Mutable Imutable Lambada ¶

In [28]:

```
#7. By Val/ByRe - ArunExamples
#To Write Examples
#The By Val and By Ref become complicated when we try to use Mutable and Immutable obje
################################
```

7.a)int is Immutable when use ByVal

In [10]:

```
#7.a)int is Immutable. When we use ByVal-Value int is NOT CHANGED
##(ByValue)Passing Immutable Objects like Integer- value is not changed
```

In [11]:

```
def increment(n):
    n += 1
    print (hex(id(n)))
    print("n:",n)

    #though we are increment n+1 there is no change in the data


a = 3
print (hex(id(a)))
increment(a)
print("a:",a)

#memory address of a
print (hex(id(a)))


#Output
#a = 3   # a is still referring to the same object

```

```
0x6a2f6c80
0x6a2f6ca0
n: 4
a: 3
0x6a2f6c80
```

7.b)Int is Immutable- But when used as a ByRef-Value is CHANGED by using the return value

In [13]:

```
#7.b)Int is Immutable- But when used as a ByRef-Value is changed by using the return v
#Passing Immutable Objects- value is changed by using the return value

#Does that mean we will never be able to manipulate immutable objects by passing it to
#Turns out, we can still “modify” immutable objects by capturing the return of the fun
```

In [16]:

```
def increment(n):
    n += 1
    return n

a = 3
#memory address of a

print("a:",a)
print (hex(id(a)))
a = increment(a)  # the return value of increment() is captured!
print("a:",a)
print (hex(id(a)))
#Output#a = 4   # a now refers to the new object created by the function

```

```
a: 3
0x6a2f6c80
a: 4
0x6a2f6ca0
```

7.c) List is Mutable- when used as ByRef-Value is changed.

In [ ]:

```
#7.c) List is Mutable- when used as ByRef-Value is changed. we passing a mutable objec
#Passing Mutable Objects
```

In [25]:

```
def increment(n):
   #n.append([4])
   n.append(4)

L = [1, 2, 3]
increment(L)
print (hex(id(L)))
print('L:',L)
print (hex(id(L)))
#Output#L = [1, 2, 3, 4]   # a changed!

```

```
0xbcba47a508
L: [1, 2, 3, 4]
0xbcba47a508
```

7.d)List is Mutable- When used as ByValue- Value is NOT changed.

In [27]:

```
#7.d)List is Mutable- When used as ByValue- Value is NOT changed. we passing a mutable
#Passing Mutable Objects
#

def assign_value(n, v):
    n = v
    print('n:',n)
    print('v:',v)

L1 = [1, 2, 3]
L2 = [4, 5, 6]
assign_value(L1, L2)
print('L1:',L1)
print('L2:',L2)



#L1: [1, 2, 3]
#L2: [4, 5, 6]
```

```
n: [4, 5, 6]
v: [4, 5, 6]
L1: [1, 2, 3]
L2: [4, 5, 6]
```

lambada function

We use lambda functions when we require a nameless function for a short period of time. In Python, we generally use it as an argument to a higher-order function (a function that takes in other functions as arguments). Lambda functions are used along with built-in functions like filter() , map() etc

#Example 1lambada function

##################################

#lambda arguments: expression

#This function can have any number of arguments but only one expression, which is evaluated and returned.

#example1

#example of a normal def function

In [31]:

```
def cube(y):
    return y*y*y;
print (cube(7))
```

343

In [32]:

```
#example of a normal Lambada function
g = lambda x: x*x*x
print(g(7))
```

343

In [38]:

```
#example2 for Lambada
#A lambda function that adds 10 to the number passed in as an argument, and print the r

x = lambda a : a + 10
print(x(5))
```

15

In [39]:

```
def add(b):
    return b+10;

print (add(5))
```

15

In [ ]:

```
```

In [ ]:

```
```