Spickzettel: Python – `unittest` Framework

Ziel

Automatisierte Unit-Tests mit der Python-Standardbibliothek schreiben – ohne zusätzliche Pakete.

Grundlagen

- Bestandteil der Standardbibliothek
- Klassenbasiert (`unittest.TestCase`)
- Erkennt Methoden, die mit `test_` beginnen

Struktur eines Tests

```
import unittest

class TestMath(unittest.TestCase):
    def test_addition(self):
        self.assertEqual(2 + 2, 4)

    def test_division(self):
        self.assertAlmostEqual(10 / 4, 2.5)

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

Wichtige Assertions

Methode	Bedeutung
`assertEqual(a, b)`	a == b
`assertNotEqual(a, b)`	a != b
`assertTrue(x)`	bool(x) is True
`assertFalse(x)`	bool(x) is False
`assertIsNone(x)`	x is None
`assertRaises(error)`	Fehler wird ausgelöst

Testausführung

```
python -m unittest               # alle Tests entdecken
python -m unittest test_modul.py # gezielt Testdatei ausführen
```

Best Practices

- Pro Modul eine Testdatei (`test_*.py`)
- Testklasse = Modul oder Komponente
- Klare, sprechende Methodennamen (`test_division_by_zero`)
- Setup mit `setUp()`/`tearDown()` bei Bedarf nutzen

`unittest` ist perfekt für einfache und portable Testumgebungen – stabil, standardisiert und CI-tauglich.