Lecture I - Introduction

Programming with Python

Dr. Tobias Vlćek

About this Course

About me

- Post-doctoral researcher from the University of Hamburg
- **Field:** Optimizing and simulating complex systems
- **Languages:** of choice: Julia, Python and Rust
- **Interest:** Modelling, 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

Participation

- Prequisite for course *Management Science* (Prof. Goel)
- Try actively participating in this course
- You will find it much (!) easier to follow Prof. Goel's course
- Materials will be provided in the KLU portal
- Slides are hosted at beyondsimulations.github.io/Introduction-to-Python

Teaching

- **Lecture:** Presentation of tools and concepts, based on small examples and code snippets
- **Tutorial:** Hands-on examples to be solved in groups
- **Difficulty:** Difficult at first, but gradually easier

Passing the Course

- Pass/fail course
- 75% attendance required for passing the course
- 2 assignments and 1 little project
- You will be given programming exercises to solve with Python
- You can group up (3 students) and work together
- Each student group submits one solution together

Solution

- Provide a code solution to the problem (`.py` files)
- Code files need to be executable
- Detailed explanations of your code should be provided
- Use comments or docstrings in your code
- Provide a general (verbal) introduction to each problem

. . .

Tip

I'd encourage you to start and submit your solution early

Difficulty of the Course

- We'll cover the basics of programming (in Python) at first
- This is similar to learning a **new foreign language**
- First, you have to get used to the language and learn first words
- **Later**, you'll be able to apply the language and see results
- Similar to learning a language: *Practice, practice, practice!*

What to expect

- Some **investment** in the beginning to see the **return** later
- You can ask questions and get support anytime
- After completing the course, you will be able to read code
- **and** write your own program using Python
- **That's quite something!**

Goals of the Course

- Essential concepts and tools of modern programming
- Automated solutions for recurrent tasks
- Algorithm-based solutions of complex problems
- **Usage of AI** in a specific context

Python as Language

- **Origins**: Conceived in late 1980s as a teaching and scripting language

- **Simple Syntax**: Python's syntax is straightforward and easy to learn
- **Versatility**: Used in web development, data analysis, artificial intelligence, and more
- **Community Support**: A large community of users and extensive documentation

Help from AI

- You are allowed to use AI (GitHub Copilot, ChatGPT, LLama3 …)
- These new tools are really powerful for learning Python!
- They can help you a lot to get started with programming

. . .

⚠ Warning

But you should *not* simply use them to *replace* your learning.

Why learn programming?

Analytics

Photo by Choong Deng Xiang on Unsplash

Research

Photo by National Cancer Institute on Unsplash

Visualization

Photo by Clay Banks on Unsplash

Finance

Photo by Ishant Mishra on Unsplash

Logistics

Photo by Denys Nevozhai on Unsplash

How to learn programming

My Recommendation

1. Be present: Attend the lecture and participate
2. Put in some work: Repeat lecture notes and try to understand the examples yourself
3. Do coding: Run code examples on your own, play around, *google/find help*, modify, and solve problems on your own

. . .

Tip

Great resources to start are books and small challenges. In my opinion both are much more helpful than watching videos! You can find a list of book recommendations at the end of the lecture. Small challenges to solve can for example be found on Codewars.

Don't give up!

- Programming is **problem solving**, don't get **frustrated** too easily at the start!
- Learn something new: Expect to **stretch** your comfort zone

Learning Path

- The learning path can be quite steep!
- First of all help each other!
- Try to find help in lecture materials and books, the Python documentation, and online (e.g. Google, ChatGPT, StackOverflow, …)
- In case you get frustrated with programming, read the following helpful blog post about the challenges on medium.com

Errors

In case you find errors and typos in the lecture notes, please report them in the following form: https://tally.so/r/w7oapa

Setting up Python

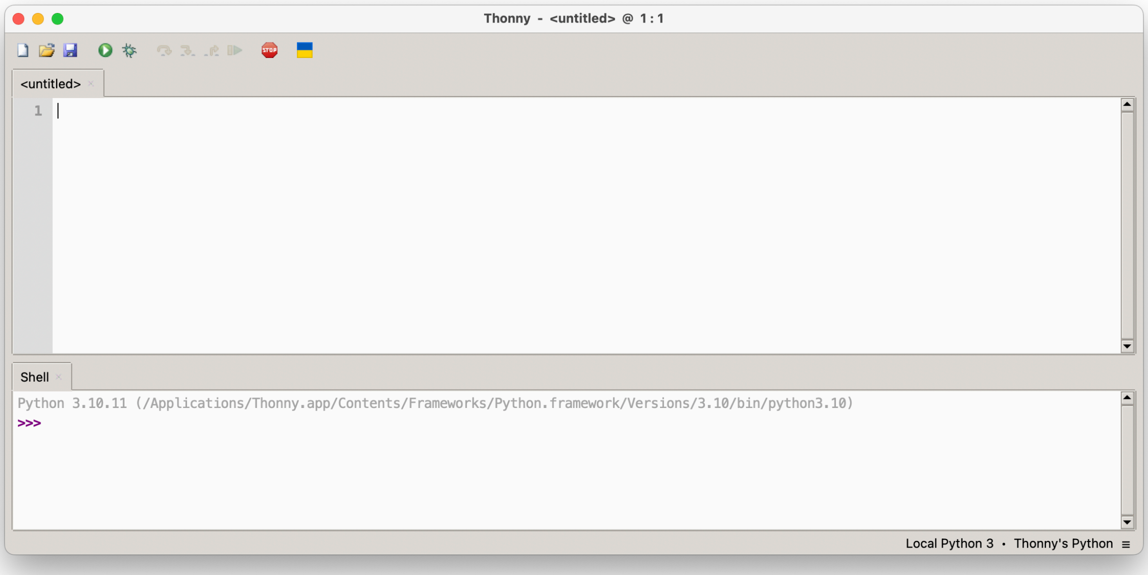
Install Python

- You could download it from the Python website or with Anaconda
- **But I would recommend we start by installing Thonny**
- It is an open source IDE that runs on Windows, Linux and Mac
- It comes with a built-in Python interpreter and package management!

What is an IDE?

- An **IDE** (Integrated Development Environment) is an application
- It allows you to write, run and debug code scripts
- Thonny is an IDE specifically for Python and aimed at beginners
- It does not use the latest Python and has the most features
- **But it is easy to use as beginner!**
- Other IDEs include for example PyCharm from JetBrains or Visual Studio Code from Microsoft

Thonny



```
Python 3.10.11 (/Applications/Thonny.app/Contents/Frameworks/Python.framework/Versions/3.10/bin/python3.10)
>>>
```

Figure 1: First start of Thonny

Python on iPads

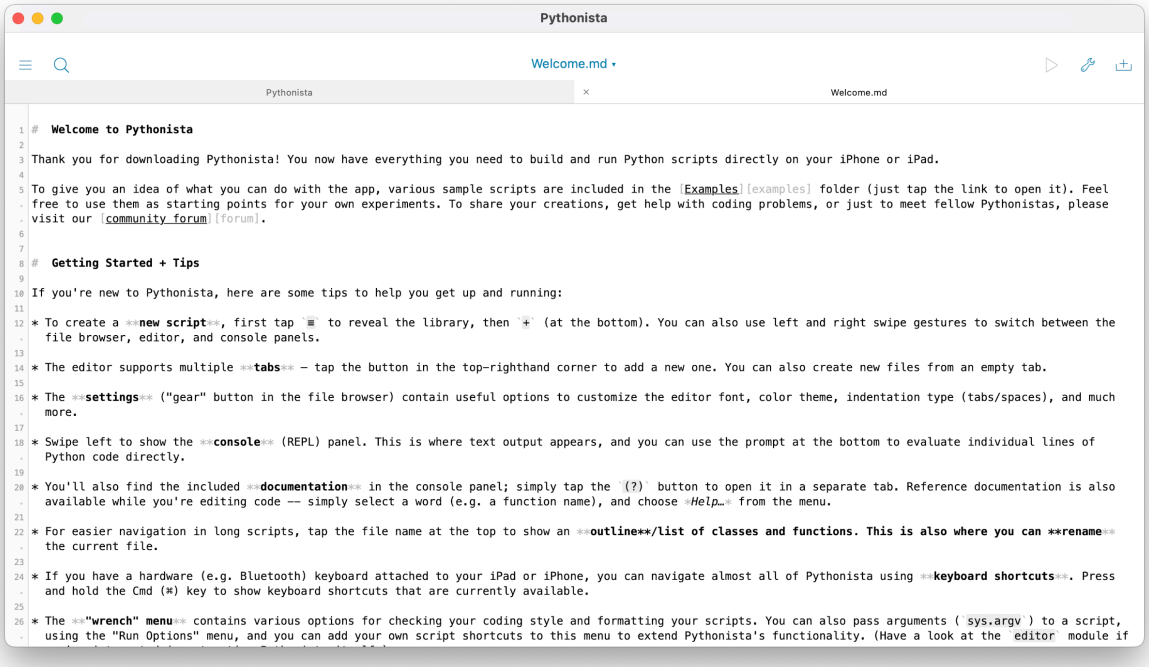
- Although you can run Python scripts from on your iPad, it is not recommended for the course
- Nonetheless, if you have no other option, you could use Pythonista
- It works locally on your iPad and can run most Python scripts

. . .

Caution

Not all packages available in Python are available in Pythonista, thus you might need a computer to solve certain problems.

First start of Pythonista



```
#  Welcome to Pythonista

Thank you for downloading Pythonista! You now have everything you need to build and run Python scripts directly on your iPhone or iPad.

To give you an idea of what you can do with the app, various sample scripts are included in the [Examples][examples] folder (just tap the link to open it). Feel
free to use them as starting points for your own experiments. To share your creations, get help with coding problems, or just to meet fellow Pythonistas, please
visit our [community forum][forum].


#  Getting Started + Tips

If you're new to Pythonista, here are some tips to help you get up and running:

* To create a **new script**, first tap `≡` to reveal the library, then `+` (at the bottom). You can also use left and right swipe gestures to switch between the
  file browser, editor, and console panels.

* The editor supports multiple **tabs** — tap the button in the top-righthand corner to add a new one. You can also create new files from an empty tab.

* The **settings** ("gear" button in the file browser) contain useful options to customize the editor font, color theme, indentation type (tabs/spaces), and much
  more.

* Swipe left to show the **console** (REPL) panel. This is where text output appears, and you can use the prompt at the bottom to evaluate individual lines of
  Python code directly.

* You'll also find the included **documentation** in the console panel; simply tap the `(?)` button to open it in a separate tab. Reference documentation is also
  available while you're editing code -- simply select a word (e.g. a function name), and choose *Help…* from the menu.

* For easier navigation in long scripts, tap the file name at the top to show an **outline**/list of classes and functions. This is also where you can **rename**
  the current file.

* If you have a hardware (e.g. Bluetooth) keyboard attached to your iPad or iPhone, you can navigate almost all of Pythonista using **keyboard shortcuts**. Press
  and hold the Cmd (⌘) key to show keyboard shortcuts that are currently available.

* The **"wrench" menu** contains various options for checking your coding style and formatting your scripts. You can also pass arguments (`sys.argv`) to a script,
  using the "Run Options" menu, and you can add your own script shortcuts to this menu to extend Pythonista's functionality. (Have a look at the `editor` module if
```

Figure 2: First start of Pythonista

Your first code

Hello, World!

Your Task: Create a directory for the course and create a new file called `hello_world.py` with the following code and save it:

```python
# This is a comment in Python
print("Hello, World!")
```

. . .

Run the code with the green 'run' button at the top or by going to the line and pressing Shift+Enter:

. . .

```python
# This is a comment in Python
print("Hello, World!")
```

```
Hello, World!
```

. . .

Note

"Hello world" is a classic example to start with. It is often used as a test to check if your computer is working properly and that you have installed the necessary software.

Hello, World in a Message

Your Task: Change the code in your `hello_world.py` file. Assign the string `"Hello, World!"` to a variable called `message` and print the variable.

. . .

- Use the equals sign (=)
- Variable name goes on the left
- Value to be assigned goes on the right

. . .

```python
# Here we assign the string "Hello, World!" to variable message and print it
message = "Hello, World!"
print(message)
```

```
Hello, World!
```

Hello, World in Parentheses

We can also mix `"` and `'` in a string. We just have to be consistent:

```python
# This code works
message = 'I shout "Hello, World!"'
print(message)
```

```
I shout "Hello, World!"
```

...

```python
# This code does not!
message = 'I shout 'Hello, World!""
print(message)
```

...

Try it yourself! What does happen, if you try to run this code?

First Errors

SyntaxError: invalid syntax

- The code is not valid Python syntax
- This is likely the most common error that you will encounter!
- It happens when you make a mistake in your code, e.g., using an illegal character, missing a colon, parentheses or wrong quotations
- You can fix this by correcting the code and re-running
- **In the course of the lecture you will encounter many more errors!**

Program

What is a Program?

- A sequence of instructions telling a computer what to do
- Written in a programming language the computer can understand
- Basic operations in most languages:
  - **Input**: Get data from keyboard, file, network, sensors, etc.
  - **Output**: Display data on screen, save to file, send over network, etc.
  - **Processing**: Perform calculations, analyze data, make decisions, find patterns, etc.

Key concepts

- Key concepts in most languages:
  - **Variables**: Store and manipulate data
  - **Conditional execution**: Check conditions and execute accordingly
  - **Loops**: Perform actions repeatedly, often with variations
  - **Functions**: Group instructions for reusability

How Python executes code

- Python is an **interpreted language**
- The source code is executed line by line
- The interpreter checks the syntax and executes the code
- This is in contrast to compiled languages, where the code is compiled into machine code before execution

Hello again, World!

Let's go back to our first program:

```python
# Our first program
message = "Hello, World!"
print(message)
```

. . .

- **Comment**: In the first line we define a comment with `#`
  - It is not executed but used to explain what code does
- **Variable**: In the second line we define a variable `message`
  - It points to a string that contains the text "Hello, World!"
- **Function**: In the third line we call a function `print`

    - It prints out whatever is stored in `message`

Don’t worry!

- Already confused?
- Don’t worry about it for now!
- We’ll learn more about variables and functions later

Python’s Syntax

The Zen of Python

- Python’s name originally comes from Monty Python
- Style is based on a philosophy called Zen of Python: A collection of 19 statements intended to communicate general principles

```
# Try this code in Python to see the Zen of Python
import this
```

Variables

- A variable in Python is a name that points to a value
- Created by using the assignment operator =
- Python does not require a declaration of variable types before

```
a = 2 # Variable a assigned the value 2
b = "Time" # Variable b assigned the value "Time"
c = print # Variable c assigned the print function
c(b) # Now we can call the print function with c
```

```
Time
```

. . .

But there are certain rules to variable names!

Variable Naming Conventions

- Must start with a letter or underscore _
- Can contain letters, numbers and underscores
- Names are case sensitive, e.g., `a` and `A` are different!
- Cannot be a reserved word, e.g., `for`, `if`, `def`, etc
- Good names are short and meaningful for humans!

. . .

Question: Which of the following fulfill these conditions?
a, _duration, 1x, time_left, 1_minute, oneWorld, xy4792

Functions

- Functions are named blocks of code
- Can take arguments `function([arguments])`
- Can return results or `None`

...

```python
# Print is such a function
print("Hello, World!") # It takes an argument and prints it to the console
print("Hello","World!", sep=", ") # It can also take multiple arguments
```

```
Hello, World!
Hello, World!
```

Note

We will cover functions in more detail **later** in the course.

Values and Types

What are Values and Types?

- **Value**: Fundamental thing that a program manipulates
  - In Python, values are either numbers or strings
- **Type**: Type of a value
  - Determines what operations can be performed on it
  - `type()` is a function that returns the type of a value
  - It takes one argument (a value) and returns its type as string

Strings

Back to our example of "Hello, World!"

```python
# We define the variable message and assign it the value "Hello, World!"
message = "Hello, World!"

# We save its type in another variable called message_type
message_type = type(message)

# We print the value of our new variable
print(f"{message} is a {message_type}")
```

```
Hello, World! is a <class 'str'>
```

. . .

Result: "Hello, World" is a string - in short 'str'.

. . .

But what about the f"?

Formated Strings

- **f-strings** are strings that start with `f`
- They contain **expressions** (here variables) in braces
- They are evaluated at run time and inserted into the string
- This is called **interpolation**

. . .

Note

In older code bases, f strings were not available. Here, interpolation could be done as shown below with `print()` and `.format()`. But this method is less concise and arguably less readable.

```python
print("{} is a {}".format(message, message_type))
```

```
Hello, World! is a <class 'str'>
```

Expressions

- Produce a value when evaluated
- Can be used as part of larger expressions or statements
- Statements are expressions that don't produce a value
- Examples: arithmetic operations, function calls, variables

```python
print(1 + 2) # Expression 1 + 2 produces the value 3
```

```
3
```

```python
print("The result is", 1 + 2) # Expression embedded in a string
```

```
The result is 3
```

```python
x = 1 # Statement that assigns the value 3 to x
y = x + 2 # Expression on the right side assigned to a variable y
print(f"Again, the result is {y}")
```

```
Again, the result is 3
```

What is a String?

- Remember: "Hello, World" is a string - in short 'str'
- A string is a sequence of characters enclosed in quotes
- Examples: `"Hello"`, `'World'`, `"123"`, `'1World23'`

```python
hello = "Hello"
world = 'World!'
print(hello,world,sep=", ") # We can specify the separator with the argument sep
```

```
Hello, World!
```

...

Note

Strings are immutable, we can't change single characters in them once they are created.

String Operations

But we can also do much more with strings!
E.g. string concatenation, indexing, slicing, length, repeat, etc.

```python
two_strings = "Hello" + ", " + "World!" # String concatenation
print(two_strings)
```

```
Hello, World!
```

. . .

```python
print(two_strings[0]) # Indexing starts at zero!
```

```
H
```

. . .

```python
print(two_strings[0:4]) # To slice we need to specify the start and end index (excluded)
```

```
Hell
```

. . .

```python
print(len(two_strings)) # With len we can find the length of our string
```

```
13
```

. . .

```python
print("--x--"*3) # We can also repeat strings
```

```
--x----x----x--
```

Booleans

- Booleans represent two values: `True` and `False`
- Internally they are represented as `1` and `0`, respectively
- They are used for logical operations and control flow
- E.g.: `if`, `while`, `for`, `elif`, ‘else

. . .

```python
x = True
y = False
print(x)
print(type(y))
```

```
True
<class 'bool'>
```

. . .

> More on them in our next lecture!

Integers and Floats

- Integers are whole numbers, e.g.: `1`, `-3`, `0` or `100`
- Floats are decimal numbers, e.g.: `2.5`, `-4.789123`, `0.0` or 1.234e2
- Bit size does not have to be specified (e.g.: 64 bits) in Python

. . .

```python
x = 1
y = 1.2864e2
print(f"{x} is of type {type(x)}")
print(f"{y} is of type {type(y).__name__}")
```

```
1 is of type <class 'int'>
128.64 is of type float
```

...

Warning

The interpreter will automatically convert booleans to integers to floats when necessary, **but not the other way around!**

First Functions and Operators

Arithmetic operators

...

```python
# We can separate multiple operations in one line with semicolons
addition = 1 + 2;             print(f"Result: addition is {addition}")
substraction = 1 - 2;         print(f"Result: substraction is {substraction}")
multiplication = 3 * 4;       print(f"Result: multiplication is {multiplication}")
division = 7 / 4;             print(f"Result: division is {division}")
floor_division = 7 // 4;      print(f"Result: floor_division is {floor_division}")
exponentiation = 9 ** 0.5;    print(f"Result: exponentiation is {exponentiation}")
modulo = 10 % 3;              print(f"Result: modulo is {modulo}")
```

```
Result: addition is 3
Result: substraction is -1
Result: multiplication is 12
Result: division is 1.75
Result: floor_division is 1
Result: exponentiation is 3.0
Result: modulo is 1
```

...

Note

Note, how the integers in the division are converted to floats before the division is performed.

Precedence

- The operators are the same as in most other languages
- They can be combined with each other, and with variables
- Normal rules of precedence apply

...

```python
# Operator precedence works as on paper
combined_operation = 2 + 3 * 4
print(f"2 + 3 * 4 = {combined_operation}")
```

```
2 + 3 * 4 = 14
```

...

```
# Parentheses change precedence as expected
parentheses_operation = (2 + 3) * 4
print(f"(2 + 3) * 4 = {parentheses_operation}")
```

```
(2 + 3) * 4 = 20
```

The input() Function

- Used to get user input as string from the console
- Syntax: `input([userprompt])`
- Displays optional prompt and waits for user input

. . .

```
name = input("What's your name? ")
print(f"Hello, {name}!")
```

. . .

❗ Important

The function always returns the input as **string**!

. . .

> Try it yourself!

Type Conversion

Use type conversion for other data types

1. Integer: `int(input())`
2. Float: `float(input())`
3. Boolean: `bool(input())`
4. String: `str(input())`

. . .

```
# Converting to Integer
age = int(input("Enter your age: "))
next_year = age + 1
print(f"Next year, you'll be {next_year}")
```

. . .

```
# Converting to Float
height = float(input("Enter your height in meters: "))
height_in_cm = height * 100
print(f"Your height in centimeters: {height_in_cm}")
```

The round() Function

```python
# Hence, we can use the int() function to convert a float into an int
soon_int = 1.789
print(f"{soon_int} converted to {int(soon_int)} of type {type(int(soon_int))}")
```

```
1.789 converted to 1 of type <class 'int'>
```

...

```python
# We can also use `round()` to round a float to an int
soon_int = 1.789
print(f"{soon_int} converted to {round(soon_int)} of type {type(round(soon_int))}")
```

```
1.789 converted to 2 of type <class 'int'>
```

...

```python
# Or to a float with a certain number of decimals
no_int = 1.789
print(f"{no_int} converted to {round(no_int,1)} of type {type(round(no_int,1))}")
```

```
1.789 converted to 1.8 of type <class 'float'>
```

...

Note

And that's it for todays lecture!
We now have covered the basics on the Python syntax, variables, and data types.

Literature

Interesting Books to start

- 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. It's available online for free here. Schrödinger Programmiert Python is a great alternative for German students, as it is a very playful introduction to programming with lots of examples.

. . .

For more interesting literature to learn more about Python, take a look at the literature list of this course.