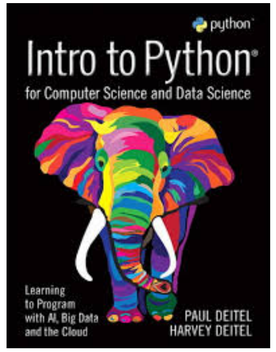
Advanced Programming in Python (APPY - APIP)

Larissa C. Shimomura

larissa.capobiancoshimomura@uhasselt.be

Overview



- Lecturer: Larissa C. Shimomura
- TA: Thomas Munoz
- TA: Sebastian Bugedo

- Lectures are from 9-10.30. They are streamed and recorded.
- Work sessions are from 11-12.30.
  - On Campus students - 11-12
  - DL students - 12-12:30

- Q&A for DL students are ***(mostly)*** scheduled at 17.00 (check calendar for the Q&A dates and times!)
  - Indicate your participation in the Q&A using this form (link also available in the course overview on BB)

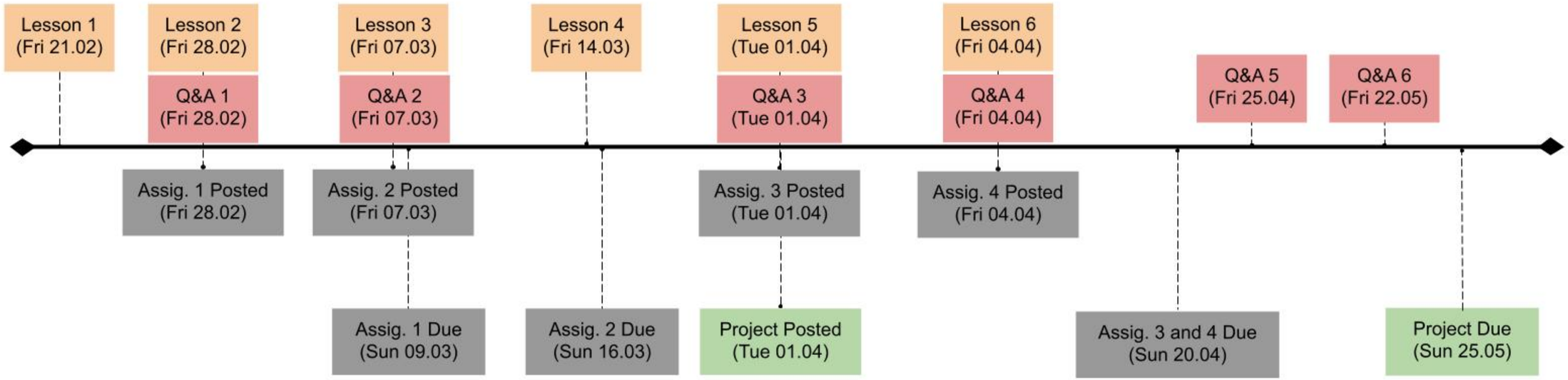
- Lesson 1: 21/02
  - Lecture:
    - Introduction
    - Python Tools: typechecking, linting, formatting, testing, jupyter notebooks [AM, Ch1]
    - Regular expressions. [DD, Ch8] and [AM, Ch9]
    - Files and exceptions. [DD, Ch9]
  - Work session:
    - Installing and using Python Tools in VSCode
    - PyTest
- Lesson 2: 28/02
  - Lecture:
    - Object-Oriented programming I [DD, Ch10.1-10.6]
    - Decorators
  - Work session:
    - Regular expressions
    - Files and exceptions
- Lesson 3: 07/03
  - Lecture:
    - Object-oriented programming II [DD, Ch10.7-10.13]
    - dataclasses
  - Work session:
    - Object-oriented programming I
- Lesson 4: 14/03
  - Lecture:
    - Recursion and sorting. [DD, Ch11] and [AM, Ch2]
  - Work session:
    - Object-oriented programming II
- Lesson 5: 01/04
  - Lecture:
    - Basic graph algorithms. [AM, Ch7+8].
    - NetworkX
  - Work session:
    - Recursion and sorting
- Lesson 6: 04/04
  - Lecture
    - Data Science packages for Python
    - Numpy and Pandas. [DD, Ch7]
    - Conda
  - Work session:
    - Basic graph algorithms
- **Q&A 5: 25/04**
    - **OC 11:00-12:00**
    - **DL 17:00-18:00**
- Start Project
  - Start: 01/04
  - Deadline: 25/05

Examination

- 30% on assignments
- 30% on project
- 40% on programming exam

Individual

No Plagiarism



Examination

- 30% on assignments
- 30% on project
- 40% on programming exam

Individual

No Plagiarism

- **Attention!! The use of AI generated code on assignments and projects is also considered plagiarism.**
- Keep in mind that you will not have access to internet during the exam! Only documentation files (see blackboard for documentation.zip)
- Very important to get used to the python documentation and be able to read it

Examination

- 30% on assignments
- 30% on project
- 40% on programming exam
- To pass the student **MUST** do all parts to pass the course - at least 40% of each part
- Partial grades are transferable to the 2nd Chance exam

Individual

No Plagiarism

Assignments and Project

- Both of the assignments and projects will be submitted through Blackboard - (deadlines are already available in the course blackboard, check the overview document for all dates)

- More information about the viewing of the exam, second exam opportunities and feedback on the project and assignments will be given later closer to the exam date.

Python
Version 3.12.9

Development environment

Package + Environment Manager
You can use it to install python + libraries in
a self contained environment

- https://www.anaconda.com/

Python packages

- PyTest - Framework for creating software tests
- Installation steps available in the setup.pdf (part of worksession 1 - later today)
- PyTest will be used for checking the assignments and project - important to make sure that it is working as expected!

Lecture 1

- Files
- JSON
- Exceptions
- Regular Expressions
- Some tools

Today: short intro in these topics

Learn by doing

Consult Python documentation !!

https://docs.python.org/3/

Tools

- Typing: mypy
  - https://mypy.readthedocs.io/en/stable/cheat_sheet_py3.html
- Linter: flake8, pylint, pylance, mypy, pep8, …
  - Flag syntactic and stylistic problems
  - Settings: search for “type checking mode“, set value to “basic”
- Formatter: autopep8, black, …
  - Rewrites/formats code
  - Formatting makes code easier to read by human beings
  - https://realpython.com/python-pep8/
  - https://peps.python.org/pep-0008/

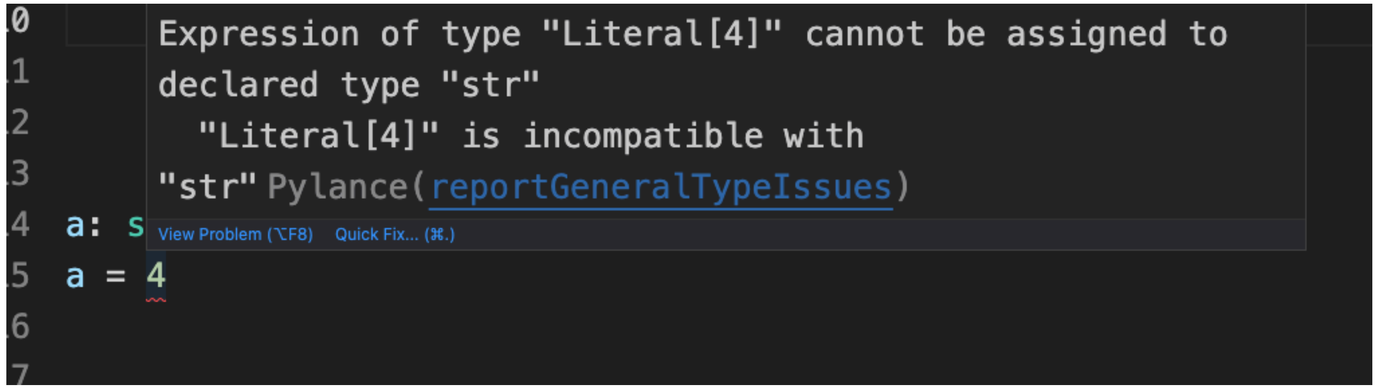
Python and Typing

- Python is dynamically typed
  - Variables take the type of the object that is assigned to them

```
a = "I am a string"
a = 4
```

- Python allows to add type hints

```
a: str = "I am a string"
a = 4
```



Linter will warn about a possible error

Why do we want typing?

Capture errors early

```python
def my_sum(a: int, b: int) -> int:
    return a + b


number1 = input("Enter a number:")
number2 = input("Enter a number:")
print(f"The sum is {my_sum(number1,number2)}")
```

```python
def my_sum(a: int, b: i
    return a + b


number1 = input("Enter
number2 = input("Enter
print(f"The sum is {my_sum(number1,number2)}")
```

```
(variable) number1: str
Argument of type "str" cannot be assigned to parameter "a"
of type "int" in function "my_sum"
  "str" is incompatible with
"int" Pylance(reportGeneralTypeIssues)
Argument 1 to "my_sum" has incompatible type "str";
```

Why do we want typing?

Capture errors early

```
def my_sum(a: int, b: int) -> int:
    return a + b


number1 = input("Enter a number:")
number2 = input("Enter a number:")
print(f"The sum is {my_sum(number1,number2)}")
```

On input 1 and 2, the above code prints 12.

Example

https://mypy.readthedocs.io/en/stable/cheat_sheet_py3.html

```python
# This is how you declare the type of a variable
age: int = 1

# You don't need to initialize a variable to annotate it
a: int  # Ok (no value at runtime until assigned)

# Doing so is useful in conditional branches
child: bool
if age < 18:
    child = True
else:
    child = False
```

Useful types

https://mypy.readthedocs.io/en/stable/cheat_sheet_py3.html

```python
# For most types, just use the name of the type.
# Note that mypy can usually infer the type of a variable from its value,
# so technically these annotations are redundant
x: int = 1
x: float = 1.0
x: bool = True
x: str = "test"
x: bytes = b"test"

# For collections on Python 3.9+, the type of the collection item is in brackets
x: list[int] = [1]
x: set[int] = {6, 7}

# For mappings, we need the types of both keys and values
x: dict[str, float] = {"field": 2.0}  # Python 3.9+

# For tuples of fixed size, we specify the types of all the elements
x: tuple[int, str, float] = (3, "yes", 7.5)  # Python 3.9+

# For tuples of variable size, we use one type and ellipsis
x: tuple[int, ...] = (1, 2, 3)  # Python 3.9+
```

Functions

https://mypy.readthedocs.io/en/stable/cheat_sheet_py3.html

```python
# This is how you annotate a function definition
def stringify(num: int) -> str:
    return str(num)

# And here's how you specify multiple arguments
def plus(num1: int, num2: int) -> int:
    return num1 + num2

# If a function does not return a value, use None as the return type
# Default value for an argument goes after the type annotation
def show(value: str, excitement: int = 10) -> None:
    print(value + "!" * excitement)
```

Tools

- Typing: mypy
  - https://mypy.readthedocs.io/en/stable/cheat_sheet_py3.html
- Linter: flake8, pylint, pylance, mypy, pep8, …
  - Flag syntactic and stylistic problems
  - Settings: search for “type checking mode“, set value to “basic”
- Formatter: autopep8, black, …
  - Rewrites/formats code
  - Formatting makes code easier to read by human beings
  - https://realpython.com/python-pep8/
  - https://peps.python.org/pep-0008/

https://code.visualstudio.com/docs/languages/python