The design of “Intermediate Python Programming” in the Computer Science minor programme at the TU Delft

Frank Mulder

Target audience of the course

- Students with a non-CS background
- In their 3rd year of the BSc, they sign up to learn about Computer Science for 5 months. The Python course runs for 10 weeks (5 EC).
- They have experience with Python, but only as a “glue language” to call libraries like Numpy
- Number of students: 250

Focus

- Python for software engineering
- From “scripting language” to “serious programming language”
- Usage of Numpy is prohibited

Design decisions

Prerequisites

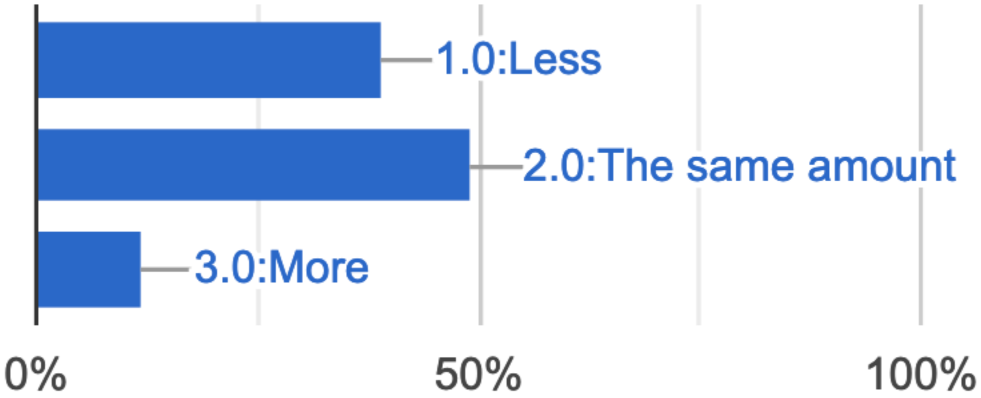
- Until 2021, the course started from zero
- Problem: big difference between levels of students
- Many students have used Python in some way in their major
- From 2022: soft prerequisite for CS minor
  - Basic programming skills (variables, if statements, for loops)

Prerequisites: student evaluation results

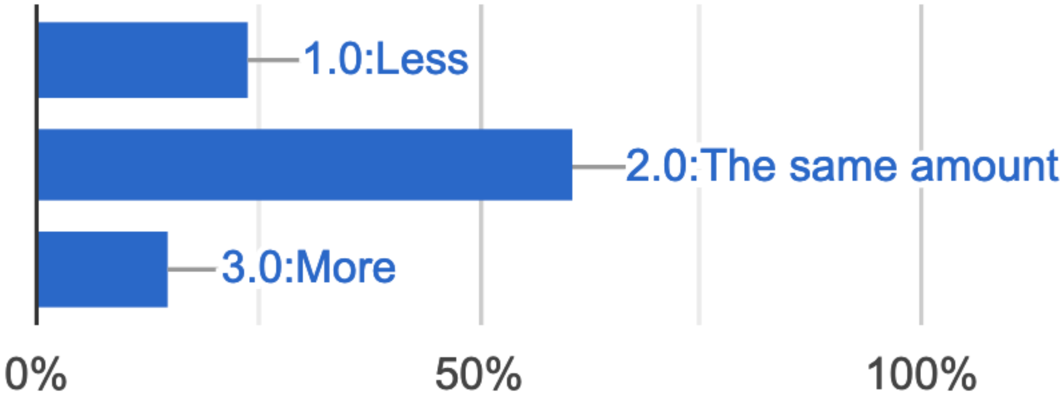
Prerequisites were introduced in 2022.

"Taking into account the number of ECs for this course (5 EC = 14 hours per week on average), I spent ….. hours on this course."

2021:



2022:

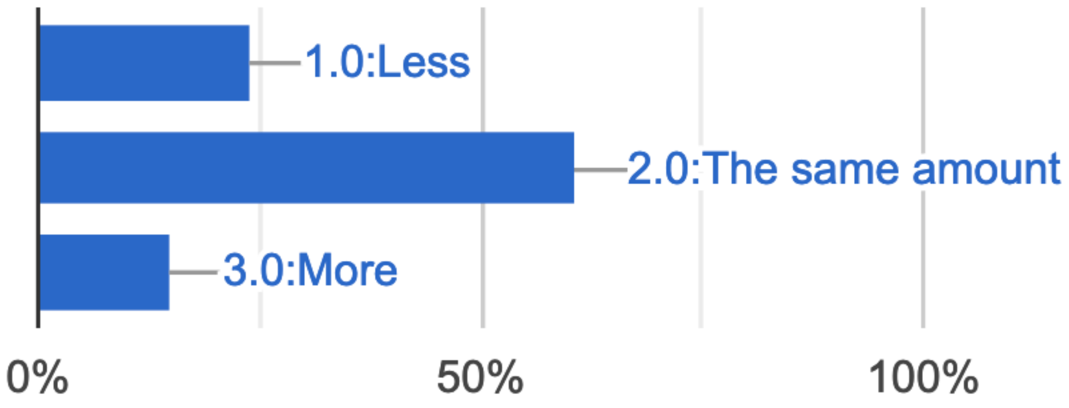


Prerequisites: student evaluation results

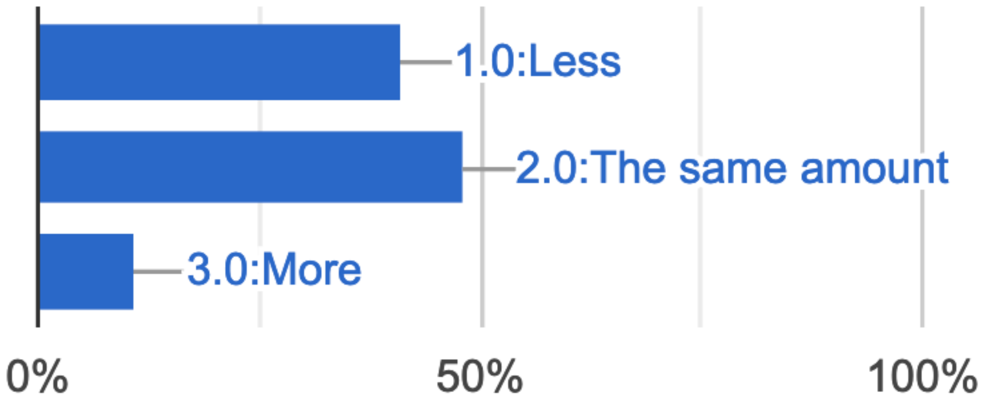
The course setup hardly changed between 2022 and 2023.

“Taking into account the number of ECs for this course (5 EC = 14 hours per week on average), I spent ….. hours on this course.”

2022:



2023:



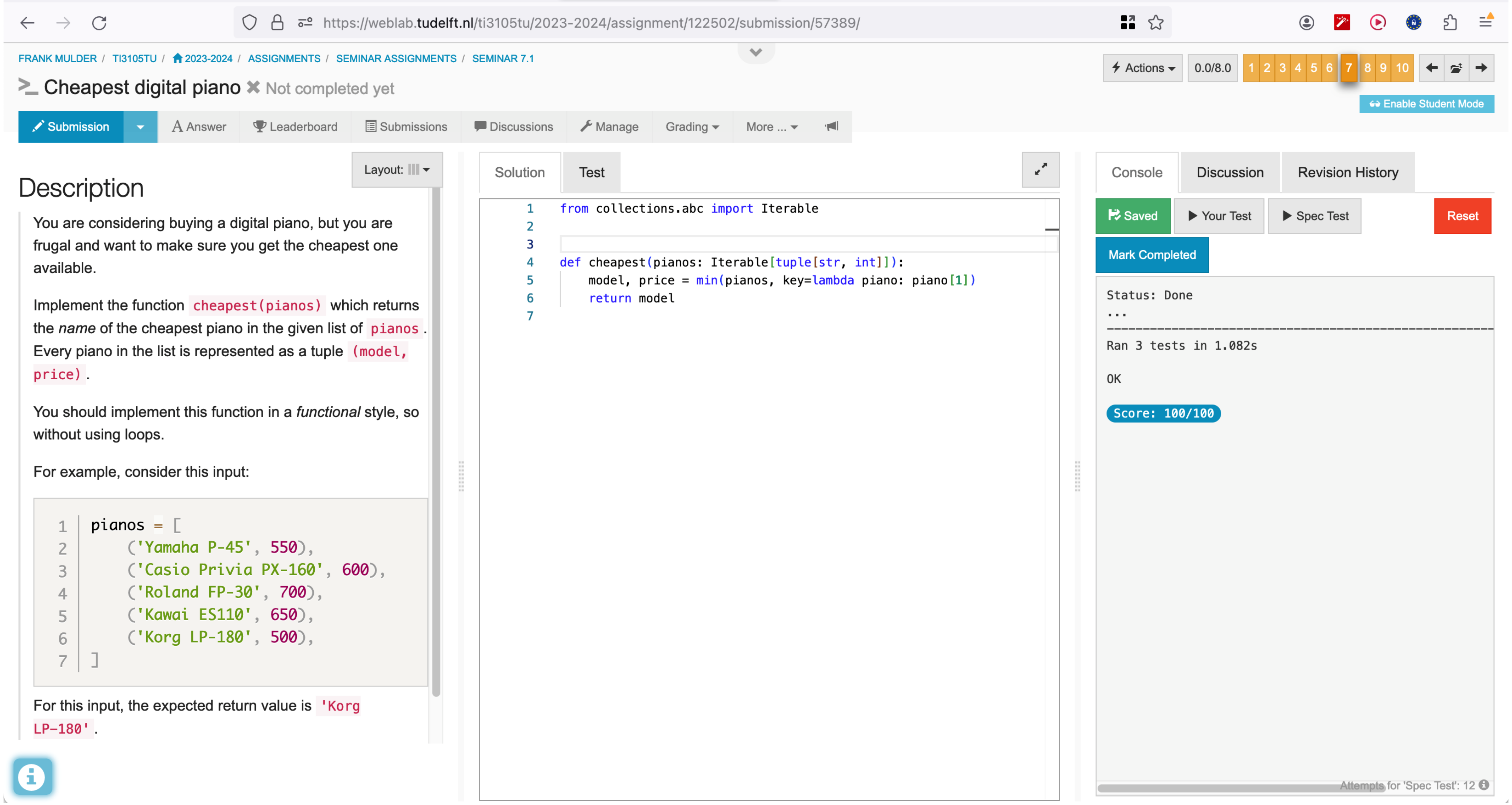
Topics

- Strings
- Lists
- File reading/writing
- Dictionaries
- Object-oriented programming (including operator overloading and properties)
- Structural pattern matching
- Functional programming (list comprehensions, higher-order functions)
- Iterators and generators
- Writing type hints

Activities

- Most of the students’ time during the course is spent solving programming problems.
- The exam also consists 100% of programming problems.

WebLab: online programming environment



Cheapest digital piano ✖ Not completed yet

Description

You are considering buying a digital piano, but you are frugal and want to make sure you get the cheapest one available.

Implement the function `cheapest(pianos)` which returns the *name* of the cheapest piano in the given list of `pianos`. Every piano in the list is represented as a tuple `(model, price)`.

You should implement this function in a *functional* style, so without using loops.

For example, consider this input:

```python
pianos = [
    ('Yamaha P-45', 550),
    ('Casio Privia PX-160', 600),
    ('Roland FP-30', 700),
    ('Kawai ES110', 650),
    ('Korg LP-180', 500),
]
```

For this input, the expected return value is `'Korg LP-180'`.

```python
from collections.abc import Iterable


def cheapest(pianos: Iterable[tuple[str, int]]):
    model, price = min(pianos, key=lambda piano: piano[1])
    return model

```

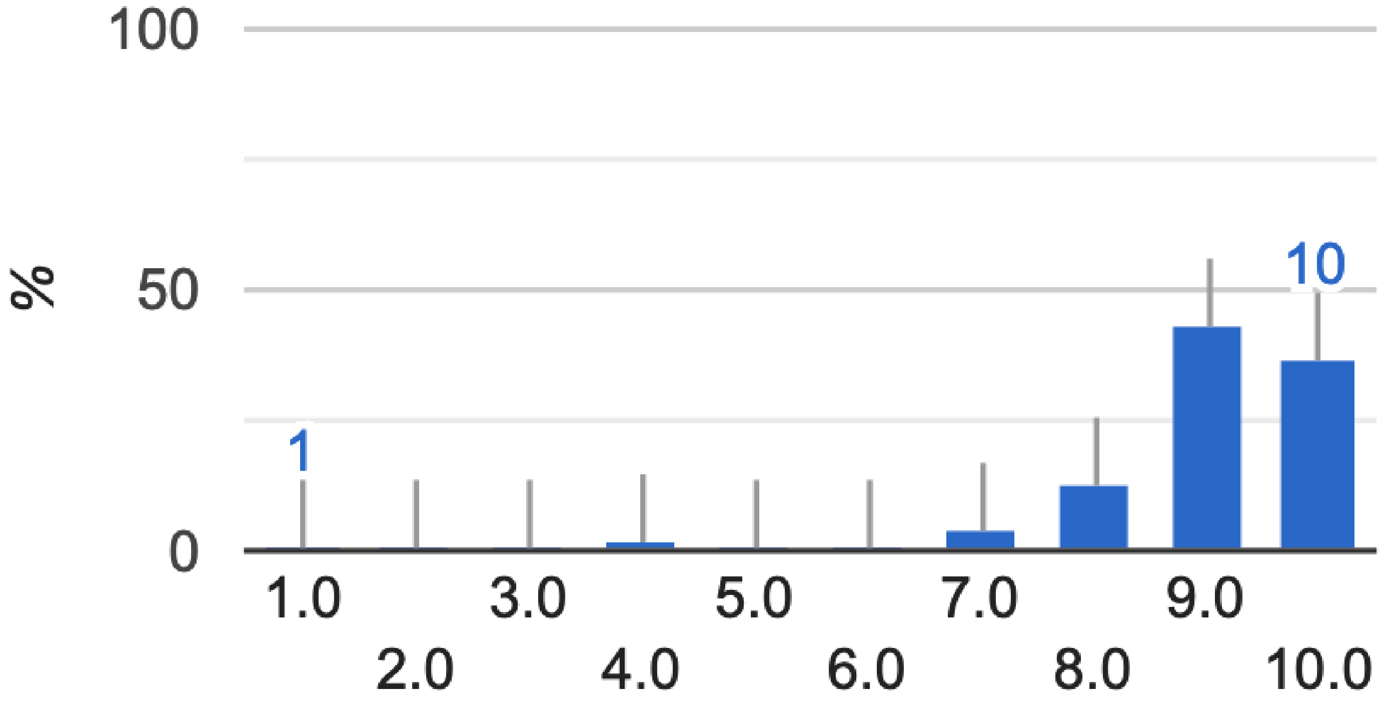
```
Status: Done
...
----------------------------------------------------------------------
Ran 3 tests in 1.082s

OK

Score: 100/100
```

Automated grading/marking everywhere (during the course and on the exam)

- Reasons:
  - To make it feasible to give feedback to 250 students
  - To ensure consistency and transparency etc.
  - Because it is possible in this subject without too many compromises
- Scores are visible to students during exam (!)
- Experiences:
  - Workload shifts from “grading after exam” to “carefully writing tests before the exam”.
  - Most students like it, but some students are annoyed when they lose all points for a question because of a typing mistake. (But they can see this during the exam and fix it.)
  - Things like “code quality” can only be tested to the extent that tools can judge it.

Overall student evaluation

“The overall grade I would give this course is: (1 = very poor; 6 = sufficient; 10 = excellent).”



Let’s have a chat!

- I’d love to hear about your course setups. ☺