نوشتن و اجرای تست‌ها با `pytest`

`pytest` یک ابزار تست‌نویسی محبوب و قدرتمند در پایتون است که امکانات زیادی برای ساده‌سازی نوشتن تست‌ها فراهم می‌کند. در این بخش، به بررسی نحوه سازمان‌دهی تست‌ها، نوشتن تست‌های پارامتریک، استفاده از fixtures و اجرای تست‌ها خواهیم پرداخت.

---

1. ساختار و سازمان‌دهی تست‌ها با `pytest`

برای استفاده بهینه از `pytest`، توصیه می‌شود که فایل‌های تست خود را در پوشه‌ای جداگانه و معمولاً با نام `tests` قرار دهید. همچنین، نام فایل‌های تست باید با `_test` شروع شود تا `pytest` بتواند به‌طور خودکار آن‌ها را شناسایی کند.

ساختار پروژه

```
my_project/
│
├── main.py          # کد اصلی برنامه
├── tests/           # پوشه تست‌ها
│   ├── test_math.py    # فایل تست
│   └── test_utils.py   # فایل تست
└── requirements.txt    # فایل وابستگی‌ها
```

---

2. نوشتن تست‌های پارامتریک با استفاده از `pytest.mark.parametrize@`

گاهی اوقات می‌خواهید یک تست را برای مقادیر مختلف بررسی کنید. در این حالت، می‌توانید از `pytest.mark.parametrize@` استفاده کنید تا همان تست را برای مجموعه‌ای از داده‌ها به‌صورت پارامتریک اجرا کنید.

نمونه تست پارامتریک

```python
import pytest

# تعریف تست با پارامترهای مختلف
@pytest.mark.parametrize("a, b, expected", [
    (2, 3, 5),
    (4, 5, 9),
    (7, 8, 15)
])
def test_addition(a, b, expected):
    result = a + b
    assert result == expected
```

در اینجا:

- `pytest.mark.parametrize@` به‌طور خودکار این تست را برای ترکیب‌های مختلف از `a`, `b` و `expected` اجرا می‌کند.
- تست‌ها به‌صورت پارامتریک اجرا خواهند شد و برای هر ترکیب یک بار اجرا می‌شوند.

---

3. استفاده از `pytest fixtures` برای آماده‌سازی داده‌ها و منابع مورد نیاز قبل از اجرای تست‌ها

`pytest fixtures` یک روش عالی برای راه‌اندازی منابع یا داده‌های مورد نیاز برای تست‌ها است. با استفاده از fixtures می‌توانیم منابع مختلفی مانند پایگاه داده، فایل‌ها یا اشیاء پیچیده را قبل از اجرای هر تست فراهم کنیم.

تعریف و استفاده از Fixtures

```
import pytest

# تعریف یک fixture برای فراهم کردن داده
@pytest.fixture
def setup_data():
    data = {"name": "John", "age": 30}
    return data

# استفاده از fixture در تست
def test_name(setup_data):
    assert setup_data["name"] == "John"
```

در اینجا:

- `setup_data` یک fixture است که داده‌های خاصی را فراهم می‌کند.
- این fixture به‌طور خودکار در تست‌هایی که آن را به‌عنوان پارامتر دریافت کرده‌اند استفاده می‌شود.

---

4. اجرای تست‌ها با `pytest` و نمایش نتایج

برای اجرای تست‌ها با `pytest` کافی است دستور زیر را در ترمینال وارد کنید:

```
pytest
```

این دستور تمام فایل‌های تست را که نام‌شان با `_test` شروع می‌شود، شناسایی کرده و اجرا می‌کند. در انتها، `pytest` گزارشی از نتایج تست‌ها به شما ارائه می‌دهد.

اجرای تست‌ها با گزینه‌های اضافی

- `v-` : برای نمایش گزارش‌های دقیق‌تر و اطلاعات بیشتر درباره هر تست.

```
pytest -v
```

- `maxfail=1--` : برای توقف اجرای تست‌ها پس از اولین خطا.

```
pytest --maxfail=1
```

---

5. نحوه بررسی خروجی تست‌ها با استفاده از `pytest` و گزینه‌های مختلف آن

`pytest` از گزارش‌دهی‌های مختلف پشتیبانی می‌کند که می‌توانید از آن‌ها برای بررسی نتایج تست‌ها استفاده کنید.

- **خروجی ساده**: اگر فقط دستور `pytest` را اجرا کنید، گزارشی کوتاه از وضعیت تست‌ها خواهید دید.
- **خروجی دقیق (verbose)**: برای دریافت جزئیات بیشتر از هر تست می‌توانید از گزینه `v-` استفاده کنید.

```
pytest -v
```

- **توقف پس از اولین خطا**: با استفاده از گزینه `maxfail--` می‌توانید تعداد حداکثر خطاها را تنظیم کنید تا پس از آن اجرای تست‌ها متوقف شود.

```
pytest --maxfail=1
```

- **تست کردن یک فایل یا تابع خاص**: اگر بخواهید فقط یک فایل یا تابع خاص را تست کنید، می‌توانید از نام آن استفاده کنید.

```
pytest tests/test_math.py  # اجرای فقط فایل خاص
pytest tests/test_math.py::test_addition  # اجرای فقط یک تابع خاص
```

---

6. استفاده از `pytest` برای تست‌های غیرهمزمان (async) با `pytest-asyncio`

`pytest-asyncio` یک افزونه برای `pytest` است که امکان انجام تست‌های غیرهمزمان (asynchronous) با استفاده از `asyncio` را فراهم می‌کند.

نصب `pytest-asyncio`

برای استفاده از `pytest-asyncio`، ابتدا باید آن را نصب کنید:

```
pip install pytest-asyncio
```

نوشتن و اجرای تست‌های غیرهمزمان

```
import pytest
import asyncio

@pytest.mark.asyncio
async def test_async_addition():
    await asyncio.sleep(1)
    assert 2 + 3 == 5
```

در اینجا:

- `@pytest.mark.asyncio` به `pytest` می‌گوید که این یک تابع غیرهمزمان است که باید با `asyncio` اجرا شود.
- از `await` برای فراخوانی توابع غیرهمزمان استفاده می‌کنیم.

نتیجه‌گیری

`pytest` یکی از بهترین ابزارها برای نوشتن و اجرای تست‌ها در پایتون است. با استفاده از قابلیت‌هایی مانند تست‌های پارامتریک، fixtures، و پشتیبانی از تست‌های غیرهمزمان می‌توانید کدهای خود را به‌طور مؤثر تست کنید و مشکلات احتمالی را شناسایی کنید. این ابزار علاوه بر سادگی، انعطاف‌پذیری زیادی برای تست‌های پیچیده نیز دارد.