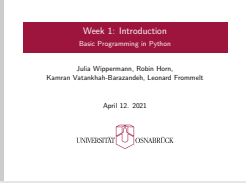
Week 1: Introduction

Basic Programming in Python

Julia Wippermann, Robin Horn,
Kamran Vatankhah-Barazandeh, Leonard Frommelt

April 12. 2021

UNIVERSITÄT OSNABRÜCK



1 About this course
- Who we are
- Who should attend this Course
- Schedule
- Structure

2 What is programming?
- Why Coxies need Programming
- Algorithms
- Formalizing Algorithms
- Hierarchy of Languages

3 Programming with Python
- Why Python?
- The Python Shell
- Using the Terminal
- Python Scripts

Who we are

1. Julia Wippermann (jwippermann@uos.de):
   2nd Semester Info Master / Bachelor Lehramt
2. Robin Horn (rhorn@uos.de):
   6th Semester CogSci
3. Kamran Vatankhah-Barazandeh (kvatankhahba@uos.de):
   6th Semester CogSci
4. Leonard Frommelt (lfrommelt@uos.de):
   10th Semester CogSci

Who we are
1. Julia Wippermann (jwippermann@uos.de): 2nd Semester Info Master / Bachelor Lehramt
2. Robin Horn (rhorn@uos.de): 6th Semester CogSci
3. Kamran Vatankhah-Barazandeh (kvatankhahba@uos.de): 6th Semester CogSci
4. Leonard Frommelt (lfrommelt@uos.de): 10th Semester CogSci

Who this course is for

You are in the right course if...

- **a** You are a master student coming from a non-technical discipline
- **b** You have little to no experience with programming
- **c** You felt a little overwhelmed by Informatik A (Algorithmen & Datenstrukturen) and would like to repeat the core principles of programming with another language

Who this course is for

You are in the right course if...

- a You are a master student coming from a non-technical discipline
- b You have little to no experience with programming
- c You felt a little overwhelmed by Informatik A (Algorithmen & Datenstrukturen) and would like to repeat the core principles of programming with another language

Who this course is for

You are NOT in the right course if...

a. Informatik A / AuD was a piece of cake for you and you would just like to learn another language
   → Scientific Programming in Python
b. You already know to program in Python or another language
   → You will not learn anything in this class
c. You have a specific application area that you want to learn about in detail
   → Specialized courses (CV, CL, ML etc.)

Who this course is for
You are NOT in the right course if...
a. Informatik A / AuD was a piece of cake for you and you would just like to learn another language
→ Scientific Programming in Python
b. You already know to program in Python or another language
→ You will not learn anything in this class
c. You have a specific application area that you want to learn about in detail
→ Specialized courses (CV, CL, ML etc.)

Tentative Schedule

1. Hello World
2. Variables & Assignments
3. Control Structures
4. Data Structures
5. Strings & Formatting
6. Input & Output
7. Debugging & Good Practices
8. Built-In Packages
9. Object-Oriented Programming

→ More lectures on external packages → Working on projects

Tentative Schedule
1. Hello World
2. Variables & Assignments
3. Control Structures
4. Data Structures
5. Strings & Formatting
6. Input & Output
7. Debugging & Good Practices
8. Built-In Packages
9. Object-Oriented Programming
→ More lectures on external packages → Working on projects

Structure

1. Lecture:
   - Uploaded in Courseware on Monday 8am
2. Coding Support
   - Live-Sessions for questions and help with the homework
   - Each Thu 16.00-18.00 and Mo 12.00-14.00
   - BBB (StudIP → Meetings)
3. Homework
   - Uploaded under Files&Vips on Monday 8am
   - Hand in until next Tuesday 23:59:59 via StudIP → Vips
   - → You have 1.5 weeks to work on it

Structure

1. Lecture:
   - Uploaded in Courseware on Monday 8am
2. Coding Support
   - Live-Sessions for questions and help with the homework
   - Each Thu 16.00-18.00 and Mo 12.00-14.00
   - BBB (StudIP → Meetings)
3. Homework
   - Uploaded under Files&Vips on Monday 8am
   - Hand in until next Tuesday 23:59:59 via StudIP → Vips
   - → You have 1.5 weeks to work on it

Homework and Grading

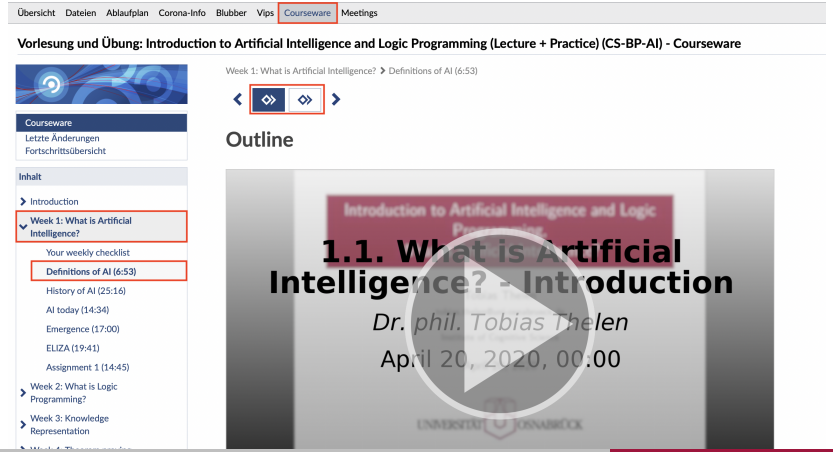
Homework Regulations

1. One homework sheet per week (12 in total)
2. Sheet submission in groups of 2-3 via Vips on StudIP
3. You need 50% of the points to pass a sheet
4. You have to pass **10 out of 12** homeworks to pass the course
5. Grading will be optional, likely as some kind of final project / assignment. Information on this will follow.

Homework and Grading
Homework Regulations
1. One homework sheet per week (12 in total)
2. Sheet submission in groups of 2-3 via Vips on StudIP
3. You need 50% of the points to pass a sheet
4. You have to pass 10 out of 12 homeworks to pass the course
5. Grading will be optional, likely as some kind of final project / assignment. Information on this will follow.

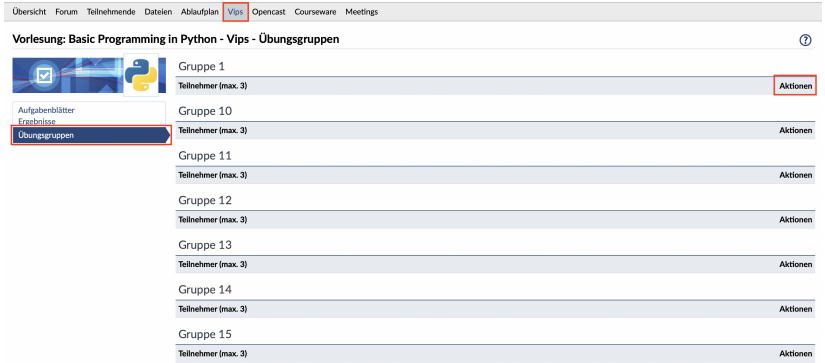
Courseware

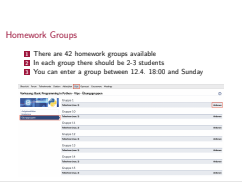
This is where you find Lectures



Homework Groups

1. There are 42 homework groups available
2. In each group there should be 2-3 students
3. You can enter a group between 12.4. 18:00 and Sunday





Why do we want to know Programming?

- **Analyzing/Visualizing Data**
  Data preprocessing, statistical analysis (anything from simple mean to ANOVA or PCA), plotting of graphs
- **Machine Learning**
  Artificial Neural Networks, Reinforcement learning, Computer Vision, etc...
- **Make life easier** Automatize tasks, python as programmable calculator, extract information from weird files
- **And lots more...**
  There will be a teaser for python use cases in our first meeting.

Why do we want to know Programming?
- Analyzing/Visualizing Data
  Data preprocessing, statistical analysis (anything from simple mean to ANOVA or PCA), plotting of graphs
- Machine Learning
  Artificial Neural Networks, Reinforcement learning, Computer Vision, etc...
- Make life easier Automatize tasks, python as programmable calculator, extract information from weird files
- And lots more...
  There will be a teaser for python use cases in our first meeting.

- live/uploaded demo

What is an Algorithm?

Definition

[...] an *Algorithm* is an unambiguous specification of how to solve a class of problems.[1]

[1] Wikipedia contributors. *Algorithm — Wikipedia, The Free Encyclopedia*.

What is an Algorithm?
Definition
[...] an *Algorithm* is an unambiguous specification of how to solve a class of problems.[1]
[1] Wikipedia contributors. *Algorithm — Wikipedia, The Free Encyclopedia*. [Online; accessed 24-February-2018]. 2018. URL:

- *Problem-specific* is fairly loosely used here: If it only solves **one** specific problem, we might as well just memorize the solution. But of course, we cannot have a general algorithm for **all** problems. Good classes of problems are somewhere inbetween, like sorting an **arbitrary list of numbers** in ascending order.

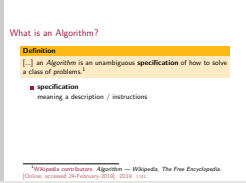
What is an Algorithm?

Definition

[...] an *Algorithm* is an unambiguous **specification** of how to solve a class of problems.[1]

- **specification**
  meaning a description / instructions

[1] Wikipedia contributors. *Algorithm — Wikipedia, The Free Encyclopedia.*



- *Problem-specific* is fairly loosely used here: If it only solves **one** specific problem, we might as well just memorize the solution. But of course, we cannot have a general algorithm for **all** problems. Good classes of problems are somewhere inbetween, like sorting an **arbitrary list of numbers** in ascending order.

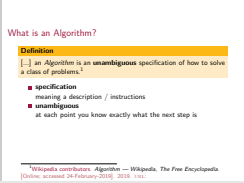
What is an Algorithm?

Definition

[...] an *Algorithm* is an **unambiguous** specification of how to solve a class of problems.[1]

- **specification**
  meaning a description / instructions
- **unambiguous**
  at each point you know exactly what the next step is

[1] Wikipedia contributors. *Algorithm — Wikipedia, The Free Encyclopedia.*



- *Problem-specific* is fairly loosely used here: If it only solves **one** specific problem, we might as well just memorize the solution. But of course, we cannot have a general algorithm for **all** problems. Good classes of problems are somewhere inbetween, like sorting an **arbitrary list of numbers** in ascending order.

What is an Algorithm?

Definition

[...] an *Algorithm* is an unambiguous specification of how to solve a **class of problems**.[1]

- **specification**
  meaning a description / instructions
- **unambiguous**
  at each point you know exactly what the next step is
- **problem-specific**
  an algorithm for sheering sheep won't help milking cows

[1] Wikipedia contributors. *Algorithm — Wikipedia, The Free Encyclopedia*.

What is an Algorithm?
Definition
[...] an Algorithm is an unambiguous specification of how to solve a class of problems.[1]
specification
meaning a description / instructions
unambiguous
at each point you know exactly what the next step is
problem-specific
an algorithm for sheering sheep won't help milking cows
[1] Wikipedia contributors. Algorithm — Wikipedia, The Free Encyclopedia. [Online; accessed 24-February-2018]. 2018. URL:

- *Problem-specific* is fairly loosely used here: If it only solves **one** specific problem, we might as well just memorize the solution. But of course, we cannot have a general algorithm for **all** problems. Good classes of problems are somewhere inbetween, like sorting an **arbitrary list of numbers** in ascending order.

What is an Algorithm?

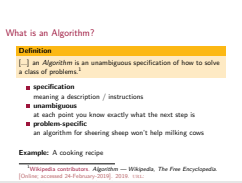
Definition

[...] an *Algorithm* is an unambiguous specification of how to solve a class of problems.[1]

- **specification**
  meaning a description / instructions
- **unambiguous**
  at each point you know exactly what the next step is
- **problem-specific**
  an algorithm for sheering sheep won't help milking cows

Example: A cooking recipe

[1] Wikipedia contributors. *Algorithm — Wikipedia, The Free Encyclopedia.*



- *Problem-specific* is fairly loosely used here: If it only solves **one** specific problem, we might as well just memorize the solution. But of course, we cannot have a general algorithm for **all** problems. Good classes of problems are somewhere inbetween, like sorting an **arbitrary list of numbers** in ascending order.

Pseudocode

We need a way of writing down algorithms!

Pseudocode
We need a way of writing down algorithms!

- there are no strict rules for writing pseudocode, but the idea is that you can tell exactly where an instruction ends, which instructions belong to a loop and which don't (indentation, brackets), etc.
- pseudocode is mostly used to communicate ideas for algorithms between humans, computers cannot understand pseudocode
- Do try to write down some pseudocode for your morning routine or the process of charging up your campus card!

Pseudocode

We need a way of writing down algorithms!

Example: Baking a Cake

```
start: gather all ingredients
REPEAT
        add the next ingredient to the bowl
UNTIL all ingredients are used
stir dough thoroughly
put dough into oven at 200°C
wait 50 minutes
REPEAT
        bake for another minute
UNTIL cake looks good
IF cake tastes bad GOTO start
```


Pseudocode
We need a way of writing down algorithms!
Example: Baking a Cake
start: gather all ingredients
REPEAT
add the next ingredient to the bowl
UNTIL all ingredients are used
stir dough thoroughly
put dough into oven at 200°C
wait 50 minutes
REPEAT
bake for another minute
UNTIL cake looks good
IF cake tastes bad GOTO start

- there are no strict rules for writing pseudocode, but the idea is that you can tell exactly where an instruction ends, which instructions belong to a loop and which don't (indentation, brackets), etc.
- pseudocode is mostly used to communicate ideas for algorithms between humans, computers cannot understand pseudocode
- Do try to write down some pseudocode for your morning routine or the process of charging up your campus card!

Pseudocode

We need a way of writing down algorithms!

Example: Baking a Cake

```
start: gather all ingredients
REPEAT
        add the next ingredient to the bowl
UNTIL all ingredients are used
stir dough thoroughly
put dough into oven at 200°C
wait 50 minutes
REPEAT
        bake for another minute
UNTIL cake looks good
IF cake tastes bad GOTO start
```

Good:

- individual steps
- structure
- fairly readable

Bad:

- not specific enough
- dough, oven, etc. not defined

Pseudocode
We need a way of writing down algorithms!
Example: Baking a Cake
start: gather all ingredients
REPEAT
add the next ingredient to the bowl
UNTIL all ingredients are used
stir dough thoroughly
put dough into oven at 200°C
wait 50 minutes
REPEAT
bake for another minute
UNTIL cake looks good
IF cake tastes bad GOTO start
Good:
individual steps
structure
fairly readable
Bad:
not specific enough
dough, oven, etc. not defined

- there are no strict rules for writing pseudocode, but the idea is that you can tell exactly where an instruction ends, which instructions belong to a loop and which don't (indentation, brackets), etc.
- pseudocode is mostly used to communicate ideas for algorithms between humans, computers cannot understand pseudocode
- Do try to write down some pseudocode for your morning routine or the process of charging up your campus card!

Programming Languages...

...are an even more formal way of writing algorithms.

- easier to understand for computers
- strict rules regarding syntax etc.
- there are tons and Python is one of them!
- even this presentation is written in a programming language[2]

[2] Stephen Hicks. "Rapid Prototyping in TEX." In: *The Monad Reader* 13 (2009), pp. 5–17.

Programming Languages...
...are an even more formal way of writing algorithms.
- easier to understand for computers
- strict rules regarding syntax etc.
- there are tons and Python is one of them!
- even this presentation is written in a programming language[2]
[2] Stephen Hicks. "Rapid Prototyping in TEX." In: The Monad Reader 13 (2009), pp. 5–17.

some other programming languages you may have heard of are:

- Java
- C, C++, C#, Arnold-C
- PHP
- Matlab
- Haskell

From High-Level to Low-Level

Actually, computers really only understand binary

Some binary code

0100110111100101101101101001...

- only a few, very basic instructions
- higher-level programming languages build on top of that
- all programs must be translated into binary code (compilation, interpretation)
- we don't need to worry about that

From High-Level to Low-Level
Actually, computers really only understand binary
Some binary code
0100110111100101101101101001...
only a few, very basic instructions
higher-level programming languages build on top of that
all programs must be translated into binary code (compilation, interpretation)
we don't need to worry about that

- 1s and 0s correspond to high and low voltage in the computer
- Programming in binary would be incredibly inconvenient
- High-level languages solve that by for instance giving meaningful names to blocks of 1s and 0s
- Imagine it like replacing "go to store; collect stuff; go back home" with "go buy stuff"
- **Compilation** is the process of translating a high-level program into machine-code
- **Interpretation** is similar, but the program is translated as it is being executed

Soo.. what is programming?

Two aspects for solving a problem with programming:

Soo.. what is programming?

Two aspects for solving a problem with programming:

- Note that having a clearly defined problem is often a very hard first step
- **Designing** includes everything up to having pseudo-code
- **Implementation** includes everything from choosing an appropriate language up to running the program
- We will of course stick to Python
- A good scheme for solving a problem is useless if you cannot tell your computer how it works

Soo.. what is programming?

Two aspects for solving a problem with programming:

- **Designing an algorithm**

Soo.. what is programming?
Two aspects for solving a problem with programming:
- Designing an algorithm

- Note that having a clearly defined problem is often a very hard first step
- **Designing** includes everything up to having pseudo-code
- **Implementation** includes everything from choosing an appropriate language up to running the program
- We will of course stick to Python
- A good scheme for solving a problem is useless if you cannot tell your computer how it works

Soo.. what is programming?

Two aspects for solving a problem with programming:

- **Designing an algorithm**
- **Implementing said algorithm**

Both are equally important for a good program

Soo.. what is programming?
Two aspects for solving a problem with programming:
- Designing an algorithm
- Implementing said algorithm
Both are equally important for a good program

- Note that having a clearly defined problem is often a very hard first step
- **Designing** includes everything up to having pseudo-code
- **Implementation** includes everything from choosing an appropriate language up to running the program
- We will of course stick to Python
- A good scheme for solving a problem is useless if you cannot tell your computer how it works

Soo.. what is programming?

Two aspects for solving a problem with programming:

- **Designing an algorithm**
- **Implementing said algorithm**

Both are equally important for a good program

We will focus more on implementation

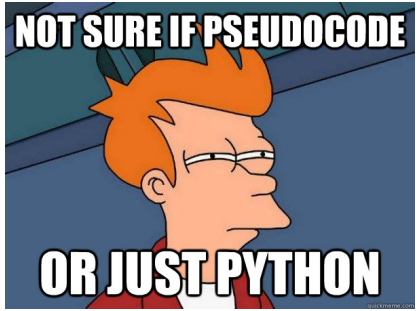
Soo.. what is programming?
Two aspects for solving a problem with programming:
- Designing an algorithm
- Implementing said algorithm
Both are equally important for a good program
We will focus more on implementation

- Note that having a clearly defined problem is often a very hard first step
- **Designing** includes everything up to having pseudo-code
- **Implementation** includes everything from choosing an appropriate language up to running the program
- We will of course stick to Python
- A good scheme for solving a problem is useless if you cannot tell your computer how it works

Python

Python

A high-level language that is easy to learn, read and write.



Python
Python
A high-level language that is easy to learn, read and write.

Why Python?

Advantages

1. Widespread usage (especially in academia)
2. Open source environment
3. Steep learning curve
4. Multiplatform support (Windows, Linux, Mac)
5. Large ecosystem of libraries and packages

Why Python?
Advantages
1. Widespread usage (especially in academia)
2. Open source environment
3. Steep learning curve
4. Multiplatform support (Windows, Linux, Mac)
5. Large ecosystem of libraries and packages

Popular Python Packages

1. Scikit-Learn for Machine Learning
2. Numpy and Pandas for Data Processing
3. OpenCV for Computer Vision
4. Spacy and NLTK for Natural Language Processing
5. Tensorflow and Keras for Neural Networks

Why Python?

Disadvantages

1. Slow execution
2. High memory usage
3. Requires Python Interpreter

Why Python?
Disadvantages
1. Slow execution
2. High memory usage
3. Requires Python Interpreter

These disadvantages can be handled in most cases. However, for some things like programming mobile apps there are much more efficient languages.

```python
print("hello world!")
```

```python
# live demo
```


print("hello world!")
# live demo

A **terminal** (or command line interface) is a way to give instructions to the computer in text form. You type in a command, hit [↵] and then the computer usually gives a response in text form as well.

How to open a terminal:

- **Windows:** Press [⊞]+[R] and type `cmd`. Hit [↵].
- **Ubuntu:** Press [Ctrl]+[Alt]+[T].
- **macOS:** Press [⌘]+[ ] and type `terminal`. Hit [↵].

The first command we will use is `python`. It opens an **interactive Python shell**.

print("hello world!")

```
>>> print("hello world!")
hello world!

>>> print(hello world!)
  File "<stdin>", line 1
    print(hello world!)
                      ^
SyntaxError: invalid syntax
```


print("hello world!")

>>> print("hello world!")
hello world!

>>> print(hello world!)
File "<stdin>", line 1
print(hello world!)

SyntaxError: invalid syntax

A **Python shell** is much like the normal terminal, but it understands Python code. The first thing you need to know about is the `print()` function (we'll explain later what *function* means). It writes whatever is inside the parentheses to the terminal.

Note the `""` around the text. It indicates that whatever is inside is **not** to be interpreted as code but as text (called a **string**). Running `print(hello world!)` will result in an error.

We will talk about error messages and what they mean in a later lecture, but for now, know this: They happen all the time to experienced programmers and beginners alike and are part of the process of writing a good program. Please do not be discouraged, but instead try to find what went wrong - either on your own or with help - and fix it!

Python Shell as a Calculator

```
>>> print(42)
42

>>> print(20 + 22)
42

>>> print("4" + "2")
42

>>> print("42" * 5)
4242424242
```


Python Shell as a Calculator
>>> print(42)
42
>>> print(20 + 22)
42
>>> print("4" + "2")
42
>>> print("42" * 5)
4242424242

Of course, Python can deal with numbers. Now we do not need the "", because we do not want Python to treat our numbers as text!

With numbers, we can do math: There are symbols for many mathematical operations, called **operators**. Here are some:

- + and −: addition and subtraction
- ∗ and /: multiplication and division
- ∗∗: exponentiation (2 ∗∗ 4 reads 2^4)

The + operator can also be used on strings, but it has a different effect: "ab" + "cd" means appending "cd" to "ab", resulting in "abcd" (called **concatentation**).

Using the Terminal

```
# live demo
```


Using the Terminal
live demo

Most commands are relative to the current working directory (shown to the left of your cursor).

List of basic commands:

- `python` – opens an interactive Python shell
- `dir` – lists the files and subdirectories in the current directory (windows)
- `ls` – like `dir` but on linux and macOS
- `cd <directory_name>` – change into a diretory
- `mkdir <directory_name>` – create a new directory
- `rmdir <directory_name>` – remove a directory

Python Scripts

With algorithms in mind, we often want to execute many lines of code in immediate succession

```
print("I am a script!")
print("All I do is print stuff.")
print("But I can do this: " + "blub" * 10)
```

If we save this in a file `my_script.py`, we can run everything with `python my_script.py`

Python Scripts

With algorithms in mind, we often want to execute many lines of code in immediate succession

print("I am a script!")
print("All I do is print stuff.")
print("But I can do this: " + "blub" * 10)

If we save this in a file my_script.py, we can run everything with python my_script.py

- `python my_script.py` only works if your terminal is in the same directory as `my_script.py` (see `cd`)

This Week's Homework

1. Install Python
2. First experiments with terminal
3. Use the Python turtle environment

→ For details see file 01_Introduction_Ex.pdf
→ For help come to the Walk-In Practice Session on Thursday from 12:15

This Week's Homework
1. Install Python
2. First experiments with terminal
3. Use the Python turtle environment
→ For details see file 01_Introduction_Ex.pdf
→ For help come to the Walk-In Practice Session on Thursday from 12:15

- Each assignment should be solved in a separate file!
- Pack all files into a .zip archive and upload it to your groups folder on StudIP

References

Hicks, Stephen. "Rapid Prototyping in TEX." In: *The Monad Reader* 13 (2009), pp. 5–17.

Wikipedia contributors. *Algorithm — Wikipedia, The Free Encyclopedia*. [Online; accessed 24-February-2019]. 2019. URL: https://en.wikipedia.org/wiki/Algorithm.

References

Hicks, Stephen. "Rapid Prototyping in TEX." In: *The Monad Reader* 13 (2009), pp. 5–17.

Wikipedia contributors. *Algorithm — Wikipedia, The Free Encyclopedia*. [Online; accessed 24-February-2019]. 2019. URL: https://en.wikipedia.org/wiki/Algorithm.