Classroom 50: Wie bewertet wird

Die Bewertung macht pygrader50 — pytest für die Funktion, pylint für den Stil. Beides zählt zur Note.

Die drei Konfigurationsdateien

Format wie bisher in den BZZ-Templates, es ändert sich nichts am Inhalt:

Datei	Inhalt
`unittests.json`	welche Tests laufen, mit Timeout und Punkten
`lint.json`	welche Dateien gelintet werden und wie viele Punkte das gibt
`pylintrc`	pylint-Konfiguration

unittests.json

```
[
  { "name": "test_ggt", "function": "test_ggt", "timeout": 10,
"points": 2 }
]
```

lint.json

```
{ "files": ["main.py"], "ignore": [], "max": 5 }
```

Vorlage: examples/bundle im pygrader50-Repository.

Wo die Dateien gesucht werden

Pro Datei, in dieser Reihenfolge:

1. `<CLASSROOM>/autograders/<SLUG>/` im Config-Repo (das *Bundle*). Von der Lehrperson kontrolliert, für Lernende nicht editierbar.
2. `.github/autograding/` im Studi-Repo. Der bisherige Ort, bleibt als Fallback.
3. `$PYGRADER50_CONFIG_DIR` — nur für lokale Entwicklung.

Fehlt beides, wird eine **0/0**-Abgabe aufgezeichnet und im Log gewarnt. Der Job wird dabei absichtlich nicht rot: Eine Aufgabe ohne hinterlegte Bewertung ist ein Konfigurationsstand, kein Infrastrukturfehler.

Der Fallback auf `.github/autograding/` erlaubt einen Rollout in Ruhe: Die Templates funktionieren unverändert weiter, während die Bundles nach und nach ins Config-Repo wandern.

Ein Bundle anlegen

```
classroom50/
└── <CLASSROOM>/
    └── autograders/
        └── <SLUG>/
            ├── unittests.json
            ├── lint.json
            └── pylintrc
```

Nach dem Push bündelt der Workflow `publish-pages` den Ordner zu `autograders/<SLUG>.tar.gz`; der Runner entpackt ihn und pygrader50 liest von dort.

Kein `autograder.py` und keine `tests.json` in diesen Ordner legen. Beides hat Vorrang vor dem Klassen-Default und würde damit die BZZ-Bewertung — und insbesondere das Linting — für diese Aufgabe abschalten.

Punkte

Unittests

Ein Eintrag pro Testfall. `passed` heisst: volle Punktzahl erreicht. Jeder Fall läuft als **eigener** pytest-Aufruf mit eigenem Timeout, damit ein hängender Test die übrigen nicht mitreisst.

Linting

Ein einziger Eintrag `Linting`:

```
Punkte = pylint-Note / 10 * max
passed = Punkte > 0
```

Eine Konventionsmeldung kostet also **Punkte**, färbt den Commit-Status aber nicht rot. Ohne diese Regel wäre praktisch jeder Commit rot.

Bei wenigen Statements und einem E**…**-Fehler wird die pylint-Note negativ und auf 0 geklemmt. `Linting: 0` ist in dem Fall korrekt, kein Defekt.

Rundung

`result.json` verlangt **ganze Zahlen**, es wird gerundet. Der exakte Wert steht im Feedback-Text.

Beispiel-Feedback

```
### classroom50 autograde: 0/7

## Unittests
| name     | feedback        | expected | actual | points | max |
| -------- | --------------- | -------- | ------ | ------ | --- |
| test_ggt | Assertion Error | 8        | None   | 0      | 2   |

**0.00/2.00 Points (0.00%)**
```

Die Spalten *expected* und *actual* stammen aus dem pytest-Hook `pytest_assertrepr_compare`, der vor dem Lauf ins Checkout kopiert wird.

Warum nicht die deklarativen Tests von Classroom 50

Classroom 50 bringt mit `tests.json` einen deklarativen Weg mit. Die BZZ benutzt ihn bewusst nicht:

- Der Runner löst eine aufgaben-eigene `tests.json` **vor** dem Klassen-Default auf. Eine Aufgabe mit deklarativen Tests bekommt also kein Linting mehr.
- Im deklarativen Pfad wäre entweder die proportionale Lint-Note **oder** der grüne Commit-Status zu haben, nie beides: Der Interpreter setzt `success` nur, wenn *alle* Zeilen `passed` sind, und vergibt das Flag selbst.

Pro Aufgabe lässt sich beides nicht mischen — pro Klasse schon. Deshalb stehen alle Aufgaben auf `„autograder“: „default“`.

Was pygrader50 bewusst nicht tut

- **Keine Zusatzfelder in `result.json`.** CLI und Dashboard von Classroom 50 parsen strikt; alles Menschenlesbare gehört in den Release-Text.
- **Kein `$GITHUB_OUTPUT`.** Status und Zusammenfassung leitet der Runner selbst aus `result.json` ab — ein Kanal weniger, der auseinanderlaufen kann.
- **Kein `pip install -r requirements.txt`** aus dem Studi-Repo. pygrader50 bringt eigene, gepinnte Abhängigkeiten mit; das hält die Bewertung reproduzierbar. Ausserdem sind die alten Template-Pins auf Python 3.14 nicht lauffähig.

Lokal ausprobieren

```
cd /pfad/zum/studi-repo
CLASSROOM=<CLASSROOM> \
ASSIGNMENT=<SLUG> \
SUBMISSION_TAG=submit/2026-08-13T12-00-00Z-abc1234 \
OWNER=anna \
```

```
python -m pygrader50
cat release-body.md
```

Ohne RUNNER_TEMP entfällt die Bundle-Suche, es zählt also `.github/autograding/` im Checkout. Details: CLI-Referenz.