Theory: Identity testing

22 minutes 5 / 13 problems solved

Start practicing

§1. Many copies of equal values

By now, you know how to work with values in Python. For example, you know how to perform arithmetic operations with numbers. But *what* is a value in Python? It can't be an abstract thing, like in math, because a computer should be able to work with it. In this topic, you will get some understanding of values in Python.

Equal values in Python can exist in many copies. Consider the following code:

```
a = [1, 2, 3]
b = [1, 2, 3]
c = a

print(a)  # [1, 2, 3]
print(b)  # [1, 2, 3]
print(c)  # [1, 2, 3]
```

It looks like all these variables are the same. But they aren't in some sense. To see it let's modify the list `a`.

```
a = [1, 2, 3]
b = [1, 2, 3]
c = a

a[0] = 5

print(a)  # [5, 2, 3] - changed
print(b)  # [1, 2, 3] - didn't change
print(c)  # [5, 2, 3] - also changed
```

The reason is that we created **two copies** of `[1, 2, 3]`. Variables `a` and `c` refer to the first copy, and `b` refers to the second copy. Changing one of them doesn't affect the other one.

We call these copies as **objects**. An object is stored in memory and contains information about an abstract value it represents. So there can be several objects that represent the same value. You can treat such objects as twins. At first glance, they look identical, but actually, they are different people.

Let's see how to distinguish twins in Python.

§2. Id function

Each object in Python has an associated integer called **identity**. You can get this integer by passing the object to the function `id`. Numbers, strings and other primitive types are also objects and they have an identity too. Identity never changes during the life of the object. Different objects in memory have different identities.

Using it we can distinguish two objects in Python that contain the same value. It is similar to distinguishing twins by looking at their identity documents.

```
a = [1, 2, 3]
b = [1, 2, 3]

print(a == b)  # True, they contain the same value

# But they have different identities
print(id(a))  # 4558721352
print(id(b))  # 4560238984
```

Current topic:

- ✓ Identity testing

Topic depends on:

- ✓ Comparisons (Stage 1) 16★
- ✓ Indexes (Stage 3) 7★
- ✓ Declaring a function (Stage 1) 10★

Topic is required for:

- ✓ Objects in Python
  - Float special values
  - Regexps in Python

Table of contents:

Python generates `id` on the fly, running the pieces of code above will give you other integer values. Moreover, new objects can have smaller ids than the objects created earlier.

But if two variables refer to the same object, then the `id` function will return the same value.

```
a = [1, 2, 3]
c = a

print(a == c)  # True, they contain the same value

# And they also have the same identity
print(id(a))  # 4558721352
print(id(c))  # 4558721352
```

As you can see, the variables `a` and `c` share the identity, which means they refer to the same object.

§3. Identity testing

You can check if two variables refer to the same object by comparing the results of the `id` function. But there is a better way to do it. Python has an **identity operator** `is` that checks if two objects have **the same identity**. The result is a boolean value: `True` or `False`. You should not confuse it with the **equality operator** `==` which tests whether two objects contain **the same value**.

```
a = [1, 2, 3]
b = [1, 2, 3]


identity_test = a is b  # False, because a and b refer to different objects in mem
ory
equality_test = a == b  # True, because a and b contain the same value

b = a


identity_test = a is b  # True, because now both variables refer to the same objec
t
```

The `is not` operator is the **negation** of the `is` operator. It returns `True` if its operands refer to different objects.

```
a = [1, 2, 3]
b = [4, 5]

print(a is not b)  # True, as expected
```

§4. Use the identity operator carefully

Using the identity operator instead of the equality operator might lead to lots of mistakes. The example below shows the danger of the `is` operator.

```
a = int(input())  # 10
b = int(input())  # 10
print(a is b)     # True
print(id(a))      # 4462719392
print(id(b))      # 4462719392

a = int(input())  # 10000
b = int(input())  # 10000
print(a is b)     # False

print(id(a))      # 4466218032

print(id(b))      # 4466218160
```

The reason for such weird behavior is that Python optimizes the use of small integers. They are commonly used, so Python doesn't create a new object every time, but gives a reference to an existing number. The same thing happens to short strings.

However, the case of large numbers depends on the implementation. You may get `True` for the following expression:

```
a = 10000
b = 10000
print(a is b)  # True or False depending on your system
```

§5. When to use the identity operator

The proper case to use the `is` operator is to test if something is `None`. `None` is a special keyword in Python that is used to define *no value*.

It is safe to use `is` in this case, because `None` is a **singleton**. This means that `None` object is created only once and then used whenever you refer to `None` in your code.

It is common to use `None` as a default value for optional arguments in functions.

```
def say_hello(name=None):
    if name is None:
        print('Hello!')
    else:
        print(f'Hello, {name}!')


say_hello()        # 'Hello!'
say_hello('Nick')  # 'Hello, Nick!'
```

`True` and `False` are also singletons, so you can use `is` with them too.

§6. Conclusion

In this topic, we've learned a little about objects in Python and how to test objects for identity. In order not to make mistakes in your code pay attention to the following points:

- There can be many objects containing the same value. They are **equal** but not **identical**.
- The identity operator does not compare values, but it checks if its operands refer to the same object.
- Don't use the identity operator with primitive types.
- Use the identity operator to test if something is `None`.

Report a typo

252 users liked this theory. **2** didn't like it. **What about you?**

Start practicing

Comments (12) Hints (0) Useful links (0) Show discussion