Statische Codeanalyse in Python

IT-Sicherheit

ITT-Net-IS

20. März 2025

1 Einleitung

Die Code-Qualität spielt eine entscheidende Rolle bei der Entwicklung von robusten und wartbaren Anwendungen. Python bietet eine Vielzahl von Tools zur statischen Code-Analyse, die helfen, Fehler frühzeitig zu erkennen. In diesem Dokument werden die drei populären Tools **flake8**, **pylint** und **bandit** vorgestellt.

2 Installation der Analyse-Tools

Alle drei Tools können bequem über `pip` installiert werden:

```
pip install flake8 pylint bandit
```

2.1 Installation in PyCharm

1. Öffne **PyCharm** und gehe zu `File` → `Settings` (`Preferences` auf macOS).
2. Navigiere zu `Project: <Projektname>` → `Python Interpreter`.
3. Klicke auf das **+**-Symbol, um neue Pakete zu installieren.
4. Suche nacheinander nach `flake8`, `pylint` und `bandit` und installiere sie.

3 Nutzung der Tools

3.0.1 3.1 Flake8 – Stil- und Syntaxprüfung

Flake8 kombiniert drei Analysetools:

- **PyFlakes** (Fehlersuche)
- **pycodestyle** (PEP 8-Überprüfung)
- **McCabe** (Komplexitätsanalyse)

Verwendung:

```
flake8 my_script.py
```

PyCharm-Integration:

1. Gehe zu `File` → `Settings` → `Tools` → `External Tools`.
2. Klicke auf Add und erstelle ein neues Tool:
   - Name: `Flake8`
   - Program: `flake8`
   - Arguments: `$FilePath$`
3. Klicke auf OK, um die Integration abzuschließen.

Nun kannst du Flake8 direkt über `Tools` → `External Tools` in PyCharm ausführen.

3.0.2 3.2 Pylint – Umfangreiche Code-Analyse

Pylint überprüft neben PEP-8-Konformität auch Namenskonventionen, Docstrings und Code-Duplizierungen.

Verwendung:

```
pylint my_script.py
```

PyCharm-Integration:

1. Gehe zu `File` → `Settings` → `Plugins`.
2. Suche nach `Pylint` und installiere das Plugin.
3. Nach der Installation kannst du `Pylint` direkt in PyCharm nutzen, indem du in der `Terminal`-Ansicht `pylint my_script.py` ausführst.

3.0.3 3.3 Bandit – Sicherheitsanalyse

Bandit identifiziert bekannte Sicherheitsrisiken im Python-Code.

Verwendung:

```
bandit -r my_script.py
```

PyCharm-Integration:

1. Erstelle einen **External Tool-Eintrag** (wie bei Flake8 beschrieben).
2. Setze als `Program` den Wert `bandit`, als Argumente `$FilePath$`.
3. Nun kannst du Bandit direkt aus PyCharm heraus ausführen.

4 Beispielskript mit Fehlern

Das folgende Python-Skript enthält Formatierungsfehler, Typisierungsprobleme und Sicherheitslücken, die von den genannten Tools erkannt werden können.

```python
import os
import sys
import subprocess


def bad_function(password):
    # Sicherheitslücke: Nutzung von subprocess mit unsicheren Eingaben
    command = "echo " + password
    os.system(command)


class my_class():
    def __init__(self, value):
        self.value = value

    def add(self, other):
        return self.value + other  # Typisierungsproblem


def unused_function():
    pass  # Diese Funktion wird nicht genutzt


if __name__ == "__main__":
    bad_function("mysecretpassword")  # Hardcodiertes Passwort
    obj = my_class(42)
    print(obj.add("test"))  # Fehlerhafte Typen
```

Erwartete Fehler:

- **flake8** meldet PEP-8-Fehler (z. B. fehlende Docstrings, inkonsistente Einrückungen, CamelCase-Klassen)
- **pylint** erkennt Namenskonventionen, ungenutzte Funktionen und Typinkonsistenzen
- **bandit** identifiziert Sicherheitsrisiken (z. B. `os.system()` mit unsicheren Eingaben, Hardcodierte Passwörter)

5 Fazit

Die Kombination von `flake8`, `pylint` und `bandit` verbessert die Code-Qualität erheblich. Während `flake8` sich auf Stil und Syntax konzentriert, bietet `pylint` eine tiefere Analyse der Code-Struktur und `bandit` hilft bei der Sicherheitsbewertung.
Empfohlene Praxis:

- Automatische Analysen in CI/CD-Pipelines integrieren
- Vor jedem Commit den Code mit den Tools prüfen
- Sicherheitskritischen Code mit `bandit` regelmäßig kontrollieren

Durch die Anwendung dieser Tools können viele Probleme frühzeitig erkannt und behoben werden, was zu sichererem und besser wartbarem Code führt.