Internet Engineering

Reza Maanijou
Fall 2017
University of Guilan

Section 3: Python Introduction

Titles

- Python Intro
- Installation
- Basic Python syntax
- Socket programming with python

Python

- Script language
- Dynamic type
- Easy to use
- Powerful in handling of strings
- Strong community

Installation

- https://www.python.org/downloads/
  - Python version >3.5
- Set python PATH variables (check this while installing on windows)
- Install pip (is installed by default for windows)
- Linux:
  - sudo apt install python3-pip python3-dev
- > python3 --version
- > python –version
- Start coding with python3 or python commands!

Run HTTP server

- Go to project folder (cd PATH)
- Try running a simple HTTP server using python
- Serves HTML and other type of files
- python3 -m http.server <port>
  - python3 -m http.server 8080
- Can be used as a file transfer method
- Access to the server using tools
  - ifconfig (unix)
  - ipconfig (windows)

Hello World

- Create an empty file test with .py extension
- Add the following line and save:

  print("Hello World, from python3!")
- run by:
  - python3 test.py
- Or run python3 in terminal and then write the code

Print function

- print("string")
- Can be used with multiple variables of any kind:
  - print("string1", var1, "string2" ...)
- print("Hello", end=" ", sep=",")
- end set the ending char (default: \n)
- sep sets the separation char between parameters that must be printed (default: " ")
- print(2, "+", 2, "=", 4)

Python block

- There is no need to put semicolon «;»
- Blocks are defined with indentation (spaces or tabs)
- Usually 4 spaces
- «pass» commands for empty blocks
- # sign shows commands
- Multi line comments are just multiline strings

```
if True:
        print("True") #outputs
        if False:
                print("1")
                print("2")
        print("3")

if False:
        pass
print("4")
------------------------------
Output:
True
3
4
```

Python Variables and Types (1)

- No need to declare variables or types
- Every variable is firstly an object
- Dynamic type checking
- Popular types:
  - None, int, float, str, tuple, dictionary, list ...
- type() function to check type of a variable or value.
  - type(232)
- Conversion: str(23), float(2), ...
- None is a special type similar to undefined

Python Variables and Types (2)

```
num1 = 3
num2 = 3   # num2 type is integer
num2 = 2.3  # now num2 type is float
name = "Ali"
print( name, num1, num2)
print(type(2))
```

- <class 'int'>

```
fullname = name + " " + "Mortazavi"
```

Operators

- + - * / // % & | ^ ~ << >>
- **
  - 2 ** 3 #[gives 8]
- Can be used with strings
- "*" * 10
- "Hello" + " World"
- == != > < >= <= and or in is not
- += -= *= **= /= //=
- Multiple Assignment:

  a , b, c = 1, 2, 3.4

  a, b = b, a

  a = b = c = 3

Text processing (1)

- "Hello" or 'Hello'
- str(2) # convert number to str type
- s = "Hello World"
- s[4] # o
- s[1: 5] # ello
- "World" in s # True
- """ multi line

  string """
- All strings are unicodes
- Put b before string makes it byte coded string

  b"Hello World"

Text processing (2)

- A = "Hello, World: from python "
- A.find(":") # returns 12
- A.rfind("o") # returns 23 (reverse find from end)
- A.strip() # remove white spaces from start and end
- A.split(":") # splits string with given char
  - Output: ['Hello, World', ' from python']
- A.partition(":") # similar to split with a little bit different output
  - ('Hello, World', ':', ' from python')

Lists

- Just put values (of same or different type between []
- A = [1 ,2 , 3, 4]
- B = [1, "hi", 3.34, 2]
- A[0] # 1
- A[3] = 5 # A = [1, 2, 3, 5]
- A + B # [1 ,2 , 3, 4, 1, "hi", 3.34, 2]
- 3 in A # True
- len(A) # the length 4
- A.push(3) or A.append(3)
- Other functions: pop, remove, reverse, sort, find, clear (try using them)

Slice and index

- A = [1 ,2 , 3, 4]
- A[2] # 3
- A[2:] # [3, 4]
- A[1:2] # 2
- A[0:-1:2] #0 3
- A[1:3] # [2, 3]
- A[-1] # 4
- A[-2] # 3

Dictionary

- Name value pairs!

  d = {"name": "Hooman", "age": 12, "School": "Beheshti" }
- print(d["name"])) # Hooman
- d["age"] = 13
- del d["name"] # {"age": 12, "School": "Beheshti"}
- d.keys() # get keys
- d.values() # get values
- Other functions: clear, items, pop

Tuple

- Tuples are lists that cannot be changed
- T = (1, 3, "ali")
- T[1] # 3
- T[0] = 2 # Error
- list(T) # converts tuple to list

if elif else

```
a = 1
if a == 1:
    print("a is equal to 1")
elif a == 2:
    print("a is equal to 2")
else:
    print("a is not 1 or 2")
```

while loop

```
x = 1
while x <= 10:
    print(x)
    x = x -1
    # ...
```

- “continue” and “break” statements exist like other languages

for loop (1)

```
for iterating_var in sequence:
   statements(s)

for member in [1 ,2 ,3]:
    print(member)

fruits = ['banana', 'apple',  'mango']
for index in range(len(fruits)):
   print('Current fruit :', fruits[index])
```

for loop (2)

- Range(10) # 0 1 2 3 4 5 6 7 8 9
- range(1, 10) # 1 2 3 4 5 6 7 8 9
- range(1, 10, 2) # 1 3 5 7 9

```
for a in range(2,5):
    print(a)
```

Import and Import from

- Import <package>
- from <package> import <member>
-
- import random
- from nltk import BigramTagger

Functions

```
def functionname( parameters ):
   function_suite
   return [expression] # optional


def hello(name):
    print("Hello", name)

def calculate(num1, num2):
    Sub = num1 – num2
    Add = num1 + num2
    Return Sub, Add


hello("Hooman")
calculate(2 , 3)
```

Exceptions

```
try:
    [code]
except Exception as e:
    [handle exception here]
finally:
    [code]


try:
    int("s")
except Exception as e:
    print("Error is :", e)
```

File Handling in Python (1)

- file object = open(file_name [, access_mode][, buffering])

- Reading a file:

```
fo = open("foo.txt", "r+")
st = fo.read()
print ("Read String is : ", st)

fo.close()
```

- Reading certain bytes:

```
st = fo.read(10) # the number of bytes to be read
St = fo.readline() # to read line by line
```

File Handling in Python (2)

- file object = open(file_name [, access_mode][, buffering])

- Write to the file

```
fo = open("foo.txt", "w")
fo.write( "This is my first line\n")

fo.close()
```

File Handling in Python (3)

- file object = open(file_name [, access_mode][, buffering])

Reading a file (better way):

```
with open('filename', ‘r’) as f:
    read_data = f.read()
```

```
with open('filename', ‘r’) as f:
    for line in f:
        print(line)
```

Classes and objects (1)

```
class Temprature:
   # init and a constructor
   def __init__(self, temp, temp_type):
      self.temp = temp
      self.temp_type = temp_type

   def displayTemp(self):
     print ("Temprature is:", self.temp)

   def displayType(self):
      print ("Temprature type is:", self.temp_type)
```

Classes and objects (2)

T = Temperature(25, “Celsius”)

T.temp # 25

T.temp_type # Celsius

T.displayTemp() # Temprature is: 25

T.displayType() # Temprature type is: Celsius

Classes and objects (3)

- self is like this in java
- __init__ function is a constructor and initializer (variables are defined in __init__)
- Inside class definition, use self. to access class members
  - self.temp

Socket Programming - Server

- import socket
- Create socket
- Bind addresses
- Listen to the socket
- Accept connection
- Transfer data
- Close connection
- Close socket

Socket Programming - Client

- import socket
- Create socket
- Connect to the server
- Transfer data

Server

```
import socket
HOST = ''
PORT = 3000
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen(10)
    while True:
            conn, addr = s.accept()
            with conn:
                print('Connected by', addr)
                while True:
                    data = conn.recv(1024)
                    if not data:
                        break
                    print("Got data:", data)
                    conn.send(b"Hello from server side!\n")
                    conn.send(b"This is second message\n")
                    conn.close();
                    break
```

Client

```
import socket

HOST = “”    # The remote host
PORT = 3000              # The same port as used by the server
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(b'Hello, world')
    data = s.recv(1024)
    print('Received', repr(data))
```

Attention

- Best practices
- PEP8
- Use Cheat Sheets
- Run import this

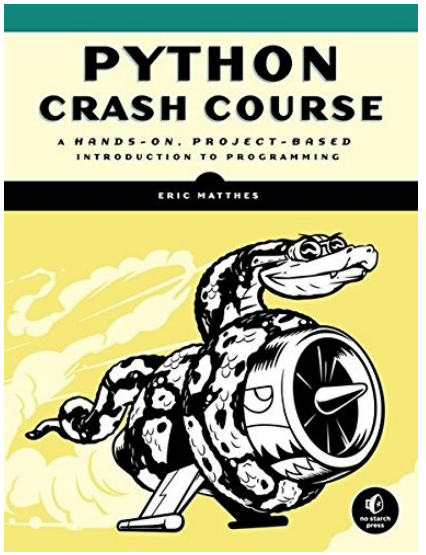
Exercise 1

- Write a simple HTTP server with python using sockets!
- The server should send HTTP requests (use HTTP headers)
- Use file api of python to serve html file and other simple resources
- Appropriate HTTP codes (404, 403, 200 ,etc)
- Browser must be able to send requests to the server
- Deadline: the next week!

Read and practice more

- Use online sources
- Python Crash Course: A Hands-On, Project-Based Introduction to Programming

  by Eric Matthes



- Learning Python, 5th Edition

  by Mark Lutz