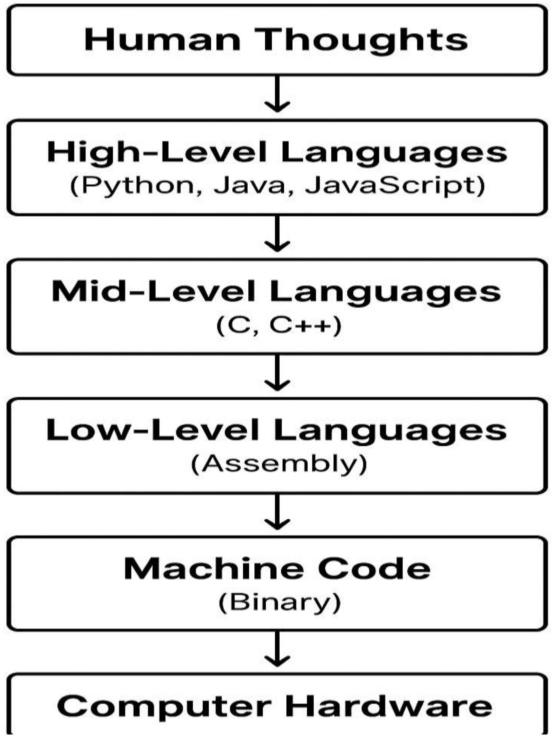
Programming Language

Think of programming languages as **translators** between human thoughts and computer operations. Just like human languages (English, Spanish, French), programming languages help us communicate - but instead of talking to other people, we're talking to computers.



The Classic Example

Assembly Code Example:

```
;Program to add two numbers and store result
MOV AX, 5          ; Move number 5 into register AX
MOV BX, 3          ; Move number 3 into register BX
ADD AX, BX         ; Add BX to AX (result: AX = 8)
MOV [200], AX      ; Store result at memory address 200
HLT                ; Stop the program
```

Python Code Example:

```python
# Program to add two numbers and display result
number1 = 5
number2 = 3
result = number1 + number2
print(f"The result is: {result}")
```

The Translation Process: How High-Level Becomes Low-Level

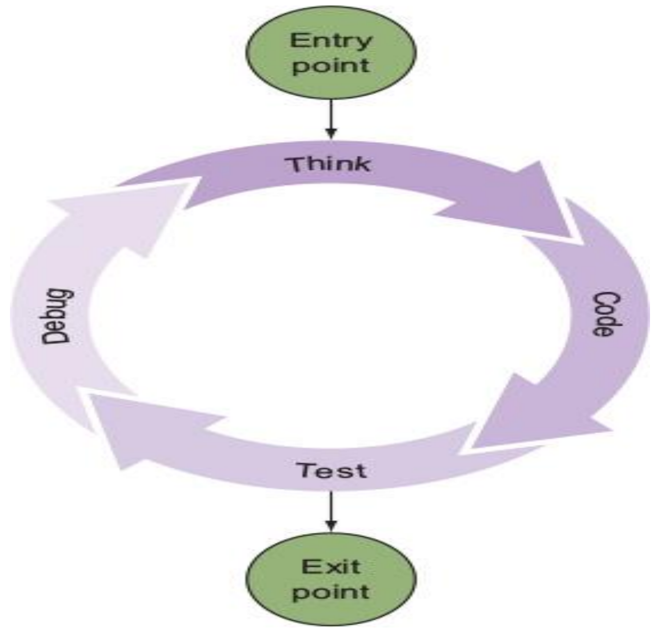
Compilation: High-level code is translated entirely to machine code before execution (C, C++, Rust)
Advantages: Fast execution, operating systems, IoT devices, optimized code **Disadvantages**: Must recompile after changes, platform-specific

Interpretation: High-level code is translated and executed line-by-line during runtime (Python, JavaScript)
Advantages: Immediate execution, web development, cross-platform **Disadvantages**: Slower execution, requires interpreter installed

Virtual Machines: Code is compiled to intermediate bytecode, then interpreted by a virtual machine (e.g., JVM for Java, CLR for C#).
Advantages: Cross-platform, Amazon, eBay backend systems, Banking software, ERP systems, good performance **Disadvantages**: Requires virtual machine, some overhead

BDPN
Data careers for everyone

Basic principles of programming



This is the ideal way to approach solving a problem with programming. Understand the problem before you write any code. Then test the code that you write and debug it as necessary. This process repeats until your code passes all tests.

1. Understanding the task
2. Visualizing the task
3. Write the pseudocode
4. Write the actual code:
    - Use descriptive and meaningful names
    - Commenting your code
    - Test your code as you write

`E.g., Write a program that` Compares word frequencies across different authors to study stylistic differences.

BDPN
Data careers for everyone

Programming Fundamentals

1. Syntax & Semantics
2. Data Types and Variables
3. Operators and Expression
4. Control Flow
5. Functions
6. Data Structures
7. Error Handling
8. Input & Output
9. Memory & References
10. Basic Object-Oriented Programming
11. Libraries & Modules
12. Testing and Debugging

BDPN

Data careers for everyone

Project Structure and Import

```
omission-git/
└── omission/
    ├── __init__.py
    ├── __main__.py
    ├── app.py
    ├── data/
    │   ├── __init__.py
    │   ├── data_loader.py
    │   ├── game_round.py
    │   ├── scoreboard.py
    │   └── settings.py
    ├── omission.py
    ├── pylintrc
    ├── README.md
    └── .gitignore
```

Modules and Packages: A module in Python is simply a single .py file that defines functions, classes, etc,. A package is a directory containing one or more modules plus a special file named __init__.py; this file tells Python that the directory should be treated as a package rather than just a plain folder.

Import

Once you have imported a module, you can access any variable, function or class defined within it.

Entry Point

The part of the project that are run first when importing or executing are called entry points. In python we have the package entry point and the module entry point.