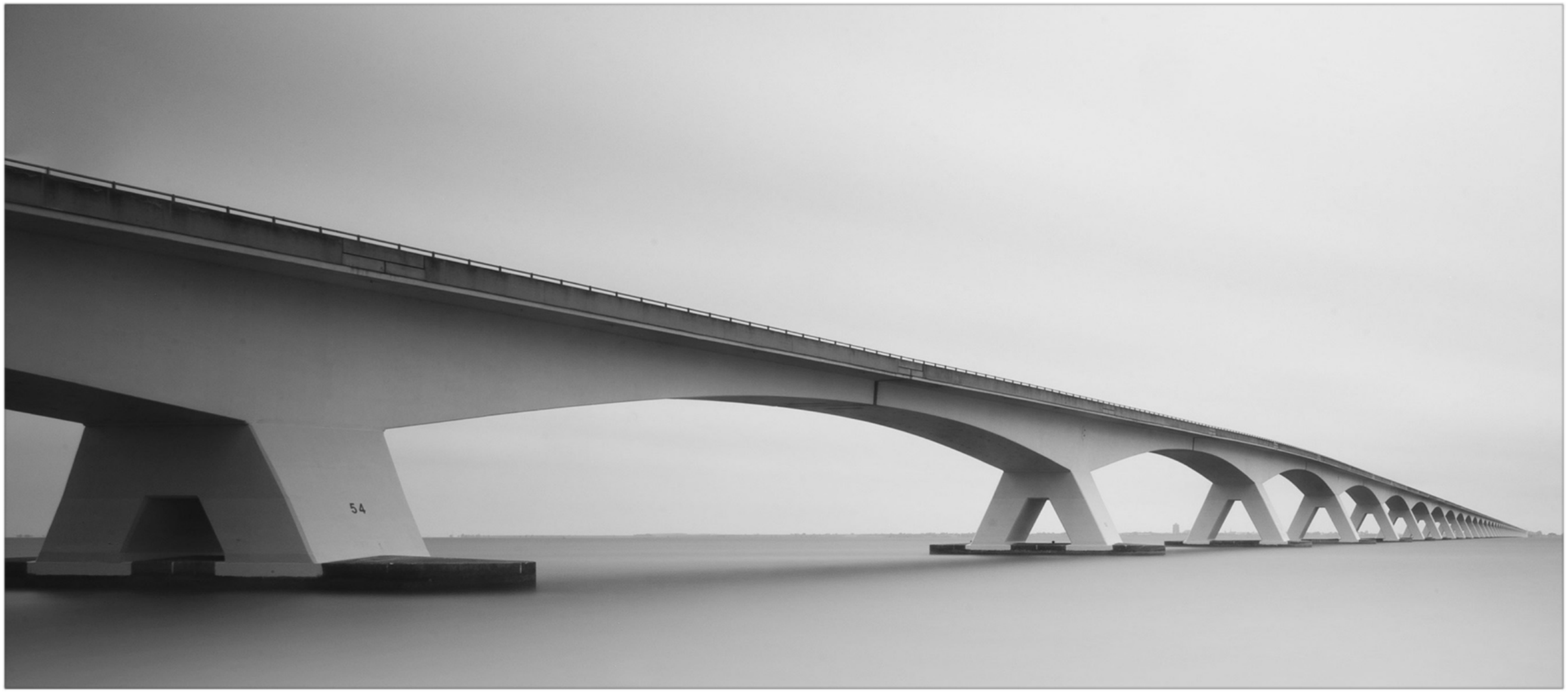


2019年1月25日

python , unittest , mock

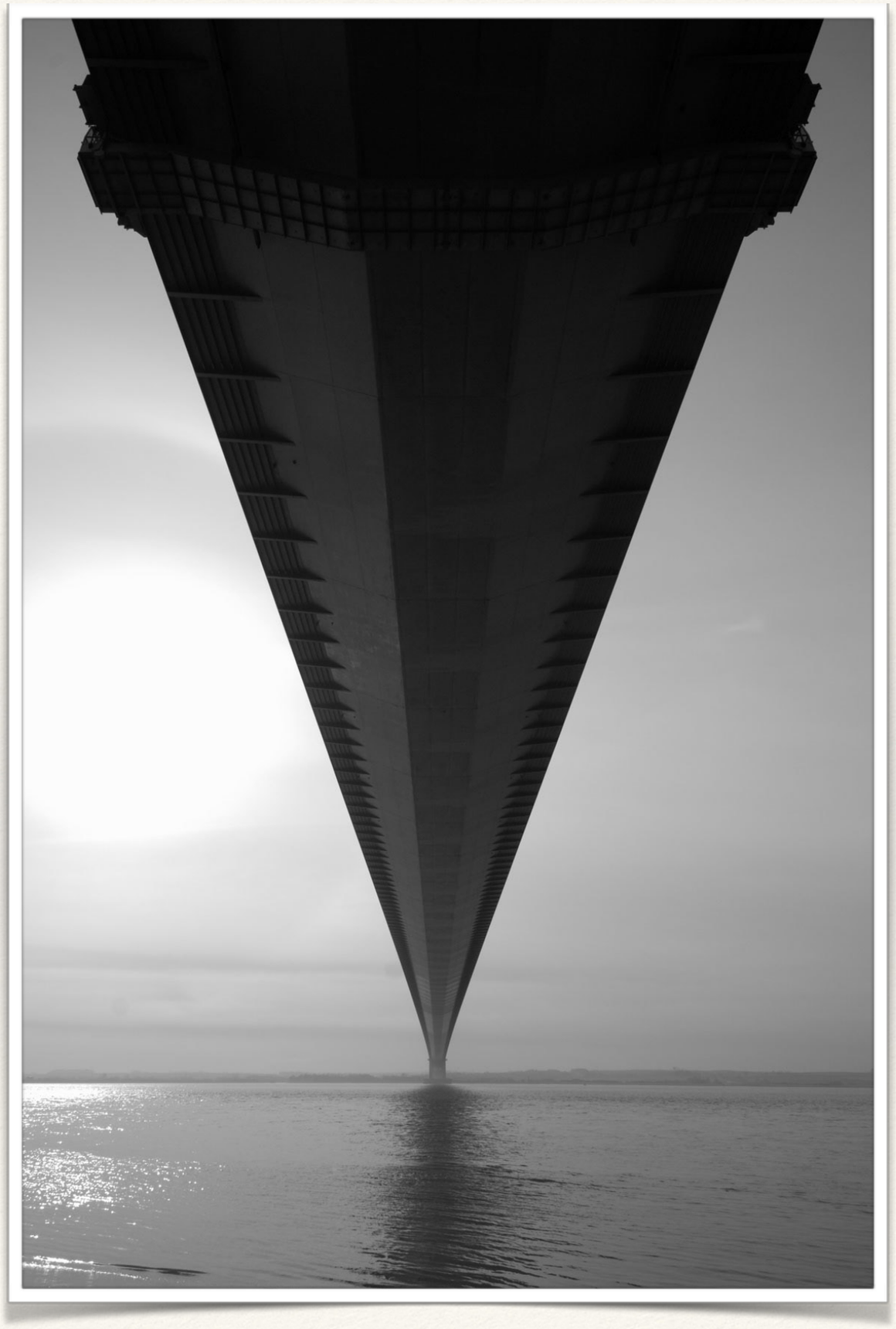
单元测试

单元测试

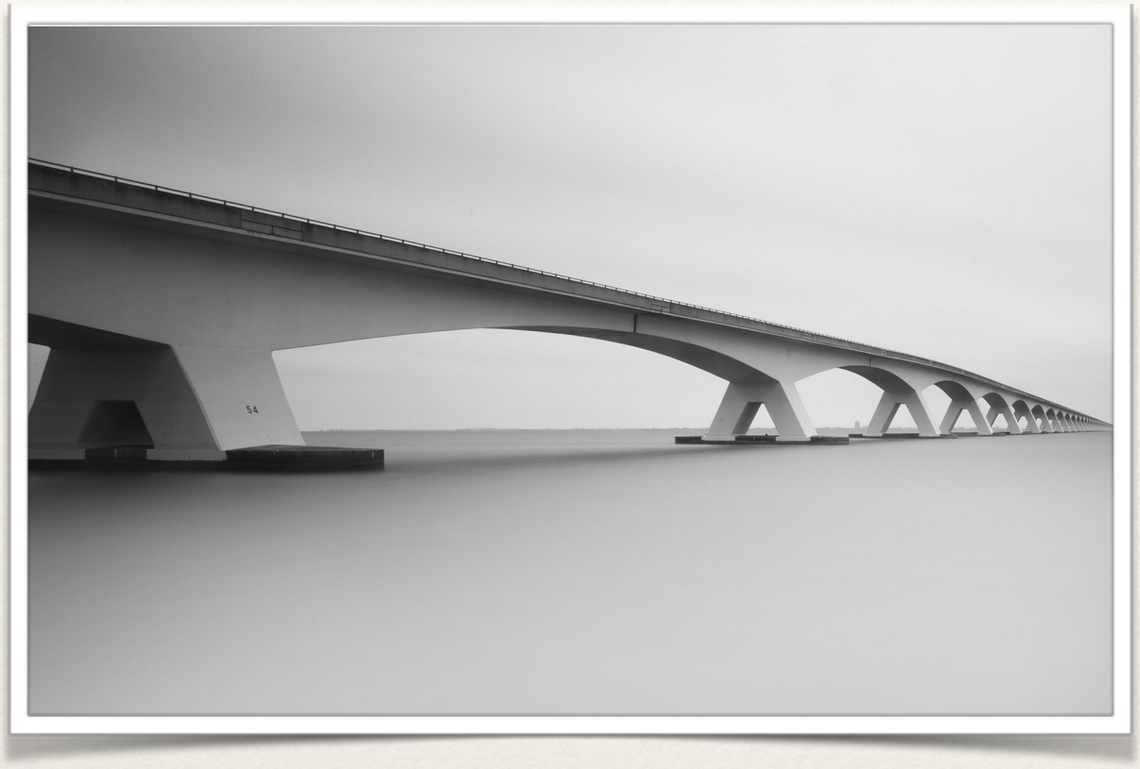
基本说明

* 我们需要编写一个测试类，从unittest.TestCase继承。

* 对每一类测试都需要编写一个test_xxx()方法。由于unittest.TestCase提供了很多内置的条件判断，我们只需要调用这些方法就可以断言输出是否是我们所期望的。



Method	Checks that
`assertEqual(a, b)`	`a == b`
`assertNotEqual(a, b)`	`a != b`
`assertTrue(x)`	`bool(x) is True`
`assertFalse(x)`	`bool(x) is False`
`assertIs(a, b)`	`a is b`
`assertIsNot(a, b)`	`a is not b`
`assertIsNone(x)`	`x is None`
`assertIsNotNone(x)`	`x is not None`
`assertIn(a, b)`	`a in b`
`assertNotIn(a, b)`	`a not in b`
`assertIsInstance(a, b)`	`isinstance(a, b)`
`assertNotIsInstance(a, b)`	`not isinstance(a, b)`



* 最简单的运行方式是在my_test.py最后加上两行代码:

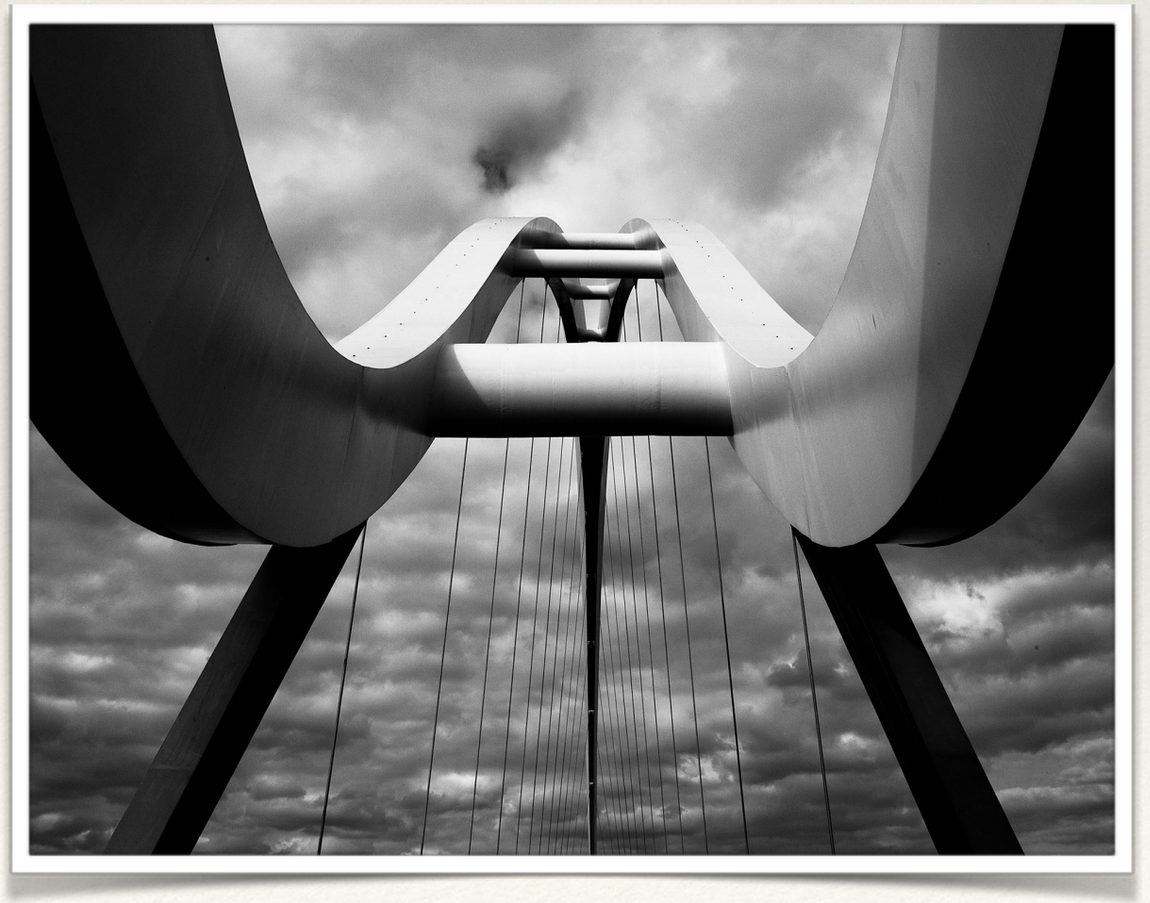
```
if __name__ == "__main__":
        unittest.main()
```

运行脚本:

```
python my_test.py
```

运行单元测试

* 最常用的方法是在命令行通过参数运行:
  - python -m unittest my_test1 my_test2
  - python -m unittest my_test.TestClass.test_method



Unittest

unittest中最核心的四部分是：TestCase，TestSuite，TestRunner，TestFixture

unittest的流程：写好TestCase，然后由TestLoader加载TestCase到TestSuite，然后由TextTestRunner来运行TestSuite，运行的结果保存在TextTestResult中。

我们通过命令行或者unittest.main()执行时，main会调用TextTestRunner中的run来执行，或者我们可以直接通过TextTestRunner来执行用例。

TestCase

1、一个TestCase的实例就是一个测试用例，每个测试方法均以test开头，否则不能被unittest识别

2、在第一行给出了每一个用例执行的结果的标识，成功是.，失败是F，出错是E，跳过是S

3、在uniitest.main()中加verbosity参数可以控制输出的错误报告的详细程度，默认是1，如果设为0， 则不输出每一用例的执行结果，即没有上面的结果中的第1行，如果设为2，则输出详细的执行结果

4、测试的执行跟方法的顺序没有关系

5、命令行参数：-v显示详情 -f遇到失败停止运行 -c等待测试结束后显示所有结果

TestSuite,TestRunner

- 多个测试用例集合在一起，就是TestSuite。我们添加到TestSuite中的case是会按照添加的顺序执行的。
- TestRunner 类作为测试用例的基本执行环境,来驱动整个单元测试过程，单元测试时一般不直接使用 TestRunner 类，而是使用其子类 TextTestRunner 来完成测试，并将测试结果以文本方式显示出来。

```
suite = unites.TestSuite()

Tests = [TestClass.funa, TestClass.funb, TestClass.func]

suite.addTests(Tests)

runner = TextTestRunner()

runner.run(suite)
```

TestLoader

- TestLoader是用来加载TestCase到TestSuite中的。

- loader = TestLoader()

  tests = loader.loadTestsFromNames(test_list)

  tests = loader.loadTestsFromName("module.Class.test_func"))

  tests = loader.loadTestsFromTestCase(TestClass)

  tests = loader.loadTestsFromModule(Module)

  suite.addTests(tests)

- tests = defaultTestLoader.discover(start_dir, pattern='test*.py')

  TextTestRunner().run(tests)

TestResulte

- 测试的结果会保存到TextTestResult实例中，包括运行了多少测试用例，成功了多少，失败了多少等信息。

```
# 输出到文件
with open("ttt.txt", "a") as f:
    runner = TextTestRunner(stream=f, verbosity=2)
    runner.run(suite)


# 输出到html
dir = os.getcwd() + "/report"
testRunner = HTMLTestRunner(output=dir, stream=f, verbosity=2)
testRunner.run(suite)
```

TestFixture

setUp用来为测试准备环境，tearDown用来清理环境。

对一个测试用例环境的搭建和销毁，是一个fixture。

- setUp()tearDown()：在每个测试方法（用例）运行时被调用一次
- setUpClass()与tearDownClass()：全程只调用一次
- setUpModule()与tearDownModule()：整个文件级别只调用一次

setUp()&tearDown() < setUpClass()&tearDownClass() < setUpModule()&tearDownModule()

skip

- @unittest.skip(reason)

  @unittest.skipIf(condition, reason)

  @unittest.skipUnless(condition, reason)

  @unittest.expectedFailure

- Class TestMethod(unittest.TestCase):

  ```
  def test_func(self):
      self.skipTest(reason)
      self.assertEqual(func(*args), 1)
  ```

mock的安装和导入

* 在Python 3.3以前的版本中，需要另外安装mock模块，可以使用pip命令来安装：

```
$ sudo pip install mock
```

* 然后在代码中就可以直接import进来：

```
import mock
```

* 从Python 3.3开始，mock模块已经被合并到标准库中，被命名为unittest.mock，可以直接import进来使用：

```
from unittest import mock
```

* Mock对象是mock模块中最重要的概念。Mock对象就是mock模块中的一个类的实例，这个类的实例可以用来替换其他的Python对象，来达到模拟的效果。Mock类的定义如下：

```
class Mock(spec=None, side_effect=None, return_value=DEFAULT, wraps=None, name=None, spec_set=None, **kwargs)
```

* name: 这个是用来命名一个mock对象，只是起到标识作用，当你print一个mock对象的时候，可以看到它的name。

* return_value: 这个字段可以指定一个值（或者对象），当mock对象被调用时，如果side_effect函数返回的是DEFAULT，则对mock对象的调用会返回return_value指定的值。

* side_effect: 这个参数指向一个可调用对象，一般就是函数。当mock对象被调用时，如果该函数返回值不是DEFAULT时，那么以该函数的返回值作为mock对象调用的返回值。

Mock VS MagicMock

- MagicMock is a subclass of Mock with default implementations of most of the magic methods. You can use MagicMock without having to configure the magic methods yourself.

- Stack overflow

```
a = MagicMock()
a                                # <MagicMock id=`4348189888’>
a.return_value = "hello world”
a()                               # ”hello world”

# Attributes and the return value of a `MagicMock` will also be `MagicMocks`.
a.func                           # <MagicMock name=`a.func' id=`4348234272’>
a[“x”]                           # <MagicMock name=`mock.__getitem__()’ id='4351398128'>

# 定义属性
a.attr1 = 3
a.attr1                          #3
```

side_effect

- side_effect给mock分配了可替换的结果，覆盖了return_value, 将其设置为None可以清除。

```
mock = MagicMock(return_value=3, side_effect=5)
mock()    #5
```

```
mock = MagicMock(side_effect = [1,2,3])
mock(), mock(), mock()      #(1,2,3)
```

```
def func(*args):
    return args
mock = MagicMock()
mock.side_effect = func
mock("a")          #("a",)
```

Test dependencies

```
from unittest.mock import patch


@patch("module.Class.method")          —"str"
def test(mock_method):
    mock_method.return_value = 3
    module.Class.method()          #3


with patch("module.Class.method") as mock_method:
    a = MyClass()
    a.method()              # "hello world"
    mock_method()       # "hello world"
```

patch.dict

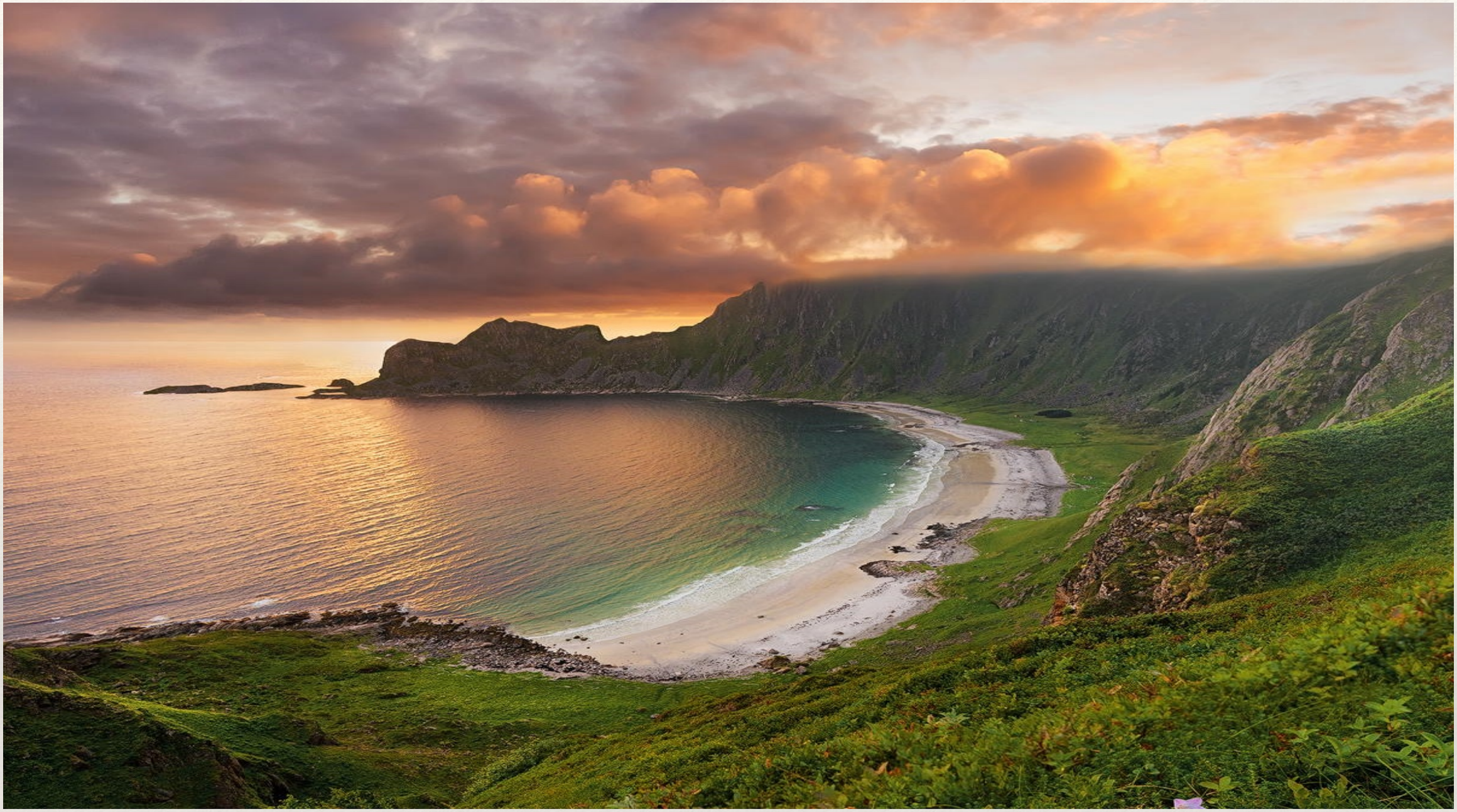
```
foo = {"x":1, "y":2}


with patch.dict(foo, {"z":3}, clear=False):
    print(foo)          #{"x":1, "y":2, "z":3}

print(foo)              #{"x":1, "y":2}
```

ensure the same api

- 确保mock对象和被替换的对象拥有相同的attributes/methods

```
from mock import create_autospec

def func(a, b, c):
    pass

mock = create_autospec(func, return_value='fishy')

mock('wrong arguments')    # TypeError:'b' parameter lacking default value


@patch("__main__.func", autospec=True)
```

dir	directions
assert_called()	Assert the mock was called at least once.
assert_called_once()	Assert the mock was called exactly once.
assert_called_with(*args, **kwargs)	Only pass if the call is the most recent one.
reset_mock()	Restore the mock object to its initial state.
called	A boolean representing whether or not the mock object has been called.
call_count	An integer telling you how many times the mock object has been called.
mock_calls	Records all calls to the mock object, its methods, magic methods and return value mocks.
configure_mock(**kw)	Set attributes on the mock through keyword arguments.



Thanks!