Theory: Objects in Python

14 minutes 5 / 7 problems solved

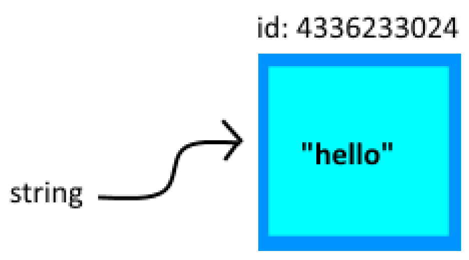
Start practicing

2441 users solved this topic. Latest completion was about 2 hours ago.

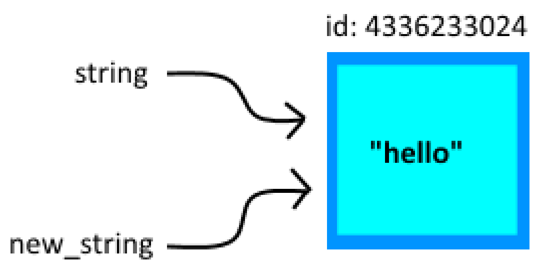
Knowing how different types of objects work in Python will help you understand some of the following topics more deeply, as well as the structure of the language in general.

§1. Reference to an object

In Python, all values are stored in objects. You can think that an object is like a box that contains information about some value and also stores some additional data such as its identity. When you assign a value to a variable, e.g. `string = "hello"`, Python creates a new object, places this value (the string `"hello"` in our case) inside the new object and then creates a **reference** from the variable name `string` to the object.



Then, if we assign one variable to another, e.g. `new_string = string`, Python will create a reference from the new variable `new_string` to the **same object**.



You can use the `id` function to see that both variables refer to the same object:

```
print(id(string))        # 4336233024
print(id(new_string))    # 4336233024
```

As a result, you can access the object using any of the two variable names. You can also assign a new value to one of these variables and this will not affect the other one.

```
string = "hello"
new_string = string
string = "world"

print(string, id(string))            # world 4336233136
print(new_string, id(new_string))    # hello 4336233024
```

Note that the identity has changed along with the value.

Current topic:

✓ Objects in Python

Topic depends on:

✓ Immutability

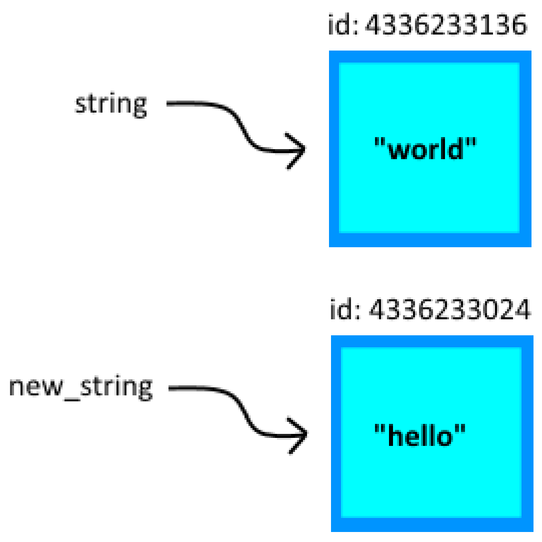
✓ Identity testing

✓ Else statement Stage 1 15

Topic is required for:

✓ Default arguments

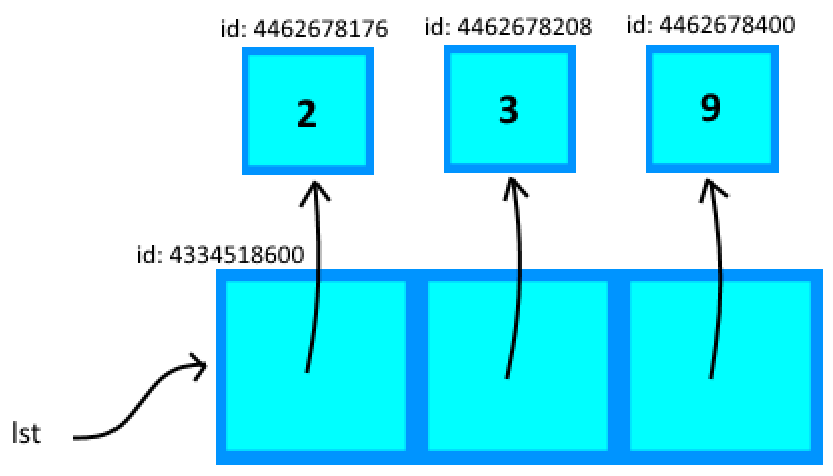
Copy of an object



However, the situation is a bit more complex when we deal with *mutable* objects, e.g. some of the containers.

§2. Mutable objects & references

Let's take a list as an example. The thing is, a list doesn't store its values inside itself. Instead, it stores **references** to objects that store values. For example, when you write `lst = [2, 3, 9]`, Python creates the following structure:



Now, if we assign our list to a new variable and then try to alter the first object, this will also affect the new list:

```
lst = [2, 3, 9]
new_lst = lst

print(lst, id(lst))            # [2, 3, 9] 4334518600
print(new_lst, id(new_lst))    # [2, 3, 9] 4334518600

# we change an element of the first list
lst[2] = 0
print(lst, id(lst))            # [2, 3, 0] 4334518600

# but the new list is also modified
print(new_lst, id(new_lst))    # [2, 3, 0] 4334518600
```

This is so because both lists refer to the same object: when it is modified, all variables continue to refer to this very object. In the next topic, you will learn how to alter a list without changing its copies.

§3. Conclusions

- Variables in Python do not store values themselves, they store **references** to objects that store values.
- When we assign one variable to another, they refer to the **same object**.
- After modifying mutable objects, other variables referring to it will also **change**.

Report a typo

Table of contents:

208 users liked this theory. 3 didn't like it. What about you?

😍 🙂 😐 🙁 😡

Start practicing

Comments (7) Hints (0) Useful links (0) Show discussion