Python Programming

EDUCATION FOR PEACE

(ESTD-1995)

RGM College of Engineering & Technology (Autonomous)

Department of Computer Science & Engineering

Academic Year : 2020-2021

PYTHON LANGUAGE FUNDAMENTALS

Guido Van Rossum

Learning Mantra

If you really strong in the basics, then remaining things will become so easy.

Agenda:

1. **Fundamental Data Types vs Immutability**
2. **Immutability vs Mutability**

Fundamental Data Types vs Immutability

- All Fundamental Data types are **immutable**. i.e., once we creates an object, we cannot perform any changes in that object.
- If we are trying to change the value of an object, then with those changes a new object will be created. This non-changeable behavior is called **immutability.**

Eg:

```
x = 10
print(id(x))      ➔140714560754784
x = x+1
print(id(x))      ➔140714560754816
```

Fundamental Data Types vs Immutability

Eg:

```
x = 10
y = x
print(id(x))      ➔140714560754784
print(id(y))      ➔140714560754784
y = y + 1
print(x)          ➔10
print(y)          ➔11
print(id(x))      ➔140714560754784
print(id(y))      ➔140714560754816
```

Fundamental Data Types vs Immutability : Need of Immutability

Why Immutability?

Who is responsible for creating an Object in Python?

- Python Virtual Machine (PVM) is responsible for creating an object in Python.
- In Python if a new object is required, then PVM wont create object immediately. First it will check is any object available with the required content or not.
- If available then existing object will be reused. If it is not available then only a new object will be created.
- The advantage of this approach is memory utilization and performance will be improved.

Fundamental Data Types vs Immutability : Need of Immutability

Eg:

```
a = 10
b = 10
c = 10
print(id(a)) ➔140714560754784
print(id(b)) ➔140714560754784
print(id(c)) ➔140714560754784
```

How many objects created?

Only one (i.e.,10) with three reference variables

Fundamental Data Types vs Immutability : Need of Immutability

- In the last example, we have proved that all the reference variables are pointing to single object, we are used **id() function** to compare their addresses.
- Instead of comparing the addresses, we can use a short-cut approach also. i.e., we can make use of **is operator**.

Eg:

```
a = Karthi
b = Karthi
print(a is b)      ➔True
```

Fundamental Data Types vs Immutability : Need of Immutability

- But the problem in this approach is, several references pointing to the same object, by using one reference if we are allowed to change the content in the existing object then the remaining references will be effected.

- To prevent this immutability concept is required. According to this, once creates an object we are not allowed to change content. If we are trying to change with those changes a new object will be created.

Immutability vs Mutability

So far, we have discussed about immutability, now we will discuss about mutability with a small example.

- After fundamental data types, next we need to discuss about advanced data types such as **lists, tuples, dictionaries** etc.,

Eg:

a = 10   # it represents only one value.

- If we consider list, multiple values can be represented.
- **List means group of objects.**

Immutability vs Mutability

lst = [10,20,30,40]   # Group of values represented by a list '**lst**'

How you can access these elements in the list?

By using it's index [starts from 0]

l = [10,20,30,40]

print(l[0])   ➔10

print(l[1])   ➔20

- **List is mutable**, that means, in existing object only you can perform any modifications. This changeable behavior is known as **Mutability.**

Immutability vs Mutability

Eg:

```
l = [10,20,30]
print(l)          → [10,20,30]
print(id(l))      → 2270650144264
l[0]=7777         → Now object 10 will replaced with 7777
print(l)          → [7777,20,30]
print(id(l))      → 2270650144264
```

- Here, we are getting modified content, but address is not changed. It means, all changes are being performed in existing object only, this changeable behavior is called as **Mutability**.

Immutability vs Mutability

Eg:

l1 =[10,20,30,40]

l2 = l1 ➔ Now, l1 and l2 are pointing to same object.

- If any reference variable make some changes to the object, then it affects on the all reference variables pointing to the object.

Eg:

```
l1 =[10,20,30,40]
l2 = l1
print(l1)          ➔[10,20,30,40]
print(l2)          ➔[10,20,30,40]
```

Immutability vs Mutability

Eg:

```
l1 =[10,20,30,40]
l2 = l1
print(l1)        ➔[10,20,30,40]
print(l2)        ➔[10,20,30,40]
l1[0] = 7777     ➔ This change will be reflected in l2 also
print(l1)        ➔[7777, 20, 30, 40]
print(l2)        ➔[7777, 20, 30, 40]
```

Any question?

If you try to practice programs yourself, then you will learn many things automatically

Spend few minutes and then enjoy the study

Thank You