Introduction to Python scripting language

Sachin Deodhar

01/07/18

Content confidentiality and copyrights

- The training material and contents of this couse are for internal KPIT training purpose
- Do not share this document with anyone outside of KPIT
- No part of this document can be copied without explicit permission from KPIT

Course structure

The course is structured is such that it contains sessions as well as practice excersises

Session 1

- Introduction and Python basics

Session 2

- Python Constructs like loops, decision making and data types like numbers and string
- Advanced objects like lists, typles and dictonaries

Session 3

- Functions, modules, File IO and exceptions

Session 4

- Advanced topics: Classes/Objects
- Advanced topics: Networking

Session 5

- Advanced topics: Multi-threading
- Advanced topics: Reg-expressions

Excersise

- Each session will have some practice excercises which attendee can try later.

Session 1: Introduction to Python basics

Duration: 2 hours

01/07/18

Introduction

Python is a general purpose high level programming language with the following key features

Interpreted language

- Unlike compiled languages like C/C++, python is an interpreted language.

- At runtime commands are interpreted and processed by the python interpreter.

Interactive language

- Python command intrepetor provides a interactive prompt

- Individual commands and the tried/executed on the prompt directly

Portable

- Python language is portable across different platforms

Easy to use

- Python language is intutive and has a clear syntax.

- This makes it readable and fairly easy to code.

- Supports both procedural as well as object oriented programming pattern,

Uses

Since Python is generic and fairly easy to learn it is used for several applications. There is a huge number of libraries available that support python.

They include areas like

- Math and scientific (SciPy, NumPy, SymPy)
- Graph plotting (mathplotlib, Plotly, pyQtGraph)
- GUI toolkits (pyQT, pyGTK, wxPython)
- Audio handling (pyAudio, Nsound, pySonic)
- Computer Vision and machine learning (openCV, bindings with TensorFlow)
- Several test automation frameworks like Robot Framework, Selenium, Cypress, etc

This is just a small list, there are several libaries for, networking, multi-threading, web applications, database management.

Installing Python

Python is available for a wide variety of platforms including Linux, Windows and MacOS. We will be convering the steps to install it on Linux and Windows

Installing on Linux

- Standard Ubuntu desktop installation will come with some version of Python 2.7 preinstalled.
- If its not already installed then it can be installed via either apt-get command or synaptic package manager
- If anyone wants to build it from sources, following are the steps
    - Download source from https://www.python.org/downloads/release/python-2715/
    - Change directory to Python unzip folder and run command `./configure`
    - Run `make` to compile and `make install` to install

Installing on Windows

- Download the Python installer from https://www.python.org/downloads/release/python-2715/
- Run the downloaded installer (*.msi) file and following the installation wizard.

Setting up Python environment

This will be required only if python was manually installed. If its pre-installed or if installed by the installation wizard then these steps may not be required.

Setting Python PATH on Linux

- For Bash or Sh: Use command `export PATH="$PATH:/usr/local/bin/python"`
- For csh: Use command `setenv PATH "$PATH:/usr/local/bin/python"`

Considering python install location is /usr/local/bin/python. In most cases the default installation path is /usr/bin/python

Setting Python PATH on Windows

- Run the command `path %path%;C:\Python`
- It can also be set using the Windows UI
  - Right Click on “My Computer” then;
  - “Properties” -> “Advanced System Settings” -> “Environment Variables” -> “New”

NOTE: Default install location is C:\Python

Python package installer

There are several useful python packages available and most of them can be installed using the Python package installer (pip)

- In Python version 2.7.9 and above or 3.4 and above, PIP is pre-installed.
- On older versions of python it can be installed using get-pip.py script
  - Download it from the location [https://bootstrap.pypa.io/get-pip.py]
  - Install it with the following command [`python get-pip.py`]
  - Installing it behind a proxy

    [`python get-pip.py –proxy="http://[user:passwd@]proxy.server:port"`]
  - If PIP needs to be upgraded
    - On Linux: `pip install -U pip`
    - On Windows: `python -m pip install -U pip`
- Installing some package with PIP

```
$ pip install SomePackage
[...]
Successfully installed SomePackage
```


Running Python interpreter

Being an interactive language it is possible to launch Python interpreter from the command line.

- On Linux it can be launched by command **`$python`**

- On windows it can be launched by command **`C:> python`**

Sample command being run in the interpreter

```
~$ python
Python 2.7.6 (default, Jun 22 2015, 18:00:18)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> print "Hello World ... from Python"
Hello World ... from Python
>>>
```


Writing the first Python Program

Writing the first Python program

- Open any text editor any create a new file, hello.py

- Enter the following text in the file

```
#!/usr/bin/python

print "Hello World ... from Python"
```

Note the first line #!/usr/bin/python. It points to the python interpreter.

Running the program

```
$ python hello.py
```

Or

```
$ chmod +x hello.py

$ ./hello.py

Hello World from Python

$
```


Identifiers in Python

Identifiers in Python are names used for variables, functions, classes, objects or modules.

Following are some restrictions on naming the identifiers

- Punctuation characters such as @, $, and % are not allowed in the identifiers
- Identifiers are case sensitive **Buffer** and **buffer** are different variables
- Identifiers can start with an letter from a to z or A to Z or an underscore _
- This can be followed by letters, digits and underscores.

Reserved words

and	exec	not	assert	finally	or
break	for	pass	class	from	print
continue	global	raise	def	if	return
del	import	try	elif	in	while
else	is	with	except	lambda	yield

Convention for Python identifiers

- Class names start with an uppercase letter.
- All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- If the identifier also ends with two trailing underscores, the identifier is a language-defined special Name. These are mostly used for built-in identifiers.

Lines identation in Python

Identation

- Unlike languages like C/C++, Python does not provide braces to indicate code blocks
- Instead code blocks are identified by line identations, which is stricly enforced
- The number of spaces used for identation is variable but the same (or mutliples of the same) needs to be followed for the entire code block

Example

```
        try:
            for line in open(self.version_log,'r').readlines():
                try:
                    if "BUILD_NUMBER" in line:
                        self.version = line.split(' ', 2)[1]
                        self.version = self.version.strip('\n')
                except IndexError:
                    print "Ignoring line [{}]".format(line.strip('\n'))
                    pass
        except IOError:
            print "Unable to open [{}] file".format(self.version_log)
            self.version = "NA"
```


Multi-line statements, Quotations and comments in Python

Mulit-line statements

- Python statements end with a new line, however it allow the use of the line continuation character (\) to denote that the line should continue

- Statements contained within the [], {}, or () do not need to use the line continuation character

Quotations

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

Comments

- The hash (#) which is not inside any string literal marks the begnining of a comment

Example

```
#!/usr/bin/python
Line = 'hello' + \
       'world'
# Printing hello world
print Line
```


Python variables declaration

Variable declaration

- Python does not need explicit declaration of the variable for the interpreter to to reserve memory

- Declaration happens automatically when the variable is assigned a value

Assigning values to variable

```
#!/usr/bin/python
index   = 100         # Integer assignment
percent = 97.5        # Floating point assignment
name    = "Hello"     # String assignment
print index
print percent
Print name
```

Multiple assignments

- Assigning same value to multiple variables

```
i = j = k = 100
```

- Assigning multiple variables with values of different types

```
i,j,k = 1, 99.5, “percent”
```


Python standard data types

Following are standard data types supported by Python

- **Numbers:** Python supports four numerical types namely; int, long, float, complex
- **String:** Strings are identified as a contiguous set of characters represented in the quotation marks.

  Python allows either single or double quotation marks.
- **List:** List are comma separated list of items, they are similar to arrays in C

  However a key difference is one list can contains items of different data types

  Eg: `list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]`

  Note: Elements call be accessed by index. First list[0], last list[-1], third list [2]
- **Tuple:** Tuples are similar to list but they cannot be updated. Like a readonly list.

  Unlike list they are represented by (()) brackets

```
tuple = ( 'abcd', 786 , 2.23, 'john', 70.2  )
```

- **Dictionary:**

  Dictionaries are hash table types, with associated key value pairs

```
dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"
```


Python advanced types

Additionally some are some python standard data types and objects. Some of them are listed below

- **Iterator Types:** Python supports a concept of built in iterator.

  Eg:

```
print("List Iteration")
l = ["How", "are", "you"]
for i in l:
   Print(i)
```

In the above example built in iterator is used.

- **Binary Types:** These include bytes, bytearray, memoryview. These are typically used for manipulating binary data

Python type of operators

Python supports the following types of operators

- Arithmetic: Add(+), Subtract(-), Multiply(*), Exponent(**), Divide(/), Floor division(//), Modulus(%)
  NOTE: In python 2.7 (/) and (//) both are same and perform floor or integer division.

- Comparison (Relational):
  Evaluates true, if left and right operands are, equal(==), not equal(!= or <>), less than(<), greater than (>), less than equal(<=) or greater than equal(>=)

- Assignment: Assign (=), add and assign (+=), subtract and assign (-=), multiply and assign (*=), divide and assign (/=), Modulus and assign (%=), exponent and assign (**=)

- Bitwise: Binary AND(&), Binary OR(|), Binary XOR(^), One's complement (~), left (<<) and right(>>)

- Membership: This operator tests if the variable in the left operand is part of the specified sequence
  For 'in' operator, true if the variable is part of the sequence and for 'not in' operator if its not

- Identity: This operator tests if both the operands are points to the same memory location.
  The 'is' operator evaluates true if they point to the same object and 'is not' evaluates true if they don't

Python built-in modules

Python provides a lot of built-in modules, following is a non-exhaustive list of those

Built-in Functions: abs(), all(), any(), hash(), zip(), bin(), chr(), int(), isinstance(), len(), map(), vars() and may more

Built-in Types: Already covered in previous slides

Text Processsing: string, re, textwrap, unicodedata, etc

Numeric and Mathematics modules: numbers, math, decimal, random, statistics etc

File and Directory access: pathlib, os.path, stat, glob, shutils etc

Data compression: zlib, gzip, bz2, lzma zipfile, tarfile

Cryptography services: hashlib, hmac, secrets, random numbers

IPC and networking: signal, mmap, socket, ssl, asyncio etc

Internet data handling and Markup processing: mimetypes, email, json, html, xml, json

..... and many more

Python scan user inputs

Following is an example to scan user inputs in the program

```
#!/bin/python

first_name = raw_input("Input your First Name:")
last_name = raw_input("Input your Last Name:")
print ("Hello  " + first_name + " " + last_name)
```

Note:
In python 2.7
raw_input() takes exactly what the user typed and passes it back as a string.

In python 3.0
The same behaviour of raw_input() is implemented in input()

Note:
Since the raw_input returns a string, so if numbers are expected, it needs to be converted to numeric types
Eg: `num = “3”`
`int_num = int(num)`
There are several such conversions possible: int(), long(), float(), str(), etc

Handling command line arguments

In most program it is required to accept some inputs from the user at the time of launching.

Python provides a built-in module to do that. This module is **'sys'**

This modules provides access to python interpreter variables and function with interact with it.

Following is a simple example of a Python script using command line arguments.

```
#!/usr/bin/python

import sys

print 'Number of arguments:', len(sys.argv), 'arguments.'
print 'Argument List:', str(sys.argv)
```

When launched with 3 arguments (arg1, arg2 and arg3), it will print the following output

```
$python main.py arg1 arg2 arg3
Number of arguments: 4 arguments.
Argument List: ['main.py', 'arg1', 'arg2', 'arg3']
```


Handling command line arguments (Continued)

There are some Python standard library modules which greatly simply handling command line arguments

Getopt:
This module provides a C stype getopt() like API's to parse command line arguments (https://docs.python.org/2/library/getopt.html)

Eg:

```
import getopt, sys

try:
    opts, args = getopt.getopt(sys.argv[1:], "ho:v", ["help", "output="])
except getopt.GetoptError as err:
    print str(err)  # will print something like "option -a not recognized"
    sys.exit(2)

output = None
verbose = False
for o, a in opts:
   .....
```


Handling command line arguments (Continued)

Argparse:
This is another module for command line argument parsing. It requires writing much lesser code to extract the argumets
(https://docs.python.org/2/library/argparse.html)

Eg:

```
import argparse

parser = argparse.ArgumentParser(description='Description of your program')
parser.add_argument('-f','--foo', help='Description for foo argument', required=True)
parser.add_argument('-b','--bar', help='Description for bar argument', required=True)
args = vars(parser.parse_args())

print args['foo']
print args['bar']
```


Exercise 1 (suggested time between 2 to 3 hours)

- Installing Python and setting up the environment

Use Window or Linux desktop system to try the python installation and environment setup

- Executing commands in Python interpreter

Run simple command like print or assigning values to variables and then printing them.

- Arithmatic operations

```
>>> x=3
>>> y=2
>>> z=7
>>> x*y*z
42
>>> a=x*y*z
>>> print a
42
```

- Arithmatic operations

```
>>> (2+4+6)*3-12/3
```

- String Concatnation

```
>>> "spam"
'spam'
>>> "spam," + " " + "eggs and spam"
'spam, eggs and spam'
>>>
```


Exercise 1 continued (suggested time between 2 to 3 hours)

Writing some simple python programs

1. Write a Python program to read user input from console and print the string
2. Write a Python program to calculate area of a circle
   - Read radius from the console (Hint: conversion to different data type needed)
   - Calculate the area (Hint: use of exponent operator)
3. Write a Python program to add two positive integers without using the '+' operator.

   Hint: Use binary operators
4. Write a Python program to display the first and last colors from the following list.

   color_list = ["Red","Green","White" ,"Black"]
5. Write a Python program to check whether a specified value is contained in a group of values.

   Test Data: 3 -> [1, 5, 8, 3] : True

   -1 -> [1, 5, 8, 3] : False
6. Accept command line arguments of different types, extract them from the list and print.

Thank you

01/07/18