Before going through this tutorial. Work out the basic python programming exercises on code academy (https://www.codecademy.com/en/tracks/python).

Python places an emphasis on readablity, simplicity and explicitiness.

Every thing is an object in python. Every number, string, data structure, class are referred to as python objects.

You can use comments to summarize a code. See the below example for comments. For printing a statement you can use 'print' command. Strings should be included in double quotes.

In [39]:
```
print "Big data examiner" #Big data examiner is a one stop place to lea
```

```
Big data examiner
```

You can return type of an object using type command. You can check whether an object is an instance of a particular type using ***isinstance*** **function.**

In [9]:
```
a= 'Big data'
print type(a)

b= 'Examiner'
print type(b)

c= 4.5

print isinstance(a, str)
print isinstance(a,int)
print isinstance(c, (int, float))
```

```
<type 'str'>
<type 'str'>
True
False
True
```

Objects and attributes of a python object can be accessed using *object.attribute_name*.

In [15]:
```
a = 'Bill gates'
a.<tab> # remove <tab> and press tab button
```

```
  File "<ipython-input-15-94d2f58585b1>", line 2
    a.<tab> # remove <tab> and press tab button
      ^
SyntaxError: invalid syntax
```

You can import a Python module (https://docs.python.org/2/tutorial/modules.html) using import command.

In [19]:
```
import numpy as np # importing numpy as np
data_new = [6, 7.5, 8, 0, 1]
data = np.array(data1) # accessing numpy as np. Here I am converting a
data
```

Out[19]:
```
array([ 6. ,  7.5,  8. ,  0. ,  1. ])
```

try these functions, these are self explanatory

In [27]:
```
x= [1,2,3,4]
y = x
z=list(x)
print x is y
print x is not z

# you can use the following operators:
# x // y -> this is called floor divide, it drops the fractional remai
# x** y -> raise x to the y the power.
# x< =y, x<y -> True if y is less than or equal to y. Same implies wit
# same applies to other logical operators such as &, |, ^, ==, !=
```

```
True
True
```

Mutable and immutable objects

Objects whose value can be changed, once they are created are called mutable objects. Objects whose value cannot be changed, once they are created are called immutable objects

In [33]:
```
# list, dict, arrays are a mutable
programming = ['Python', 'R', 'Java', 'Php']
programming[2] ='c++'
print programming

#Strings and tuples are immutable
z_tuple = (9, 10, 11, 23)
z_tuple[1] = 'twenty two' # you cant mutate a tuple
```

```
['Python', 'R', 'c++', 'Php']

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call l
<ipython-input-33-1282c7c7a358> in <module>()
      6 #Strings and tuples are immutable
      7 z_tuple = (9, 10, 11, 23)
----> 8 z_tuple[1] = 'twenty two'

TypeError: 'tuple' object does not support item assignment
```

Strings

In [10]:
```
# you can write multiline strings using triple quotes ''' or """
"""
Hi! learn Python it is fun
Data science and machine learning are amazing
"""
```

Out[10]:
```
'\nHi! learn Python it is fun \nData science and machine learning are a
```

In [43]:
```
# As I said before python strings are immutable.
x= ' This is big data examiner'
x[10] = 'f'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call l
<ipython-input-43-033ea51cd601> in <module>()
      1 # As I said before python strings are immutable.
      2 x= ' This is big data examiner'
----> 3 x[10] = 'f'

TypeError: 'str' object does not support item assignment
```

In [46]:
```
x = 'Java is a powerful programming language'
y = x.replace('Java', 'Python')
y
```

Out[46]:
```
'Python is a powerful programming language'
```

In [10]:
```
# many python objects can be converted to a string using 'str' functio
x = 56664
y = str(x)
print y
print type(y)
# strings act like other sequences, such as lists and tuples
a = 'Python'
print list(a)
print a[:3] # you can slice a python string
print a[3:]
```

```
56664
<type 'str'>
['P', 'y', 't', 'h', 'o', 'n']
Pyt
hon
```

In [18]:
```
#String concentation is very important
p = "P is the best programming language"
q = ", I have ever seen"
z = p+q
z
```

Out[18]:
```
'P is the best programming language, I have ever seen'
```

You have to do lot of string formatting while doing data analysis. You can format an argument as a string using %s, %d for an integer, %.3f for a number with 3 decimal points

In [17]:
```
print "Hii space left is just %.3f gb, and  the data base is %s"  %(0.9
print "Hii space left is just %f gb, and  the data base is %s"  %(0.987
print "Hii space left is just %d gb, and  the data base is %s"  %(0.987
```

```
Hii space left is just 0.987 gb, and  the data base is mysql
Hii space left is just 0.987000 gb, and  the data base is mysql
Hii space left is just 0 gb, and  the data base is mysql
```

Boolean and date-time

In [25]:
```
# boolean values in python are written as True and False.
print True and True
print True or False
print True and False
```

```
True
True
False
```

In [12]:
```
#Empty iterables(list, dict, strings, tuples etc) are treated as False
print bool([]), bool([1,2,3])
print bool('Hello Python!'), bool('')
bool(0), bool(1)
```

```
False True
True False
```

Out[12]:
```
(False, True)
```

In [34]:
```
x = '1729'
y = float(x)
print type(y)
print int(y)
print bool(y)
```

```
<type 'float'>
1729
True
```

In [20]:
```
#Python date and time module provides datetime, date and time types
from datetime import datetime, date, time
td = datetime(1989,6,9,5,1, 30)# do not write number 6 as 06, you will
print td.day
print td.minute
print td.date()
print td.time()
td.strftime('%m/%d/%y %H:%M:%S')#strf method converts the date and time
```

```
9
1
1989-06-09
05:01:30
```

Out[20]:
```
'06/09/89 05:01:30'
```

In [33]:
```
from datetime import datetime, date, time
datetime.strptime('1989911', '%Y%m%d') # strings can be converted to da
td = datetime(1989,6,9,5,1, 30)
td.replace(hour =0 ,minute=0, second=30)#you can replace function to ec
```

Out[33]:
```
datetime.datetime(1989, 6, 9, 0, 0, 30)
```

In [43]:
```
from datetime import datetime, date, time
td = datetime(1989,6,9,5,1, 30)
td1 = datetime(1988,8, 31, 11, 2, 23)
new_time =td1 - td # you can subtract two different date and time funct
print new_time
print type(new_time) # the type is date and time
print td +new_time
```

```
-282 days, 6:00:53
<type 'datetime.timedelta'>
1988-08-31 11:02:23
```

Handling Exceptions

Handling Exceptions is only a fancy name for *handling python errors. In Python many functions work ony on certain type of input. For example, float function returns a value error, when you feed it with a string.*

In [8]:
```
print float('7.968')
float('Big data')
```

```
7.968

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call .
<ipython-input-8-e679c5a97125> in <module>()
      1 print float('7.968')
----> 2 float('Big data')

ValueError: could not convert string to float: Big data
```

In [15]:
```
# suppose we want our float function to return the input value, we can
def return_float(x):
    try:
        return float(x)
    except:
        return x

print return_float('4.55')
print return_float('big data') # This time it didnt return a value erro
```

```
4.55
big data
```

In [13]:
```
#print float((9,8))  ->this will return a type error, remove the commen
def return_float(x):
    try:
        return float(x)
    except(TypeError, ValueError):# type error and value error are ment
        return x
print return_float((9,8))  #now you can see it returns 9,8
```

```
(9, 8)
```

In [29]:
```
# these are called ternary expressions
x = 'Life is short use python'
'This is my favourite quote' if x == 'Life is short use python'  else
```

Out[29]:
```
'This is my favourite quote'
```

Go through loops in Python(if, for and while). Refer Codeacademy

Tuples

In [7]:
```
#Tuples are one dimensional, fixed length, imutable sequence of Python
machine_learning = 77, 45, 67
print machine_learning
pythonista = (87, 56, 98), (78, 45, 33) #Nested Tuples
print pythonista
```

```
(77, 45, 67)
((87, 56, 98), (78, 45, 33))
```

In [14]:
```
#You can convert any sequence to a tuple by using 'tuple' keyword
print tuple([4,0,2])
pythonista = tuple('Python')
print pythonista
pythonista[0] # you can accessing each element in a tuple,
```

```
(4, 0, 2)
('P', 'y', 't', 'h', 'o', 'n')
```

Out[14]:
```
'P'
```

In [23]:
```
x = tuple(['Manu',[99,88], 'Jeevan'])
#x[2] = 'Prakash' # you cant modify a tuple like this
x[1].append(77)# But you can append to a object to a tuple
x
```

Out[23]:
```
('Manu', [99, 88, 77], 'Jeevan')
```

In [29]:
```
y = ('Mean', 'Median', 'Mode')+('Chisquare', 'Annova') + ('statistical
print y
('Mean', 'Median') *4  # try printing a tuple using a number
```

```
('Mean', 'Median', 'Mode', 'Chisquare', 'Annova', 'statistical signifi
```

Out[29]:
```
('Mean', 'Median', 'Mean', 'Median', 'Mean', 'Median', 'Mean', 'Median
```

In [35]:
```
deep_learning =('Theano', 'Open cv', 'Torch') # you can un pack a tuple
x,y,z= deep_learning
print x
print y
print z
```

```
Theano
Open cv
Torch
```

In [20]:
```
countries ='Usa', 'India', ('Afghanistan',' Pakistan'),
a,b,(c,d) = countries
print a
print b
print c
print d
```

```
Usa
India
Afghanistan
 Pakistan
```

In [50]:
```
countries ='Usa', 'India', ('Afghanistan',' Pakistan'), 'Usa', 'Usa'
countries.count('Usa') # .count can be used to count how many values ai
```

Out[50]:
```
3
```

Lists

I havent discussed lists, as it is covered in depth in code academy tutorials. I am going through the concepts that are not discussed in code academy. Some important list concepts are:

- *adding and removing elements from a list*
- *combining and conctenating lists*
- *sorting*
- *list slicing*

In [63]:
```
countries =['Usa', 'India','Afghanistan',' Pakistan']
y = countries.extend(['Britian', 'Canada', 'Uzbekistan', 'Turkey'])
z = countries.sort(key=len) # countries are sorted according to number
print countries
# extend can be a handy feature when your lists are large.
```

```
['Usa', 'India', 'Canada', 'Turkey', 'Britian', ' Pakistan', 'Uzbekist
```

In [83]:
```
import bisect
b = [9,9,9,9,5,6,3,5,3,2,1,4,7,8]
b.sort()
x =bisect.bisect(b,2) # bisect.bisect finds the location where an eleme
y= bisect.bisect(b, 5)
print x
print y
```

```
2
7
```

In [97]:
```
# When iterating over a sequence; to keep track of the index of the cu
languages = ['Bigdata', 'Hadoop', 'mapreduce', 'Nosql']

for i,val in enumerate(languages):
    print i,val
```

```
0 Bigdata
1 Hadoop
2 mapreduce
3 Nosql
```

In [101]:
```
#Sorted function returns a new sorted list from a sequence
print sorted([89, 99,45,63,25,53,34,56])
print sorted('Big data examiner')
```

```
[25, 34, 45, 53, 56, 63, 89, 99]
[' ', ' ', 'B', 'a', 'a', 'a', 'd', 'e', 'e', 'g', 'i', 'i', 'm', 'n',
```

In [106]:
```
hot_job = ['Big_data', 'data science', 'data scientist', 'data base de
languages = ['c', 'c++', 'java', 'python']
statistics = ['Mean', 'Median', 'Mode', 'Chi square']
print zip(hot_job, languages, statistics)

for i, (x,y) in enumerate(zip(hot_job, languages)):   #See how I use z
    print('%d: %s, %s' %(i,x,y))
```

```
[('Big_data', 'c', 'Mean'), ('data science', 'c++', 'Median'), ('data
0: Big_data, c
1: data science, c++
2: data scientist, java
3: data base developer, python
```

In [113]:
```
# you can unzip a zipped sequence as follows
rockers = [('Jame', 'Manu'), ('Govind', 'Dheepan'),('Partha', 'Reddy')]
first_names, last_names = zip(*rockers)
print first_names
print last_names
```

```
('Jame', 'Govind', 'Partha')
('Manu', 'Dheepan', 'Reddy')
```

In [114]:
```
#Use reversed keyword to reverse a sequence
list(reversed(range(20)))
```

Out[114]:
```
[19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Dictionaries

Some key concepts to remember in dictionary are:

- *How to access elements in a dictionary*
- *.keys() and .values() methods*
- *pop and del methods*

Also Go through List and dictionary comphrehensions

In [135]:
```
# you can combine two dictionaries using 'update' method
d1 = {'a' : 'octave', 'b' : 'Java'}
d1.update({'c' : 'foo', 'd' : 12})
print d1
d2 = {'a' : 'octave', 'b' : 'Java'}
d2.update({'b' : 'foo', 'c' : 12}) #the dictionary inside brackets, ove
print d2
```

```
{'a': 'octave', 'c': 'foo', 'b': 'Java', 'd': 12}
{'a': 'octave', 'c': 12, 'b': 'foo'}
```

In [137]:
```
# dict type function accepts a tuple
data_science = dict(zip(range(10), reversed(range(10)))) # see how I ar
data_science
```

Out[137]:
```
{0: 9, 1: 8, 2: 7, 3: 6, 4: 5, 5: 4, 6: 3, 7: 2, 8: 1, 9: 0}
```

In [148]:
```
# The keys of a dictionary should be immutable(int, string, float, tup
print hash('string')
print hash((1,2,3))
print hash([1,2,4]) # generates an error as lists are immutable
```

```
-1542666171
-378539185

---------------------------------------------------------------------
TypeError                                 Traceback (most recent call 
<ipython-input-148-27f144be1274> in <module>()
      2 print hash('string')
      3 print hash((1,2,3))
----> 4 print hash([1,2,4])

TypeError: unhashable type: 'list'
```

In [152]:
```
# An easy way to convert a list into a key is to convert it to a tuple
fg ={}
fg[tuple([3,4,5])] = 45
fg
```

Out[152]:
```
{(3, 4, 5): 45}
```

set

In [155]:
```
# a set is an unordered collection of unique elements.
set([3,3,4,4,4,6,7,7,7,8])
```

Out[155]:
```
{3, 4, 6, 7, 8}
```

In [166]:
```
#Sets support mathematical set operations like union, intersection, di
a = {1, 2, 3, 4, 5}
b = {3, 4, 5, 6, 7, 8}
print a|b # union
print a&b #intersection-> common elements in two dictionaries
print a-b
print a^b # symmetric difference
print {1,2,3} =={3,2,1} # if values are equal so True
```

```
set([1, 2, 3, 4, 5, 6, 7, 8])
set([3, 4, 5])
set([1, 2])
set([1, 2, 6, 7, 8])
True
```

Default dict

In [35]:
```
football_clubs = ['Manchester', 'Liverpool', 'Arsenal', 'Chelsea', 'Ma

football ={}
for clubs in football_clubs:
    club = clubs[0] # gets the first character of football_clubs
    if club not in football_clubs:
        football[club] = [clubs]
    else:
        football[club].append(clubs)
print football
```

```
{'A': ['Arsenal'], 'C': ['Chelsea'], 'B': ['Barcelona'], 'D': ['Dortmu
```

In [37]:
```
# Usually, a Python dictionary throws a KeyError if you try to get an
#The defaultdict in contrast will simply create any items that you try
#(more precisely, it's an arbitrary "callable" object, which includes

# The Same operation can be done using default dict
from collections import defaultdict # default dict is present in colle
soccer = defaultdict(list)

for clubs in football_clubs:
    soccer[clubs[0]].append(clubs)
print soccer
```

```
defaultdict(<type 'list'>, {'A': ['Arsenal'], 'C': ['Chelsea'], 'B': [
```

Functions

In [1]:
```
# a function can return multiple values
def b():
    x =34
    y =45
    z =89
    return x,y,z
```

Technically closure functions are called as dynamically-generated function returned by another function. The main property is that the returned function has access to the local variables in local namespace, where it was created. In laymans term a closure function is a function within main function.

In [14]:
```
# Example of a closure function. The function returns True, if a elemer
def dict_funct():
    new_dict = {}    # create a new dictionary
    def modifier(z):
        if z in new_dict:   # if z is in dictionary
            return True
        else:
            new_dict[z]=True
            return False
    return modifier

x = dict_funct()
list_func = [5,4,6,5,3,4,6,2,1,5]
y = [x(i) for i in list_func]
print y
```

```
[False, False, False, True, False, True, True, False, False, True]
```

Cleaning data

Raw data is messy. So you have to clean the data set, to make it ready for analysis. Here we have a list of states that consists of unnecessary punctuations,capilitalization and white space. First, I am importing a python module called regular expression (https://docs.python.org/2/library/re.html). Second, I am creating a funtion called remove_functions, to remove the unnecessary punctuations, re.sub is used to remove unnecessary punctuations in the function. Third, I am creating a list of three functions str.strip (https://www.tutorialspoint.com/python/string_strip.htm, remove_functions and [str.title](http://www.tutorialspoint.com/python/string_title.htm).

In [50]:
```
# If we are doing some data cleaning, we will be having a messy data se
import re

states = ['       Kerala', 'Gujarat!', 'Delhi', 'Telengana', 'TriPUra',


def remove_functions(strp):
    return re.sub('[!#?]', '', strp)

ooops = [str.strip, remove_functions, str.title] # create a list of fu

def clean_data(oops, funky):  # function takes two arguments
    result = []    # create a empty list
    for data in oops:    # loop over(go to each and every element) in
        for fun in funky: # loop over ooops list
            data = fun(data)  # apply each and every function in ooops
        result.append(data)  # attach formmated states data to a new l
    return result             # return the list

x = clean_data(states, ooops)
print x
```

```
['Kerala', 'Gujarat', 'Delhi', 'Telengana', 'Tripura', 'Tamil Nadu', 'T
```

In [21]:
```
# Lambda is short form of writing a function.
def f(x):
    return x**2
print f(8)
#same function using lambda
y = lambda x: x**2
print y(9)
```

```
64
81
```

Generator Expressions

In [9]:
```
def new_objjj():
    for x in xrange(100):
        yield x**2     #when using  generator functions, Use yield inst
some_variable = new_objjj()

# The above function can be written as follows
new_obj = (x**2 for x in range(100))

#Generator expressions can be used inside any Python function that wil
y = sum(x**2 for x in xrange(100))
print y

dict((i,i**2) for i in xrange(5)) #xrange is faster than range
```

```
328350
```

Out[9]:
```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

In [2]:
```
rkeys=[1,2,3]
rvals=['South','Sardinia','North']
rmap={e[0]:e[1] for e in zip(rkeys,rvals)} # use of Zip function
rmap
```

Out[2]:
```
{1: 'South', 2: 'Sardinia', 3: 'North'}
```