Python Crash Course

Creating Functions

Function is block of code that is also called by its name.
The function can have different parameters or may not have any at all. If any data (parameters) are passed, they are passed explicitly.
It may or may not return any data.
Function does not deal with Class and its instance concept.

A simple Function that prints `hello there`

In [1]:
```python
def hello():
    print('hello there')
```

Now that I've created the function above I have to call it to use it

In [2]:
```python
hello()
```

```
hello there
```

Now This time lets create another function and this time it will write hello with the name of the person we pass
Since we want this function to be dynamic and adapt to any name , we will pass the name as a parameter

In [3]:
```python
def hello2(name):
    print("hello {} how are you".format(name))
```

Now lets use the above function with different names

In [4]:
```python
hello2('Taha')
```

```
hello Taha how are you
```

In [5]:
```python
hello2("Mustafa")
```

```
hello Mustafa how are you
```

In [6]:
```
hello2()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-be96a2e6ee90> in <module>
----> 1 hello2()

TypeError: hello2() missing 1 required positional argument: 'name'
```

Right now , if you don't pass a param in the above function then it will give an error , so lets put a default param so the function always have a value to display

In [7]:
```
def hello3(name='there,'):
    print("hello {} how are you".format(name))
```

In [8]:
```
hello3('Taha')
```

```
hello Taha how are you
```

In [9]:
```
hello3()
```

```
hello there, how are you
```

Now lets make a function return a value , for that we will need to use the `return` keyword below is simple function which returns the square of any number passed in

In [11]:
```
def square(x):
    return x**2
```

In [12]:
```
square(5)
```

Out[12]:
```
25
```

Now let's have a function for multiplication

In [13]:
```
def multiply(x,y):
    return x*y
```

In [14]:
```
multiply(3,4)
```

Out[14]:
```
12
```

Now lets have another Muliplication function with an optional parameter

In [15]:
```
def multiply2(x,y,const=4): #optional parameter can be left with default values
    return x*y  + const
```

```
In [16]: multiply2(3,5)
Out[16]: 19
```

You can modify the optiional param

```
In [18]: multiply2(3,5,10)
Out[18]: 25
```

```
In [19]: def multiply3(x,y,squareit=False):
            if squareit == True:
              return x**y
            else:
              return x*y
```

```
In [20]: multiply3(3,3)
Out[20]: 9
```

```
In [21]: multiply3(3,3,squareit=True)
Out[21]: 27
```

Lambda Expressions

Actually, we don't absolutely need lambda; we could get along without it. But there are certain situations where it makes writing code a bit easier, and the written code a bit cleaner. What kind of situations? … Situations in which (a) the function is fairly simple, and (b) it is going to be used only once.

Consider the following Function in which yo multiply the element by 2

```
In [22]: def times2(var):
             return var*2
```

```
In [23]: times2(10)
Out[23]: 20
```

Same thing with **Lambda**
Note: this lambda is mostly used in places where we call one liner methods

```
In [24]: times=lambda var: var*2
         times(2)
Out[24]: 4
```

lambda `arguments : expression`

when you are using lambda u are mostly doing functional programming as you are calling functions within a function

Another Example: This compare two numbers and returns the maximum of two numbers

```
In [25]: def test(a, b):
             if a > b:
                 return "a"
             else:
                 return "b"

         test(45, 100)

Out[25]: 'b'
```

Same with Lambda

```
In [26]: test = lambda a, b: "a" if a > b else "b"
         test(5, 6)

Out[26]: 'b'
```

See By using Lambda you get the same job done by fewer lines of code

Map

`Map:` Applies a function to all the items in an input_list. Most of the times we want to pass all the list elements to a function one-by-one and then collect the output. Map allows us to implement this in a much simpler and nicer way.

```
In [27]: seq = [1,2,3,4,5]
```

In Map we pass in a function and a list and the funciton is applied to each element of the list

```
In [28]: list(map(times2,seq))

Out[28]: [2, 4, 6, 8, 10]
```

Similarly you can use Map with Lambda

```
In [29]: list(map(lambda var: var*2,seq))

Out[29]: [2, 4, 6, 8, 10]
```

Filter

As the name suggests, filter creates a list of elements for which a function returns true. The filter resembles a for loop but it is a builtin function and faster.

So here we are creating a new list from the previous one , this list contains all elements of previous list which are divisible by 2

```
In [30]: list(filter(lambda item: item %2 == 0,seq))
```

```
Out[30]: [2, 4]
```

Methods

Method is called by its name, but it is associated to an object (dependent). Different python objects have their own methods Lets look at some methods of a string

```
In [31]: st = 'hello my name is Sam'
```

Converts all letters to lowercase

```
In [32]: st.lower()
```

```
Out[32]: 'hello my name is sam'
```

Converts all letters to Uppercase

```
In [33]: st.upper()
```

```
Out[33]: 'HELLO MY NAME IS SAM'
```

Split all by space

```
In [34]: st.split()
```

```
Out[34]: ['hello', 'my', 'name', 'is', 'Sam']
```

```
In [35]: tweet = 'Go Sports! #Sports'
```

Split by '#'

```
In [36]: tweet.split('#')
```

```
Out[36]: ['Go Sports! ', 'Sports']
```

Split and extract the second element of the list

In [37]:
```
tweet.split('#')[1]
```
Out[37]:
```
'Sports'
```

Assignment 5: Auto List

Create a program which lets user add any item to a list when the user types `add_item "mango"` (ofcourse instead of a mango it can be any item) and when the the user types `show_list` it shows (prints) the user all items , A user cannot enter a duplicate item, and there should be no distinction between capital letters Mango should be equal to mango. The whole program runs in a loop and at the end of every output , an input box appears for further commands. The program ends when the user types `end`

In [ ]:
```
#ADD CODE HERE
```