

**HOCHSCHULE HEILBRONN**  
**Hochschule für Technik, Wirtschaft und Informatik**

**Studiengang Wirtschaftsinformatik (WIN)**

Fallstudie Entwicklungswerkzeuge

**Pylint**

vorgelegt bei  
Yunus Erdemir

von

Dennis Stehle  
Matr.-Nr. 209377

Robin Starkl  
Matr.-Nr. 203129

Oliver Daub  
Matr.-Nr. 203285

im  
Wintersemester 2022/2023

## Inhaltsverzeichnis

Abbildungsverzeichnis .....	iii
1. Allgemeine über Pylint .....	4
1.1. Linter .....	5
1.2. PEP 8 .....	5
1.3. Clean Code.....	7
2. Dokumentation.....	8
2.1. Voraussetzung.....	8
2.2. Installation .....	9
2.3. Anwendung.....	10
2.4. Konfigurierung .....	14
2.5. Typische Anwendungsszenarien .....	15
3. Vor- und Nachteile .....	17
3.1. Vorteile .....	17
3.2. Nachteile.....	17
4. Vergleich andere Tools .....	18
4.1. FLAKE 8 .....	18
4.2. PYFLAKES .....	19
Literature .....	iii

## Abbildungsverzeichnis

Abbildung 1: Python Version überprüfen.....	8
Abbildung 2: Pip Version überprüfen.....	9
Abbildung 3: Pip installieren.....	9
Abbildung 4: Pylint installieren .....	10
Abbildung 5: Pylint Verion überprüfen.....	10
Abbildung 6: Beispiel Pylint Ausgabe.....	11
Abbildung 7: Beispiel Auswertung.....	11
Abbildung 8: Pylint Ausgabe in JSON .....	12
Abbildung 9: Gefilterte Nachrichten.....	13
Abbildung 10: Configurations Datei.....	15
Abbildung 11: Configurationsdatei anwenden .....	15
Abbildung 12: Vergleich von Funktionen .....	19

## 1. Allgemeine über Pylint

Pylint ist ein Linter Werkzeug zur statischen Code-Analyse von Python Code. Das bedeutet, dass Pylint Code analysieren und bewerten kann, ohne diesen direkt auszuführen.<sup>1</sup> Vor allem Laufzeitfehler, wie Speicherprobleme, oder möglich Konflikte mit anderen Modulen oder Programmen werden dadurch nicht ausgelöst.

Während der Analyse durchsucht Pylint den Code nach möglichen Fehlern. Dazu zählen Programm- und Formatierungsfehler sowie verdächtige Strukturen. Bei der Überprüfung der Formatierung hält sich Pylint strikt an den PEP 8 Styleguide.<sup>2</sup> Weiteres dazu ist im Kapitel PEP 8 zu finden. Die durch Pylint aufgezeigten Probleme und Fehler helfen den Programmcode und dessen Qualität Stück für Stück zu verbessern.

Pylint wird neben der Zentralen Linter Funktion mit drei weiteren Werkzeugen ausgeliefert.

Mit *Pyverse* können aus dem Programmcode UML Diagramme erstellt werden, *similar* eignet sich zum Finden von Code Dubletten und mit *epylint* wird die Kompatibilität zu Emacs und Flymake gewährleistet.<sup>3</sup>

Pylint kann integriert oder unabhängig verwendet werden. Es kann als Plugin in verschiedenen Entwicklungsumgebungen und Codeeditoren integriert werden. In manchen Fällen ist die Integration jedoch unerwünscht, da es z.B. zu Performance Problemen führen kann, einfach nicht möglich ist oder der Nutzer bevorzugt die unabhängige Verwendung von Pylint. Dabei hilft die unabhängige Anwendungsmöglichkeit, bei der Pylint über das CLI (Kommandozeile, wie Bash, Shell, PowerShell, etc.) ausgeführt wird.

Pylint wurde von Sylvain Thénault entwickelt und 2001 das erste Mal veröffentlicht. Es selbst ist in Python geschrieben

---

<sup>1</sup> PyPI (1/6/2023).

<sup>2</sup> PyPI (1/6/2023).

<sup>3</sup> PyPI (1/6/2023).

## 1.1. Linter

Wie bereits erwähnt ist Pylint ein Linter Werkzeug. Im Allgemeinen sind Linter Werkzeuge zur Statischen Codeanalyse. Sie werden genutzt um (Potentielle-) Fehler im Programmcode aufzuzeigen und weisen auf Stilistische Fehler hin.

Ursprünglich entwickelt wurden Linter in der Unix Umgebung für die Programmiersprache C. Sie wurden eingesetzt, um Fehler und Schwächen von Compilern auszugleichen, um Kompilierung- und spätere Laufzeitfehler zu verhindern. Nachdem die Compiler immer besser und Linter unwichtiger wurden, kamen Funktionen für die Überprüfung von Formatierung und Style hinzu.<sup>4</sup>

## 1.2. PEP 8

PEP steht für Python Enhancement Proposal (dt.: Python Verbesserungsvorschlag). PEP ist eine Sammlung von Dokumenten, die den Umgang mit Python, best practices und Empfehlungen enthält. Die Nummer hinter „PEP“ Identifiziert immer genau ein Dokument. PEP 0 ist das Index PEP, in dem alle PEPs aufgeführt und gruppiert sind. PEP 8 ist der „Styleguide for Python Code“ und gehört zur Gruppe der Meta-PEPs. Sie beschreiben den Umgang mit anderen PEPs.<sup>5</sup> PEP 8 beschreibt, das Layout sowie viele Styling Richtlinien und Konventionen. Hier werden beispielhaft einige dieser Konventionen dargestellt. Für weitere Inhalte von PEP ist „peps.python.org“ aufzusuchen.

Das oberste Kredo von PEP ist es Konsistent zu sein. Es ist besser eine Konvention nicht zu beachten, anstatt innerhalb eines Codes wechseln und dann verschiedenen Stylings zu Nutzen.

### Code Layout

Die Konventionen zum Punkt Code Layout beschreiben wie die äußerliche Form von Python Code aussehen sollten. Dabei wird der Inhalt bzw. der Code selbst nicht beachtet.<sup>6</sup>

Einrückung von Code:

---

<sup>4</sup> Testim (2021).

<sup>5</sup> PEP 0 – Index of Python Enhancement Proposals (PEPs) | [peps.python.org](https://peps.python.org) (1/4/2023).

<sup>6</sup> PEP 8 – Style Guide for Python Code | [peps.python.org](https://peps.python.org) (1/4/2023).

Wie Code eingerückt werden sollte, ist klar definiert. Wird Code zum Beispiel in einer Funktion eingerückt sollten vier Leerzeichen verwendet werden. Das ist unabhängig von der Ebene auf der sich der Code befindet. Wird ein Code von der zweiten in die dritte Ebene eingerückt sollten wieder vier Leerzeichen verwendet werden (also acht Leerzeichen ab Ebene null). Das Einrücken mithilfe vom Tabulator ist nicht gewünscht. Werden in einem Bestehenden Code Leerzeichen oder Tabulator verwendet, ist ein Wechsel nicht möglich. Python lässt je Datei nur entweder Leerzeichen oder Tabulator zu.<sup>7</sup>

Position von Klammern:

Werden Klammern jeglicher Art verwendet, um Listen, Tupel, Dictionarys darzustellen, sind diese außerhalb der Liste zu setzen (siehe Beispiel).<sup>8</sup>

```
meine_liste = [
    1, 2, 3, 4
]                vs                meine_liste = [1, 2, 3, 4]
```

Zeilenlänge:

Nach PEP 8 sollte eine Zeile Code nicht länger als 79 Zeichen sein. Es wird sogar empfohlen nur maximal 72 Zeichen in einer Zeile zu Verwenden.<sup>9</sup>

Leerzeichen richtig setzen:

Leerzeichen sollten immer nach einem Operator oder Sonderzeichen wie Anführungszeichen oder Kommata gesetzt werden.<sup>10</sup>

```
if x == 4: print(x, y); x, y = y, x                vs                if x == 4 : print(x , y) ; x ,
y = y , x
```

---

<sup>7</sup> PEP 8 – Style Guide for Python Code | [peps.python.org \(1/4/2023\)](https://peps.python.org/1/4/2023/).

<sup>8</sup> PEP 8 – Style Guide for Python Code | [peps.python.org \(1/4/2023\)](https://peps.python.org/1/4/2023/).

<sup>9</sup> PEP 8 – Style Guide for Python Code | [peps.python.org \(1/4/2023\)](https://peps.python.org/1/4/2023/).

<sup>10</sup> PEP 8 – Style Guide for Python Code | [peps.python.org \(1/4/2023\)](https://peps.python.org/1/4/2023/).

## Namenskonventionen

Keine einzelnen Zeichen:

Ein Variablen-, Funktions-, Methoden- oder Klassenname sollte nie nur ein Zeichen lang sein. Ein Name aus ausschließlich Zahlen ist in Python nicht möglich. Zudem sollten alle Namen mit ASCII kompatibel sein.<sup>11</sup>

Variablennamen:

Variablennamen sollten immer dem Snake Case Format folgen. Dabei besteht der ganze Name aus Kleinbuchstaben. Wenn der Name aus mehreren Wörtern besteht, werden diese durch Unterstriche getrennt.<sup>12</sup>

Bsp: *snake\_case*

Klassennamen:

Klassennamen sollten immer dem Camel Case Format folgen. Dabei besteht der Name aus Kleinbuchstaben. Wenn der Name aus mehreren Wörtern besteht, wird der Anfangsbuchstabe jedes, außer dem ersten, Wortes großgeschrieben.<sup>13</sup>

Bsp: *camelCase*

Neben den offensichtlichen Styling Empfehlungen gibt es in PEP 8 auch einige wenige Programmierempfehlungen. Eine von ihnen empfiehlt die Return Werte aus Funktionen so schlank wie möglich zu halten. Das heißt es sollten keine Operationen im Return Wert vorgenommen werden. Diese sollten vorher in einer Variable gespeichert werden.<sup>14</sup>

### 1.3. Clean Code

Entgegen der Meinung Mancher wird Programmcode deutlich öfter gelesen, als geschrieben. Darum ist es besonders wichtig Clean Code zu schreiben. Clean Code beschreibt die Fähigkeit von Code, sauber, verständlich und wartbar zu sein. Damit wird die allgemeine Qualität des Codes erhöht. Das geschieht vor allem dadurch, dass sauberer Code für andere Beteiligte und den Schreiber selbst lesbarer wird. Durch die einheitliche Struktur und sinnvoll genutzte Kommentare wird der Code leicht

---

<sup>11</sup> PEP 8 – Style Guide for Python Code | [peps.python.org](https://peps.python.org/pep-0008/) (1/4/2023).

<sup>12</sup> PEP 8 – Style Guide for Python Code | [peps.python.org](https://peps.python.org/pep-0008/) (1/4/2023).

<sup>13</sup> PEP 8 – Style Guide for Python Code | [peps.python.org](https://peps.python.org/pep-0008/) (1/4/2023).

<sup>14</sup> PEP 8 – Style Guide for Python Code | [peps.python.org](https://peps.python.org/pep-0008/) (1/4/2023).

verständlich. Fehler und Bugs können somit deutlich leichter und schneller gefunden werden.<sup>15</sup> Das erhöht die Qualität des Codes weiter und kostet zudem weniger Ressourcen.

## 2. Dokumentation

### 2.1. Voraussetzung

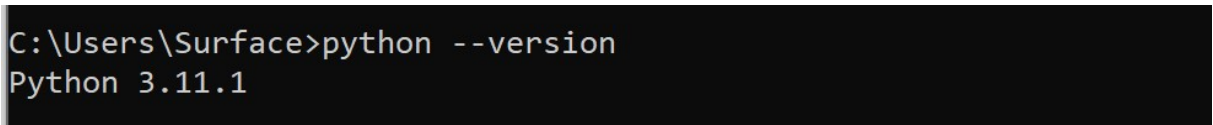
Hier werden Voraussetzungen bzw. anderes Werkzeug genannt, welche benötigt werden, um Pylint zu installieren und damit zu arbeiten.

#### Python

PyLint ist ein Tool für Python. Das setzt voraus, dass Python Version 3.7.2 oder alle darüber installiert sein müssen.<sup>16</sup>

Um Python zu installieren, gehen Sie auf die Webseite <https://www.python.org/downloads/>. Starten Sie die .exe-Datei und führen Sie die Installation durch.

Die Python Version kann über die Eingabeaufforderung überprüft werden. Um diese zu öffnen, muss WINDOWS-Taste + R gedrückt werden. Anschließend wird nach „cmd.exe“ gesucht. Schreibe „python --version“ in die Eingabezeile um die installierte Python-Version anzeigen zu lassen.<sup>17</sup>



```
C:\Users\Surface>python --version
Python 3.11.1
```

Abbildung 1: Python Version überprüfen

#### Pip

---

<sup>15</sup> t2informatik. Wir entwickeln Software. (15.9.2020).

<sup>16</sup> PyPI (1/6/2023).

<sup>17</sup> Makvana (3/5/2022).



Pip ist ein Tool, um Python packages zu verwalten<sup>18</sup>. D.h. es erlaubt die Installation und Verwaltung von Python packages, welche nicht zur Standard-Bibliothek von Python vorhanden sind.<sup>19</sup>

Falls Sie Python über <https://www.python.org/> bezogen haben oder mit einer integrierten Entwicklungsumgebung arbeiten, sollte pip bereits installiert sein.<sup>20</sup>

Um zu überprüfen, ob Sie pip installiert haben, starten Sie wieder die Eingabeaufforderung und schreiben Sie „python -m pip --version“.<sup>21</sup>

```
C:\Users\Surface>python -m pip --version  
pip 22.3.1 from C:\Users\Surface\AppData\Local\Programs\Python\Python311\Lib\site-packages\pip (python 3.11)
```

**Abbildung 2: Pip Version überprüfen**

Sollte es nicht installiert sein, können Sie das ebenso über die Eingabeaufforderung tun. Geben Sie hierfür „python -m ensurepip --upgrade“ ein.<sup>22</sup>

```
C:\Users\Surface>python -m ensurepip --upgrade
```

**Abbildung 3: Pip installieren**

## 2.2. Installation

Um Pylint zu installieren, benutzen wir den package manager „pip“. Das geschieht, indem „pip install pylint“ in die Eingabeaufforderung eingegeben wird<sup>23</sup>:

---

<sup>18</sup> Python PIP (1/6/2023).

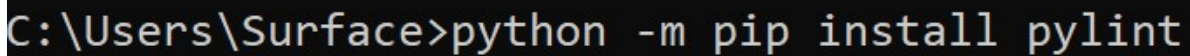
<sup>19</sup> Python (2/2/2022).

<sup>20</sup> Installation - pip documentation v22.3.1 (11/5/2022).

<sup>21</sup> Python PIP (1/6/2023).

<sup>22</sup> Installation - pip documentation v22.3.1 (11/5/2022).

<sup>23</sup> Logilab (2021, S. 11).



```
C:\Users\Surface>python -m pip install pylint
```

**Abbildung 4: Pylint installieren**

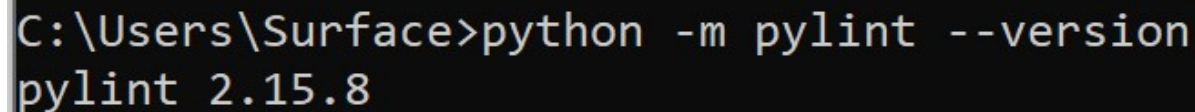
Falls Sie die Arbeit mit einer integrierten Entwicklungsumgebung bevorzugen, kann Pylint auch direkt über die package manager in der IDE Pylint installiert werden. Hier finden Sie alle IDEs, welche Pylint unterstützen:

[https://pylint.pycqa.org/en/latest/user\\_guide/installation/ide\\_integration/index.html](https://pylint.pycqa.org/en/latest/user_guide/installation/ide_integration/index.html).

### **2.3. Anwendung**

Pylint wird über die Eingabeaufforderung verwendet. In einer IDE wird es über das Terminal verwendet.

Um die installierte Version von Pylint zu prüfen, geben Sie „python -m pylint --version“ ein.<sup>24</sup>



```
C:\Users\Surface>python -m pylint --version
pylint 2.15.8
```

**Abbildung 5: Pylint Version überprüfen**

Sie können weitere Hilfe mit dem Befehl „python -m pylint --help“ anfragen, welche eine Liste von den Pylint-Befehlen ausgibt.<sup>25</sup>

#### Pylint auf Python-Dateien ausführen

Um Pylint nun auf einen in Python geschriebenen Code anzuwenden, muss „python -m pylint <dateiname>.py“ in die Kommandozeile eingetippt werden.<sup>26</sup> Hier muss nur beachtet werden, dass man sich im richtigen Verzeichnis befindet.

---

<sup>24</sup> Logilab (2021, S12)

<sup>25</sup> Logilab (2021, S. 12).

<sup>26</sup> Logilab (2021, S. 12).

Um Pylint parallel auf mehreren Datei auszuführen, werden die Dateien bei der Eingabe mit einem Komma getrennt: „python -m pylint <dateiname1>.py, <dateiname2>.py, ... „.<sup>27</sup>

Beispiel von einer Ausgabe:

```
C:\Users\Surface\Documents\Studium\PyLint\Workshop_Pylint>python -m pylint taschenrchner.py
***** Module taschenrchner
taschenrchner.py:1:41: C0303: Trailing whitespace (trailing-whitespace)
taschenrchner.py:2:0: W0311: Bad indentation. Found 8 spaces, expected 4 (bad-indentation)
taschenrchner.py:3:0: W0311: Bad indentation. Found 8 spaces, expected 4 (bad-indentation)
taschenrchner.py:75:0: C0304: Final newline missing (missing-final-newline)
taschenrchner.py:1:0: C0114: Missing module docstring (missing-module-docstring)
taschenrchner.py:11:12: R1734: Consider using [] instead of list() (use-list-literal)
taschenrchner.py:15:0: C0103: Constant name "calculation" doesn't conform to UPPER_CASE naming style (invalid-name)
taschenrchner.py:18:0: C0116: Missing function or method docstring (missing-function-docstring)
taschenrchner.py:19:4: C0103: Constant name "calculation" doesn't conform to UPPER_CASE naming style (invalid-name)
taschenrchner.py:19:4: W0603: Using the global statement (global-statement)
taschenrchner.py:36:0: C0116: Missing function or method docstring (missing-function-docstring)
taschenrchner.py:42:11: W0703: Catching too general exception Exception (broad-exception)
taschenrchner.py:38:17: W0123: Use of eval (eval-used)
taschenrchner.py:42:4: C0103: Variable name "e" doesn't conform to snake_case naming style (invalid-name)
taschenrchner.py:47:0: C0116: Missing function or method docstring (missing-function-docstring)
taschenrchner.py:61:0: C0103: Constant name "column_count" doesn't conform to UPPER_CASE naming style (invalid-name)
taschenrchner.py:62:0: C0103: Constant name "row_count" doesn't conform to UPPER_CASE naming style (invalid-name)
taschenrchner.py:63:0: C0103: Constant name "maximum_columns" doesn't conform to UPPER_CASE naming style (invalid-name)
taschenrchner.py:70:8: C0103: Constant name "column_count" doesn't conform to UPPER_CASE naming style (invalid-name)

-----
Your code has been rated at 6.04/10 (previous run: 7.29/10, -1.25)
```

**Abbildung 6: Beispiel Pylint Ausgabe**

In dem Beispiel wurde Pylint auf eine Datei namens „taschenrchner.py“ ausgeführt. Die Rückmeldungen sind wie folgt aufgebaut:

1. Name der Datei, in welcher die Meldung gefunden wurde
2. Zeile und Spalte, in welcher die Meldung zu finden ist
3. Message-id
4. Die Meldung bzw. Message an sich
5. Symbol (welches wie die message-id die Art der Rückmeldung identifiziert)

```
taschenrchner.py:3:0: W0311: Bad indentation. Found 8 spaces, expected 4 (bad-indentation)
taschenrchner.py:75:0: C0304: Final newline missing (missing-final-newline)
taschenrchner.py:1:0: C0114: Missing module docstring (missing-module-docstring)
taschenrchner.py:11:12: R1734: Consider using [] instead of list() (use-list-literal)
```

**Abbildung 7: Beispiel PyLint Nachrichten**

<sup>27</sup> Logilab (2021, S. 13).

Am Ende der Liste findet sich eine Bewertung. Pylint erstellt hierfür bei jeder Ausführung einen Score, welcher mit dem Score von der vorherigen Ausführung verglichen wird.<sup>28</sup>

### Ausgabe-Format ändern

Das Format, in welchem die Rückmeldung ausgegeben wird, kann geändert werden. Dazu hängt man am Ende vom Befehl ein „`--output-format=<format>`“ ein<sup>29</sup>. Um die einzelnen Rückmeldungen übersichtlicher auszugeben, empfiehlt sich die Rückmeldung in JSON ausgeben zu lassen:

```
C:\Users\Surface\Documents\Studium\PyLint\Workshop_Pylint>python -m pylint taschenrchner.py --output-format=json
[
  {
    "type": "convention",
    "module": "taschenrchner",
    "obj": "",
    "line": 1,
    "column": 41,
    "endLine": null,
    "endColumn": null,
    "path": "taschenrchner.py",
    "symbol": "trailing-whitespace",
    "message": "Trailing whitespace",
    "message-id": "C0303"
  },
  {
    "type": "warning",
    "module": "taschenrchner",
    "obj": "",
    "line": 2,
    "column": 0,
    "endLine": null,
    "endColumn": null,
    "path": "taschenrchner.py",
    "symbol": "bad-indentation",
    "message": "Bad indentation. Found 8 spaces, expected 4",
    "message-id": "W0311"
  },
]
```

**Abbildung 8: Pylint Ausgabe in JSON**

### Pylint-Rückmeldungen

In dem Bild kann bereits erkannt werden, dass die erste Zeile bei der Rückmeldung den „Type“ wiedergibt.

Beim Ausführen von Pylint prüft das Tool 5 verschiedenen Kategorien bzw. „Typen“ an Rückmeldungen, die auftreten können. Diese 5 Kategorien sind:

---

<sup>28</sup> Logilab (2021, S. 16).

<sup>29</sup> Logilab (2021, S. 14).

- Convention (C)
- Refactor (R)
- Warning (W)
- Error (E)
- Fatal (F)

Meldungen vom Typ „Convention“ geben Rückmeldung bezüglich der Einhaltung oder Verletzung von Coding Standards. Eine Refactor-Meldung gibt Hinweise zu „bad code smell“. Dabei handelt es sich um Code, welcher schlecht strukturiert ist, auch wenn er funktioniert<sup>30</sup>. Warning-Meldungen treten auf, wenn python-spezifische Probleme auftauchen können. Bei Meldungen von Typ „Error“ werden Syntax-fehler und (mögliche) Bugs wiedergegeben. Die letzte Kategorie von Rückmeldung ist „Fatal“, welche auftreten, wenn ein Error aufgetreten ist, welches PyLint seine Ausführung verhindert.<sup>31</sup>

Eine Liste aller möglichen Rückmeldungen finden Sie unter:  
[https://pylint.pycqa.org/en/latest/user\\_guide/messages/messages\\_overview.html](https://pylint.pycqa.org/en/latest/user_guide/messages/messages_overview.html).

### Message Control (Filtern)

Der Umfang der PyLint-Rückmeldung kann vom Nutzer angepasst werden. Mithilfe des Befehls „--disable=<symbol/message-id/message type>“, können die Nachrichten nach der Message-ID, dem Symbol oder dem Message Type gefiltert werden.<sup>32</sup>

```
C:\Users\Surface\Documents\Studium\PyLint\Workshop_Pylint>python -m pylint taschenrchner.py --disable=C
***** Module taschenrchner
taschenrchner.py:2:0: W0311: Bad indentation. Found 8 spaces, expected 4 (bad-indentation)
taschenrchner.py:3:0: W0311: Bad indentation. Found 8 spaces, expected 4 (bad-indentation)
taschenrchner.py:11:12: R1734: Consider using [] instead of list() (use-list-literal)
taschenrchner.py:19:4: W0603: Using the global statement (global-statement)
taschenrchner.py:42:11: W0703: Catching too general exception Exception (broad-exception)
taschenrchner.py:38:17: W0123: Use of eval (eval-used)
```

**Abbildung 9: Gefilterte PyLint Ausgabe**

Im Bild kann man sehen, dass keine Nachricht vom Typ „Convention“ vorhanden ist.

---

<sup>30</sup> t2informatik. Wir entwickeln Software. (13.11.2018).

<sup>31</sup> Logilab (2021, S. 15).

<sup>32</sup> Logilab (2021, S. 16).

Um alles rauszufiltern wird ein „--disable=all“ verwendet. Um bestimmte Rückmeldungen wieder anzeigen zu lassen gibt es den „--enable=<symbol/message-id/message type>“

## 2.4. Konfigurierung

Das Feature, welches Pylint besonders nützlich macht ist, dass mithilfe einer Konfigurationsdatei die zu überprüfenden Standards und Richtlinien vor-definieren werden können.

Pylint bietet einen Befehl „--generate-rcfile“, welcher eine vorstrukturierte Konfigurationsdatei ausgibt.<sup>33</sup>

Um die Ausgabe in einer Datei zu speichern, setzt man eine > Klammer und den Namen der Datei am Ende des Befehls:

„pylint --generate-rcfile > (Verzeichnis)/(Dateiname).pylintrc“. Dabei ist es wichtig, dass der Name der Konfigurationsdatei mit „.pylintrc“ endet. Die Konfigurationsdatei kann auch manuell erstellt werden.

Bei der Erstellung von Konfigurationsdateien werden „Checkers“ gesetzt. Mit den Checkers werden die Kriterien definiert, welche geprüft werden sollen. Eine Liste der Checkers findet man unter [https://pylint.pycqa.org/en/latest/user\\_guide/checkers/features.html](https://pylint.pycqa.org/en/latest/user_guide/checkers/features.html). Es muss zudem beachtet werden, dass die Checkers unter Header steht. Diese Header sind Überkategorien zu den Checkers.

---

<sup>33</sup> Configuration - Pylint 2.16.0-dev documentation (1/3/2023)



```
[BASIC]

# Naming style matching correct argument names.
argument-naming-style=snake_case

# Regular expression matching correct argument names. Overrides argument-
# naming-style. If left empty, argument names will be checked with the set
# naming style.
#argument-rgx=

# Naming style matching correct attribute names.
attr-naming-style=snake_case

# Regular expression matching correct attribute names. Overrides attr-naming-
# style. If left empty, attribute names will be checked with the set naming
# style.
#attr-rgx=

# Bad variable names which should always be refused, separated by a comma.
bad-names=foo,
          bar,
          baz,
          toto,
          tutu,
          tata
```

Abbildung 10: Configurations Datei Beispiel

Um die Nutzung einer Konfigurationsdatei zu spezifizieren wird der Befehl „--rcfile=<Dateiname>.pylintrc“ vor der Datei geschrieben:

```
\Workshop_Pylint>python -m pylint --rcfile=Config.pylintrc taschenrchner.py
```

Abbildung 11: Configurationsdatei anwenden

## 2.5. Typische Anwendungsszenarien

Pylint wird hauptsächlich verwendet, um einen Code transparent und übersichtlich zu gestalten, sodass er für die Wiederverwendung geeignet ist. Es prüft, anhand einer Konfigurations-Datei, ob Coding Standards eingehalten werden und (fatale) Errors, Warnungen und/oder Bad Code Smell vorliegt. Es wird primär benutzt, um seinen Code auf bekannte Konventionen, schlecht strukturierten Code oder logische Fehler zu prüfen, um ihn verständlicher für andere Personen zu schreiben. Das Alles ohne den Code auszuführen.

Die Verwendung von Pylint ist zum Beispiel in Open Source Projekten empfehlenswert. Bei Open Source Software ist es nämlich wichtig, dass der veröffentlichte Code von anderen Personen gelesen und verstanden werden kann. Wenn berücksichtigt ist, der Öffentlichkeit den Code zur Wiederverwendung bereitzustellen, kann Pylint ein behilfliches Tool sein. Es unterstützt das Schreiben von transparentem Code, indem es das Einhalten von bekannte Coding Standards, wie dem PEP8-Coding-Style, prüft.

Ein weiterer Fall, in welchem die Verwendung von Pylint nützlich ist, ist ein Softwareprojekt mit mehreren Entwicklern. In diesem Szenario ist es wichtig Coding Standards für alle Entwickler zu definieren, damit jeder den geschriebenen Code verstehen und nachvollziehen kann. Pylint liefert hierfür das Feature eine Konfigurations-Datei zu erstellen. In dieser kann vordefiniert werden, welche Standards und Regeln beim Code überprüft werden sollen. Haben z.B. Organisationen ihre eigenen Regeln zum Schreiben von Code, können sie es von Pylint prüfen lassen.

Für Unternehmen bzw. Organisationen, welche auf ihre Codequalität achten, kann Pylint auch von Vorteil sein. Durch die Konfigurations-Datei bietet PyLint eine Methode an, Standards und Qualitätskriterien nach einer vor-definierten Metrik prüfen zu lassen, um die Qualität des Codes zu evaluieren. Sind Kriterien definiert, um die Qualität eines Codes zu evaluieren, kann Pylint als Tool verwendet werden, um das Messen und Evaluieren der Codequalität zu beschleunigen.



### 3. Vor- und Nachteile

#### 3.1. Vorteile

Pylint kann Syntaxfehler in dem Programmcode identifizieren und Vorschläge zur Behebung machen. Dies ist besonders nützlich für große Projekte, bei denen Syntaxfehler schwer zu entdecken sind. Es kann auch dazu beitragen, Zeit zu sparen, die man sonst damit verbringen würde, Syntaxfehler manuell zu suchen und zu beheben.

Pylint kann den Programmcode auf Stilprobleme überprüfen und Vorschläge für Verbesserungen anbieten, um ihn lesbarer und einheitlicher zu machen. Dies kann dazu beitragen, dass der Programmcode einfacher zu lesen und zu verstehen ist, was wiederum die Wartbarkeit und den Support erleichtert.

Ein weiterer Vorteil gegenüber anderen Lintern ist, dass Pylint so konfiguriert werden kann, dass es bestimmte Codierungsstandards oder Stilrichtlinien durchsetzt werden können, um die Einheitlichkeit und die Qualität von dem Programmcode zu gewährleisten. Dies ist besonders nützlich für Teams oder Organisationen, die sicherstellen möchten, dass ihr Programmcode bestimmten Standards entspricht. Dazu kommt noch, dass Pylint bis heute noch weiterentwickelt und gepflegt wird.<sup>34</sup>

#### 3.2. Nachteile

Pylint ist nicht immer in der Lage, alle Probleme in dem Programmcode zu identifizieren, und kann falsch positive Ergebnisse liefern. Dies bedeutet, dass Pylint Fehler meldet, die in Wirklichkeit keine Probleme darstellen, oder Probleme übersieht, die tatsächlich vorliegen.

Dazu kommt, dass Pylint bei vielen Zeilen Programmcode eine längere Zeit braucht, um den Programmcode zu analysieren. Dies kann dazu führen, dass der Programmierer warten muss, bis Pylint die Analyse abgeschlossen hat, bevor er weiterarbeiten kann.

Pylint kann manchmal schwierig zu konfigurieren und anzupassen sein, insbesondere für Benutzer, die noch nicht vertraut mit dem Tool sind. Pylint bietet viele Optionen und Einstellungen, die verwirrend sein können.<sup>35</sup>

---

<sup>34</sup> Logilap(2022)

<sup>35</sup> Magnus Lie Hetland (2017)

## 4. Vergleich andere Tools

Die Werkzeuge PYFLAKES und FLAKE8 werden mit Pylint verglichen

### 4.1. FLAKE 8

Pylint und Flake8 sind beide Werkzeuge zur statischen Analyse von Python-Code, die dazu beitragen sollen, die Qualität und den Stil von Programmcode zu verbessern.

- Funktionen: Pylint hat mehr Funktionen als Flake8, einschließlich der Identifizierung von Syntaxfehlern, Stilproblemen und Code-Komplexität. Flake8 ist jedoch auf die Identifizierung von Syntaxfehlern und Stilproblemen beschränkt und bietet keine Vorschläge zur Behebung von Komplexitätsproblemen.
- Geschwindigkeit: Pylint ist in der Regel langsamer als Flake8, da es eine umfassendere Analyse des Programmcodes durchführt und mehr Funktionen und Optionen bietet. Pylint durchsucht den Programmcode gründlicher und prüft ihn auf eine größere Anzahl von Problemen, was zu längeren Analysezeiten führen kann.
- Verwendung: Flake8 hat weniger Optionen und Einstellungen wie Pylint und ist somit leichter zu verwenden, jedoch kann es nicht Individuell angepasst werden. Pylint kann direkt in das Python-Interpreter-Modul integriert werden und kann daher von jedem Python-Skript aus aufgerufen werden. Flake8 kann nicht direkt in Python integriert werden und muss als externe Anwendung aufgerufen werden.<sup>36</sup>

Im Allgemeinen ist Pylint ein umfassenderes und mächtigeres Tool als Flake8, da es mehr Funktionen bietet und den Programmcode auf eine größere Anzahl von Problemen überprüft. Flake8 ist jedoch einfacher zu verwenden und kann schnell auf Syntaxfehler und Stilprobleme überprüfen, was es für Benutzer attraktiv macht, die schnelles Feedback benötigen oder die sich nicht mit zu vielen Optionen und Einstellungen auseinandersetzen möchten.

---

<sup>36</sup> Flake8 — flake8 6.0.0 documentation (2022)

## 4.2. PYFLAKES

Pyflakes ist ein Werkzeug zur statischen Analyse von Python-Code.

- Funktionen: Pyflakes ist auf die Identifizierung von Syntaxfehlern beschränkt und bietet keine Vorschläge zur Behebung. Pylint bietet umfassendere Funktionen als Pyflakes, einschließlich der Identifizierung von Syntaxfehlern, Stilproblemen und Code-Komplexität.
- Geschwindigkeit: Pyflakes ist schneller als Pylint, da es nur auf Syntaxfehler sucht.
- Verwendung: Pylint hat eine größere Lernkurve als Pyflakes, da es viele Optionen und Einstellungen bietet, die es ermöglichen, das Tool an die Bedürfnisse des Benutzers anzupassen. Pyflakes ist einfacher zu verwenden, da es weniger Funktionen und Optionen hat, jedoch ist Pyflakes leichter zu bedienen.<sup>37</sup>

Im Allgemeinen ist Pylint ein umfangreicheres Tool als Pyflakes, da Pyflakes sich nur auf Syntaxfehlern beschränkt, jedoch ist die Ausführung schneller.

Problem	Pylint output	Flake8 output	PyFlakes output
unnecessary semicolon	✓	✓	✗
missing newlines	✓	✓	✗
missing whitespaces	✗	✓	✗
missing docstring	✓	✗	✗
unused arguments	✓	✓	✓
incorrect naming style	✓	✗	✗
selecting element from the list with incorrect index	✗	✗	✗
dividing by zero expression	✗	✗	✗
missing return statement	✓	✗	✗
calling method that doesn't exist	✓	✓	✓
unnecessary pass statement	✓	✗	✗

Abbildung 12: Vergleich von Funktionen <sup>38</sup>

<sup>37</sup> PyPL(2022)

<sup>38</sup> Kulkowski (2022).

In der Abbildung 11 gehen wir noch auf verschiedene Probleme ein. Jedes Programm wurde ohne weitere Einstellungen ausgeführt. Pylint hat die meisten Fehler gemeldet. Flake8 hat weniger Fehler gemeldet als Pylint. Es hat nicht über fehlende Docstrings, falsche Benennungsstile und die unnötige Pass-Anweisung informiert. Allerdings meldete nur Flake8 fehlende Leerzeichen. PyFlakes hat die kürzeste Fehlerliste zurückgegeben. Es wurde nur über das unbenutzte Argument, den undefinierten Wert und die nicht vorhandene Aufrufmethode berichtet. Keiner der getesteten Linters warnte uns vor der Auswahl eines Elements aus der Liste mit falschem Index oder dem Division-durch-Null-Ausdruck. Dies ist jedoch erklärbar, da es sich nicht um Stil-, sondern um Logikfehler handelt.<sup>39</sup>

---

<sup>39</sup> Kulkowski (2022).

## Literature

Logilab. 2022. Pylint - code analysis for Python | [www.pylint.org](http://www.pylint.org). <https://pylint.org/>. Zugriffen: 5. Januar 2023.

Flake8: Your Tool For Style Guide Enforcement — flake8 6.0.0 documentation. <https://flake8.pycqa.org/en/latest/>. Zugriffen: 5. Januar 2023.

PyPI. 2023. pyflakes. <https://pypi.org/project/pyflakes/>. Zugriffen: 5. Januar 2023.

Kulkowski, Maciej. 2022. Improve your Python code with Python linters | DS Stream. <https://dsstream.com/improve-your-python-code-quality/>. Zugriffen: 5. Januar 2023.

Magnus Lie Hetland (2017):Beginning Python, DOI 10.1007/978-1-4842-0028-5

2023. PEP 0 – Index of Python Enhancement Proposals (PEPs) | [peps.python.org](https://peps.python.org). <https://peps.python.org/pep-0000/>. Zugriffen: 6. Januar 2023.

2023. PEP 8 – Style Guide for Python Code | [peps.python.org](https://peps.python.org). <https://peps.python.org/pep-0008/>. Zugriffen: 6. Januar 2023.

PyPI. 2023. pylint. <https://pypi.org/project/pylint/>. Zugriffen: 6. Januar 2023.  
t2informatik. Wir entwickeln Software. 2020. Was ist Clean Code? - Wissen kompakt - t2informatik. *t2informatik GmbH*, 15. September.

Testim. 2021. What Is a Linter? Here's a Definition and Quick-Start Guide. <https://www.testim.io/blog/what-is-a-linter-heres-a-definition-and-quick-start-guide/>. Zugriffen: 6. Januar 2023.

2022. Installation – pip documentation v22.3.1. <https://pip.pypa.io/en/stable/installation/> Zugriffen: 6. Januar 2023

PyPI. 2023. pylint. <https://pypi.org/project/pylint/> Zugriffen: 6. Januar 2023

2023. Python PIP. [https://www.w3schools.com/python/python\\_pip.asp](https://www.w3schools.com/python/python_pip.asp) Zugriffen: 6. Januar 2023

Python, Real. 2022. Using Python's pip to Manage Your Projects' Dependencies. Real Python, 2. Februar.

Makvana, Mahesh. 2022. How to Check the Python Version on Windows, Mac, and Linux. How-To Geek, 5. März

Kanjilal, Joydip. 2022. Understanding code smells and how refactoring can help. TechTarget, 3. November.

t2informatik. Wir entwickeln Software. 2018. Was ist ein Code-Smell? - Wissen kompakt - t2informatik. t2informatik GmbH, 13. November.

Codeac. 2023. Pylint configuration | Codeac. <https://www.codeac.io/documentation/pylint-configuration.html>. Zugegriffen: 6. Januar 2023.

Logilab, PyCQA and contributors. 2021. Pylint Documentation. Release 2.7.1. <https://pylint.pycqa.org/en/2.7.1/pdf/>.

2023. Configuration - Pylint 2.16.0-dev documentation. [https://pylint.pycqa.org/en/latest/user\\_guide/configuration/index.html](https://pylint.pycqa.org/en/latest/user_guide/configuration/index.html). Zugegriffen: 6. Januar 2023.