SAMSUNG

SRPOL R&D Center

Python Training

Oct 10, 2017

SRPOL MQA/FTE/DA Part

Contents

Python Introduction

Python is...

Interpreted

- Processed in run-time
- No compilation required

Interactive

- You can interact with interpreter directly

Object-Oriented

- Support of Object-Oriented style or technique of programming

Easy-to-learn

- Great for beginners
- Simplicity helps with automation

Installation & Hello World Application

Local environment setup

Download Python

- Go to https://www.python.org/
- Download Python 2.7.X
- Run downloaded file

Run Python

- Open Command Line (cmd.exe)
- Type „python"
- Press Enter

Hello World

- While in interpreter type „print "Hello World!"„
- It just works ☺
- Or:
  - Create new file with .py extension
  - Write „print "Hello World!"„ in it
  - Run in cmd: „python filename.py"

Syntax

Indentation

No braces in python code – blocks of code are denoted by line indentation

```
if True:
    print True
else:
    print False
```

Comments

Comments are starting with hash (#) character

```
print "Hello, Python!" # comment
```

Variable assignment

No initial variable type required in python

```
var1 = 5
var2 = True
```

Variable Types - Numbers

- Integer
  - int – signed integers

```
var1 = 1
var2 = -5
```

- Long
  - long – big numbers, python automatically converts big ints to longs

```
var1 = 11312312L
var2 = -513212312L
```

- Float
  - float – floating point real values

```
var1 = -1.23
var2 = 0.0
```

- Complex
  - complex numbers with imaginary values

```
var1 = 3.14j
var2 = 9.322-36j
```

Variable Types - Strings

- String
  - Denoted by either single ('), double (") or triple quote (''')

```
word = 'word'
sentence = "This is a sentence"
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

- String parsing
  - Access specific letter or letters via index

```
>>> sentence = "Hello Dubai!"
>>> sentence[0]
'H'
>>> sentence[2:6]
'llo '
>>> sentence[:6]
'Hello '
```

  - You can also build up strings

```
>>> var1 = "Hello"
>>> var2 = "Dubai"
>>> var1 + " " + var2
'Hello Dubai'
```

Variable Types - List

- List
  - List of objects, created via square brackets []
  - Can store any type of object
- Basic List operations
  - Access specific objects via index
  - Can be added or multiplied
  - Easily change elements

```
>>> my_list = [1, 2, "str"]

>>> my_list[1]
2
>>> my_list[:2]
[1, 2]

>>> my_list * 2
[1, 2, 'str', 1, 2, 'str']

>>> my_list + [3, 4]
[1, 2, 'str', 3, 4]

>>> my_list
[1, 2, 'str']
>>> my_list[1] = "cat"
>>> my_list
[1, 'cat', 'str']
```

Variable Types - Tuple

- Tuple
  - List of objects, created via standard brackets ()
  - Can store any type of object
  - Unlike list – cannot be changed after creation
- Basic Tuple operations
  - Access specific objects via index
  - Can be added or multiplied

```
>>> my_tuple = (1, "str", 3)

>>> my_tuple
(1, 'str', 3)

>>> my_tuple[1]
'str'
>>> my_tuple[:2]
(1, 'str')
>>> my_tuple[2] = 1
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    my_tuple[2] = 1
TypeError: 'tuple' object does not support item assignment
```

Variable Types - Dictionary

- Dictionary
  - Hashmap, created via curly brackets {}
  - Values stored in key: value manner
- Basic Dictionary operations
  - Access specific objects via its key
  - Keys need to be static objects like ints, and strings
  - Fast variable assignment and access

```
>>> my_dict = {"key": "value", "one": 1}

>>> my_dict
{'key': 'value', 'one': 1}

>>> my_dict["one"]
1
>>> my_dict["key"]
'value'

>>> my_dict["one"] = "two"
>>> my_dict
{'key': 'value', 'one': 'two'}

>>> my_dict["new key"] = "new value"
>>> my_dict
{'new key': 'new value', 'key': 'value', 'one': 'two'}
```

Operators – Basic Arithmetic

Basic Arithmetic

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
// Floor Division	The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity):	9//2 = 4 9.0//2.0 = 4.0 -11//3 = -4, -11.0//3 = -4.0

Operators – Comparison

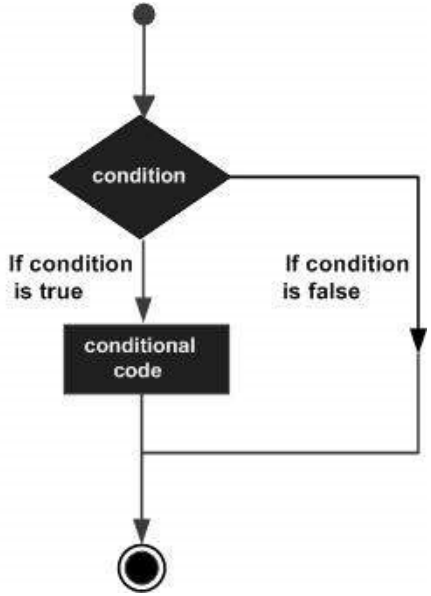
Comparison		
Operator	**Description**	**Example**
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes tru e.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then conditi on becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Decisions – if/elif/else

if/elif/else

Statement	Description
if statements	An **if statement** consists of a boole an expression followed by one or m ore statements.
if...else statements	An **if statement** can be followed by an optional **else statement**, which executes when the boolean expressi on is FALSE.
nested if statements	You can use one **if** or **else if** statem ent inside another **if** or **else if** state ment(s).

```
if var == True:
    print "True"
elif var == False:
    print "False"
else:
    print "None"
```



Boolean conversion

- Conversion to boolean is done via **bool** method
- Empty strings, lists and tuples will return False by default
- Python does boolean conversion automatically in if/else/loop statements

```
>>> bool("Hello")
True
>>> bool("")
False

>>> bool(1)
True
>>> bool(0)
False

>>> bool([])
False
>>> bool([1, 2])
True

>>> if "string":
        print "Converted automatically"


Converted automatically
>>> if "":
        print "This one won't print"
```

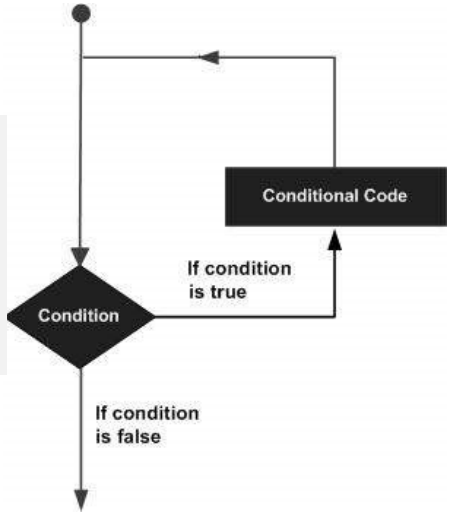
Loops

Loops

Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
nested loops	You can use one or more loop inside any another while, for or do..while loop.

```
i = 0
while i < 5:
    i += 1
    print i

for i in [1, 2, 3]:
    print i+
```



Loops – Loop control

Loops

Control Statement	Description
break statement	Terminates the loop statement and transfers execution to the statement immediately following the loop.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
pass statement	The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

```
i = 0
while i < 5:
    i += 1
    if i == 3:
        break  # ends loop

for i in [1, 2, 3]:
    if i == 2:
        continue  # skips to next iteration
    print i

for i in [1, 2, 3]:
    if i == 2:
        pass  # no action is performed
    else:
        print i
```

Functions

Creation

Function blocks begin with the keyword **def** followed by the function name and parentheses ( )
Return value with **return <your variable or value>** statement

```
def my_method(foo, bar):
    print 123
    return True
```

Arguments

Arguments can be passed either in order or via keywords

```
my_method(123, bar=456)
```

Reference or value?

Some parameters in Python are passed by reference (lists and dictionaries) and some are passed by value (ints, strings)

```
>>> def my_method(arg):
            arg += "1"
            return arg

>>> var = ""
>>> my_method(var)
'1'
>>> var
''
```

Modules

- You can import all methods and classes from other files
- By default you can import all .py files from your current directory
- If you want to import file from a directory you need to add empty __init__.py file into it

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

>>> from math import pi
>>> pi
3.141592653589793

>>> from math import *
>>> pow(2, 2)
4.0

>>> from my_dir.my_module import method
```

Files I/O

- Opening files in python is very easy
  - There are three ways of opening files in Python: read, write and append
  - Remember about closing file!
- Writing to file
- Reading file
- Appending to file

```
>>> f = open("new_file.txt", "w")
>>> f.write("test!")
>>> f.close()

>>> f = open("new_file.txt", "r")
>>> f.read()
'test!'
>>> f.close()

>>> f = open("new_file.txt", "a")
>>> f.write("₩nnewline")
>>> f.close()

>>> f = open("new_file.txt", "r")
>>> f.read()
'test!₩nnewline'
>>> f.close()
```

Classes

- Classes in python are defined by **class** keyword
- Python supports Object-Oriented programming style (overloading, inheritance)
- __init__ method is a constructor
- Access to internal values is done via **self** attribute
- Functions require self argument to have access to internal attibutes of the class
- Everything is public

```
>>> class Vegetable:
        name = 'vegetable'

>>> class Tomato(Vegetable):
        name = 'tomato'
        def __init__(self, size):
            self.size = size


>>> class Vegetable:
        name = 'vegetable'
        size = 0


>>> tomato1 = Tomato(size=10)
>>> tomato2 = Tomato(size=5)
>>> tomato1.size
10
>>> tomato2.size
5
```

Exceptions

- Exceptions function like in every other language
- Can be handled by **try/except** blocks
- Raising exception can be done via **raise Exception(message)**
- Most base Exception is called BaseException
- Common exceptions:
  - IOError - problems with file reading/writing
  - NameError – not defined method/variable
  - ZeroDivisionError – division by zero
  - AttributeError – missing object attribute
  - KeyError – missing key in dictionary
  - IndexError – lack of corresponding index in the list/tuple/string

```
>>> 0/0
Traceback (most recent call last):
  File "<pyshell#69>", line 1, in <module>
    0/0
ZeroDivisionError: integer division or modulo by zero

>>> try:
                0/0
except ZeroDivisionError, e:
                print e

integer division or modulo by zero

>>> raise Exception("message")
Traceback (most recent call last):
  File "<pyshell#75>", line 1, in <module>
    raise Exception("message")
Exception: message
```

Useful methods

- Changing object to int/string/float
- Checking length of the list/tuple/string
- Replacing values in a string
- Splitting string into list

```
>>> int("123")
123
>>> str(123)
'123'
>>> float(123)
123.0

>>> len("Hello!")
6
>>> len([1, 2, 3])
3

>>> "Hello World!".replace("World", "Dubai")
'Hello Dubai!,

>>> "This is a sentence".split(" ")
['This', 'is', 'a', 'sentence']
```

Useful methods

- Getting list of keys from the dictionary
- Sorting list
- Reverting list

```
>>> my_dict = {"key1": "value1", "key2": "value2"}
>>> my_dict.keys()
['key2', 'key1']

>>> my_list = [1, 5, 10, 3]
>>> my_list.sort()
>>> my_list
[1, 3, 5, 10]

>>> my_list.reverse()
>>> my_list
[10, 5, 3, 1]
```

Useful methods

- List comprehension
- Range method

```
>>> my_list = [i for i in "Iterable object"]
>>> my_list
['I', 't', 'e', 'r', 'a', 'b', 'l', 'e', ' ', 'o', 'b', 'j', 'e', 'c', 't']

>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Thank You

SAMSUNG