Applied object-oriented programming

Teacher: Carlos Natalino / Examiner: Paolo Monti

Course channel on Chalmers Play

Before you turn this assignment list in, make sure everything runs as expected. First, **restart the kernel** and then **run all cells**. Then, check if all the tests run correctly. Note that if one of the problems present an error, the following ones **will not** be tested.

In case of discrepancies between the problem command and the tests, you should solve it having in mind the tests.

There are two types of cell:

1. *solution cells:* These are the cells where you write your answer, or modify the existing code to solve the problem.
2. *test cells:* These cells are used to test whether your solution is correct or not. If the tests run correctly, you should see a message `tests passed`. Otherwise, you should see an error message.

Delete the line `raise NotImplementedError()` from the problems that you solve.

Do not delete or add any cell in this file. All cells that you need are already in place.

If you want to execute a cell, select the cell and press **CTRL+Enter** (in Windows) or **CMD+Enter** (in macOS) or click on the **Run cell** button.

If you want, you can solve this programming assignment using Google Colab

Link: https://colab.research.google.com/

Just uncomment the following installation line to do so.

Preparation: Run the cell below every time you start working on this file, and every time you restart the kernel.

In [1]:
```
# the line below installs all packages in Google Colab.
# !pip install redis types-redis python-dotenv matplotlib flask flask-testing toml
```

The command below will list all the packages installed, confirming that the installation was successful.

In [2]:
```
!pip freeze
```

```
anyio==4.8.0
argon2-cffi==23.1.0
argon2-cffi-bindings==21.2.0
arrow==1.3.0
astroid==3.3.8
asttokens==3.0.0
async-lru==2.0.4
attrs==24.3.0
babel==2.16.0
beautifulsoup4==4.12.3
black==24.10.0
bleach==6.2.0
blinker==1.9.0
bs4==0.0.2
certifi==2024.12.14
cffi==1.17.1
charset-normalizer==3.4.1
click==8.1.8
colorama==0.4.6
comm==0.2.2
contourpy==1.3.1
coverage==7.6.10
cryptography==44.0.0
cycler==0.12.1
debugpy==1.8.11
decorator==5.1.1
defusedxml==0.7.1
dill==0.3.9
executing==2.1.0
fastjsonschema==2.21.1
flake8==7.1.1
Flask==3.1.0
Flask-Testing==0.8.1
fonttools==4.55.3
fqdn==1.5.1
greenlet==3.1.1
gunicorn==23.0.0
h11==0.14.0
httpcore==1.0.7
httpx==0.28.1
idna==3.10
iniconfig==2.0.0
ipykernel==6.29.5
ipython==8.31.0
ipywidgets==8.1.5
isoduration==20.11.0
isort==5.13.2
itsdangerous==2.2.0
jedi==0.19.2
Jinja2==3.1.5
json5==0.10.0
jsonpointer==3.0.0
jsonschema==4.23.0
jsonschema-specifications==2024.10.1
jupyter==1.1.1
jupyter-console==6.6.3
```

```
jupyter-events==0.11.0
jupyter-lsp==2.2.5
jupyter_client==8.6.3
jupyter_core==5.7.2
jupyter_server==2.15.0
jupyter_server_terminals==0.5.3
jupyterlab==4.3.4
jupyterlab_pygments==0.3.0
jupyterlab_server==2.27.3
jupyterlab_widgets==3.0.13
kiwisolver==1.4.8
lorem-text==2.1
MarkupSafe==3.0.2
matplotlib==3.10.0
matplotlib-inline==0.1.7
mccabe==0.7.0
mistune==3.1.0
mypy==1.14.1
mypy-extensions==1.0.0
nbclient==0.10.2
nbconvert==7.16.5
nbformat==5.10.4
nest-asyncio==1.6.0
notebook==7.3.2
notebook_shim==0.2.4
numpy==2.2.1
overrides==7.7.0
packaging==24.2
pandocfilters==1.5.1
parso==0.8.4
pathspec==0.12.1
pep8-naming==0.14.1
pillow==11.1.0
platformdirs==4.3.6
playwright==1.49.1
pluggy==1.5.0
prometheus_client==0.21.1
prompt_toolkit==3.0.48
psutil==6.1.1
pure_eval==0.2.3
pycodestyle==2.12.1
pycparser==2.22
pyee==12.0.0
pyflakes==3.2.0
Pygments==2.19.1
pylint==3.3.3
pyparsing==3.2.1
pytest==8.3.4
python-dateutil==2.9.0.post0
python-dotenv==1.0.1
python-json-logger==3.2.1
pywin32==308
pywinpty==2.0.14
PyYAML==6.0.2
pyzmq==26.2.0
redis==5.2.1
```

```
referencing==0.35.1
requests==2.32.3
rfc3339-validator==0.1.4
rfc3986-validator==0.1.1
rpds-py==0.22.3
Send2Trash==1.8.3
setuptools==75.8.0
six==1.17.0
sniffio==1.3.1
soupsieve==2.6
stack-data==0.6.3
terminado==0.18.1
tinycss2==1.4.0
toml==0.10.2
tomlkit==0.13.2
tornado==6.4.2
traitlets==5.14.3
types-cffi==1.16.0.20241221
types-pyOpenSSL==24.1.0.20240722
types-python-dateutil==2.9.0.20241206
types-redis==4.6.0.20241004
types-requests==2.32.0.20241016
types-setuptools==75.6.0.20241223
typing_extensions==4.12.2
uri-template==1.3.0
urllib3==2.3.0
wcwidth==0.2.13
webcolors==24.11.1
webencodings==0.5.1
websocket-client==1.8.0
Werkzeug==3.1.3
widgetsnbextension==4.0.13
```

In [3]:
```
%load_ext autoreload
import sys
try:
    from utils import validate_python_code
except:
    print("It seems this file is in the wrong folder. "
          "Make sure to place it in the `programming-assignments` folder/project.",
          file=sys.stderr)
```

Validation of your installation

In this notebook, students check if their installation is working correctly.

Note that there are some errors that are purposefully placed here to test your setup. Run all cells in this notebook, and send the result screenshots in canvas. There is an appropriate *computer installation* assignment.

In [4]:
```
%%writefile initial_file.py
import datetime
import getpass
import os
import platform
import random
from typing import Sequence

from matplotlib.figure import Figure


def validating_sum_of_squares(seq: Sequence[str | float | int]) -> Sequence[bool]:
    return_seq = []
    for element in seq:
        try:
            temp = False
            num = int(element)
            if num < 0:
                return_seq.append(False)
                continue
            for i in range(int(num ** 0.5) + 1):
                remainder = (num - i ** 2) ** 0.5
                if remainder.is_integer():
                    return_seq.append(True)
                    temp = True
                    break
            if temp is False:
                return_seq.append(False)
        except (ValueError, TypeError):
            return_seq.append(False)
    return return_seq

def generate_plot(single_line_fig: Figure):
    Numbers = []
    for i in range(100):
        if i < 50:
            Numbers.append(random.randint(0, 10))
        else:
            Numbers.append(random.randint(10, 20))

    axes: Axes = single_line_fig.gca()
    axes.set_title(
        f"""If you see this, your installation was successful!
        Date: {datetime.datetime.now()}
        Folder: {os.getcwd()}
        User: {getpass.getuser()}
        OS: {platform.system()}"""
    )
    axes.plot(Numbers)
    single_line_fig.tight_layout()
    return single_line_fig
```

```
Overwriting initial_file.py
```

In [5]:
```
%%writefile tests_validating_sum_of_squares_solution.py
# make sure to run this cell before running the next one
```

```
from initial_file import validating_sum_of_squares


def tests_validating_sum_of_squares() -> None:

    test_cases = [
        (
            ["2", "x", -10, 3.3, "asd", None, "b", 4.0],
            [True, False, False, False, False, False, False, True],
        ),
        ([9, "x", -9, None], [True, False, False, False]),
    ]

    for _in, _out in test_cases:
        _res = validating_sum_of_squares(_in)
        assert (
            _res == _out
        ), f"The function with input `{_in}` should return the value \
    `{_out}` of type `{type(_out)}`\n but returned the value `{_res}` \
    of type `{type(_res)}`."
```

```
Overwriting tests_validating_sum_of_squares_solution.py
```

In [6]:
```
# test cell
print("Running the tests", file=sys.stderr)
try:
    import initial_file
except:
    raise ValueError("You did not execute your solution cell!")
try:
    from initial_file import validating_sum_of_squares
except:
    raise ValueError("Your solution does not contain the right function!")

!coverage run -m pytest tests_validating_sum_of_squares_solution.py
```

```
Running the tests
============================= test session starts ==============================
platform win32 -- Python 3.13.1, pytest-8.3.4, pluggy-1.5.0
rootdir: C:\Users\Student\Downloads\programming-assignments
configfile: pyproject.toml
plugins: anyio-4.8.0
collected 1 item

tests_validating_sum_of_squares_solution.py .                            [100%]

============================== 1 passed in 0.96s ===============================
C:\Users\Student\Downloads\programming-assignments\venv\Lib\site-packages\coverage\i
norout.py:508: CoverageWarning: Module codeapp was never imported. (module-not-impor
ted)
  self.warn(f"Module {pkg} was never imported.", slug="module-not-imported")
C:\Users\Student\Downloads\programming-assignments\venv\Lib\site-packages\coverage\c
ontrol.py:892: CoverageWarning: No data was collected. (no-data-collected)
  self._warn("No data was collected.", slug="no-data-collected")
```

In [7]:
```
print("Validating Python code", file=sys.stderr)
validate_python_code("tests_validating_sum_of_squares_solution.py")
print('tests passed', u'\u2713')
```

```
Validating Python code
```

Code Quality Analysis: Pass

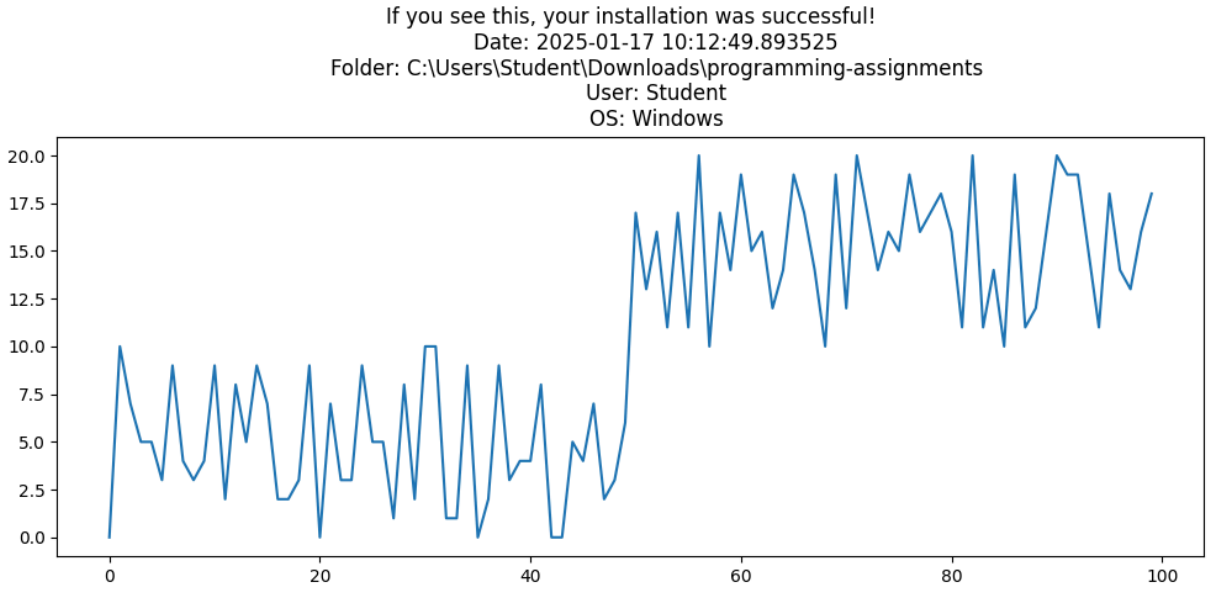
No problem was found

```
tests passed ✓
```

In [8]:
```
%autoreload 2
# %matplotlib inline
import matplotlib.pyplot as plt

from initial_file import generate_plot

fig = plt.figure(figsize=(10, 5))
fig = generate_plot(fig)
plt.show()
```



In [9]:
```
# test cell
validate_python_code("initial_file.py")
print('tests passed', u'\u2713')
```

Code Quality Analysis: Fail

General formatting:

```
16    ····temp·=·False
17    ····num·=·int(element)
18    ····if·num·<·0:
19    ····return_seq.append(False)
20    ····continue
21  - ····for·i·in·range(int(num·**·0.5)·+·1):
21  - ····remainder·=·(num·-·i·**·2)·**·0.5
21  + ····for·i·in·range(int(num**0.5)·+·1):
22  + ····remainder·=·(num·-·i**2)·**·0.5
23    ····if·remainder.is_integer():
24    ····return_seq.append(True)
25    ····temp·=·True
26    ····break
27    ····if·temp·is·False:
28    ····return_seq.append(False)
29    ····except·(ValueError,·TypeError):
30    ····return_seq.append(False)
31    ··return·return_seq
32
33  +
34    def·generate_plot(single_line_fig:·Figure):
35    ··Numbers·=·[]
36    ··for·i·in·range(100):
37  - ····if·i·<·50:·
37  + ····if·i·<·50:
38    ····Numbers.append(random.randint(0,·10))
39    ····else:
40    ····Numbers.append(random.randint(10,·20))
41
42    ··axes:·Axes·=·single_line_fig.gca()
```

Variable types and use:

Line	Description	Code line
33	expected 2 blank lines, found 1 def generate_plot(single_line_fig: Figure): ^	`def·generate_plot(single_line_fig:·Figure):`
33	Function is missing a return type annotation [no-untyped-def]	`def·generate_plot(single_line_fig:·Figure):`

Line	Description	Code line
34	variable 'Numbers' in function should be lowercase Numbers = [] ^	····Numbers·=·[]
36	trailing whitespace if i < 50: ^	········if·i·<·50:·
41	undefined name 'Axes' axes: Axes = single_line_fig.gca() ^	····axes:·Axes·=·single_line_fig.gca()
41	Name "Axes" is not defined [name-defined]	····axes:·Axes·=·single_line_fig.gca()

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[9], line 2
      1 # test cell
----> 2 validate_python_code("initial_file.py")
      3 print('tests passed', u'\u2713')

File ~\Downloads\programming-assignments\utils.py:84, in validate_python_code(filename, **args)
     79     raise ValueError(
     80         f"Error while decoding the response. "
     81         f"Detail: {e}. Response: {response_html.text}"
     82     )
     83 if data["fail"]:
---> 84     raise ValueError("The code needs change. Check messages above.")

ValueError: The code needs change. Check messages above.
```