PASS BY ASSIGNMENT

1. Pass by assignment

00:00 - 00:12

The way that Python passes information to functions is different from many other languages. It is referred to as "pass by assignment", which I will explain in this lesson.

2. A surprising example

00:12 - 01:10

Let's say we have a function foo() that takes a list and sets the first value of the list to 99. Then we set "my_list" to the value [1, 2, 3] and pass it to foo(). What do you expect the value of "my_list" to be after calling foo()? If you said "[99, 2, 3]", then you are right. Lists in Python are mutable objects, meaning that they can be changed. Now let's say we have another function bar() that takes an argument and adds ninety to it. Then we assign the value 3 to the variable "my_var" and call bar() with "my_var" as the argument. What do you expect the value of "my_var" to be after we've called bar()? If you said "3", you're right. In Python, integers are immutable, meaning they can't be changed.

3. Digging deeper

01:10 - 01:18

Let's look at another example to understand what's going on. Imagine that this gray bar is your computer's memory.

4. Digging deeper

01:18 - 01:29

When we set the variable "a" equal to the list [1, 2, 3], the Python interpreter says, "Okay, now 'a' points to this location in memory."

5. Digging deeper

01:29 - 01:39

Then if we type "b = a", the interpreter says, "Okay, now 'b' points to whatever 'a' is pointing to."

6. Digging deeper

01:39 - 01:47

So if we were to append 4 to the end of "a", both variables get it because there is only one list.

7. Digging deeper

01:47 - 01:53

Likewise, if we append 5 to "b", both variables get it.

8. Digging deeper

01:53 - 02:06

However, if we assign "a" to a different object in memory, that does not change where "b" is pointing. Now, things that happen to "a" are no longer happening to "b", and vice versa.

9. Pass by assignment

02:06 - 02:11

How does this relate to the example functions we saw earlier?

10. Pass by assignment

02:11 - 02:17

When we assign a list to the variable "my_list", it sets up a location in memory for it.

11. Pass by assignment

02:17 - 02:24

Then, when we pass "my_list" to the function foo(), the parameter "x" gets assigned to that same location.

12. Pass by assignment

02:24 - 02:32

So when the function modifies the thing that "x" points to, it is also modifying the thing that "my_list" points to.

13. Pass by assignment

02:32 - 02:39

In the other example, we created a variable "my_var" and assigned it the value 3.

14. Pass by assignment

02:39 - 02:46

Then we passed it to the function bar(), which caused the argument "x" to point to the same place "my_var" is pointing.

15. Pass by assignment

02:46 - 03:02

But the bar() function assigns "x" to a new value, so the "my_var" variable isn't touched. In fact, there is no way in Python to have changed "x" or "my_var" directly, because integers are immutable variables.

16. Immutable or Mutable?

03:02 - 03:17

There are only a few immutable data types in Python because almost everything is represented as an object. The only way to tell if something is mutable is to see if there is a function or method that will change the object without assigning it to a new variable.

17. Mutable default arguments are dangerous!

03:17 - 03:54

Finally, here is a thing that can get you into trouble. foo() is a function that appends the value 1 to the end of a list. But, whoever wrote this function gave the argument an empty list as a default value. When we call foo() the first time, we get what you would expect, a list with one entry. But, when we call foo() again, the default value has already been modified! If you really want a mutable variable as a default value, consider defaulting to None and setting the argument in the function.

18. Let's practice!

03:54 - 03:57

You can check your understanding with the following exercises.