

## Inhaltsverzeichnis

LS 2 – Infos – Programmierrichtlinien.....	2
LS 1 – Infos – Code-Reviews.....	4

# LS 2 – Infos – Programmierrichtlinien

## Programmierrichtlinien – Warum, Ziele?

- Einheitliche Struktur, Layout, Namensgebung
  - ✓ Bessere Übersicht
  - ✓ Einfacher zu lesen
  - ✓ Einfacher zu ändern und zu erweitern
- Gängelung des Programmierers, Freiheitsbeschränkung

Programme werden  
öfters gelesen als  
geschrieben.

## Programmierrichtlinien für Python

<https://peps.python.org/pep-0008/>  
<https://pep8.org/>

Python  
Programmierrichtlinien  
PEP8

## Python Programmierhilfsmittel

- **Formatierung:**
  - black
  - autopep8
- **Statische Code Analyser**
  - flake8
  - pylint

Python IDE  
Autoformatierer  
Code Analyzer (Linter)

## Python Programmierrichtlinien BK-GuT

- Code entspricht den in pep8 festgelegten Standards
- Einheitlicher Kopfkomentar am Anfang jeder Datei mit Docstrings;
- Funktionen (und Klassenbeschreibungen) mit Docstrings
- Kommentare deutsch
- Bezeichner (Funktionsnamen, Variablennamen, ...) nur Standard-ASCII-Zeichen (Buchstaben, Ziffern, \_) benutzen; keine Umlaute und Sonderzeichen benutzen, für Bezeichner beschreibende Namen benutzen; deutsche Namen benutzen
- Strings in "Gänsefüßchen"
- Zeilen können bis zu 130 Zeichen lang sein.
- Importe am Dateianfang; jeder Import in eine eigene Zeile
- Encoding: UTF-8
- Ein- und Ausgaben nicht mit Berechnungen / Verarbeitung mischen, sondern jeweils in eine eigene Zeile.
- Code mit einem Formater (z.B. autopep8, black) überarbeitet
- Code mit statischen Code Analyzern (flake8, pylint) geprüft

For Python, **PEP 8** has emerged as the style guide that most projects adhere to; it promotes a very readable and eye-pleasing coding style. Every Python developer should read it at some point; here are the most important points extracted for you:

- Use 4-space indentation, and no tabs.  
4 spaces are a good compromise between small indentation (allows greater nesting depth) and large indentation (easier to read). Tabs introduce confusion, and are best left out.
- Wrap lines so that they don't exceed 79 characters.  
This helps users with small displays and makes it possible to have several code files side-by-side on larger displays.
- Use blank lines to separate functions and classes, and larger blocks of code inside functions.
- When possible, put comments on a line of their own.
- Use docstrings.
- Use spaces around operators and after commas, but not directly inside bracketing constructs: `a = f(1, 2) + g(3, 4)`.
- Name your classes and functions consistently; the convention is to use **CamelCase** for classes and **lower\_case\_with\_underscores** for functions and methods. Always use `self` as the name for the first method argument (see *A First Look at Classes* for more on classes and methods).
- Don't use fancy encodings if your code is meant to be used in international environments. Python's default, UTF-8, or even plain ASCII work best in any case.
- Likewise, don't use non-ASCII characters in identifiers if there is only the slightest chance people speaking a different language will read or maintain the code.

# LS 1 – Infos – Code-Reviews

## Was ist ein Code-Review?

- Bei einem Code-Review wird ein Programm (Programmabschnitt) durch eine oder mehrere andere Personen begutachtet.
- Ein Code-Review ist eine systematische Untersuchung von Quellcode mit dem Ziel, Fehler, Mängel und Verbesserungsmöglichkeiten zu finden.
- Ein Code-Review ist eine Aktivität im Bereich der Qualitätssicherung.
- Bei einem Code-Review findet auch ein Wissenstransfer statt.

## Was sind die erhofften Vorteile?

- Verbesserte Softwarequalität
- Wissensteilung im Team
- Einhaltung von Programmierrichtlinien/Konventionen/Standards
- Einarbeitung neuer/unerfahrener Mitarbeiter
- Verbesserte Zusammenarbeit
- Reduzierte Projektkosten und -zeiten

## Auf was soll bei einem Code-Review geachtet werden?

- Wird die geforderte Funktionalität erreicht?
- Ist der gewählte Algorithmus gut oder unnötig kompliziert? Gibt es einen besseren und sollte man diesen noch einbauen?
- Ist das Programm zu verstehen?
- Ist die Benutzeroberfläche für den Bediener verständlich? Ist sie konsistent und frei von Rechtschreibfehlern und Vertippern?
- Werden Programmierrichtlinien, z.B. Namenskonventionen, Formatierung, Kommentare, eingehalten?
- Welche Testmöglichkeiten gibt es? Was wurde getestet?
- Werden Programmierwerkzeuge optimal genutzt?

## Was ist bei der Durchführung zu beachten?

- Es wird der Code begutachtet, nicht der Ersteller.
- Wenn möglich, konkrete Verbesserungsmöglichkeiten nennen und nicht nur Fehler oder Mängel auflisten.
- Positive Sachen erwähnen.
- Ersteller und Reviewer sind auf der selben Seite.
- Den zu begutachtenden Code vorher verfügbar machen.
- Der Code-Review soll nicht zu lang sein( <60 min, in Schule deutlich kürzer).
- Der zu begutachtende Code soll nicht zu umfangreich sein( < 400 Zeilen, in Schule deutlich kleiner).
- Bei Code-Reviews mit Checklisten arbeiten.