Gowtham SB
www.linkedin.com/in/sbgowtham/ Instagram - @dataengineeringtamil

Introduction to Programming & Python

📌 What is Programming? (Concepts & Importance)

◇ 1. Definition of Programming

Programming is the process of **giving instructions** to a computer to perform a specific task. It involves **writing, testing, and debugging** code in a programming language like Python, Java, or C++.

◇ 2. Why is Programming Important?

✅ **Automation** – Reduces manual work by automating repetitive tasks
✅ **Data Processing** – Used in data analysis, AI, and machine learning
✅ **Software Development** – Builds applications (web, mobile, desktop)
✅ **Problem-Solving** – Helps in logical thinking and structuring solutions
✅ **IoT & Robotics** – Used in smart devices and robotics development

◇ 3. How Does a Computer Understand Programs?

Computers only understand **binary (0s and 1s)**. Programming languages act as a **bridge** between **humans and machines**.

- **High-level languages** (Python, Java, C++) → Easy for humans to write
- **Low-level languages** (Assembly, Machine Code) → Close to hardware

🆚 Difference Between Coding & Programming

Feature	Coding 🖥️	Programming 💻
Definition	Writing simple instructions in a programming language	Full process of designing, writing, testing, and debugging software
Complexity	Simple (writing code snippets)	More complex (involves algorithms, logic, and system design)
Tools Required	Just a code editor (VS Code, Notepad++)	Requires IDEs, debuggers, version control (Git)
Scope	Writing syntax for specific tasks	Involves problem-solving, architecture, and software development
Example	Writing a Python function to add two numbers	Developing a full calculator app with UI, error handling, and optimization

📝 Key Takeaways

✓ Coding is a part of Programming
✓ Programming includes problem-solving, designing, and debugging
✓ A good programmer knows both coding & software development concepts

Types of Programming Paradigms

Programming languages follow different paradigms (styles of programming) based on how they solve problems. Here are the major types:

1️⃣ Procedural Programming

- 🔹 Definition: Uses step-by-step instructions (procedures) to solve problems.
- 🔹 Example Languages: C, Pascal, Python (supports procedural style)
- 🔹 Key Features:

✅ Follows a top-down approach
✅ Uses functions (procedures) to divide code
✅ Easy to read and understand

🔹 **Where is it Used?**

- Small programs
- Simple automation scripts

2️⃣ Object-Oriented Programming (OOP)

- 🔹 Definition: Organizes code into objects (real-world entities) with properties and methods.
- 🔹 Example Languages: Python, Java, C++, C#
- 🔹 Key Features:

✅ Uses classes and objects
✅ Supports Encapsulation, Inheritance, Polymorphism, Abstraction
✅ Best for scalable and reusable code

🔹 **Example in Python:**

🔹 **Where is it Used?**

- Large applications
- Web development
- Game development

3️⃣ Functional Programming

- 🔹 **Definition:** Treats computation as the evaluation of mathematical functions and avoids changing state.

- 🔹 **Example Languages:** Python, Haskell, Lisp, Scala
- 🔹 **Key Features:**

✅ Uses **pure functions** (no side effects)
✅ Supports **higher-order functions** (`map()`, `filter()`, `reduce()`)
✅ Good for **parallel computing**

- 🔹 **Where is it Used?**
  - Data processing (Pandas, PySpark)
  - AI and ML applications

Functional Programming (FP) vs Object-Oriented Programming (OOP) – Technical Differences

1️⃣ State Management

- **OOP** → Uses objects that store data (stateful). Example: A `BankAccount` class with a balance that updates.
- **FP** → No stored state, only pure functions. Example: A function that calculates interest without modifying the balance.

2️⃣ Mutability vs Immutability

- **OOP** → Objects are mutable (can be modified).
- **FP** → Data is immutable; changes return new data instead of modifying existing ones.

3️⃣ Concurrency & Parallelism

- **OOP** → Shared states can lead to race conditions (requires locks & synchronization).
- **FP** → No shared state, making it **better for parallel processing** (e.g., Spark, Dask).

4️⃣ Code Structure

- **OOP** → Organizes code using **classes and objects** with methods.
- **FP** → Uses **pure functions** that take input and return output without side effects.

5️⃣ Reusability & Testing

- **OOP** → Testing is harder due to dependencies between objects.
- **FP** → Easier to test as functions are independent and don't depend on object state.

When to Use What?

✅ **Use OOP** for applications with long-lived objects (e.g., web apps, games).
✅ **Use FP** for **data transformations, parallel computing, and big data processing** (e.g., Pandas, PySpark).

How is FP Different from OOP?

Feature	Functional Programming (FP)	Object-Oriented Programming (OOP)
State	No state (stateless)	Uses objects with state
Data	Immutable (unchangeable)	Mutable (changeable)
Functions	Pure functions, no side effects	Methods modify object state
Testing	Easy to test (no dependencies)	Harder due to object dependencies
Concurrency	Easy (no shared state)	Harder (requires locking)
Example Use	Data processing (Pandas, Spark)	Web apps, games, UI development

Conclusion

✅ **Use FP for data transformations, analytics, machine learning, and parallel computing.**
✅ **Use OOP for building applications, user interfaces, and complex systems.**

✅ **Here are some popular languages, platforms, and tools developed using Python:**

◇ 1. YouTube (Initial Prototype)

- The early version of YouTube was developed using Python.
- Even today, Python is used in parts of YouTube's backend for automation and data.

◇ 2. Instagram

- Instagram's backend is heavily powered by Python + Django.
- It handles billions of users and photos using Python.

◇ 3. Dropbox

- Dropbox's desktop client and server-side code are written in Python.

- Even the founder, Guido van Rossum (creator of Python), worked at Dropbox.

◇ 4. Reddit

- Originally written in Lisp, then moved to Python for scalability and ease of development.

◇ 5. BitTorrent

- The peer-to-peer file sharing protocol BitTorrent was initially created using Python.

◇ 6. OpenStack

- A popular cloud computing platform used by many enterprises — mostly written in Python.

◇ 7. Blender (Scripting Language)

- The 3D modeling and animation software Blender uses Python for scripting and automation.

◇ 8. Ansible

- A powerful infrastructure automation tool, written in Python.

◇ **9. TensorFlow (Some parts)**

- While TensorFlow is mainly in C++, it has a Python API, and many training scripts are written in Python.

◇ **10. Pip, Flask, Django, Pandas, NumPy, etc.**

- All these Python frameworks and libraries are, of course, developed in Python.

✅ Languages Developed in Python (Extended List)

1) Hy

- What Is It? A Lisp dialect that compiles to Python's Abstract Syntax Tree (AST).
- Key Point: You can mix Lisp code with Python libraries, making it easy to use Lisp features (macros, s-expressions) alongside the entire Python ecosystem.

---

2) Coconut

- What Is It? A functional programming superset of Python.
- Key Point: Coconut adds pattern matching, lazy lists, and other functional features, then compiles down to standard Python code.
- Usage: Great if you want Haskell-like features in a Python-based project.

---

3) MyHDL

- What Is It? A hardware description language (HDL) built in Python.
- Key Point: Instead of using VHDL or Verilog, you can describe hardware logic in Python (MyHDL). It translates your Python code into Verilog or VHDL for actual circuit synthesis.

Why Are They Built in Python?

1. Easy to Parse & Transform: Python has powerful libraries for parsing, AST manipulation, and code generation.
2. Large Ecosystem: They can reuse the entire Python ecosystem (e.g., packaging, tooling, libraries).
3. Rapid Prototyping: Python's readability and flexibility make it ideal for building new languages or DSLs quickly.

✅ Popular Frameworks Developed in Python

🕸 Web Development

Framework	Description
Django	Full-stack web framework – batteries included (ORM, Auth, Admin panel)
Flask	Lightweight micro-framework – ideal for APIs and small services
FastAPI	Modern async framework – super fast and great for building APIs
Tornado	Scalable, non-blocking web server & framework
Bottle	Simple micro-framework for building small web apps and APIs
Pyramid	Flexible and scalable – sits between Flask and Django in complexity

📊 Data Science / Machine Learning

Framework	Description
TensorFlow (Python API)	While core is in C++, the Python API is the primary interface
PyTorch	Built by Facebook, completely in Python + C++ backend
Scikit-learn	ML framework for classification, regression, clustering, etc.
Statsmodels	Statistical modeling framework in Python
Pandas	For data manipulation and analysis
Dask	Parallel computing with DataFrame and array APIs similar to Pandas/NumPy

Gowtham SB
www.linkedin.com/in/sbgowtham/ Instagram - @dataengineeringtamil

📦 Web Scraping & Automation

Framework	Description
Scrapy	Powerful web crawling and scraping framework
Selenium (Python binding)	Automate browser actions for testing/scraping
BeautifulSoup	Parsing HTML/XML – often used with requests or Scrapy

💻 GUI Development

Framework	Description
Tkinter	Standard GUI library bundled with Python
PyQt / PySide	Python bindings for Qt (rich GUI apps)
Kivy	For multi-touch apps, mobile-friendly
wxPython	Native-looking desktop apps across platforms

☁️ Cloud, DevOps, Automation

Framework	Description
Ansible	IT automation and configuration management (written in Python)
SaltStack	Infrastructure automation tool
Fabric	SSH command execution and automation
Invoke	Pythonic task execution tool like Makefiles
AWS Boto3	AWS SDK for Python (for interacting with AWS services)

🧪 Testing

Framework	Description
Pytest	Most popular testing framework – clean, readable, powerful
Unittest	Built-in unit testing module
Nose2	Successor to the older Nose testing framework
Behave	BDD testing like Cucumber for Python

🛠️ Other Interesting Frameworks

Framework	Description
OpenCV (Python bindings)	Computer vision (written in C++, but Python used extensively)
Home Assistant	Smart home automation platform (written in Python)
Sphinx	Documentation generator (used by Python itself)
Airflow	Workflow orchestration (data pipelines)
Celery	Distributed task queue for background jobs

Python has frameworks in:

- ✅ Web Dev (Django, Flask, FastAPI)
- ✅ Data Science (PyTorch, Scikit-learn)
- ✅ Automation (Ansible, Airflow)
- ✅ Desktop GUI (Tkinter, PyQt)
- ✅ Testing (Pytest, Unittest)
- ✅ Scraping (Scrapy, BeautifulSoup)

📝 Python Software Foundation License (PSFL)

◇ What is PSFL?

- The Python Software Foundation License is a free and open-source license, approved by both:
    - OSI (Open Source Initiative)
    - FSF (Free Software Foundation)

🔹 Key Points of the PSFL:

Feature	Description
✅ Free to Use	You can use Python for **personal, educational, or commercial** purposes with no cost.
✅ Open Source	Source code is available to everyone.
✅ Permissive	You can **modify, distribute, or embed** Python in your own software.
✅ No Copyleft	Unlike GPL, you **don't need to open-source** your project if you use Python.
✅ Compatible with other licenses	Works well alongside BSD, MIT, Apache, etc.
✅ No Warranty	The license disclaims warranties (use at your own risk).

◇ Which version of Python uses this license?

- Since Python 2.1, the language is licensed under the PSFL.
- Earlier versions (before 2.1) had a license similar to the CNRI license.

🔓 Is Python truly "Free"?

YES – Free as in:

- **Free Speech** (freedom to use, change, distribute)
- **Free Beer** (no cost)

🚀 Real-World Implication:

You can:

- Use Python to build apps (even commercial ones)
- Modify the source code of Python itself
- Embed Python in your product
- Distribute your tools without worrying about licensing restrictions

📜 Python History in Short

◇ 1991 – Python was born

- Created by **Guido van Rossum** in the Netherlands.
- First version (**Python 0.9.0**) was released on **February 20, 1991**.
- Inspired by **ABC language** and named after **Monty Python's Flying Circus** (not the snake! 🐍😂)

🐍 Python Compilation Process with PVM

Python is an **interpreted** language, but it still involves a **compilation step** — just not like C or Java.

Let's go step-by-step:

◇ Step 1: Writing Code

You write code in a `.py` file (example: `hello.py`)

```
print("Hello, World!")
```

◇ Step 2: Compilation to Bytecode (.pyc)

- When you run your Python script:
    - Python first **compiles** it to an intermediate code called **bytecode**.
    - This bytecode is saved as a `.pyc` file in the `__pycache__` folder.

```
hello.py  👉  Compiled to  hello.cpython-311.pyc
```

☑ Bytecode is:

- A **low-level**, platform-independent set of instructions.
- Not human-readable.
- Faster to execute than re-parsing `.py` files every time.

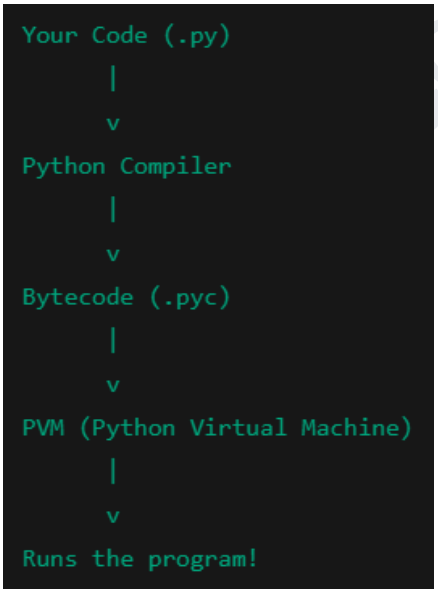
◇ Step 3: Execution by PVM (Python Virtual Machine)

- The **PVM** reads the `.pyc` bytecode and **executes** it line by line.
- It handles:
    - Memory management
    - Variable storage

- Function calls
- Object creation
- Exception handling

✅ The **PVM is the actual runtime engine** that runs your Python code.

🔷 Visualization:



💡 **Key Points:**

Component	Purpose
Compiler	Converts `.py` to `.pyc` (bytecode)
Bytecode	Intermediate code, faster to execute
PVM	Executes bytecode, line-by-line interpreter

❓ Why Not Directly to Machine Code?

Because:

- Python is platform-independent (write once, run anywhere).
- Using bytecode + PVM allows for flexibility and easier debugging.
- Machine code would tie Python to a specific OS/CPU.

☑ Summary:

- Python compiles `.py` → `.pyc` (bytecode)
- **PVM** reads and executes bytecode
- This makes Python **interpreted**, but with an internal **compilation step**

🔁 Contrast with a Compiled Language (like C):

In C:

- Entire `.c` code is **compiled into machine code (.exe)** before running
- Then you run the executable

Aspect	Interpreted (Python)	Compiled (C, C++)
Execution	Line-by-line (at runtime)	Entire code compiled first
Output	No .exe, just runs via interpreter	.exe or binary file generated
Debugging	Easier (fails at the line)	Harder (whole compile fails)
Speed	Slower	Faster

💡 Why Bytecode?

Reason	Explanation
✅ **Faster execution**	Python code is parsed only once. The bytecode is reused, avoiding repeated parsing.
✅ **Platform-independent**	Bytecode can run on any OS that has a Python Virtual Machine (PVM).
✅ **Simplified interpretation**	The interpreter (PVM) doesn't need to understand complex Python syntax — just reads bytecode.
✅ **Caching**	Python stores `.pyc` files in `__pycache__` so future runs are faster.

🧠 Final Analogy:

- Python file (.py) = Your original recipe
- Bytecode (.pyc) = The pre-prepared dish
- PVM = The waiter that serves it to the customer (computer) line by line

🌍 What Does Platform-Independent Mean?

A language is platform-independent if you can write code once and run it on any operating system (Windows, Mac, Linux, etc.) *without changing it*.

🐍 How Python Achieves Platform Independence:

◇ 1. Python Code → Compiled to Bytecode (`.pyc`)

- When you run a `.py` file, Python **compiles it into bytecode** — a **platform-neutral intermediate format**.
- This bytecode is **not specific to Windows or Linux** — it's a common format.

◇ 2. Bytecode is Executed by PVM (Python Virtual Machine)

- The **PVM is platform-specific**, but the **bytecode is the same** on all platforms.
- Think of the **PVM as the waiter who understands your dish (bytecode) no matter the country (OS)**.

📦 Example:

You write a script:

python

```
print("Hello World")
```

You can:

- Run it on **Windows** using Python installed on Windows → compiles to bytecode → executed by Windows PVM
- Run the **same script** on **Mac or Linux** → same bytecode → executed by Mac/Linux PVM

You **don't need to change your code** 👌

🧠 Real-World Analogy:

Think of **bytecode as a "universal recipe" written in symbols**.
Every country (OS) has its **own chef (PVM)** who knows how to read it and cook it.
So the same recipe can be cooked anywhere — just by using the local chef.

✅ Summary:

Component	Role
.py file	Your source code
.pyc file	Platform-independent bytecode
PVM	Platform-specific interpreter

So, the magic lies in:

- Bytecode being universal
- PVM being customized per platform

That's how Python = Write once, run anywhere ✅

✅ How to Install Python on Windows (Official Method)

◇ Step 1: Download Python

1. Go to the **official Python website**:
   👉 https://www.python.org/downloads/
2. Click on the **latest version for Windows**
   Example: "Download Python 3.12.x"

◇ Step 2: Run the Installer

1. **Double-click** the downloaded `.exe` file.
2. 🟡 **Important! Before clicking "Install Now"**: ✅ **Check the box** that says:

   ☐ **Add Python 3.x to PATH**
   This allows you to run Python from anywhere in your terminal.

3. Click **"Install Now"**

◇ Step 3: Wait for Installation

- Python will install along with **pip (Python package manager)**.
- Once done, click **“Close”**.

◇ Step 4: Verify Installation

1. Open **Command Prompt (cmd)**
2. Type:

```
python --version
```

➡️ You should see:

```
Python 3.12.x
```

3. Also try:

```
pip --version
```

➡️ Confirms that **pip** is also installed.

◇ (Optional) Open IDLE or VS Code

- You can search for **IDLE** (Python's built-in editor) in the Start menu
- Or install **VS Code** for a better coding experience

🎯 You're Ready to Go!

Now you can start writing `.py` files and run them from:

- IDLE
- VS Code
- Command Line

❗ What If You Already Installed Python Without Adding to PATH?

No worries! You can **add it manually**:

🔧 How to Add Python to PATH Manually (Windows)

Find Python installation path
Usually it's in:

```
C:\Users\<YourName>\AppData\Local\Programs\Python\Python3x\
```

1. Copy the full path, for example:

```
C:\Users\Gowtham\AppData\Local\Programs\Python\Python312\
```

2. Now do this:
    - Search **"Environment Variables"** in Start
    - Click **“Edit the system environment variables”**
    - In the window, click **“Environment Variables…”**
    - In **System variables**, find `Path` → click **Edit**
    - Click **New**, then paste the Python path

```
C:\Users\<YourName>\AppData\Local\Programs\Python\Python3x\Scripts\
```

    -
3. Click OK → OK → OK
4. Restart your Command Prompt

🧪 **Test it:**

```
python --version

pip --version
```

If it works → you're good to go! 🚀

Step 1: Download PyCharm Community Edition

1. **Visit the Official Download Page:**
    - Navigate to the PyCharm Download page.
2. **Choose Your Operating System:**
    - The page will automatically detect your OS (Windows, macOS, or Linux). If not, select the appropriate tab for your system.
3. **Select the Community Edition:**
    - You'll see two options: **Professional** and **Community**. Under the **Community** section, click the **Download** button.

Step 2: Install PyCharm Community Edition

For Windows:

1. **Run the Installer:**
    - Locate the downloaded `.exe` file (usually in your 'Downloads' folder) and double-click to run it.
2. **Follow the Installation Wizard:**
    - **Welcome Screen:** Click **Next**.
    - **Choose Install Location:** You can use the default location or specify a different folder. Click **Next**.
    - **Installation Options:**
        - *Create Desktop Shortcut:* (Optional) Check this box if you want a shortcut on your desktop.
        - *Update PATH Variable:* (Recommended) Check this to add PyCharm to your system PATH.

        - *Create Associations:* (Optional) Check this to associate `.py` files with PyCharm.

    - Click **Next** and then **Install**.

3. **Complete Installation:**

    - Once the installation is complete, you can choose to run PyCharm immediately or finish the setup.

For macOS:

1. **Open the Disk Image:**

    - Locate the downloaded `.dmg` file and double-click to open it.

2. **Install PyCharm:**

    - Drag the PyCharm icon into the **Applications** folder.

3. **Launch PyCharm:**

    - Navigate to your **Applications** folder and double-click the PyCharm icon to start the application.

For Linux:

1. **Extract the Tarball:**

    - Open your terminal and navigate to the directory containing the downloaded `.tar.gz` file.

Run the following command to extract:

```
tar -xzf pycharm-community-*.tar.gz -C
/desired/installation/path/
```

2. **Run PyCharm:**

Navigate to the `bin` directory inside the extracted folder:

```
cd /desired/installation/path/pycharm-community-*/bin
```

- Run PyCharm using:

  ```
  ./pycharm.sh
  ```

Step 3: Initial Configuration

1. **Import Settings:**
   - **If you've previously used PyCharm and have settings to import, choose the appropriate option. Otherwise, select Do not import settings.**
2. **Customize UI (Optional):**
   - You can choose a theme (Light or Dark) and configure other UI settings as per your preference.
3. **Install Plugins (Optional):**
   - PyCharm may suggest installing additional plugins based on your development needs. You can choose to install them now or later.
4. **Create or Open a Project:**
   - Once the initial setup is complete, you can create a new project or open an existing one to start coding.

Gowtham SB
www.linkedin.com/in/sbgowtham/ Instagram - @dataengineeringtamil

About the Author

Gowtham SB is a **Data Engineering expert, educator, and content creator** with a passion for **big data technologies, as well as cloud and Gen AI** . With years of experience in the field, he has worked extensively with **cloud platforms, distributed systems, and data pipelines**, helping professionals and aspiring engineers master the art of data engineering.

Beyond his technical expertise, Gowtham is a **renowned mentor and speaker**, sharing his insights through engaging content on **YouTube and LinkedIn**. He has built one of the **largest Tamil Data Engineering communities**, guiding thousands of learners to excel in their careers.

Through his deep industry knowledge and hands-on approach, Gowtham continues to **bridge the gap between learning and real-world implementation**, empowering individuals to build **scalable, high-performance data solutions**.

Socials

📽️ **YouTube** - https://www.youtube.com/@dataengineeringvideos

📸 **Instagram** - https://instagram.com/dataengineeringtamil

📸 **Instagram** - https://instagram.com/thedatatech.in

🤝 **Connect for 1:1** - https://topmate.io/dataengineering/

💼 **LinkedIn** - https://www.linkedin.com/in/sbgowtham/

🌐 **Website** - https://codewithgowtham.blogspot.com

💻 **GitHub** - http://github.com/Gowthamdataengineer

💬 **Whats App** - https://lnkd.in/g5JrHw8q

📧 **Email** - atozknowledge.com@gmail.com

📱 **All My Socials** - https://lnkd.in/gf8k3aCH