Spickzettel: Unit Testing mit `unittest` & GitHub Actions

Voraussetzungen

Projektstruktur z. B.:

```
mein_projekt/
├── src/
│   └── mein_modul/
│       └── core.py
├── tests/
│   └── test_core.py
├── test_runner.py
└── .github/workflows/test.yml
```

`test_runner.py` zum manuellen Testen & für CI/CD

```
import sys
import os
import unittest

# Pfade einrichten
sys.path.insert(0, os.path.abspath(os.path.dirname(__file__)))
sys.path.insert(0, os.path.join(os.path.abspath(os.path.dirname(__file__)),
'src'))

# Tests entdecken
tests_dir = os.path.join(os.path.dirname(__file__), "tests")
test_loader = unittest.TestLoader()
test_suite = test_loader.discover(start_dir=tests_dir, pattern="test*.py")

# Testlauf starten
runner = unittest.TextTestRunner(verbosity=2)
result = runner.run(test_suite)

# Fehler melden
sys.exit(not result.wasSuccessful())
```

GitHub Actions Workflow (`.github/workflows/test.yml`)

```
name: Python Unit Tests

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout Code
```

```
    uses: actions/checkout@v3

  - name: Python einrichten
    uses: actions/setup-python@v4
    with:
      python-version: '3.10'

  - name: Abhängigkeiten installieren
    run: |
      pip install -r requirements.txt

  - name: Tests ausführen
    run: python test_runner.py
```

Best Practices

- Tests strikt in `tests/` organisieren
- Pfade über `sys.path` sauber vorbereiten
- `test_runner.py` als Einstiegspunkt für alle lokalen & CI-Testläufe
- Klarer Exitcode für GitHub Actions
- Optional: Testabdeckung mit `coverage.py`, Erweiterung für `pytest` möglich