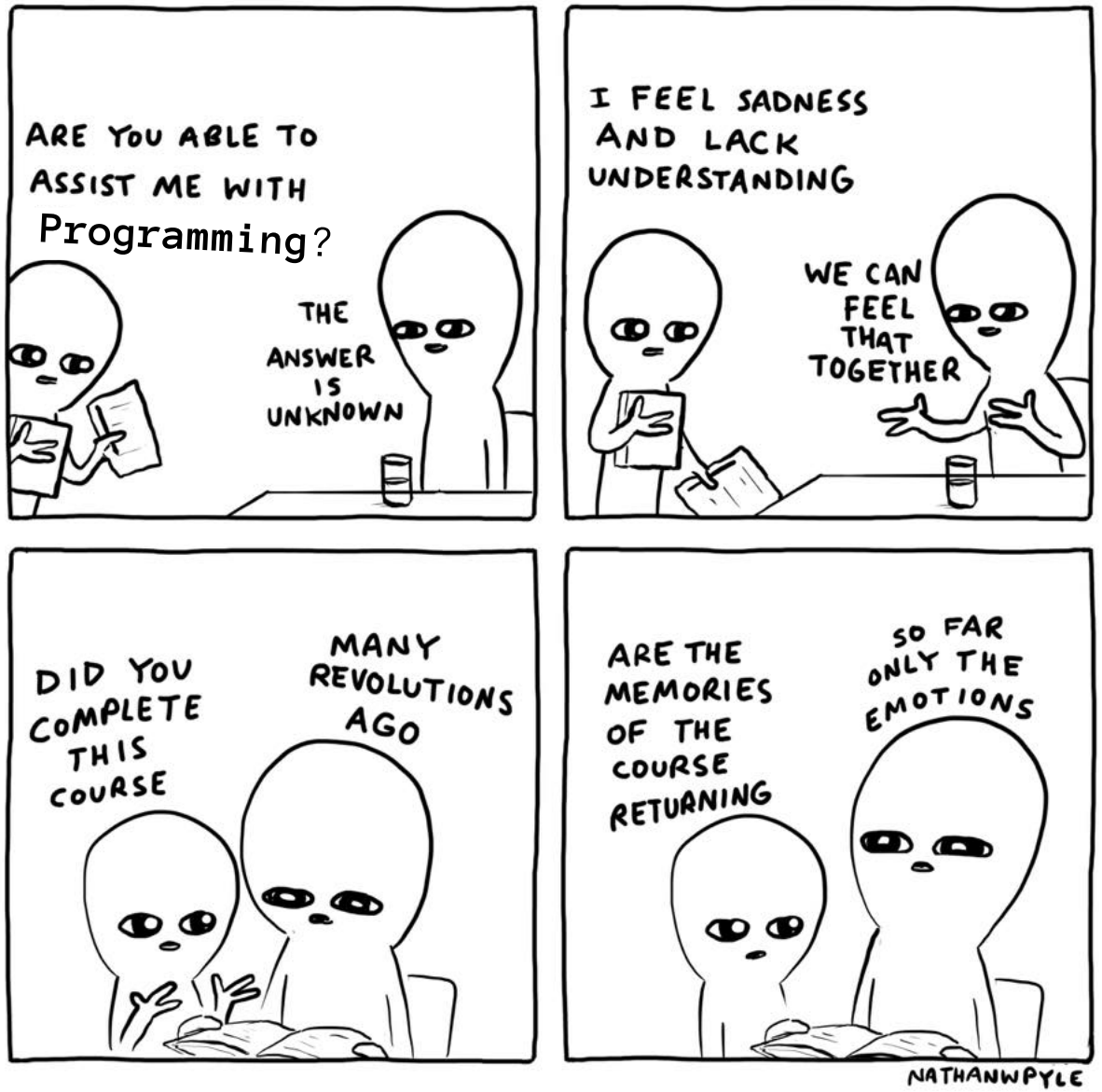

ARE YOU ABLE TO ASSIST ME WITH Programming?
THE ANSWER IS UNKNOWN
I FEEL SADNESS AND LACK UNDERSTANDING
WE CAN FEEL THAT TOGETHER
DID YOU COMPLETE THIS COURSE
MANY REVOLUTIONS AGO
ARE THE MEMORIES OF THE COURSE RETURNING
SO FAR ONLY THE EMOTIONS
NATHANWPYLE

Introduction to Programming (DDIT & GBI)

August 28, 2019

Lecturers:
Helge Pfeiffer, ropf@itu.dk
Martin Aumüller, maau@itu.dk


BACK
to
SCHOOL

Overview

- Organizational
  - Who are we
  - Content
  - Organization and Style
  - Credits
  - Mandatory Assignments
  - Exam
- Content
  - A brief introduction to programming and Python

Who are we?

- Lecturers
  - Martin Aumüller (maau@itu.dk), 4B05
  - Helge Pfeiffer (ropf@itu.dk), 4C03

- Teaching Assistants

Who are you? (CHANGE TO MENTIMETER)

1. Who of you went to school in Denmark?
2. Who of you has programming experience?
3. Who of you thinks programming is difficult?

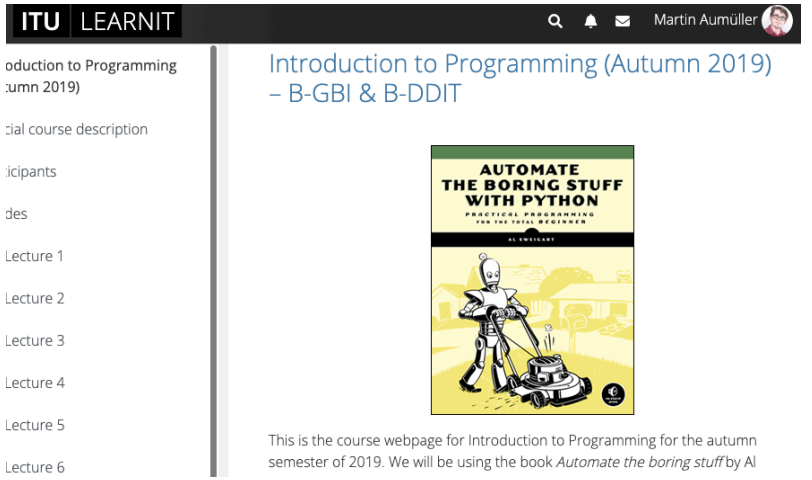
Contents of the course (Intended learning outcomes)

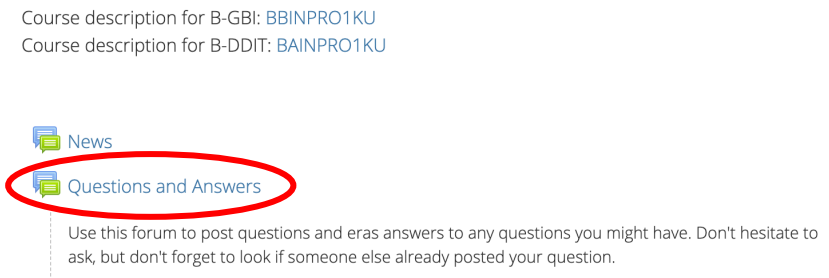
After the course, the student is able to

- **analyse a** given, simple computational **task** such as manipulation of a text-based database or external hardware device **to the extent of designing a programmatic solution** and **implementing** it in a modern, text-base, domain-neutral programming language
- **test** the **correctness** of a piece of code
- **write** program **documentation**
- **reason about** the computational **complexity** of an algorithm
- **express functionality** in terms of abstract data type or application programming interface
- **use text-based tools** of program developement, including an editor and command-line tools

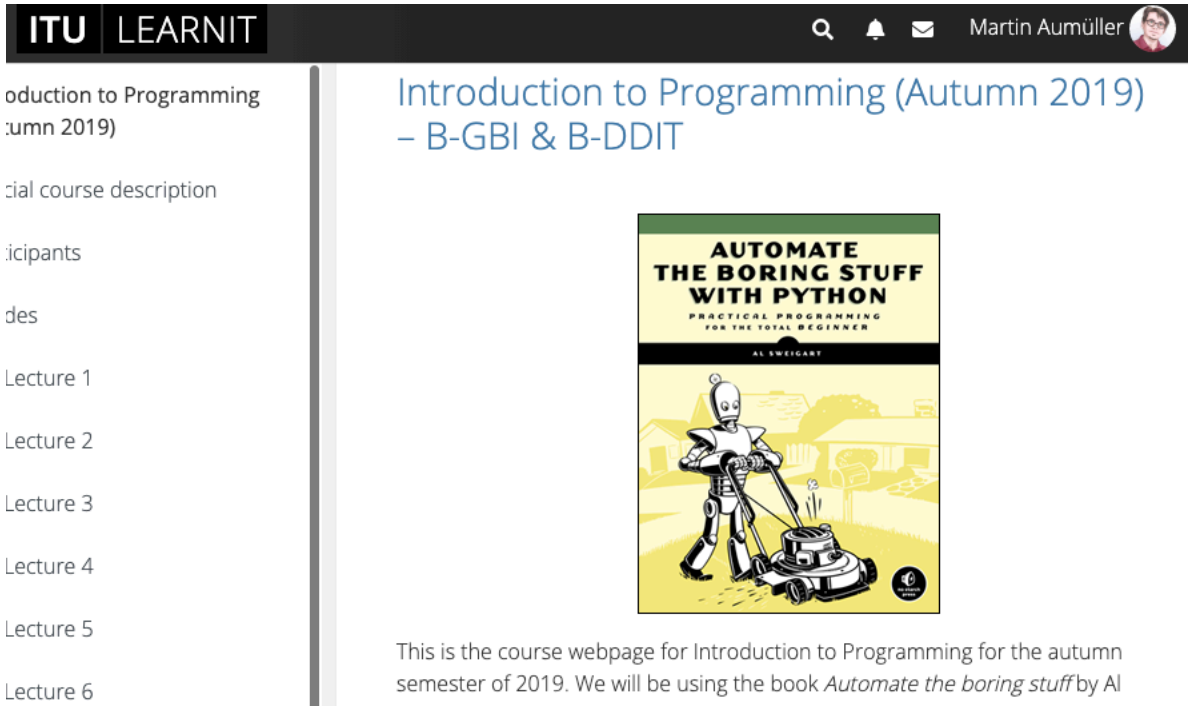
Organization and Style

- Course webpage on learnIT (https://learnit.itu.dk/course/view.php?id=3018513)
- 14 lectures
- use the forum on learnIT for questions of all kinds
- 10-12: Lecture on the topic of the week
- 12-14: Exercise sessions
- Screencasts of lectures available on learnIT





Course webpage



https://learnit.itu.dk/course/view.php?id=3018513

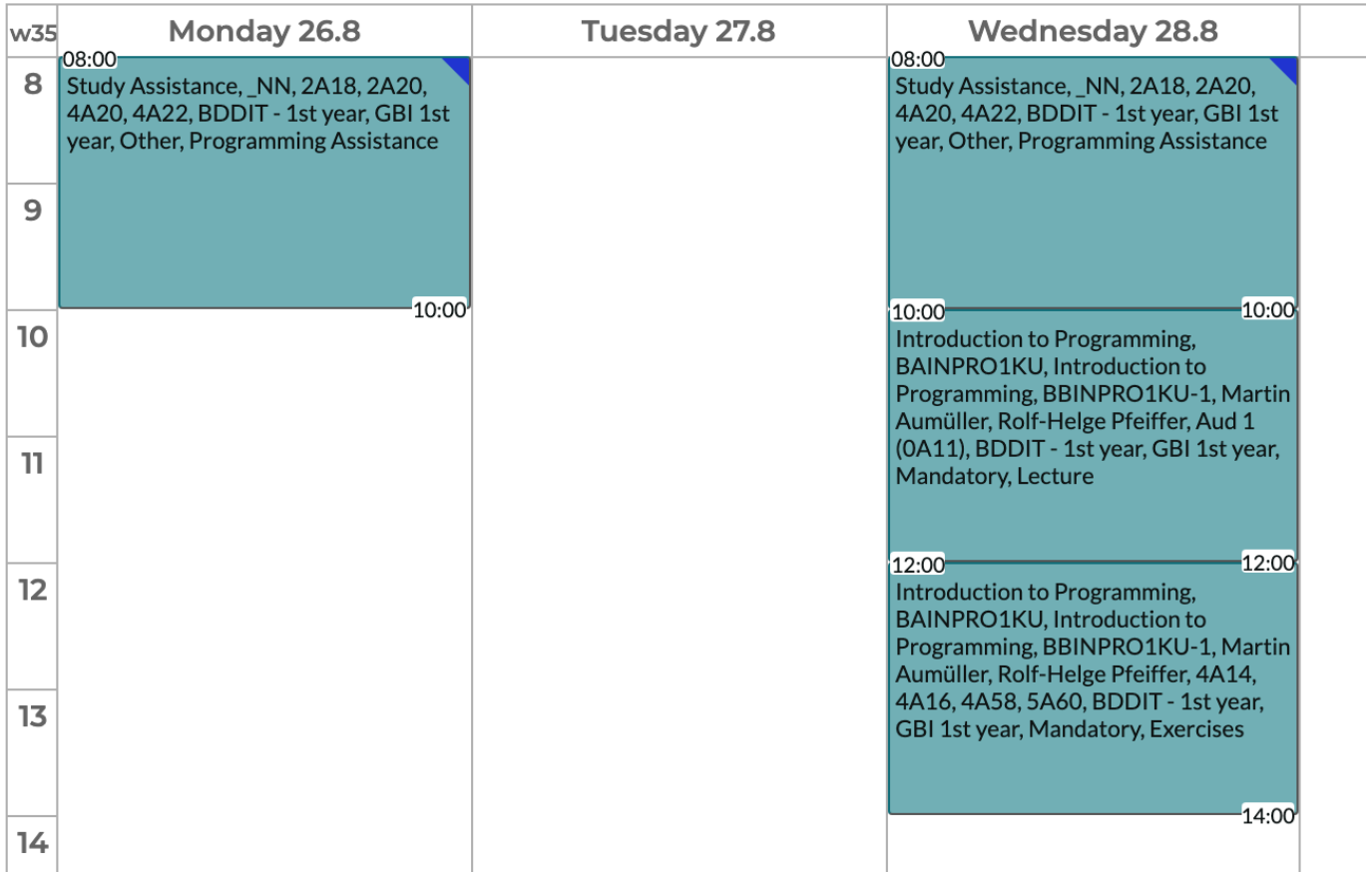
Exercises

- You work in groups
  (see groups on learnIT)
- Each room has TAs that will help you with all of your questions

Structure of exercises

- no blackboard-style instruction
- you work on solving exercises, usually from the book
- work in your group
- try to find help in your group, if this fails contact a TA
- solutions are made available after the exercise session

Additional help through Study Assistance



Start in Week 3!

Book

- We will be using *"Automate the boring stuff"* by Al Sweigart
- Free to read: https://automatetheboringstuff.com/
- Follow progression relatively closely
- Rule of Thumb: One chapter per week
- Additional material at the end

Credits

- 7.5 ECTS translate to 11.5 hours/week
- 4 hours lecture/exercises
- Whole work-day from 9am to 5 pm (excl. lunch) on your own!
- Preparing for the exam will take some time as well

Mandatory assignments

- During the semester, **5 programming assignments** will be set
- **At least 4** of these **must be approved** to qualify for the exam
- You are **allowed to work in the groups**, but **hand-in is individual**

- Assignment is posted two weeks before it is due
- Handing in solution on learnIT, always on a Monday, 23:59
- Details will be discussed when the first hand-in is posted
- Current plan: Hand-in on Sept 16, Sept 30, Oct 21, Nov 4, Nov 18

Exam

- Written, individual exam on January 8, 2019
- 4 hours exam on premises at computers not connected to the internet.
- Test exams on learnIT

- **Focus**: Show that you learned the technical skills to solve programming tasks

Questions?

Introduction to Programming

(for real)

Why learn programming?

Career Prospects

The competencies of Global Business Informatics
perspective of the programme prepares you for a
Danish and global/international context, working

- Project coordinator
- Process analyst
- Quality professional
- IT architect
- Support Chain Manager
- Business developer

https://en.itu.dk/programmes/bsc-programmes/global-business-informatics

Career Prospects

The competences you acquire during your studies for the BSc in Digital Design and Interactive Technologies and the global perspective of the programme prepare you for a career in both a Danish and an international context. The courses in the programme support and develop your competences in subjects that are attractive on a global job market, for example Scandinavian approaches to co-design, and democratic media formats and work organisations.

You can work as e.g.:

- Interaction designer
- Usability consultant
- Information architect
- Project coordinator

https://en.itu.dk/programmes/bsc-programmes/digital-design-and-interactive-technologies

Your new super power! (automation & speed up)

Prerequiste for technical courses at ITU

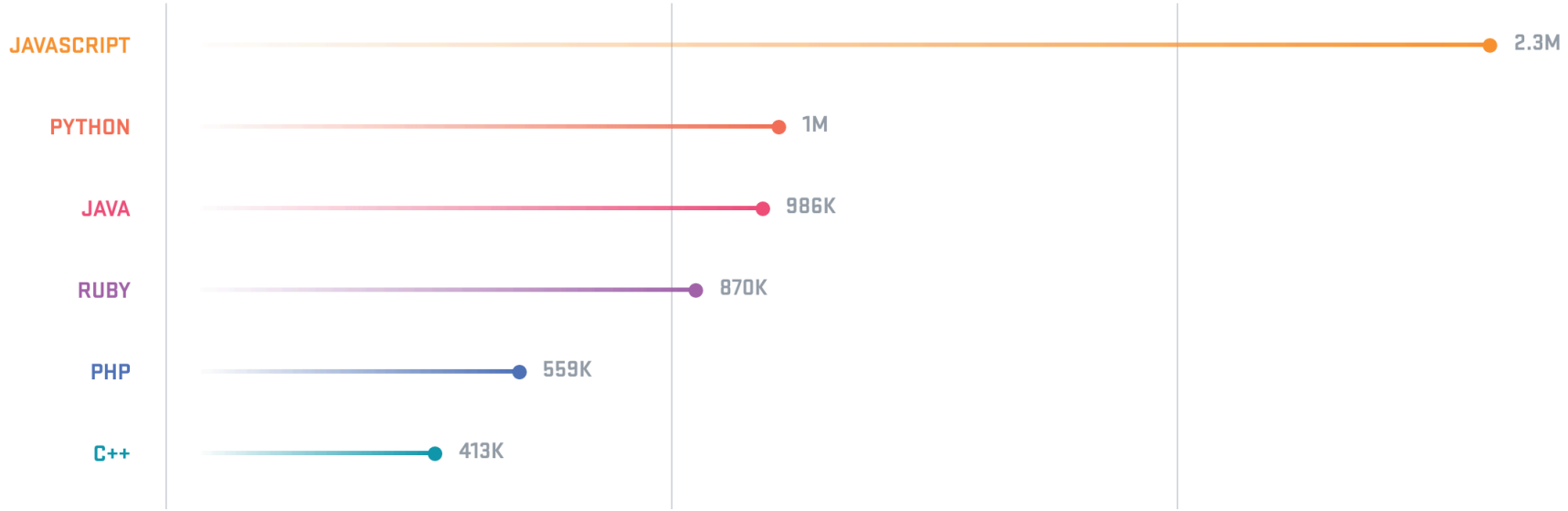
Understand possibilities & limitations of IT systems

Why learn Python?

The fifteen most popular languages on GitHub

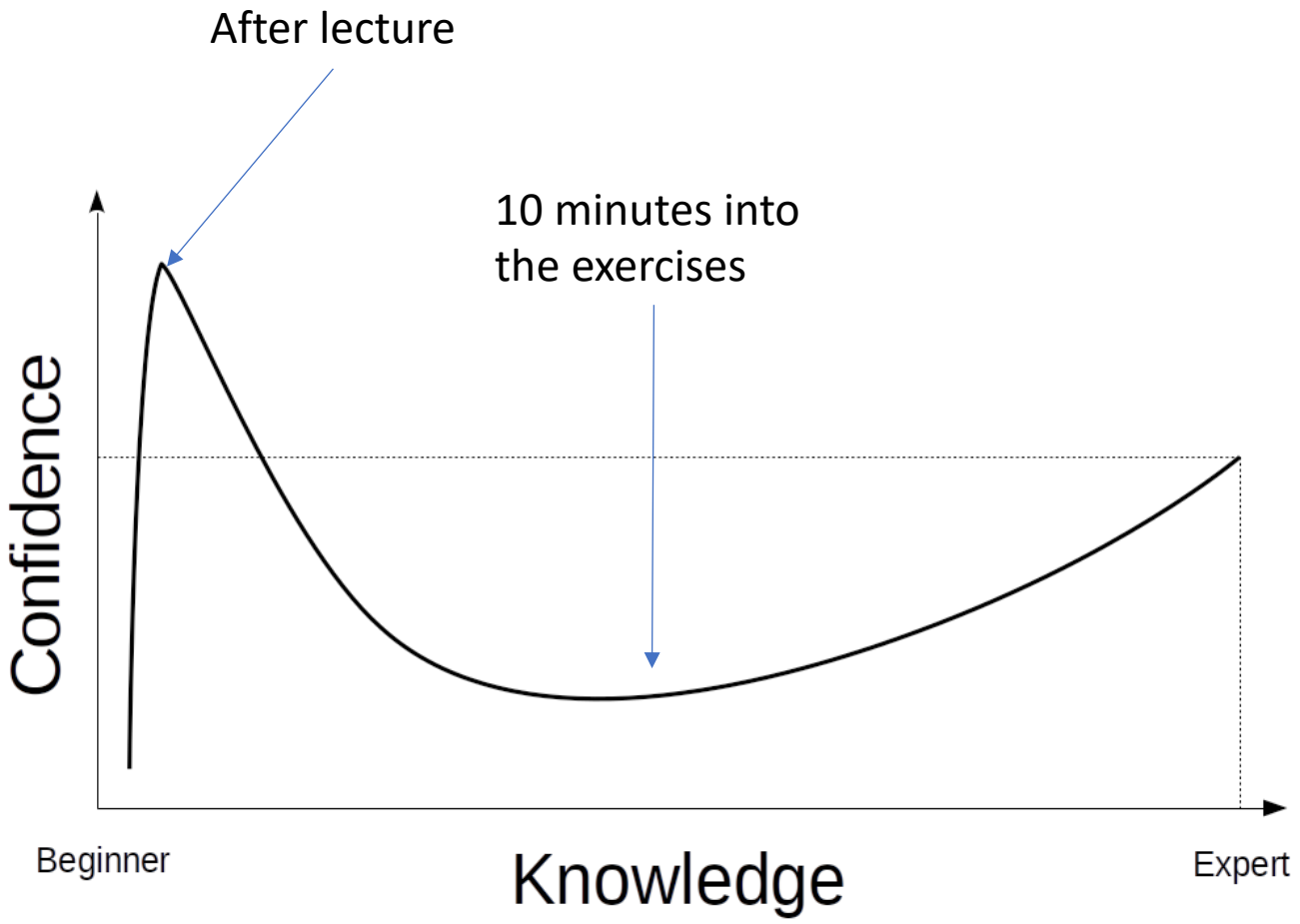
by opened pull request

GitHub is home to open source projects written in 337 unique programming languages—but especially JavaScript.



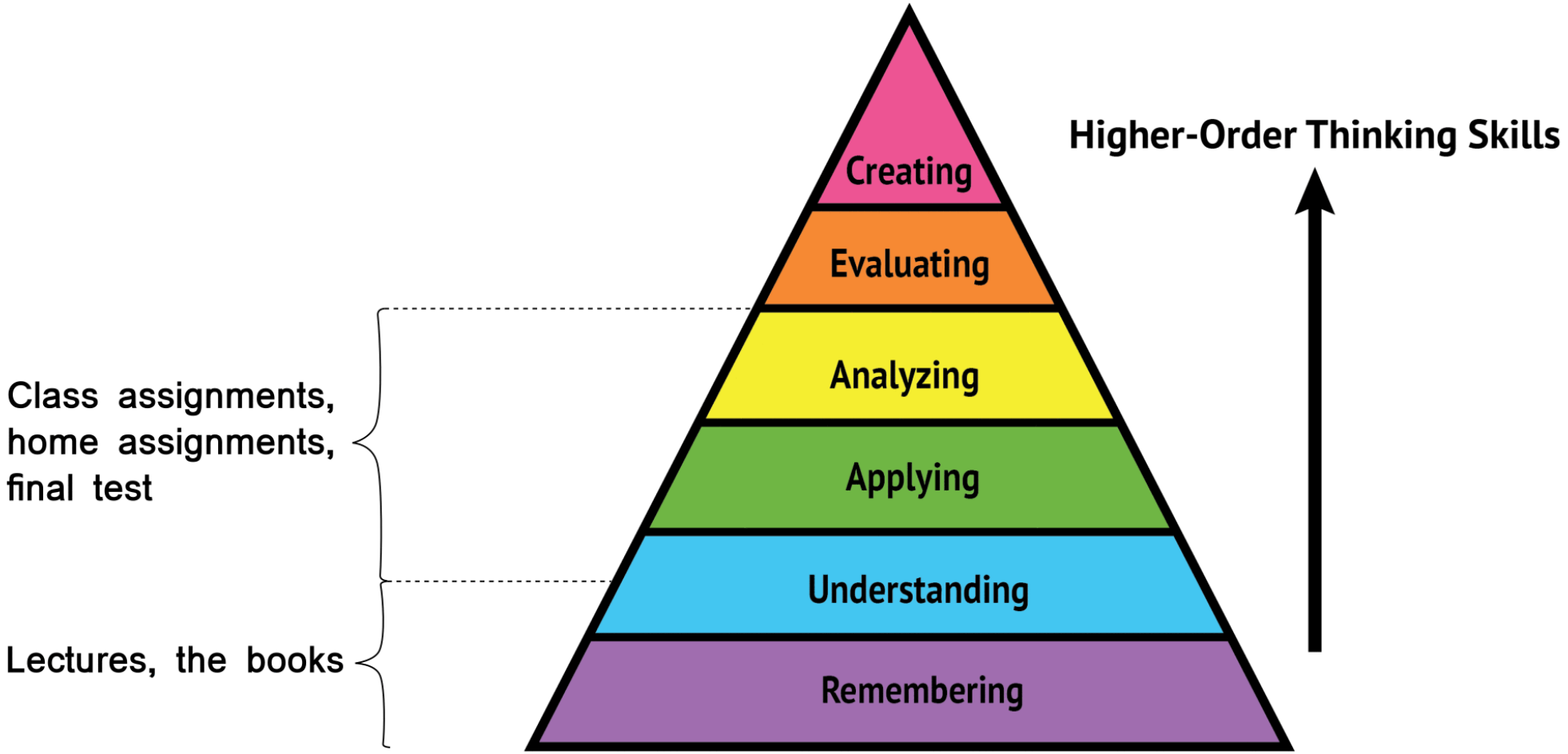
https://octoverse.github.com/

Beware: It’s difficult!



Beware: It’s frustrating

Bloom's taxonomy



What is programming?

- Programming is the act of entering instructions for the computer to perform
- Building blocks: **Basic instructions**, for example (in English):
  - “Do this; then do that”
  - “If this condition is true, perform this action; otherwise, do that action”
  - “Do this action a number of times”
  - ”Keep doing this until this condition is true”

An example

```
1   passwordFile = open('SecretPasswordFile.txt')
2   secretPassword = passwordFile.read()
3   print('Enter your password.')
4   typedPassword = input()
5   if typedPassword == secretPassword:
6       print('Access granted')
7   else:
8       print('Access denied')
```

- Read from top to bottom
- Even without prior programming experiences, you can probably make a reasonable guess what this code is doing!

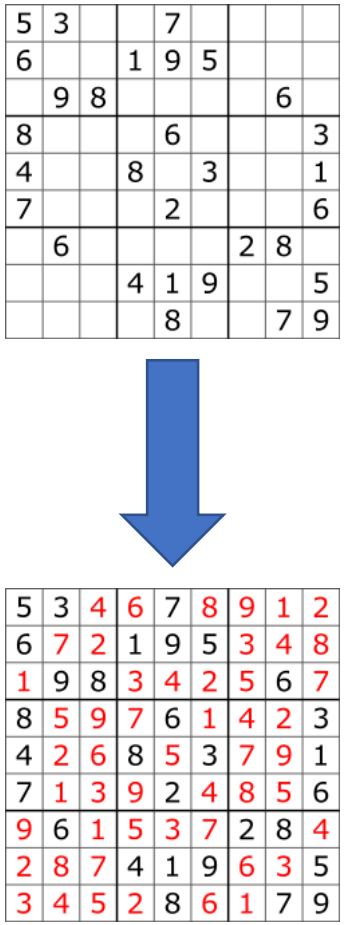
An example

```
1   passwordFile = open('SecretPasswordFile.txt')
2   secretPassword = passwordFile.read()
3   print('Enter your password.')
4   typedPassword = input()
5   if typedPassword == secretPassword:
6       print('Access granted')
7   else:
8       print('Access denied')
```

1. Open the file “SecretPasswordFile.txt”, this is stored on a computer
2. Read the secret password from the file
3. Prompt the user to enter his/her password
4. Store the typed in password
5. The program checks whether the two passwords match
6. It prints “Access granted” on the screen if they do
7. Otherwise..
8. It prints “Access denied”

Programmers don’t need to know much Math

- Most programming doesn’t require Math skills beyond basic arithmetic
- It’s more like solving a Sudoku puzzle
- Just because it involves numbers doesn’t mean you have to be good at Math
- Both Sudoku and Programming involves breaking down a problem into individual, detailed steps.
- Like for all skills: The more you do it, the better you will become



It’s a creative task!

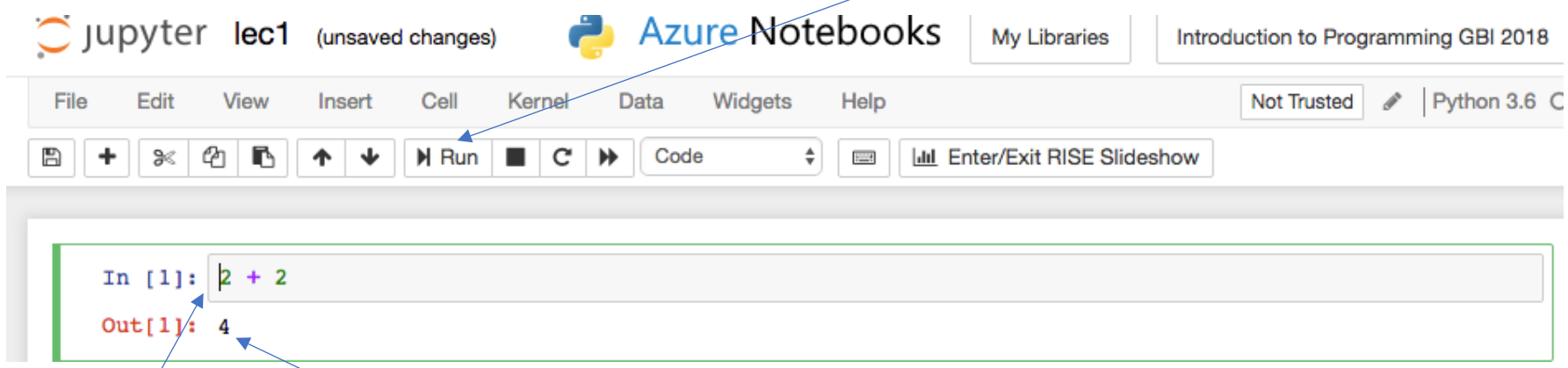
(but you don’t need to go to the store to buy new bricks, everything is already in your computer)

Python Programming Basics

Running Python Programs

- Running Python programs requires
  - A working Python installation
  - An editor to write your Python code
- Goal of first exercise session
  - Get a working Python installation and write your first program
- In class
  - Editor is ‘Jupyter notebook’
  - comes with great advantages, but many disadvantages for the beginner

What we will do



2. Click on Run.

1. Enter expression in the shaded area.

3. Result is displayed under the cell, after Out[…]:

What you *should* do

- Use the **Mu editor**

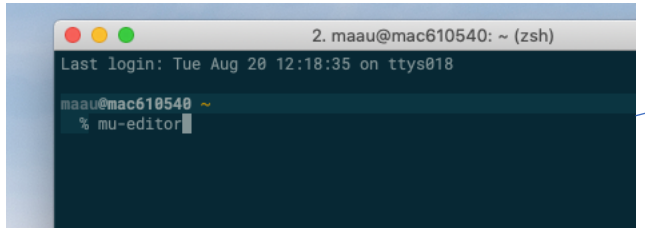
https://codewith.mu/

Install as described on learnIT!

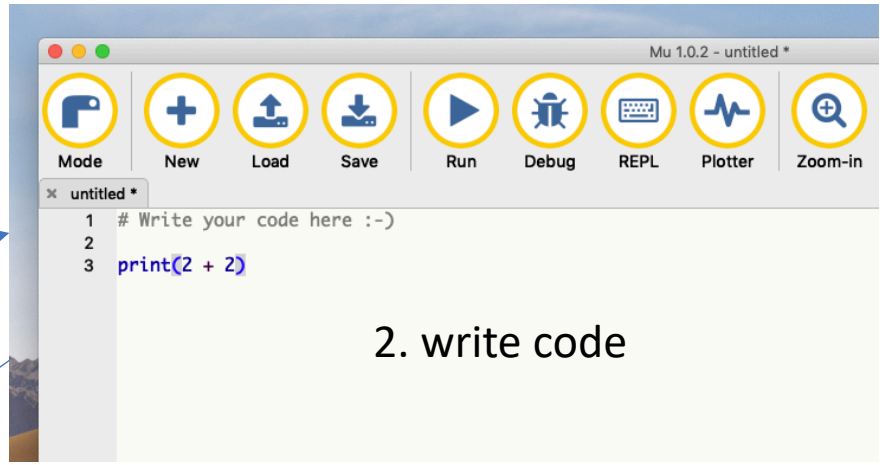
- If you get used to programming
  - choose whichever editor you like
  - we will probably switch to VSCode

https://code.visualstudio.com/

Workflow with Mu

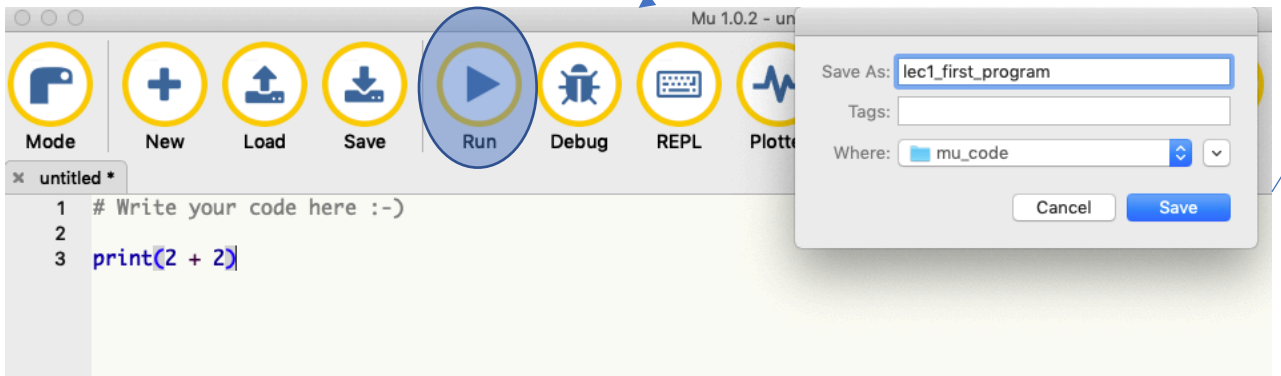


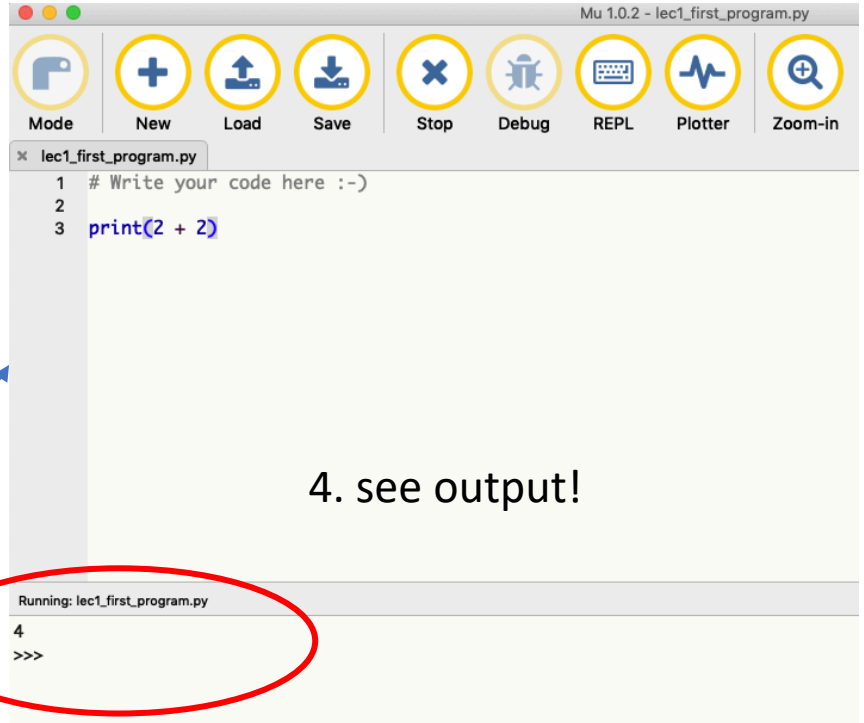
1. start mu-editor from terminal (see learnIT)



2. write code

3. click run, save document





4. see output!

Why different systems

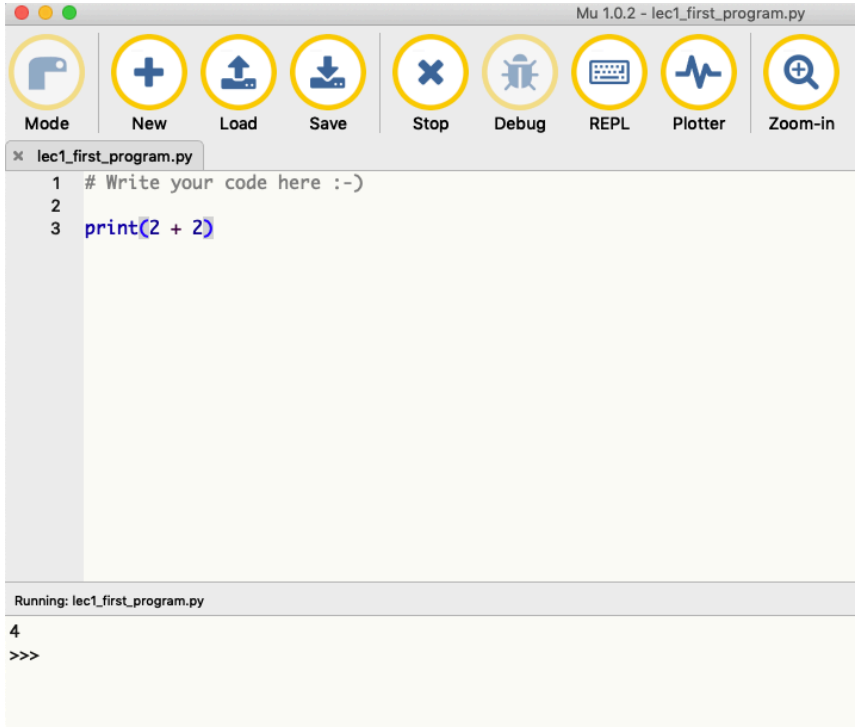
- Programming in Jupyter notebooks is the not “real programming” that we want to teach you!
- It’s:
  - difficult to share programs
  - difficult to find errors in programs
  - difficult to explain certain concepts
- However:
  - It’s easy to develop step-by-step programs, and share the whole path until you get to result

An important difference

- Jupyter notebook ‘prints automatically’

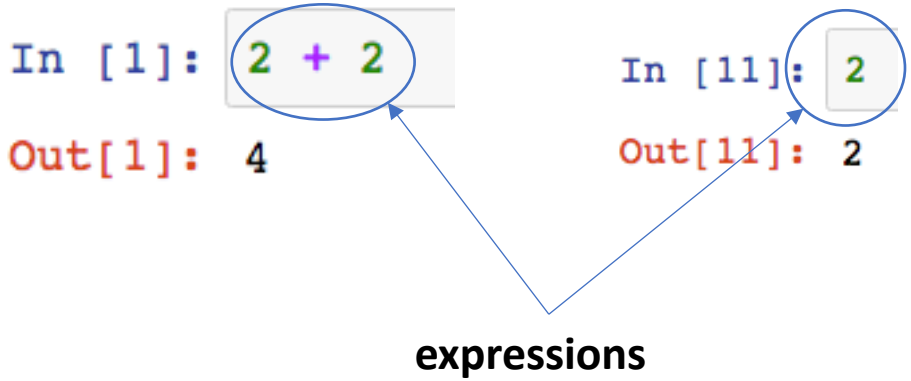
```
In [1]: 2 + 2

Out[1]: 4
```

- In **mu-editor** you have to say **print(…)** to print the output



Python 101: What is an Expression?

- ”2 + 2” is called an **expression**
- Expressions consist of values (here, 2) and operators (here, +)
- Expression can be **evaluated down to a single value**
- Even just writing “2” is an expression, consisting of one value, and no operator.



Using Python as a Calculator

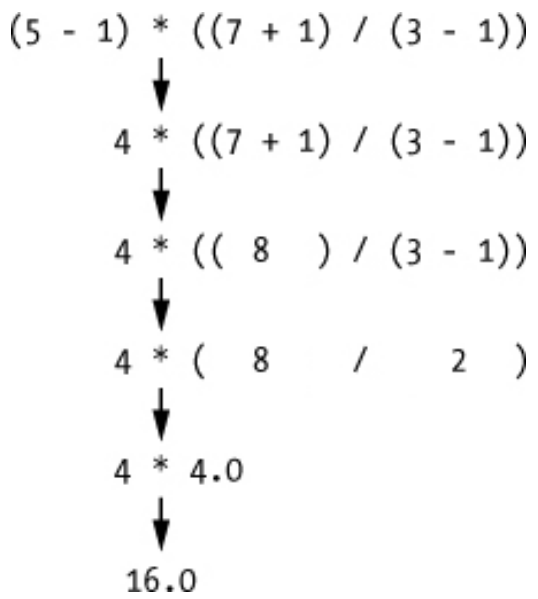
A "#" starts a comment, which is used to explained code parts.

Operator	Operation	Example	Evaluates to...
**	Exponent	2 ** 3	8
%	Modulus/remainder	22 % 8	6
//	Integer division/floored quotient	22 // 8	2
/	Division	22 / 8	2.75
*	Multiplication	3 * 5	15
-	Subtraction	5 - 2	3
+	Addition	2 + 2	4

```
In [1]: 2 + 2 # addition
Out[1]: 4

In [3]: 2 + 3 * 6 # addition and multiplication
Out[3]: 20

In [4]: 124890 * 12415124125 # multiplication of two large numbers
Out[4]: 1550524851971250

In [5]: 2 ** 6 # Compute 2 to the power of 6, i.e., 2^6 = 2 * 2 * 2 * 2 * 2 * 2
Out[5]: 64

In [6]: 23 / 7 # Divide 23 by 7
Out[6]: 3.2857142857142856

In [7]: 23 // 7 # Divide 23 by 7, round down to an integer number
Out[7]: 3

In [8]: 23 % 7 # What is the remainder of dividing 23 by 7?
Out[8]: 2

In [9]: 2          +          2 # Number of spaces don't carry meaning
Out[9]: 4

In [10]: (5 - 1) * ((7 + 1) / (3 - 1)) # a more complex expression
Out[10]: 16.0
```

“Evaluated down to a single value?”

```
In [10]: (5 - 1) * ((7 + 1) / (3 - 1))
Out[10]: 16.0
```



- Order of operations (*precedence*):
  - **, *, /, //, %, +, -
  - From left to right
- Use parentheses to override the standard precedence

Errors are okay!

- Programs “crash” if they contain code that the computer doesn’t understand
- Expressions have to be formulated syntactically correct
- Just like in English:
  - “This is a grammatically correct English sentence” vs.
  - “This grammatically is sentence not English correct a”.

```
In [12]: 2 + 3 3
  File "<ipython-input-12-d03b490806c2>", line 1
    2 + 3 3
          ^
SyntaxError: invalid syntax

In [13]: 1 * *
  File "<ipython-input-13-7315209b7185>", line 1
    1 * *
        ^
SyntaxError: invalid syntax
```

String concatenation and replication

- We discussed operators such as “+” and “*” for values that belong to the data type **integers**.
- Straight-forward: Their behavior for **float.**
- But what do they do if string values are involved?

Operator	Values	Example
+ (concatenate)	Two Strings	‘Hello ‘ + ‘Kitty’ is ‘Hello Kitty’
* (replicate)	One Integer, One String	3 * “Kitten” is “KittenKittenKitten”

```
In [17]: 3 * 'Kitten' + 2
---------------------------------------
--
TypeError
t)
<ipython-input-17-9d5745293635> in
----> 1 3 * 'Kitten' + 2

TypeError: must be str, not int
```

”+”-Operator is not defined for one integer and one string (TypeError).

Data types

- We have seen “values”
- Data types put values into different categories
- Every value belongs to exactly one data type

Both ‘cat’ and “cat” are (the same) strings.

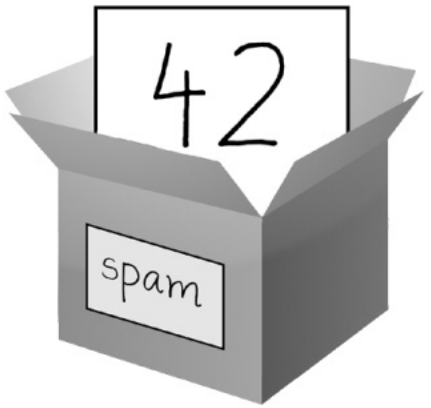
Data type	Examples
Integers (int)	-2, 1, 0, 12314, 20
Floating-point numbers (float)	-1.0, 0.0, 3.14159
Strings	‘a’, ‘one’, “zero”, ‘0’, ‘cat’

```
In [14]: 'This is invalid
  File "<ipython-input-14-91442feb44bd>", line 1
    'This is invalid
                    ^
SyntaxError: EOL while scanning string literal
```

Don’t forget to end a string!
(‘ is missing)
Don’t mix quotation mark types unintentionally.

Storing Values in Variables

- What if we want to re-use values that we have evaluated before?
- That's what we use **variables** for
- Analogy: A box in your computer memory that can store a value
- Storing a value in a variable is achieved with an assignment statement.
- Syntax: "variable name = value"

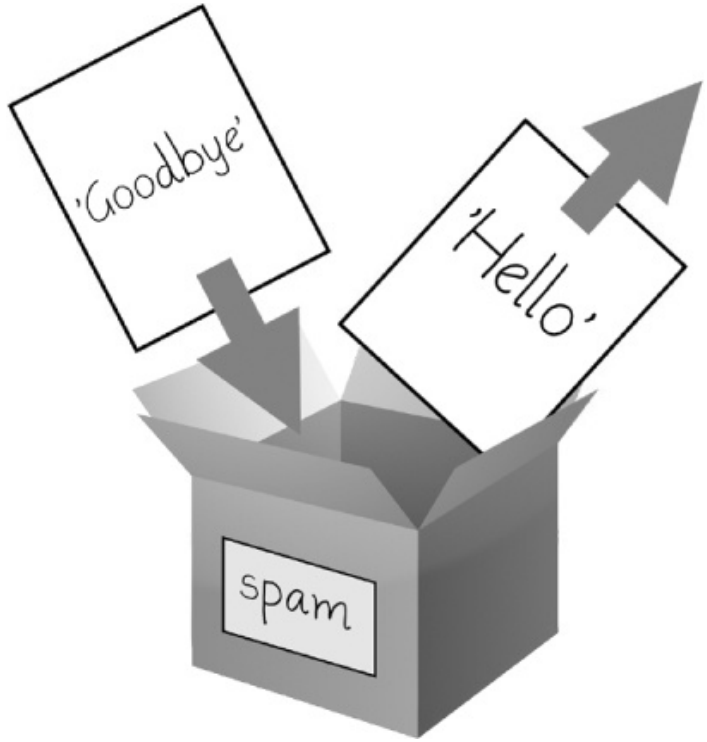



```
In [18]: spam = 42

In [19]: spam
Out[19]: 42

In [20]: spam + 2
Out[20]: 44
```

Overwriting Variables

- “Hello” (string) is stored in variable spam
- “Goodbye” is stored in variable spam, “Hello” is forgotten

```
In [21]: spam = 'Hello'

In [22]: spam
Out[22]: 'Hello'

In [23]: spam = 'Goodbye'

In [24]: spam
Out[24]: 'Goodbye'
```



But: Why would you ever want to overwrite a variable?

Variable Names

- 3 rules a variable name has to obey:
  1. It can only be one word.
  2. It can use only letters, numbers, and the underscore (_) character.
  3. It cannot begin with a number.
- Variable names are case-sensitive, e.g., “spam” and “Spam” are two different variables
- A good variable name describes the data it contains.

Valid variable names	Invalid variable names
`balance`	`current-balance` (hyphens are not allowed)
`currentBalance`	`current balance` (spaces are not allowed)
`current_balance`	`4account` (can’t begin with a number)
`_spam`	`42` (can’t begin with a number)
`SPAM`	`total_$um` (special characters like `$` are not allowed)
`account4`	`'hello'` (special characters like `'` are not allowed)

A first program

```
In [27]: # This program says hello and asks for my name.

         print('Hello world!')
         print('What is your name?')
         myName = input()
         print('It is good to meet you, ' + myName)
         print('The length of your name is:')
         print(len(myName))
         print('What is your age?')
         myAge = input()
         print('Next year, you will be ' + str(int(myAge) + 1) + ".")

         Hello world!
         What is your name?
         Martin
         It is good to meet you, Martin
         The length of your name is:
         6
         What is your age?
         32
         Next year, you will be 33.
```

- Many unknown concepts!
- **Two variables, myName and myAge.** They get a value from something called “input()”.
- We see many strings and some string operations, such as concatenation.
- But what do all the other lines do?

The *print()* function

We will formally introduce a “function” in the third lecture. For now, think about them as something special.

- *print()* displays the **string value** inside the parentheses on the screen, e.g.,
  print(“Hello world!”)
  displays Hello world! on the screen.
- We say that *print* is **called**, the string value in parentheses is **passed** to it.
- Quotes are not printed, they mark the value as being a string value.

```
In [28]: print("Hello world!")
         Hello world!

In [31]: print("Na" * 16 + "Batman!")
         NaNaNaNaNaNaNaNaNaNaNaNaNaNaNaNaBatman!

In [32]: print("Hello " + 42)
         ---------------------------------------------------------
         --
         TypeError                                 Traceback
         t)
         <ipython-input-32-85de8e9ed0c5> in <module>()
         ----> 1 print("Hello " + 42)

         TypeError: must be str, not int
```

Not a string value => TypeError

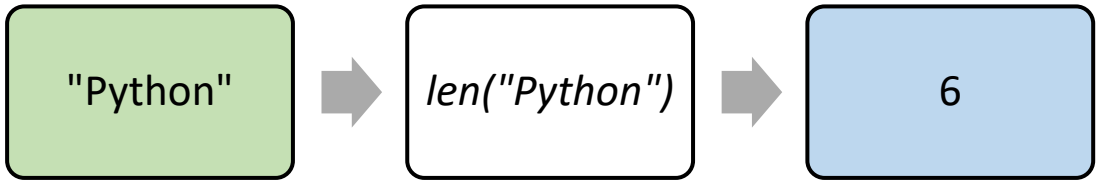
The *input()* function

- The *input()* function waits for the user to input something on the keyboard and press ENTER
- The variable (*name*/*age*) is then assigned this **string value**

```
In [33]: name = input()
         name

         Martin

Out[33]: 'Martin'

In [34]: age = input()
         print(age + 1)

         123

         ---------------------------------
         --
         TypeError
         t)
         <ipython-input-34-8d17ea96f0aa> in
               1 age = input()
         ----> 2 print(age + 1)

         TypeError: must be str, not int
```

The *len()* function

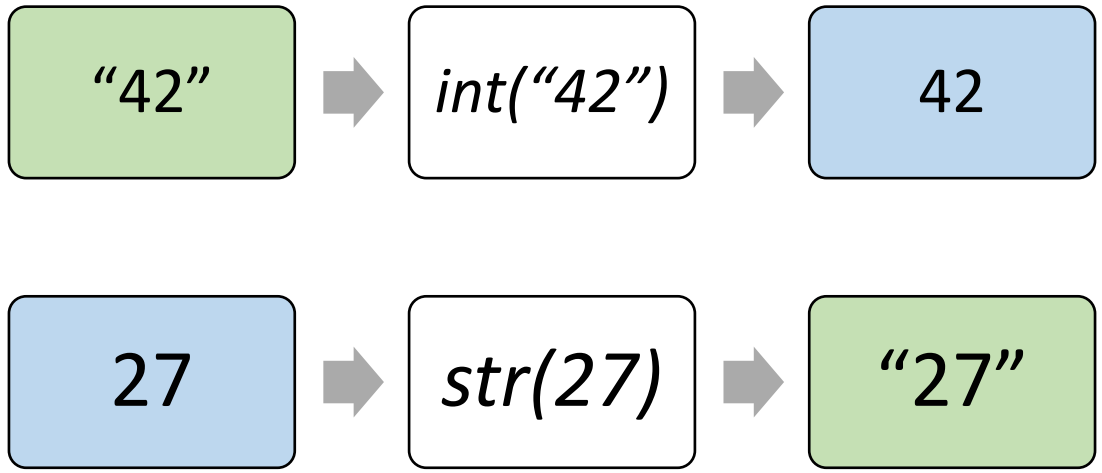
- *len()* expects a **string value** and evaluates to the **integer value** of the number of characters in the string




```
In [35]: len("Python")
Out[35]: 6

In [36]: len("This is a very long string!")
Out[36]: 27

In [37]: len("")
Out[37]: 0

In [38]: len(3.176)
---------------------------------------------
--
TypeError                                 Trace
t)
<ipython-input-38-023fe90b9c61> in <module>()
----> 1 len(3.176)

TypeError: object of type 'float' has no len()
```

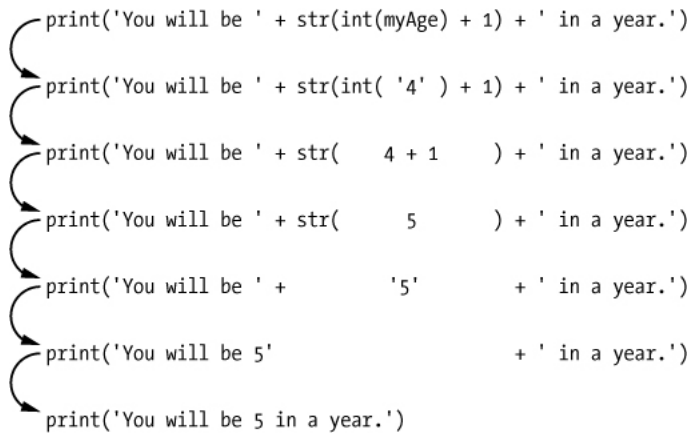
int(), str(), float()

- These functions convert between different data types, e.g.,




```
In [40]: age = "27"
         next_year = int(age) + 1
         print("Next year, you are " + str(next_year) + ".")

         Next year, you are 28.

In [42]: age = "27"
         print("Next year, you are" + (age + 1))

         ---------------------------------------------------------
         --
         TypeError                                 Traceback
         t)
         <ipython-input-42-f630863a99bd> in <module>()
               1 age = "27"
         ----> 2 print("Next year, you are" + (age + 1))

         TypeError: must be str, not int
```

This should explain why we wrote…



Evaluation steps, if *"4"* was stored in *myAge*.
(Slightly different wording in the book.)

Summary

- Expressions as basic building blocks
  - they consist of values and operators
- Values are split up into categories, we introduced Integers, Floating-point numbers, and Strings
- Variables are used to store values for future usage
- Special functions allow to display text, read user input, and convert values from one data type to another

Next: Exercises!

- This week: Sharp start at 12:30
- TAs are going to introduce themselves and the general exercise style
- DDIT/DDK: 4A14-16
- GBI: Groups 1-7: 4A58
- GBI other groups + guest students: 5A60
- **First priority**: Install Python!
- **Second priority:** Try to solve the exercises
- All information on learnIT!