Lists - Aliases

In this notebook we briefly discuss the difference between a memory location and the contents at that location. We also study aliases.

Everything in Python is an object. Every object has an identity (id) which corresponds to the object's location in memory. The `id()` function can be used to retrieve the memory location of an object. Variable names point to the memory location.

It is important to understand that two memory locations can contain the same values but are still two distinctly different objects. Changing or removing the contents in one will not affect the other location.

On the other hand, two different names can be used to point to the same memory location. In this case, the variable names are said to be aliases of each other.

When you have two aliases for a mutable object, changing the value of one variable means that the alias also changes. So, it is good practice to avoid aliasing when working with mutable objects.

In [1]:

```
'''
In the following example, the identifiers lst_1 and lst_2 point to two different memory locations.
Hence, although their contents are the same, their identities are different.
'''
lst_1 = [1,2,3]
lst_2 = [1,2,3]
print(lst_1 is lst_2) # The is operator will return False as the two memory locations are differen
t
print(id(lst_1), id(lst_2)) # The id() function will return two different id's for the two objects
print(lst_1 == lst_2)   # However, since the contents are the same, the `==` operator will return
`True`
```

```
False
933163518024 933163519944
True
```

In [1]:

```
'''
In the following assignment statement, the identifier lst_1 is assigned the same address as
that for lst_2.
Since both identifiers are therefore pointing to the same memory location, they are the same.
We say lst_1 and lst_2 are aliases of each other.
'''
lst_1 = [1,2,3]
lst_2 = lst_1
print(lst_1 is lst_2) #The is operator will return True as the two variables point to the same mem
ory location
print(id(lst_1), id(lst_2)) # The id() function will return the same id for the two variables.
print(lst_1 == lst_2) # The == operator will also return True
```

```
True
1453569665608 1453569665608
True
```

In [2]:

```
'''
In the above, since lst_1 and lst_2 refer to the same object, changing lst_2 changes lst_1.
Avoid aliasing when working with mutable objects.
'''
lst_1 = [1,2,3]

'''
This statement just sets the memory location of lst_2 to be the same as the location of lst_1.
Thus lst_1 and lst_2 are aliases of each other.
'''
lst_2 = lst_1
```

```
#Since both lst_1 and lst_2 are pointing to the same memory location, their id will be the same
print(id(lst_1))
print(id(lst_2))

print(lst_1)
print(lst_2)

lst_1[1] = 9 #  This statement change the second element of lst_2 to 8.  However, this means that
lst_1 is also changed.
print(lst_1)
print(lst_2)
```

```
933183381576
933183381576
[1, 2, 3]
[1, 2, 3]
[1, 9, 3]
[1, 9, 3]
```

In [3]:

```
'''
Lists can be passed as arguments to a function.  What is actually being passed is the address of t
he location
containing the list.  However, since lists are mutable, any change to the list inside the function
will permanently
modify the list. Hence care needs to be exerted when passing lists (or any other mutable objects)
as arguments.
'''
def delete_head(lst):
    del lst[0]

def squaren(n):
    n = n**2
    return n

numbers=[1,2,3,4,5]
delete_head(numbers) #After this statement is executed, the list will be changed
print('numbers:',numbers)

n = 5
print(squaren(n)) # The value of n stays the same even after this statement is executed.
print(n)
```

```
numbers: [2, 3, 4, 5]
25
5
```