Introduction to Python

What is Python ?

A modern, object-orientated, high-level programming language adopted by a wider community !

- Easy-to-learn: simple syntax and intuitive code
- Expressive: can do a lot with a few lines of code
- Interpreted: no need to compile
- Dynamically typed: no need to define the types of variables
- Memory managed: no c-style `memory leak' bugs

Why Python ?

- Python is widely used in the scientific computing community
- Extensive ecosystem of rapidly maturing libraries
- Good performance - closely integrated with time-tested C and Fortran code : blas, atlas, lapack etc.
- No license costs (i.e., free!) and easy to install
- Useful language for data analytics and beyond !

More Information

Official website: http://python.org/

- documentation, tutorials, beginners guide, core distribution, ...

Books include:

- ✓ *Learning Python* by Mark Lutz
- ✓ *Python Essential Reference* by David Beazley
- ✓ *Python Cookbook*, ed. by Martelli, Ravenscroft and Ascher
- ✓ http://wiki.python.org/moin/PythonBooks

Versions of Python

- ❑ Python Version 2.x (2.7 widely used) , Version 3.x (3.5 and above)
- ○ Please be aware that code written for 2.7 may not work under 3.5 and vice versa due to syntactical differences and changes.
- In a terminal window, type >>> python –V to check the python version.
- For this class all code will use Python 3.6.6

Activating Python 3.6.6 version (from Virtual environment)

- Activate/Load 3.6.6 version Python by initiating the following command

```
>>> module load python/3.6.6
```

- Deactivate the module once you finish work

```
>>> module unload python/3.6.6
```

Development Environments

What IDE to use?

http://stackoverflow.com/questions/81584

- Spyder
- PyDev with Eclipse
- PyCharm
- Vim
- TextMate
- Gedit
- Idle
- PIDA (Linux)(VIM Based)
- NotePad++ (Windows)

Python Interactive Shell

Initiate an interactive session with Python by simply typing “python” without the quotes from the command-line. You will get back a message giving Python version information.

You can type things directly into a running Python session

```
>>> 2+3*4
14
>>> name = "Andrew"
>>> name
'Andrew'
>>> print ("Hello“), name
Hello Andrew
>>>
```

Basic data types

Variable are dynamically typed, i.e. no need to explicitly declare the type.

Eg, In the python interactive shell

```
>> a =1
>> type(a)              outputs  <class ‘int’>
>> b = 1.0
>> type(b)              outputs  <class ‘float’>
>> x = True
>> y = False            outputs  <class ‘bool’>
>> type(x)
>> name = “Adam”
>> type(name)           outputs  <class ‘str’>
```

Operators

Arithmetic (+, -, *, /, %) : assume a = 10, b =4

- a+b # add
- a-b # subtract
- a*b # multiply
- a/b # division
- a%b # modulo operation, returns the remainder

Relational/ Comparison (>, <, >=, <=, ==, !=)

a > b True if a greater than b else False

a < b True if a lesser than b else False

a >= b True a is greater than or equal to b else False

a <= b True if a is less than or equal to b else False

a == b True if a is equal to b else False

a ! = b True if a is not equal to b else False

Operators contd ..

Logical (and, or, not) : a=10,b=4,c=2

a > b and a > c   True if a is greater than b and c else False

a > b or a > c   True if a is greater than either a or b else False

not a > b   True if a lesser than b else False

Decision making and looping

if- elif- else

```
marks = 50

if marks > = 70:
   print(‘1st class’)
elif marks >= 40:
   print(‘Passed’)
else:
   print(‘Failed !’)
```

In python 2.7 just use print ‘….’. No parenthesis in 2.7

for loop

```
data=[1,2,3,4,5]
sum=0

for item in data:
      sum += item

# loop ends here

print (sum)
```

Note:
sum += item is short-hand notation of writing
sum = sum + item

While loop

```
data = [1,2,3,4,5]
n = 5
i =0
sum=0

while i < n:
      sum += data[i]
      i += 1

# loop ends here

print(sum)
```

Function

We can write our own functions using the keyword ‘def’. General syntax:

```
def function_name(parameters):
      body of the function
```

Example: finding sum of elements in a list using function

```
def sum_list(data):
      """ Returns the sum of a list of numbers"""    # Good habit to put docstring !!

      sum = 0
      for x in data:
            sum += x
      return sum
```

Using the above function:

```
result = sum_list([1,2,3,4,5,6,7,8])
print(result)
```

Functions: returning multiple values

```
def max_min(data):
     """ Returns min, max from the list """
     min, max = data[0], data[0]

    for item in data:
       if item < min:
           min = item
       elif item > max:
           max = item
    return max, min
```

This function returns two values that will be returned as a tuple !

USAGE:

```
max, min = max_min([1,7,2,9,12,-7])
print(‘Max  is ’ + max + ‘and min is ’+ min)
```

Functions: named and default parameters

- In function call, parameters can be explicitly named and we can initialize them with default values.
- Default and named parameters can also be conveniently combined.

1) Named parameters

```
def compute_sum(a,b):
      return a+b
```

Usages:

```
ans = compute_sum(10,20)
ans = compute_sum(b=20,a=10)
```

2) Named+default parameters

```
def compute_sum(a, b, addC=False, c=99):
      if addC:
         return a+b+c
      else:
         return a+b
```

Usages:

```
ans = compute_sum(10,20,True)
ans = compute_sum(b=10,a=20,True,200)
```

Compound types

- Strings
- Lists
- Tuples
- Dictionaries

Strings

Functionalities associated with strings are defined in “str” class

```
hello = 'hello'                               # String literals can use single quotes
world = "world"                               # or double quotes; it does not matter.

print(hello)                                  # Prints “hello”
print(len(hello))                             # String length; prints “5”
msg = hello + '  ' + world                    # String concatenation
print(msg)                                    # prints “hello world”
print( '%s %s %d' % (hello, world, 2018))     # prints “hello world 2018”
```

Note: statements starting with # symbol are treated as comment and ignored during execution.

String methods.

```
1. s = "hello"
2. print(s.capitalize())          # Capitalize a string; prints "Hello"
3. print(s.upper())               # Convert a string to uppercase; prints "HELLO"
4. print(s.rjust(7))              # Right-justify a string, padding with spaces;
                                   # prints "  hello"
5. print(s.center(7))              # Center a string, padding with spaces;
                                   # prints " hello "
6. print(s.replace('l', '(ell)'))   # Replace “l” with “(ell)” in hello ;
                                    # prints "he(ell)(ell)o"
7. print('  world  '.strip())       # Strip leading and trailing whitespace;
                                    # prints “world”
```

We have rstrip and lstrip functions also !!

```
8. print(‘  world  ‘.lstrip())      # prints “world  ”    # left whitespace removed
9. print(‘  world  ‘.rstrip())      # prints “  world”    # right whitespace removed
```

String Methods: find, split

```
Smiles = “C(=N)(N)N.C(=O)(O)O”


>>> smiles.find("(O)")
15
>>> smiles.find(".")
9
>>> smiles.split(".")
['C(=N)(N)N', 'C(=O)(O)O’]
```

Use “find” to find the start of a substring.

Find returns -1 if it couldn’t find a match.

Split the string into parts with “.” as the delimiter

String operators: in, not in

```
if “Br” in “Brother”:
    print “contains brother”
```

```
email_address = “clin”
if “@” not in email_address:
    email_address += “@qmul.ac.uk”
```

You can find a list of all string methods in the documentation.

Lists

A list is a compound data type that is resizable and can contain elements of different types:

```
1. x = [5,6,8]            # Create a list
2. print(x, x[2])         # Prints "[5,6,8] 8"
3. print(x[-1])           # Negative indices count from the end of the list; prints ??
4. x[2] = ‘hello'         # Lists can contain elements of different types
5. print(x)                # Prints "[5,6, ‘hello’]”
6. x.append(‘qmul’)        # Add a new element to the end of the list
7. print(x)                # Prints "[5,6,’hello', ‘qmul’]”
8. y = x.pop()             # Remove and return the last element of the list
9. print(y, x)             # Prints “qmul [5,6, ‘hello’]”

10. print(len(x))           # gives the length i.e. the number of elements in the list
```

Detailed documentation can be found here

Handy list functions

append, count, extend, index, insert, pop, remove, reverse, sort

```
days = ['Mon', 'Tues', 'Weds', ‘ Thur', 'Fri', 'Sat', 'Sun’]

>>>days.reverse()      # reverses a given lists
>>>days.sort()          # sorts the lists (here alphabetically, Fri appears first)

x = [1, 2, 3, 4]
>>> x.extend([5, 6, 7])   # now, x = [1,2,3,4,5,6,7]

x = list('Let us go then, you and I’)
>>>x.count(‘e’)      # returns the number of times ‘e’ appeared in x;
                          # Prints 2
>>> x.count(‘ ’)       # prints 6
```

Slicing operation

Simple mechanism to access a sub-lists from a given lists.

```
nums = list(range(5))     # range is a built-in function that creates a list of integers
print(nums)               # Prints "[0, 1, 2, 3, 4]"
print(nums[2:4])          # Get a slice from index 2 to 4; prints "[2, 3]“

print(nums[2:])           # Get a slice from index 2 to the end; prints "[2, 3, 4]"
print(nums[:2])           # Get a slice from the start to index 2 (exclusive);
                          # prints  "[0, 1]"
print(nums[:])            # Get a slice of the whole list; prints "[0, 1, 2, 3, 4]"
print(nums[:-1])          # Slice indices can be negative; prints "[0, 1, 2, 3]“
print(nums[:-2])          # Slice indices can be negative; prints "[0, 1, 2]“
nums[2:4] = [8, 9]        # Assign a new sublist to a slice
print(nums)               # Prints "[0, 1, 8, 9, 4]“
```

** This operation would be applicable with NumPy arrays as well !

Looping in lists

```
names = [‘John’, ‘Mikey’, ‘David']
for name in names:
    print(name)

# Prints the following
John
Mikey
David
```

List comprehension

```
nums = [0, 1, 2, 3, 4]
squares = [ ]
for x in nums:
    squares.append(x ** 2)
print(squares)                    # Prints [0, 1, 4, 9, 16]
```

Using list comprehension (compact form)

```
squares = [x ** 2 for x in nums]
print(squares)                    # Prints [0, 1, 4, 9, 16]
```

List comprehension with conditions

```
even_squares = [x ** 2 for x in nums if x % 2 == 0]
print(even_squares)          # Prints "[0, 4, 16]"
```

Tuples

Tuples are immutable lists, i.e., they cannot be modified once created.

```
x = (1, 2)              # note ( ) for tuple and [ ] for lists
type(x)                 # prints <class ‘tuple’>
x = 1, 2                # the brackets are not strictly necessary
type(x)                 # prints <class ‘tuple’>
print x[0]              # prints 1
x[0] = 5                # Error. Tuples are immutable!

pos = (10, 20, 30)
(x, y, z) = pos                      # 'unpack' tuple into separate variables
pos1 = (10, 20, 30)
pos2 = (10, 25, 30)
pos1 == pos2                         # true iff all elements equal
x, y = 10, 15
x, y = y, x                          # Can swap variables with a single line!
print x, y
```

Dictionaries

A dictionary maps from a unique key to a value. Duplicate keys are not allowed. Duplicate values are just fine

```
>>> office = {‘Dan’: 104, ‘John':146, ‘Chris':245}
>>> type(office)         # prints <class ‘dict’>
>>> office[‘Dan’]       # look up a value for a given key; prints 104
>>> office[‘Jose’]       # throws exception if key doesn't exist
```

Operations with dictionaries

```
>>>office[‘Jose'] = 282           # add a new key-value pair
>>>print(office)                  # prints all content with Jose added as last entry

>>>office.keys()                  # return the list of keys
>>>office.values()                # return the list of values
>>>office.has_key(‘Jose’)        # check if a key exists

>>>del office[‘John']            # remove an element from a dictionary
>>>print(‘Dan' in office)        # check if a dictionary has a given key
```

Loops: It is easy to iterate over the keys in a dictionary:

```
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal in d:
    legs = d[animal]
    print('A %s has %d legs' % (animal, legs))

# Prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs"
```

If you want access to keys and their corresponding values, use the items method:

```
for animal, legs in d.items():
    print('A %s has %d legs' % (animal, legs))

# Prints "A person has 2 legs", "A cat has 4 legs", "A spider has 8 legs“
```

Dictionary comprehension

These are similar to list comprehensions, but allow you to easily construct dictionaries. For example:

```
nums = [0, 1, 2, 3, 4]
even_num_to_square = {x: x ** 2 for x in nums if x % 2 == 0}
print(even_num_to_square)  # Prints "{0: 0, 2: 4, 4: 16}“
```

More detailed documentation here

File reading/writing

General syntax:

```
f = open(filename, ‘mode’)
```

Example:

```
input_file = open(“in.txt“, “r”)
output_file = open(“out.txt", "w")
for line in input_file:
        output_file.write(line)
```

“w” = “write mode”
“a” = “append mode”
“wb” = “write in binary”
“r” = “read mode” (default)
“rb” = “read in binary”

Program writing basics !

- Python programs are written and saved in a file with .py extension. Example first_program.py
- Executing the program: python first_program.py
- Use modular programming approach: use of functions !
- Keep a habit of documenting your code !

Modules

- ❑ When a Python program starts it only has access to a basic set of functions and classes. Example “int”, “dict”, “len”, “sum”, “range”, ... etc
- ❑ “Modules” contain additional functionality.
- ❑ Use “import” to tell Python to load a module.
- ❑ Example: **import math**

Other forms of import statements

- ✓ from math import cos, pi
- ✓ from math import *

Classes

The syntax for defining classes in Python is straightforward:

```
class Message(object):

    # Constructor
    def __init__(self, message):
        self.msg= message                 # Create an instance variable

    # Instance method
    def display(self, capitalize=False):
        if  capitalize:
            print('%s !' % self.msg.upper())
        else:
            print(' %s' % self.msg)

m = Message(‘Welcome all’)   # Construct an instance of the Message class
m.display()                  # Call an instance method; prints “Welcome all“

m.display(capitalize=True)   # Call an instance method; prints “WELCOME ALL !"
```

Next week

Week 7: 6th March

Lecture : 9:00 – 11:00

- ✓ Introduction to Numpy - numerical Python library
- ✓ Introduction to Machine learning and Scikit Learn

Lab Classes: 11:00 – 13:00

- ✓ Numpy tutorial and some exercises
- ✓ Tutorial on ML using scikit learn library

  * Walk-through notebook files