نوشتن تست‌های واحد (Unit Testing) در پایتون

تست واحد یا **Unit Testing** فرآیند نوشتن و اجرای کدهایی است که برای بررسی صحت عملکرد واحدهای مختلف برنامه (مانند توابع، کلاس‌ها، و متدها) طراحی شده‌اند. هدف از این کار اطمینان از این است که کد به درستی عمل می‌کند و رفتار مورد انتظار را در شرایط مختلف ایجاد می‌کند. تست‌های واحد ابزار بسیار مهمی برای افزایش قابلیت نگهداری، شفافیت، و کیفیت کد هستند.

---

1. معرفی مفهوم تست واحد و اهمیت آن در برنامه‌نویسی

تست واحد به‌طور خاص روی قسمت‌های کوچکی از برنامه تمرکز می‌کند تا صحت عملکرد آنها را تأیید کند. این تست‌ها معمولاً توسط برنامه‌نویسان به‌طور خودکار نوشته می‌شوند و می‌توانند به شما در شناسایی مشکلات و اشکالات کد کمک کنند.

مزایای استفاده از تست واحد:

- **کاهش خطاها**: با اجرای تست‌های واحد به‌طور مداوم، می‌توانید سریعاً مشکلات را شناسایی و رفع کنید.
- **افزایش قابلیت نگهداری**: تست‌های واحد باعث می‌شوند که کدهای شما به راحتی قابل تغییر و گسترش باشند.
- **مستند سازی عملکرد کد**: تست‌ها به‌عنوان مستنداتی از نحوه عملکرد کد عمل می‌کنند و می‌توانند به‌طور غیرمستقیم کمک به درک بهتر کد کنند.
- **تست خودکار**: می‌توانید تست‌ها را به‌طور خودکار اجرا کنید و از انجام آن به‌صورت دستی صرفه‌جویی کنید.

---

2. استفاده از `unittest.TestCase` برای تعریف کلاس‌های تست

در پایتون، ما می‌توانیم با استفاده از ماژول `unittest` تست‌های واحد بنویسیم. `unittest` یکی از کتابخانه‌های استاندارد پایتون است که ابزارهای لازم برای نوشتن، سازمان‌دهی، و اجرای تست‌ها را فراهم می‌کند.

برای شروع، باید یک کلاس تست تعریف کنیم که از کلاس `unittest.TestCase` ارث‌بری می‌کند.

مثال:

```
import unittest

class MyTestClass(unittest.TestCase):
    def test_addition(self):
        self.assertEqual(1 + 1, 2)  # بررسی جمع دو عدد
```

---

3. نوشتن متدهای تست با استفاده از `assertEqual()`, `assertTrue()`, `assertFalse()`, `assertRaises()` و دیگر متدهای بررسی

در `unittest`، متدهای مختلفی برای بررسی شرایط مختلف در داخل تست‌ها وجود دارد. در اینجا به تعدادی از مهم‌ترین متدها اشاره می‌کنیم:

- `assertEqual(a, b)`: بررسی می‌کند که `a` برابر با `b` است.
- `assertTrue(x)`: بررسی می‌کند که `x` درست است (True).

- `assertFalse(x)`: بررسی می‌کند که `x` غلط است (False).
- `assertRaises(exception, callable)`: بررسی می‌کند که در هنگام اجرای یک تابع، یک استثنا خاص رخ دهد.
- `assertIn(item, container)`: بررسی می‌کند که `item` در `container` وجود دارد.
- `assertNotIn(item, container)`: بررسی می‌کند که `item` در `container` وجود ندارد.

مثال‌ها:

```
import unittest

class TestMathOperations(unittest.TestCase):
    def test_addition(self):
        self.assertEqual(1 + 2, 3)

    def test_is_true(self):
        self.assertTrue(1 < 2)

    def test_is_false(self):
        self.assertFalse(1 > 2)

    def test_raises_exception(self):
        with self.assertRaises(ValueError):
            raise ValueError("An error occurred")

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

---

4. ساختار فایل‌های تست و تقسیم‌بندی آنها

تست‌ها معمولاً در فایل‌های جداگانه نگهداری می‌شوند و می‌توانند در پوشه‌های خاصی سازمان‌دهی شوند. یک ساختار متداول برای پروژه‌ها به‌صورت زیر است:

```
project/
│
├── my_module.py          # کد اصلی برنامه
└── tests/                # پوشه‌ای برای فایل‌های تست
    ├── __init__.py
    ├── test_my_module.py  # فایل تست برای my_module.py
    └── test_helpers.py    # فایل تست برای helper functions
```

در این ساختار:

- فایل‌های تست در پوشه `/tests` قرار دارند.
- هر فایل تست باید با `_test` شروع شود تا به راحتی توسط `unittest` شناسایی شود.
- بهتر است که هر فایل تست معادل بخش خاصی از کد باشد و نام آن متناسب با نام ماژول یا کلاس تست‌شده باشد.

---

5. اجرای تست‌ها با استفاده از `unittest.main()` و بررسی نتایج تست‌ها

برای اجرای تست‌ها و مشاهده نتایج، می‌توانیم از `unittest.main()` استفاده کنیم که تمام تست‌ها را اجرا کرده و نتایج را نمایش می‌دهد.

مثال اجرای تست‌ها:

```
import unittest

class TestMathOperations(unittest.TestCase):
    def test_addition(self):
        self.assertEqual(1 + 1, 2)

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

برای اجرای این تست‌ها، کافی است که فایل را به‌طور مستقیم اجرا کنید. `unittest.main()` تست‌ها را شناسایی کرده و آنها را اجرا می‌کند. پس از اجرای تست‌ها، نتایج به‌صورت گزارشی در کنسول نمایش داده می‌شود.

نمونه خروجی:

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
```

اگر همه تست‌ها موفق باشند، پیام `OK` ظاهر می‌شود. در غیر این صورت، جزئیات خطاها و شکست‌ها نمایش داده می‌شود.

نتیجه‌گیری

- **تست واحد** برای اعتبارسنجی صحت عملکرد بخش‌های مختلف برنامه و جلوگیری از بروز خطاهای غیرمنتظره اهمیت دارد.
- با استفاده از ماژول `unittest` می‌توانیم به‌طور مؤثر و ساختاریافته تست‌های واحد را برای برنامه‌های پایتون بنویسیم.
- متدهای مختلف بررسی در `unittest` به ما این امکان را می‌دهند که شرایط مختلف را بررسی و تست کنیم.
- تست‌ها را می‌توان در فایل‌های جداگانه نگهداری کرده و به‌صورت خودکار با `unittest.main()` اجرا نمود.