения пользователю в виде диалоговых окон с помощью методов, представляемых модулем tkinter.messagebox. Данный модуль должен быть импортирован отдельно. Диалоговое окно может быть создано с помощью указания заголовка окна, а также самого сообщения в виде двух аргументов.

Для получения данных, введенным пользователем в приложении с графическим интерфейсом, в модуле tkinter существует виджет Entry, который предоставляет однострочное поле ввода. Объект ввода создается при помощи конструктора Entry() указанием ему в качестве аргументов имени родительского контейнера (например, окна или фрейма), а также используемых опций, каждая из которых передается в виде пары опция=значение.

Элементы добавляются в список при помощи метода Insert(), которому в качестве аргументов указывается порядковый номер элемента в списке и строка, определяющая сам элемент. Любой элемент списка можно получить с помощью метода get(), указав его порядковый номер в качестве аргумента. Кроме того, у объекта listbox существует полезный метод curselection(), возвращающий порядковый номер текущего выбранного элемента, так что его можно использовать как аргумент для метода get(), чтобы получить текущий выбор.

Виджет Radiobutton позволяет добавить в графическое приложение один элемент положения переключателя, в которое тот может быть установлен пользователем. Если используется группа таких положений, то пользователь имеет возможность выбрать из этой группы только одно. Объекты положений переключателя группируются вместе, когда необходимо задать одну управляющую переменную и присвоить ей значение в зависимости от выбора пользователя. Для создания такой переменной-объекта применяется конструктор StringVar(), создающий пустую строку, или конструктор IntVar(), инициализирующий пустую целочисленную переменную-объект.

Виджет Checkbutton обеспечивает добавление в графическое приложение одного элемента «флажок», который может быть установлен (выбран) пользователем. Обычно используется группа таких флажков, и пользователь имеет возможность выбрать из этой группы один или несколько. Объекты флажков задают каждый свою управляющую переменную-объект, и ей присваивается значение в соответствии с тем, отмечен флажок пользователем или нет. Для создания такой переменной-объекта применяется конструктор StringVar(), создающий пустую строку, или конструктор intvar(), инициализирующий пустую целочисленную переменную-объект.

Модуль tkinter позволяет работать с файлами изображений в форматах GIF или PGM/PPM, которые могут быть выведены на виджетах Label, Text, Button или Canvas. Для этих целей используется конструктор Photoimage(), который создает объект изображения. Достаточно указать ему в качестве аргумента имя нужного файла с изображением в виде file = 'имя-файла'. Можно воспользоваться методом subsample() для уменьшения изображения, указав в качестве аргументов параметры дискретизации по горизонтали и вертикали в виде х=значение и у=значение. Например, указанные значения х=2, у=2 приведут к отбрасыванию каждого второго пиксела, то есть изображение уменьшится наполовину по отношению к оригиналу.

**Разработка приложений**

Графическое приложение, рассматриваемое далее, будет генерировать шесть случайных чисел из определенного диапазона. Первоначально данный функционал мы реализуем при помощи консольного приложения, а потом продемонстрируем, как его можно применить для работы компонентов приложения с графическим интерфейсом. В стандартной библиотеке языка Python содержится модуль random, который обеспечивает методы для генерирования псевдослучайных чисел. Для настройки генератора случайных чисел используется системное время, поэтому каждый раз при его инициализации будет порождаться новая последовательность.

Для того чтобы сгенерировать псевдослучайное число с плавающей точкой в диапазоне от 0 до 1, используется метод random() модуля random. Диапазон генерируемых значений можно увеличить, применив оператор умножения, \*, а если требуется в результате получить целые числа, то придется воспользоваться преобразованием типов при помощи встроенной функции int(). Например, если вам нужно случайное целое от 0 до 9, используйте инструкцию: int(random.random() \* 10 ). А если от 1 до 10, то: int(random.random() \* 10 ) + 1.

Для генерирования нескольких значений можно использовать данную инструкцию в цикле, но при этом генерируемые числа могут повторяться. Чтобы гарантировать их уникальность, необходимо использовать метод sample() из модуля random. Этому методу в качестве аргументов указывается диапазон и количество возвращаемых чисел из этого диапазона. Для указания диапазона полезно пользоваться функцией range(), которая создает последовательность чисел в виде списка. Например, для того, чтобы сгенерировать шесть уникальных целых чисел в диапазоне от нуля до девяти: random.sample(range( 10 ) , 6 ). А для диапазона от одного до десяти: random.sample(range( 1, 11 ) , 6 ).

Перед созданием нового графического приложения полезно потратить некоторое количество времени на подготовку плана его построения. Необходимо четко определить задачи будущей программы, решить, какая функциональность будет необходима и какие элементы интерфейса потребуются для воплощения идеи. Приблизительный план простого приложения, моделирующего выпадение номеров числовой лотереи «6 из 49», может выглядеть следующим образом. *Назначение.* Программа должна генерировать набор из шести уникальных случайных чисел в диапазоне от 1 до 49 и иметь возможность перезагрузки (например, по кнопке Reset). *Функциональность:* генерирование и последующий вывод шести уникальных случайных чисел; удаление всех сгенерированных значений с экрана.

Элементы интерфейса:

* Одно базовое окно постоянного размера, в котором будут содержаться все остальные виджеты и отображаться заголовок нашего приложения.
* Один виджет Label, на который мы поместим статическое изображение логотипа приложения, — просто чтобы улучшить внешний вид интерфейса.
* Шесть виджетов Label для динамического отображения набора сгенерированных чисел — по одному числу на каждый виджет.
* Один виджет Button для генерации и вывода чисел на виджеты Label при нажатии его кнопки. Данный виджет должен быть неактивным, если сгенерированные числа отображаются на экране.
* Еще один виджет Button для очистки данных с виджетов Label при нажатии его кнопки. Должен быть неактивным в случае, когда сгенерированные числа на экране не отображаются (рис. 20).

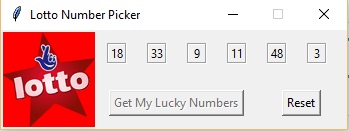


Рис. 20. Интерфейс программы Lotto