Python

- Python® is a dynamic object-oriented programming language that can be used for many kinds of software development
- Created by Guido Van Rossum in 1990 from CWI - The National Research Institute for Mathematics and Computer Science in the Netherland.
- An interpreting language, no compilation and links are needed. Simple to use, a “real” programming language
- Two running modes
  - Interactive
  - Normal (python script)
- Support multiple platforms
  - Unix, Linux, MS Windows, MacOS X
- It is FREE, GNU GPL

References

- http://www.python.org
- Python tutorial by Rossum released Sept 19, 2006 http://docs.python.org/tut/
- Online book: Dive into Python at http://www.diveintopython.net/
- Some good resources for Python GUI
  - http://www.python-course.eu/python_tkinter.php
  - https://docs.python.org/3/library/tk.html
  - https://www.tutorialspoint.com/python/python_gui_programming.htm

History of Releases

- The Linux systems in CS lab have version 2.7 and version 3.7

- Oldest but widely used release
  - Python 1.5.2 (April 13th, 1999)

Interactive Python

- Command line within python
- Return immediate feedback for each statement
- Run another program within python shell (this is for Python2,removed from python3): `execfile("hello.py")`
- Exit/quit interactive python with `exit(),quit(),Ctrl-d"`

```
hlin@shrike:~$ pyhon
No command 'pyhon' found, did you mean:
 Command 'python' from package 'python-minimal' (main)
pyhon: command not found
hlin@shrike:~$ python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> x=10
>>> y=20
>>> x+y
30
>>> print "hello world"
hello world
>>>
```

Python Scripts

```
  # compute compound-interest
Print("Hello World!“)
amount = 1000
rate = 0.05
years=5
year=1
while year <= years:
    amount = amount*(1+rate)
    print(year, amount)
    year = year +1
```

- No semi-colons needed for each statement
- No “$” for variable names
- No curly braces for blocks
- Indentation is used to denote different blocks of the code
  - Body of the functions
  - Loop
  - class
- Indentation MUST be consistent

Data Types- Number

Python supports int, float, complex number, etc

```
>>> 2+2
4
>>> 50-6*6
14
>>> 50-6*6/4
41.0
>>> (50-6*6)/4
3.5
>>> 8/5
1.6
>>> 8%5
3
>>> 8//5
1
>>> 5*2 + 2
12
>>> 2**16
65536
>>> round(10/3,2)
3.33
```

Data Types-String

- Create string literals, enclose them in single, double or triple quotes
  - `a = 'Hello World'`
  - `b = "Python Programming"`
  - `c = """what is your name? """`
    - Triple quotes for long string of multiple lines
- Strings are sequences of characters indexed by integers starting with zero
  - Get sub string using slicing operator `[i:j]`, from index `i ≤ k < j`
    - `a='1234567890'`
    - `SubA = a[0:5]` ➔ you get: 12345
    - `SubA = a[6:]` ➔ 7890
    - `subA = a[:6]` ➔ 123456
    - `subA = a[2:5]` ➔ 345; a[-6]?
- Strings are concatenated with plus(+)

```
newstr = old1 + old2 + " This is a test"
Newstr = 2*class + " " + classname
```

String Functions

- In string module
  - `S.capitalize()`: Capitalize the first letter
  - `S.upper()`
  - `S.lower()`
  - `S.atoi()`
  - `S.atof()`
  - `len(S)`
  - `S.split(sep[, maxsplit]])`
  - `Sept.join(wordlist)`
- Regular expression in `re` module

  http://www.regular-expressions.info/python.html

Date Types- Sequence

- List and tuple are sequences of arbitrary objects
- List
  - enclosed by square bracket. `alist=["Amy", "John", "Alex"]`
  - Index: integer starting at zero
    - `print alist[2]` ➔ `Alex`
  - Add new member to the list with append function
    - alist.append("Smith")
  - Sub list
    - `kylist = alist[1:2]`
    - `print alist[1:3]` ➔ `['John', 'Alex']`
- tuple: members are enclosed by **`parentheses`**
  - `t1 = (x, y, z) or t1 = x, y, z`
- set: unordered collection with no duplicate elements
  - `basket=['apple', 'apple', 'pear', 'peach', 'pear']`
  - `fruit=set (basket)`

List Methods

- `s.append(x)`
- `s.extend(newlist)`
- `s.count(x)`
- `s.index(x)`
- `s.insert(i,x)`
- `s.pop([i])`
- `s.remove(x)`
- `s.sort()`
- `etc…`

Differences b/w LIST & TUPLE

- Generally, lists for a set of homogeneous data

, but tuple can be a set of heterogeneous data

- You can modify the elements of the list, but You can not modify individual elements or append new elements to a tuple after its creation

Data Types: Dictionary

- **A dictionary** is an associative array or hash table that contains objects indexed by keys
  - Generally, string is used for the key
  - But numbers or tuples can be used too
- A dictionary is created by enclosing elements with curly brackets **{ }**

```
phone={"Amy": 5164,
       "John": 5128
      }
```

- **len(phone)** ➔ number of elements
- To access members of a dictionary
  - **amy = Phone['Amy']; john=phone['John']**
- Insert or Modify elements
  - **phone['Smith'] = '5188'**
  - **phone['Amy'] = '8888'**

More about Dictionary

- Key testing using: `has_key('key')`

```
if phone.has_key("Jones"):
  print "I have Jone's phone"
else:
  print "Sorry, I have no Jones' phone"
```

- to get the list of keys or values or both

```
names = phone.keys()
tels = phone.values()
namelist = phone.items()  # list of tuples (name and phone pairs)
```

- Delete elements of a dictionary
  - Remove one element: `del phone["Amy"]`
  - Remove all element: `phone.clear()`
  - Remove entire variable: `del phone`

File I/O

- File read methods
  - `f.read(n)`: read at most `n` bytes, read whole content if n is omitted
  - `f.readline()`: read a line
  - `f.readlines()`: read all lines and return a list of lines
- File write methods
  - `f.write(S)`: write string S to file
  - `f.writelines(L)`: write all strings in list L
  - `f.readlines()`:read all lines and return a list

```
#Open file for read
fr=open("foo.txt")
fw=open("out.txt","w")
line = f.readline()
while line:
  print line,line=f.readline()

f.close()
```

```
name="LIN"
# no need to call close
# function if use "with"
with open("out.txt",'w') as f:
    f.write(name)

print(f.closed)
# should print True
```

Standard I/O

```
date=input("Enter date %s (in yyyy-dd-mm):")
print(date)
```

Formatted String Literals

```
>>> num=3.1415
>>> print(f'The pi is {num:.3f}')
The pi is 3.142
>>> name="pi"
>>> print(f'the {name:10} is {num:5.2f}')
the pi         is  3.14
```

The string format() function

```
print("the {0:10s} is {1:5.2f}".format(name, num))
```

Old Format:

```
>>> print("the value of %10s is %5.2f" % (name, num))
the value of         pi is  3.14
```

File Validation

```
import os.path
# os.path - The key to File I/O
os.path.exists("bob.txt")
os.path.isfile("bob.txt")
```

Reading Options and ENV Variables

- The options are placed in the list: sys.argv

```
#argv.py: print all the command line options
import sys
for i in range(len(sys.argv):
        print "sys.argv[%d]=%s" %(i, sys.argv[i])
```

➔ `python argv.py –n 10 'Lin'`

output ➔ ?

- Environmental variables are saved in the dictionary os.environ
  - `path=os.environ['PATH']`
  - `login = os.environ['LOGNAME']`

  These variables can be modified within python script

Looping Structure

```
# A Python script

alist = ["John", "David", "Alex"]
for name in alist:
   print name

for i in range(len(alist)):
    print i, alist[i]

for n in range(2, 10):
    prime=1
    for x in range(2, n):
      if n % x == 0:
          prime=0
          break
    if prime == 1:
       print n, "is a prime number"
    else:
       print n, "is not a prime
number"
```

```
# A perl script

@alist = ("John", "David", "Alex");
foreach $name  (@alist){ print "$name\n";}

for ($i=0; $i<=$#alist; $i++ )
{  print "$i, $alist[$i]\n";}

for($n=2; $n<10; $n++)
{
   $prime=1;
   for ($x=2; $x<$n; $x++)
   {
      if ( not ($n % $x)) { $prime=0; last;}
   }
   if ($prime){print "$n is a prime number\n";}
   else
   {    print "$n is not a prime number\n";  }
}
```

Exceptions

- Errors detected during execution are called **exceptions**
  - Divided by zero: 10 * (1/0)
  - Failed open file for read or write, etc
- Handling exceptions with **try… except … finally**
  - the finally clause will always be executed before leaving the "try" block

```
try:
    f = open(arg, 'r')
except IOError: print 'cannot open', arg
else:
    print arg, 'has', len(f.readlines()), 'lines'
    f.close()
finally:
    print "always execute this part"
```

Pattern Search in Python

- Simple cases, such as index, find, count can be done using **`string`** functions
  - **`replace(s,old, new [,max])`**
  - **`find(s, sub [, start [, end]])`**
    - Returns the index where “sub” is found
  - **`split(s [,sep [, maxsplit]])`**
    - Returns a list of words
  - **`join(words [, sep])`**
  - **`strip(s)`** remove leading and/or trailing white spaces from s
    - **`lstrip(s)`**
    - **`rstrip(s)`**

Python Regex: re module

- All the operations/functions related to regex are defined in standard module: re, commonly used functions, all these functions return a MatchObject
  - `search(pattern, string [, flags])`
  - `match(patter, string [, flags])`
  - `split(pattern, string [, flags])`
  - `sub(pattern, repl, string [, count=0]`
- The Backslash Plague
  - for pattern: \section, you need to use patter \\\\section
  - Use python “raw” string, re.complie(r’\section’)

Python Function

- Python function is declared with keyword **def**
- Arguments are passed in parenthesis just like C
- Return single value ONLY
  - Use tuple, list, dictionary to return multiple values

```
def fibonacci(n):
   a, b = 0, 1
   print a,
   while b < n:
      print b,
      a,b = b, b+a
   print a, b

fibonacci(4)
fibonacci(80)
```

```
# multiple-returns.py
a, b, c = 0, 0, 0
def getabc():
    a = "Hello"
    b = "World"
    c = "!"
    return a,b,c

#defines a tuple on the fly
def gettuple():
    a,b,c = 1,2,3
    return (a,b,c)

def getlist():
    a,b,c = (3,4),(4,5),(5,6)
    return [a,b,c]
a,b,c = getabc()
d,e,f = gettuple()
g,h,i = getlist()
```

Python Module

- Module
  - Kind of like libraries in C/C++
  - A module is a file containing Python definitions and statements.
  - The file name is the module name with the suffix `.py` appended.
  - Keyword: *import*, *from* module *import* sub_module
  - Python offers A LOT standard modules
    - `sys, os, os.path, re, string`, etc.
- Type of modules
  - Standard module: `os, sys, etc`
  - Customer built-modules
  - Extension modules, built with other languages (c/c++, etc)
    - ADaM python module

NOTE: the code is for python2.7. Will not work on python3!

```
# fibmodule.py
def fibonacci(n):
   a, b = 0, 1
   while b < n:
      a,b = b, b+a
   return (a,b)
```

```
# fib2.py
import sys,string
import fib_module

if len(sys.argv) != 2:
    print
    print "Please provide an interger!";
    print
    sys.exit(1)

print sys.argv[0]
print sys.argv[1]
num=sys.argv[1]
(a,b) = fib_module.fibonacci(string.atoi(num))
print a,b
```

Classes

- Classes defines new types of objects for OO programming
- The first argument of each method always refers to itself, conventionally name "`self`" is used
- All operations involving the attributes of an object must explicitly refer to the self variable

```
class Stack:
    def __init__(self):
        self.stack=[]
    def push(self, object):
        self.stack.append(object)
    def pop(self, object):
        return self.stack.pop()
    def length(self):
        return len(self.stack)
```

```
s = Stack()
s.push("Dave")
s.push(42)
s=s.pop()
s=s.pop()
del s # destroy s
```

```
#!/usr/bin/python
# file io.py

import sys
import ADaM

print "system input"
a=sys.stdin.readline()
print a

sys.stdout.write(a)
sys.stdout.writelines(a)
```

Python Packages

- Packages
  - A collection of modules
  - `import package.moduleName`
  - `from package import moduleName`