LU09.L01 - Code mit Linter verbessern

Musterlösung zu LU09.A01 - Code mit Linter verbessern. Analysiert wurden die Python-Dateien im Repository m450-lu09-a01 (ohne das Hilfsskript `_run_pylint.py`) mit drei verbreiteten Lintern.

Die Zahlen gelten für den Stand des Repositorys im November 2026 und die genannten Einstellungen. Andere Versionen oder Einstellungen ergeben andere Zahlen. Bewertet wird, ob die Gruppe ihre Einstellungen begründet und die Meldungen kritisch einordnet.

Konfiguration

Die BZZ Codingstandards basieren auf PEP 8, weichen aber in einigen Punkten davon ab. Diese Punkte müssen in jedem Linter angepasst werden:

Codingstandard	**pylint**	**flake8**	**ruff**
Zeilen maximal **120** Zeichen (PEP 8: 79)	`max-line-length=120`	`max-line-length = 120`	`line-length = 120`
Docstrings für Klassen und Methoden zwingend	Standard (C0115, C0116)	nur mit Plugin `flake8-docstrings`	Regelgruppe `D` aktivieren
Docstring-Format `:param:` / `:return:`	Erweiterung `docparams`	–	–
Ausrichten von = untereinander ist **erlaubt**	wird nicht gemeldet	E221 ausschalten (`extend-ignore = E221`)	E221 ist standardmässig aus
snake_case, PascalCase, Konstanten gross	Standard (C0103)	nur mit Plugin `pep8-naming`	Regelgruppe `N`

Wo wird konfiguriert?

Linter	**Konfigurationsdatei**	**Kommandozeile**
pylint	`.pylintrc` oder `pyproject.toml` (`[tool.pylint]`)	`pylint –max-line-length=120 *.py`
flake8	`.flake8`, `setup.cfg` oder `tox.ini` (`[flake8]`)	`flake8 –max-line-length=120 *.py`
ruff	`ruff.toml` oder `pyproject.toml` (`[tool.ruff]`)	`ruff check –line-length 120 –select E,W,F,N,D *.py`

Beispiel `.flake8`:

```
[flake8]
max-line-length = 120
extend-ignore = E221
```

Beispiel `ruff.toml`:

```
line-length = 120

[lint]
select = ["E", "W", "F", "N", "D", "B", "PL"]
ignore = ["D300", "D212", "D205", "D400", "D415", "D208", "D200"]  #
Docstring-Formatierung nach PEP 257, nicht Teil der BZZ-Standards

[lint.per-file-ignores]
"test_*.py" = ["PLR2004"]  # feste Werte in Tests sind erwünscht
```

Ergebnisse

Linter (Einstellung)	**Meldungen**	**Arten von Meldungen**	**false positive**
pylint 4.0 (Zeilenlänge 120)	63	Convention (C): 41, Warning (W): 22	0
flake8 (Zeilenlänge 120)	50	E (pycodestyle): 43, W (Whitespace): 7	4 (E221)
ruff (E, W, F, N, D, B, PL, Zeilenlänge 120)	169	D (Docstrings): 147, PL: 6, W/E: 7, weitere: 9	ca. 125

pylint

Anzahl	**Code**	**Meldung**	**Bedeutung**
21	W0311	bad-indentation	Einrückung nicht 4 Leerzeichen
18	C0116	missing-function-docstring	Testfunktionen ohne Docstring
9	C0114	missing-module-docstring	Modul ohne Docstring
4	C0304	missing-final-newline	letzte Zeile ohne Zeilenumbruch
4	C0115	missing-class-docstring	Klasse ohne Docstring
3	C0305	trailing-newlines	zu viele Leerzeilen am Dateiende
3	C0301	line-too-long	Zeile länger als 120 Zeichen
1	W0246	useless-parent-delegation	Methode ruft nur die Oberklasse auf

Bewertung: **6.50/10**. Alle Meldungen entsprechen einem Verstoss gegen die Codingstandards.

Mit der Erweiterung `–load-plugins=pylint.extensions.docparams` findet pylint zusätzlich, dass im Docstring von `Person.address` der Parameter `adresse` statt `address` beschrieben ist. Das findet kein anderer Linter.

flake8

Anzahl	**Code**	**Bedeutung**
21	E111	Einrückung kein Vielfaches von 4
7	E302	erwartet 2 Leerzeilen vor Klasse oder Funktion
4	W292	kein Zeilenumbruch am Dateiende
4	E221	mehrere Leerzeichen vor Operator (**false positive**, Ausrichten ist laut Standard erlaubt)

Anzahl	Code	Bedeutung
3	W391	Leerzeile am Dateiende
3	E501	Zeile zu lang
3	E305	erwartet 2 Leerzeilen nach Klasse oder Funktion
3	E265	Blockkommentar muss mit ''# '' beginnen
2	E303	zu viele Leerzeilen

flake8 findet ohne Plugins **keine fehlenden Docstrings**, dafür die fehlenden Leerzeilen, die pylint nicht meldet.

ruff

Die meisten Meldungen betreffen die Formatierung von Docstrings nach PEP 257 (D300: **„“„** `statt '''`, D212/D205: Zusammenfassung in der ersten Zeile, D400/D415: Punkt am Satzende). Diese Regeln stehen nicht in den BZZ-Standards und sind deshalb **false positive**. Ebenso PLR2004 (feste Zahlen in Tests). Übrig bleiben die fehlenden Docstrings (D100, D101, D102), Leerzeilen am Dateiende und zu lange Zeilen.

Was kein Linter findet

- **Attribut ausserhalb des Konstruktors:** `Employee._salary` wird erst im Setter angelegt. Wer `print()` vor dem Setzen des Lohns aufruft, erhält einen `AttributeError`.
- **Deutsche Bezeichner und Kommentare** wie `# nur für Test` (Standard: Bezeichner in Englisch).
- **Sprechende Namen:** Variablen wie `p` oder `m` im `__main__`-Block.
- **Tippfehler in Docstrings** wie «Initlalisiert» oder «Telefo-Nummer».

Ein Linter ersetzt also kein Code-Review (siehe LU09c - Codereview).

Vergleich

Kriterium	pylint	flake8	ruff
Konfiguration	sehr viele Optionen, eigene rcfile	einfach, wenige Optionen	einfach, Regeln nach Gruppen wählbar
Docstrings	ja	nur mit Plugin	ja, sehr streng (PEP 257)
Formatierung	Einrückung, Zeilenlänge	sehr genau (Leerzeilen, Abstände)	wie flake8
Logikfehler	einige (z. B. unnötige Methoden)	nur mit pyflakes (unbenutzte Imports)	viele, mit Gruppen B und PL
Geschwindigkeit	langsam	mittel	sehr schnell
Automatisch korrigieren	nein	nein	ja (`–fix`, 61 Meldungen)
false positive bei BZZ-Standards	kaum	wenige	viele, wenn alle Regeln aktiv sind

Fazit: Für die BZZ-Standards passt **pylint** am besten, weil Docstrings ohne Zusatz geprüft werden und kaum false positive auftreten. **ruff** ist schnell und kann vieles automatisch korrigieren, muss aber

sorgfältig konfiguriert werden. **flake8** ergänzt pylint gut bei Leerzeilen und Abständen.

M450-LU09, M450-D1B, M450-D1A

Marcel Suter

From:
https://wiki.bzz.ch/ - **BZZ - Modulwiki**

Permanent link:
https://wiki.bzz.ch/modul/m450/learningunits/lu09/loesungen/linter?rev=1794933828

Last update: **2026/11/17 17:43**