**GM ACADEMY**

**PYTHON PROGRAMMING**

**Course Introduction:** Python for Machine Learning, Data Science, and Web Development

**Course Overview**

This course is designed for beginners who want to master Python with a focus on Machine Learning (ML), Data Science, and Web Development. It provides a structured learning path, starting from basic Python programming to advanced topics like data analysis, model building, and web application development.

**By the end of this course, you will be able to:  
1.** Write efficient Python code for real-world applications.  
2. Analyze and visualize data using Python libraries.  
3. Build machine learning models using Scikit-Learn.  
4. Develop web applications using Flask/Django.  
5. Work with databases and APIs for full-stack development.

**Course Objectives**

By completing this course, you will achieve the following objectives:

1. **Python Programming Fundamentals**
   * + Master Python syntax, data types, functions, and object-oriented programming (OOP).
     + Learn error handling, file operations, and working with libraries.
2. **Data Science & Machine Learning**
   * + Understand data manipulation using NumPy and Pandas.
     + Perform data visualization with Matplotlib and Seaborn.
     + Implement machine learning models using Scikit-Learn.
3. **Web Development with Python**
   * + Develop backend applications using Flask and Django.
     + Work with databases like SQLite and PostgreSQL.
     + Learn API development and deployment strategies.
4. **Project-Based Learning**
   * + Work on real-world projects in ML, Data Science, and Web Development.
     + Build portfolio-worthy applications to showcase your skills.

**Prerequisites**

✔ No prior programming experience required!  
✔ Basic familiarity with computers and mathematics will be helpful.

**Nice to hear? Will this not be easy? I believe so.**

**Course Structure & Duration**

The course is stipulated to take about 5 months. However, much is dependent on how we prepare ourselves well. It can always take less. Here are the sections that we ought to cover, Divided into 5 key sections:

1. Python Fundamentals (1 weeks)
2. Object-Oriented Programming (OOP) (1 week)
3. Data Science & Machine Learning (2-4 weeks)
4. Web Development (3–6 weeks)
5. We spend the rest of the time in doing Advanced Topics & Projects

**Learning Resources**

* Python Docs, Kaggle Courses, Real Python, Flask/Django Docs
* Books: *Python Crash Course* (Eric Matthes), *Hands-On ML* (Aurélien Géron)

**Summary**

Let us get the unto recap of our intended outcome:

At the end of the course, you will have:

1. A solid foundation in Python programming.  
2. Practical experience in data science & machine learning.  
3. The ability to build and deploy web applications.  
4. A portfolio of projects to showcase your expertise.

While this is the intention of this course, it is important to take note that we must think like entrepreneurs, because at the end of it, we will figure out how we can use what we have learned to make some good money. Is it not interesting?

**WHY PYTHON?**

Python is the best choice for Machine Learning, Data Science, and Web Development due to its simplicity, versatility, and strong ecosystem. Here’s why it stands out:

**1️. Easy to Learn & Readable**

Python has a simple, English-like syntax, making it beginner-friendly as we shall see.  
Code readability reduces development time and makes debugging easier.

**2️. Strong Support for Machine Learning & Data Science**

**Rich Libraries:** Libraries like NumPy, Pandas, Matplotlib, Scikit-Learn, TensorFlow, and Py-Torch make ML and data analysis efficient.  
**Huge Community:** Thousands of developers contribute to open-source ML tools in Python.

**3️. Web Development Capabilities**

Frameworks like Flask and Django make it easy to develop powerful web applications.

Supports APIs, database integration, and security for full-stack development**.**

**4️. Versatility & Cross-Platform**

Python runs on Windows, Mac, Linux, and even mobile devices.  
Can be used for automation, AI, cybersecurity, finance, and more.

**5️. Community & Career Opportunities**

Python has massive online resources (Kaggle, GitHub, Stack Overflow).  
High demand in job markets for AI, ML, Data Science, and Web Development.  
Many Fortune 500 companies (Google, Facebook, Netflix, Tesla) use Python.

I hope you are becoming interested with python? Of course you should. I believe it is your dream to work in one of those big tech companies, or to come up with something that will be a form of solution to problems in the society.

Let’s go!

**PYTHON PROGRAMMING FUNDAMENTALS**

**2. Installing Python**

Python can be installed in several ways depending on your needs.

**Option 1: Installing Python from the Official Website**

1. Download the latest version from [python.org](https://www.python.org/downloads/).
2. Install by checking **"Add Python to PATH"** before proceeding.

**Option 2: Using Anaconda (Best for Data Science & ML)**

1. Download **Anaconda** from [anaconda.com](https://www.anaconda.com/).
2. Install and open **Anaconda Navigator** or use **Conda environments**.
3. Launch **Jupyter Notebook**, **Spyder**, or **VS Code** from Anaconda Navigator.

**Option 3: Using Miniconda (Lightweight Alternative to Anaconda)**

Download from miniconda.

**Why Python is Important for Importing Files**

Python is one of the most versatile and widely used programming languages, and its ability to import files is a critical feature that makes it indispensable for various tasks. Importing files in Python allows you to work with external data, libraries, and modules, enabling you to build powerful and scalable applications.

Here is an explanation why importing files is important in Python and how it is used in real-world scenarios.

**Accessing External Data**

Python is widely used for data analysis, machine learning, and automation, all of which require working with external data stored in files.

Importing files allows Python programs to:

- Read data from CSV files, Excel spreadsheets, JSON files, databases, and more.

- Process and analyze large datasets efficiently.

- Store and retrieve configuration settings or user inputs from files.

*Table 1: python libraries imported*

|  |  |  |  |
| --- | --- | --- | --- |
| *Library* |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**4. Choosing the Right IDE or Code Editor**

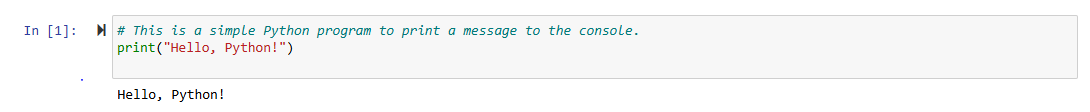
|  |  |  |  |
| --- | --- | --- | --- |
| **IDE** | **Best for** | **Advantages** | **Disadvantages** |
| **VS Code** | Web Dev, ML, Data Science | Lightweight, Customizable | Needs Extensions |
| **Visual Studio** | Enterprise apps | Powerful, Integrated tools | Heavy |
| **PyCharm** | Django, Flask, Data Science | Smart features, great for web dev | Heavy on RAM |
| **Jupyter Notebook** | ML, Data Science | Interactive, supports visualization | Not for web dev |

**TASK FOR TO DAY**

1. Write a Python program to print "Hello, Python!" to the console.

This is the code as written in Jupiter notebook.

Open your Anaconda and launch the Jupiter notebook. The notebook will open, please type the code as indicated.



**Explanation:**

In Python, a # symbol is used to write comments.

Comments are ignored by Python but help **explain the code** to developers.

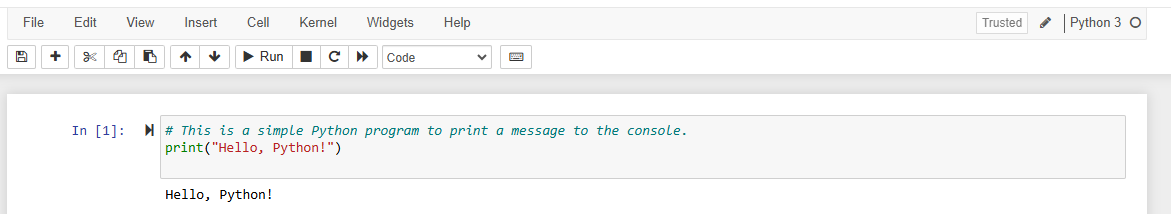
**print("Hello, Python!")**

The print () function is a **built-in Python function** that outputs text to the console.

The text inside **quotation marks ("Hello, Python!")** is called a **string**.

**Running the code**

Just click **Run**.



1. Create a program that declares and prints the types of the following variables:

name = "John"

age = 25

height = 5.9

is\_student = True



**name = "John"**

Stores "John" as a **string** (text).

**age = 25**

Stores 25 as an **integer** (whole number).

**height = 5.9**

Stores 5.9 as a **float** (decimal number).

**is\_student = True**

Stores True as a **boolean** (True/False).

**print(f"Name: {name} - Type: {type(name)}")**

Prints name and its **type** (str → string).

The f"" helps insert values inside {} easily.

**Other print() lines work the same way**

1. Write a program that asks the user for their name and age, then prints a greeting message like:  
   *"Hello, [YOUR NAME]! You are [your age] years old."*
2. Write a Python program that takes two numbers as input from the user and prints their sum, difference, product, and quotient.
3. Write a program that asks the user for a number and prints whether it is even or odd.
4. Write a Python program to print numbers from 1 to 10 using a for loop.
5. Write a Python program that prints numbers from 10 down to 1 using a while loop.
6. Create a list of five favorite colors and print the third color in the list.
7. Write a function called greet that takes your name as an argument and prints *"Hello, [name]!"*.
8. Write a program that asks the user for a number and tries to divide 100 by that number. Handle the case where the user enters zero using a try-except block.

**DEVELOP WEB APPLICATIONS USING FLASK/DJANGO.**

Web development involves creating websites and web applications that run in a browser. It includes: **Frontend** – What users see (HTML, CSS, JavaScript), **Backend** – How the website works behind the scenes (Python, Databases, Servers).

Python is mostly used for the backend part of web development, but it can also be used for **some frontend tasks** and full-stack development (*to be discussed later*). For now, let us discuss some backend tasks in which python is applied:

1. **Server-side logic**-Processing requests and responses.
2. **Database management**- Storing and retrieving data.
3. **User authentication**- Login, signup, security.
4. **API development**- Creating RESTful APIs.

**SERVER-SIDE LOGIC**

Server-side logic involves processing client requests and generating responses on a server, handling tasks like data retrieval, manipulation, and storage, and is crucial for dynamic and interactive web applications.

Here's a more detailed explanation of server-side logic and its role in web development:

**Key Concepts:**

**Client-Server Model**

Web applications typically operate using a client-server model where a client (e.g., a web browser) sends requests to a server, and the server processes those requests and sends back responses.

**Server-Side Processing**

Server-side processing refers to the execution of code and operations on the server, rather than on the client's device.

**Request-Response Lifecycle**

The process of a client sending a request to the server, the server processing it, and then sending a response back to the client is known as the request-response lifecycle.

**Server-Side Languages**

Popular server-side programming languages include PHP, Python, Ruby, Java, and Node.js.

**Web Servers**

Web servers, such as Apache or Nginx, are responsible for receiving client requests, processing them, and sending responses back to the client.

**TASKS PERFORMED BY SERVER-SIDE LOGIC**

1. **Data Retrieval and Manipulation**

Server-side logic can access and manipulate data from databases or other data sources.

**Business Logic**

Server-side code can implement complex business logic, such as validating user input, processing payments, or updating inventory levels.

**Session Management**

Server-side logic can manage user sessions, ensuring that users are authenticated and that their data is securely stored.

**Dynamic Content Generation**

Server-side logic can generate dynamic content, such as HTML pages, based on user requests and data.

**Database Interaction**

Server-side code interacts with databases to store, retrieve, and manage data.

**Security**

Server-side logic can implement security measures to protect data and prevent unauthorized access.

**BENEFITS OF SERVER-SIDE PROCESSING**

**Enhanced Security**

Processing data on the server reduces security risks, as sensitive information is not exposed to the client.

**Improved Performance**

Server-side processing can offload complex tasks from the client, leading to faster response times and a smoother user experience.

**Dynamic Content**

Server-side logic enables the creation of dynamic and interactive web applications.

**Scalability**

Server-side applications can be designed to scale to handle large numbers of users and requests.

**Maintainability**

Server-side logic can be more easily maintained and updated than client-side code.

**WHY PYTHON FOR WEB DEVELOPMENT?**

**Readability and Simplicity**

Python's syntax is known for being easy to read and understand, which makes it a good choice for both beginners and experienced developers.

**Large and Active Community**

Python has a large and active community, meaning there's plenty of support and resources available for developers.

**Extensive Libraries and Frameworks**

Python offers a wide array of libraries and frameworks, such as Django and Flask, that simplify web development tasks.

**Scalability and Performance**

Python is a scalable language, meaning it can handle large and complex web applications.

**Versatility**

Python can be used for various web development tasks, including building web applications, APIs, and web services.

**Backend Development**

Python is primarily used for backend development, handling server-side logic, database interactions, and API development.

**Frameworks**

**Django:** A high-level Python web framework that promotes rapid development and clean, pragmatic designs.

**Flask:** A lightweight microframework that provides flexibility and scalability for web applications.

**FastAPI:** A modern, fast (high-performance) web framework for building APIs with Python 3.6+.

More of this will be discussed later.

**DATABASE MANAGEMENT**

A **database** is a structured collection of data that can be **stored, retrieved, and managed** efficiently. It consists of **tables, rows, and columns**, just like an Excel sheet all of which are expressed as shown below:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Types of Databases**  
**Relational Databases (SQL-based)**- Store data in structured tables (**MySQL, PostgreSQL, SQLite**).  
 **NoSQL Databases**- Store flexible and unstructured data (**MongoDB, Firebase**).

The database management system is a collection of programs that help in storing and retrieving data from the database. DBMS consists of programs that are helpful in manipulating databases.

**TYPES OF DBMS**

Based on the data model, there are four types of DBMS**.**

* Relational Database (RDBMS)
* Object Oriented Database (OODBMS)
* Hierarchical Database
* Network Database

1. **Relational Database**

The relational database management system is used to store data in two-dimensional tables using rows and columns. RDBMS is used in industries, and it is a very popular data model. In a database, every table has a key field which identifies each record uniquely. This key field is known as a primary key. We can make a whole column as a primary key which can be helpful in identifying or referencing any record easily in a table. A few examples of RDBMS are oracle database, MySQL, and Microsoft SQL server.

Python supports **relational databases** such as **SQLite, MySQL, and PostgreSQL** for web applications, data analysis, and backend systems.

1. **Object Oriented Database (OODBMS)**

An **Object-Oriented Database Management System (OODBMS)** is a type of database that **stores data as objects**, similar to how objects are used in **Object-Oriented Programming (OOP)**. Instead of traditional **tables and rows (Relational Database - RDBMS)**, OODBMS stores **objects, their attributes, and methods** directly into the database.

Best for applications dealing with complex data like Graphs, Multimedia, and Hierarchical data.

In Summary:

* + - **OODBMS stores data as objects** instead of tables
    - **Supports inheritance, polymorphism, and encapsulation** (like OOP).
    - **Easier mapping between objects in code and objects in the database**.
    - **Used in AI, CAD, and real-time systems** where complex relationships exist.

**FRONTEND DEVELOPMENT**

**Front-end development** is the process of creating the visual and interactive part of a website or web application. It involves using **HTML, CSS, and JavaScript** to design pages that users can see and interact with.

Technologies Used in Front-End Development

|  |  |
| --- | --- |
| **Technology** | **purpose** |
| HTML (HyperText Markup Language) | Defines the structure of a webpage. |
| CSS (Cascading Style Sheets) | Styles and formats the appearance of the webpage. |
| JavaScript (JS) | Adds interactivity and dynamic content |
| Bootstrap/Tailwind CSS | Frameworks for responsive design and faster styling. |
| React/Vue/Angular | JavaScript frameworks for building dynamic UIs. |

For frontend, use JavaScript, HTML, and CSS (but Python can generate dynamic HTML). The better choice depends on your project requirements and use case.

Below is a comparison to help you decide.

**JavaScript + HTML + CSS (Traditional Frontend)**

* + - Best for Modern, dynamic, and interactive websites.
    - Used when You want a fast, user-friendly UI with animations and interactivity.

**How It Works**

1. **HTML** → Defines the page structure.
2. **CSS** → Styles the page for better appearance.
3. **JavaScript** → Adds interactivity (e.g., buttons, forms, animations).
   * + Fast and responsive since JavaScript runs in the browser.
     + Works with modern frameworks like React, Vue, and Angular.
     + Easy to integrate APIs for fetching data from a Python backend.

**Python-Generated HTML (Backend-Rendered UI)**

Best for Websites that generate dynamic content on the server (e.g., dashboards, reports, blogs).

A dynamic server refers to a backend system that generates web content on the fly, meaning the webpage changes based on user input, database data, or real-time updates. This is different from a static server, which serves fixed content that doesn’t change unless manually updated.

|  |  |  |
| --- | --- | --- |
| Feature | Static Server | Dynamic Server |
| content | Fixed (HTML, CSS only) | Changes based on user input or database |
| Example | Simple HTML website | E-commerce site, dashboard, social media |
| How It Works | Serves pre-made files | Generates pages in real time |
| Best For | Blogs, portfolios | Online stores, dashboards, forums |

A basic HTML page (index.html) that never changes unless updated manually is a good example of a static page. The content is the same for every visitor.

An example of a Dynamic Page is A social media feed that updates automatically with new posts and A user’s profile page that displays personalized data.

A dynamic server processes user requests and generates content dynamically using:  
 - Backend programming languages (**Python, PHP, Node.js, etc**.).  
 - Databases (MySQL, PostgreSQL, MongoDB) to fetch/store data.

**-APIs** (REST or GraphQL) to send/receive data from different services.

Below is an example of how the Dynamic Server Processes a User Request

1. User requests a page (e.g., /profile in a web app).
2. Server fetches data (e.g., user info from the database).
3. Server generates a dynamic HTML page with the user’s details.
4. Browser displays personalized content based on the fetched data.

**Dynamic Servers in Real-World Applications**

**E-commerce** → A product page that changes based on the selected item.  
**Social Media** → A feed that updates with new posts.  
**Dashboards** → Real-time analytics, stock prices, or user activity.

We can create a dynamic web page that shows different content based on a database using python (flask) using the following major steps. We shall discuss all of them as we establish our first project.

* 1. Install Flask
  2. Create app.py (Python Backend)
  3. Create index.html (Dynamic HTML)

But before then, let us rewind what we really needs.

Before we **start building a dynamic web page using Flask**, we need the right tools and setup. Let’s **explore in detail** what is required for a successful Flask web development project.

**Basic Requirements for Flask Web Development**

To develop a dynamic web application, we need:  
1. A Programming Language (Python).  
2. A Web Framework (Flask).  
3. A Code Editor to write and edit the code.  
4. A Database to store and manage data.  
5. A Web Browser to view the website.  
6. A Virtual Environment to keep project dependencies organized.

Flask is a Python-based web framework, so the first requirement is Python. Anaconda can be used for Flask web development, but VS Code is often a better choice. Anaconda is a Python distribution that includes many pre-installed packages, including data science libraries (NumPy, Pandas, Jupyter Notebook).

Pros of Using Anaconda for Flask

1. Comes with a built-in virtual environment manager (conda).
2. Ideal if you’re working with Flask + Data Science (Jupyter Notebooks, AI, ML)
3. Includes Spyder & Jupyter Notebook for testing Python scripts.

**Cons of Using Anaconda for Flask**

Not lightweight – Includes many unnecessary libraries for web development.  
Setting up Flask in Anaconda requires extra steps.  
Not optimized for full-stack development (better for AI, ML, and Data Science).

If you want to install flask in Anaconda, here is the procedure.

conda create --name flask\_env python=3.10

conda activate flask\_env

pip install flask

create **app.py** and run:

python app.py

**Best if:** You plan to combine **Flask with Data Science/AI applications**.

**VS Code for Flask Development**

**VS Code (Visual Studio Code)** is a lightweight and **fully-featured** code editor, perfect for Flask development.

**Pros of Using VS Code for Flask**

Fast & lightweight – Uses fewer system resources than Anaconda.  
 Full control over project structure.  
Supports multiple languages (Python, JavaScript, HTML, CSS).  
Has built-in Git support & terminal.  
Easy to install Flask & dependencies.

**🔹 Setting Up Flask in VS Code**

1. Install **VS Code**
2. Install **Python extension** from **VS Code Marketplace**.
3. Install Flask in a virtual environment:

python -m venv myenv

myenv\Scripts\activate # Windows

pip install flask

Create **app.py** and run:

python app.py

**Best if:** You want **a lightweight, efficient development environment**.

**Comparison: Anaconda vs. VS Code for Flask**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Anaconda** | **VS Code** |
| Ease of Use | More setup needed | Simple & fast |
| Performance | Heavy, slow | Light weight |
| Best For | AI, ML, Data Science | Full-stack & web development |
| Pre-installed Packages | Yes (NumPy, Pandas, etc.) | No (but easy to install) |
| Integration with Flask | Extra steps required | Works smoothly |

In summary:

**Use Anaconda if:** You are building a Flask app that integrates with data science tools (Jupyter, AI, ML, Pandas, NumPy, etc.).  
**Use VS Code if:** You are focusing on web development (Flask + HTML + CSS + JavaScript + Databases).

PROJECTS

1. DOUBLE M COLLECTION

We are to make a website for this website that sell clothing. What are the key elements to put unto consideration?