Common Gotchas

Hephzibah Pon Cellat Arulprakash
@chennaipy

Table of contents

Mutable Default Arguments

Code

```
def append_to(element, to=[]):
    to.append(element)
    return to

my_list = append_to(12)
print(my_list)

my_other_list = append_to(42)
print(my_other_list)
```

Mutable Default Arguments

What You Might Have Expected to Happen

```
my_list = append_to(12)
print(my_list)

my_other_list = append_to(42)
print(my_other_list)
```

A new list is created each time the function is called if a second argument isn't provided, so that the output is:

```
[12]
[42]
```

Mutable Default Arguments

What actually happens

```
[12]
[12, 42]
```

A new list is created once when the function is defined, and the same list is used in each successive call.

Mutable Default Arguments

What actually happens

```
[12]
[12, 42]
```

Python’s default arguments are evaluated once when the function is defined, not each time the function is called (like it is in say, Ruby). This means that if you use a mutable default argument and mutate it, you will and have mutated that object for all future calls to the function as well.

Mutable Default Arguments

What you should do instead

```
def append_to(element, to=None):
    if to is None:
        to = []
    to.append(element)
    return to
```

Create a new object each time the function is called, by using a default arg to signal that no argument was provided (None is often a good choice).

Late Binding Closures

Code

```
def create_multipliers():
    return [lambda x : i * x for i in range(5)]

for multiplier in create_multipliers():
    print(multiplier(2))
```

Late Binding Closures

What You Might Have Expected to Happen

- A list containing five functions
- That each have their own closed-over i variable that multiplies their argument, producing:

```
0
2
4
6
8
```

Late Binding Closures

What actually happens

```
8
8
8
8
8
```

- Five functions are created;
- instead all of them just multiply x by 4

Late Binding Closures

What you should do instead

you can create a closure that binds immediately to its arguments by using a default arg like so:

```
def create_multipliers():
    return [lambda x, i=i : i * x for i in range(5)]
```

List copy

Code

```
array1 = [1, 2, 3, 4, 5]

array2 = array1
array2[0] = 10

print(array1)
print(array2)
```

List copy

What You Might Have Expected to Happen

```
[1, 2, 3, 4, 5]
[10, 2, 3, 4, 5]
```

- Array1 has values 1, 2, 3, 4, 5
- Array2 is created with same values as like array1
- Array2 first element is modified to 10
- Array1 values are not changed

List copy

What actually happens

```
[10, 2, 3, 4, 5]
[10, 2, 3, 4, 5]
```

- Variables are simply names that refer to objects.
- Doing array2=array1 doesn’t create a copy of the list
- It creates a new variable array2 that refers to the same object array1 refers to.
- This means that there is only one object (the list), and both array1 and array2 refer to it.
- Lists are mutable, which means that you can change their content.

List copy

What you should do instead

```
array2 = array1.copy()

or

array2 = array1[:]

or

array2 = list(array1)
```

Local Variable

Code

```
x = 10

def foo():
       print(x)


foo()
```

```
x = 10

def foo():
       x += 1
       print(x)


foo()
```

Local Variable

What You Might Have Expected to Happen

```
10
```

```
11
```

- First code snippet prints x value 10
- Second code snippet should be x = x+1 and print 11

Local Variable

What actually happens

```
10
```

```
UnboundLocalError: local variable
'x' referenced before assignment
```

- when you make an assignment to a variable in a scope, that variable becomes local to that scope and shadows any similarly named variable in the outer scope.
- Since the first statement in foo assigns a new value to x, the compiler recognizes it as a local variable.
- Consequently when the next statement print(x) attempts to print the uninitialized local variable and an error results.

Local Variable

What you should do instead

```
x = 10

def foo():
       global x
       x += 1
       print(x)


foo()
```

References

- https://docs.python-guide.org/writing/gotchas/
- https://docs.python.org/3/faq/programming.html#why-did-changing-list-y-also-change-list-x
- https://stackoverflow.com/questions/2612802/how-do-i-clone-a-list-so-that-it-doesnt-change-unexpectedly-after-assignment
- https://docs.python.org/3/faq/programming.html#why-am-i-getting-an-unboundlocalerror-when-the-variable-has-a-value
- https://realpython.com/python-scope-legb-rule/