anti_pattern1

October 8, 2019

[1]:
```
# Anti-pattern
def func(a, x=[]):
    x.append(a)
    print(x, id(x))
```

Here, the default argument is empty list (mutable object).
NOTE: The mutable objects have global scope
This function works perfect when the default argument is passed.

[2]:
```
func(3, [])
```

```
[3] 84332040
```

[3]:
```
func('b', [3])
```

```
[3, 'b'] 79122376
```

Problem occurs when the default argument is not passed. For first time when we call, it creates an empty list object.

[5]:
```
func(1)
```

```
[1] 79354312
```

In subsequent new calls, instead of create new list object. It will use the existing created new list. This is the leakage problem

[6]:
```
func('a')   # leakage occurs here
```

```
[1, 'a'] 79354312
```

[7]:
```
func(2)       # leakage occurs here
```

```
[1, 'a', 2] 79354312
```

0.0.1 How to address this anti-pattern

[8]:
```
def func(a, x=None):
    if not x:
        x = []
    x.append(a)
    print(x, id(x))
```

[9]:
```
func(1)
```

```
[1] 79362376
```

[10]:
```
func('a')
```

```
['a'] 79361736
```

[11]:
```
func(2)
```

```
[2] 79361672
```

As seen here, all three functions calls gave results independently.