Why are strings in Python immutable?

immutable | string | Python | communitycreator

An immutable object is one that, once created, will not change in its lifetime.

Example

Try the following code to execute:

```
name_1 = "Varun"
name_1[0] = 'T'
```

Attempt to modify the content of the string

> You will get an error message when you want to change the content of the string.

Solution

One possible solution is to create a new string object with necessary modifications:

```
name_1 = "Varun"
name_2 = "T"+ name_1[1:]
print("name_1 = ", name_1, "and name_2 = ", name_2)
```

Create new string object

To identify that they are different strings, check with the id() function:

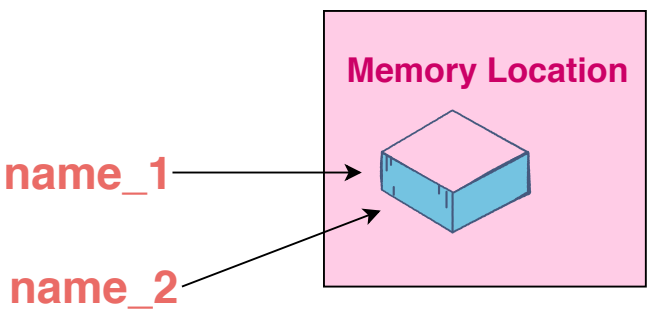
```
name_1 = "Varun"
name_2 = "T" + name_1[1:]
print("id of name_1 = ", id(name_1))
print("id of name_2 = ", id(name_2))
```

id of the two strings

To understand more about the concept of string immutability, consider the following code:

```
name_1 = "Varun"
name_2 = "Varun"
print("id of name_1 = ", id(name_1))
print("id of name_2 = ", id(name_2))
```

Multiple references to the same string object



When the above lines of code are executed, you will find that the id's of both name_1 and name_2 objects, which refer to the string "Varun", are the same.

To dig deeper, execute the following statements:

```
name_1 = "Varun"
print("id of name_1 = ", id(name_1))

name_1 = "Tarun"
print("id of name_1 afer initialing with new value = ", id(name_1))
```

Creating new string objects

As can be seen in the above example, when a string reference is reinitialized with a new value, it is creating a new object rather than overwriting the previous value.

In Python, strings are made immutable so that programmers cannot alter the contents of the object (even by mistake). This avoids unnecessary bugs.

Some other immutable objects are integer, float, tuple, and bool.

> More on mutable and immutable objects in Python.

Contributor: Suraj Aravind Bollapragada