Python from Zero

Computer Fundamentals

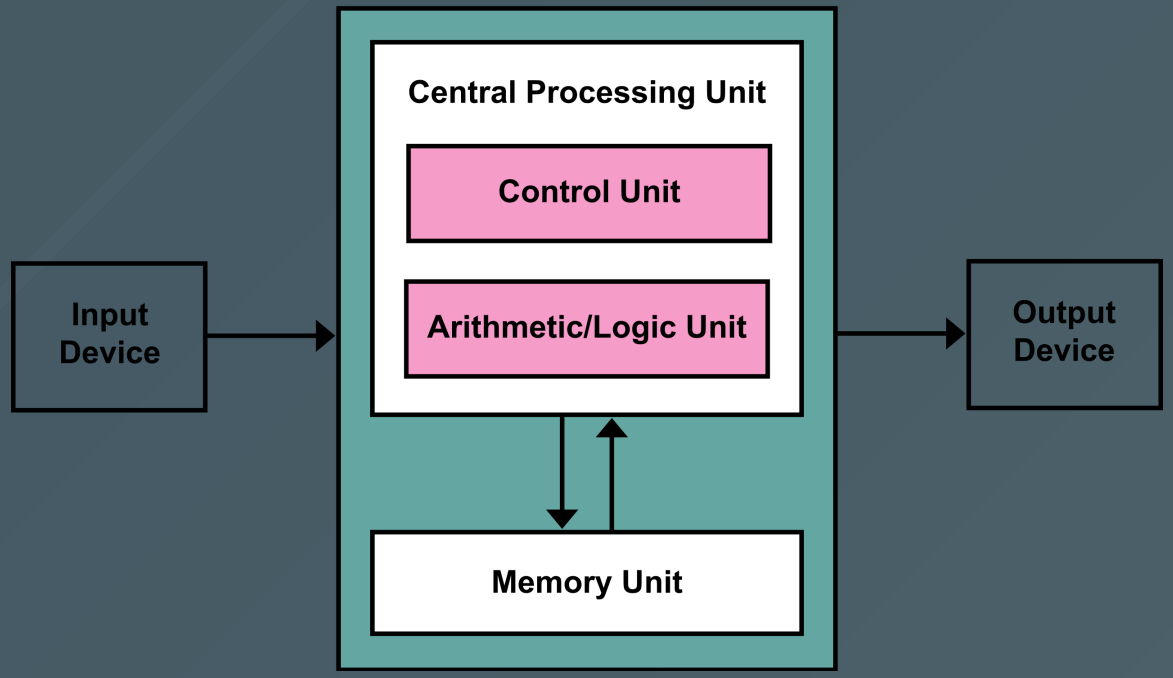
How a PC Works

Von Neumann architecture

- Memory (RAM): Stores *data* and *instructions*
- CPU: Fetches instructions from memory, Reads/writes data from/to memory, performs computations, controls data flow

(Reality is more complicated)

Image: wikipedia



Instructions

The lowest level: Tell the CPU directly what to do

- Store something in a certain place
- Fetch data from this location, put it in some cache
- Take values from two places, add them, store the result in another place
- Go to the next instruction
- … … …

Big set of instructions hardwired into the CPU, ultimately each program gets translated into a series of commands using these instructions

Example: The instruction `MOV AL,10` *(or* `A0 0A` *or* `10100000 00001010`*) tells the CPU to write ("move") the number 10 to a place ('register', similar to memory) called* `AL`

Instructions

Luckily, we don't have to worry about that at all!

Python is an interpreted language

- We can enjoy writing high-level Python code
- A program called the *Python interpreter* reads our code "line-by-line" and takes care of creating and executing the appropriate CPU instructions on the fly

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

Screenshot: Python code and immediate execution

The Memory

Imagine a 1-dimensional array of 0's and 1's ("bits"): `0101000 10011010 10101010 ...`

- Every piece of data (numbers, text, images, audio, ...) is stored as a sequence of 0's and 1's ("bits")
- These sequences are (almost always) grouped by 8 digits (a "byte"), and each byte has a unique address

- **Python manages for us many of the details of how and where exactly the data is stored**
  - Thus, we don't need to worry about how many bytes we need and at which address to store them and so on

Python Basics

Writing and Executing Python

Different options:

- Write a textfile ("script") with Python code and let the interpreter execute it
- Start the interpreter in a *terminal* and write Python line-by-line, directly executing each line as you go ("read-eval-print loop", REPL)

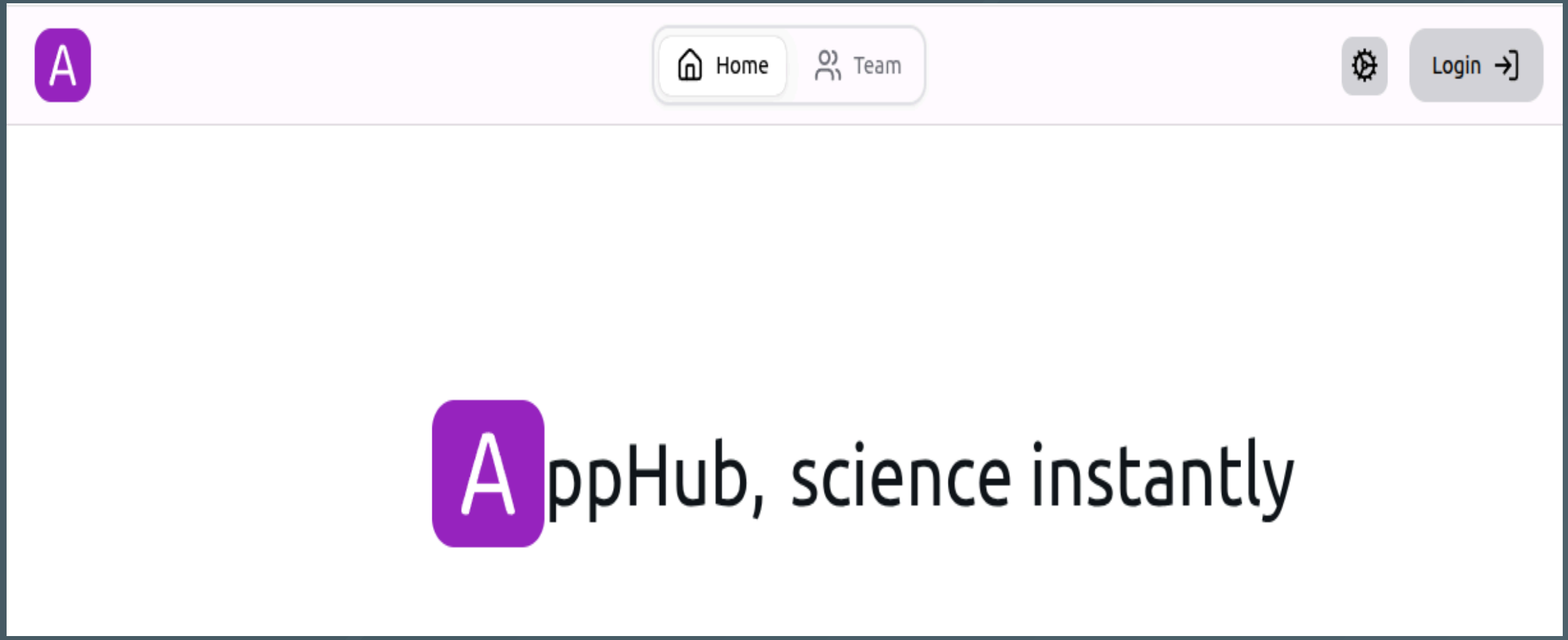
```
>>> print("Hello world!")
Hello world!
```

- A mixture: Jupyter Notebooks!
  - Write single or multiple lines of Python in separate "code cells"
  - Execute code inside individual cells as you need
  - You can have text cells in between where you can take notes - very helpful!

We will use Notebooks in this course!

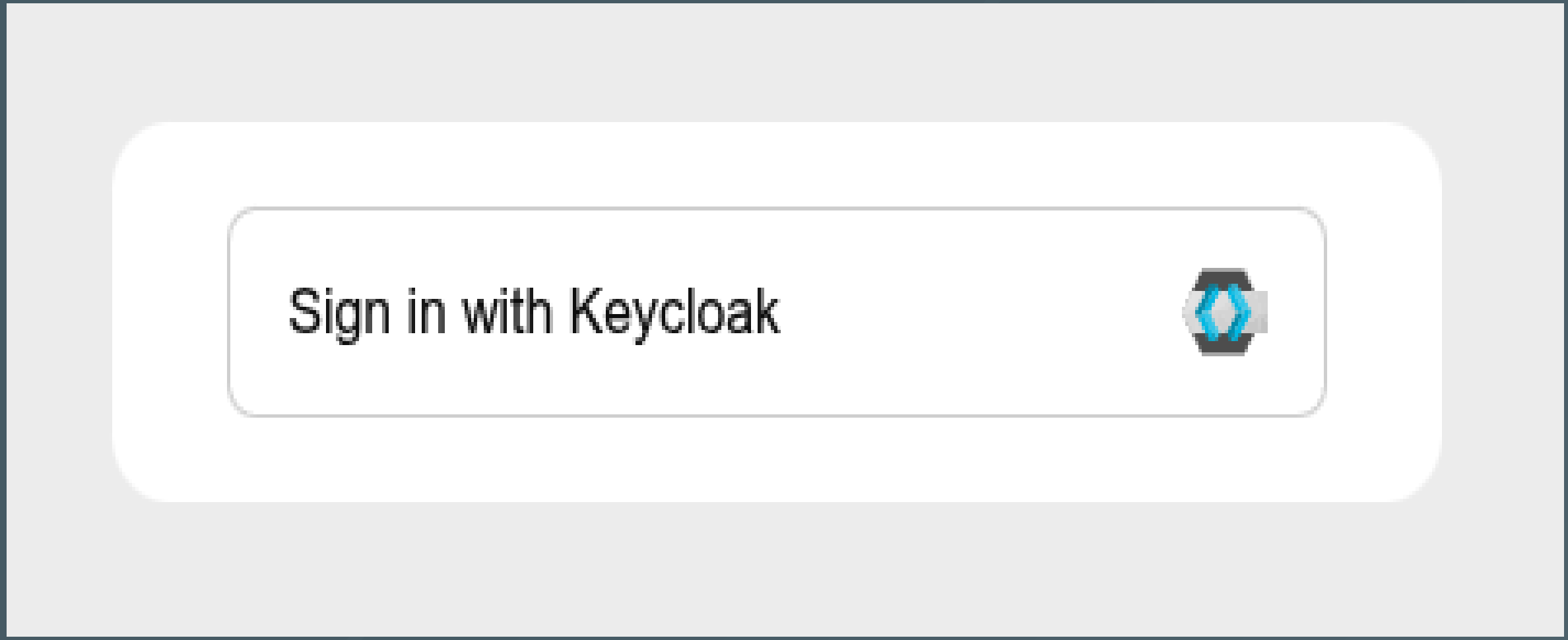
It's Your Turn!

- Go to https://apphub.wolke.uni-greifswald.de/ and login



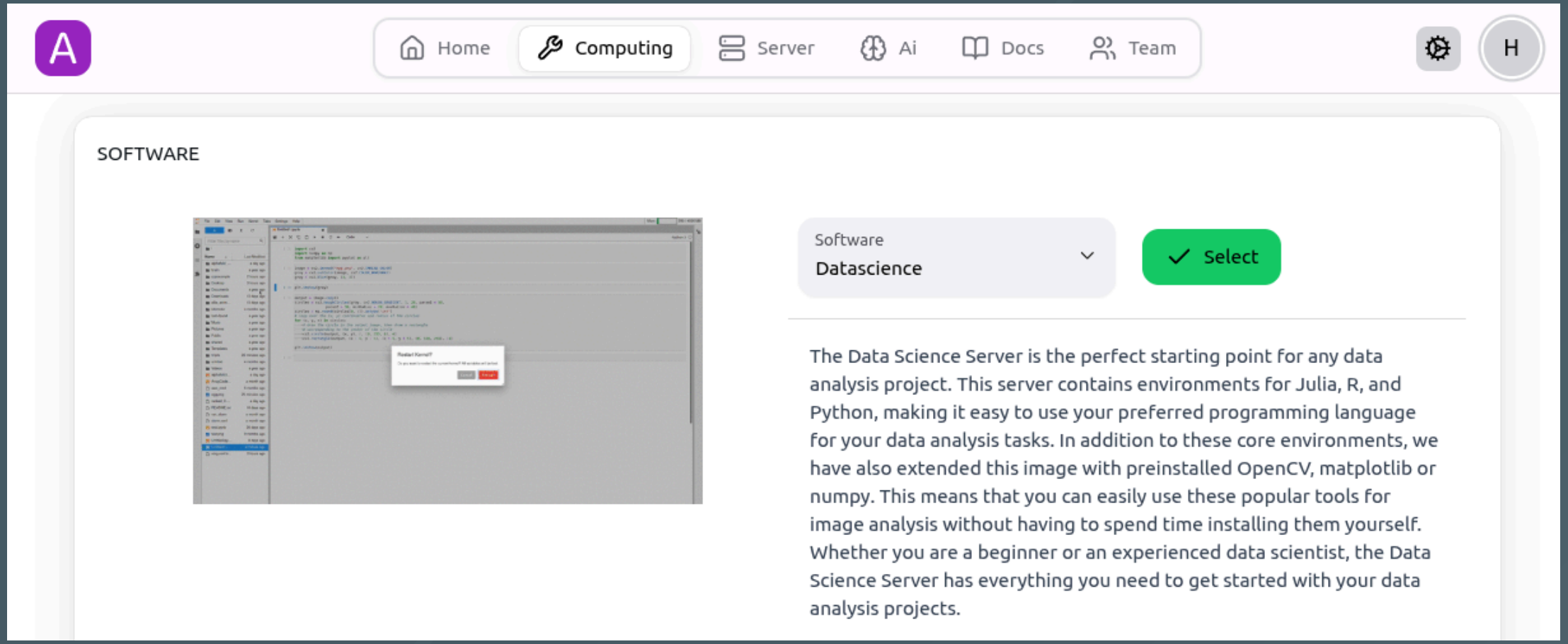
It's Your Turn!

- Use Shibboleth to sign in



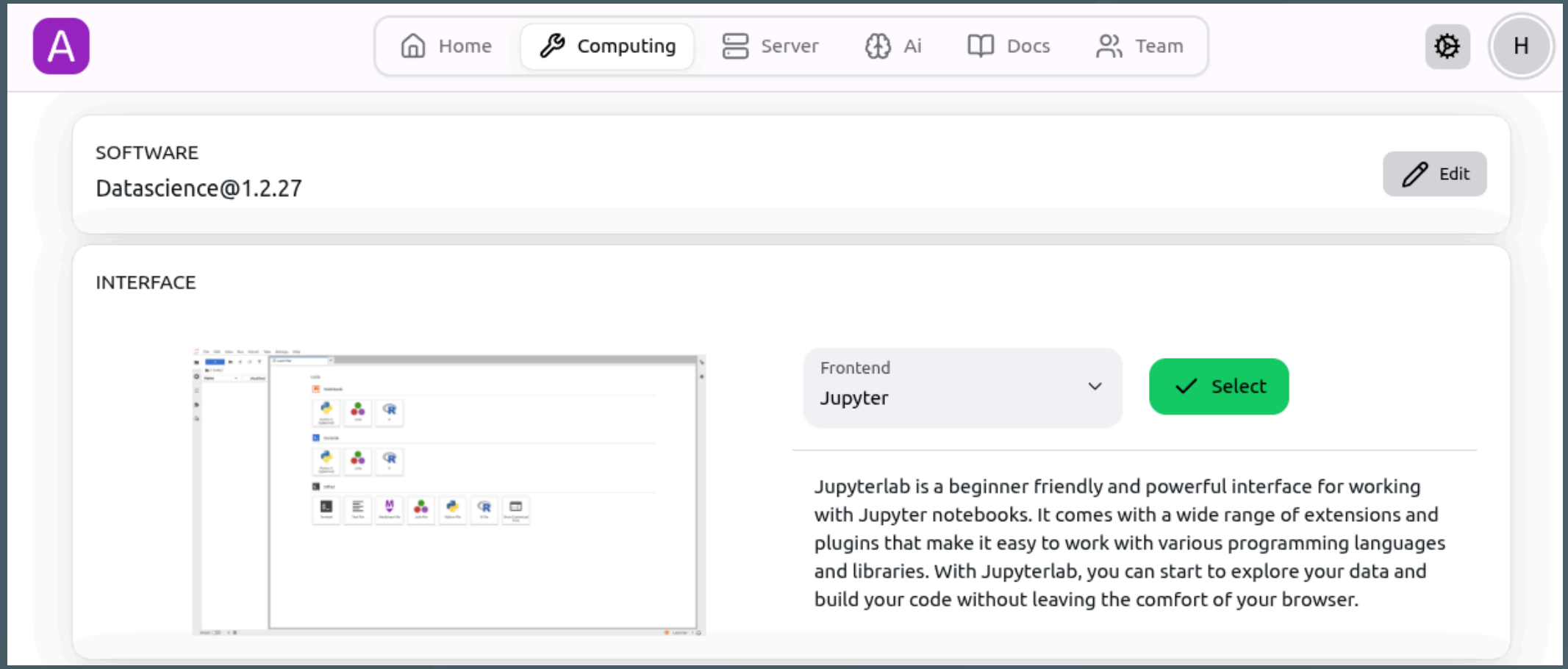
It's Your Turn!

- Select "Datascience"



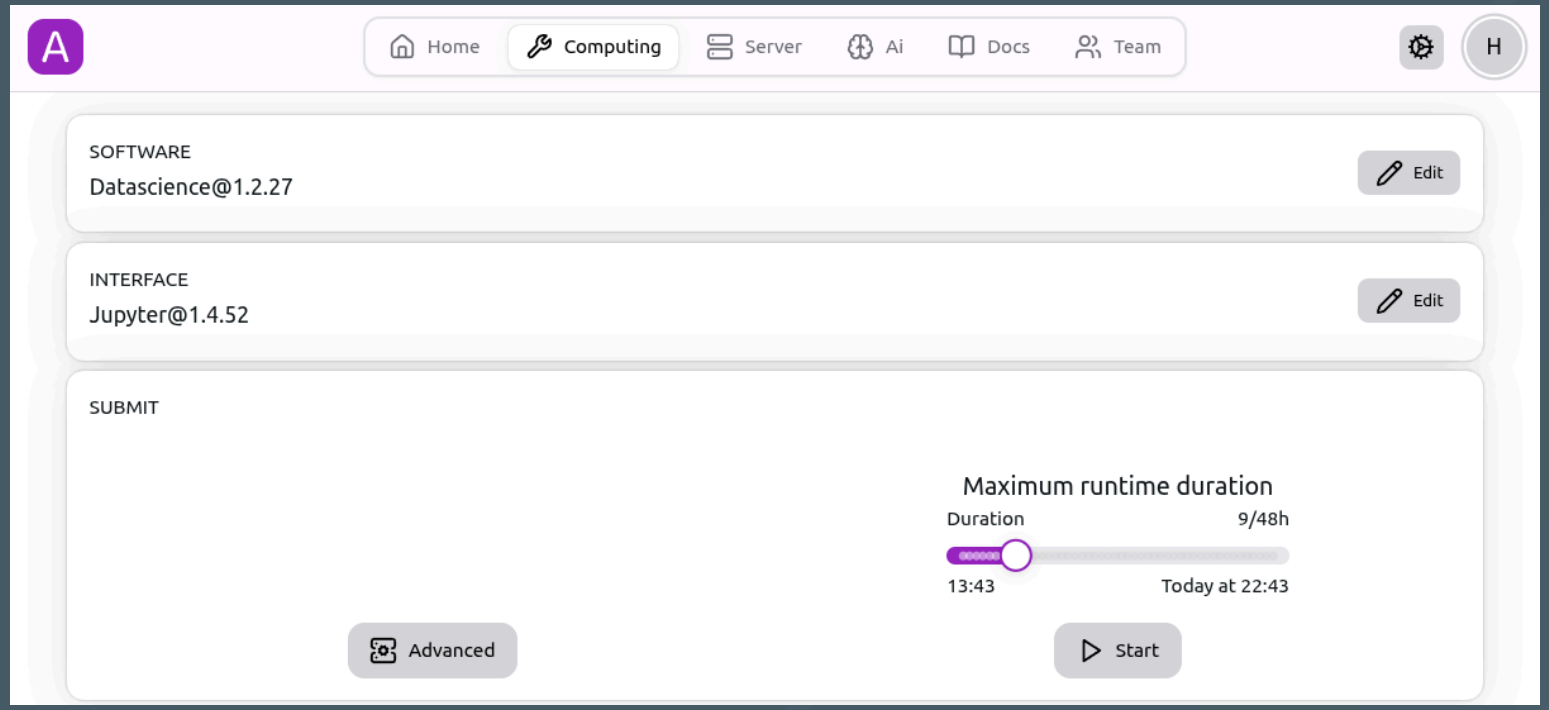
It's Your Turn!

- Select "Jupyter"



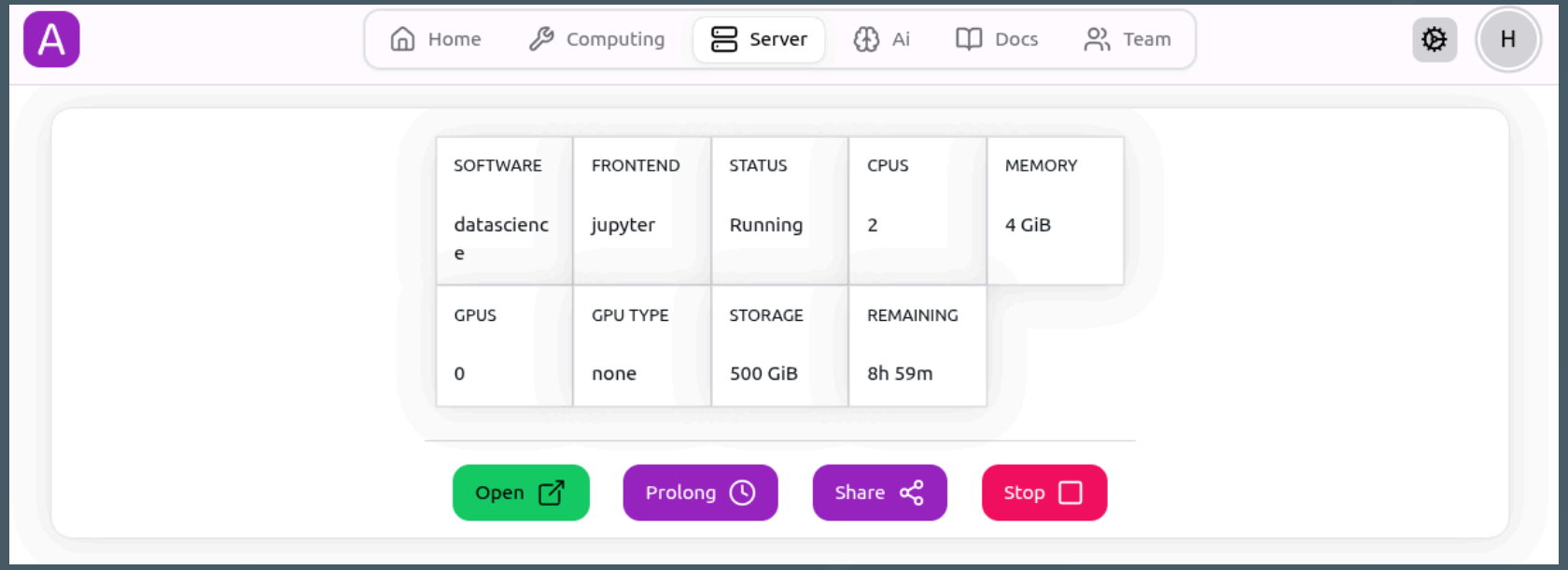
It's Your Turn!

- Your own server will start up! Select a time you want to keep it running:



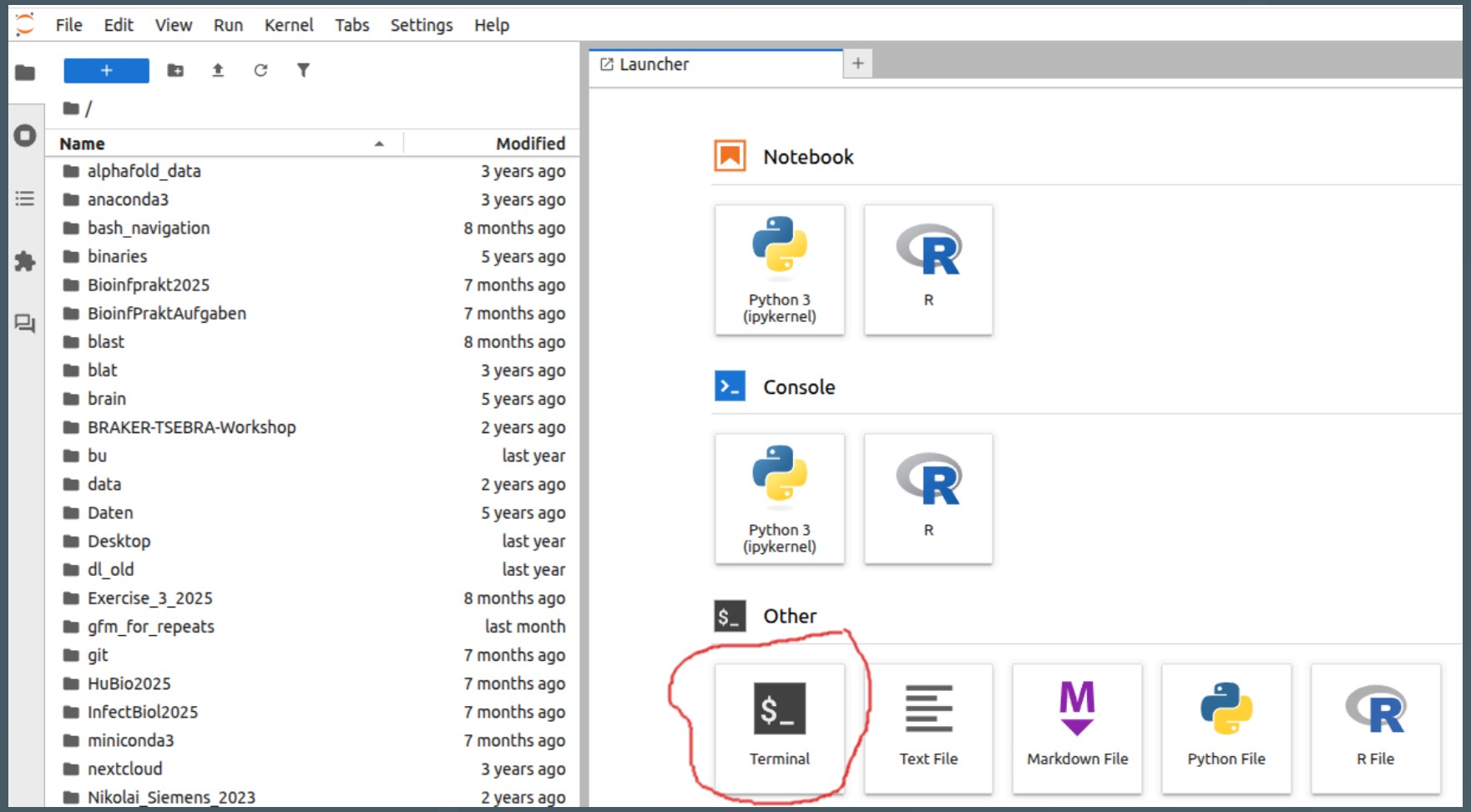
It's Your Turn!

- Wait until the server has booted, and then click "Open" at the bottom



It's Your Turn!

- Click on "Terminal" (main window, section "Other")



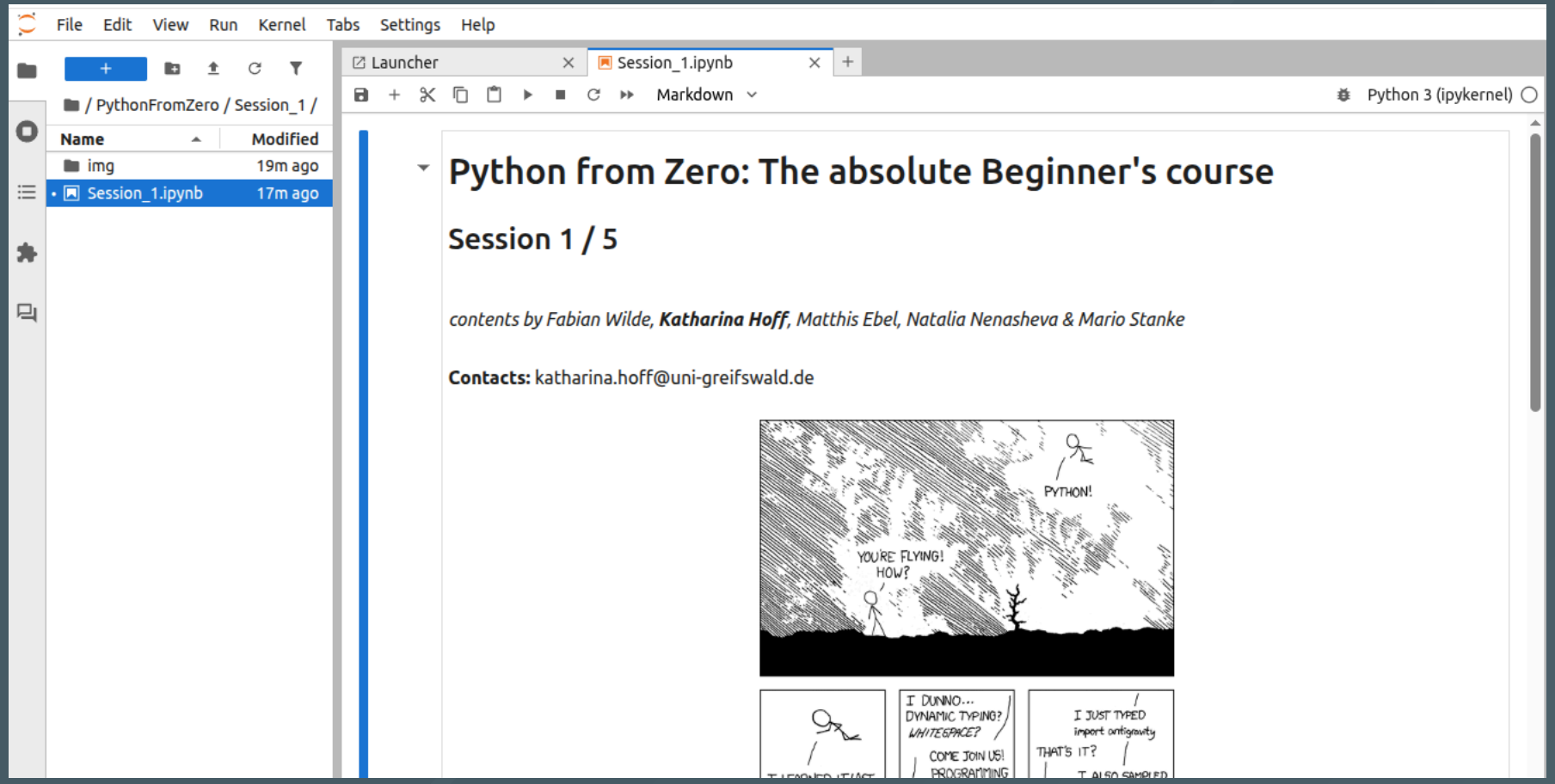
It's Your Turn!

- Download course material
  In the terminal, type

```
git clone https://github.com/DataCompetency/PythonFromZero
```

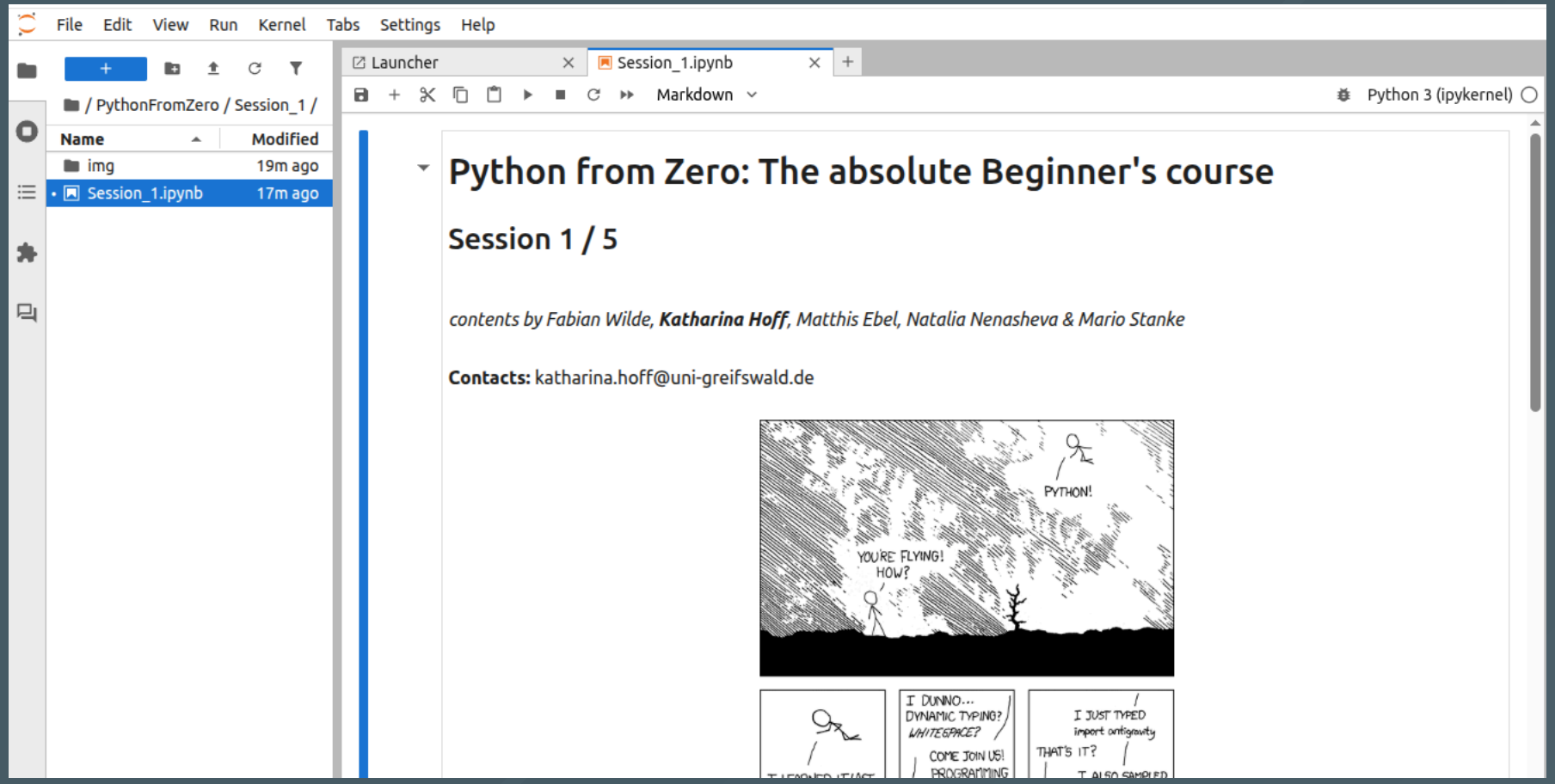
It's Your Turn!

- In the file tree on the left, navigate to `PythonFromZero/Session_1/` and double click on `Session_1.ipynb`



It's Your Turn!

- Congratulations! You are now ready to write and execute Python code!



It's Your Turn!

- Please read the notebook until the section
  "Useful Keyboard Shortcuts in a Jupyter Notebook"

Can you identify the three code cells under "Usage modes for Python"? Try to execute them!

Change the code inside the code cells (e.g. change the summands or the text inside `print()`) and execute them!

Python Basics

Variables

Variables

A piece of memory with a *name*, a *type* and a *value*

- You can choose the name (relatively) freely, e.g. `x` or `foo` or `my_variable` etc.
  - Python takes care of grabbing a piece of memory and remembers that your variable name now corresponds to the address of the memory piece
- You do not specify the type yourself! Python determines the type based on the value you assign
- You can only create a variable by assigning a value to a name:

```
x = 42          # an integer
y = "hello"     # a string
```

Variables

Types

Although you don't specify them, each variable in Python has a type

The most basic types are

Type	Description
`int`	"Whole numbers", e.g. 0, 1, 2, 3, -1, -2, -3, ...
`float`	Floating point numbers, e.g. 3.14, 0.1, -1.23
`bool`	Boolean, can only have the values `True` or `False`
`str`	Holds text values (actually not that basic)

Variables

Types

Unlike other programming languages, variable names do not have to keep their initial type

```
x = 42   # int
print(x)
x = "Now I'm a string!"
print(x)
```

```
42
Now I'm a string!
```

No errors here, this is perfectly valid!

Variables

Types

You can use the built-in function* `type()` to check the type of a variable

```
x = 42    # int
print(type(x))
x = "Now I'm a string!"
print(type(x))
```

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

* we will learn about functions later

Feel free to code along!
Press [Esc], then the [A] or [B] key to create a new cell in your Notebook.

Working with Basic Variables

The numeric types `int` and `float` allow basic arithmetic operations with special *operators*

```
1 + 1   # add two integers
2 + 0.1 # add an int and a float, the result is the float 2.1
0 - 1   # subtract
2 * 2   # multiply
3 / 4   # divide two integers, the result is the float 0.75
4 / 2   # divide two integers, the result is still a float (2.0)
1 / 0   # ERROR, as in mathematics, division by zero is impossible

2**8    # 2 to the power of 8, result is 256
3 // 4  # divide and round down to the nearest integer, result is the integer 0
```

Working with Basic Variables

The `+` operator is also defined for `string`s (here, it *concatenates* strings!)

```
a = "Hello "
b = "world!"
print(a+b)
```

```
Hello world!
```

Working with Basic Variables

Mixing variable types sometimes works, but sometimes is illegal!

```
a = 1
b = 2.2
c = "a string"

a + b # okay, Python implicitly converts `a` to float
a + c # not okay, Python does not know what to do!
```

`"a string"` cannot be converted to a number, and Python does *not* assume you wanted to turn `a` into a string and concatenate. This you would have to do explicitly:

```
str(a) + c # this is fine, results in "1a string"
```

Use `int()`, `float()`, `bool()` and `str()` to explicitly convert something to a type

It's Your Turn!

- Please continue reading the notebook until *before* the section "Functions"

Do the exercise!

Exercise:
Write some code which defines 3 numbers, sums them and outputs the three numbers seperated by commas in one line and their sum in a new line.

Go to our Moodle page (https://moodle.uni-greifswald.de/course/view.php?id=9565) and take the first quiz! ("Quiz 1 - Variables")

Python Basics

Functions

Functions

We've already seen and used functions:

```
print()
type()
```

...but what *are* functions?

Functions

Remember mathematics:

$$f(x) = 2x + 5$$

$$f(3) = 2 \cdot 3 + 5 = 11$$

- `f` is the name of the function
- `x` is the argument of the function
- `2x + 5` is what happens to the argument in the function
- a function returns some result, e.g. `f(3) = 11`

Functions

In Python, functions also

- have a *name*
- can take one or more *arguments* (or *parameters*)
- have a *block of code* (the *function body*) that does things
- can return a result

Functions are self-contained modules of code that accomplish a specific task. Functions usually take in data, process it, and return a result. ***Once a function is written, it can be used over and over and over again****. Functions can be called from the inside of other functions."*

www.cs.utah.edu

Functions

`print()` - takes a list of arguments and writes it to the output (e.g. the Notebook)

```
x = 2
y = "foo"
print(1, x, y, y+"bar")
```

```
1 2 foo foobar
```

`type()` - takes a variable name or a value and returns the type of that variable (or value)

Functions

We can also write our own functions:

```
def FUNCTION_NAME(arg1, arg2):
    # some code
    # some more code
    result = arg1+arg2 # even more code
    return result
```

- Always write the keyword `def`, followed by a function name (choose something meaningful here), followed by `( )` that may contain one or more argument names, followed by `:`
  - the argument names are available in the function as variables
- In an **indented block of code**, write what the function should do
- If the function should return something, write the keyword `return` followed by the *return value* as the last line

Functions

Indentation

This is a key concept of Python!

- Other languages have special symbols to organize code blocks
- In Python, code blocks (like function bodies) are solely indicated by *indentation*

*Each line in a function body **must** be indented by the same number of whitespaces (usually 4)*

If indentation is not consistent in your code, you will get an error!

Functions

```
# The same function as above
def f(x):              # def statement is not indented
    result = 2*x + 5 # note the 4 whitespaces at the beginning of this line -> function body
    return result    # indented -> function body

f(3)                   # no 4 whitespaces here, this line does not belong to the function!
```

```
11
```

It's Your Turn!

- Please continue reading the notebook, read the "Functions" section

Do the exercise!

Exercise 2:
Define a function to greet a person with his/her individual name. Use the print function to output the greeting. The function should expect one argument containing the person's name.