Data Analysis Using Python (https://www.python.org)

History of Python

- #### *Dec1989* - As a successor of ABC to provide exception handling
- #### *Feb1991* - First public release 0.9.0 had classes with inheritance, exception handling, functions and core data types
- #### *Jan1994* - Version 1.0 with functional programming
- #### *Oct2000* - Version 2.0 brings garbage collectors, Version 2.2 improves types to be fully object oriented
- #### *Dec2008* - Version 3.0 reduce feature duplication by removing old ways of doing things

Why Python ?

- #### Easy to learn
- #### Has efficient high level data structures
- #### Elegent Syntax and Dynamic typing
- #### Many third party libraries/modules
- #### Active community support

Application

- #### Web and Internet Development
- #### Scientific and Numeric
- #### Software automation and testing

Installation

- #### We will be using Anaconda distributed by CONTINUUM (https://www.continuum.io/downloads) for this training
- #### You can download and install python and jupyter - here are the instructions (https://github.com/sdonapar/python_training/blob/master/python_Installation_instructions.md)
- #### Anaconda is a completely free Python (https://www.python.org) distribution (including for commercial use and redistribution). It includes more than 400 of the most popular Python packages for science, math, engineering, and data analysis
- #### Check the python version installed on our training desktop/laptop
- #### Setup the environment variables if not set already

Python Interpreter

- The Python interpreter is usually installed as /usr/local/bin/python or /usr/bin/python
- On Windows machines, the Python installation is usually placed in C:\Python27
- If you are using Anaconda - it will be usually present in C:\anaconda2\bin

REPL - read–eval–print loop

```
>>> print("Let us start learning python")
Let us start learning python
>>>
```

Zen of Python

```
>>> import this
>>>
```

Introduction to Jupyter (http://jupyter.org) notebooks

- #### Starting jupyter notebook
- #### How to get python help
- #### Walk thru basic operations
- #### Line and Cell Magic commands (https://damontallen.github.io/IPython-quick-ref-sheets)

In [1]:

```
# My first program
print("I am learning python")
```

```
I am learning python
```

In [2]:

```
%lsmagic
```

Out[2]:

```
Available line magics:
%alias  %alias_magic  %autocall  %automagic  %autosave  %bookmark  %
cat  %cd  %clear  %colors  %config  %connect_info  %cp  %debug  %dhi
st  %dirs  %doctest_mode  %ed  %edit  %env  %gui  %hist  %history  %
killbgscripts  %ldir  %less  %lf  %lk  %ll  %load  %load_ext  %loadp
y  %logoff  %logon  %logstart  %logstate  %logstop  %ls  %lsmagic  %
lx  %macro  %magic  %man  %matplotlib  %mkdir  %more  %mv  %notebook
  %page  %pastebin  %pdb  %pdef  %pdoc  %pfile  %pinfo  %pinfo2  %po
pd  %pprint  %precision  %profile  %prun  %psearch  %psource  %pushd
  %pwd  %pycat  %pylab  %qtconsole  %quickref  %recall  %rehashx  %r
eload_ext  %rep  %rerun  %reset  %reset_selective  %rm  %rmdir  %run
  %save  %sc  %set_env  %store  %sx  %system  %tb  %time  %timeit  %
unalias  %unload_ext  %who  %who_ls  %whos  %xdel  %xmode

Available cell magics:
%%!  %%HTML  %%SVG  %%bash  %%capture  %%debug  %%file  %%html  %%ja
vascript  %%js  %%latex  %%perl  %%prun  %%pypy  %%python  %%python2
  %%python3  %%ruby  %%script  %%sh  %%svg  %%sx  %%system  %%time
 %%timeit  %%writefile

Automagic is ON, % prefix IS NOT needed for line magics.
```

In [3]:

```
%run run.py
```

```
Python is awesome!!
```

In [4]:

```
%timeit [a for a in range(0,100000)]
```

```
100 loops, best of 3: 10 ms per loop
```

How to get help ?

In [5]:

```
%quickref
```

In [6]:

```
help(sum)
```

```
Help on built-in function sum in module __builtin__:

sum(...)
    sum(sequence[, start]) -> value

    Return the sum of a sequence of numbers (NOT strings) plus the v
alue
    of parameter 'start' (which defaults to 0).  When the sequence i
s
    empty, return start.
```

In [7]:

```
sum?
```

In [8]:

```
#Python is dynamically typed language
#Dynamically typed programming languages do type checking at run-time as opposed
 to Compile-time.
iam_integer = 100
iam_float = 3.14
iam_str = "Hellow"
iam_bool = True
iam_complex = 3+4j
```

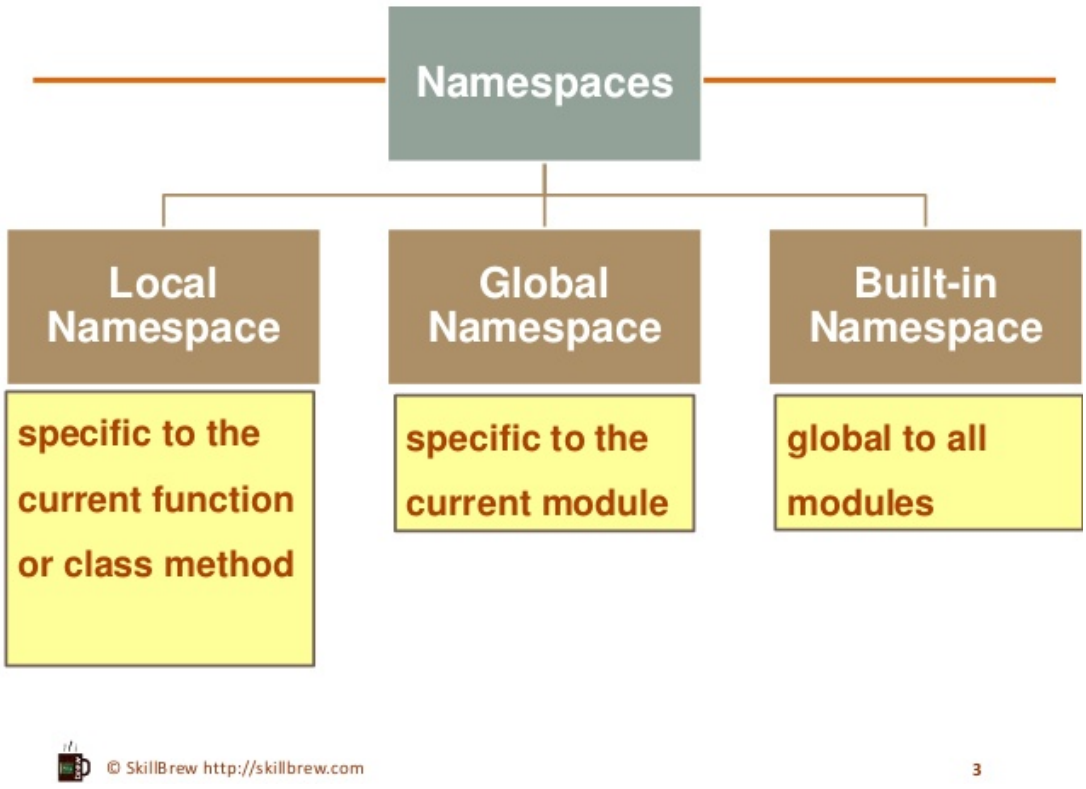
- Everything in Python is an object
- Everything in Python has a type
- **type** and **object** are special objects in python

In [9]:

```
print(type(iam_integer))
print(type(iam_float))
print(type(iam_str))
print(type(iam_bool))
print(type(iam_complex))
```

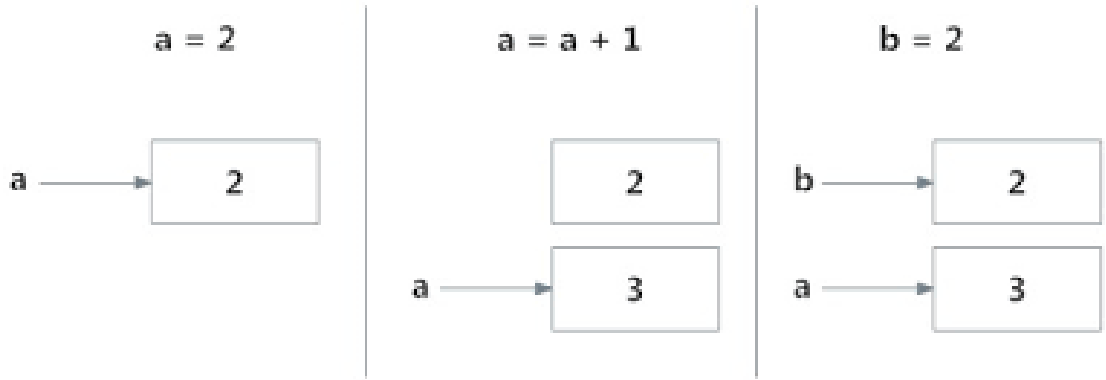
```
<type 'int'>
<type 'float'>
<type 'str'>
<type 'bool'>
<type 'complex'>
```

Intriduction to objects and namespace



In [10]:

```
a = 2
a = a +1
b = 2
```



- Every objects has an identity which is going to be unique
- **variable a** in the namespace points to object 2
- **variable a** in the namespace points moves to object 3
- new name b is created in the namespace and points to object 2

In [11]:

```
print(id(a))
print(id(b))
print(id(2))
```

```
17903944
17903968
17903968
```

In [12]:

```
# There are many other names in this namespace which are brought in by Jupyter
print(dir())
```

```
['In', 'Out', '_', '_2', '__', '___', '__builtin__', '__builtins__',
'__doc__', '__name__', '__nonzero__', '_dh', '_i', '_i1', '_i10', '_
i11', '_i12', '_i2', '_i3', '_i4', '_i5', '_i6', '_i7', '_i8', '_i
9', '_ih', '_ii', '_iii', '_oh', '_sh', 'a', 'b', 'exit', 'get_ipyth
on', 'iam_bool', 'iam_complex', 'iam_float', 'iam_integer', 'iam_st
r', 'quit']
```

In [13]:

```
# %load utilities.py
#!/usr/bin/env python

def my_dir(mylist):
    import re
    pattern = re.compile('_[0-9a-z]+')
    return [x for x in mylist if not pattern.match(x) and
            x not in ('In','Out','_','__','___','exit','quit','get_ipython')]
```

In [14]:

```
import utilities
print(utilities.my_dir(dir()))
```

```
['__builtin__', '__builtins__', '__doc__', '__name__', '__nonzero_
_', '__package__', 'a', 'b', 'iam_bool', 'iam_complex', 'iam_float',
'iam_integer', 'iam_str', 'my_dir', 'utilities']
```

Python Library Reference (https://docs.python.org/2/library/index.html)

Built-in Functions (https://docs.python.org/2/library/functions.html) - Loaded when python is started

Standard Library (https://docs.python.org/2/library/) - These are installed along with standard python installation, ex: sys, os, etc

External modules (https://pypi.python.org/pypi) - Can be downloaded from the Python Package Index, ex: numpy, pandas, etc

Numbers

In [15]:

```
2 + 2
```

Out[15]:

4

In [16]:

```
5 * 3
```

Out[16]:

```
15
```

In [17]:

```
100/21
```

Out[17]:

```
4
```

In [18]:

```
100/21.0
```

Out[18]:

```
4.761904761904762
```

In [19]:

```
import math
radius = 10 # 10 centimeters
area = math.pi * radius**2
print(area)
```

```
314.159265359
```

Strings

- Strings in python are immutable

In [20]:

```
str_a = 'This string uses single quotes'

str_b = "This string uses double quotes"

str_c = """This is a multi line string
This is second line
This is third line
"""

str_d = "This doesn't contain escape characters"
str_e = 'There are some "SPECIAL" words in this sentence'
str_f = 'It is fine to use escape character\'s some times'

# There are some special character \t, \n, etc

str_g = "Everything in Python is an object\nEvery object in Python has type\nPyt
hon is dynamically typed language"
```

In [21]:

```
print(str_g)
```

```
Everything in Python is an object
Every object in Python has type
Python is dynamically typed language
```

String methods

- Strings can be indexed
- startswith, endswith
- strip, split, replace, partition
- index,count, find
- upper, lower
- join, format
- string slicing

In [22]:

```
my_string = "Assets under administration : $5.2 trillion, including managed asse
ts : $2.1 trillion"
```

In [23]:

```
len(my_string) # returns length of string
```

Out[23]:

```
85
```

In [24]:

```
my_string.startswith("Assets") # Returns True or False
```

Out[24]:

```
True
```

In [25]:

```
my_string.endswith("Fidelity") # Returns True or False
```

Out[25]:

```
False
```

In [26]:

```
"   This is a test String ".strip() # removes the leading and trailing spaces
```

Out[26]:

```
'This is a test String'
```

In [27]:

```
"This line has return line characters at the end\n\n\n".strip("\n")
```

Out[27]:

```
'This line has return line characters at the end'
```

In [28]:

```
print(my_string.split()) # default delimiter is space
```

```
['Assets', 'under', 'administration', ':', '$5.2', 'trillion,', 'inc
luding', 'managed', 'assets', ':', '$2.1', 'trillion']
```

In [29]:

```
print(my_string.split(":")) # passing a delimiter
#What is the type of output of split ?
# What if there is no delimiter present in my_string ? would split operation fai
l ?
```

```
['Assets under administration ', ' $5.2 trillion, including managed
 assets ', ' $2.1 trillion']
```

In [30]:

```
my_string.find("$") # returns the index first occurance of character $
```

Out[30]:

```
30
```

In [31]:

```
my_string.count("trillion") # returns the number of occurances of word/character
```

Out[31]:

```
2
```

In [32]:

```
my_string.count("Fidelity") # returns 0 if the substring is not found
```

Out[32]:

```
0
```

In [33]:

```
my_string.upper() # conversts to uppercase
```

Out[33]:

```
'ASSETS UNDER ADMINISTRATION : $5.2 TRILLION, INCLUDING MANAGED ASSE
TS : $2.1 TRILLION'
```

In [34]:

```
my_string.index("trillion") # returns the starting index position of the sting
```

Out[34]:

```
35
```

In [35]:

```
# string slicing
print(my_string[0:15])  # returns the character starting from zero till 15 ( exc
luding 15)
print(my_string[10:25]) # returns the character starting from 10 till 25 ( exclu
ding 25)
print(my_string[25:])   # starting with 25 till the end of the string
print(my_string[:25])   # starting from the begining till 25 ( excluding 25)
print(my_string[:])     # complete string
```

```
Assets under ad
er administrati
on : $5.2 trillion, including managed assets : $2.1 trillion
Assets under administrati
Assets under administration : $5.2 trillion, including managed asset
s : $2.1 trillion
```

In [36]:

```
# String concatenation

my_statement = "This" + " " + "is" + " a " + "test statemet"
my_statement
```

Out[36]:

```
'This is a test statemet'
```

In [37]:

```
print("*"*3 + " Title " + "*"*3)
```

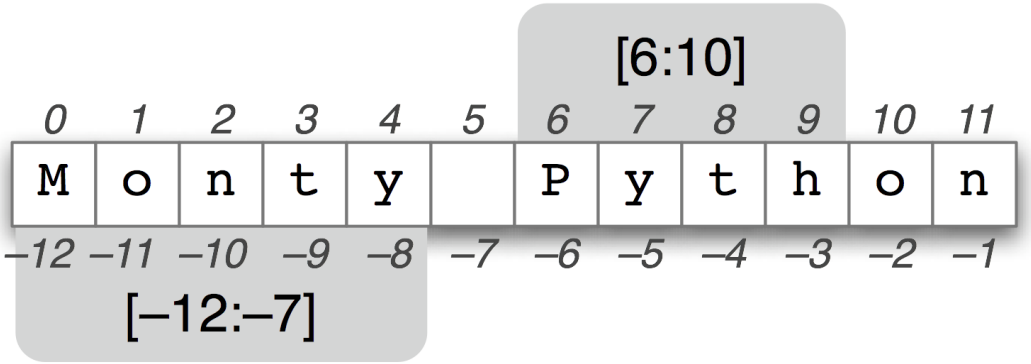
```
*** Title ***
```

In [38]:

```
# what happens if string "Title" is divided by 3 ?
# what happens if string integer 5 is added to string "Title" ?
```

Exercises

Explore string "Monty Python"



- Find the lenght of string "String in Python is an array of characters"
- How many occurance of "people" word are there in below sentence

Fidelity's goal is to make financial expertise broadly accessible and effective in helping people live the lives they want. With assets under administration of $5.2 trillion, including managed assets of $2.1 trillion as of April 30, 2015, we focus on meeting the unique needs of a diverse set of customers: helping more than 24 million people invest their own life savings, nearly 20,000 businesses manage employee benefit programs, as well as providing nearly 10,000 advisory firms with technology solutions to invest their own clients' money.

- Extract substring "assets under administration of $5.2 trillion" from above sentence using indicies
- Remove "." from the above sentence and split the sentence using "," as the delimiter

Lists

- List is the most versatile compound data type, which can be written as a list of comma-separated values (items) between square brackets
- Lists in python are mutable
- Items of list can be any python object
- List methods (https://docs.python.org/2/tutorial/datastructures.html#more-on-lists): append, extend, insert, remove, pop, index, count, sort, reverse
- in statement to check the presence of an element

In [39]:

```
# list can have different types of objects
my_list = ['Python','java',25,32,43.55,'C++']
```

In [40]:

```
len(my_list) # returns lenght of the list
```

Out[40]:

```
6
```

In [41]:

```
my_list[1] # returns second element of list
```

Out[41]:

```
'java'
```

In [42]:

```
my_list[1] = 'Java' # list are mutable
```

In [43]:

```
my_list
```

Out[43]:

```
['Python', 'Java', 25, 32, 43.55, 'C++']
```

In [44]:

```
new_list = my_list[1:4] # list slice
```

In [45]:

```
new_list
```

Out[45]:

```
['Java', 25, 32]
```

In [46]:

```
my_list.append("DotNet") # appends string at the
```

In [47]:

```
my_list
```

Out[47]:

```
['Python', 'Java', 25, 32, 43.55, 'C++', 'DotNet']
```

In [48]:

```
my_list.extend(['R','SPSS','MATLAB']) # extending a list using another list
```

In [49]:

```
my_list
```

Out[49]:

```
['Python', 'Java', 25, 32, 43.55, 'C++', 'DotNet', 'R', 'SPSS', 'MAT
LAB']
```

In [50]:

```
# list can contain duplicate items
my_list.append("Python")
```

In [51]:

```
print(my_list)
```

```
['Python', 'Java', 25, 32, 43.55, 'C++', 'DotNet', 'R', 'SPSS', 'MAT
LAB', 'Python']
```

In [52]:

```
my_list.count("Python")
```

Out[52]:

```
2
```

In [53]:

```
# this modifies the original list, sort is in place
my_list.sort()
```

In [54]:

```
print(my_list)
```

```
[25, 32, 43.55, 'C++', 'DotNet', 'Java', 'MATLAB', 'Python', 'Pytho
n', 'R', 'SPSS']
```

In [55]:

```
# Please do not run this multiple times, pop removes an element each time
last_element = my_list.pop()
last_element
```

Out[55]:

```
'SPSS'
```

In [56]:

```
my_list.index("Java")
```

Out[56]:

```
5
```

In [57]:

```
'Python' in my_list # checking if object is present inside the list
```

Out[57]:

```
True
```

In [58]:

```
# this modifies the original list, in place reverese
my_list.reverse()
```

In [59]:

```
print(my_list)
```

```
['R', 'Python', 'Python', 'MATLAB', 'Java', 'DotNet', 'C++', 43.55,
 32, 25]
```

In [60]:

```
# inserting at 1st position ( Please note the index starts at 0)
my_list.insert(1,'SPSS')
```

In [61]:

```
print(my_list)
```

```
['R', 'SPSS', 'Python', 'Python', 'MATLAB', 'Java', 'DotNet', 'C++',
43.55, 32, 25]
```

In [62]:

```
# List of Lists
list_of_lists = [['Python','C++','Java'],[2.7,4.2,8.0],['Object',2.5],'Main']
list_of_lists
```

Out[62]:

```
[['Python', 'C++', 'Java'], [2.7, 4.2, 8.0], ['Object', 2.5], 'Mai
n']
```

In [63]:

```
list_of_lists[0]
```

Out[63]:

```
['Python', 'C++', 'Java']
```

In [64]:

```
# Accessing list
for item in my_list: # iterates from the first element to last element
    print "Programming Language : ", item
```

```
Programming Language :  R
Programming Language :  SPSS
Programming Language :  Python
Programming Language :  Python
Programming Language :  MATLAB
Programming Language :  Java
Programming Language :  DotNet
Programming Language :  C++
Programming Language :  43.55
Programming Language :  32
Programming Language :  25
```

Tuples

- Tuples are very similar to Lists except that they are not mutable
- A tuple consists of a number of values separated by commas enclosed in round brackets
- Tuples can contain mutable objects like lists

In [65]:

```
my_tuple = 'Equity', # observe the comma at the end
```