



# Python I

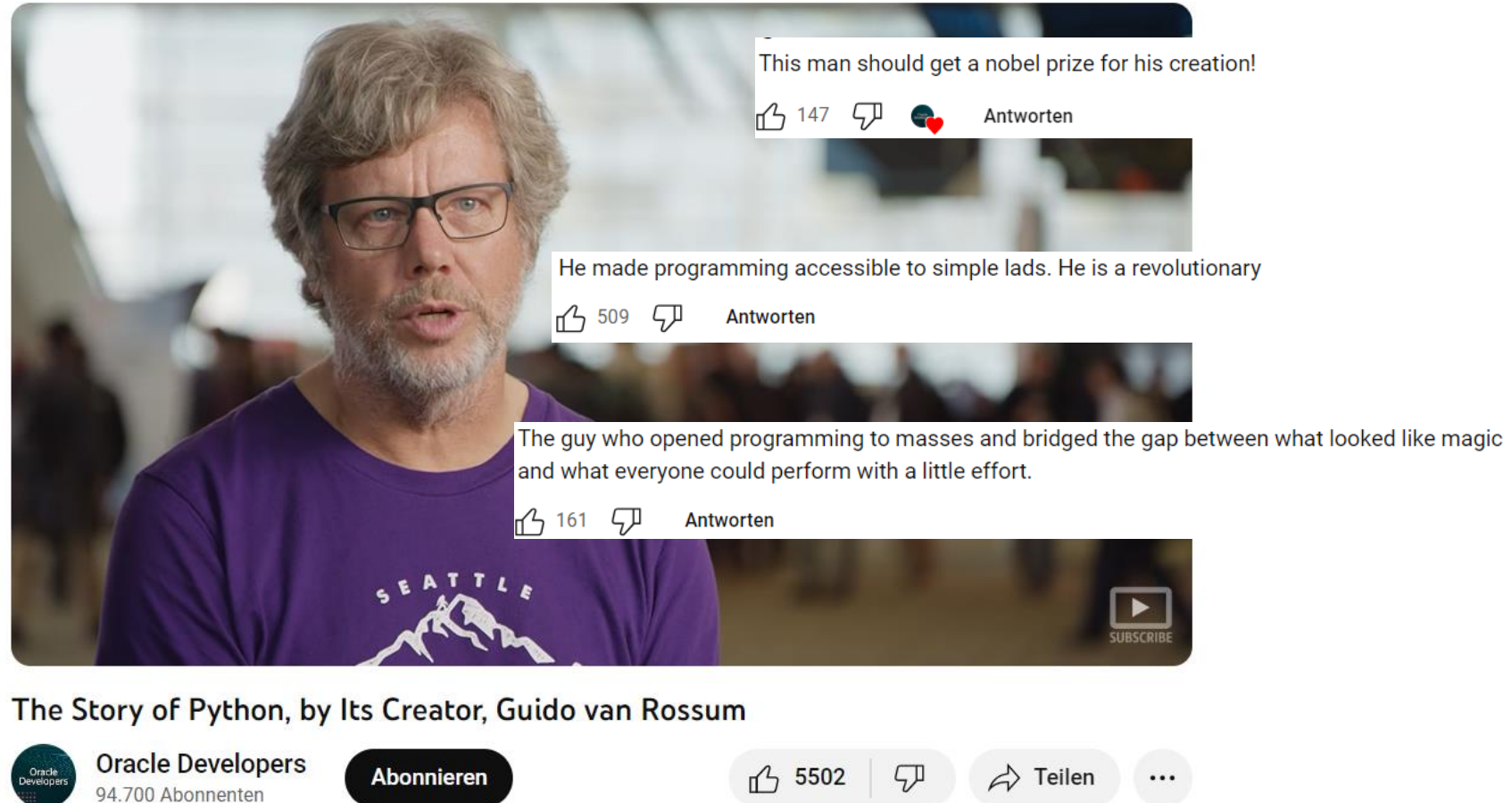
## Programmierkurs 2 Data Science

### WS24/25

Leonard Traeger  
M. Sc. Information Systems  
[leonard.traeger@fh-dortmund.de](mailto:leonard.traeger@fh-dortmund.de)



# Who is Guido van Rossum?



# Voraussetzungen

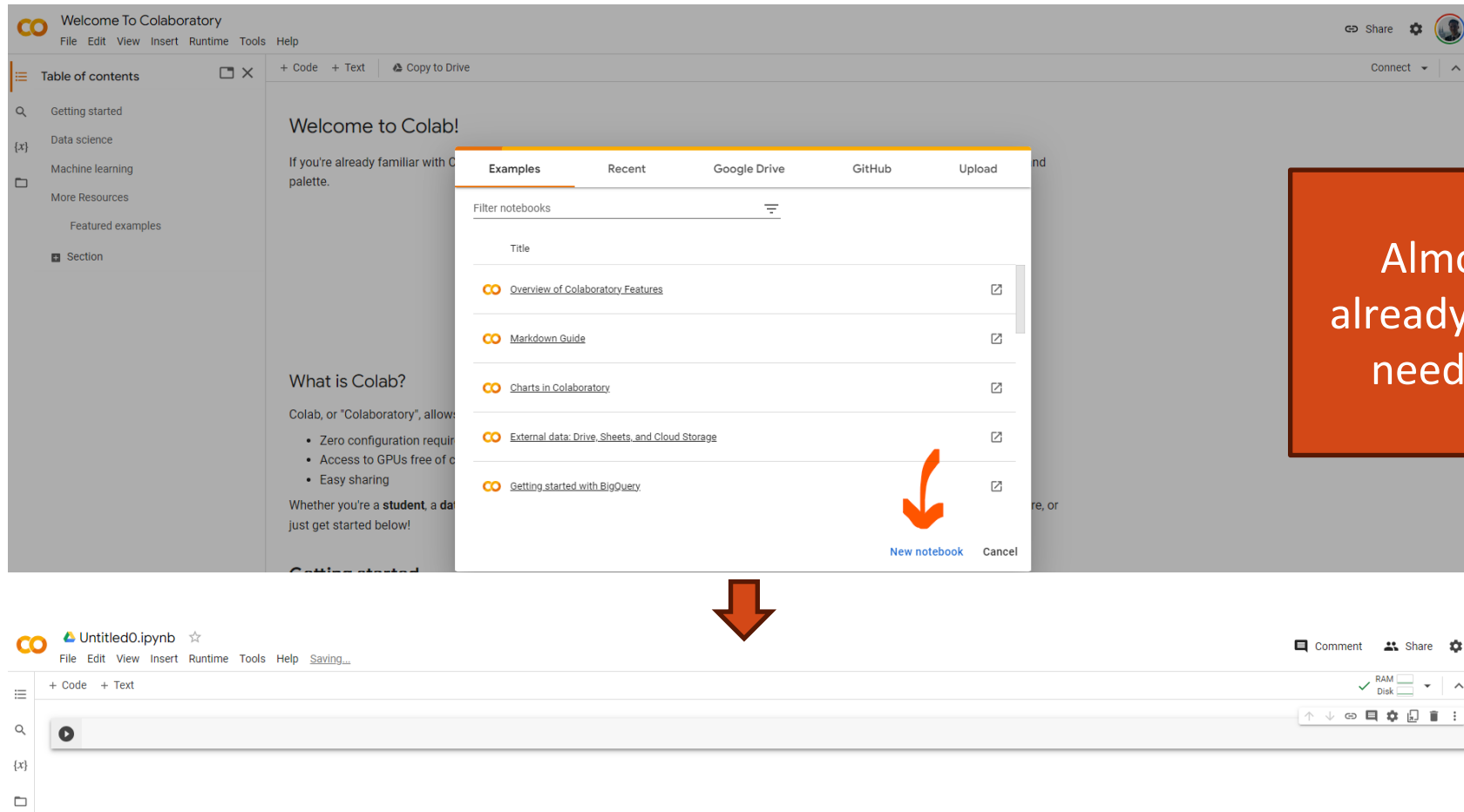
## Hardware

- 4GB RAM, 2 CPU-Kerne und 20GB Speicher oder mehr.
- Windows 10, macOS 10.14, oder Ubuntu 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

- Alternative 1: Jupyter Python Notebook in [Google Collab](#) als Web IDE (Vorteil: in Teams arbeiten, kommentieren, dokumentieren und programmieren).
- Alternative 2: Jupyter Python Notebook in Lokaler Laufumgebung mit Visual Studio Code (Vorteil: Offline arbeiten, in Teams mittels GIT o.ä. Share-Drive arbeiten):  
[Python 3.10 oder neuer unter https://www.python.org/downloads/](https://www.python.org/downloads/),  
[Download Visual Studio Code unter https://code.visualstudio.com/](https://code.visualstudio.com/),  
[Python in VS Code Anleitung unter https://code.visualstudio.com/docs/python/python-tutorial#\\_install-a-python-interpreter](https://code.visualstudio.com/docs/python/python-tutorial#_install-a-python-interpreter)  
und Python Erweiterungen in Visual Studio Code Extension Marketplace installieren.
- Alternative 3: Jupyter Python Notebook über Anaconda Package Platform (Vorteil: Offline arbeiten, RStudio und weitere Data Science Plattformen inkludiert): [Python 3.10 oder neuer unter https://www.python.org/downloads/](https://www.python.org/downloads/) und [Anaconda](#).

# 1. Colab Research <https://colab.research.google.com/> (only with Google Account)

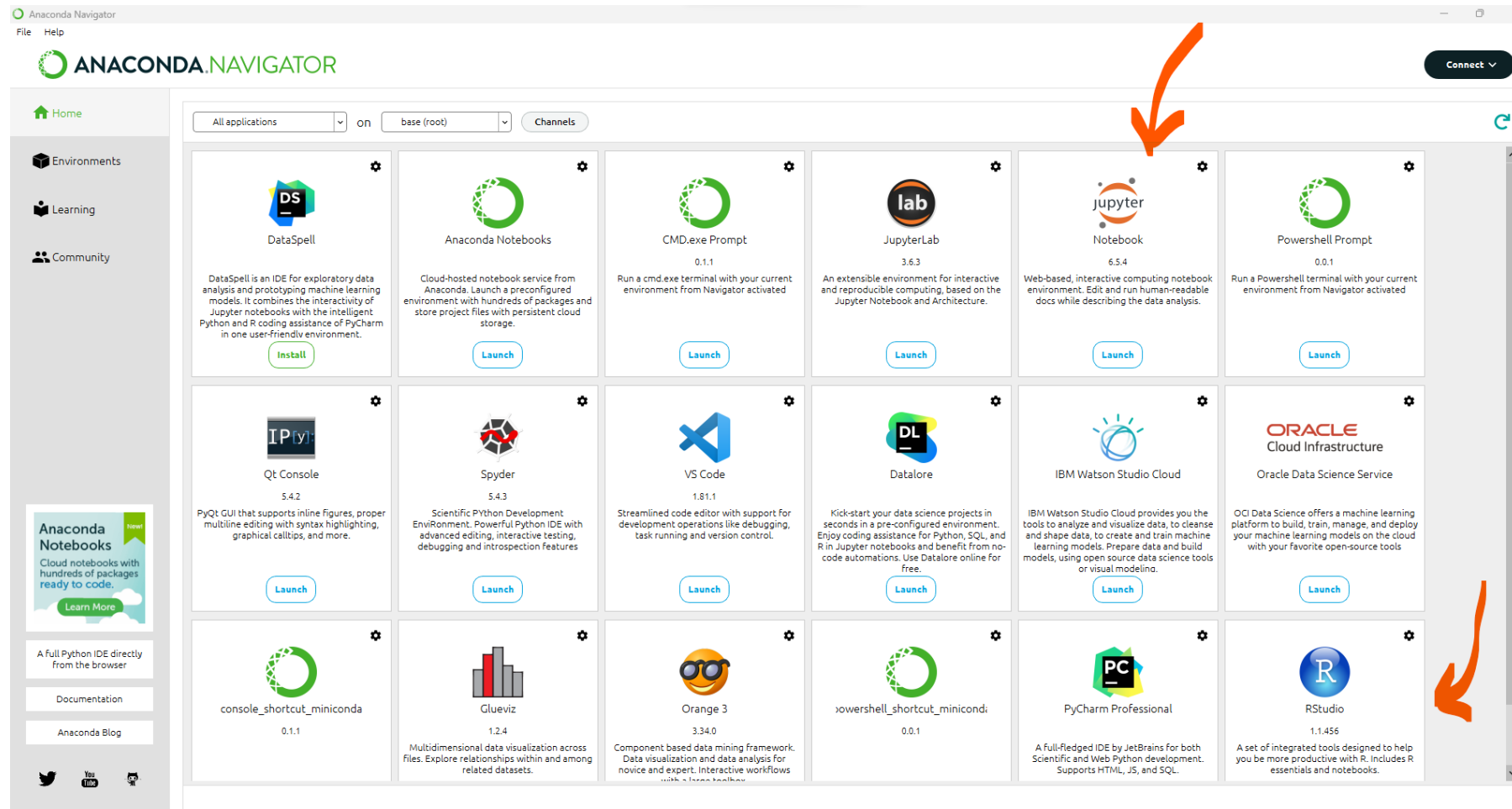


Almost all packages  
already installed and just  
need to be imported



<https://www.anaconda.com/>

## 2. Anaconda I



R wird in  
Woche 10-12  
behandelt

## 2. Anaconda II

The screenshot displays the JupyterLab web interface. At the top, the 'jupyter' logo is on the left, and 'Quit' and 'Logout' buttons are on the right. Below the logo are tabs for 'Files', 'Running', and 'Clusters'. A message states 'Select items to perform actions on them.' The file browser shows a directory structure: '0' (selected), 'Meine Ablage', 'WS23\_24 PhD', 'FHDTM-P2DS-WS2324', and 'Data Science Projekt'. Below this are folders for '..', 'Dateien', and 'Skripte'. An orange arrow points from the 'New' button to a dropdown menu. The menu has a 'Name' field and lists options: 'Python 3 (ipykernel)' (highlighted), 'Text File', 'Folder', and 'Terminal'. Another orange arrow points from the 'Python 3 (ipykernel)' option to the main interface. A large brown arrow points down to the next screenshot. The second screenshot shows the JupyterLab notebook interface. It has a header with 'jupyter', 'Untitled', and 'Last Checkpoint: a few seconds ago (unsaved changes)'. On the right, there's a Python logo and a 'Logout' button. Below the header is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. A 'Trusted' status indicator and 'Python 3 (ipykernel)' are also shown. The main area contains a toolbar with icons for saving, creating, deleting, and running cells, along with a 'Code' dropdown and a console icon. The bottom part of the notebook shows a code input area with 'In [ ]:'.

# 3. Local Python Runtime + VS Code

Use Python 3 in Visual Studio Code to create, run, and debug:

[https://code.visualstudio.com/docs/python/python-tutorial#\\_install-a-python-interpreter](https://code.visualstudio.com/docs/python/python-tutorial#_install-a-python-interpreter)

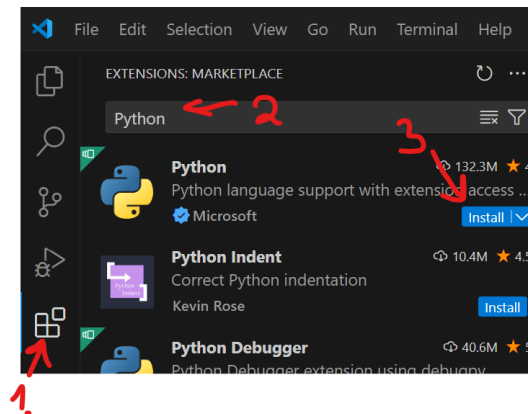
1. Download and install stable Python 3.xx version for your OS:

<https://www.python.org/downloads/>

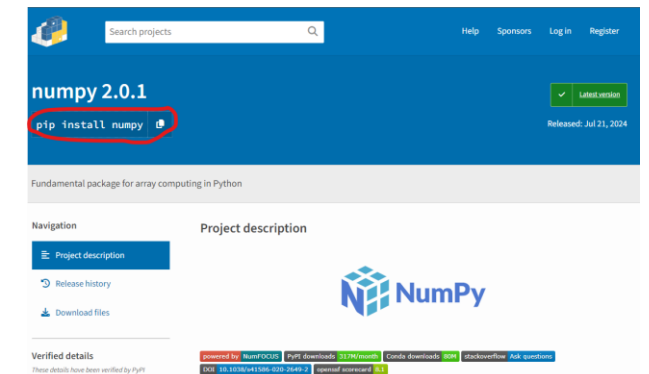
2. Download Visual Studio Code: <https://code.visualstudio.com/>

3. Search and install Python extension in Visual Studio Code Extension Marketplace

1. Create new markdown file .ipynb
2. Install packages with `pip install packname`
3. Import packages with `import packname`
4. ...



*pypi.org*



# Learning Goals Python I

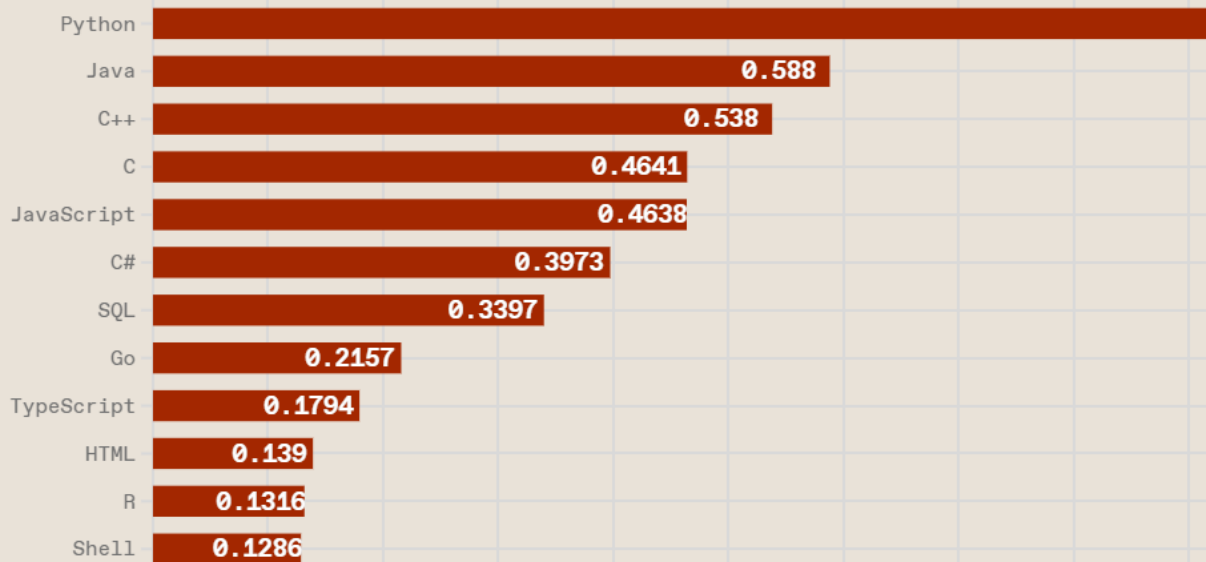
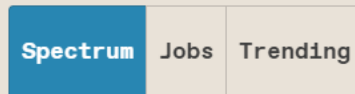
- **Describe** the reasons for the popularity of Python as a programming language and Jupyter as a development environment.
- **Explain** the advantages and disadvantages between dynamic and static typed programming languages.
- **List** potential limitations with Python and Jupyter.
- **Demonstrate**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



# Top Programming Languages

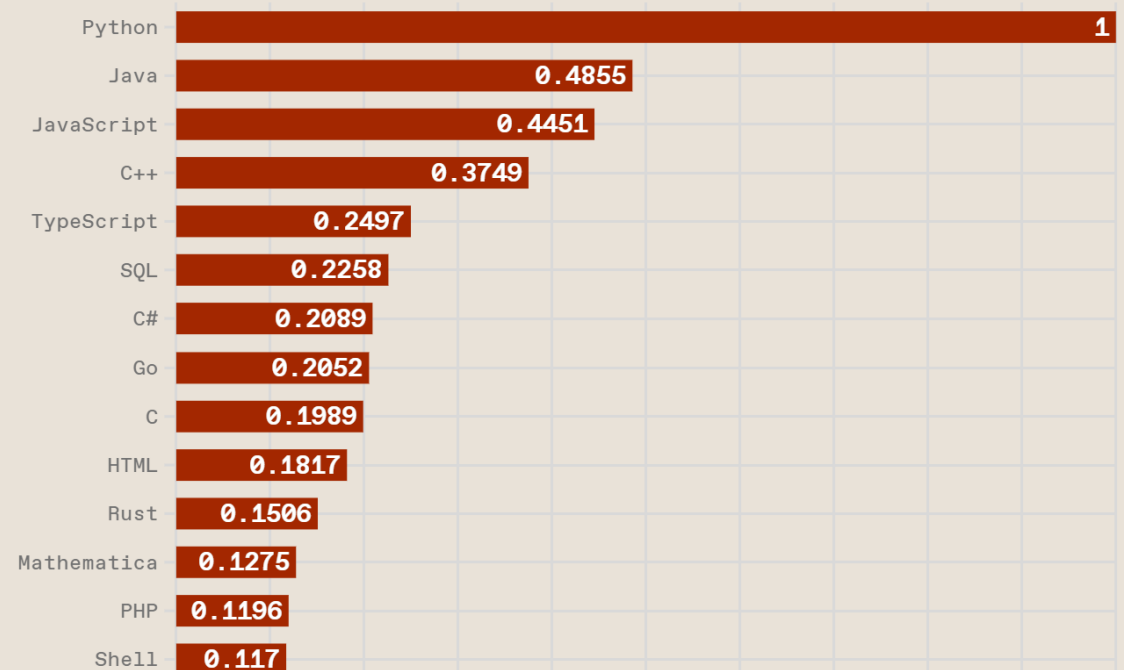
## Top Programming Languages 2023

Click a button to see a differently weighted ranking



## Top Programming Languages 2024

Click a button to see a differently weighted ranking



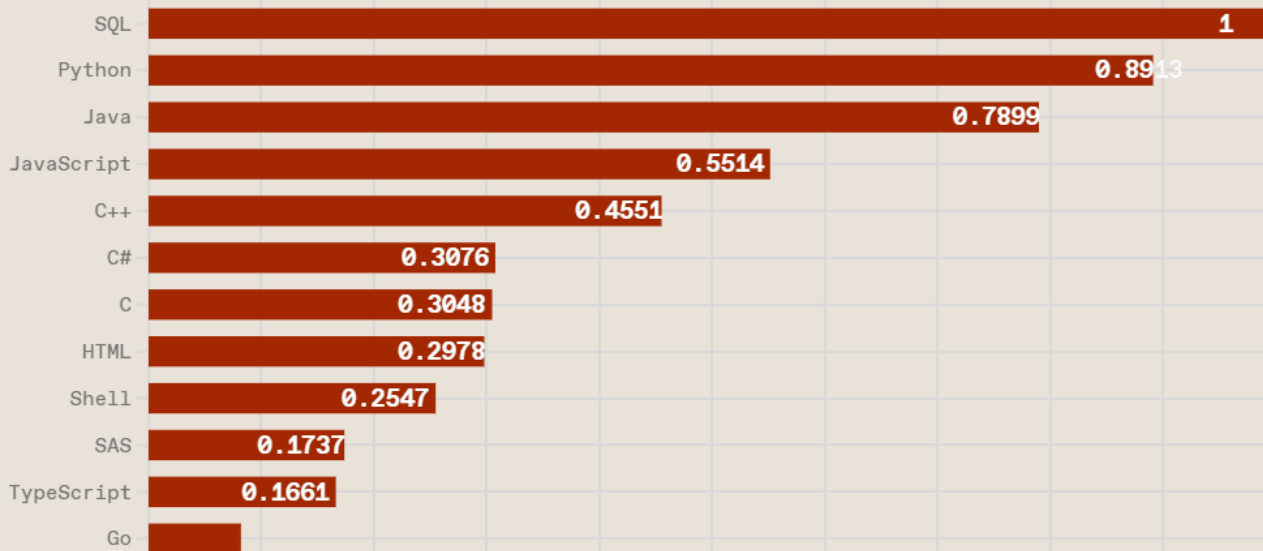
<https://spectrum.ieee.org/top-programming-languages-2024>

# Top Programming Languages

## Top Programming Languages 2023

Click a button to see a differently weighted ranking

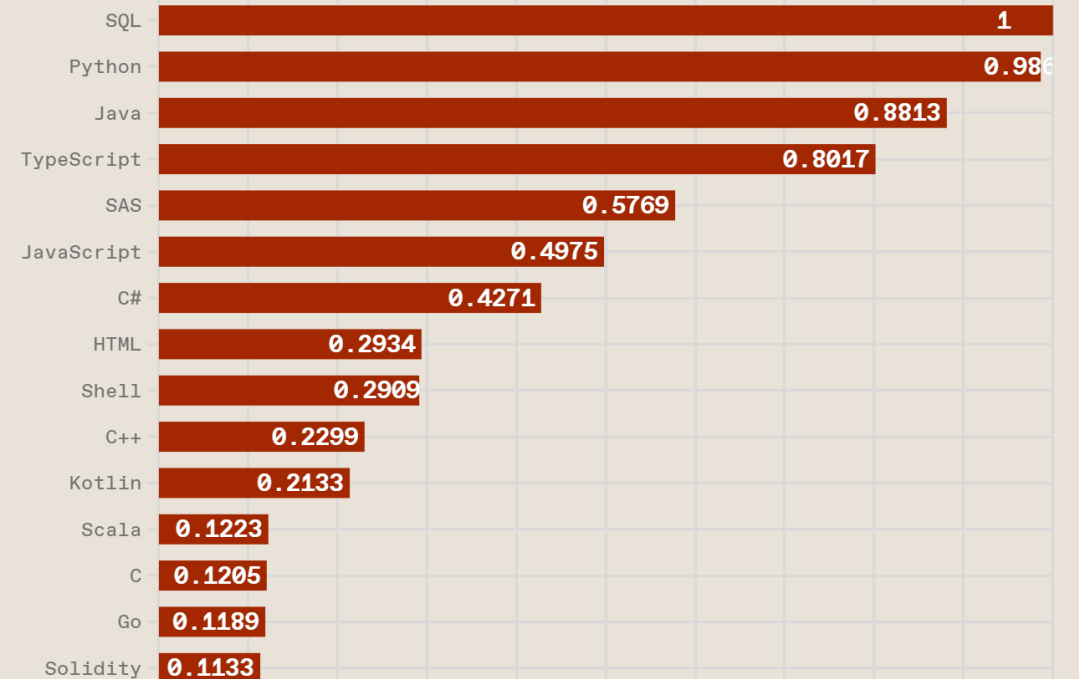
Spectrum **Jobs** Trending



## Top Programming Languages 2024

Click a button to see a differently weighted ranking

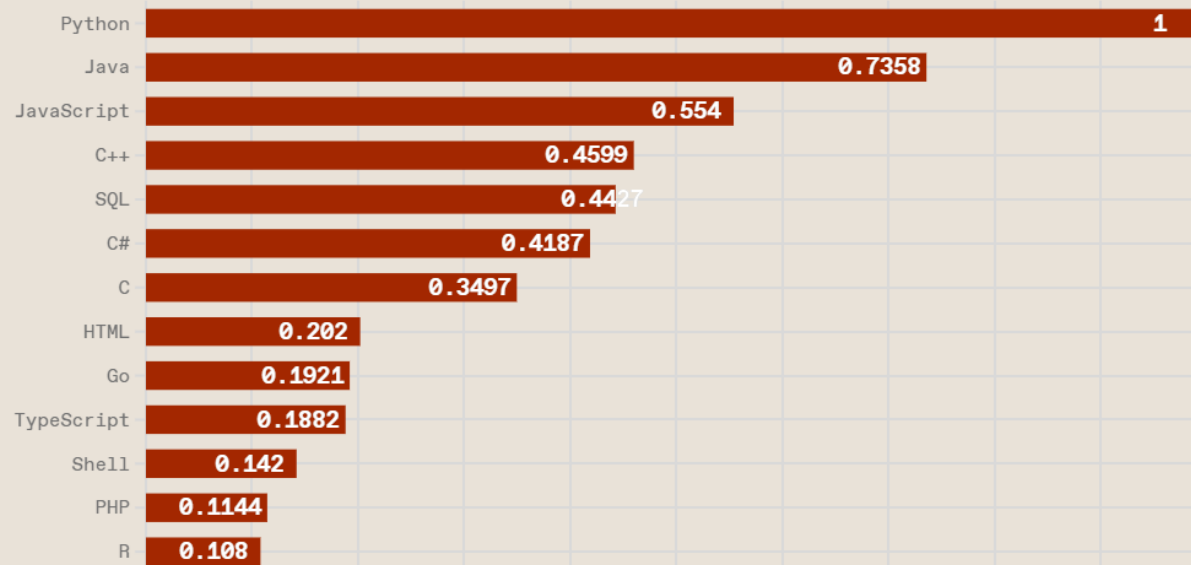
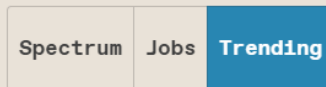
Spectrum Trending **Jobs**



# Top Programming Languages

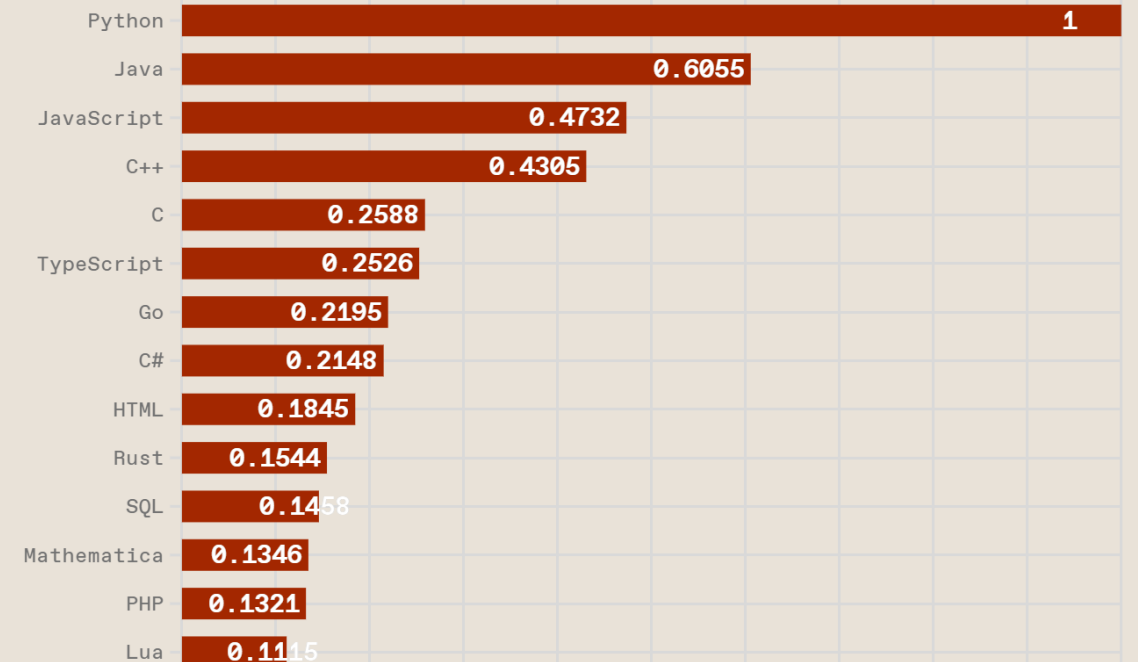
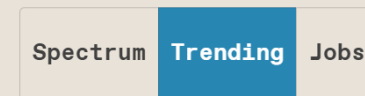
## Top Programming Languages 2023

Click a button to see a differently weighted ranking



## Top Programming Languages 2024

Click a button to see a differently weighted ranking



# 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
  - Pytorch/TensorFlow: deep neural networks from Meta / Google

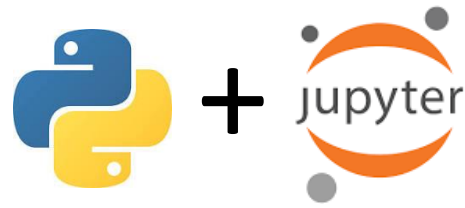
# What is Jupyter Notebook?



- 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](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*.

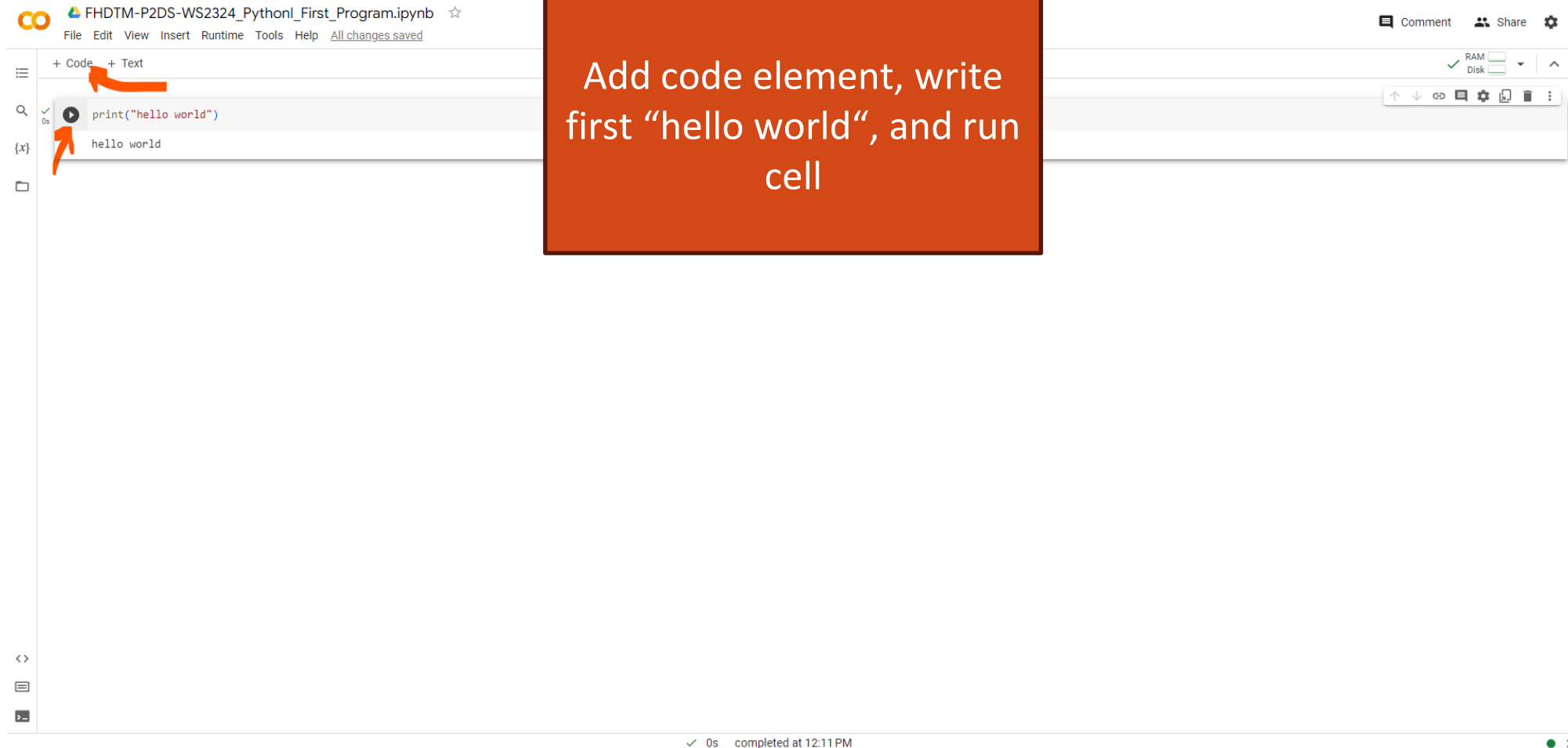
In this class we use Jupyter and Python because

- Speed and
- Security

are typically not your priority during exploration.



# Colab Jupyter Notebook Demo I



# Python Interpreter

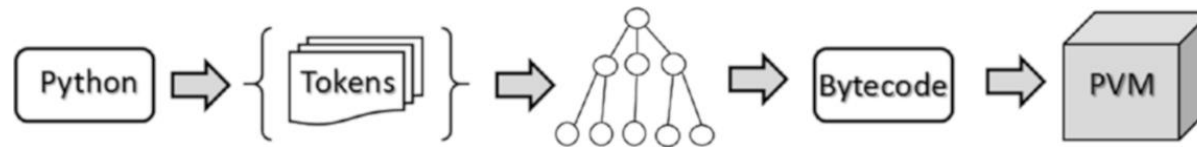
This class requires  
Python version 3.6 or  
higher

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

```
1 import sys
2 print(sys.version)
✓ 0.0s
```

3.12.5 (tags/v3.12.5:ff3bc82, Aug 6 2024, 20:45:27) [MSC v.1940 64 bit (AMD64)]

- 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.
  - Code scan and tokenization
  - Tree arrangement (logical structure of the program)
  - Bytecode conversion (.pyc or .pyo file)
  - Python Virtual Machine execution (PVM)

# Colab Jupyter Notebook Demo II

The screenshot shows the Google Colab Jupyter Notebook interface. The top menu bar includes File, Edit, View, Insert, Runtime, Tools, and Help. The main area contains a code cell with the text `print("hello world")` and its output, "hello world".

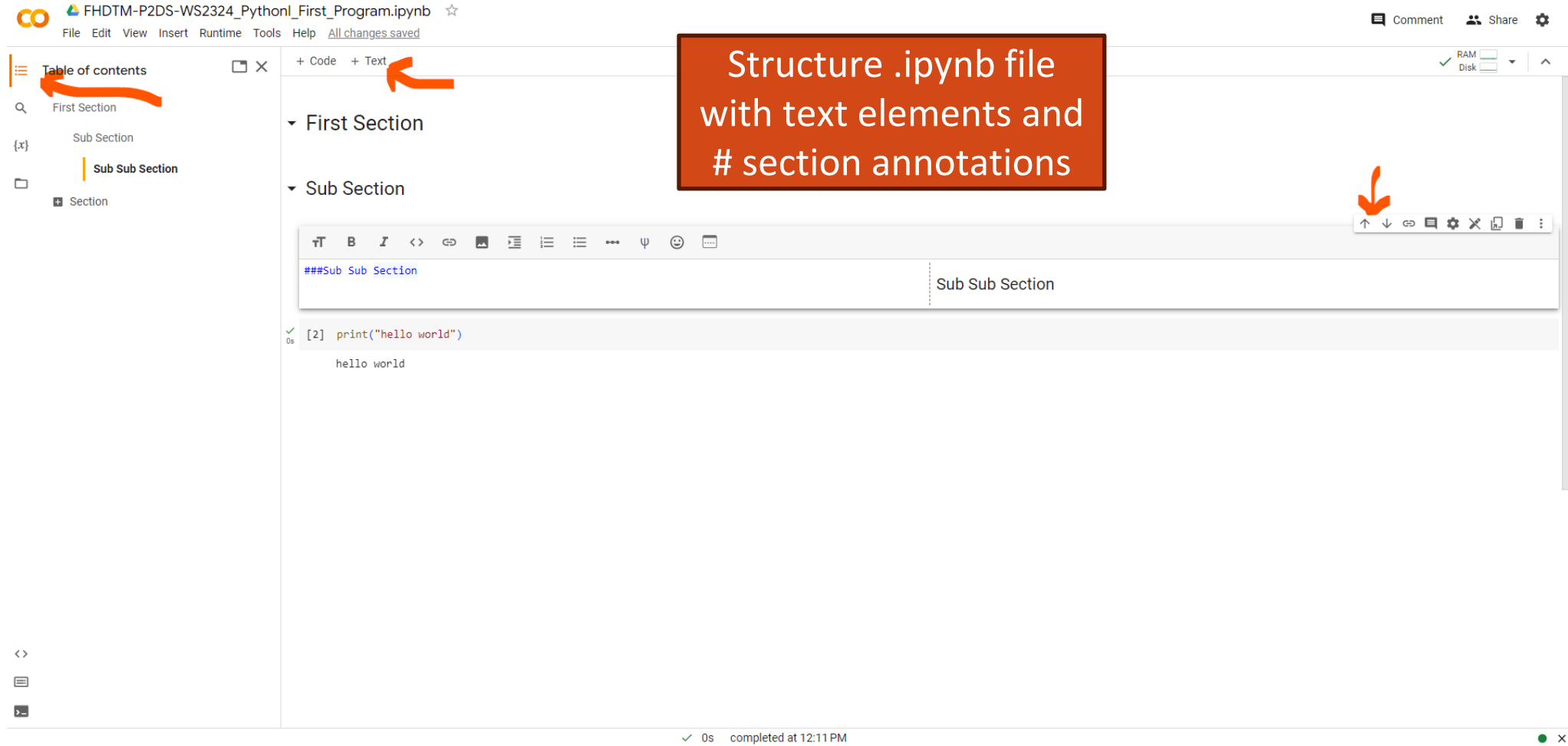
Four red boxes highlight specific menus and their contents:

- File menu:**
  - Locate in Drive
  - Open in playground mode
  - New notebook
  - Open notebook Ctrl+O
  - Upload notebook
  - Rename
  - Move
  - Move to trash
  - Save a copy in Drive
  - Save a copy as a GitHub Gist
  - Save a copy in GitHub
  - Save Ctrl+S
  - Save and pin revision Ctrl+M S
  - Revision history
  - Download
  - Print Ctrl+P
- Insert menu:**
  - Undo insert cell Ctrl+M Z
  - Redo Ctrl+Shift+Y
  - Select all cells Ctrl+Shift+A
  - Cut cell or selection
  - Copy cell or selection
  - Paste
  - Delete selected cells Ctrl+M D
  - Find and replace Ctrl+H
  - Find next Ctrl+G
  - Find previous Ctrl+Shift+G
  - Notebook settings
  - Clear all outputs
- View menu:**
  - Table of contents
  - Notebook info
  - Executed code history
  - Comments sidebar
  - Collapse sections Ctrl+]
  - Expand sections Ctrl+[
  - Show/hide code
  - Show/hide output Ctrl+M O
  - Focus next tab Ctrl+Shift+]
  - Focus previous tab Ctrl+Shift+[
  - Move tab to next pane
  - Move tab to previous pane
- Runtime menu:**
  - Code cell Ctrl+M B
  - Text cell
  - Section header cell
  - Scratch code cell Ctrl+Alt+N
  - Code snippets Ctrl+Alt+P
  - Add a form field

The Runtime menu is also shown in a separate box on the right side of the interface:

- Run all Ctrl+F9
- Run before Ctrl+F8
- Run the focused cell Ctrl+Enter
- Run selection Ctrl+Shift+Enter
- Run after Ctrl+F10
- Interrupt execution Ctrl+M I
- Restart runtime Ctrl+M .
- Restart and run all
- Disconnect and delete runtime
- Change runtime type
- Manage sessions
- View resources
- View runtime logs

# Colab Jupyter Notebook Demo III



The screenshot shows a Google Colab Jupyter Notebook interface. At the top, the file name is "FHDTM-P2DS-WS2324\_PythonI\_First\_Program.ipynb". The interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help), a left sidebar with a "Table of contents" panel, and a main editing area. The "Table of contents" panel on the left lists "First Section", "Sub Section", and "Sub Sub Section", with "Sub Sub Section" highlighted. In the main editing area, the "Text" tab is selected, and the content shows a section header "Sub Sub Section" followed by a code cell containing `print("hello world")`. An orange box with the text "Structure .ipynb file with text elements and # section annotations" is overlaid on the interface, with orange arrows pointing to the "Table of contents" panel and the "Text" tab. Another orange arrow points to the "Sub Sub Section" header in the code cell. The bottom status bar indicates the notebook is "completed at 12:11 PM".

# Colab Jupyter Notebook Demo IV

The screenshot shows a Google Colab Jupyter Notebook titled "FHDTM-P2DS-WS2324\_PythonI\_First\_Program.ipynb". The interface includes a top menu bar (File, Edit, View, Insert, Runtime, Tools, Help), a left sidebar with a "Variables" panel, and a main code editor area. The "Variables" panel displays a table of variables:

Name	Type	Shape
x	int	
y	str	3 chars
z	str	8 chars

The code editor shows three cells: a code cell with `x = 9`, a code cell with `y, z = "BVB", "Dortmund"`, and a code cell with `print(y + " " + z + " " + str(x))` which has executed and shown the output "BVB Dortmund 09".

Annotations include:

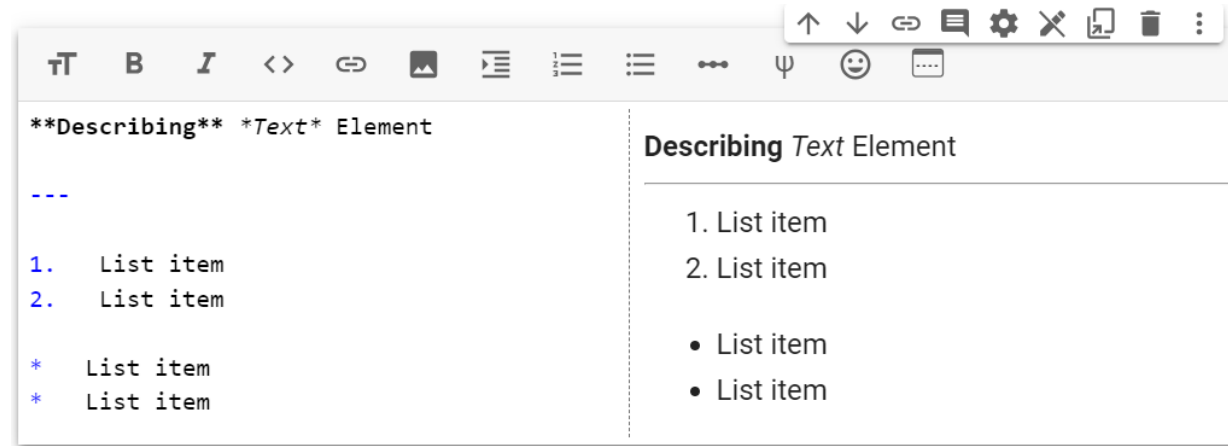
- An orange arrow pointing to the "Variables" panel.
- An orange arrow pointing to the "First Section" header in the code editor.
- A large orange box with the text: "Inspect run time (dynamically typed) variables; comment code elements with peers".
- An orange arrow pointing to the "Comment" button in the top right corner.

A comment thread is visible in the bottom right corner, showing a conversation about a football club.

At the bottom of the notebook, a status bar indicates "0s completed at 12:39 PM".

# Documentation

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



- Code: inline comment with preceding #.

```
#inline comment
```

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

```
'''multiple  
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 its power and popularity.
- Modules must be imported before they can be used.
- **Avoid** at all costs `from module import *`

```
#import libraries preferably in a separate cell  
#of running 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

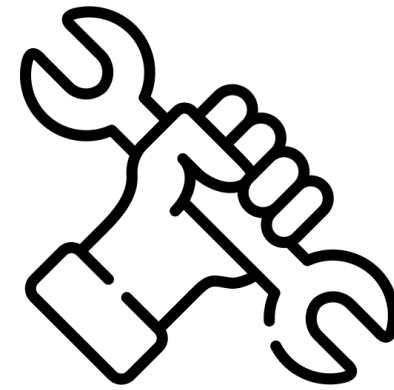
- Integers and floats work as you would expect from other languages.

```
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.

# Boolean

- 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.)

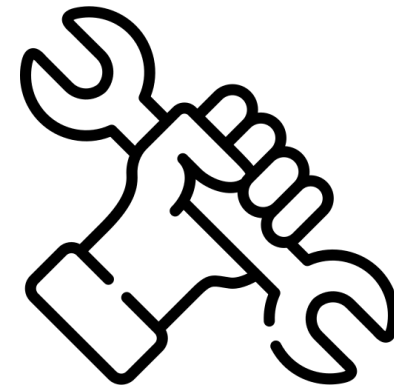
- Some very useful string methods:

```
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 length 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>

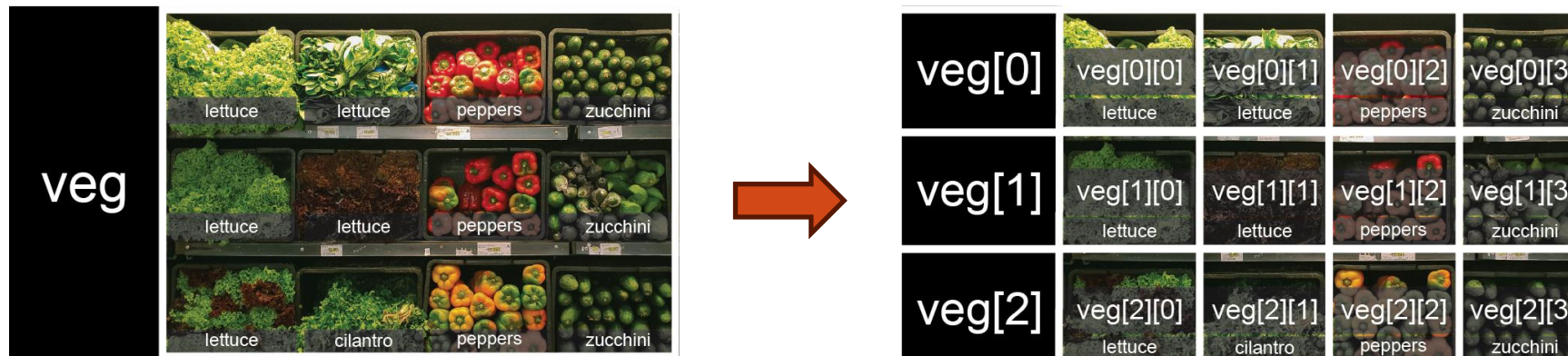
# Training #2



1. Assign "o jyVT TyamyknYMakLmktDYNtyklnyhmyrn" 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 **next week in person!**

Questions?