Python - Copiar Listas

Em Python, uma variável é apenas um rótulo ou referência ao objeto na memória. Conseqüentemente, a atribuição "lst1 = lst" refere-se ao mesmo objeto de lista na memória. Dê uma olhada no exemplo a seguir -

```
lst = [10, 20]
print ("lst:", lst, "id(lst):",id(lst))
lst1 = lst
print ("lst1:", lst1, "id(lst1):",id(lst1))
```

Ele produzirá a seguinte **saída** -

```
lst: [10, 20] id(lst): 1677677188288
lst1: [10, 20] id(lst1): 1677677188288
```

Como resultado, se atualizarmos "lst", ele será refletido automaticamente em "lst1". Altere lst[0] para 100

```
lst[0]=100
print ("lst:", lst, "id(lst):",id(lst))
print ("lst1:", lst1, "id(lst1):",id(lst1))
```

Ele produzirá a seguinte **saída** -

```
lst: [100, 20] id(lst): 1677677188288
lst1: [100, 20] id(lst1): 1677677188288
```

Portanto, podemos dizer que “lst1” não é a cópia física de “lst”.

Usando o método Copy da classe List

A classe de lista do Python possui um método copy() para criar uma nova cópia física de um objeto de lista.

Sintaxe

```
lst1 = lst.copy()
```

O novo objeto de lista terá um valor id() diferente. O exemplo a seguir demonstra isso -

```
lst = [10, 20]
lst1 = lst.copy()
print ("lst:", lst, "id(lst):",id(lst))
print ("lst1:", lst1, "id(lst1):",id(lst1))
```

Ele produzirá a seguinte **saída** -

```
lst: [10, 20] id(lst): 1677678705472
lst1: [10, 20] id(lst1): 1677678706304
```

Mesmo que as duas listas tenham os mesmos dados, elas têm valores id() diferentes, portanto, são dois objetos diferentes e "lst1" é uma cópia de "lst".

Se tentarmos modificar “lst”, isso não refletirá em “lst1”. Veja o exemplo a seguir -

```
lst[0]=100
print ("lst:", lst, "id(lst):",id(lst))
print ("lst1:", lst1, "id(lst1):",id(lst1))
```

Ele produzirá a seguinte **saída** -

```
lst: [100, 20] id(lst): 1677678705472
lst1: [10, 20] id(lst1): 1677678706304
```