Portfolio

Fingerübung 5 - Testing!

Diesmal geht es um die Themen Repository anlegen, Testing (Session 15.11.) und Continuous Integration (Session 22.11).

a) Neues GitHub repository erstellen und darin eine Funktion `fair_sharer` bereitstellen.

Es soll ein neues eigenes GitHub repository erstellt werden, z.B. `fairsharer`. Darin soll in einem Unterordner (z.B. `fairsharer/` oder `src/`) eine Python .py-Datei mit einer Funktion erstellt werden die Werte (z.B. Vermögen...) über die Zeit verteilt.

Die Funktion hat folgende Form:

```
def fair_sharer(values, num_iterations, share=0.1):
    """Runs num_iterations.
    In each iteration the highest value in values gives a fraction (share)
    to both the left and right neighbor. The leftmost field is considered
    the neightbor of the rightmost field.

    Examples:
    fair_sharer([0, 1000, 800, 0], 1) --> [100, 800, 900, 0]
    fair_sharer([0, 1000, 800, 0], 2) --> [100, 890, 720, 90]

    Args
    values:
        1D array of values (list or numpy array)
    num_iteration:
        Integer to set the number of iterations
    """
    # code
    return values_new
```

Der Algorithmus für die Funktion ist wie folgt:

Es wird `num_iterations` mal wiederholt:

- Der höchste Wert in `values` gibt jeweils den Teil `share` an die beiden Nachbarn ab. Also `value * share` an den "linken" Nachbarn (i - 1) und den "rechten" Nachbarn (i +1).
- Die Felder mit Indizes [0] und [-1] sind ebenfalls Nachbarn, also das erste und das letzte Feld.

Beispiele (wie oben im docstring) wären:

```
fair_sharer([0, 1000, 800, 0], 1)  # --> [100, 800, 900, 0]
fair_sharer([0, 1000, 800, 0], 2)  # --> [100, 890, 720, 90]
```

Achtung: Mit dieser Funktion werden wir in ca. 2 Wochen im Live Coding arbeiten.

b) Erstellt eine `requirements.txt` Datei im Root-Ordner

Erstellt eine `requirements.txt` Datei die alle Bibliotheken aufzählt, die für die verwendung eures Codes nötig sind. Das werden hier nicht viele sein, es sollten aber zumindest auch ein Linter (Vorschlag: `ruff`) und `pytest` enthalten sein.

c) Eine passende test-Funktion erstellen und im selben Ordner im Repository hinterlegen.

Es soll passend zur oben beschriebenen Funktion `fair_sharer()` eine Test-Funktion (unit test) erstellt werden. Diese sollte `test_fair_sharer()` heißen und in einer eigenen Datei, z.B: `test_fairsharer.py` zu finden sein, so dass diese mit `pytest` ausgeführt werden kann.

d) Continuous Integration mit GitHub workflows erstellen

Das Thema *Continuous Integration* behandeln wir in der Session am 22.11.

Lege einen GitHub workflow an.
Dazu muss ein Ordner `.github/workflows/` angelegt werden.
In diesen Ordner kommt dann eine .yaml-Datei, z.B `my_first_actions.yaml`

Eine Vorlage dafür könnte folgender yaml Code sein (den ruhig erst einmal so verwenden):

```yaml
name: Our first python CI

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  My-first-CI:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3
    - name: Set up Python
      uses: actions/setup-python@v3
      with:
        python-version: 3.10
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt
    - name: Lint with Ruff
      run: |
        # ***add your command to run ruff here***
    - name: Test with pytest
      run: |
        pytest
```

Sobald der Workflow läuft (wie/wann wird er eigentlich ausgeführt?), soll er noch erweitert werden.

In der oben angegeben Variante werden zwei Python Versionen getestet. Es sollen nun aber auch **verschiedene Betriebssysteme getestet** werden (ubuntu-latest, windows-latest, macos-latest). Dafür bitte die matrix-Methode der GitHub workflows nutzen, siehe auch: https://docs.github.com/en/actions/using-jobs/using-a-matrix-for-your-jobs.

Abgabe

Die folgenden Punkte bitte alle in einem Textdokument (Word oder pdf) beantworten und auf Moodle hochladen.

1. **Link** zum oben erstellten Repository (in dem dann die Funktion, die Testfunktion und ein funktionierender Workflow sein sollten).