Python конспективно за 15 минут

Итак, вы хотите выучить язык программирования Python, но не может найти краткий и в то же полнофункциональный учебник. С помощью этой статьи мы попытаемся научить вас программировать на Python в течении 10 минут. Это, вероятно, не столько учебник, сколько нечто среднее между обзором основных возможностей и шпаргалкой, так что мы будем просто показывать вам некоторые основные понятия, чтобы вы могли начать программировать на Python сразу после прочтения статьи. Представленные здесь коды рекомендуется скопировать в редактор IDLE Python и немедленно проверять по ходу пьесы.

Очевидно, что если вы действительно хотите выучить язык, вам нужно иметь некоторый опыт разработки на этом языке. Мы предполагаем, что вы уже знакомы с основами программированием и, следовательно, будем пропускать большую часть не относящегося напрямую к Python материала. Важные ключевые слова будут выделены, так что вы можете легко найти их. Кроме того, обратите внимание, что из-за краткости статьи, некоторые вещи, которые будут вводиться непосредственно в коде, будут только кратко прокомментированы.

Свойства языка

Python является строго типизированным языком (т.е. обладает типобезопасностью), но, в тоже время, и динамически, неявно типизированным (т.е. вы не должны объявлять переменные). Кроме того, Python чувствителен к регистру (то есть var и VAR — две разные переменные) и объектно-ориентирован (т.е. всё в Python — объект: числа, словари, пользовательские и встроенные классы).

Help!

Помощь в Python всегда доступен прямо в интерпретаторе. Если вы хотите знать, как работает какой-либо объект, все, что вам нужно сделать, это выполнить help()! Также полезна команда dir(), которая показывает все методы объекта, а <object_name>.__ doc__, показывает строку документации для объекта:

```
>>> help(5)
Help on int object:
(etc etc)

>>> dir(5)
[‘__abs__’, ‘__add__’, ...]

>>> abs.__doc__
‘abs(number) -> number

Return the absolute value of the argument.’
```

Ключевые слова Python

Ключевое слово Python – это уникальный программный термин, предназначенный для выполнения какого-либо действия. В Python насчитывается до 33 таких ключевых слов, каждое из которых служит своей цели. Вместе они создают словарный запас языка Python. Словарный запас Эллочки-людоедочки из «12 стульев» Ильфа и Петрова больше!

Они представляют синтаксис и структуру программы Python. Так как все они зарезервированы, вы не можете использовать их имена для определения переменных, классов или функций.

Все ключевые слова в Python чувствительны к регистру. Таким образом, вы должны быть осторожны при использовании их в своем коде. Мы только что сделали снимок возможных ключевых слов Python. Попробуйте в консоли Python набрать следующий код:

```
>>> words = keyword.kwlist
>>> words
['False', 'None', 'True', 'and', 'as', 'assert', 'break',
'class', 'continue', 'def', 'del', 'elif', 'else', 'except',
'finally', 'for', 'from', 'global', 'if', 'import', 'in',
'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise',
'return', 'try', 'while', 'with', 'yield']
>>> len(words)
33
```

Всего 33 слова надо запомнить. Вот здесь о них поробнее:

1. **False** – Ложь
2. **None** – Пусто
3. **True** – Истина
4. **and** –
5. **as** – менеджер контекста
6. **assert** <условие> – возбуждает исключение при ложном значении условия
7. **break** – прервать
8. **class** – пользовательский тип данных, включающий атрибуты и методы манипулирования ими
9. **continue** – продолжить
10. **def** – определение функции
11. **del** – удаление объекта
12. **elif** – элемент конструкции выбора
13. **else** – элемент конструкции выбора
14. **except** – элемент конструкции исключений
15. **finally** – элемент конструкции исключений
16. **for** – цикл с известным количеством повторений
17. **from** – импорт функций из пакета
18. **global** – описание глобальной переменной внутри функии
19. **if** – элемент конструкции выбора
20. **import** – импорт пакета
21. **in** – включение
22. **is** – принадлежность
23. **lambda** – описание лямбда-фунции
24. **nonlocal** –

25. **not** — логическое отрицание НЕТ
26. **or** — логическое ИЛИ
27. **pass** —
28. **raise** — элемент конструкции исключений
29. **return** — возврат в вызывающую функцию
30. **try** — элемент конструкции исключений
31. **with** — менеджер контекста
32. **while** — цикл с неизвестным количеством повторений
33. **yield** — элемент конструкции исключений

Синтаксис

Python не имеет обязательных символов завершения оператора, а границы блоков определяются отступами. Отступ начинает новый блок, отсутсвие отступа его заканчивает. Выражения, которые ожидают после себя новый отступ заканчиваются символом двоеточия (:). Однострочные комментарии начинаются с символа фунта (#), для многострочных комментариев используются строковые литералы, заключенные в тройные апострофы или тройные кавычки. Значения присваиваются (на самом деле, объекты связаные с именами значений) с помощью знака равенства («=»), а проверка равенства осуществляется с помощью двух знаков равенства («==»). Вы можете увеличивать/уменьшать значения при помощи операторов += и -= соответственно на величину, указанную справа от оператора. Это работает для многих типов данных, в т.ч. и строк. Вы можете также использовать несколько переменных в одной строке. Например:

```
>>> myvar = 3
>>> myvar += 2
>>> myvar
5
>>> myvar -= 1
>>> myvar
4

"""Это многострочный комментарий.
```

```
Он содержит две строки """

>>> mystring = "Hello"
>>> mystring += " world."
>>> print mystring
Hello world.

# Здесь переменные меняет местами и код записан в одной строке
(!).
# Строга типизация не нарушается так, как фактически значения
# не назначается, а новые объекты просто связываются со
# старыми именами.
>>> myvar, mystring = mystring, myvar
```

Типы данных

В Python доступны следующие структуры данных: списки (lists), кортежи (tuples) и словари (dictionaries). Наборы доступны в библиотеке sets (но, она встроенна только в Python 2.5 и более поздние версии). Списки похожи на одномерные массивы (но вы также можете создавать списки, состоящие из других списков и получить многомерный массив), словари — ассоциативные массивы (так называемые хэш-таблицы, индексом в которых может быть любой тип данных), а кортежи представляют собой неизменяемые одномерные массивы (в Python «массивы» могут быть любого типа, так что вы можете смешивать например, целые числа, строки и т.д. в списках/словарях/кортежах). Индексом первого элемента в массивах всех типов является 0, а последний элемент можно получить по индексу -1. Переменные могут указывать на функции. Использование описанных типов данных выглядит следующим образом:

```
>>> sample = [1, ["another", "list"], ("a", "tuple")]
>>> mylist = ["List item 1", 2, 3.14]
>>> mylist[0] = "List item 1 again" # Здесь происходит
изменение элемента 0.
>>> mylist[-1] = 3.21 # Здесь вы назначаете последнему
элементу списка значение 3.21
```

```
>>> mydict = {"Key 1": "Value 1", 2: 3, "pi": 3.14}
>>> mydict["pi"] = 3.15 # Здесь вы изменяете значение элемента словаря.
>>> mytuple = (1, 2, 3)
>>> myfunction = len
>>> print myfunction(mylist)
3
```

Вы можете работать только с частью элементов массива используя двоеточие (:). В таком случае, индекс до двоеточия указывает на первый элемент используемой части массива, а индекс после двоеточия – на элемент идущий ПОСЛЕ последнего элемента используемой части массива (он в подмассив не включается). Если первый индекс не указан – используется первый элемент массива, если не указан второй – последним элементом будет последний элемент массива. рассчитывать Отрицательные значения определяют положение элемента с конца. Пример:

```
>>> mylist = ["List item 1", 2, 3.14]
>>> print mylist[:]
['List item 1', 2, 3.1400000000000001]
>>> print mylist[0:2]
['List item 1', 2]
>>> print mylist[-3:-1]
['List item 1', 2]
>>> print mylist[1:]
[2, 3.14]
# Добавляя третий параметр «шаг» для вывода элементов списка
# Теперь индекс увеличивается не 1, как это принято в Python.
# Например, здесь будет напечатан первый элемент, затем третий (элементы 0 и 2, отсчет индекса начинается с 0).
>>> print mylist[::2]
['List item 1', 3.14]
```

Строки

Строки в Python ограничиваются как одиночными так и двойными кавычки, и вы можете использовать одинарные кавычки внутри строки, которая ограничена двойными кавычками и наоборот (т.е. "Он сказал 'привет'." и на экран будет выедено "Он сказал

‘привет’!"). Многострочные строки ограничиваются в тремя двойными (""") или тремя одиночными (''') кавычками. Python поддерживает Unicode из коробки, используя вот такой синтаксис: u"Это строка Unicode". Знак процента «%» между строкой и кортежем, заменяет в строке символы «%s» на элемент кортежа. Словари позволяют вставлять в строку элемент под заданным индексом. Для этого надо использовать в строке конструкцию «%(индекс)s». В этом случае вместо «%(индекс)s» будет подставлено значение словаря под заданным индексом, например, так:

```
>>>print("Name: %s\
Number: %s\
String: %s" % (myclass.name, 3, 3 * "-")
Name: Poromenos
Number: 3
String: ——

strString = """Это есть
многострочная
строка""

# ВНИМАНИЕ: Остерегайтесь концевых s в "%(key)s".
>>> print("This %(verb)s a %(noun)s." %{"noun": "test",
"verb": "is"}
This is a test.
```

Операторы управления потоком исполнения

В Python операторы управления потоком данных представленны операторам if, for и while. В Python нет оператора switch; вместо этого следует использовать оператор if. Использование для перечисления через членов списка. Чтобы получить список цифр до числа – используйте функцию range(). Вот пример использования операторов:

```
rangelist = range(10)
>>> print rangelist
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
for number in rangelist:
    # Check if number is one of
    # the numbers in the tuple.
    if number in (3, 4, 7, 9):
        # «Break» terminates a for without
        # executing the «else» clause.
        break
    else:
        # «Continue» starts the next iteration
        # of the loop. It’s rather useless here,
        # as it’s the last statement of the loop.
        continue
else:
    # The «else» clause is optional and is
    # executed only if the loop didn’t «break».
    pass # Do nothing

if rangelist[1] == 2:
    print «The second item (lists are 0-based) is 2»
elif rangelist[1] == 3:
    print «The second item (lists are 0-based) is 3»
else:
    print «Dunno»

while rangelist[1] == 1:
    pass
```

Функции

Для объявления функции используется ключевое слово def. Аргументы функции задаются в скобках после названия функции. Необязательные аргументы задаются в объявлении функции после обязательных аргументов путем присваивается необязательным аргументам значения по умолчанию. Функции могут возвращать кортежи (и с помощью кортежа вы можете эффективно возвращать из функции несколько значений). Лямбда-функции — это специальные функции, которые состоят из одного оператора. Параметры передаются по ссылке, но значения неизменяемых типов (кортежей, строк и т.д.) всё равно не может быть изменены

внутри функции. Например:

```
# Same as def funcvar(x): return x + 1
funcvar = lambda x: x + 1
>>> print funcvar(1)
2

# an_int and a_string are optional, they have default values
# if one is not passed (2 and «A default string», respectively).
def passing_example(a_list, an_int=2, a_string=«A default string»):
    a_list.append(«A new item»)
    an_int = 4
    return a_list, an_int, a_string

>>> my_list = [1, 2, 3]
>>> my_int = 10
>>> print passing_example(my_list, my_int)
([1, 2, 3, 'A new item'], 4, «A default string»)
>>> my_list
[1, 2, 3, 'A new item']
>>> my_int
10
```

Классы

Python поддерживает ограниченную форму множественного наследования с использованием классов. Внутренние переменные и методы могут быть объявлены (по соглашению, это не является требованием самого языка) путем добавления по меньшей мере, двух начальных символов подчеркивания и не более чем одного завершающего (например, «__spam»). Мы можем также присвоить значение переменной класса извне. Это видно в следующем примере:

```
class MyClass(object):
    common = 10
    def __init__(self):
        self.myvariable = 3
```

```
    def myfunction(self, arg1, arg2):
        return self.myvariable

    # This is the class instantiation
>>> classinstance = MyClass()
>>> classinstance.myfunction(1, 2)
3
# This variable is shared by all instances.
>>> classinstance2 = MyClass()
>>> classinstance.common
10
>>> classinstance2.common
10
# Note how we use the class name
# instead of the instance.
>>> MyClass.common = 30
>>> classinstance.common
30
>>> classinstance2.common
30
# This will not update the variable on the class,
# instead it will bind a new object to the old
# variable name.
>>> classinstance.common = 10
>>> classinstance.common
10
>>> classinstance2.common
30
>>> MyClass.common = 50
# This has not changed, because «common» is
# now an instance variable.
>>> classinstance.common
10
>>> classinstance2.common
50

# This class inherits from MyClass. The example
# class above inherits from «object», which makes
# it what’s called a «new-style class».
# Multiple inheritance is declared as:
# class OtherClass(MyClass1, MyClass2, MyClassN)
```

```
class OtherClass(MyClass):
    # The «self» argument is passed automatically
    # and refers to the class instance, so you can set
    # instance variables as above, but from inside the class.
    def __init__(self, arg1):
        self.myvariable = 3
        print arg1

>>> classinstance = OtherClass(«hello»)
hello
>>> classinstance.myfunction(1, 2)
3
# This class doesn’t have a .test member, but
# we can add one to the instance anyway. Note
# that this will only be a member of classinstance.
>>> classinstance.test = 10
>>> classinstance.test
10
```

Исключения

Исключения в Python обрабатываются с помощью блоков try-except [exceptionname]:

```
def some_function():
    try:
        # Division by zero raises an exception
        10 / 0
    except ZeroDivisionError:
        print «Oops, invalid.»
    else:
        # Exception didn’t occur, we’re good.
        pass
    finally:
        # This is executed after the code block is run
        # and all exceptions have been handled, even
        # if a new exception is raised while handling.
        print «We’re done with that.»

>>> some_function()
Oops, invalid.
```

We‘re done with that.

Импорт

Внешние библиотеки загружаются с помощью ключевого слова import [libname]. Вы можете также использовать from [libname] import [funcname] для отдельных функций. Вот пример:

```
import random
from time import clock

randomint = random.randint(1, 100)
>>> print randomint
64
```

Чтение/запись файлов

Python имеет широкий спектр встроенных библиотек. Например, вот как выполняется сериализация (преобразование структуры данных в строки с помощью библиотеки pickle) с использованием записи/чтения файлов:

```
import pickle
mylist = [«This», «is», 4, 13327]
# Open the file C:\\binary.dat for writing. The letter r before the
# filename string is used to prevent backslash escaping.
myfile = open(r«C:\\binary.dat», «w»)
pickle.dump(mylist, myfile)
myfile.close()

myfile = open(r«C:\\text.txt», «w»)
myfile.write(«This is a sample string»)
myfile.close()

myfile = open(r«C:\\text.txt»)
>>> print myfile.read()
‘This is a sample string’
myfile.close()
```

```
# Open the file for reading.
myfile = open(r«C:\\binary.dat»)
loadedlist = pickle.load(myfile)
myfile.close()
>>> print loadedlist
[‘This’, ‘is’, 4, 13327]
```

Особенности

- Условия могут комбинироваться. Например, 1 < a < 3 проверяет, что a одновременно меньше 3 и больше 1.
- Вы можете использовать del для удаления переменных или элементов в массивах.
- Python предлагает большие возможности для работы со списками. Вы можете использовать операторы объявления структуры списка. Оператор for позволяет задавать элементы списка в определенной последовательности, а if – позволяет выбирать элементы по условию, например, так:

```
>>> lst1 = [1, 2, 3]
>>> lst2 = [3, 4, 5]
>>> print [x * y for x in lst1 for y in lst2]
[3, 4, 5, 6, 8, 10, 9, 12, 15]
>>> print [x for x in lst1 if 4 > x > 1]
[2, 3]
# Check if a condition is true for any items.
# «any» returns true if any item in the list is true.
>>> any([i % 3 for i in [3, 3, 4, 4, 3]])
True
# This is because 4 % 3 = 1, and 1 is true, so any()
# returns True.

# Check for how many items a condition is true.
>>> sum(1 for i in [3, 3, 4, 4, 3] if i == 4)
2
>>> del lst1[0]
>>> print lst1
[2, 3]
>>> del lst1
```

- Глобальные переменные объявляются вне функций и могут быть прочитаны без каких-либо специальных объявлений, но если вы хотите, изменить значение глобальной переменной из функции, нужно объявить их в начале функции используя ключевое слово global, в противном случае Python будет считать эту переменную новой локальной переменной. Например:

```
number = 5

def myfunc():
    # This will print 5.
    print number

def anotherfunc():
    # This raises an exception because the variable has not
    # been bound before printing. Python knows that it an
     # object will be bound to it later and creates a new,
local
    # object instead of accessing the global one.
    print number
    number = 3

def yetanotherfunc():
    global number
    # This will correctly change the global.
    number = 3
```

Заключение

Эта статья не претендует на исчерпывающий перечень всех (или даже основных) возможностей языка Python. Python имеет широкий спектр библиотек и огромный набор функциональных возможностей, которые вы изучите сами если вы захотите и в дальнейшем изучать этот язык программирования. Я надеюсь, что я сделал ваш переход к Python проще и вы уже сможете решить большинство задач Практикума 1