Spickzettel: Python – `pytest` vs. `unittest`

Ziel

Unterschiede, Vor- und Nachteile der beiden verbreitetsten Testframeworks in Python verstehen.

`unittest` (Standardbibliothek)

- In Python enthalten (`import unittest`)
- Klassenbasiert, inspiriert von JUnit
- Wenig Abhängigkeiten, guter Einstieg

Beispiel

```
import unittest

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

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

Vorteile

- Keine externen Pakete nötig
- Kompatibel mit CI/CD-Systemen

Nachteile

- Mehr Boilerplate
- Weniger elegant für einfache Tests

`pytest` (externes Paket)

- Erweiterbares, modernes Framework
- Unterstützt Funktionen, Fixtures, Plugins, Markierungen

Beispiel

```
def test_addition():
    assert 1 + 1 == 2
```

Vorteile

- Kürzerer, lesbarer Code
- Fixtures und Plugins (z. B. `pytest-cov`, `pytest-mock`, `pytest-django`)
- Flexibel & für große Test-Suiten geeignet

Nachteile

- Muss separat installiert werden (`pip install pytest`)
- Zu viele Plugins können Setup komplex machen

Test-Ausführung

```
# unittest
python -m unittest

# pytest
pytest
```

Empfehlung

Projektgröße	Empfehlung
Klein/Standard	`unittest` reicht
Mittel/Groß	`pytest` bevorzugt
Frameworks (z. B. Django)	meist `pytest` + Plugins

Für moderne Projekte ist `pytest` oft die erste Wahl – aber `unittest` bleibt relevant für Kompatibilität & Standardisierung.