Python crash course for beginners

CENTRE DE RECHERCHE CERVO BRAIN RESEARCH CENTRE

Sandrine Poulin & Antoine Daigle

March 2024

Plan of this course

1. Setting up your Python environment
   Installation, IDE and Jupyter Notebook
2. The basics
   Syntax basics, control structures and function
3. Working with data
   Import and manipulate scientific data
   Modules used to present data
4. Online references
   Official documentation, ChatGPT, Stackoverflow
5. Project
   Analyse and visualise fiber photometry data

Setting up your Python environment

1. Installation
   - Python or Anaconda
2. IDE (Interactive Development Environment)
   - VSCode, Spyder
3. Jupyter Notebook
   - How to use

Python installation

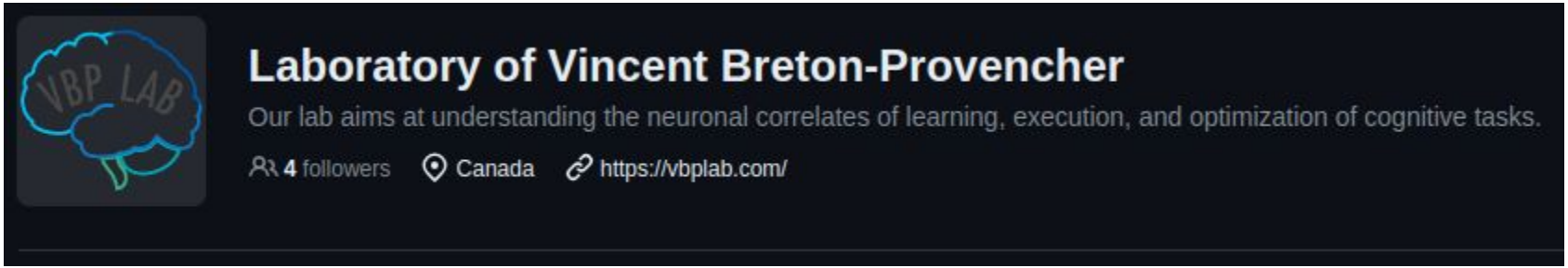
- There is different way to install Python on your computer
  - On Windows, by the Microsoft Store or web site - Download Python
  - By using Anaconda - Free Download | Anaconda (Recommended)

IDE installation

- You need an interface to code, there's two recommended choice:
  - VSCode - Setting up Visual Studio Code
  - Spyder - Comes with Anaconda
- We recommend VSCode
  - Git integration
  - Possibility to open other file types
  - Don't forget to add the Python extension in VSCode

Download the projects

Go on the [lab Github]() and download in a zip file the *Python-Crash-Course* repository.



GitHub

Jupyter Notebook

- Jupyter Notebook is like a code interpreter that is widely used to visualize data. Jupyter supports many coding languages, such as Python.
- Installation
  - Jupyter Notebook comes with Anaconda, if you do not have anaconda, it is fairly simple to install jupyter notebook.
  - From terminal: pip install jupyter notebook
- Launch:
  - From terminal: jupyter notebook

It's Fun O'Clock

ChatGPT warning

For this course, **do not use ChatGPT**.

While ChatGPT is a very useful tool when coding, an underlying goal of the course is to **make you think like a programmer**. You can use online references, but no generative AI, since it's doing the thinking for you.

The basics

1. Syntax structure
   Variables, Int, Float, Tuple, String, List, Dictionary
2. Control structures
   if, elif, else, for loop
3. Function
   Syntax and documentation

Variables

- Assign to a **name** a **value** or an **expression**.
- The **print function** allows us to see the result of the equation.
  - In this example, we evaluate the perimeter of a circle (p = 2πr) of a radius of 5.

```
pi = 3.1416
perimeter = 2 * pi * 5
print(perimeter)

>>>> 31.416
```

How to add comments to the code

- Add comments by using the “#” character
  - You can also comment and uncomment a block of code by selecting it and press a combination unique to your system (Mac, Linux, Windows).

```
pi = 3.1416 # Variable pi
perimeter = 2 * pi * 5  # Evaluation of the perimeter
# Let's print the result
print(perimeter)

>>>> 31.416
```

Naming variables

What is the syntax of the identifier you can use?

- a-z
- A-Z
- 0-9 (but not in first position)
- underline character “_”

It is conventional to write in lower case with underscores for better clarity.

```
pi = 3.1416
radius circle 5 = 5
perimeter = 2 * pi * radius_circle_5
```

print function

One of the most useful function in *Python*.

```
print(2, pi, 5)
print()
print('perimeter =', 2*pi*5)

>>>> 2 3.1416 5
>>>>
>>>>perimeter = 31.416
```

Numbers

There is 3 categories of numbers:

- Integers (**int**): …, -2, -1, 0, 1, 2, …
- Floating point number (**float**): 32.90871, …
- Complex numbers (complex)

```
a = 3.86
print(a, type(a))
b = int(a)
print(b, type(b))

>>>> 3.86 <class 'float'>
>>>>3 <class 'int'>
```

Arithmetic operators

Use those operators to manipulate numbers.

- +, -, *, / (addition, subtraction, multiplication and regular division)
- //, % (floor division, modulus)
- ** (exponential)

```
print("7/4 =", 7/4)
print("7//4 = ", 7//4)
print("7%4 =", 7%4)

>>>> 7/4 = 1.75
>>>> 7//4 =  1
>>>> 7%4 = 3
```

Strings

- String (str) is a type that allows us to manipulate text.
- You can use some operators on this type:
  - “+”: Add two or more strings together
  - “*”: Multiply the string sequence
  - “len()”: Determine the length of the string

```
x = "Bonjour l'monde"
print(x)
>>>> Bonjour l’monde
```

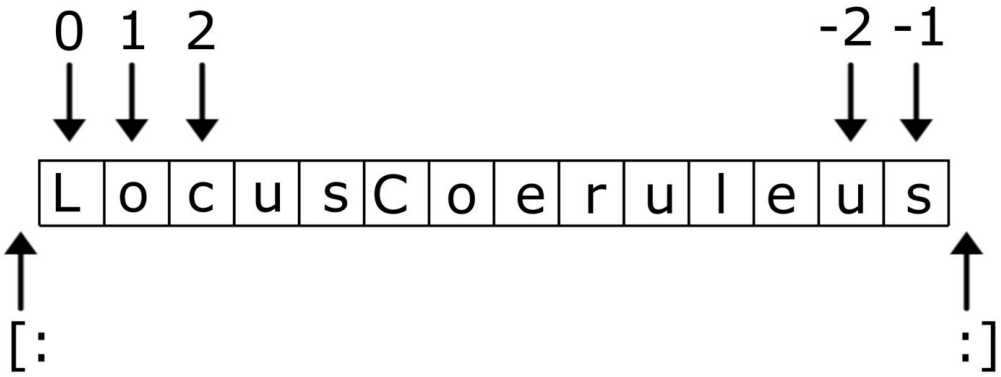
```
print(10 * "-")
print("-" * 3 + " Mouse ", "-" * 3)
print(len("Mouse"))
>>>> ----------
>>>> --- mouse  ---
>>>> 5
```

Index and slicing

By using the “[ ]” operator, we can index and slice a string (or a list).

- **[i]**: Get one element at this position “i”.
- **[i : j]**: Get all element between “i” and “j”.
- **[i : j : k]**: Between “i” and “j”, but jumping of “k” indexes each time.

```
a = "LocusCoeruleus"
print(a[1])
print(a[3:8])
print(a[-2:])
>>>> o
>>>> usCoe
>>>> us
```



ezercisse

Using the phrase:

"The;zebrafish;model;is;valid;and;great"

1. Replace the ; with spaces
2. Isolate the word zebrafish

Notes:

In python, there is a function to replace a character in a string with another one

```
yourstring.replace(character_to_replace,character_to_replace_with)
```

The list

Can stock object, data, … The notation of a list is the “[ ]” and the elements are separated by a comma. **Careful**, “[ ]” is used for list and indexing/slicing.

- You can index and slice a list.

```
a = [] # This is an empty list
b = [2, 3, "cervo", 5] # This is another list
print(a, b)
print(b[2])

>>>> [] [2, 3, 'cervo', 5]
>>>> cervo
```

Some function of the list

Multiple functions can be applied to the list, such as

- list.append()
- len(list)

```
a = []
a.append("Mouse")
print(a)

>>>> [“Mouse”]
```

```
b = [2, 3, "cervo", 5]
print(len(b))

>>>> 4
```

Dictionary

Another way to encapsulate data is with a dictionary. There are **two** components to every **element of a dictionary**. The *key* and the *value*. Basically, you call the key to get the value.

```
my_dic = {"KEY": "value"}
print(my_dic)

my_dic["neuron_type"] = ["VIP", "NDNF"]
print(my_dic)

>>>> {“KEY”: “value”}
>>>> {“KEY”: “value”, “neuron_type”: [“VIP”, “NDNF”]}
```

Some python modules (FaceMap, Suite2p, …) will give you their output in a dictionary.

Exercise

- Create an empty dictionary named “country_capital”.
  - Add 4 countries and their capitals as key-value pairs.
  - Check if the country “France” is in the dictionary.
  - Print all the keys of the dictionary.
  - Print all the values of the dictionary.

The function

The function is one of the most important concepts in python.

- Maximise code recycling
- Minimise bugs
- Increase readability and maintenance

Each function aims to **solve one step of the problem**.

```
def name(arg1, arg2, ... , argn):
    # Indented block with the code
    return expression # Facultative
```

A simple function

```
def addition(a, b):
    return a + b

print(addition(3, 5))

>>>> 8
```

Create documentation in your code

It's important if:

- you want to understand what you wrote 1 month before!
- You plan to share your code with someone.

```
def addition(a, b):
    """Function used to add two number
together.

    Args:
        a (int/float): First number.
        b (int/float): Second number

    Returns:
        float: The two number added together.
    """
    return a + b
```

Another example of documentation

```
def G(tau:float, N:float, tau d:float, r 0:float, z_0:float):
    """Diffusion model. Equation 2 of the paper.

    Args:
        tau (float): Lag time
        N (float): Average molecule number in detection volume
        tau d (float): Average time of molecules diffusing through the detection volume
        r_0 (float): Lateral distance over which the intensity decay in 1/e²
        z_0 (float): Axial distance over which the intensity decay in 1/e²

    Returns:
        float: G(tau)
    """
    return (1/N) * (1 /(1 + tau/tau_d)) * (1/np.sqrt(1 + (tau/tau_d) * (r_0**2/z_0**2)))
```

Conditional statement

There are three statements that allows us to implement conditions in the code.

- “if”: If the condition is True, execute this block of code. (required)
- “elif”: If the condition is True, execute this block of code. (optional)
- “else”:If no conditions is True, execute this block of code. (optional)

```
if expression 1:
    # Statement block 1

elif expression 2:
    # Statement block 2

elif expression n:
    # Statement block n

else:
    # Statement block n+1
```

Exercise: Conditional statements

Given a number (x):

if x is between 1 and 50, we want to print:

“The number is between 1 and 50”

if x is between 51 and 100, we want to print:

“The number is between 51 and 100”

in other cases (x is not between 1 and 100), we want to print:

“The number is not between 1 and 100”

For loop

- “Used to iterate over an iterable.” ~ Marc Parizeau
- When **you want to go over elements** in a list, tuple, string, …

```
for target in iterable:
    # Do something with the target
```

```
names = ["Enton", "Sendryn"]

for name in names:
    print(f"I appreciate {name} efforts to learn neuroscience.")

>>>> I appreciate Enton efforts to learn neuroscience.
>>>> I appreciate Sendryn efforts to learn neuroscience.
```

Range function: a great tool for iterating

If you need to iterate over a sequence of numbers, the built-in function range comes in handy. It generates arithmetic progressions:

```
print(range(3))
print(list(range(0, 10, 3)))

>>>> range(0, 3)
>>>> [0, 3, 6, 9]
```

```
for i in range(3):
    print(i)

>>>> 0
>>>> 1
>>>> 2
```

Enumerate function

When looping through a sequence, the position index and corresponding value can be retrieved at the same time with the enumerate() function.

```
cafeteria_menu = ["merlu", "hot hamburger", "guedille"]
for menu_index, menu_item in enumerate(cafeteria_menu):
    print(menu_index, " is ", menu_item)

>>>> 0 is merlu
>>>> 1 is hot hamburger
>>>> 2 is guedille
```

```
cafeteria_menu = ["merlu", "hot hamburger", "guedille"]

for k in range(len(cafeteria_menu)):
    print(k, "is", cafeteria_menu[k])

>>>> 0 is merlu
>>>> 1 is hot hamburger
>>>> 2 is guedille
```

The debugger

This is a tool to help debug your code.

- It allows you to go step by step.
- Use breakpoints to stop the code.

References:

- Debugging configurations for Python apps in Visual Studio Code
- Debugger — Spyder 5 documentation

break, pass and continue

- **break:** This statement breaks out of the innermost enclosing of the loop.
- **pass:** Used as a placeholder: nothing happens and the rest of the loop is executed as usual.
- **continue:** Continue to the next iteration of the loop

```
for num in range(10):
    print(num)
    if num == 2:
        break

>>>> 0
>>>> 1
>>>> 2
```

```
for num in range(2, 5):
    if num % 2 == 0:
        print("Found an even number", num)
        continue
    print("Found an odd number", num)

>>>> Found an even number 2
>>>> Found an odd number 3
>>>> Found an even number 4
```

Recap exercise

You have this list of dna sequences:

```
# Input
dna_sequences = ["ATCGA", "TTAAGC", "CGATG"]

# Output
# GC content for each DNA sequence respectively
# [40.0, 33.33333333333333, 60.0]
```

Create a function that calculates the ratio of guanine and cytosine in the DNA. The function must take a string as argument and return a float.

Built-in functions

Python itself has a number of functions and types built into it that are always available.

- range()
- enumerate()
- int(), float(), …
- abs()
- max(), min()
- print()
- …

Those functions are great, but so much more can be done with modules!

Built-in Functions

A	**E**	**L**	**R**
abs()	enumerate()	len()	range()
aiter()	eval()	list()	repr()
all()	exec()	locals()	reversed()
anext()			round()
any()	**F**	**M**	
ascii()	filter()	map()	**S**
	float()	max()	set()
B	format()	memoryview()	setattr()
bin()	frozenset()	min()	slice()
bool()			sorted()
breakpoint()	**G**	**N**	staticmethod()
bytearray()	getattr()	next()	str()
bytes()	globals()		sum()
		O	super()
C	**H**	object()	
callable()	hasattr()	oct()	**T**
chr()	hash()	open()	tuple()
classmethod()	help()	ord()	type()
compile()	hex()		
complex()		**P**	**V**
	I	pow()	vars()
D	id()	print()	
delattr()	input()	property()	**Z**
dict()	int()		zip()
dir()	isinstance()		
divmod()	issubclass()		**_**
	iter()		__import__()

Modules

Modules contains pre-coded functions! There are modules for almost everything:

- Math (numpy, math, scipy, sympy, …)
- Visualisation (matplotlib, seaborn, PyQt5, turtle, …)
- Data & data acquisition (pyserial, pandas, …)

You need to install the packages to use them.

- With anaconda: command “conda install XXXX” in the terminal
- With pip: PyPi or with the command “pip install XXXX”

Working with data

1. Import data
   Open, Os module and Pandas module
2. Manipulate data
   Pandas and Numpy modules
3. Visualise data
   Matplotlib and Seaborn module

Open a file with the standard library

If you want to read, write or create a file. The basic example.

```
# Read the content of the file
file_path = "Write the path of your file"
with open(file path, "r") as file:
   file_content = file.read()

print("Content read from file:", file_content)
```

The **OS** module

This module is used to manipulate files and paths on your machine. You can:

- List files in a directory
- Rename files and folder
- Join some paths
- …

If you want to open multiple files, use the OS module to list the files and the open() function to extract data.

```
# Get the current working directory
current_dir = os.getcwd()
print("Current working directory:", current_dir)
```

Exercise: Find a file and write a string

In this exercise, we want to get the path of a text file on your desktop and write in the file:

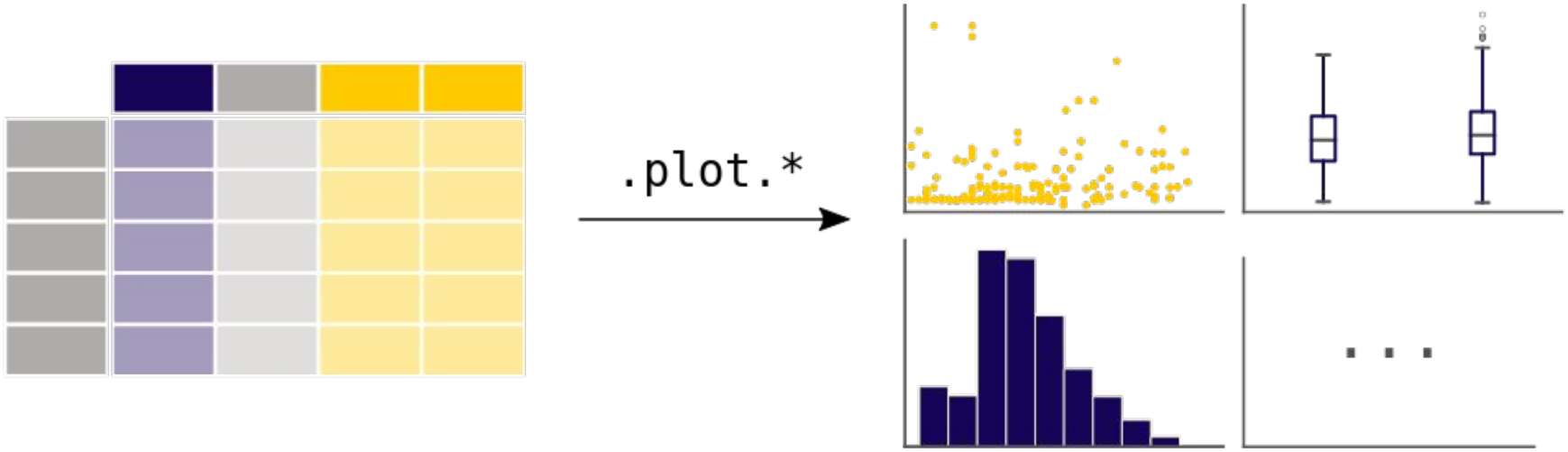
"I hope we have 'Merlu en croute' for lunch. I love 'Merlu en croute'."

1. Create an empty .txt file on your desktop
2. Use the OS module and open() method.

The **Pandas** module

Pandas allows you to work with a dataframe just like in excel.

- Load a csv or xlsx file and work with the column name or localisation.

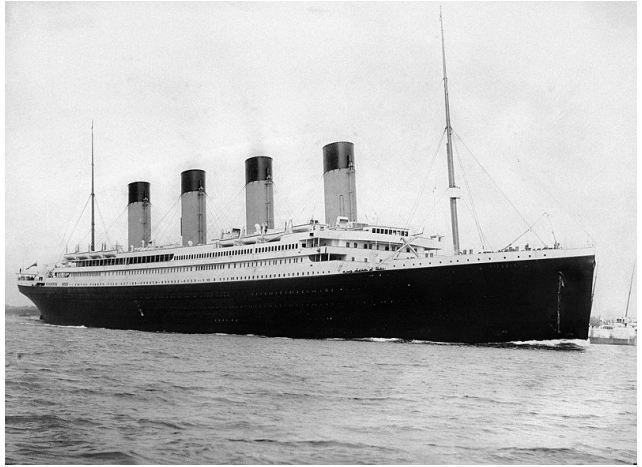


Exercise: Manipulation and plot

With Pandas, load the titanic dataset. We want:

- Print the name of the columns
- Print a sample, we want the data on one passenger.
- Print basic statistics of the ages.

Search the web to find the correct methods.



The **Numpy** module

The fundamental [package] for scientific computing. Allows you to work with an array (matrix). It's like a boosted list.

- You can apply mathematical operations directly on them.
- You can generate numbers.

```
data = np.array([1,2])
```

data
1
2

data		ones		
1	-	1	=	0
2		1		1

data		data		
1	*	1	=	1
2		2		4

data		data		
1	/	1	=	1
2		2		1

Numpy 2D and some functions

```
np.array([[1,2],[3,4],[5,6]])
```

1	2
3	4
5	6

data

	0	1
0	1	2
1	3	4
2	5	6

data[0,1]

	0	1
0	1	2
1	3	4
2	5	6

data[1:3]

	0	1
0	1	2
1	3	4
2	5	6

data[0:2,0]

	0	1
0	1	2
1	3	4
2	5	6

data

1	2
3	4
5	6

.max() = 6

data

1	2
3	4
5	6

.min() = 1

data

1	2
3	4
5	6

.sum() = 21

Exercise: Generate time stamps

Generate an array of integers between 1 and 15.

Generate 1000 points between 0 and 30.

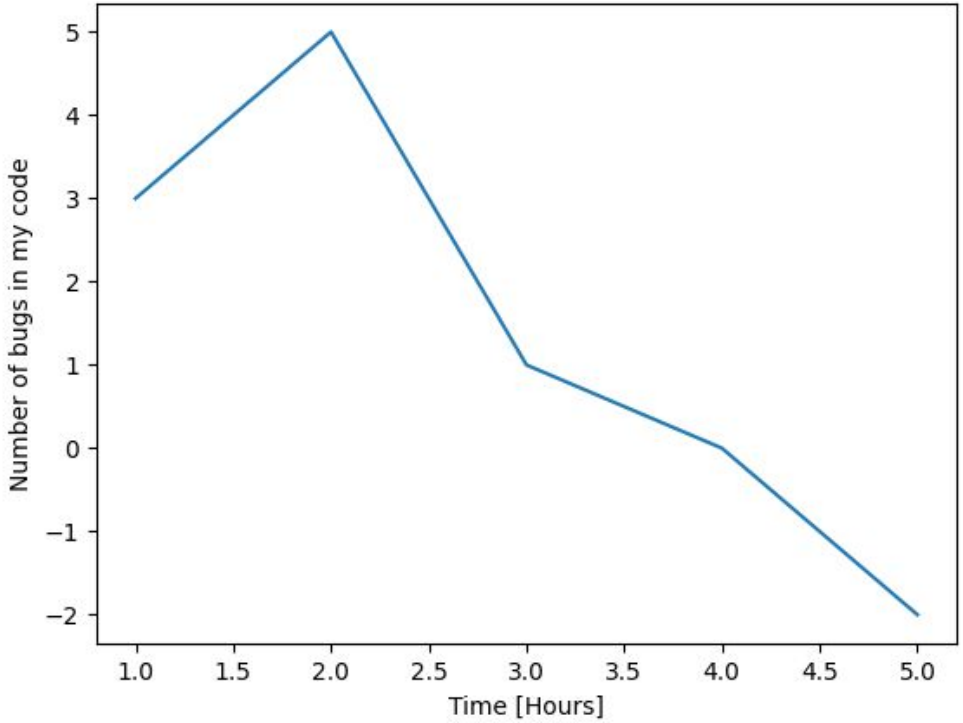
For each array you generate, give us the:

- Mean
- STD
- Max
- Min
- Median

What happens if in the array there is a Nan (Not a number). Is there a simple solution?

The **Matplotlib** module

This module is used to plot and visualise data.

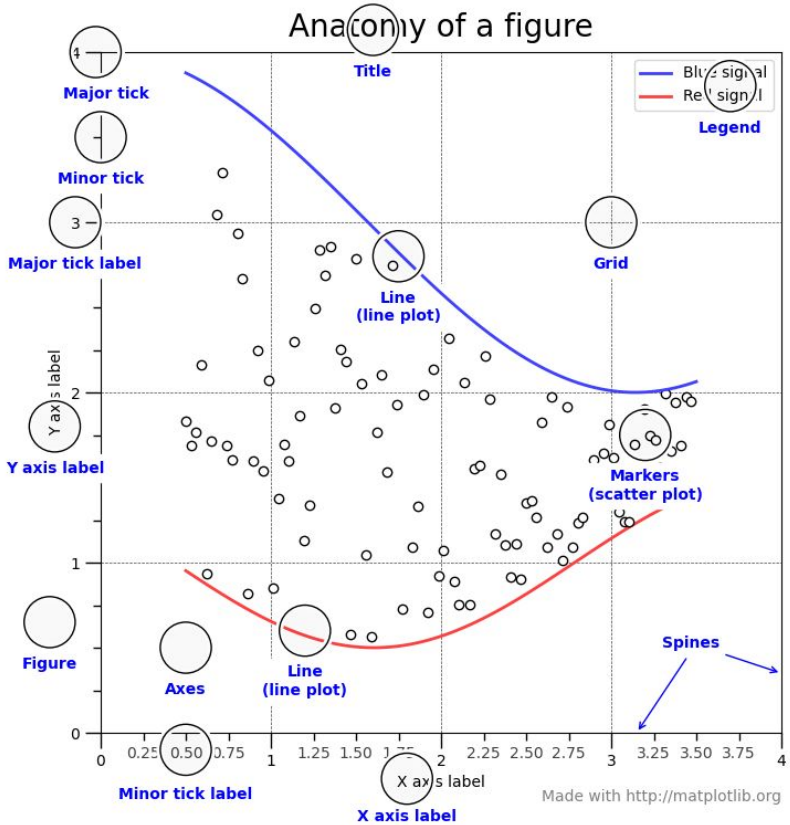



```
import matplotlib.pyplot as plt

x = [1, 2, 3, 4, 5]
y = [3, 5, 1, 0, -2]

plt.plot(x, y)
plt.xlabel("Time [Hours]")
plt.ylabel("Number of bugs in my code")
plt.show()
```

Anatomy of a figure



1 Initialize

```
import numpy as np
import matplotlib.pyplot as plt
```

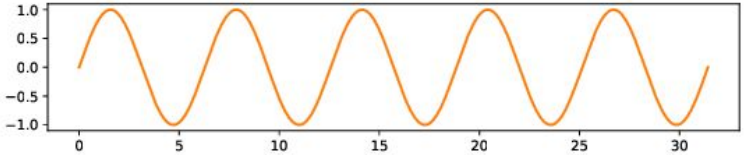
2 Prepare

```
X = np.linspace(0, 10*np.pi, 1000)
Y = np.sin(X)
```

3 Render

```
fig, ax = plt.subplots()
ax.plot(X, Y)
plt.show()
```

4 Observe



Exercise

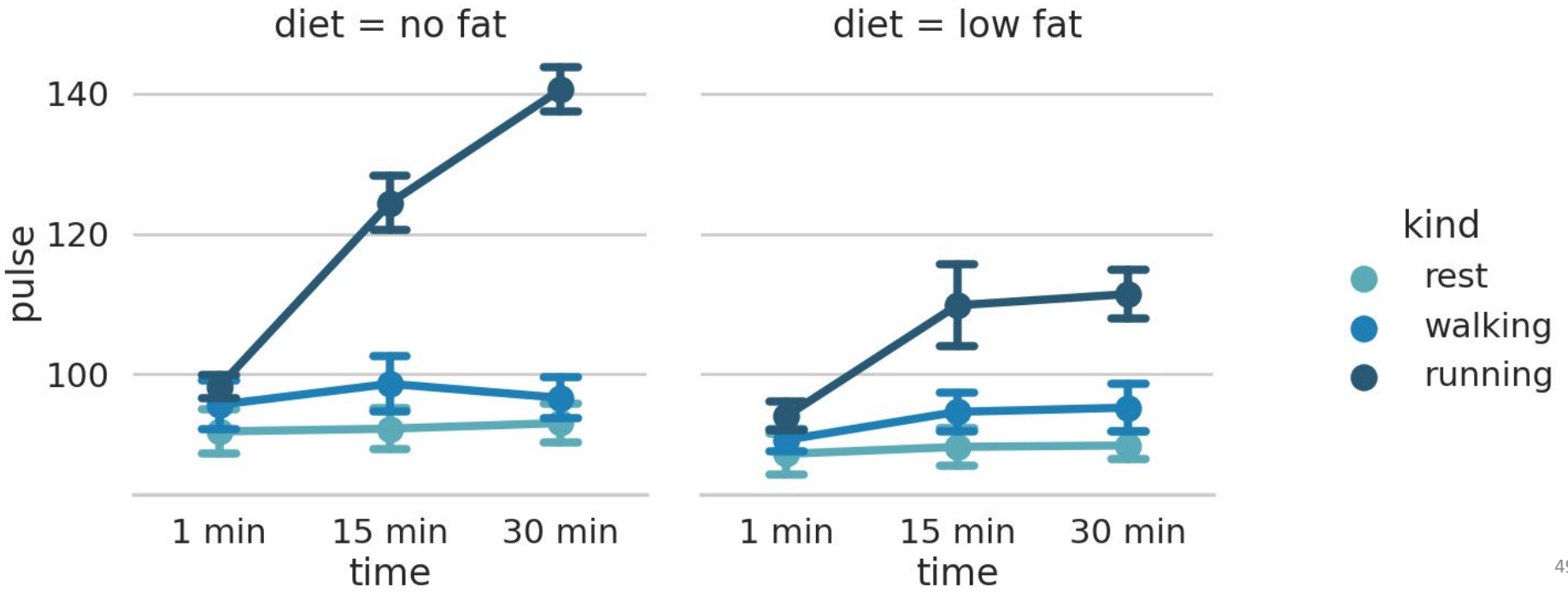
Generate a simple plot with 2 lines and a scatter plot. For each set, add and change those parameters:

- Axis name
- Legend
- Line colors
- Scatter markers
- Title of the plot

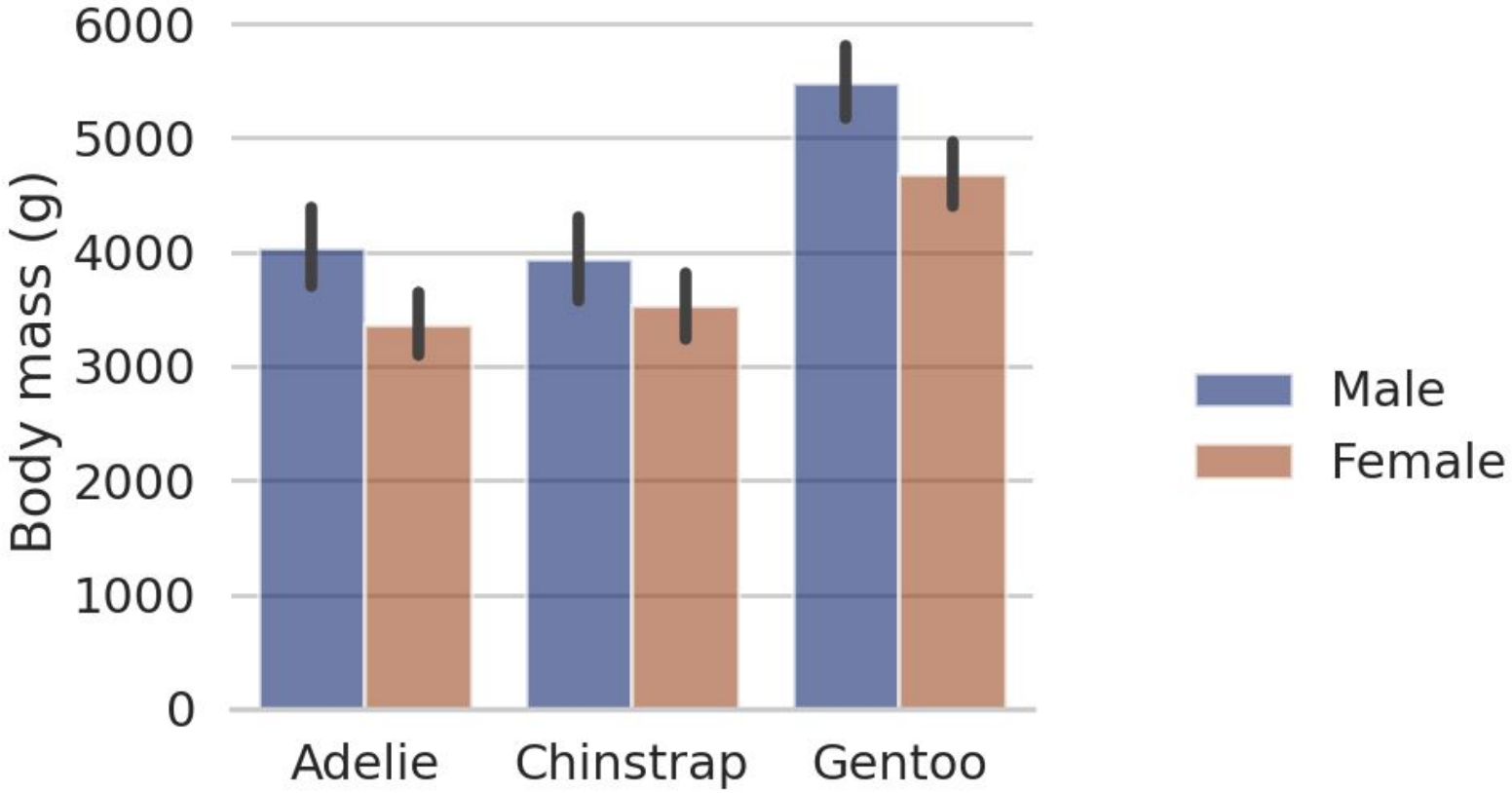
Also, save the plot in a vector graphics format (pdf or svg).

The **Seaborn** package

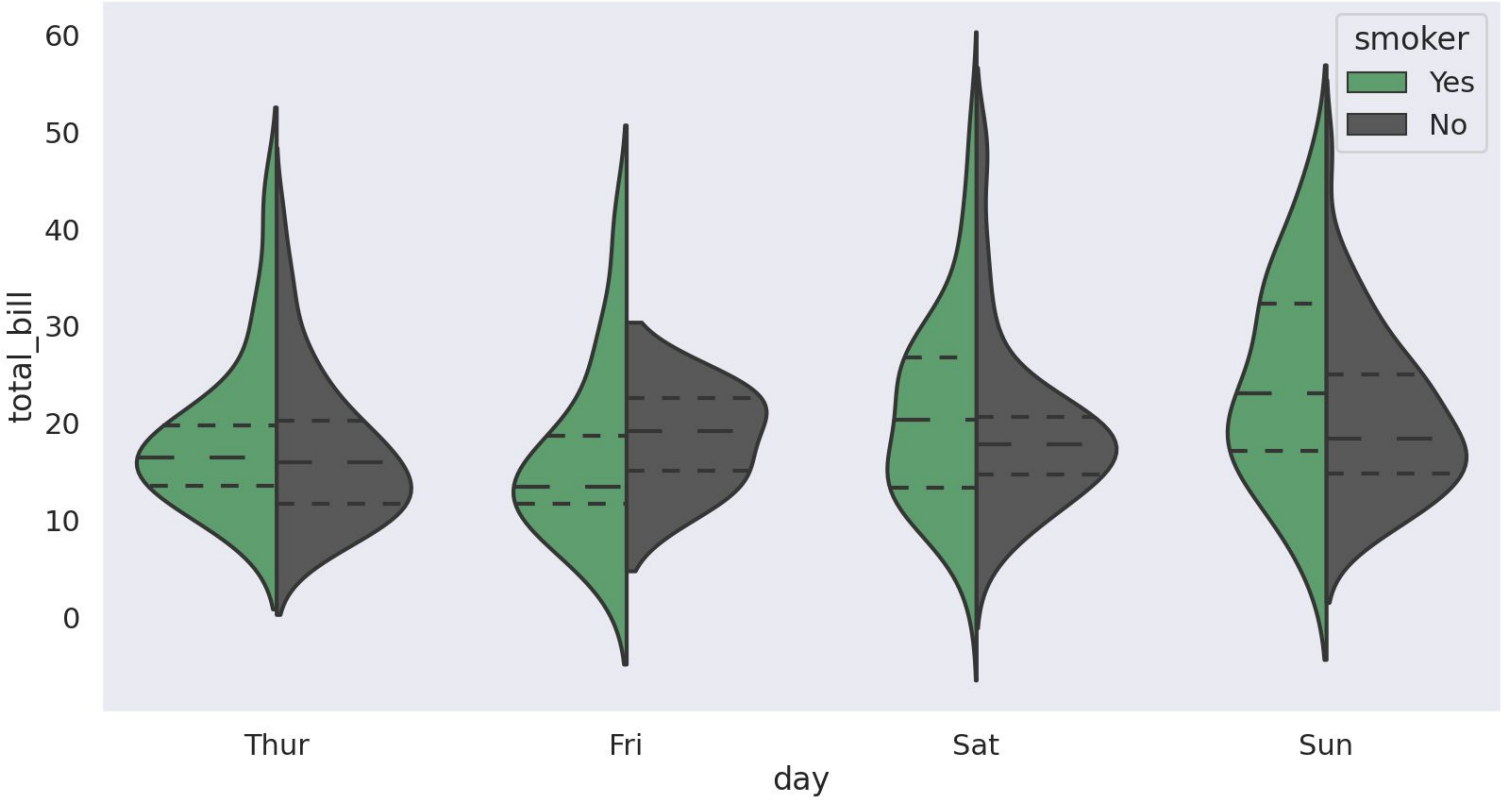
This package allows for more visualisation option. It's more high level compared to Matplotlib, but still simple.



Seaborn examples



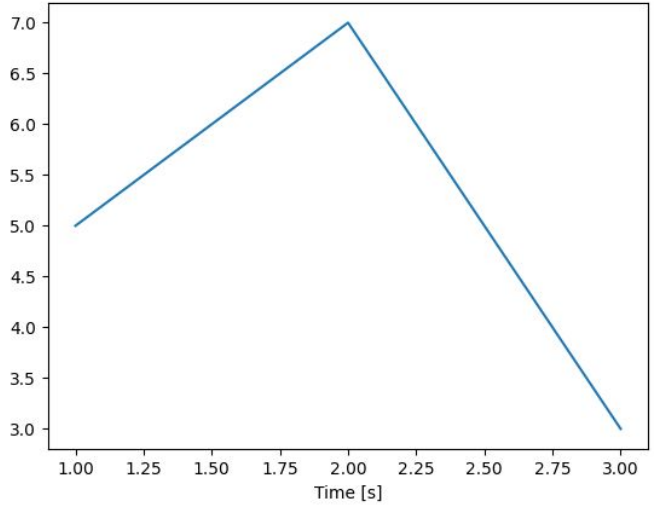
Another Seaborn example



Seaborn command you can already use

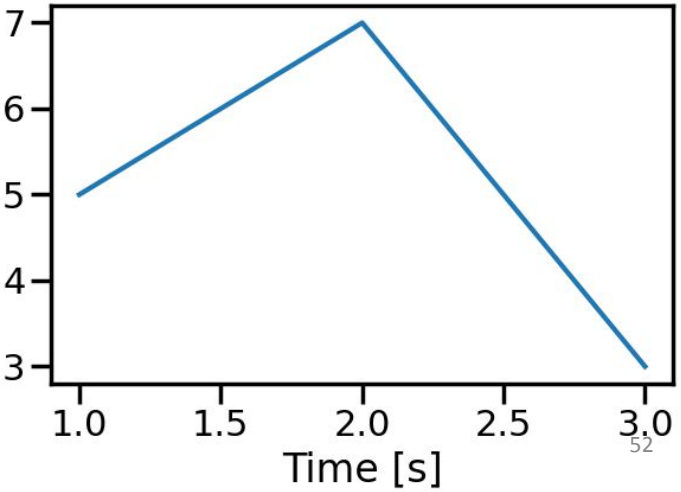
Seaborn and matplotlib are cross-compatible. You can set a context to the graph.

- Easier visualisation of your graph for poster, presentation, …
- Different context available (and styles).




```
import matplotlib.pyplot as
plt
import seaborn as sns

# sns.set context("poster")
plt.plot([1, 2, 3], [5, 7,
3])
plt.xlabel("Time [s]")
plt.tight layout()
plt.show()
```



Online references

1. Official documentation
2. ChatGPT
3. Stackoverflow

Online resources

What are the online resources that can help you?

- ChatGPT
  - Generative AI, useful to get ideas and optimise your code.
  - Always double check.
- Stackoverflow
  - Forum that is massively used by the community.
  - Someone already had your question.
- The documentation
  - The most important resource.
  - Learning to read the documentation will help you in the long term.

How to read the documentation

The documentation is available

- on their website
- in your IDE (hold the cursor over the function).

numpy.mean

Name of the function

Arguments of the function

```
numpy.mean(a, axis=None, dtype=None, out=None, keepdims=<no value>, *,
where=<no value>)
```

[source]

Small description of the function

Compute the arithmetic mean along the specified axis.

Returns the average of the array elements. The average is taken over the flattened array by default, otherwise over the specified axis. `float64` intermediate and return values are used for integer inputs.

Parameters: a : *array_like*

Array containing numbers whose mean is desired. If *a* is not an array, a conversion is attempted.

axis : *None or int or tuple of ints, optional*

Axis or axes along which the means are computed. The default is to compute the mean of the flattened array.

New in version 1.7.0.

Detailed description of the function

Exercise

Find the documentation for the;

- numpy linspace() function.
  - What is the purpose of this function?
  - How many point will this function generate by default?
- print() function.
  - Can I change the end character of the print() function?
  - Someone said I can write in a file with the print() function. Is it true?

How to read a Stackoverflow question

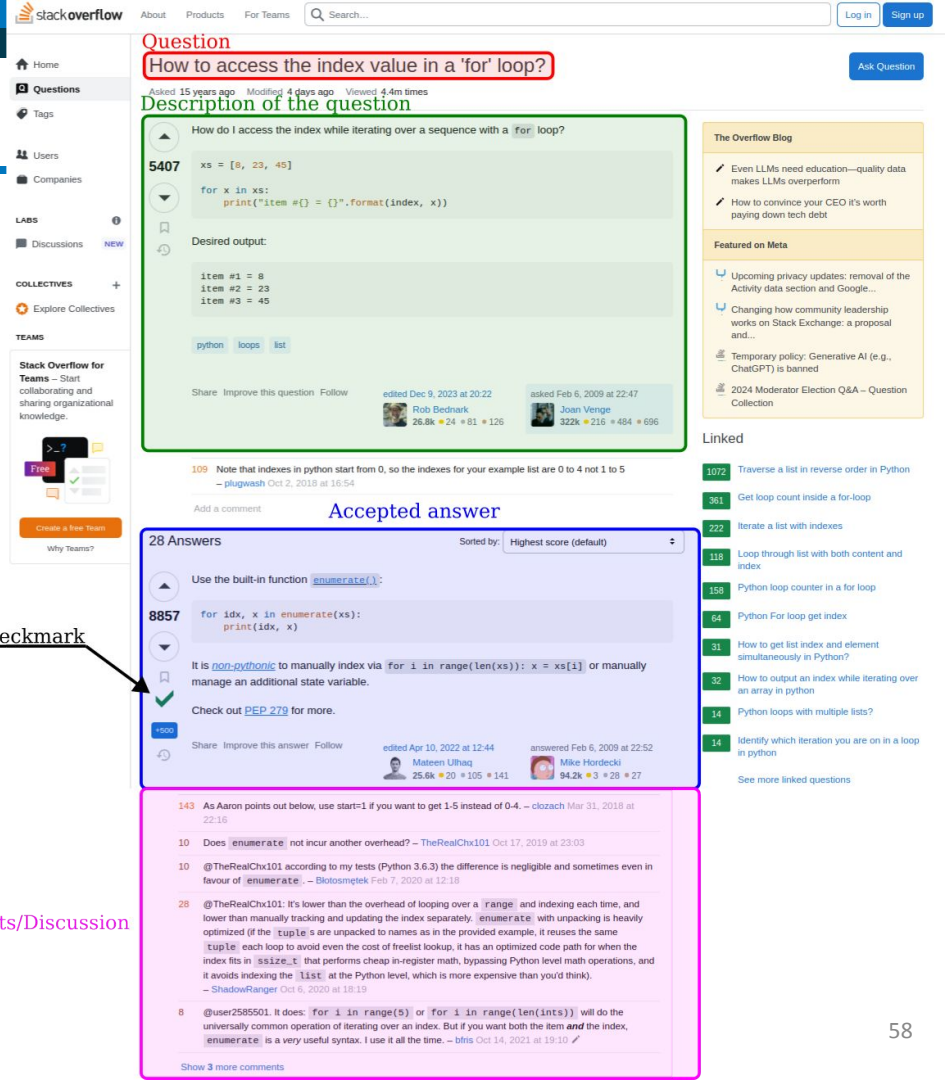
Stackoverflow is the main website for all of your questions:

- Over 2 186 475 questions tagged with *Python* (from 27/02/2024).
- Multiple people can propose answer. The accepted answer have a green checkmark.
- The first post is the question (the code is not working).
- Multiple proposed answer can work.

Example with Stackoverflow

[Link of this thread]

Notice the details of some answers and how active this old question is!



Final tips

- Comment your code (please)
- Give your variables explicit names (I beg you)
- Write functions
- Always test what you wrote (assume you are always wrong)
- Use the debugger
- Small simple steps are always good.

Projects

1. Fused fiber photometry
2. Action potential
3. FRAP simulation analysis

The projects

We propose three different project.

1. Fused fiber photometry analysis
2. Action potential
3. FRAP simulation analysis

Each project comes with a solution. Try it yourself before looking up the answer.

The goal of those project is to make you *think* like a programmer in front of a task. You can **use everything except ChatGPT**.