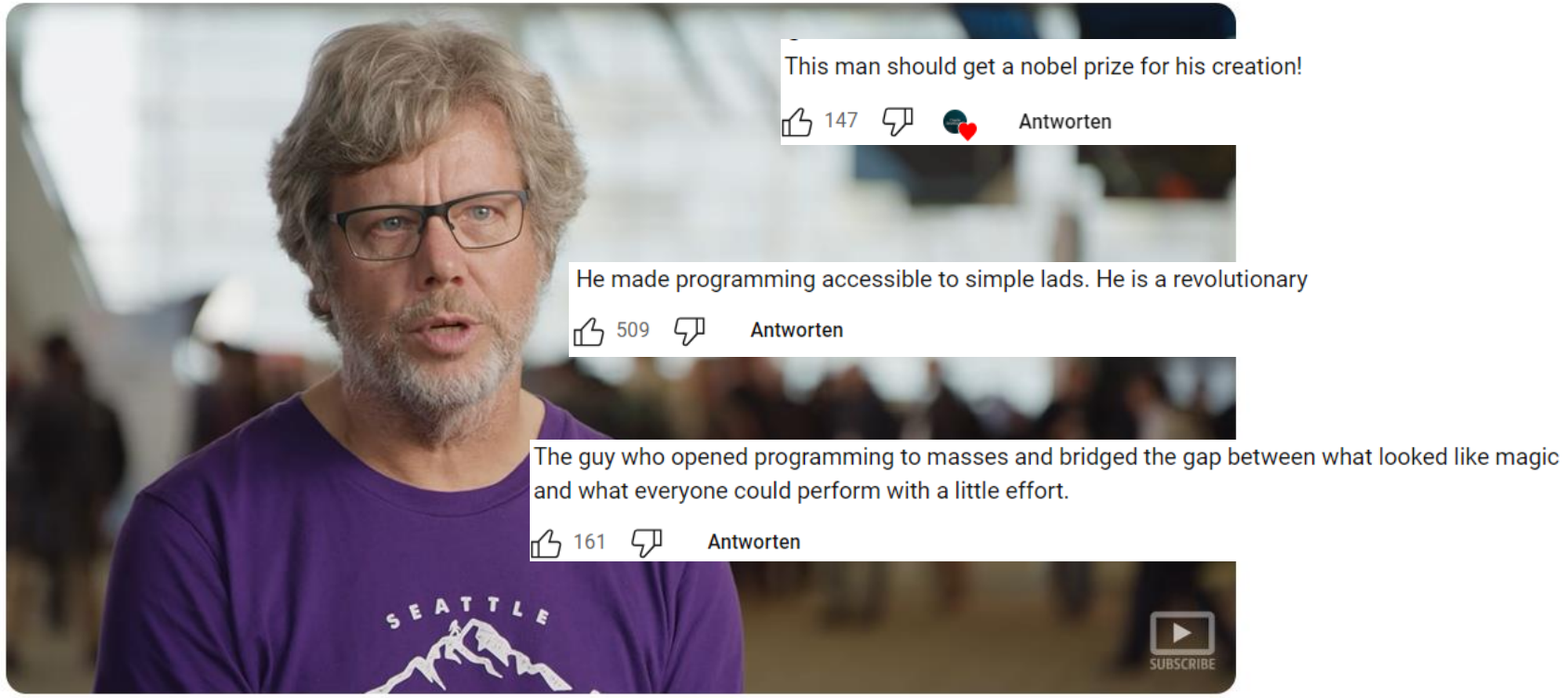
Python I

Programmierkurs 2 Data Science WS23/24

Leonard Traeger
M. Sc. Information Systems
leonard.traeger@fh-dortmund.de

Who is **Guido van Rossum**?



Voraussetzungen

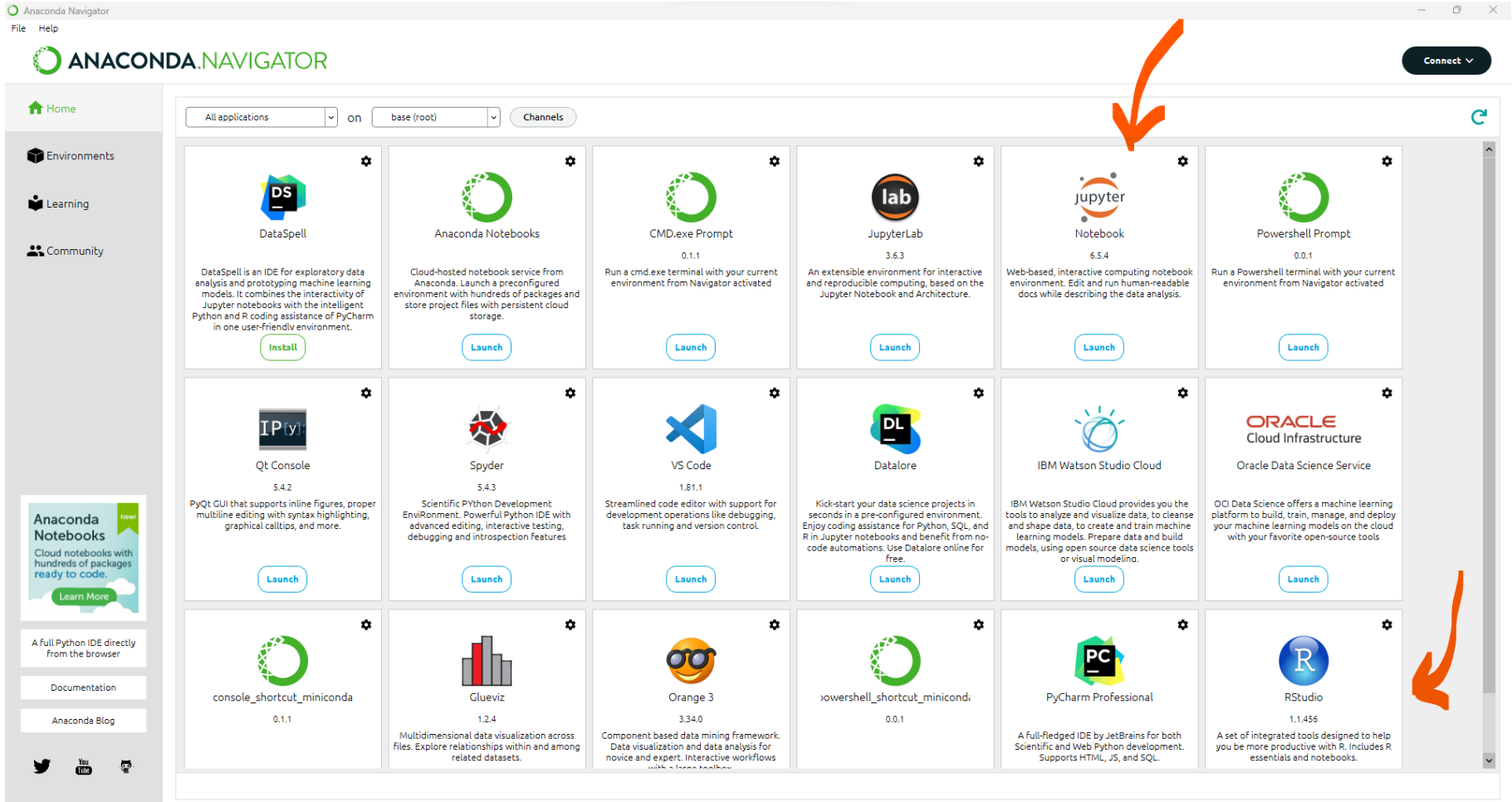
Hardware

- 4GB RAM, 2 CPU-Kerne und 20GB Speicher oder mehr.
- Windows 10, macOS 10.14, oder Ubunutu 14+/Centos7 (in 64-bit) oder neuer.
- Ein Computer mit ausreichender Internetgeschwindigkeit. Sollte das Vorlesungsgeschehen zu einem synchronen Onlineformat wechseln, stellen Sie sicher, dass Ihr Computer über einen Video- und Mikrofonanschluss verfügt.

Software

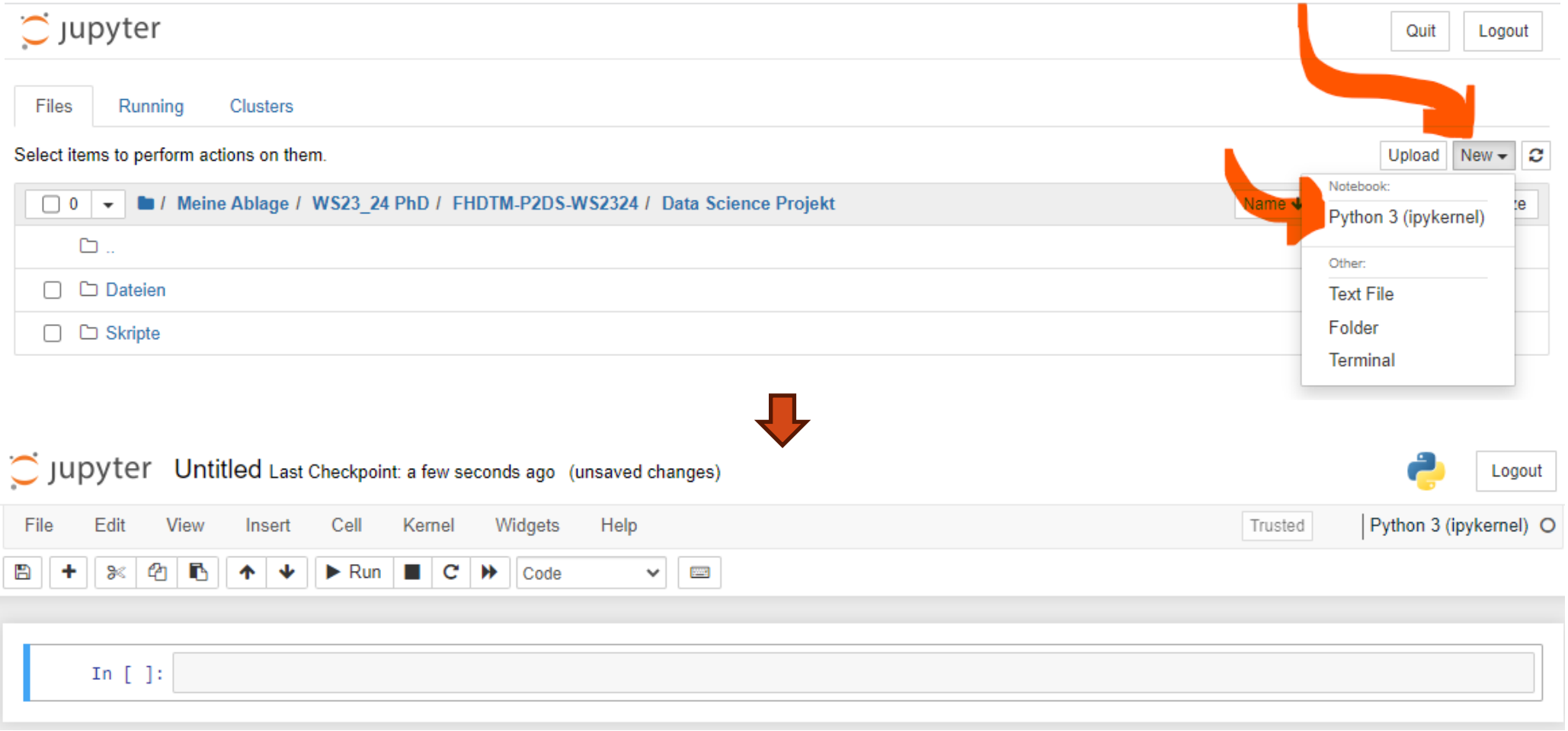
- Webbrowser, der technische Notebooks ausführen kann.
- Jupyter Notebook in Google Collab als Web IDE *https://colab.research.google.com/*
  (Vorteil: in Teams arbeiten, kommentieren, dokumentieren und programmieren).
- Jupyter Notebook als Stand-Alone Installation *https://www.anaconda.com/*
  (Vorteil: Offline arbeiten, RStudio und weitere Data Science Platformen inkludiert):
  Python 3.10 oder neuer und Anaconda.

Anaconda I https://www.anaconda.com/



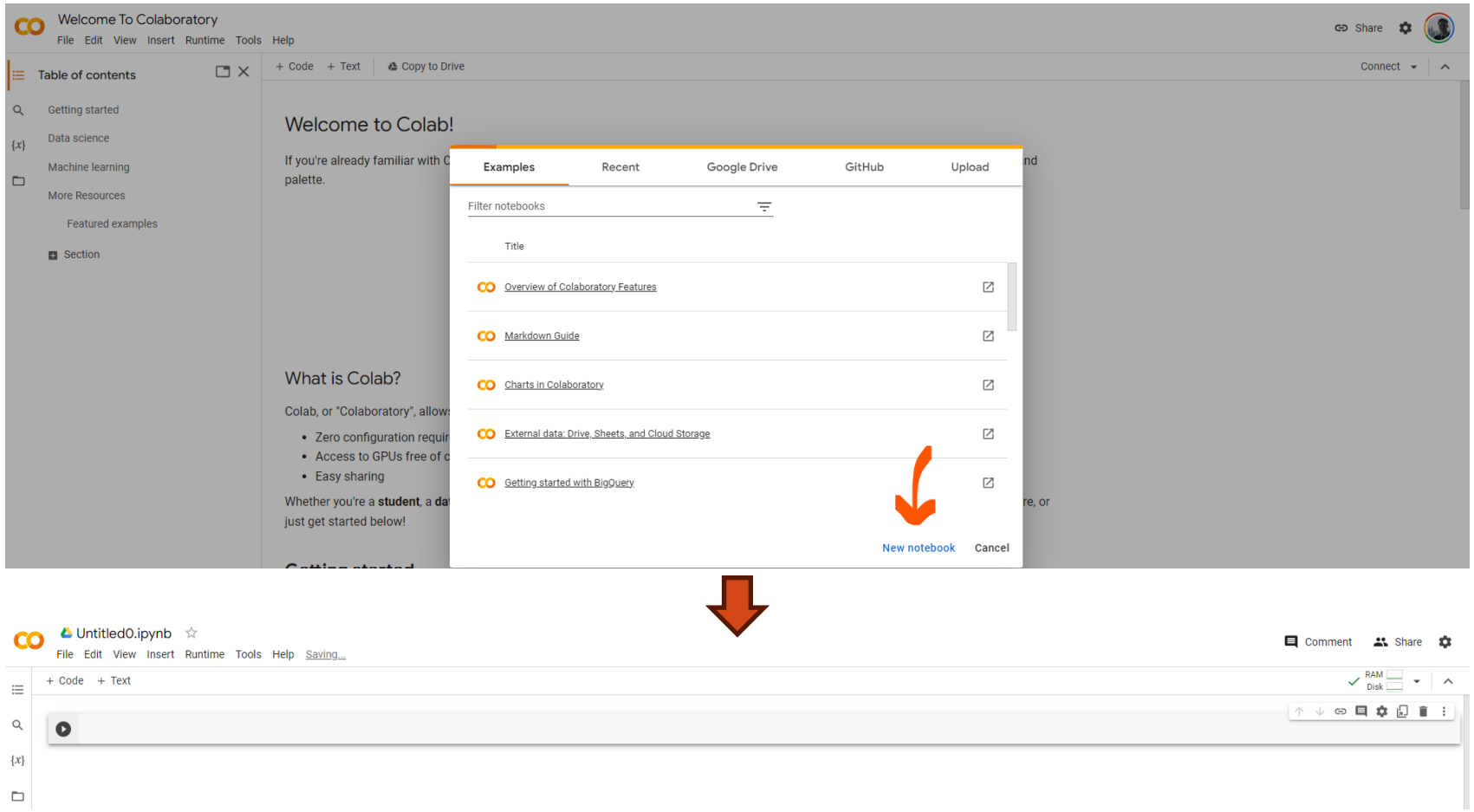
R wird in Woche 10-12 behandelt

Anaconda II


jupyter
Quit
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
0
/ Meine Ablage / WS23_24 PhD / FHDTM-P2DS-WS2324 / Data Science Projekt
Name
Notebook:
Python 3 (ipykernel)
Other:
Text File
Folder
Terminal
..
Dateien
Skripte
jupyter
Untitled
Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3 (ipykernel)
Run
Code
In [ ]:

Colab Research

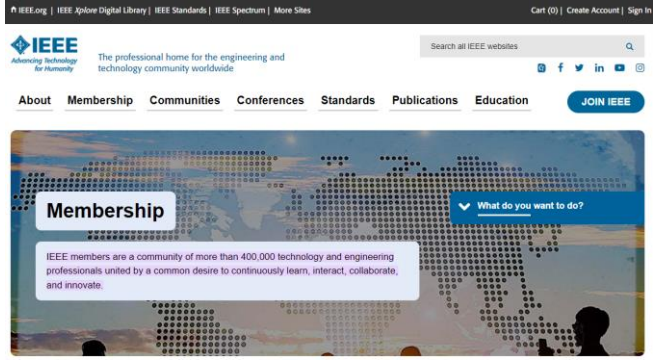
https://colab.research.google.com/ (only with Google Account)

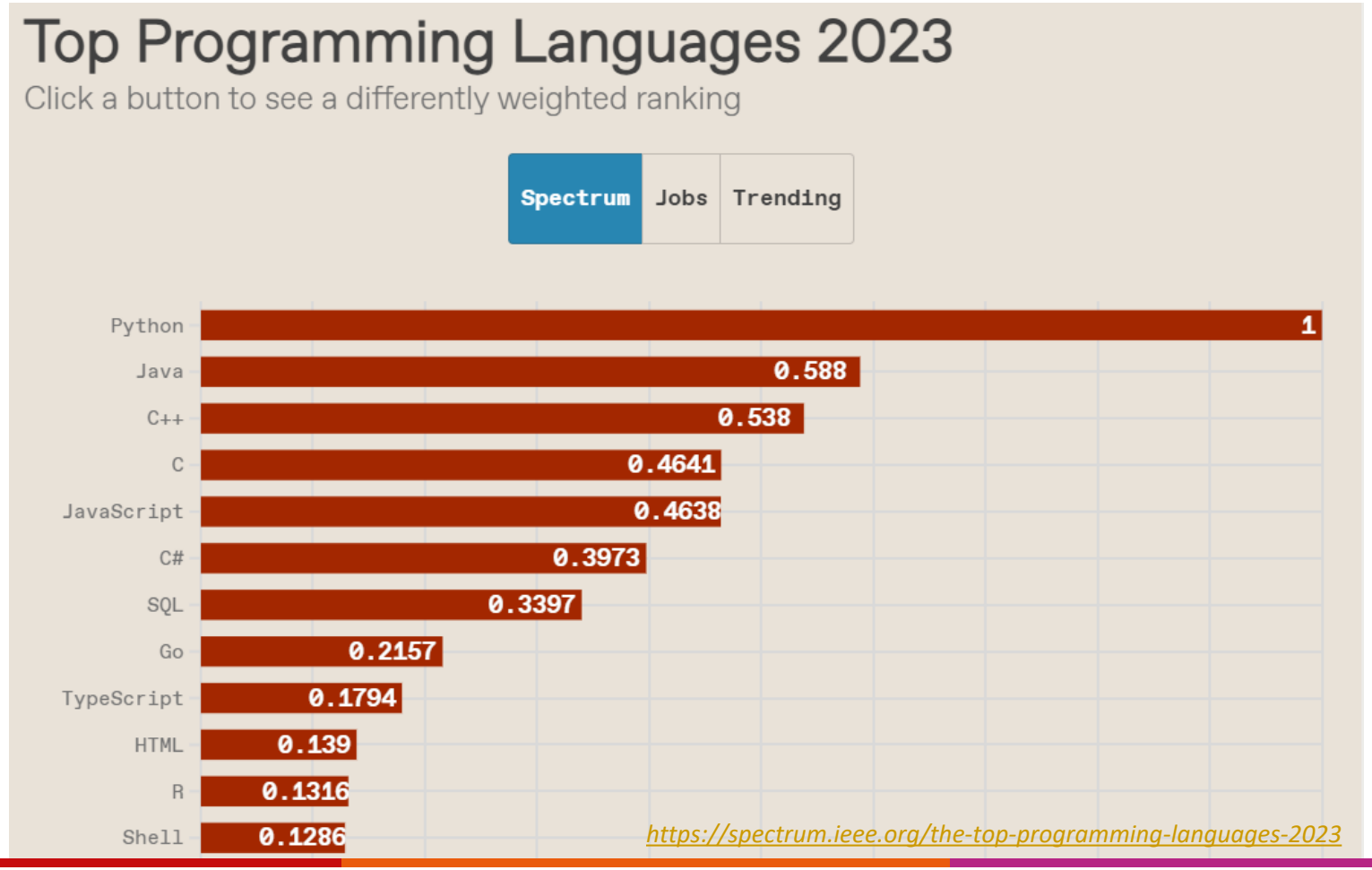


Learning Goals Python I

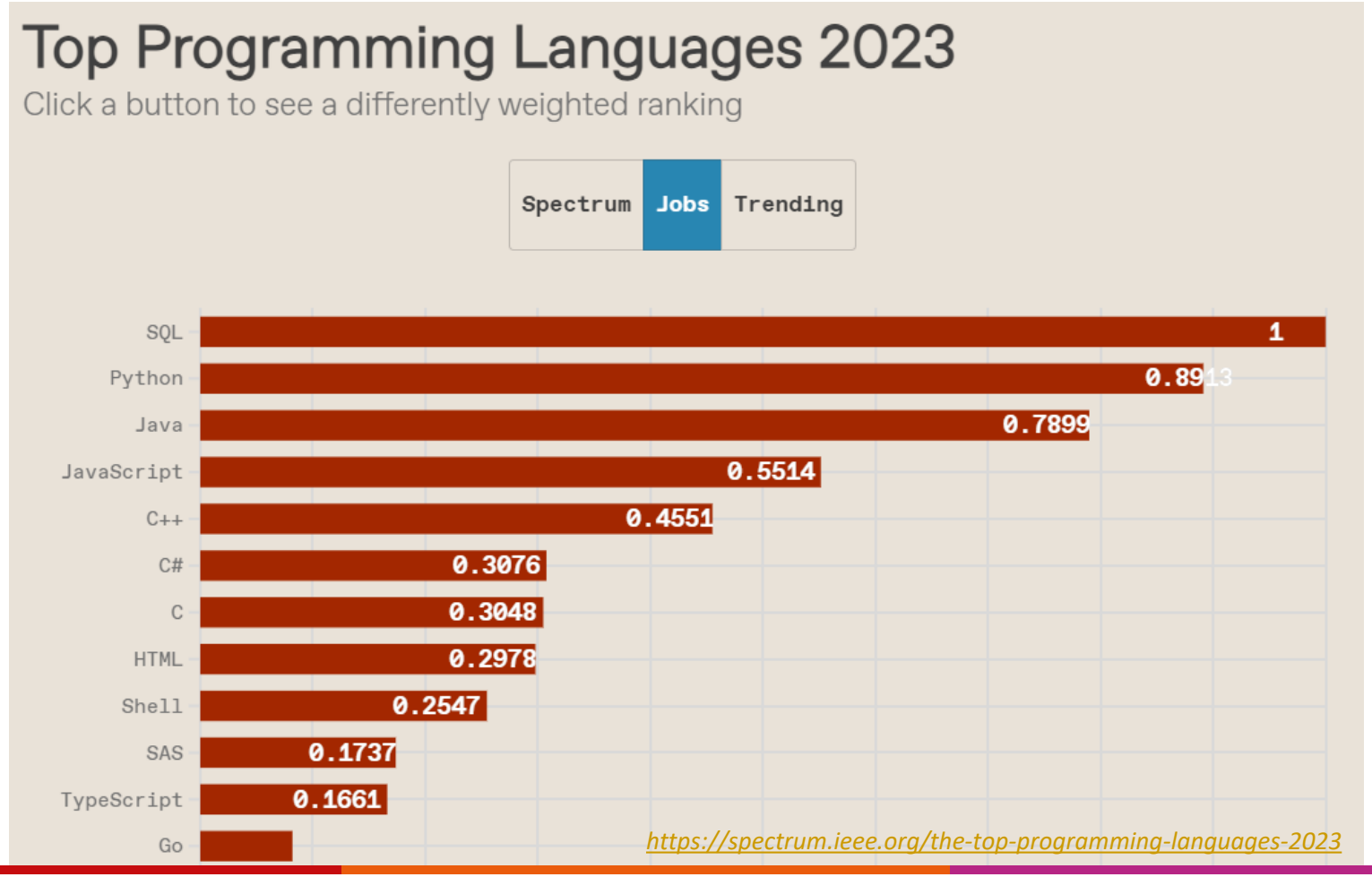
- **Describe** the reasons for the popularity of Python as a programming language and Jupyiter as a development environment.
- **Explain** the advantages and disadvantages between dynamic and static typed programming languages.
- **List** potential limitations with Python and Jupyiter.
- **Demonstrate** different documentation alternatives in Jupyiter notebook and justify when to chose which alternative.
- **Create, change,** and **apply methods on** simple number, boolean, and string data typed variables.

Top Programming Languages





Top Programming Languages



Top Programming Languages

Why Python?

- Open-source: widely used in science and more and more in industry.
- Supports structured, object-oriented and functional programming.
- Syntax with focus on readability “pythonic”.
- Dynamically typed and garbage-collected.
- Interface to other programming languages (C, Fortran,…).
- **Large**, **active**, and **growing ecosystem** of third-party packages.
  - NumPy: manipulation of numeric array-based data
  - Pandas: manipulation of tabular data
  - SciPy: scientific computing tasks
  - Matplotlib: data visualizations
  - Scikit-Learn: machine learning algorithms

What is Jupyter Notebook?

- Web-based interactive computing platform.
- Document activities (enable reproducibility).
- Collaborative functionalities.
- Big data integration (Apache ecosystem, **Python**, **R**, Matlab,...).

https://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html

Limitations 

With Jupyter and Python, many obstacles such as **infrastructure management**, **documentation**, **collaboration**, and **automated version control** are eased.

For sufficiently large data sets and complex analytics, Jupyter and Python may not be ideal.

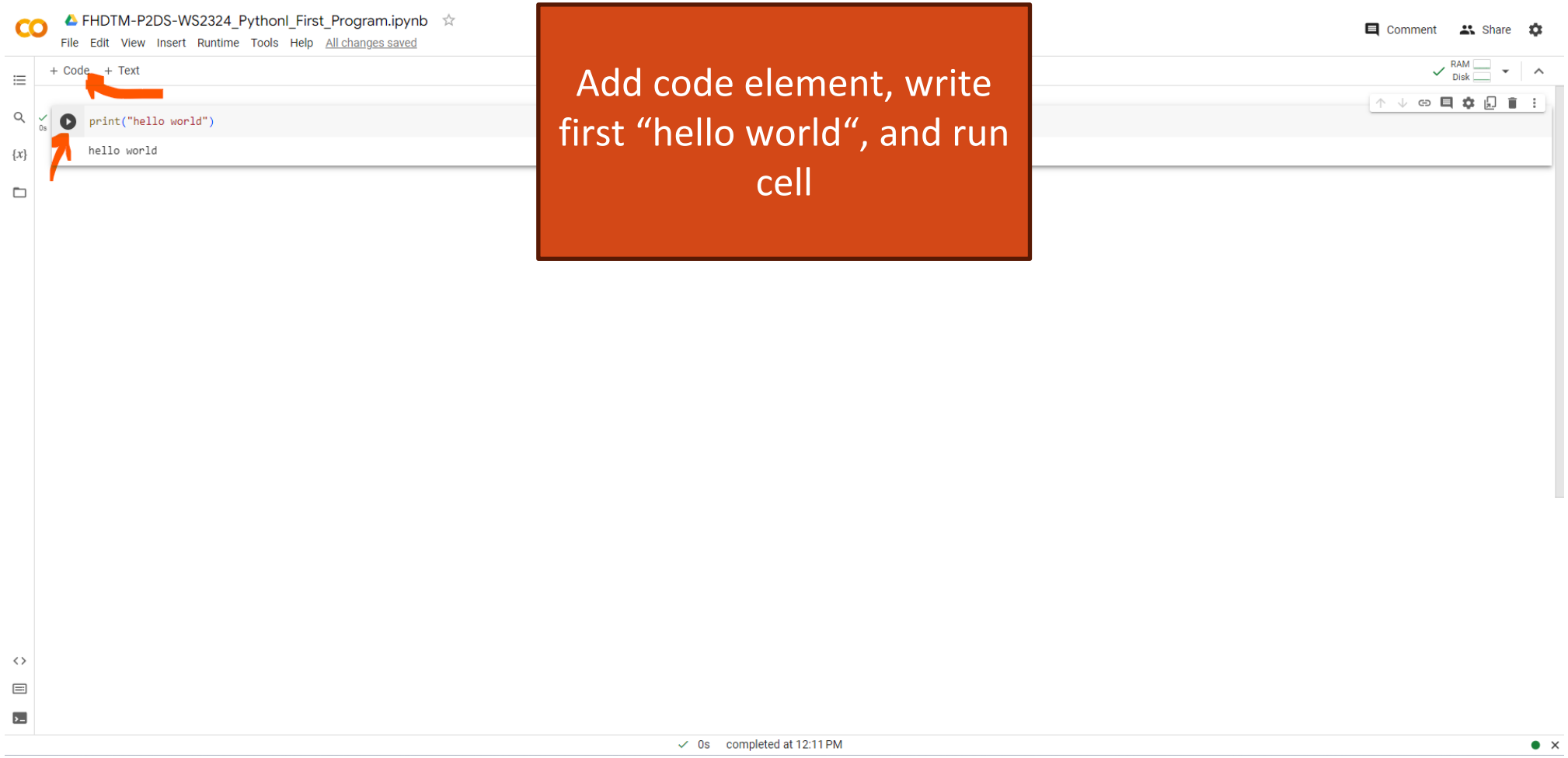
- Knowing *when to invest in switching tools* is a skill.
- Evaluate *trade-offs of flexibility, security, and speed for a given scale*.

In this class we use Jupyter and Python because

- Speed and
- Security

are typically not your priority during exploration.

Colab Jupyter Notebook Demo I



Python Interpreter

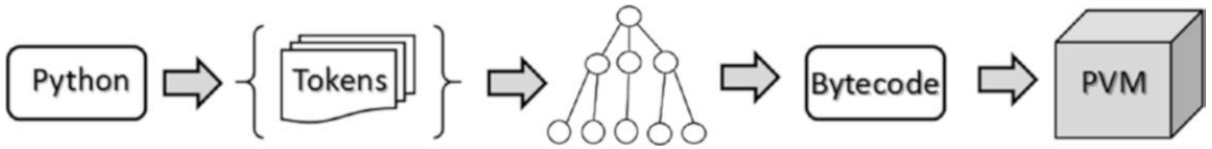
- Check version by importing the system-specific parameters and functions.

```
import sys
print(sys.version)
```

```
3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0]
```

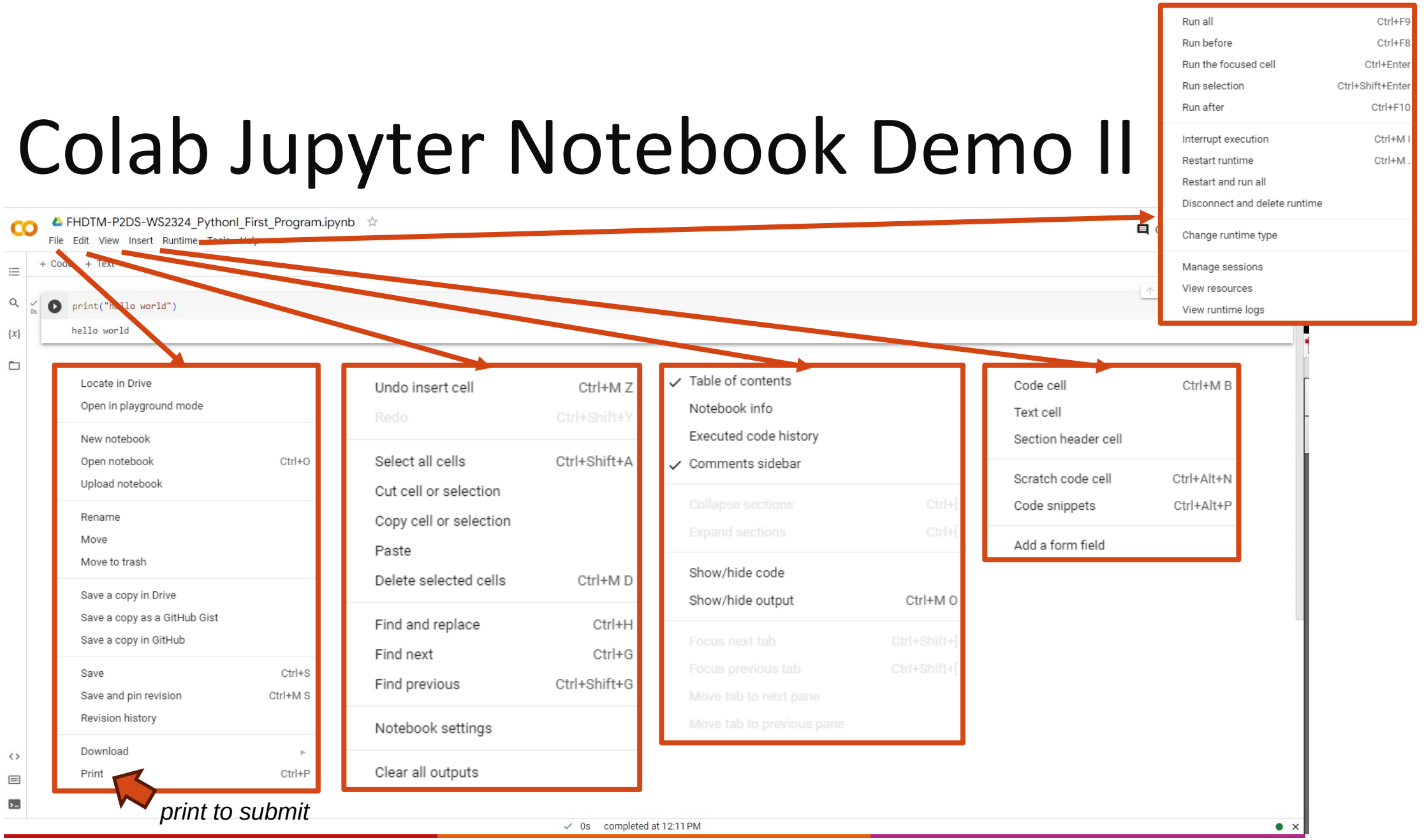
This class requires Python version 3.6 or higher

- The Python interpreter reads and interprets the commands passed to the prompt.

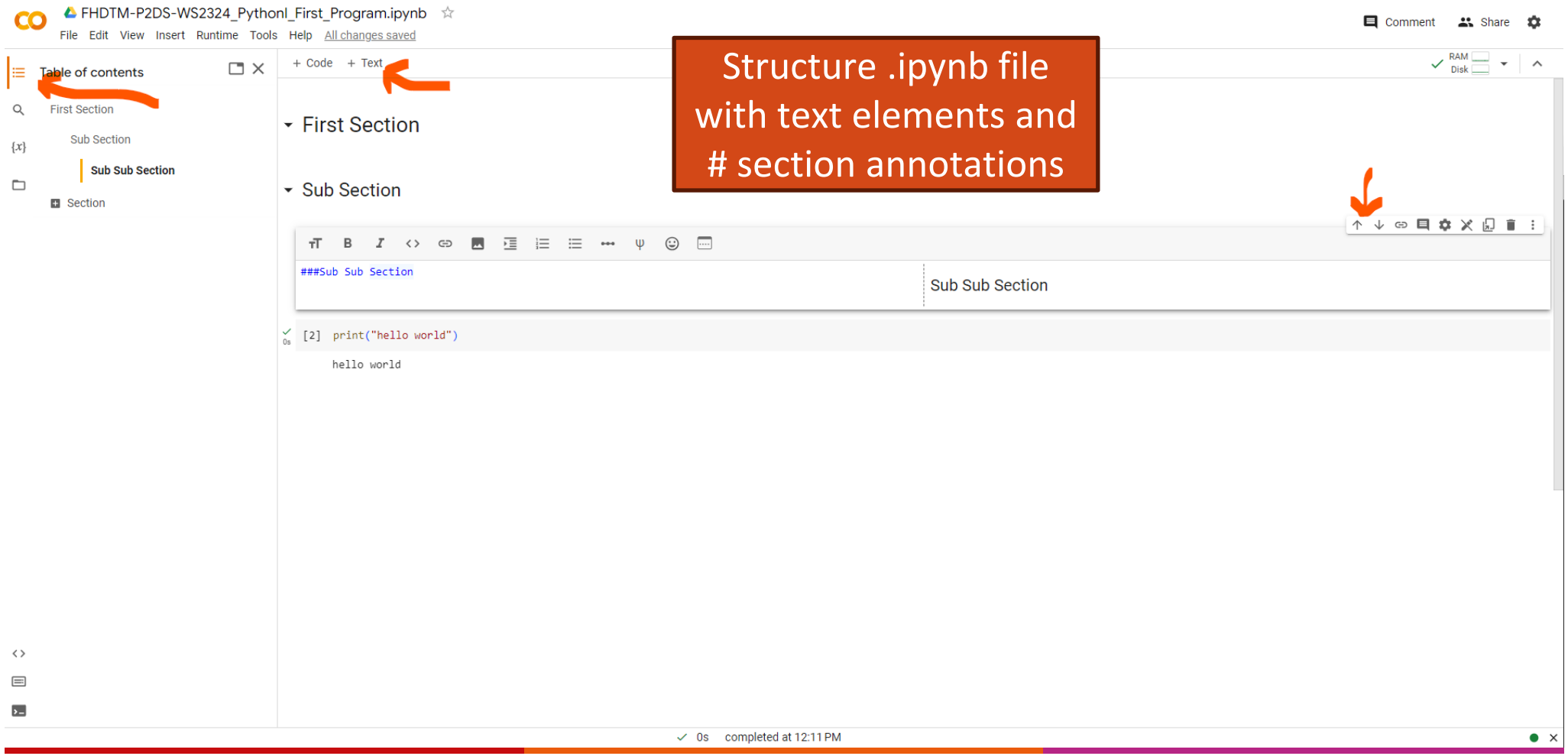


- The interpreter accepts single commands at a time or entire files of Python code.
  1. Code scan and tokenization
  2. Tree arrangement (logical structure of the program)
  3. Bytecode conversion (.pyc or .pyo file)
  4. Python Virtual Machine execution (PVM)

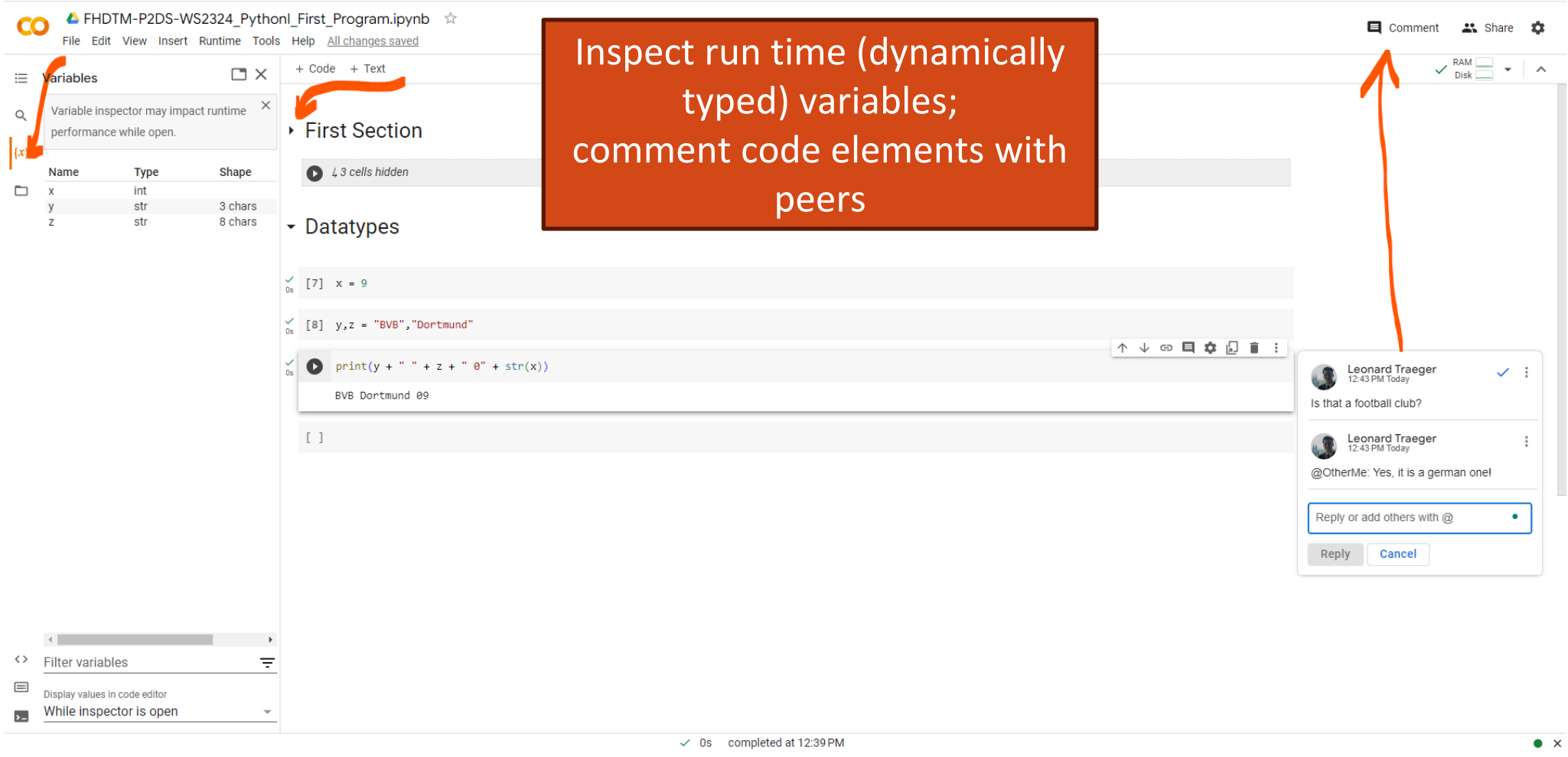
Colab Jupyter Notebook Demo II



Colab Jupyter Notebook Demo III



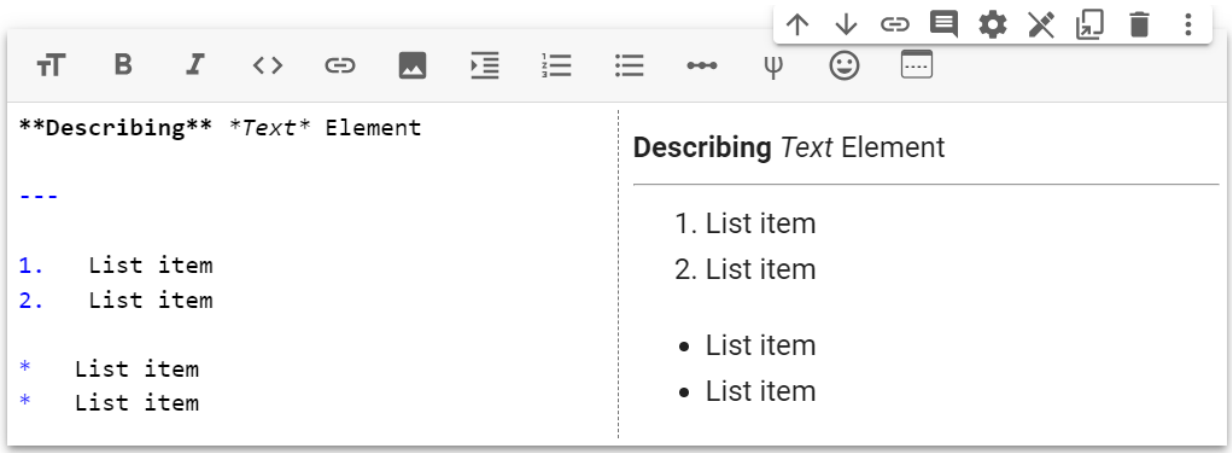
Colab Jupyter Notebook Demo IV



Inspect run time (dynamically typed) variables;
comment code elements with peers

Documentation

- Text element
  - Few formatting styles
  - Emojis
  - Lists
  - Links
  - Images



```
**Describing** *Text* Element

---

1.   List item
2.   List item

*   List item
*   List item
```

Describing *Text* Element

1. List item
2. List item

- List item
- List item

- Code: inline comment with preceding #.

```
#inline comment
```

- Code: multiple lines comment block with preceding ‘’’ and closing ‘’’.

```
'''mutiple
line
comment'''
```

Variables and Datatypes

Combination of an identifying label and a value.

```
x = 9
```

```
y,z = "BVB","Dortmund"
```

→ *Variable **value assignment.***

```
print(y + " " + z + " 0" + str(x))
```

```
BVB Dortmund 09
```

→ *Use **variable label** to get value.*

→ *Convert it for subsequent processing step e.g., concatenation: str(9) = “9”.*

Variable and Datatypes (cont.)

Assignment uses **dynamic referencing**.

- The **type/class** is determined from the **value**, not declared.
- **Type/class information** belongs to the **data**, not the name bound to that data.

```
x = 10000
```

- x is not just a “raw” integer.
- x is a pointer to a compound C structure, which contains several values.
- **Dynamic referencing** in Python is **more flexible** but also **more time** and **space** consuming than compared to raw C.

Variables and Datatypes (cont.)

- Anything can be a variable in Python: number, string, function, module, object, ...
- Names can be **any continuous** string but must **start** with a **letter** or **underscores**.
- The **more meaningful** your **names**, the **easier** it will be to **understand** the **code**.

```
snake_case_style_answer_to_everything = 42
```

```
camelCaseStyleAnswerToEverything = "42"
```

```
print( type(snake_case_style_answer_to_everything), type(camelCaseStyleAnswerToEverything))
```

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

Indentation

- Python uses indentation to structure code unlike many other languages, e.g., R, C++, which use braces { }.
- In Python, **indentation** is **functional**, not just good style.
- Functional whitespace makes Python **more readable** to **humans** but also potentially harder to debug (in the beginning).
- A **colon :** is used to denote the **start** of an **indented block**.

```
k = 0
for i in range(4):
  k += i
  print(k)
```

```
k = 0
for i in range(4):
  k += i
print(k)
```

Modules / Libraries

- Open-source code-reuse: one of Python‘s essential reason for it‘s power and popularity.
- Modules must be imported before they can be used.
- **Avoid** at all costs `from module import *`

```
#import libraries preferably in a seperate cell
#of runing code to avoid conflicts and save time
from random import randint
from random import randint as ri
```

- Give imported object **an alias** to avoid writing the whole name every time.

```
print(randint(0,101))
print(ri(0,101))
#Prints (any random number between zero and including 100)"
```

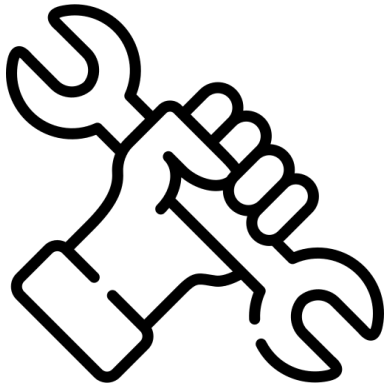
Numbers

- Integers and floats work as you would expect from other languages.

```
x = 3
print(type(x)) # Prints "<class 'int'>"
print(x)       # Prints "3"
print(x + 1)   # Addition; prints "4"
print(x - 1)   # Subtraction; prints "2"
print(x * 2)   # Multiplication; prints "6"
print(x ** 2)  # Exponentiation; prints "9"
x += 1
print(x)  # Prints "4"
x *= 2
print(x)  # Prints "8"
y = 2.5
print(type(y)) # Prints "<class 'float'>"
print(y, y + 1, y * 2, y ** 2) # Prints "2.5 3.5 5.0 6.25"
```

- Note: Python does not have unary increment (x++) or decrement (x--) operators!!!

Training #1



1) Count the number of people in the lecture room and the number of local computers.
2) Compute the average number of computers per person and assign this value to a variable.
3) Print the value and the data type of the computed variable.

Booleans

- Python implements all Boolean logic, but uses English instead symbols (&&, ||, etc.):

```
t = True
f = False
print(type(t)) # Prints "<class 'bool'>"
print(t and f) # Logical AND; prints "False"
print(t or f)  # Logical OR; prints "True"
print(not t)   # Logical NOT; prints "False"
print(t != f)  # Logical XOR; prints "True"
```

Only exception:
Pandas Masking!
Week 7+

String

```
hello = 'hello'    # String literals can use single quotes
world = "world"    # or double quotes; it does not matter.
print(hello)       # Prints "hello"
print(len(hello))  # String length; prints "5"
hw = hello + ' ' + world  # String concatenation
print(hw)  # prints "hello world"
hw12 = '%s %s %d' % (hello, world, 12)  # sprintf style string formatting
print(hw12)  # prints "hello world 12"
```

String (cont.)

- Some very useful string methods:

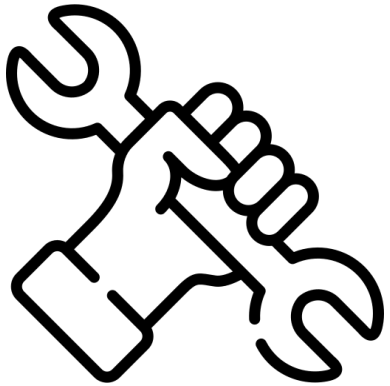
```
string = "BVB Dortmund 09 "
print(string[14])           # prints 14th element of the string "9"
print(string.capitalize())  # Capitalize a string; prints "Bvb dortmund 09 "
print(string.strip())       # Removes any whitespace from the beginning or the end;
                            # prints "BVB Dortmund 09"
print(string.replace('B', 'Borussia'))  # Replace all instances of one substring with another;
                            # prints "BorussiaVBorussia Dortmund 09 "
print(string.lower())       # Convert a string to lower case; prints "bvb dortmund 09 "
print(string.upper())       # Convert a string to uppercase; prints "BVB DORTMUND 09 "

string_split = string.split(" ") # Splits string into substrings based on separator
print(len(string_split))    # prints lenght of array list "4"
print(string_split[0])      # prints first value of array list "BVB"
```

More to arrays and lists next class!

https://docs.python.org/3.5/library/stdtypes.html#string-methods

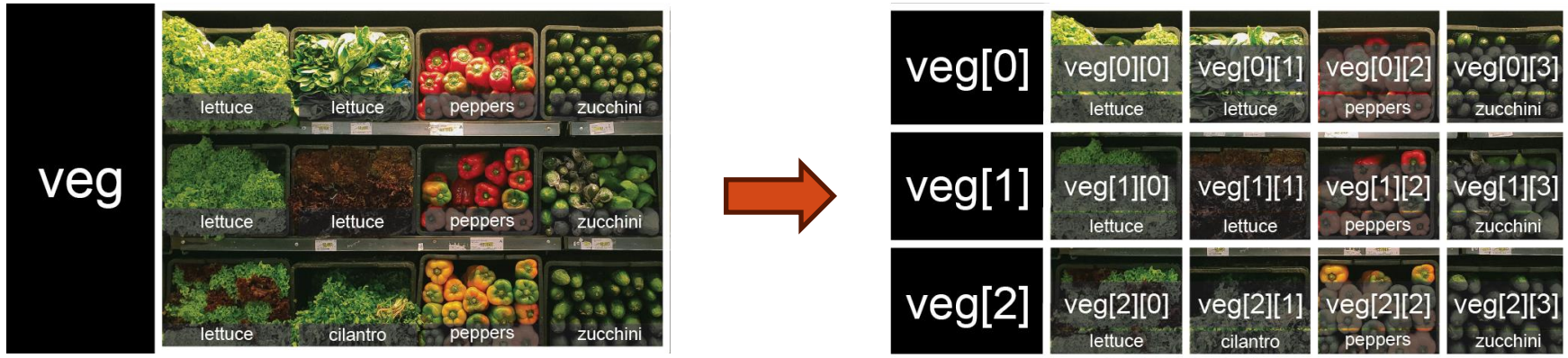
Training #2



1. Assign "ojyVTTyamyknYMakLmktDYNtyklnyhmyrn" to a variable s1.
2. Assign "kNZwykwoCHynsynDYtkhrMKrjYpr" to another variable s2.
3. Concat both strings in s2+s1 order.
4. Cast the lower-case method on the concatenated string.
5. Replace "k" with "i".
6. Replace "v" with "k".
7. Replace "y" with "e".
8. Print the output.

Outlook

- In week 3 we will start with Data Science Life Cycles, Data Literacy, and Ethics as you will need these for structuring your DS project.
- Afterwards, we will dive deep into containers and functions in Python.



- Refine expectations for project milestone #1.

See you again in **two weeks in person**!

Questions?