Python: The Journey from Idea to Application

A foundational guide to the world’s most versatile programming language.

Less Code, More Power: Python's Elegant Simplicity

C

```c
#include<stdio.h>
#include<conio.h>

void main()
{
  printf("Hello World");
}
```

Java

```java
public class Demo
{
  public static void main(String args[])
  {
    System.out.println("Hello World");
  }
}
```

Python

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

Python lets you focus on your ideas, not on complex syntax.
It is developer-friendly, more understandable, and more readable.

Python programs are typically 3-5 times shorter than equivalent Java programs, and 5-10 times shorter than C++.
Python programs are typically 3-5 times shorter than equivalent Java programs, and 5-10 times shorter than C++.

A General Purpose Tool for Every Task

Data Science & Machine Learning

For data analysis and building AI applications.

Web Development

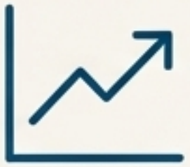
Using powerful frameworks like Django and Flask.

Automation & Scripting

Borrowing features from Perl and Shell script.

Desktop & GUI Applications

For creating user-facing software like calculators.



Scientific & Numeric Computing

For database and networking applications.

Game Development & IoT

For creating games and Internet of Things applications.

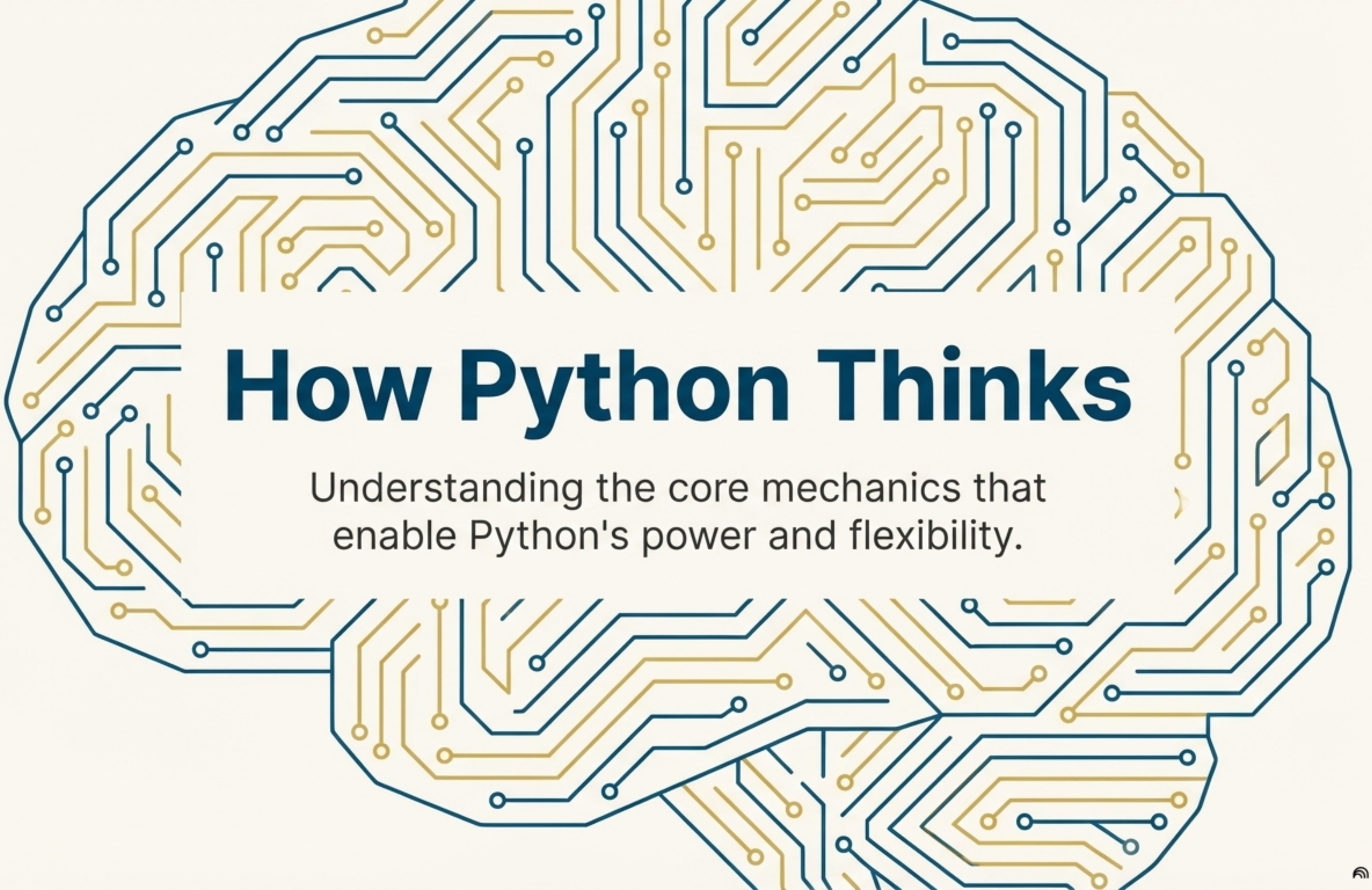
We can use Python for multiple purposes.
Therefore Python is a general purpose programming language.

Born from a Need for Clarity

- **Creator:** Guido van Rossum, developed in 1989 at the National Research Institute in the Netherlands.
- **Public Release:** Available to the public on 20 February 1991.

The Name: Guido van Rossum was also reading the published scripts from Monty Python's Flying Circus… he needed a name that was short, unique, and slightly mysterious, so he decided to call the language Python.

Philosophy: To create a language that borrowed the best features from others, with a focus on readability. Most of its syntax was borrowed from C and ABC.

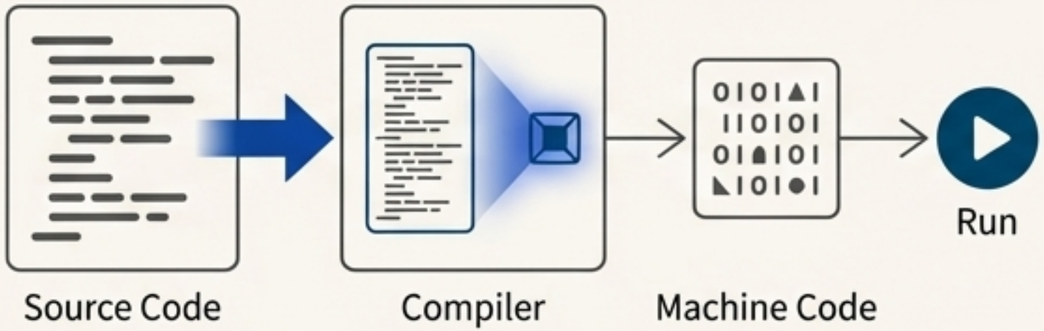


How Python Thinks

Understanding the core mechanics that enable Python's power and flexibility.

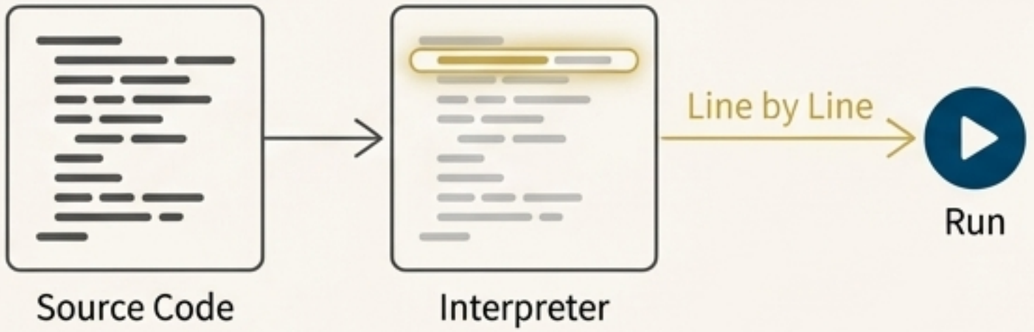
A Different Approach to Execution

Compiled (C, Java)



- Takes entire program as input.
- Errors displayed after the entire program is read.

Interpreted (Python)



- Takes one line of program as input at a time.
- Errors displayed after each instruction is read.

Python is an **interpreted language** i.e. interpreter executes the code **line by line** at a time. This makes the development **and debugging** fast because there is no compilation step… the **edit-test-debug** cycle is very fast.

The Freedom of Flexibility: Dynamic vs. Static Typing

Statically Typed (e.g., Java, C)

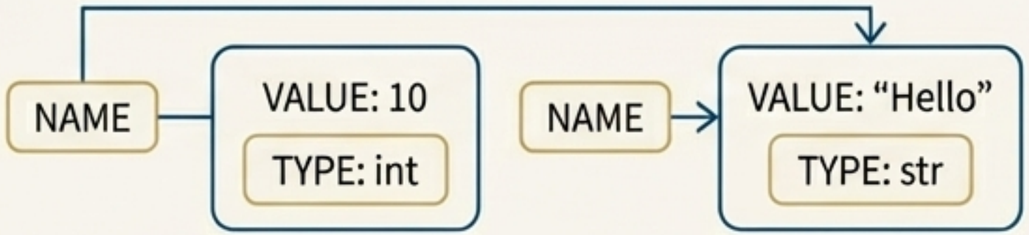
The type of a variable is known at compile-time.

```
int data = 50;
data = "Hello"; // ERROR: Illegal
```

Dynamically Typed (Python)

The type of a variable is checked during run-time.

```
data = 10       # data is an integer.
data = "Hello" # Now data is a string.
                 Works perfectly.
```



In Python, **variables are just a label given to any data value.** You do not need to **declare the data types** of your variables before you use them.

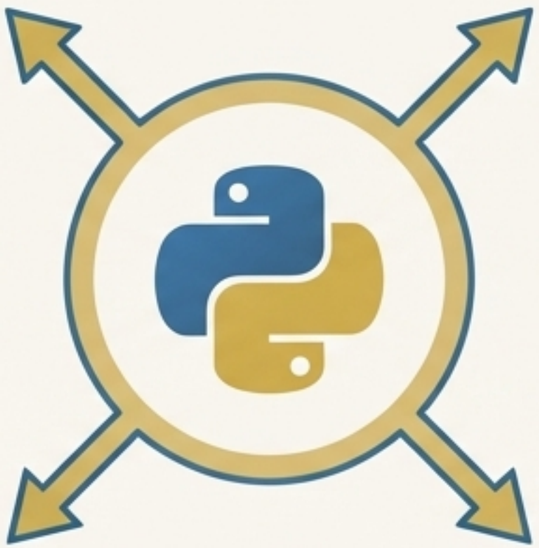
A Multi-Paradigm Language

Procedural Programming

From C. (Organised into functions, top-down approach).

Object-Oriented Programming (OOP)

From C++. (Organised into objects containing data and methods).



Scripting Language Features

From Perl and Shell Script. (Ideal for automation).

Functional Programming Features

From C. (Treating functions as first-class objects).

Python gives you the right tool for the job. It is Functional, Object oriented, scripting and modular language.

The Building Blocks of Python

Your first practical steps in writing Python code.

Storing Information: Variables, Data Types & Comments

Variables

A variable is created the moment you first assign a value to it.

Rules:
Must start with a letter (a-z, A-Z) or underscore (_).
Can only contain alpha-numeric characters and underscores.
Case-sensitive (`age`, `Age`, and `AGE` are different).

```
age = 25
```

Core Scalar Data Types

List:
`int`. Integers (e.g., `10`, `-56`)
`float` Floating-point numbers (e.g., `3.14`, `1.0e10`)
`bool`: Boolean values (`True` or `False`. Note: Python is case-sensitive).
`str`: A sequence of characters (`"hello"`, `'world'`)
None: Represents ‘no value’ or empty.

Bottom Note: Check type with: `type(variable_name)`

Comments

Single-line:

```
# for single-line comments.
```

Multi-line:

```
"""..."""
```

for multi-line string literals, which can be used as comments.

The Operator's Toolkit

Arithmetic

+	add	-	subtract
*	multiply	/	floor division
//	divide	**	exponent
%	modulus		

Comparison

==	equal	!=	not equal
<	less than	>	greater than
<=	less/ equal	>=	greater/ equal

Logical

and	Source Sans Pro
or	Source Sans Pro
not	Source Sans Pro

Key Note

Python follows the basic PEMDAS order of operations. Exponent operator `**` has right-to-left associativity; almost all others are left-to-right.

Interesting Fact

Python does not have pre/ post increment/decrement operators like `++` or `--`.

The Art of Input and Output

Getting Input with `input()`

Used to take input from the user. By default, it takes input in ‘string’ type. If you need a number, you must perform a type conversion.

```
# The int() function converts the string input to an integer
age = int(input("Enter your age: "))
```

Displaying Output with `print()`

Used to display output to the console.

- Separating items with commas `print("Name:", name)`
- Using `sep` parameter `print('how', 'are', 'you', sep='$')` -> Output: howareyou
- Using `end` parameter `print("Hello", end="!")` -> Output: Hello! (no new line)
- Using f-strings for formatting `print(f"You are {age} years old.")`

Making Decisions: `if`, `elif`, and `else`

Controlling program flow with conditional statements.

```
score = 85

if score >= 90:
    grade = "A"
elif score >= 80:
    grade = "B"
else:
    grade = "C"

print(f"Your final grade is {grade}")
```

Indentation is Mandatory

In other programming languages, indentation is for readability only, but in Python it is used to indicate a block of code. It is not for style, it's syntax!

Note: Python does not have a `switch` statement.

Repeating Actions: `for` and `while` Loops

`for` loop

For iterating over a known sequence (like a range of numbers or characters in a string).

```
# Prints numbers 0, 1, 2, 3, 4
for i in range(5):
    print(i)
```

`while` loop

For looping as long as a certain condition is true.

```
count = 0
while count < 5:
    print(count)
    count = count + 1 # or count += 1
```

Loop Control Statements

- **break**: Terminates the loop entirely.
- **continue**: Skips the rest of the current iteration and moves to the next.
- **pass**: A placeholder; does nothing. Used when a statement is required syntactically but no code needs to be executed.

You Now Have the Keys

WHY

Python is simple, readable, and incredibly versatile, letting you build almost anything.

WHAT

It is an interpreted, dynamically-typed language with a flexible, multi-paradigm design that borrows the best from other languages.

HOW

You can now use variables, operators, and control structures (`if/else`, `for/while`) to build your first programs.

Next Steps

Explore Python's rich standard library, including powerful data structures like Lists, Dictionaries, and Tuples, and learn how to organise your code with Functions. Happy coding!