Python TUTORIAL

Python assertIs()

Summary: in this tutorial, you'll learn how to use the Python `assertIs()` to test if two objects are the same.

Introduction to Python assertIs() method

The `assertIs()` allows you to test if two objects are the same. The following shows the syntax of the `assertIs()` method:

```
assertIs(first, second, msg=None)
```

If the `first` and `second` reference the same object, the test will pass. Otherwise, it'll fail.

The `msg` is optional. It's displayed in the test result in case the test fails.

Technically, the `assertIs()` method uses the `is` operator:

```
first is second
```

Python assertIs() method example

First, create a `Logger` singleton class in the `logger.py` module:

```
from datetime import datetime



class Logger:
    _instance = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super(Logger, cls).__new__(cls)
        return cls._instance

    def log(self, message):
        print(f'{datetime.now().strftime("%Y-%m-%d %H:%M:%S")} {message}')
```

The singleton is a design pattern that limits the instantiation of a class to a single instance. In other words, you'll have the same `Logger` object regardless of how many times you call the `Logger()` constructor.

Second, create a `TestLogger` class that tests the `Logger` class:

```
import unittest

from logger import Logger



class TestLogger(unittest.TestCase):
    def setUp(self):
        self.logger = Logger()

    def test_singleton(self):
        new_logger = Logger()
        self.assertIs(self.logger, new_logger)
```

In the `TestLogger` class:

- First, create a new instance of the `Logger` class in the `setUp()` method and assign it to the `self.logger` instance variable.
- Second, create a new instance of the Logger class and use the `assertIs()` method to check if two instances are the same.

If you run the test:

```
python -m unittest -v
```

you'll get the following output:

```
test_singleton (test_logger.TestLogger) ... ok


----------------------------------------------------------------------
Ran 1 test in 0.000s


OK
```

The test passed.

Python assertIsNot() method

The `assertIsNot()` tests if the first object is not the same as the second one:

```
assertIsNot(first, second, msg=None)
```

For example:

```
import unittest



class TestInteger(unittest.TestCase):
    def test_integer_different_value(self):
        x = 10
        y = 20
        self.assertIsNot(x, y)

    def test_integer_same_value(self):
        x = 10
        y = 10
        self.assertIs(x, y)
```

Run the test:

```
python -m unittest -v
```

Output:

```
test_integer_different_value (test_integer.TestInteger) ... ok
test_integer_same_value (test_integer.TestInteger) ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
```

In this example, we use the `assertIsNot()` method to test if two integer variables reference different objects. Since their values are different, they reference different objects.

In the second test case, we use the `assertIs()` method to test if two integer variables reference the same object. Because their values are the same, they reference the same object.

Summary

- Use the `assertIs()` method to test if two objects are the same.
- Use the `assertIsNot()` method to test if two variables reference different objects.