Python from Zero

Control Flow and Loops

Control Flow

There is not much use in writing a bunch of variable assignments and calculations line after line

```
x = 1
y = 1.2
l = []
l.append(x+y, x-y)
d = {'list': l,
     'x': x,
     'y': y}
```

Control Flow

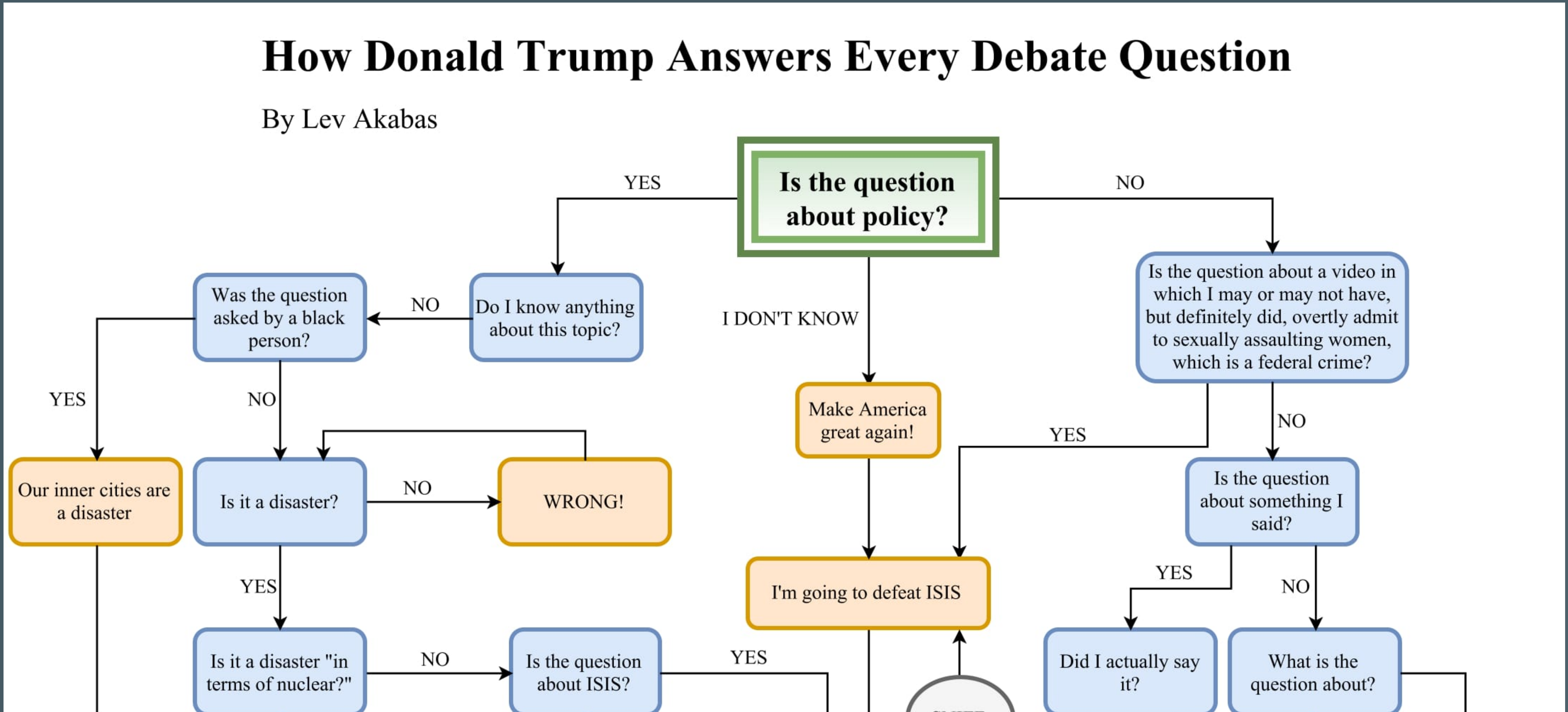
Meaningful programs need to react to things that happen, and act accordingly!

The basic construct to do such things is the `if ... elif ... else` clause:

```
if condition:
    # do something if condition is True
elif other_condition:
    # do something else if other_condition is True and condition was False
else:
    # do this if neither condition nor other_condition are True
```

Control Flow

Source: https://www.liberalforum.org/topic/231679-flow-chart/


How Donald Trump Answers Every Debate Question
By Lev Akabas
Is the question about policy?
YES
NO
I DON'T KNOW
Do I know anything about this topic?
NO
Was the question asked by a black person?
YES
NO
Our inner cities are a disaster
Is it a disaster?
NO
WRONG!
YES
Is it a disaster "in terms of nuclear?"
NO
Is the question about ISIS?
YES
Make America great again!
I'm going to defeat ISIS
Is the question about a video in which I may or may not have, but definitely did, overtly admit to sexually assaulting women, which is a federal crime?
YES
NO
Is the question about something I said?
YES
NO
Did I actually say it?
What is the question about?

Control Flow

Conditions

```python
if condition:
    # do something
```

Conditions are essentially "yes or no" questions

- A condition must evaluate to `True` or `False` (i.e. boolean values)
- If the condition evaluates to `True`, the associated code block is executed

```python
if True:
    print("This is always printed")

if False:
    print("This is never printed")
```

Control Flow

Writing Conditions

Usually, we want to *check* something with a condition

For basic data types, we have a set of comparison operators:

```
a == b # checks if a has the exact same value as b

a != b # checks if a and b have different values

a < b  # checks if a's value is smaller than b

a <= b # checks if a's value is smaller or equal to b

a > b  # checks if a's value is bigger than b

a >= b # checks if a's value is bigger or equal to b
```

Control Flow

Writing Conditions

```
a = 42
b = 3.14
if a > b:
    print("Condition is True, thus this message is printed")
```

Comparison operators return a boolean value:

```
print(a != b)
```

```
True
```

Control Flow

Writing Conditions

For container types, you have already seen the `in` operator, which also returns a boolean:

```
x = [1, 2, 3, 4, 5]
if 6 in x:
    print("6 is already in the list")
else:
    x.append(6)
```

Control Flow

Writing Conditions

Logical operators

- Combine boolean expressions
- Keywords `and`, `or`, `not`

```
x = 0
if not type(x) == str and x != 0:
    print(1/x)
else:
    print("Division by str or by zero is not allowed")
```

```
Division by str or by zero is not allowed
```

Control Flow

Writing Conditions

Logical `and`:

```
x = a and b
```

`a`	`b`		`x`
`True`	`True`		`True`
`False`	`True`		`False`
`True`	`False`		`False`
`False`	`False`		`False`

Control Flow

Writing Conditions

Logical `or`:

```
x = a or b
```

`a`	`b`		`x`
`True`	`True`		`True`
`False`	`True`		`True`
`True`	`False`		`True`
`False`	`False`		`False`

Control Flow

Writing Conditions

Logical `not`:

```
x = not a
```

`a`		`x`
`True`		`False`
`False`		`True`

Control Flow

Alternatives

```python
if condition1:
    # do something
elif condition2:
    # do something else
    # but only of condition1 was False
elif condition3:
    # do this instead
    # but only if both condition1 and condition2 were False
else:
    # do something else
    # but only if none of above conditions were True
```

```python
if conddition1:
    if condition2:
        print("Nested conditions are also possible")
```

Short Interlude

Python has the special object `None`

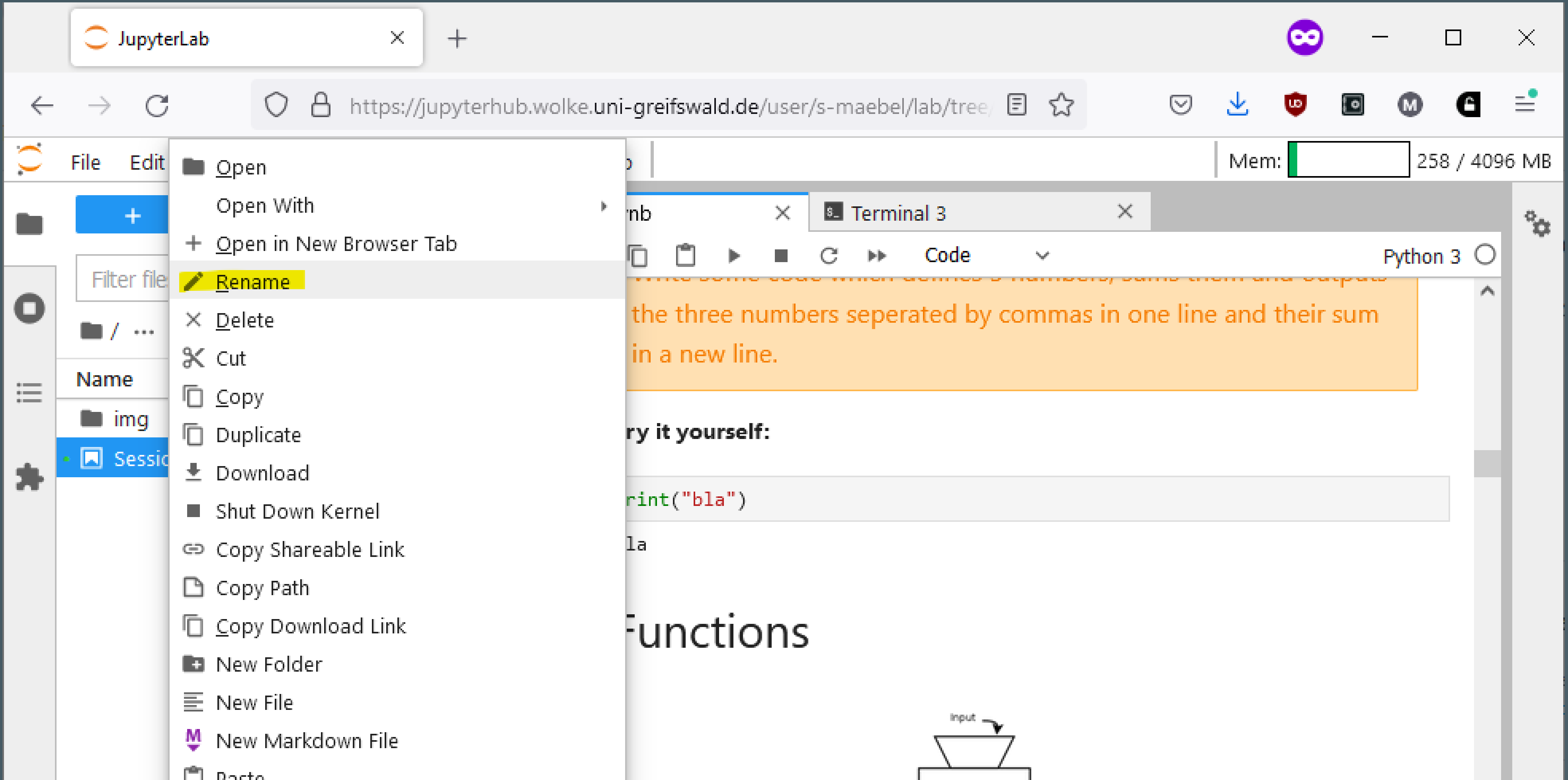
- You can assign `None` to a variable name to signal that this variable *has no value*
- `None` is ***not*** the same as `False` or `Zero` (those are both values)!

You can test if a variable has no value with the keyword `is`

```
x = None
if x is None:
    print("x has no value! Let's assign one...")
    x = 0
elif x is not None:
    print("x has the value", x)
```

It's Your Turn!

- Go to the AppHub and *rename* `PythonFromZero/Session_1/Session_1.ipynb` if you want to keep your changes!



the three numbers seperated by commas in one line and their sum in a new line.

ry it yourself:

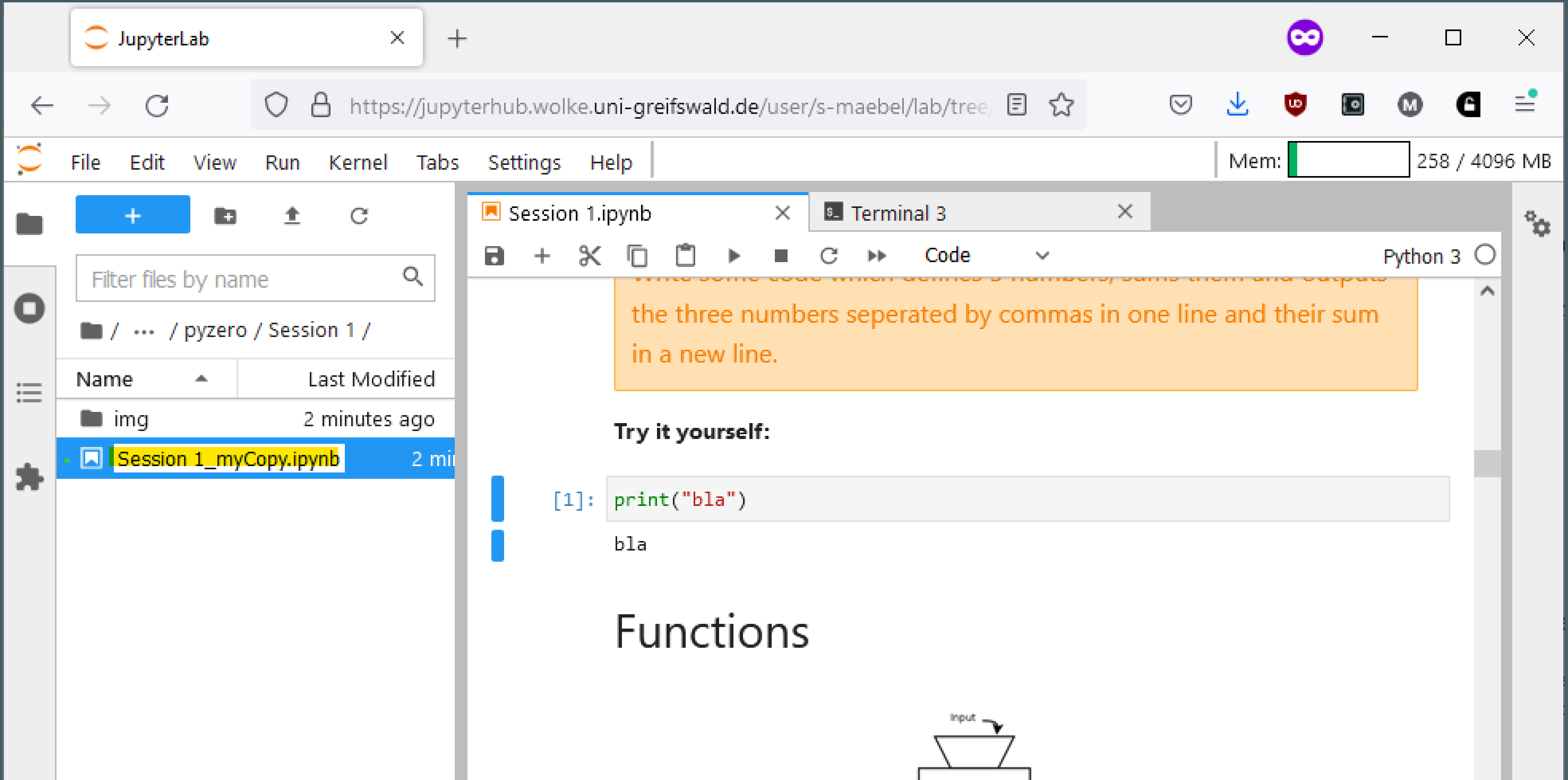
```
rint("bla")
```

```
la
```

Functions

It's Your Turn!

- Go to the AppHub and *rename* `PythonFromZero/Session_1/Session_1.ipynb` if you want to keep your changes!



the three numbers seperated by commas in one line and their sum in a new line.

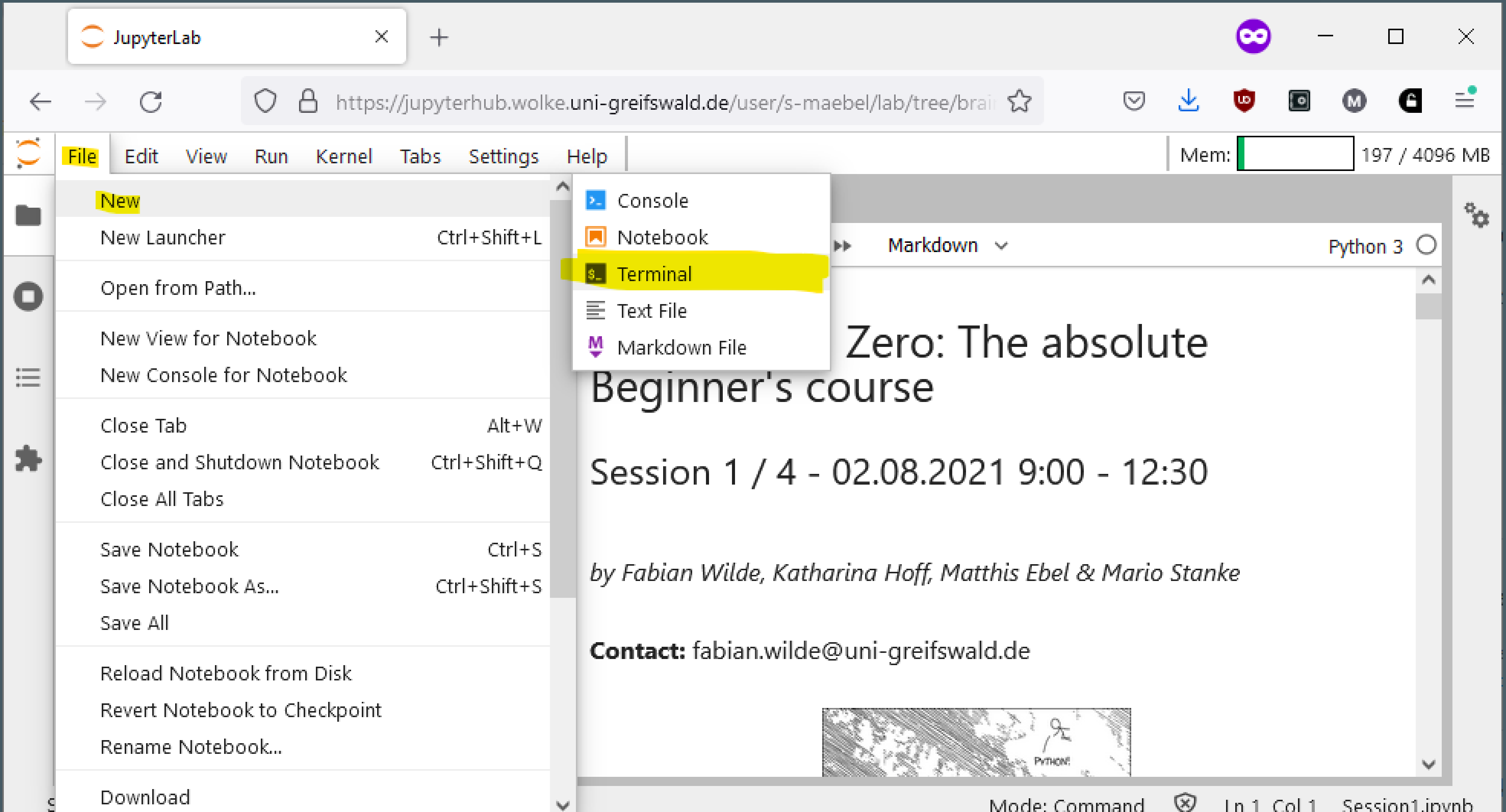
Try it yourself:

```
[1]: print("bla")
bla
```

Functions

It's Your Turn!

- Open a new terminal (or use the old one if it is still open)



Zero: The absolute Beginner's course

Session 1 / 4 - 02.08.2021 9:00 - 12:30

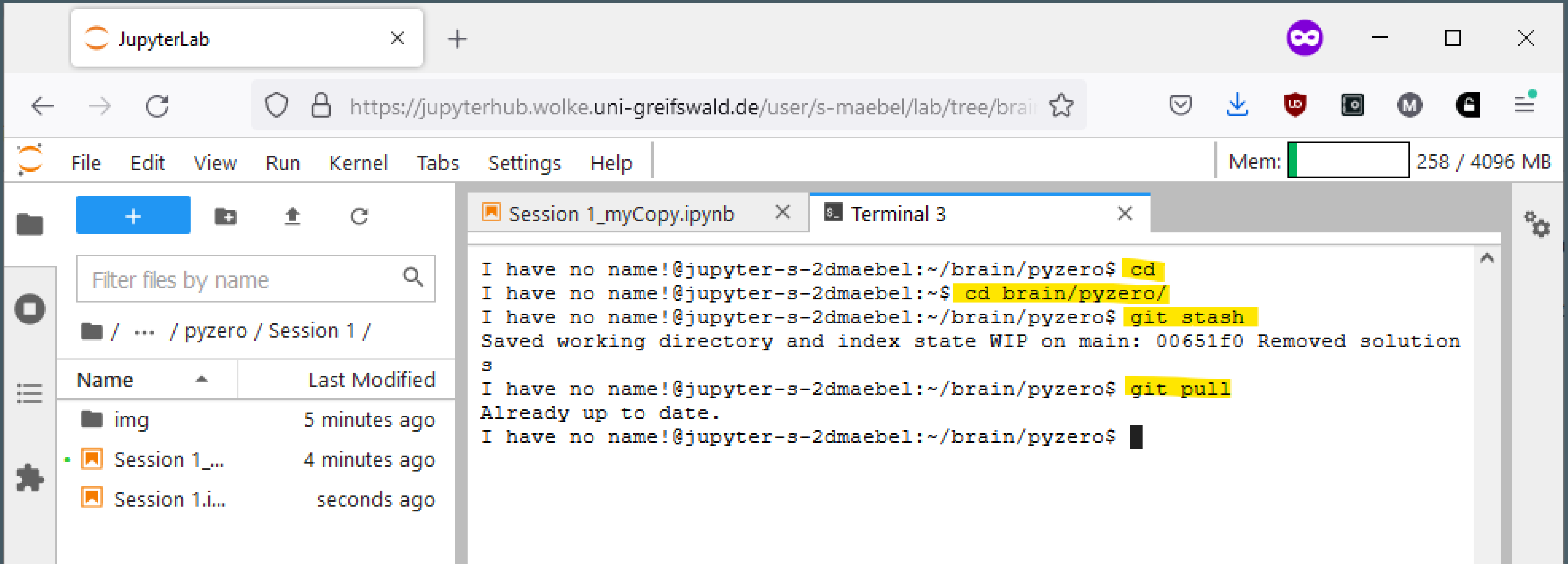
by Fabian Wilde, Katharina Hoff, Matthis Ebel & Mario Stanke

Contact: fabian.wilde@uni-greifswald.de

It's Your Turn!

- Type the following commands to get the latest material

```
cd
cd PythonFromZero
git stash
git pull
```



```
I have no name!@jupyter-s-2dmaebel:~/brain/pyzero$ cd
I have no name!@jupyter-s-2dmaebel:~$ cd brain/pyzero/
I have no name!@jupyter-s-2dmaebel:~/brain/pyzero$ git stash
Saved working directory and index state WIP on main: 00651f0 Removed solution
s
I have no name!@jupyter-s-2dmaebel:~/brain/pyzero$ git pull
Already up to date.
I have no name!@jupyter-s-2dmaebel:~/brain/pyzero$
```

It's Your Turn!

- Open the notebook `Session_2.ipynb` and work through it until before the "Loops" section

Do the exercise!

Exercise 1:

Implement a simplified version of the dice game "Kniffel". Roll two dices. Output the results for the two dices. If the two random integers are equal, additionally notify the user that he got an "n-er Pasch" where n stands for the random integer. Otherwise, also inform the user.

Go to our Moodle page (https://moodle.uni-greifswald.de/course/view.php?id=9565) and take the third quiz! ("Quiz 3 - Conditions")

Loops

Loops

Repeat the same block of code over and over again

- For-loops: repeat the code *for a certain number of times*

```
for i in [0,1,2,3,4,5]:
    print(i, end=" ") # end=" " avoids the new line after each print()
```

```
0 1 2 3 4 5
```

Loops

Repeat the same block of code over and over again

- While-loops: repeat the code *while some condition is True*

```
while True:
    print("I will run forever!")
```

```
I will run forever!
I will run forever!
I will run forever!
I will run forever!
I will run forever!
I will run forever!
I will run forever!
I will run forever!
I will run forever!
I will run forever!
I will run forever!
I will run forever!
```

Loops

For-Loop

Needs an *iterable* object or a *generator*

- Iterables are `list`s, `tuple`s, `set`s or the keys of a `dict`, also the characters in a `str`
- Generators can be thought of as special "functions" that return a new value each time they are called
  - Probably most popular: `range(start, stop[, step])` generator
  - Generates all numbers from `start` up to, but not including, `stop` (`step` is an optional parameter setting the difference between two consecutive elements, default is 1)

Loops

For-Loop

```
for element in iteratable_or_generator:
    # code block, note that this needs to be indented again!
```

Example:

```
for i in range(0,10):
    print(i, end=" ")
```

```
0 1 2 3 4 5 6 7 8 9
```

(Note that 10 is not part of the loop!)

It's Your Turn!

- Continue reading the "For-Loops" section of the notebook

Do the exercise!

Exercise 2: Implement a for-loop running up to an arbitrary number which outputs the number of the running variable and outputs a message whether the number is odd or even.

It's Your Turn!

Consider yesterday's Exercise about the *Flat Shopping List*

```
1   shopping_list = (('Paprika', 'Tomatoes', 'Fennel'),
2                    ['Oranges'],
3                    'Bananas', 'Apples', 'Milk', 'Yoghurt', 'Butter')
4   flat_list = []
5   flat_list.extend(shopping_list[0])
6   flat_list.append(shopping_list[1][0])
7   flat_list.extend(shopping_list[2:])
8   print(flat_list)
9   print(len(flat_list))
```

Write a loop that does this job for you, i.e. ***replace lines 5-7***

Hint: You might need to check if you are dealing with a `list` or `tuple`, e.g. as

```
if type(element) is tuple
```

Loops

While-Loop

These loops run as long as their loop condition evaluates to `True`

```python
while condition:
    # indented code block
    # that is executed as long as bool(condition) == True
```

Loops

While-Loop

```python
i = 0
while i < 10:
    print(i, end=" ")
    i += 1 # shorthand notation for `i = i + 1`
```

```
0 1 2 3 4 5 6 7 8 9
```

Loops

While-Loop

```python
import numpy as np

def roll_the_dice():
    eyes = np.random.randint(1,7)
    return eyes

attempts = 1
while roll_the_dice() != 6:
    attempts += 1

print("Needed", attempts, "attempts to roll a 6")
```

```
Needed 10 attempts to roll a 6
```

Loops

Loop Control

There are two special keywords to control loop behaviour

- `break` ends the loop immediately

```python
for i in range(0,10):
    if i == 7:
        break

    print(i, end = " ")
```

```
0 1 2 3 4 5 6
```

Loops

Loop Control

There are two special keywords to control loop behaviour

- `continue` ends the *current iteration* immediately and starts with the next iteration

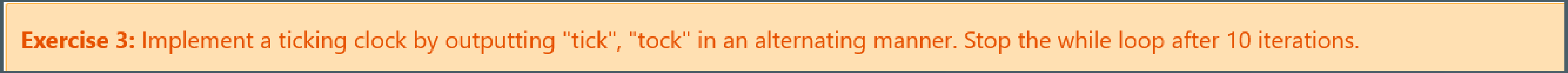
```
for i in range(0,10):
    if i == 7:
        continue

    print(i, end = " ")
```

```
0 1 2 3 4 5 6 8 9
```

It's Your Turn!

- Continue reading the "While-Loops" section of the notebook

Do the exercise!



Exercise 3: Implement a ticking clock by outputting "tick", "tock" in an alternating manner. Stop the while loop after 10 iterations.

Go to our Moodle page (https://moodle.uni-greifswald.de/course/view.php?id=9565) and take the third quiz! ("Quiz 4 - Loops")

Congratulations

You have learned the most fundamental concepts of programming in Python

- Variables and built-in types (simple and advanced)
- Functions
- Conditions and control flow
- Loops

With that, you can start writing meaningful Python programs!

But there is more...

Next Up

- Functions in more detail
- Scopes
- File I/O
- Important modules