Introduction to python

Introduction to python

- Python is a interpreter, high-level programming language for general purpose programming source: wikipedia (http://www.wikipedia.org/wiki/Python_(programming_language))
- Created by Guido van Rossum in 1991.
- Python puts an emphasis on code readability which has inspired many other languages to do so.
- Some of the programming language features include duck typing, (dynamic typing), support for multiple programming paradigms, iterators and decorators.
- It can be extended or embedded.
- It can be used to glue with other code written in other programming languages like C or C++.

Getting Started

- Installation: * Python can be downloaded in Windows, OS X and Linux operating systems from the python website (http://www.python.org/downloads)

IDLE : Idle is a development environment for python available for both Windows and OS X platforms.

Note about versions :
There are currently two major versions available Python 2 and Python 3. There are enormous differences in both the versions, the older version is not prefered of late.

Diving In - Hello World

In the shell as well as using as a program

```
print "Hello World!"
```

A more illustrative code

```
def greet():
    '''
    prints a hello world message to the screen
    '''
    print("Hello World!")

#execution starts here
greet()
```

Variables

- In python, variables are references to python objects.

- In python **everything** is an object.

```
x = 2
```

- Variables are dynamically typed which means you can assign any type of data to any variable.

```
x = 2
x = "hello"
x = 39.9769
x = [1,2,3,4,5]
x = {'a': 1,'b':2,'c':3}
```

Variable naming rules and conventions

- Variable names can contain any letter, number or underscore (_).
- Variable names should not begin with a number, they can begin with an underscore
- Python variables are generally named in lower_case_with_underscores
- Checkout python's coding style guide (https://www.python.org/dev/peps/pep-0008/) for naming conventions.

Basic Data types

bool: True, False
str: "Python", 'programming', r'[a-z]+'
"""
This string
spans
multiple lines
"""
int: 2,90,10000
float: 2.5,2.788
complex: 1+2j

Basic Data types - Part II

```
x = None
```

None is a special object that is its own type.

NaN

Basic Data types - Inspecting a data type

```
type(obj)
dir(obj)
isinstance(obj,<type>)
obj is None
obj is not None
```

Data types Contd. - Collections

- lists [] , [1,2,3.5]
- tuples () (1,‘2’,3)
- dict {} {‘a’:1,’b’:2,’c’:3.5}
- set

Mutable vs Immutable objects

- A mutable object is one which can change its properties dynamically.
- Certain builtin object types are immutable (ex. string, tuples, integers, floats)
- Most of the custom objects in python are mutable.

Operators

+, -, /, *,//,%,**

Relational operators
!=,\<,>,>=,<=,==,

Assignment operators:
\=,+=,-=,*=,/=,//=,%=

Logical operators:
and, or, not

List of operators can be seen in the documentation in the python website (https://docs.python.org/3/reference/lexical_analysis.html#operators)

A Note on assignment operator: Since there is no static typing or variable declaration in python, an object is created when the object is assigned to a variable for the first time. And similarly a variable type is determined by the value assigned to it.

- When a variable is assigned a value, an object is created and its value and type are binded to the variable.
- values can be assigned to more than one variable in a single statement. This is known as variable unpacking.

```
x,y = 1,2
```

Sequences in python

- Lists
- Heterogenous collection of objects
- List operations include add, remove, extend,

```
>>> a = [1,2,3,4,5]
>>> a.append(6)
>>>a
[1,2,3,4,5,6]
>>> a.remove(4)
[1,2,3,5,6]
>>> a.pop()
6
>>>a
[1,2,3,5]
>>>a.extend([1])
>>>a
[1,2,3,5,1]
>>>a.count(1)
2
>>>
```

- Lists can be concatenated

```
>>>a = [1]
>>>b= [2,3]
>>>c = a+b
>>>c
[1,2,3]
```

*List elements can be viewed by using the [] operator

```
>>> a = [1,2,3]
>>> a[0]
1
```

Sequence contd. - Dictionaries

- Is a collection of heterogeneous key-value pairs.

```
>>> a = { 'a': 3,'b' : 4 }
>>> a['a']
3
>>> a.get('c','No key')
'No key'
>>> a.pop('b')
4
>>> a['a']
3
>>> a['d'] = -1
>>> a
{'a':3,''d':-1}
```

Sequence contd. - Tuples

- Is a collection of heterogeneous collection of objects.

```
>>> obj  = (1,2.34,"string")
```

- Tuples are immutable. (i.e) the values in the tuple cannot be changed.

```
>>> obj[0] = 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>
```

Sequences slicing and indices

```
>>> t = [1,2,3,4,5]
>>>t[0]
1
```

- Python allows negative indexing. -1 would refer to the last value of the list and -2 the second last and so on.

```
>>> a = [1,2,3,4]
>>> a[-1]
4
>>> a[-3]
2
```

- Python also allows selecting more than one index. This is called slicing.

```
>>> a [1:3]
[2,3]
```

- Slicing can take the form obj[start:stop:step], but the stop, and step values can be avoided.
- Step can be given to select nth value of the list

```
>>> a = [1,2,3,4,5,6,7,8,9]
>>> a[0:8:2]
[1,3,5,7]
>>> a[1:8:2]
[2,4,6,8]
>>> a[1:8:3]
[2,5,8]
```

- Step and stop can be negative

```
>>> a[8:5:-1]
[9,8,7]
```

The *in* operator

Consider a scenario where you need to check whether a given element is in the sequence or not and based on that you need to do some operation.
You can loop through the list and compare every element or

```
>>> 0 in a
False
>>>1 in a
True
```

Control Flow statements

- if-else statements

```
string = ""
x = 3
if x % 2 = 1:
    string = "Odd"
else:
    string = "Even"
```

- elif

```
if x > 0:
    string = "Positive Integer"
elif x < 0:
    string = "Negative Integer"
else:
    string = "Zero"
```

- While loops

```
while x < 3:
    x = x + 1
```

- break and continue
- break statement is used to leave the loop completely.

```
x = 0
a = []
while x<5:
    x = x+1
    a.append(x)
    if x == 3:
        break
#a = [1,2]
```

- continue statement is used to force the python to go to the next iteration of the loop

```
x = 0
a = []
while x<5:
    x = x+1
    if x == 3:
        continue
    a.append(x)

#a = [1,2,4,5]
```

- assert statement is used to ensure whether a certain condition is satisfied in the program
- when condition within the assert statement fails, python raises a AssertionError and stops execution

```
>>>assert 1==True
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
```

- for loops
- for loops in python can only be iterated over an iterator object.
- Any of the sequence object mentioned above are iterators
- A for loop steps over each item in the sequence or an ‘iterable’ and performs the steps inside the loop.

```
for <item> in <iterable>:
    ....
```

ex:

```
a = [1,2,3]
b= []
for i in a:
    b.append(i*2)
```

- can be more complex than a single object. For example enumerate function takes a list and returns a tuple of list of index,value pairs or the items method in the dictionary

```
for index,value in enumerate(a):
    ....
d = {'a':1,'b':2}
for key,value in d.items():
    ....
```

Functions

- Functions are a block of code, that can be reused.
- It is used to make the program more modular.
- enumerate function, range function are builtin functions

ex:

```
def greet(name):
    string = 'Hello, ' + name '!'
    return string
```

- Calling the function:

```
>>> greet("John")
Hello, John!
```

Every function has
a) name : the name with which it is called
b) arguments : variables that are inputs to the function
c) return value: values that are the output of the function

- Functions can be without return value or arguments
- When a function does not return a value, python automatically returns the None object

Remember **everything** is an object in python.

- Functions are first class objects in python. which means functions can be
  1) passed as an arguments to other functions
  2) returned as a result
  3) can be assigned to variables
  4) can be assigned to indices in lists, tuples or dictionaries
- Some standard (builtin) python functions
- str: convert any object to string object.

```
>>> str(5)
'5'
>>>str(4.5)
'4.5'
>>> str([1,2,3])
'[1,2,3]'
```

- print: print some objects to the console

```
>>>print(1,'Hello',5.5
1 Hello 5.5
```

print automatically automatically converts each object to string using the str function and prints them with a space as a separator.
Of course we can change what we want as separator too.

```
>>>print(1,'Hello',5.5,sep=',')
1,Hello,5.5
>>>print(1,'Hello',5.5,sep=',')
1
Hello
5.5
```

- Also print function adds a newline at the end which can also be modified if required

```
>>> print("Hi",end=",")
Hi, >>>
```

- input: gets a string as input from the user in the console
- the string can be converted to any format using the respective type functions.
- a string can be passed to the input function which will printed before asking for input

```
a = input("Enter an integer")
b = int(a)
print( type(b))
# <class 'int'>
```

*zip: takes two or more iterables and returns a list of aggregated tuples

```
a = [0,1,2]
b= [4,5,6]
for item in zip(a,b):
    print(item,end='|')
print
#(0,4)|(1,5)|(2,6)|
```

List of all built functions available can be found in the python docs here (https://docs.python.org/3/library/functions.html)

Exceptions

- An exception is an error that happens during execution of a program.
- Python generate an exception that can be handled, which avoids your program to crash.

Handling exception

- Exceptions are handled by specifying the code for special case in the *except* block

```
try:
    #code where error occur
    ....
except A:
    #code to handle the special case
    ...
except B:
    ....
except C:
    .....
else:
    #action to perform if there is no exception
    ...
finally:
    #cleanup action
    ....
```

- **try** block has the code that might raise an error or exception
- **except** block has the code that is to be executed when an exception occurs
- **else** block contains the code that is to be executed when no error occurs
- **finally** block contains the cleanup code that is to be performed on any case (error occurs or does not occur)

Example:

```
data = ''
try:
    fobject = open("test.txt")
except OSError as e:
    print(e)
    exit(0)
else:
    data = fobject.read()
finally:
    fobject.close()
```

Modular programming

- Logical organisation of the python code
- Contains python objects like functions, classes, variables, code etc.
- Modules can contain submodules

Creating a module

- A module can be created by simply creating a python file
- Any folder that contains the python file `__init__.py` is a module and all the other python files in it are considered submodules.

```
wc
|-- __init__.py
|-- core.py
|-- helpers.py
tests
|-- test_core.py
|-- test_helpers.py
```

importing a module using `import`

- A module can be imported using the import statement
  Ex:

```
import os
```

- Submodules can be imported by specifying the name with a `.`
  ex:

```
import os.path
```

- Objects can be used in the code using the `.` operator
  ex:

```
import os
files = os.listdir()
```

- Objects can be selectively imported into the global namespace using the `from <module> import <object>` syntax
  ex:

```
from os import listdir
files = listdir()
```

- All the objects in the module can be imported into the global namespace using the `from <module> import *` syntax

ex:

```
from os import *
```

Standard Library

- A huge amount of libraries to help get things done
- Just for fun:

```
import this
import antigravity
```

Third party modules

- Apart from the sea of standard library modules there is an ocean of 3rd party packages
- Some of the famous ones include SQLAlchemy, numpy, matplotlib, django, requests
- Python has a python package index [Pypi] for python packages.
- pip is a tool that is available in python 3 and above, to install, uninstall, upgrade python packages.

An example code

```
'''
Takes a file and prints the number of words, number of lines and the number of characters in that
file.
'''
import sys

def count_words(string):
    words = string.split()
    return len(words)

def count_lines(string):
    lines = string.splitlines()
    return len(lines)

def count_chars(string):
    return len(string)

def wc_facade(string):
    return count_words(),count_lines(),count_chars()

if __name__ == "__main__":
    filepath = sys.argv[1]
    try:
        fobject = open(filepath)
    except FileNotFoundError:
        print("File not found in the path specified")
    finally:
        fobject.close()
    string = fobject.read()
    words,lines,chars = wc_facade(string)
    print(words,lines,chars)
```

- argv is the list in the sys module that stores the command line arguments to the program
- wc_facade function implements the facade pattern that simplifies the task of calling all three functions
- `if __name__ == '__main__':` line ensures that code executes only when its is called by interpreter directly and not when the file is imported.

Detour to functional programming

- A functional programming language is one where the functions in the language are pure.
- Some of the concepts of functional language available in python are higher order functions, lambdas, list comprehension, function currying , closures

Higher order functions

- A function is said to be a higher order function when it takes another function as a parameter or returns a function as output or both.
- Decorator or wrapper functions etc.

map,filter, reduce

- Map takes two parameters : a function and an iterable, and applies the function for each element in the iterable and returns the new iterable
  ex:

```
def inc_by_2(num):
    return num+2

a = [1,2,3,4]
b = list(map(inc_by_2,a)) #b = [3,4,5,6]
```

```
def clean_string(string):
    string = string.strip()
    string = string.lower()
    return string

a = ["Hello", " hello", "    hello   ", "  HEllo","HeLLO"]
b=  list(map(clean_string,a)) #b contains 5 "hellos"
```

- Filter is similar to map but the function can only return True or False. the filter returns an iterable which return true when applied to the function

ex:

```
def even(num):
    return num%2 == 0

a = [1,2,3,4]
b = list(filter(even,a)) #b = [2,4]
```

- Reduce takes a function and iterable. the function takes two parameters and returns a single. The reduce function applies the function for the first two, then the result with the next item and its result with the next item until it reduces the iterable to a single value.
  ex:
  from functools import reduce
  def add(a,b):

return a+b

a = [1,2,3,4]
b = reduce(add,a) #b = 10

lambdas

- Lambdas are anonymous functions that are meant to be used within the context of the place where it is defined.
  ex:

```
a = [1,2,3,4]
b = list(map(lambda i : i+2,a)) #b = [3,4,5,6]
c = list(filter(lambda i: i%2==0,a) # b= [2,4]
d = reduce(lambda a,b: a+b, a)
```

An example code

```
'''
Get the count of each word in a file
'''
def word_histogram(string):
    words = string.split()
    words = map(lambda i :i.strip().lower(), words)
    keys = set(words)
    wh = {}
    for key in keys:
        wh[key] = words.count(key)
    return wh
```

Friday, 29. June 2018 01:46AM