____ ## Pedram Jahangiry (Fall 2019)

Python Crash Course

Basic topics to be covered:

- Data types
    1. Numbers
    2. Strings
    3. Booleans
    4. Tuples
    5. Lists
    6. Dictionaries
    7. Sets
- Comparison Operators
- if, elif, else Statements
- for Loops
- while Loops
- list comprehension
- functions
- map and filter

Data types

1. Numbers

```
3+5
```

```
8
```

```
2*3
```

```
6
```

```
2 ** 3
```

```
8
```

```
5 / 2
```

```
2.5
```

```
5 % 2
```

```
1
```

Variable Assignment

```
my_var = 2
```

```
x = 2
```

```
y= my_var + x
y
```

```
4
```

2. Strings

```
Hello World
```

```
  File "<ipython-input-9-648a6a6ffffd>", line 1
    Hello World
              ^
SyntaxError: invalid syntax
```

```
"Hello World"
```

```
'Hello World'
```

```
ID = 1234
Name = "Pedram"
```

```
print('my ID number is {}, and my name is {}'.format(ID,Name))
```

```
my ID number is 1234, and my name is Pedram
```

```
my_string = 'Hellow world, I am learning #ML'
```

```
my_string.lower()
```

```
'hellow world, i am learning #ml'
```

```
my_string.upper()
```

```
'HELLOW WORLD, I AM LEARNING #ML'
```

```
my_string.split()
```

```
['Hellow', 'world,', 'I', 'am', 'learning', '#ML']
```

```
my_string.split('#')
```

```
['Hellow world, I am learning ', 'ML']
```

```
my_string.split('#')[1]
```

```
'ML'
```

```
# using in method
'x' in [1,2,3]
```

```
False
```

```
'x' in ['x','y','z']
```

```
True
```

3. Booleans

```
# unlinke R, you cannot use T or TRUE for boolean True
T
TRUE
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-21-f022eef2fd0f> in <module>
      1 # unlinke R, you cannot use T or TRUE for boolean True
----> 2 T
      3 TRUE


NameError: name 'T' is not defined
```

```
True
```

```
True
```

```
False
```

```
False
```

4. Tuples

```
# paranthesis are used for tuples
t = (1,2,3)
```

```
# extracting from tuples. Becareful! unlike R, indexing starts from 0 not 1
t[0]
```

```
1
```

```
# Note that tuples are immutable. You cannot assign some NEW variables to them. Where do we use it?
t[0] = 10
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-26-b41a8ccf82e8> in <module>
      1 # Note that tuples are immutable. You cannot assign some NEW variables to them. Where do we use
----> 2 t[0] = 10


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

```
# tuple unpacking
# read "for" loops and come back here!
x= [(1,2),(3,4),(5,6)]
```

```
x
```

```
[(1, 2), (3, 4), (5, 6)]
```

```
for a,b in x: print(a)
```

```
1
```

```
3
5
```

5. Lists

```
# Bracket [] are used to make lists. Note: [] does the same job as list() function in R.
[1,2,3]
```

```
[1, 2, 3]
```

```
my_list = ['Hello World',100,[1,"2",3]]  # notice that 1 and 3 are not coerced into string (this is dif
my_list
```

```
['Hello World', 100, [1, '2', 3]]
```

```
my_list.append('Adding variable')
```

```
my_list
```

```
['Hello World', 100, [1, '2', 3], 'Adding variable']
```

```
my_list[0]  # notice that you don't need to use [[]]. this is different than R
```

```
'Hello World'
```

```
my_list[2]
```

```
[1, '2', 3]
```

```
my_list[2][1]
```

```
'2'
```

```
my_list[2][-1]  # extracting the last element. The logic is completely different than R
```

```
3
```

```
my_list[1:3]  # starting from index 1 (the second element) and NOT including index 3 (the forth element

# NOTE: in R, my_list[1:3] will extract elements 1 through 3 (including 3rd element)
```

```
[100, [1, '2', 3]]
```

```
my_list[1:]  # extracting from index 1 (second element) to the end. There is no such a thing in R
```

```
[100, [1, '2', 3], 'Adding variable']
```

```
my_list[:2]  # extracting from index 0 upto (and not including) index 2
```

```
['Hello World', 100]
```

```
my_list[0] = 'NEW variable'  # unlike tuples, lists are mutable
```

```
my_list
```

```
['NEW variable', 100, [1, '2', 3], 'Adding variable']
```

```
# poping the last element permanantly
my_list.pop()
```

```
'Adding variable'
```

```
my_list
```

```
['NEW variable', 100, [1, '2', 3]]
```

```
my_list.pop(2)
```

```
---------------------------------------------------------------------------

IndexError                                Traceback (most recent call last)

<ipython-input-20-e023038d0fa8> in <module>
----> 1 my_list.pop(2)


IndexError: pop index out of range
```

```
my_list
```

```
['NEW variable', 100]
```

6. Dictionaries

```
# Braces and colons {  :  ,  : }  are used for dictionaries. You can use keys (and not indexes) to grab
# Dictionaries do not retian any order! very useful when you don't want to extract info based on index
my_dict = {'key1':'item1','key2':[1,2,3,4]}
```

```
my_dict
```

```
{'key1': 'item1', 'key2': [1, 2, 3, 4]}
```

```
my_dict[1]
```

```
---------------------------------------------------------------------------

KeyError                                  Traceback (most recent call last)

<ipython-input-24-b223affbf9a6> in <module>
----> 1 my_dict[1]


KeyError: 1
```

```
my_dict['key2']
```

```
[1, 2, 3, 4]
```

```
my_dict['key2'][2]
```

```
3
```

```
my_dict.keys()
```

```
dict_keys(['key1', 'key2'])
```

```
my_dict.items()
```

```
dict_items([('key1', 'item1'), ('key2', [1, 2, 3, 4])])
```

```
my_dict.values()
```

```
dict_values(['item1', [1, 2, 3, 4]])
```

7. Sets

```
# set is a collection of UNIQUE elements.
# Braces {} are used for sets. Sets
{1,2,3}
```

```
{1, 2, 3}
```

```
{1,2,3,1,2,1,2,3,3,3,3,2,2,2,1,1,2}
```

```
{1, 2, 3}
```

```
set([1,2,3,1,2,1,2,3,3,3,3,2,2,2,1,1,2])
```

```
{1, 2, 3}
```

```
my_set={1,2,3}
```

```
my_set.add(4)
my_set
```

```
{1, 2, 3, 4}
```

Comparison Operators

```
2 > 1
```

```
True
```

```
2 < 1
```

```
False
```

```
4 <= 4
```

```
True
```

```
2 == 2
```

```
True
```

```
'BUY' == 'BUY'
```

```
True
```

```
'BUY' == 'buy'
```

```
False
```

```
'buy' != 'sell'
```

```
True
```

```
(1 > 2) and (2 < 3) # all the statements must be true
```

```
False
```

```
(1 > 2) or (2 < 3)  # only one of the true elements safices!
```

```
True
```

```
(1 == 2) or (2 == 3) or (4 == 4)
```

```
True
```

if,elif, else Statements

```
# in Python: if condition: action           # in R: if (condition) {action}
if 1 < 2:
    print('condition is true')
```

```
condition is true
```

```
if 1 < 2:
    print('consition is true')
else:
    print('condition is false')
```

```
consition is true
```

```
if 1 > 2:
    print('consition is true')
else:
    print('condition is false')
```

```
condition is false
```

```
if 1 == 2:
    print('first condition is true')
elif 3 == 3:
    print('second condition is true')
else:
    print('last condition is true')
```

```
second condition is true
```

for Loops

```
# in python     for item in seq: do something.
# in R          for (item in seq){do something}
```

```
# Note: in python we don't have the luxury of using 1:5 like in R. Instead we use range() function
range(5)
```

```
range(0, 5)
```

```
list(range(5))
```

```
[0, 1, 2, 3, 4]
```

```
for item in range(5):
    print(item)
```

```
0
1
2
3
4
```

```
for item in range(4,10,2):
    print(item)
```

```
4
6
8
```

while Loops

```
# in python     while condition true: do something.
# in R          while (condition true) {do something}
```

```
count = 1
while count <= 4:
    print('count is: {}'.format(count))
    count+=1              # this is equivalent to count = count+1
```

```
count is: 1
count is: 2
count is: 3
count is: 4
```

list comprehension

This is very useful for **conditional mutation**

```
x = [1,2,3,4,5]
```

```
my_list = []  # empty list. In R we use list() to create empty list.
for item in x:
    my_list.append(item*2)
print(my_list)

# we can do all this in one line
```

```
[2, 4, 6, 8, 10]
```

```
[x*2 for x in range(1,6)]    # this is like mapply(function(x) x^2, 1:5) in R
```

```
[2, 4, 6, 8, 10]
```

functions

```
# in Python:   def my_func(inputs): do something
# in R     :   my_function <- function(inputs){do something}
```

```
def my_func(x=0):
    """
```

```
    documentation of your function
    """
    print(x+2)
```

```
my_func    # press shift tab to see the documentation
```

```
<function __main__.my_func(x=0)>
```

```
my_func() # if you don't assign default values you will get an error.
```

```
2
```

```
my_func(10)
```

```
12
```

lambda expressions (anonymous function)

```
def power(x):
    return x*2
```

```
power(2)
```

```
4
```

```
lambda x: x*2          # function(x) x*2     in R
```

```
<function __main__.<lambda>(x)>
```

map and filter

```
seq = [1,2,3]
```

```
map(power,seq)     # this is like family of apply functions in R
```

```
<map at 0x19a144a85c0>
```

```
list(map(power,seq))
```

```
[2, 4, 6]
```

```
list(map(lambda x: x*2,range(1,4)))
```

```
[2, 4, 6]
```

```
filter(lambda x: x>3,range(10))
```

```
<filter at 0x19a144051d0>
```

```
list(filter(lambda x: x>3,range(10)))
```

```
[4, 5, 6, 7, 8, 9]
```