Advanced Python / Kurs rozszerzony języka Python

Martin Böhm (University of Wrocław)

October 2022

Lab 01: Introduction

About me 1

- Email: `boehm@cs.uni.wroc.pl`
- Assistant professor (adiunkt), UWr, second year of this position.
- Czech national (I speak English, German, Czech, but not Polish).
- Research focus: Combinatorial optimization / Approximation and online algorithms.
- Using Python since 2007 – contributions to an old Ubuntu “proprietary drivers” manager.
- Other language proficiency: C++, C#.

About me 2

I am:

About me 2

I am:

- Somebody you can always contact via email.

About me 2

I am:

- Somebody you can always contact via email.
- Somebody who will do their best to help with any issues in Python, homework and such.

About me 2

I am:

- Somebody you can always contact via email.
- Somebody who will do their best to help with any issues in Python, homework and such.

About me 2

I am:

- Somebody you can always contact via email.
- Somebody who will do their best to help with any issues in Python, homework and such.

I am not:

- Somebody who knows every little detail about Python (and wants you to know the same).

About me 2

I am:

- Somebody you can always contact via email.
- Somebody who will do their best to help with any issues in Python, homework and such.

I am not:

- Somebody who knows every little detail about Python (and wants you to know the same).
- Focused that much on non-academic programming (design patterns, coding styles, etc).

Main goal

Main goal of the lab:

Main goal

Main goal of the lab: Build something cool in Python.

Main goal

Main goal of the lab: Build something cool in Python.

- Learn more about the fairly vast libraries/modules of Python;
- Code quickly while keeping in mind the time complexities and costs;
- Finally, create a final course program that actually does something useful

Main goal

Main goal of the lab: Build something cool in Python.

- Learn more about the fairly vast libraries/modules of Python;
- Code quickly while keeping in mind the time complexities and costs;
- Finally, create a final course program that actually does something useful . . . and put it on Github/your CV.

Structure of the lab

Common section (∼45 minutes):

1. Additions to lecture that might be useful for the homework;
2. Answering questions about the new homework/last lecture;
3. Explaining common mistakes in the last homework.

Individual section (∼45 minutes):

1. Answering one-on-one questions.
2. The present students can present their homework.
   (Potentially faster feedback and grading!)

Q: Is lab attendance mandatory?

Structure of the lab

Common section (∼45 minutes):

1. Additions to lecture that might be useful for the homework;
2. Answering questions about the new homework/last lecture;
3. Explaining common mistakes in the last homework.

Individual section (∼45 minutes):

1. Answering one-on-one questions.
2. The present students can present their homework. (Potentially faster feedback and grading!)

Q: Is lab attendance mandatory?

- Not all the time.
- Do not block the time slot – you should be available to attend when requested.
- Prof. Młotkowski's "once a month" rule.

Python IDEs

- *A good choice of IDE:* PyCharm, `https://www.jetbrains.com/community/education/`, academic license available free of charge for UWr students.
- VS Code is another good choice, if you are proficient with it.
- No hard requirements.

One appetizer for the course

1. Code snippet 1: How complex can $x+=1$ really be?

My homework philosophy

Every homework exercise should make you think.

My homework philosophy

Every homework exercise should make you think.

- Make sure the code is well documented. Any homework longer than 20 lines requires some form of in-code comments.
- If a vague requirement is given, or the requirement is too strong, try to support it broadly and explain what is not supported and why.
- If you know an algorithm with better time complexity, but you did not program it, defend your choice in the documentation.
- Try your solution on large(r) data, to check that it performs reasonably well.
- Do not copy the code from your colleagues, even if you edit it slightly, we *can and will find out*!

Homework philosophy, part 2

Every single homework that gets full points must contain:

1. 5-10 testing inputs and outputs of your code. (If you are used to testing with sample input/output text files, it is okay.)
   - Inputs should be of different sizes and should be creative – test the limits of your code!
2. Documentation. Usually either in a form of a .txt file or inside the homework itself.
   - Necessary documentation: what packages or functions are the building blocks?
   - What is the main algorithmic idea of the code?
   - What is the worst-case running time of your code?
   - If a vague requirement is given, specify the parameters of your program.

Homework philosophy, part 3

High-school mentality vs. professional mentality through the lens of homework:

"If the task is vaguely defined, I will make use of this to do the least amount of work necessary. Surely I must get full points as I did what is asked."

Homework philosophy, part 3

High-school mentality vs. professional mentality through the lens of homework:

"If the task is vaguely defined, I will make use of this to do the least amount of work necessary. Surely I must get full points as I did what is asked."

vs.

"If the task is vaguely defined, I will first try to communicate to learn the expected parameters; if not possible, I will set up reasonable defaults and I will clearly document the limits of my approach."

Notes on strings

The builtin type str

- Type `str` – A sequence of *Unicode code points*.
- No separate "character type" – `chr()` returns a string of length 1.
- Random access is cheap, like lists/arrays.
- Strings are `immutable sequences` – you cannot edit them in place.

The builtin type str

- Type `str` – A sequence of *Unicode code points*.
- No separate “character type” – `chr()` returns a string of length 1.
- Random access is cheap, like lists/arrays.
- Strings are `immutable sequences` – you cannot edit them in place.
- $\Rightarrow$ Any editing takes (at least) linear time in the length of a string.

The builtin type str

- Type `str` – A sequence of *Unicode code points*.
- No separate “character type” – `chr()` returns a string of length 1.
- Random access is cheap, like lists/arrays.
- Strings are `immutable sequences` – you cannot edit them in place.
- $\Rightarrow$ Any editing takes (at least) linear time in the length of a string.
- No analog of `StringBuilder` in Python.

Useful methods of str

- `str.join([iterable])` – joins an iterable (list) to a big string in linear time.
- `str.split()` – splits a string into a list, by default stripping and splitting by *whitespaces*.
- *whitespace* – `str.isspace()` – checks the Unicode category.
- `str.casefold()` – returns a *casefolded* copy of the string, (ideally) for caseless matching.
- **Warning!** Casefolding is not a panacea – see e.g. `https://www.w3.org/TR/charmod-norm/#definitionCaseFolding`.

Fancier string formatting

1. *Triple quoted string* – can contain newlines. Triple quoted string at the start of a function – *docstring* – essentially a base documentation for the function.
2. *Formatted string literals* – Combining text and short code expressions, popular in many languages.
   ```
   print(f"Number of results:  {len(results)}.")
   ```

The module `unicodedata`

- A few more Unicode-related methods, such as `unicodedata.decimal(chr)`, `unicodedata.category(chr)`.
- Unfortunately **no** `unicodedata.punctuation(chr)`.

Note: Module `string` should be avoided, as it only contains historical ASCII function support!

Supporting multiple languages

"Make sure that your function supports multiple languages / Sprawdź, czy funkcja poprawnie działa dla tekstów obcojęzycznych:"

Supporting multiple languages

"Make sure that your function supports multiple languages / Sprawdź, czy funkcja poprawnie działa dla tekstów obcojęzycznych:"