Lecture I - Introduction

Programming with Python

Dr. Tobias Vlćek
Kühne Logistics University Hamburg - Fall 2026

About this Course

About me

- **Field:** Optimizing and simulating complex systems
- **Languages:** of choice: Julia, Python and Rust
- **Interest:** Modeling, Simulations, Machine Learning
- **Teaching:** OR, Algorithms, and Programming
- **Contact:** vlcek@beyondsimulations.com

. . .

> Tip
>
> I really appreciate active participation and interaction!

Course Outline

- **Part I:** Introduction to Programming with Python
- **Part II:** Data Science Tools with Python
- **Part III:** Programming Projects

. . .

Materials live in the KLU portal and are hosted at python.tobiasvlcek.com.

Passing the Course

- Pass/fail course, 100 points in total
- 5 in-class checkpoints × 12 points = 60 points
- Final project + presentation = 40 points
- You pass with 60 points and 75% attendance
- No make-ups for missed checkpoints, but the point math absorbs one miss
- Your project is done in pairs; solo works if the numbers don't come out even

Checkpoints

- Short, supervised exercises at the start of some sessions
- Solved in the browser, just like our lab notebooks
- ~6 small tasks each: write code, trace code, fix a bug, MCQs
- At the end you download your `.py` file and upload it to Moodle

- **Session II** has an ungraded 15-minute practice run (Checkpoint 0) before the lab

. . .

> Tip
>
> The weekly lab exercises rehearse exactly this routine. Do them, and the checkpoints will feel familiar.

How a session works

- **Lecture first:** new concepts, with short exercises on your phone or laptop
- **Then a break**
- **Then the lab:** the episode's notebook, in class, in your browser, with help in the room
- Done early? You're free to go. Not finished when the session ends? Keep the tab open on your laptop and finish at home

How to use AI

- We base our assessment on the KLU classification:
  - Level 1: Pause: Use of AI defined by the educator
- Part I (Sessions I–V) is AI-free: no Claude, ChatGPT, Mistral & Co.
- Your sanctioned helper is the **course chatbot** on the learning website
- It gives **hints**, not solutions, and guides your problem-solving
- AI tools are introduced (and then allowed) in Part II, from Session VI on

. . .

> Warning
>
> Learning the basics *without* AI first is what lets you judge AI output later.

How to use the Chatbot

- Just click the chatbot bubble on the website
- The chat will open and you can **ask your questions**
- It is programmed by us and uses Mistral AI as backend
- Ask your question as specific as possible
- This ensures enough **context for the model**
- We can see aggregated logs, but cannot identify you
- Please don't provide personal information

No setup needed today

- Everything in Part I runs in the browser: nothing to install
- The lab notebooks and exercises run on **marimo**: click a link, start coding
- Before **Session X** you install Python (via `uv`) and the Zed editor on your own laptop, with our guides; Session X then runs on your machine
- Until then: a laptop, a browser, and **one free account** (coming up) is all you need

How a notebook works

- A notebook is a page of **cells**; each cell holds a few lines of code
- **Run a cell**: click into it and press `Cmd/Ctrl+Enter` (or the ▶ button at its edge). The output appears right below it
- Change a cell and run it again: every cell that depends on it updates by itself
- A cell shows the value of its **last line**. An assignment like `x = 5` shows nothing, so the check under each exercise echoes **your result** next to the verdict
- **Save your work**: the notebook menu (top right) → *Download* → *Download Python code*. The tab remembers you until you close it; the download is forever

One account, today: GitHub

- Open github.com → **Sign up**. This takes **five minutes, and we do it right now in class**
- Use an e-mail you'll still read next year; your **KLU address** works well
- Pick a username you'd be happy to show a future employer in case you might continue programming in the future
- Already have one? Great, you're done

. . .

> **i** Note
>
> **Why now?** From Session VI you get Zed's paid editor plan for free as a student, and Zed only accepts GitHub accounts that are **at least 30 days old**. In Session X your project lives on GitHub. Nothing else to install until Session IX.

Episode 1: The Founding

A New Venture

This semester you are founding a campus food-delivery startup. Today it gets a name, a menu, and its first revenue calculation.

. . .

(Your co-founder Tobi has already spent the marketing budget on stickers.)

Variables & Types

What is a program?

- A **program** is a list of instructions the computer follows
- It runs **top to bottom**, one line at a time
- Writing a program *is* writing that list, in a language the computer understands

. . .

Today your startup's first program does three small things: record the founding data, do one calculation, and print a summary.

Printing output with `print()`

`print()` is a **function**: you hand it something, it shows it on screen.

```
# The line starting with # is a comment. Python ignores it
print("Welcome to your food-delivery startup!")
print("Stickers bought:", 1200)    # commas print several things, a space between
```

```
Welcome to your food-delivery startup!
Stickers bought: 1200
```

Variables: named values

- A **variable** is a name that points to a value
- Create one with `=`: name on the **left**, value on the **right**
- You don't declare a type; Python figures it out from the value

```
company_founded = 2026        # store a whole number under a name
print(company_founded)
```

```
2026
```

The founding data

Store the facts of the new company, then use them by name:

```
company_founded = 2026
first_employee = "Tobi"          # co-founder, but he put himself on the payroll first
sticker_budget = 300          # what's left after Tobi's spending spree

print(first_employee)
print(sticker_budget)
```

```
Tobi
300
```

Naming rules

- Must start with a **letter** or an underscore `_`
- Can contain letters, numbers and underscores
- **Case-sensitive**: `budget` and `Budget` are two different names
- Cannot be a reserved word (`for`, `if`, `def`, …)
- Good names are short **and** meaningful for humans

. . .

Question (we solve this together, out loud): Which of these are valid variable names?
`sticker_budget`, `1st_employee`, `firstEmployee`, `company-name`, `_tobi`

Four basic types

Every value has a **type**, which decides what you can do with the value:

- `str`: text, in quotes (`"Tobi"`, `"Falafel Wrap"`)
- `int`: whole numbers (`2026`, `300`)
- `float`: decimal numbers (`9.99`, `0.15`)
- `bool`: truth values (`True`, `False`)

Checking a type with `type()`

`type()` tells you the type of any value, and it's priceless when a bug surprises you:

```
print(type(2026))       # int
print(type("Tobi"))     # str
print(type(9.99))       # float
```

```
<class 'int'>
<class 'str'>
<class 'float'>
```

Predict: quotes around a number

Tobi typed the price **with quotes**. What does this print?

```
print(type("9.99"))
```

a) `<class 'float'>` b) `<class 'int'>` c) `<class 'str'>`

. . .

Predict first. Pick a letter, then I reveal the answer.

Answer: `type("9.99")`

c) `str`: the quotes make it text, however numeric it looks. Tobi's "price" can't be multiplied until it is converted to a number.

```
print(type("9.99"))
```

```
<class 'str'>
```

Your turn — 10 minutes

Open the exercise (scan the QR or type the link):

python.tobiasvlcek.com/notebooks/ex_01_a/

First **predict** what happens, then run it.

Numbers & Arithmetic

Tobi's 9.99 theory

Tobi's grand plan: **sell everything for 9.99**. To see whether that survives contact with arithmetic, we need Python's operators.

```
# margin = price - cost
print(9.99 - 7.40)
```

```
2.59
```

Arithmetic operators

```
print(9.99 + 1.50)   # addition
print(9.99 - 7.40)   # subtraction
print(9.99 * 3)      # multiplication
print(300 / 8)       # division, always gives a float
```

```
11.49
2.59
29.97
37.5
```

`//` and `%`: how many fit, what's left

- `//` **floor division**: how many whole times something fits
- `%` **modulo**: the remainder left over

```
# The 300 EUR budget, boxes at 7 EUR each
print(300 // 7)      # whole boxes we can buy
print(300 % 7)       # EUR left over
print(10 // 2.5)      # 4.0: floor division on floats floors, but returns
a float
```

```
42
6
4.0
```

Precedence: like on paper

`*` and `/` happen **before** `+` and `-`. Parentheses override that.

```
print(2 + 3 * 4)     # 14, not 20
print((2 + 3) * 4)   # 20
```

```
14
20
```

Predict: the sticker budget

Tobi buys 12 boxes at 25 EUR from the 300 EUR budget. What prints?

```
print(300 - 12 * 25)
```

a) `0` b) `7200` c) `Error`

. . .

Predict first. Pick a letter, then I reveal the answer.

Answer: `300 - 12 * 25`

a) `0`: `12 * 25 = 300` happens first, so the budget is wiped out. Reading left to right would give `7200`, but Python follows precedence, not reading order.

```
print(300 - 12 * 25)
```

```
0
```

Whole numbers vs decimals

- `int` + `int` stays an `int`; `/` **always** produces a `float`
- Mixing an `int` and a `float` gives a `float`
- Money is decimals, so margins are floats

```
margin = 9.99 - 7.40
print(margin)
print(type(margin))
```

```
2.59
<class 'float'>
```

`round()`: clean money

Division can leave a long tail of decimals. `round(value, 2)` keeps two. Cents.

```
# Split a 10 EUR order three ways
print(10 / 3)              # 3.3333333333333335, ugly
print(round(10 / 3, 2))    # 3.33, a proper amount
```

```
3.3333333333333335
3.33
```

Your turn — 10 minutes

Open the exercise (scan the QR or type the link):

python.tobiasvlcek.com/notebooks/ex_01_b/

First **predict** what happens, then run it.

Strings & f-strings

The first receipt line

A string is **text in quotes**. Single or double quotes both work, just be consistent.

```
item = "Miso Ramen"
print(item)
print('Tobi shouts "9.99!"')   # mix quotes to include a quote inside
```

```
Miso Ramen
Tobi shouts "9.99!"
```

Concatenation, and why it hurts

You can glue strings with `+`, but numbers must be converted first, and it gets clumsy fast:

```
qty = 2
item = "Miso Ramen"
total = 23.00
print(str(qty) + "x " + item + ": " + str(total) + " EUR")
```

```
2x Miso Ramen: 23.0 EUR
```

...

(`str(...)` turns a value into text, which is clumsy. We're about to see the better way.) Forget one `str()` and Python raises a `TypeError`.

f-strings: the clean way

Put an `f` before the quote and write values in `{ }`, and Python fills them in:

```
qty = 2
item = "Miso Ramen"
total = 23.00
print(f"{qty}x {item}: {total} EUR")
```

```
2x Miso Ramen: 23.0 EUR
```

...

No `str()`, no `+`. Just the sentence you want.

Formatting money with `:.2f`

Inside the braces, `:.2f` forces **two decimals**, exactly what a receipt needs:

```
total = 23.00
print(f"{total} EUR")            # 23.0 EUR: one decimal, looks broken on
a receipt
print(f"{total:.2f} EUR")      # 23.00 EUR, a proper price
```

```
23.0 EUR
23.00 EUR
```

. . .

Python stores `23.00` as the number `23.0`. Only *you* know it's money, and `:.2f` says so.

Multi-line receipts with `\n`

`\n` inside a string starts a new line: one string, several lines.

```
print(f"Falafel Wrap: 6.90 EUR\nPad Thai: 8.90 EUR")
```

```
Falafel Wrap: 6.90 EUR
Pad Thai: 8.90 EUR
```

Predict: braces or no braces?

What does each line print?

```
print(f"{2 * 3}")
print("2 * 3")
```

a) `6` and `6` b) `6` and `2 * 3` c) `2 * 3` and `2 * 3`

. . .

Predict first. Pick a letter, then I reveal the answer.

Answer: `f"{2 * 3}"` vs `"2 * 3"`

b) `6` and `2 * 3`: the f-string evaluates what's in the braces; plain quotes keep the text literal. The `f` and the `{ }` are what do the work.

```
print(f"{2 * 3}")
print("2 * 3")
```

```
6
2 * 3
```

Your turn — 10 minutes

Open the exercise (scan the QR or type the link):

python.tobiasvlcek.com/notebooks/ex_01_c/

First **predict** what happens, then run it.

To the Lab

After the break: the lab

- We do it **together in class** after a short break; whatever's left, you finish at home
- Head to the lab notebook: Episode 1 — The Founding
- You'll name the company, set prices, and put Tobi's 9.99 theory on trial
- It runs entirely in your browser: no setup, just click and code
- Didn't finish the **GitHub sign-up** in class? Do it tonight, the 30-day clock is ticking

. . .

> ! Important
>
> **Download your `.py` before you leave.** Closing the tab without downloading loses your work, and downloading is exactly how you hand in the checkpoints.

Wrap-up

Three things to remember

1. **Variables** name values, and every value has a **type** (`str`, `int`, `float`, `bool`)
2. Arithmetic follows **precedence**: `*` and `/` before `+` and `-`; `round(x, 2)` for money
3. **f-strings** with `{value:.2f}` turn numbers into clean receipt lines

. . .

> **Note**
>
> **Next time — Episode 2:** the city bans delivery after 22:00, and Tobi suggests "we just deliver yesterday's orders." We'll need **conditionals and loops** to survive it. Plus a 15-minute practice checkpoint before the lab, ungraded.

Literature

Books to start with

- Downey, A. B. (2024). Think Python: How to think like a computer scientist (Third edition). O'Reilly. Link to free online version
- Elter, S. (2021). Schrödinger programmiert Python: Das etwas andere Fachbuch (1. Auflage). Rheinwerk Verlag.

. . .

> **Note**
>
> Think Python is a great book to start with and is free online. Schrödinger programmiert Python is a playful German-language alternative with lots of examples.

. . .

For more, see the literature list of this course.