

Introduction to Python

Outline

- 1 Python Locally
- 2 Google Colab
- 3 Python basics
- 4 NumPy & Matplotlib

What is Anaconda?

- Anaconda is a free distribution of Python
- Includes:
 - Python interpreter
 - Popular scientific libraries (NumPy, pandas, matplotlib, etc.)
 - Package and environment manager (conda)
- Available for Windows, macOS, and Linux



Why Use Anaconda?

- Easy installation with minimal setup
- Manages packages and dependencies automatically
- Allows multiple isolated Python environments
- Widely used in data science and research



Step 1: Download Anaconda

- 1 Go to: <https://www.anaconda.com/download/success>
- 2 Choose your operating system:
 - Windows
 - macOS
 - Linux
- 3 Click **Download**
- 4 Follow this guide here

Step 2: Install (Windows / macOS)

- Run the downloaded installer
- Follow the setup wizard
- Recommended options:
 - Accept the license agreement
 - Use default installation location
 - (Windows) Check “Add Anaconda to PATH” only if you know what it means
- Click **Install**

Step 2: Install (Linux)

- 1 Open a terminal
- 2 Navigate to the download folder
- 3 Run `bash Anaconda3-*.sh`
- 4 Follow the prompts and accept the license
- 5 Restart the terminal after installation

Step 3: Verify Installation

- Open a terminal (or Anaconda Prompt on Windows)
- Check conda: `conda --version`
- Check Python version: `python --version`
- If versions are shown, installation was successful

Step 4: Create a New Environment

- Create a new environment: `conda create -n myenv python=3.11`
- Activate the environment: `conda activate myenv`
- Keeps projects and dependencies isolated.
- Makes it easy to reproduce experiments and share environments

Using VS Code with Anaconda

- Visual Studio Code (VS Code) is a lightweight code editor
- Works well with Python and Anaconda environments
- Supports debugging, linting, and Jupyter notebooks



Visual Studio Code

Step 1: Installation VS Code

- ➊ Go to: <https://code.visualstudio.com/>
 - ➋ Download the version for your operating system
 - ➌ Run the installer and follow the instructions
- Open VS Code
 - Go to the Extensions panel
 - Search for **Python** (by Microsoft)
 - Click **Install**
 - Enables Python support, debugging and Jupyter

Step 2: Select the Anaconda Environment

- Open the Command Palette:

`Ctrl+Shift+P` (Windows/Linux)

`Cmd+Shift+P` (macOS)

- Select Python: Select Interpreter
- Choose the desired conda environment

Step 3: Run Python Code in VS Code

- Create a new file: `example.py`
- Write Python code
- Run the file using:
 - The **Run** button
 - Or the integrated terminal:

```
python example.py
```

Using the Integrated Terminal

- Open terminal in VS Code:

View → Terminal

- Automatically activates the selected conda environment
- Allows use of:

```
conda install, pip install
```

Using Jupyter Notebooks in VS Code

- Create a new file: `notebook.ipynb`
- VS Code opens a notebook interface
- Select the Anaconda environment as the kernel
- Run cells interactively

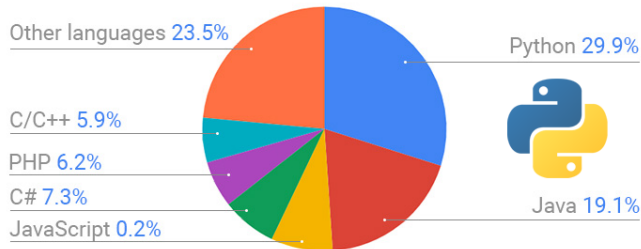
What is Google Colab?

- **Google Colab** (Colaboratory) is a free cloud-based Jupyter notebook environment.
- It allows users to write and execute Python code in the browser.
- No setup required, runs entirely on Google Drive.
- Seamless integration with Google Drive and GitHub.
- Enables collaboration by sharing notebooks.
- Supports popular ML libraries like TensorFlow and PyTorch.
- Provides free access to GPUs and TPUs.



Why Python?

- It is one of the most popular programming languages in the world.
- It is easy to learn and has a simple syntax.
- It is widely used in various fields such as data science, web development, and automation.

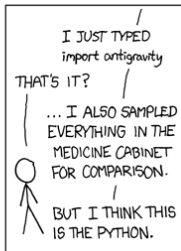
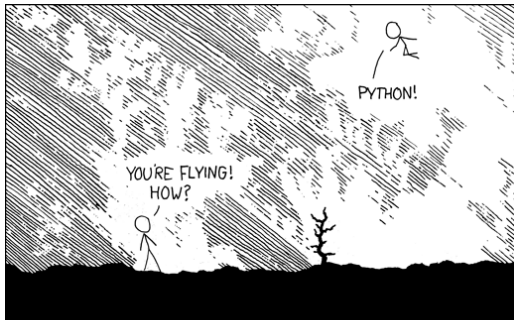


Why Python?

- It is one of the most popular programming languages in the world.
- It is easy to learn and has a simple syntax.
- It is widely used in various fields such as data science, web development, and automation.

Extensive libraries and frameworks for different applications.

- Data Science and Machine Learning (Pandas, NumPy, TensorFlow)
- Web Development (Django, Flask)
- Scripting and Automation
- Cybersecurity and Ethical Hacking

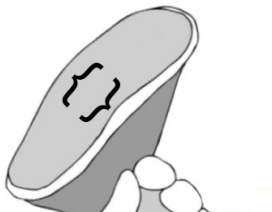


Example: Printing a Message

```
print("Hello world!")
```

- Variables can have any name but they have start with a letter.
- Case-sensitive: the two variables `x` and `X` are not the same.
- A variable is assigned using the symbol `=`.
- A variable can be deleted using the command `del`.
- No need for semicolons or curly braces.
- Indentation is used to define blocks of code.
- `help(function)` prints a description of the function.

Ew, I stepped in shit.



Data Types in Python

- Numbers (int, float, complex)
 - Boolean
 - Strings
 - Lists
 - Tuples
 - Dictionaries
 - null value
- `a=1 b=1. c=1+0j`
 - `a=True b=False`
 - `a='Alice' or a="Alice"`
 - `a=[1, 'a', []]`
 - `a=(1, 'a', [])`
 - `a={'model': 'Mustang', 'year': 1964}`
 - `a=None`

The command `type` returns the type of a variable given in input.

Tuples/Lists in Python

- Tuples are ordered, immutable collections of items.
- Lists are ordered, mutable collections of items.
- They can hold different data types.

Example: Creating a Tuple

```
my_tuple = (1, 2, 3, "apple", "banana")  
print(my_tuple[1])
```

Example: Creating a List

```
my_list = [1, 2, 3, "apple", "banana"]  
print(my_list[0])
```



Example: If Statement

```
x = 10
if x == 5:
    print("x is 5")
elif x<5:
    print("x is smaller than 5")
else:
    print("x is a bigger than 5")
```

- Python uses indentation instead of braces for blocks.

Loops in Python

Example: For Loop

```
for i in range(5):  
    print(i)
```

- Used to iterate over sequences (lists, tuples, strings, etc.).

Example: while Loop

```
i = 1  
  
while i <= 5 :  
    print(i)  
i += 1
```

- Used to repeat a block of code until a certain condition is met.

Operators syntax

Arithmetic Operators

+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation
%	modulus/remainder
//	floor/integer division

Relational operators

==	equal to
!=	not equal to
>	greater than
>=	greater than or equal
<	less than
<=	less than or equal

Logical Operators

not	negation
and	AND
or	OR
is	identity
in	membership

Functions in Python

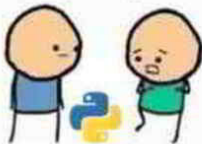
Example: Defining a Function

```
def mean(x,y):  
    m = (x + y)/2  
    return m
```

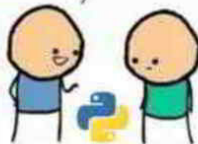
```
mean(1,5)
```

- Functions help in code reusability and organization.
- Local variables can only be used inside the function.
- Global variables can be modified inside a function.

Does your
python bite?



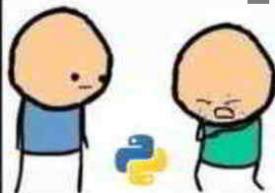
No, but it can hurt you
in other ways.



Indentation Error:
Expected an indented block



He doesn't.



DEVS. LOL

Classes in Python

- Python supports object-oriented programming (OOP).
- A class is a blueprint for creating objects.
- Objects are instances of classes with attributes and methods.

Defining a Class

Example: Creating a Class

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def greet(self):
        print(f"Hello, my name is {self.name}.")

p1 = Person("Alice", 25)
p1.greet()
```

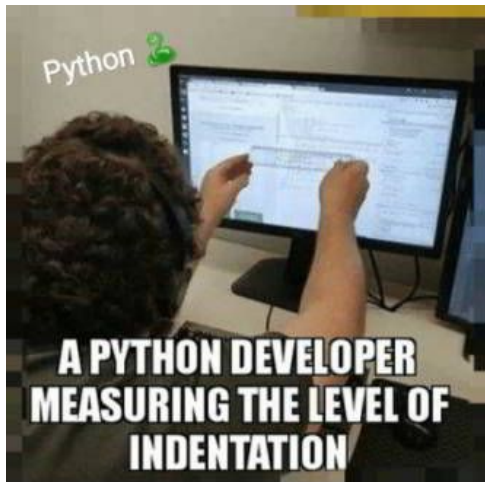
- The `__init__` method initializes the object's attributes.
- The `greet` method prints a greeting message.

Inheritance in Python

- Inheritance allows one class to inherit properties from another.
- It promotes code reusability and organization.

Example: Inheriting a Class

```
class Student(Person):  
    def __init__(self, name, age, grade):  
        super().__init__(name, age)  
        self.grade = grade  
  
    def show_grade(self):  
        print(f"{self.name} is in grade {self.grade}.")  
  
s1 = Student("Bob", 20, "A")  
s1.greet()  
s1.show_grade()
```

Indentation checking in Python
is like:

Introduction to NumPy

- NumPy (Numerical Python) is a powerful library for numerical computing.
- It provides support for large, multi-dimensional arrays and matrices.
- Offers a collection of mathematical functions to operate on arrays efficiently.

Why Use NumPy?

- Faster computations compared to Python lists.
- Supports multi-dimensional arrays.
- Provides efficient mathematical operations.
- Widely used in data science, machine learning and scientific computing.



Creating NumPy Arrays

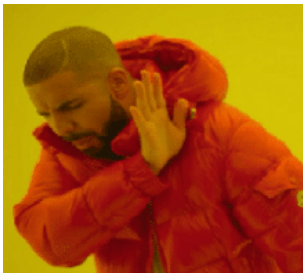
- NumPy arrays are created using the `numpy.array()` function.
- Supports different data types (integers, floats, etc.).
- Allows efficient vectorized operations.

Example: Creating an Array

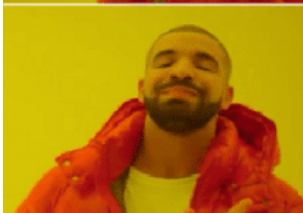
```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])
print(arr)

arr_2d = np.array([[1, 2, 3], [4, 5, 6]])
print(arr_2d)
```



```
1 import numpy
```



```
1 import numpy as np
```

There is no other way

Array Operations

- Element-wise addition, subtraction, multiplication, and division.
- Supports broadcasting for operations on different-sized arrays.
- Provides functions like `np.sum()`, `np.mean()`, `np.max()`, etc.

Example: Array Arithmetic

```
a = np.array([1, 2, 3])  
b = np.array([4, 5, 6])  
  
print(a + b)  # [5 7 9]  
print(a * b)  # [4 10 18]  
print(np.mean(a))  # 2.0
```

Indexing and Slicing

- Access elements using indices (similar to lists).
- Slicing allows extracting subarrays.
- Supports negative indexing and step slicing.

Example: Indexing and Slicing

```
arr = np.array([10, 20, 30, 40, 50])
```

```
print(arr[1]) # 20
```

```
print(arr[1:4]) # [20 30 40]
```

```
print(arr[-1]) # 50
```

Reshaping and Transposing

- Reshaping allows changing the shape of an array.
- Transposing swaps rows and columns in a 2D array.

Example: Reshaping and Transposing

```
arr = np.array([1, 2, 3, 4, 5, 6])  
reshaped = arr.reshape(2, 3)  
print(reshaped)  
  
transposed = reshaped.T  
print(transposed)
```

Random Numbers with NumPy

- NumPy provides a random module for generating random numbers.
- Supports uniform, normal, and other probability distributions.

Example: Generating Random Numbers

```
np.random.seed(42)  # Set seed for reproducibility
random_numbers = np.random.rand(5)  # Generate 5 random numbers
print(random_numbers)

random_matrix = np.random.randint(1, 10, (3,3))
print(random_matrix)
```


Introduction to Matplotlib

- Matplotlib is a popular Python library for data visualization.
- It allows creating static, animated, and interactive plots.
- Easy to create a variety of plots (line, bar, scatter, histogram, etc.).
- Customizable plots with labels, colors, and legends.
- Works well with NumPy and pandas for data visualization.



Creating a Simple Plot

- Use the `matplotlib.pyplot` module to create plots.
- Define `x` and `y` values and plot them.

Example: Simple Line Plot

```
import matplotlib.pyplot as plt
```

```
x = [1, 2, 3, 4, 5]
```

```
y = [10, 20, 25, 30, 40]
```

```
plt.plot(x, y)
```

```
plt.xlabel('X-axis')
```

```
plt.ylabel('Y-axis')
```

```
plt.title('Simple Line Plot')
```

```
plt.show()
```

Different Types of Plots

- Line Plot: `plt.plot()`
- Bar Chart: `plt.bar()`
- Scatter Plot: `plt.scatter()`
- Histogram: `plt.hist()`

Example: Bar Chart

```
categories = ['A', 'B', 'C', 'D']  
values = [5, 7, 3, 8]
```

```
plt.bar(categories, values, color='blue')  
plt.xlabel('Categories')  
plt.ylabel('Values')  
plt.title('Bar Chart Example')  
plt.show()
```

Customizing and Saving Plots

- Change colors, line styles, markers, and legends.
- Add grid, annotations, and adjust axis limits.

Example: Custom Line Plot

```
plt.plot(x, y, color='red', linestyle='--', marker='o')  
plt.grid(True)  
plt.legend(['Data'])  
plt.show()
```

- Plots can be saved as images using `plt.savefig()`.
- Supports formats like PNG, JPG, and PDF.

Example: Saving a Plot

```
plt.plot(x, y)  
plt.savefig('plot.png')
```

