**PYTHON (Everything is object)**

In this blog, we’re going to learn some basics about python ‘id’ and ‘type’, the difference between mutable and immutable objects in python and their implications, how aliases relate with mutable and immutable objects and Cpython with integer pre-allocation.

**‘id’**

‘id’ - It’s used to retrieve *identity* of and object.

It can be used to compare the identity of two objects.

Example:

a = 3

b = 3

print(id(a))

print(id(b))

print(a is b)

# Outputs ‘True’ because integers are immutable and python reuses existing immutable objects.

**‘type’**

‘type’ – It’s used to retrieve the *type* of an object.

It can be used to determine the class of an object or to create new types.

Example:

a = 5

b = “hello”

c = (1, 2, 3)

print(type(a)) #outputs ‘int’

print(type(b)) #outputs ‘str’

print(type(c)) #outputs ‘list’

**Mutable objects**

Their state and value can be changed after they’re created.

They allow adding, removing and changing.

Examples include:

*Lists;*

my\_list = (1, 2, 3)

my\_list.append(4) #adds 4 to the list

my\_list(0) = 0 #adds 0 to the list

*Dictionaries;*

my\_dict = (‘a’:1, ‘b’:2)

my\_dict(‘c’) = 3 #adds ‘c’:3 to the list

my\_dict(‘a’) = 0 #now ‘a’:1 becomes ‘a’:0

*Sets;*

my\_set = (1, 2, 3)

my\_set.add (4) #adds 4 to the list

my\_set.remove (2) #removes 2 from the list

\*When mutable objects are passed to a fuction, changes made to the object within the function affect the original object.

**Immutable objects**

They cannot be changed after they’re created.

Unlike mutable, they do not allow adding, removing and changing.

Examples include:

Strings;

my\_string = “hello”

new\_string = my\_string.replace (‘h’ , ‘j’) #new string is “jello” , “hello” is still there.

Tuples;

my\_tuple = (1, 2, 3) #modifying will result in TypeError

\*While tuples are immutable objects, they can contain mutable objects. You cannot change the tuple structure itself but you can modify the mutable objects.

Example:

my\_tuple = (1, (2,3), 4)

my\_tuple(1).append(5)

print (my\_tuple) #output- (1, (2, 3, 5), 4)

--The list inside the tuple can be modified.

\*When you pass immutable objects to a function, any modifications do not affect the original object.

**Aliases**

* For **immutable** objects, aliases to do not causes issues coz any modifications result to creation of new objects. Thus changes made through one alias do not affect the other
* For **mutable** objects, aliases can lead to unintended side effects

**NSMALLPOSINTS and NSALLNEGINTS**

By default, **Cpython** pre allocates integer objects for values in the range -5 to 256.

**\***POSINTS arethe number of small positiveintegers that are pre allocated.

\*NEGINTS are the number of small negative integers that are pre allocated.

Positive integers:

POSINTS is set to 257, pre allocating integers from 0 to 256.

Negative integers:

NEGINTS is set to -5, pre allocating integers from -1 to -5.

*Why these numbers?*

\*Integers in this range are frequently used.

\*By re-using these common integers, python reduces object creation and garbage collection.

\*Pre-allocating a larger range would also increase memory consumption.