Python Programming

What is Python?

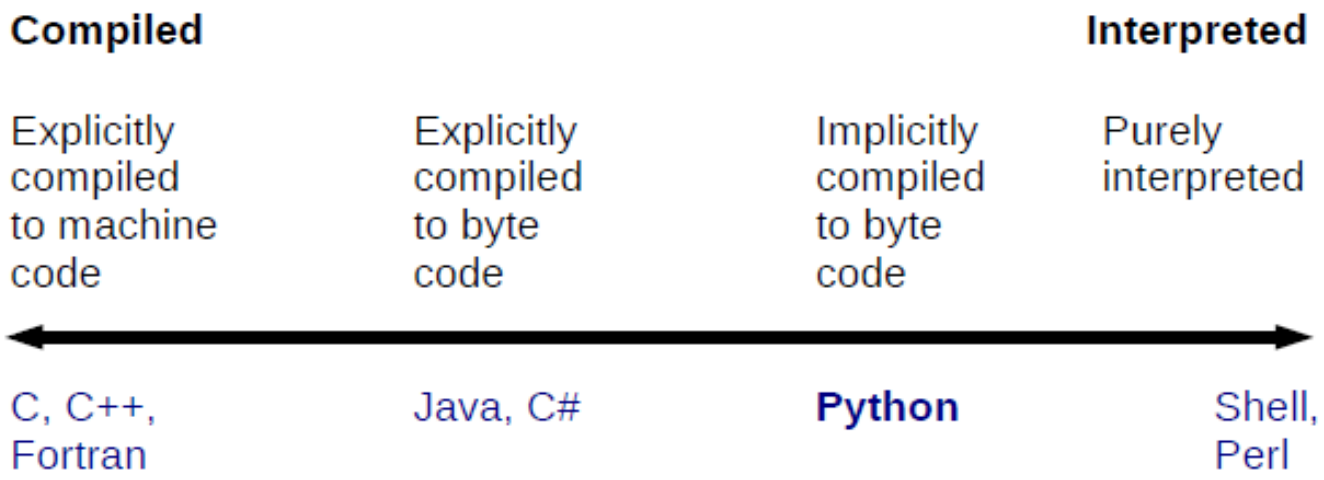
- General purpose, high level, and object-oriented programming language
- Interpreted and interactive language
- Dynamically and Strongly typed
- Created by Guido van Rossum, released in 1991
- Works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc)
- Easy to Learn and Maintain
- IDE support (Spyder, Pycharm, Netbeans or Eclipse)
- Open-source and maintained by the **Python Software Foundation**

Python Characters

- Python supports multiple programming paradigm
  - ✓ Structured programming
  - ✓ Object oriented programming
  - ✓ Functional programming

- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It uses **indentation** to highlight the blocks of code
- Can be used for web, desktop, data science, automation, AI, etc
- Thousands of third-party libraries (e.g., NumPy, Django, TensorFlow)
- Strong documentation and community support

Python Implementations



Python Installation

Python 3.10 — Best balance of compatibility, stability, and support.

Step 1: Download Python 3.10 Installer

Go to the official Python website:
https://www.python.org/downloads/release/python-3100/

Scroll down and select the Windows installer and Download:

Step 2: Run the Installer
Goto Downloads folder, Double-click and install it

Step 3: Configure Installation
Important: Select Check box:

"Add Python 3.10 to PATH"

Step 4: Complete Installation
Wait for the installation to finish.

Click "Close" when done.

Step 5: Verify Python Installation
Open Command Prompt

Type:

python --version
pip --version

Python PIP

- pip stands for "**Pip Installs Packages**".
- It is the default package manager for Python.
- Used to install, upgrade, and uninstall Python packages.

Command	Description
pip install <package_name>	Install a package (e.g., pip install numpy)
pip uninstall <package_name>	Remove a package
pip list	Show installed packages
pip freeze	List installed packages with versions
pip show <package_name>	Show details about a package
pip install -r requirements.txt	Install all packages listed in a file

Virtual Environment in Python

A **virtual environment** is an isolated space to install project-specific Python packages without changing the global Python setup.

Why Use Virtual Environments

- Prevent package conflicts across projects
- Use different versions of the same library
- Keep the global Python setup uncluttered
- Ensure consistent environments for teams and deployments

Creating and Using Virtual Environments

python -m venv env_name

.\venv\Scripts\activate

deactivate

PyCharm Installation

Step 1: Download Pycharm Installer

1. Go to the official Pycharm website: https://www.jetbrains.com/pycharm/download/?section=windows

2. Scroll to the page and find "**PyCharm Community Edition"** and Click Download button to download

Step 2: Run the Installer
Goto Downloads folder, Double-click the exe **"pycharm-community-2025.1.3.1"** and install it

Step 3: Configure Installation
1. Choose an installation location (default is fine).
2. Click **Next** → then **Install**.

Step 4: Finish Installation
1. Wait for the installation to finish.
2. Click **Finish** and select **"Run PyCharm Community Edition"**.

Step 5: Initial Setup
Click New Project to start coding.

Step 6: Configure Python Interpreter

When creating a new project, PyCharm will prompt to select a Python interpreter.

We can use:
Existing Python Or create a new virtual environment.

Visual Studio Code Installation

Step 1: Download VS Code Installer

1. Go to the official VS Code website:
https://code.visualstudio.com/
Click on "Download for Windows" (.exe file)

Step 2: Run the Installer
Goto Downloads folder, Double-click the exe
“VSCodeUserSetup-x64-*.exe” and install it

Step 3: Configure Installation
1. Choose an installation location (default is fine).
2. Click **Next** → then **Install**.

Step 4: Install Python Extension
To use Python in VS Code:

- Open Extensions (Ctrl+Shift+X)
- Search for Python
- Click Install (by Microsoft)

Python Data Types

Int: An int in Python is an immutable data type that represents whole numbers, positive or negative, of arbitrary precision.

Float: A float in Python is an immutable data type that represents real numbers with decimal points, supporting both positive and negative values.

Strings: Strings are used to store textual information. They are used to carry out operations that perform positional ordering among items.

Boolean: Boolean values are the two constant objects False and True. They are used to represent truth values (other values can also be considered false or true)

Lists: The list data type is the most generic data type. Lists can consist of a collection of mixed data types, stored by relative positions.

Tuples: Tuples are one among the immutable data types that can store values of mixed data types. Basically a list that cannot be changed.

Dictionaries: Dictionaries can store multiple objects, but unlike lists, in dictionaries, the objects are stored by keys and not by positions.

Sets: Sets are a data type that can be considered as an unordered collection of data without any duplicate items.

Built-In Type conversion functions

id() returns the identity of an object as an integer

type() returns the type of an object

str() converts any object into its string

bool() converts an argument to a Boolean value

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

chr() returns string representation of character given by integer argument

float() returns a floating-point object constructed from a number or string

int() returns an integer object constructed from a number or string

list(), tuple(), dict() and **set()** are the convert functions in collections

Python Control Flow

- ✓ The if...else statement is used if you want perform different action (run different code) on different condition.
- ✓ There can be zero or more elif parts, and the else part is optional.
- ✓ Most programming languages use {} to specify the block of code. Python uses indentation.
- ✓ A code block starts with indentation and ends with the first unintended line. The amount of indentation is up to you, but it must be consistent throughout that block.

Syntax of if...else

```
if test expression:
    Body of if
else:
    Body of else
```

```
if h > 50:
    print("Greater than 50")
elif h < 20:
    print("Less than 20")
else:
    print("Between 20 and 50")
```

Loops

- ✓ While loop

```
while expr:
    print("loop while expr is True")
```

```
while True:
    if expr:
        break
print("Go here on break")
```

- ✓ For loop

```
for i in <collection>
    <loop body>
```

```
for <var> in <iterable>:
    <statement(s)>
```

Functions in Python

- A function is a block of code written to carry out a specified task.
- Defined using the **def** keyword, with optional parameters and return values
- Parameters are specified after the function name inside the parentheses.
- We can add as many parameters as we want. Parameters must be separated with a comma.
- Promotes reusability – write once, call many times.
- Easy to test, debug, and maintain.

Syntax of Function

```
def function_name(parameters):
        """docstring"""
        statement(s)
```

```
def my_function():
  print("Hello from a function")
```

Three types of functions in Python:

- **Built-in function** :- Python predefined functions that are readily available for use like min() , max() , sum() , print() etc.
- **User-Defined Functions**:- Function that we define ourselves to perform a specific task.
- **Anonymous functions** : Function that is defined without a name. Anonymous functions are also called as **lambda functions**.

Lambda Functions

- ✓ A lambda function is a small anonymous function.
- ✓ A lambda function can take any number of arguments, but can only have one expression.
- ✓ Functions are defined using def keyword, anonymous functions are defined using the lambda keyword.
- ✓ Function has no name. It returns a function object which is assigned to the identifier and now call it as a normal function
- ✓ Lambda functions are used along with built-in functions like filter(), map() etc

Syntax of Lambda Function

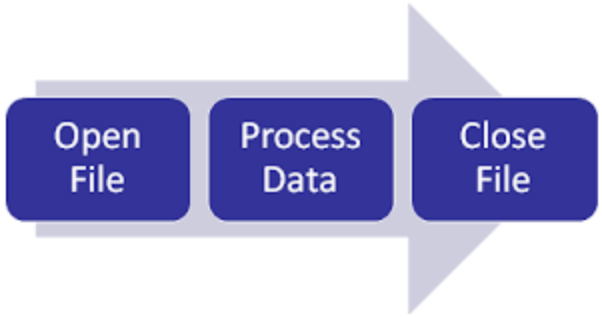
```
lambda arguments: expression
```

File Handling

- Python allows reading and writing to files using the file object
- The open function is used to get a file object and the syntax is:

  open (filename, opening mode)

- The mode can be read (r) , write (w), append (a), read and write (r+ or w+), read binary (rb), write binary (wb) etc.
- After the operation a file must be closed with close function



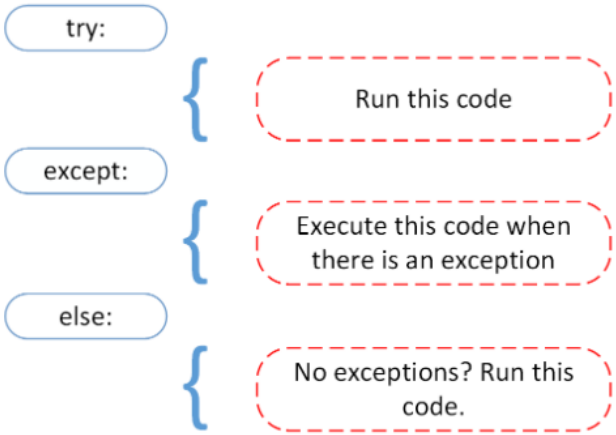
Character	Function
r	Open file for reading only. Starts reading from beginning of file. This default mode.
rb	Open a file for reading only in binary format. Starts reading from beginning of file.
r+	Open file for reading and writing. File pointer placed at beginning of the file.
w	Open file for writing only. File pointer placed at beginning of the file. Overwrites existing file and creates a new one if it does not exists.
wb	Same as **w** but opens in binary mode.
w+	Same as **w** but also alows to read from file.
wb+	Same as **wb** but also alows to read from file.
a	Open a file for appending. Starts writing at the end of file. Creates a new file if file does not exist.
ab	Same as **a** but in binary format. Creates a new file if file does not exist.
a+	Same a **a** but also open for reading.
ab+	Same a **ab** but also open for reading.

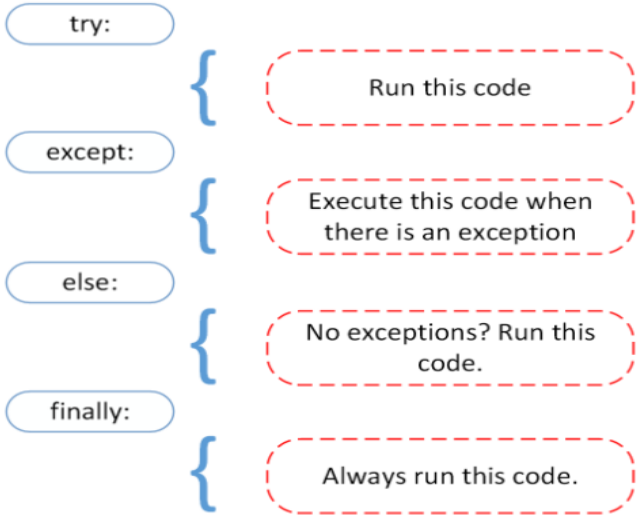
Modules & Packages

- ✓ Python file contains statements and definitions called Python Module.
- ✓ Python code file saved with the extension (.py) is treated as the module
- ✓ A module = a single .py file.
- ✓ A package = a folder that contains multiple modules and usually an __init__.py file.
- ✓ Helps keep code **organized**, **reusable**, and **maintainable**.

Exception Handling

- Python Exception Handling deals with errors during program execution.
- It prevents the program from crashing when an error occurs.
- Allows to catch and respond to errors.
- Makes code more robust and user-friendly.





Class and Objects

- A class is like a blueprint for creating objects (also called instances).
- It defines attributes (variables) and methods (functions) that describe the behavior and properties of something.
- Use **__init__()** to initialize object data.
- **self** refers to the current instance of the class.
- We can define methods inside a class to perform actions.
- Objects are created from classes to use the defined properties and methods.

Syntax

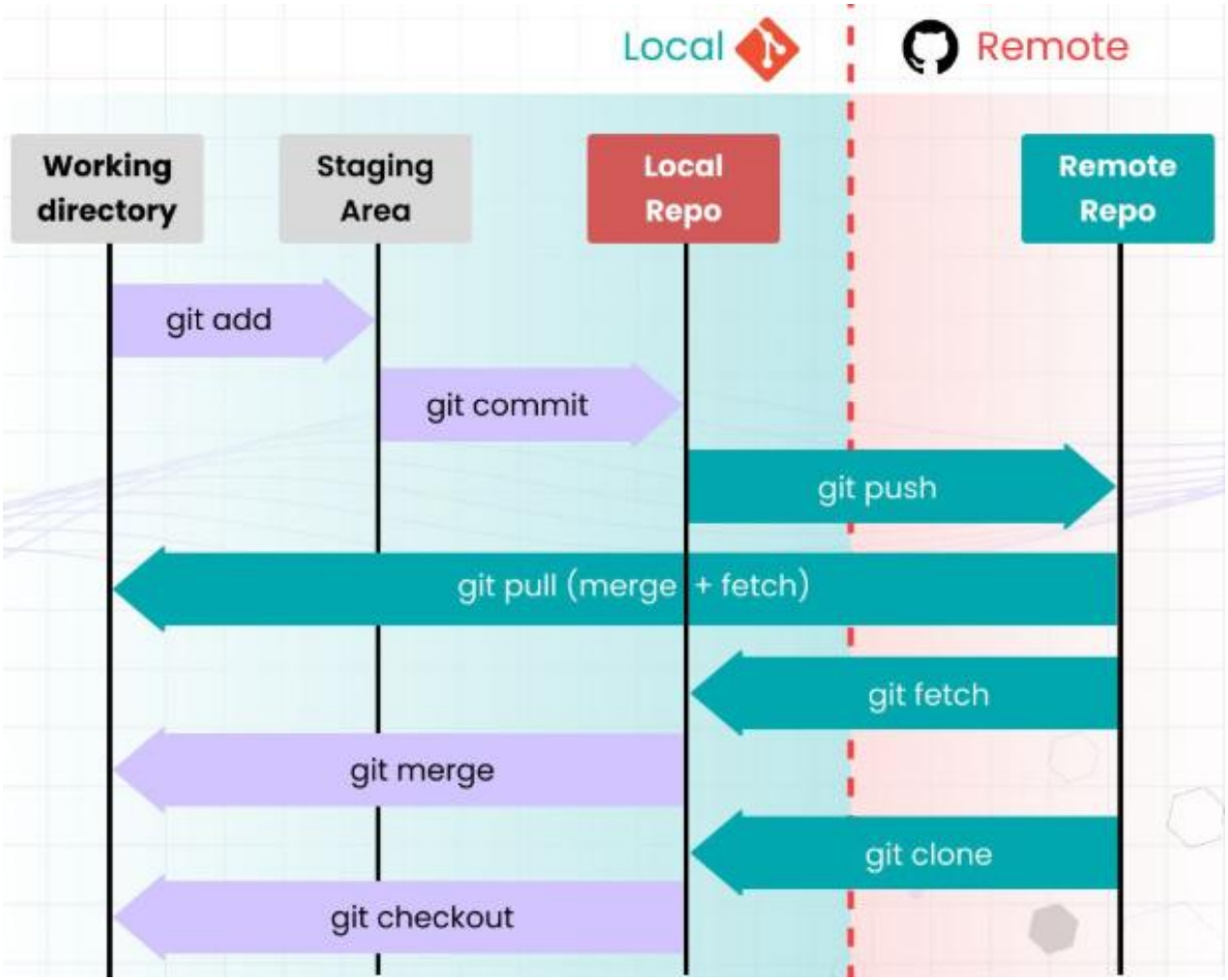
```
class ClassName:
    def __init__(self, param1, param2):
        self.param1 = param1
        self.param2 = param2

    def method_name(self):
        # action
```

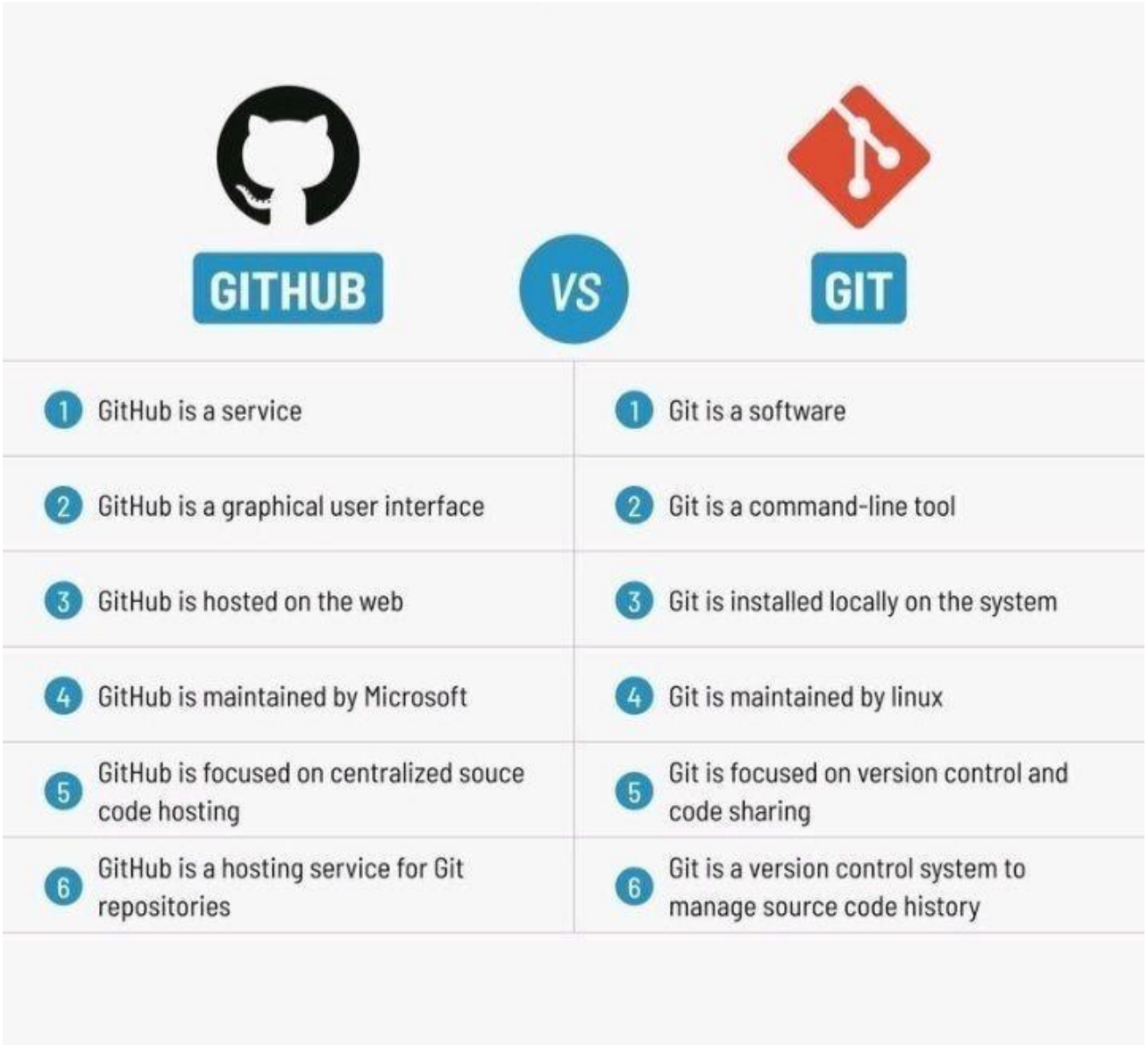
Others

- Date Functions
- Database Connections
- Logging
- Config Settings
- Github Source Code Integration

GIT


Local
Remote
Working directory
Staging Area
Local Repo
Remote Repo
git add
git commit
git push
git pull (merge + fetch)
git fetch
git merge
git clone
git checkout

GIT


GITHUB
VS
GIT
1 GitHub is a service
2 GitHub is a graphical user interface
3 GitHub is hosted on the web
4 GitHub is maintained by Microsoft
5 GitHub is focused on centralized souce code hosting
6 GitHub is a hosting service for Git repositories
1 Git is a software
2 Git is a command-line tool
3 Git is installed locally on the system
4 Git is maintained by linux
5 Git is focused on version control and code sharing
6 Git is a version control system to manage source code history

About Pandas

- Open-source Python library for data analysis and manipulation
- Built on top of **NumPy**
- Handles data from multiple sources: CSV, Excel, SQL, JSON, etc.
- Powerful tools for **data cleaning, transformation, and aggregation**
- Supports **label-based indexing** and **fast data operations**
- Widely used for **data analytics**
- Provides two main data structures:
  - **Series** – 1D labeled array
  - **DataFrame** – 2D labeled data table

pip install pandas

Pandas - Series

- **One-dimensional** labeled array in Pandas
- Can store data of any type: integers, floats, strings, etc.
- Has two parts:
  - Data – actual values
  - Index – labels for each value (default is 0, 1, 2, …)
- Similar to a column in an Excel sheet or a single column in a DataFrame
- Supports vectorized operations (fast arithmetic, filtering, etc.)
- Useful for time series data and single-column datasets

```
import pandas as pd
s = pd.Series([10, 20, 30], index=['a', 'b', 'c'])
print(s)
```

Pandas - Dataframe

- 2D data structure with rows and columns, like an Excel sheet or SQL table
- Each column can hold different data types (int, float, string, etc.).
- Supports data cleaning, filtering, aggregation, and visualization.
- Easy to import/export from CSV, Excel, SQL, JSON, etc.
- Fast row and column indexing using loc[], iloc[].
- Built-in methods for statistics & analysis (mean(), sum(), describe()).
- Handles missing data and duplicates efficiently.

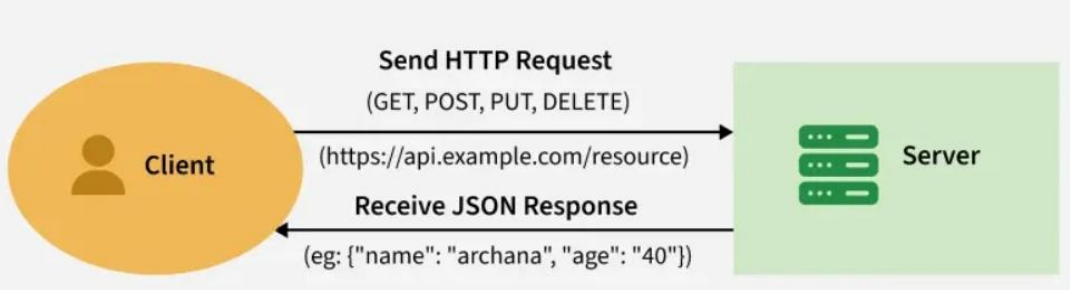
```
import pandas as pd
data = {'Name': ['Varun', 'Karthik'], 'Age': [25, 30]}
s = pd.DataFrame(data)
print(s)
```

FAST API

- **FastAPI** - Modern, fast (high-performance) web framework for building APIs with Python.
- **High performance** – among the fastest Python frameworks.
- Ideal for building **RESTful APIs**, microservices, and backend services.
- **Large community support** and widely used in production.
- In-built of Swagger(documentation and testing) and Pydantic(data validation)

CRUD Action	HTTP Verb	FastAPI Decorator	Example Path
Create	POST	@app.post()	/users
Read (all / one)	GET	@app.get()	/users, /users/{id}
Update (full)	PUT	@app.put()	/users/{id}
Update (partial)	PATCH	@app.patch()	/users/{id}
Delete	DELETE	@app.delete()	/users/{id}

API Architecture & Response Code



Code	Meaning
200	OK – Success
201	Created – Resource successfully created
204	No Content – Success, nothing to return
400	Bad Request – Invalid input/request
401	Unauthorized – Authentication required
403	Forbidden – No permission
404	Not Found – Resource doesn’t exist
405	Method Not Allowed – Wrong HTTP method
409	Conflict – Duplicate or conflicting request
422	Unprocessable Entity – Validation error (very common in FastAPI)
500	Internal Server Error – General server failure

Final Project – Expense Manager

Purpose:

Streamline financial tracking with three core operations: **Add, View, Delete Expenses**

Add Expense

- Record new transactions with description, amount, date, category.

View Expense

- Retrieve and review all expenses in a structured format.

Delete Expense

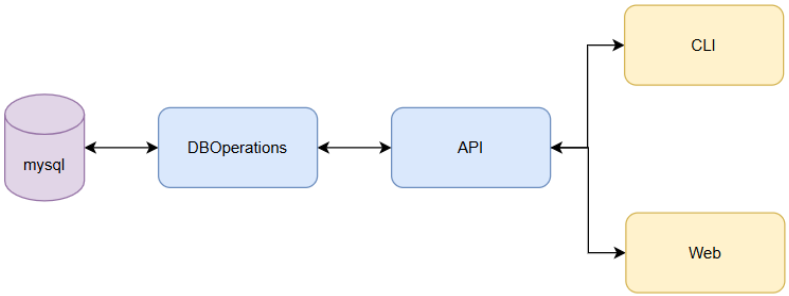
- Remove erroneous or duplicate entries.

Project Documentation

The Expense Manager is a **full-stack Python application** designed to manage personal or business expenses.

It provides:

- **Web UI (HTML/JavaScript)** for adding, viewing, and deleting expenses.
- **FastAPI-based backend (app.py)** exposing REST APIs.
- **MySQL database** for storing expenses and categories.
- **Centralized logging** (`expense_logger.txt`).
- **Robust exception handling** across UI, API, and DB layers.
- **Config-based architecture** using `.env`.




```
expense-tracker/
│
├── app/                        # Backend application (FastAPI + DB logic)
│   ├── __init__.py             # (optional) Makes this folder a Python package
│   ├── main.py                 # Entry point (FastAPI runner)
│   ├── app.py                  # FastAPI routes/endpoints
│   ├── db.py                   # Database connection & CRUD logic
│   └── utils.py                # Logging, base URL, helper functions
│
├── ui/                         # Frontend (Static files)
│   └── index.html              # Main web UI
│
├── .env                        # Environment variables (DB credentials, API keys, etc.)
├── .ignore                     # Files/folders to ignore (like .gitignore)
├── requirements.txt            # Python dependencies
├── expense_logger.txt          # Log file
└── README.md                   # (recommended) Project documentation
```

Thank You