Welcome to the World of Python!

> *"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."*
> — Martin Fowler

📘 Topics to be Covered

1. **What is Python ?**
2. **Key Features of Python**
3. **Limitations of Python**
4. **Scope of Python**
5. **Python — A Programming or a Scripting Language?**
6. **Python Internals: Stages of Execution (The Python Execution Model)**
7. **Primitive data types in Python**

---

1. What is Python?

Python is a high-level, interpreted programming language known for its **simplicity**, **readability**, and **versatility**.

- Created by **Guido van Rossum**
- First released in **1991**
- Now used by companies like **Google**, **Netflix**, **NASA**, **Tesla**, and **Meta**

Python makes it **easy for beginners to start**, while still being **powerful enough** for experts to build complex systems.

Python is one of the most popular programming language. why ?

- Used in **cutting-edge fields**: AI, Machine Learning, Data Science, Automation, and more.
- Beginner-friendly syntax.
- Fast development and prototyping.
- A huge collection of libraries for **everything**.
- A massive, supportive community.

---

2. Key Features of Python

- **Interpreted** – No compilation required; run code line-by-line.
- **Dynamically Typed** – No need to declare variable types.
- **High-Level Language** – Write code like you're writing English.
- **Object-Oriented** – Build reusable and organized code.
- **Cross-Platform** – Runs on Windows, Mac, Linux seamlessly.

---

3. Limitations of Python

Even the best tools have limitations:

- **Slower than C++/Java** because it is interpreted.
- **Not ideal for mobile app development.**
- **GIL (Global Interpreter Lock)**: Limits true multithreading.
- **Runtime Errors** due to dynamic typing.
- **High memory usage** compared to lower-level languages.

---

4. Scope of Python

Python shines in:

Domain	Tools/Libraries	What You Can Build
Data Science	`pandas`, `numpy`	Analyze real-world data
Machine Learning	`scikit-learn`, `xgboost`	Predict house prices, recommend videos
AI / Deep Learning	`tensorflow`, `pytorch`	Build chatbots, autonomous systems
Web Development	`Flask`, `Django`	Build websites and REST APIs
Automation	`selenium`, `pyautogui`	Auto-fill forms, schedule emails
Games	`pygame`	Build 2D games and interactive apps

---

5. Python — A Programming or a Scripting Language?

- Python is a general-purpose programming language that is often used as a scripting language.
- Python as a scripting language - Used for quick automation or system-level tasks.

```python
# Rename all .txt files in a folder
import os
folder = "C:/Users/Sanjeev/Desktop/textfiles"
for filename in os.listdir(folder):
    if filename.endswith(".txt"):
        new_name = filename.replace(" ", "_")
        os.rename(os.path.join(folder, filename), os.path.join(folder, new_name))
```

- Python as a programming language - Used to build applications by organizing code using modules/classes/functions.

```python
# Simple Student Management System
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def get_grade(self):
        if self.marks >= 90:
            return "A"
        elif self.marks >= 75:
            return "B"
        else:
            return "C"

def main():
    students = [
        Student("Sanjeev", 88),
        Student("Shambhavi", 95)
    ]

    for s in students:
        print(f"{s.name} got grade: {s.get_grade()}")

if __name__ == "__main__":
    main()
```

- **Python is an object-oriented programming language like Java** – you can write class-based code, just like Java.
- **Python is interpreted**, but **Java is compiled** – it runs via an interpreter rather than being compiled to machine code directly.

Let's understand this more clearly with an analogy:

Analogy: Chef & Recipe

Concept	Python	Java
Cooking Style (OOP)	Object-Oriented	Object-Oriented
Cooking Method (Execution)	Interpreted *(live cooking)*	Compiled *(pre-cooked and served)*

---

6. Python Internals: Stages of Execution (The Python Execution Model)

- Sample Python code for Analysis :

```
x = 5 + 3

The above code is saved in .py file in local system.
```

6.1. Lexical Analysis and Tokenization

- The first step in the execution process is **lexical analysis**, also known as **tokenization**.
- During this phase, the source code is converted into a sequence of **tokens**.
- **Tokens**:
  `[ x (identifier) , = (operator), 5 (literal), + (operator), 3 (literal) ]`

6.2. Syntax Analysis and Abstract Syntax Tree (AST)

- During this phase, the sequence of tokens is analyzed to determine its grammatical structure.
- The output of this phase is an Abstract Syntax Tree (AST), which represents the hierarchical structure of the source code.

```
Assignment
├── Target: x
└── Value
    ├── Left: 5
    └── Right: 3
        └── Operator: +
```

6.3. Bytecode Compilation

- The AST is then compiled into bytecode, which is a low-level, platform-independent representation of the source code.
- Bytecode is a set of instructions that can be efficiently executed by the Python Virtual Machine (PVM).
- Generates a .pyc file
- Bytecode Examples

```
1 LOAD_CONST 5
2 LOAD_CONST 3
3 BINARY_ADD
4 STORE_NAME x
```

6.4. Execution by the Python Virtual Machine (PVM)

- PVM is the Python Interpreter that converts the Python byte code into machine-executable code.
- PVM interpreter reads and executes the given file line by line.
- The AST is then compiled into bytecode, which is a low-level, platform-independent representation of the source code.
- CPU runs the native machine instructions triggered by the PVM

---

7. Fundamental Data types in Python

In Python, datatypes define what kind of value a variable can hold. Python is dynamically typed, so you don't have to declare types explicitly—they're inferred at runtime.

Fundamental (or primitive) data types are the basic building blocks of data in Python. They are simple, core types from which more complex structures can be made.

A literal is a value. A variable is a name pointing to that value.

A variable in Python is defined through assignment. There is no concept of declaring a variable outside of that assignment.

Integers

Integer literals are created by any number without a decimal or complex component.

In [10]:
```
# integers
x = 1234

print(f'Type : {type(x)}')
print(f'val : {x}')
```

```
Type : <class 'int'>
val : 1234
```

In Python, everything — even primitive data types like int, str, bool, etc. — is a class

Every value (even 5, "hello", True) is an instance of a class.

This makes Python powerful and flexible.

Floats

Float literals can be created by adding a decimal component to a number.

In [8]:
```
# float
x = 1.0

print(f'Type : {type(x)}')
print(f'val : {x}')
```

```
Type : <class 'float'>
val : 1.0
```

Boolean

Boolean can be defined by typing True/False without quotes

In [7]:
```
# boolean
b1 = True
b2 = False

print(f'Type : {type(b1)}')
print(f'val : {b1}')
```

```
Type : <class 'bool'>
val : True
```

Strings

String literals can be defined with any of single quotes ('), double quotes (") or triple quotes (''' or """). All give the same result with two important differences.

If you quote with single quotes, you do not have to escape double quotes and vice-versa. If you quote with triple quotes, your string can span multiple lines.

In [15]:
```
name = "Hello I'm double quotes !"          # Double quotes
print(f'name Type : {type(name)}')
print(f'name val : {name}')
#--------------
```

```
greet = 'Hello I\'m single quotes!'            # Single quotes
print(f'greet Type : {type(greet)}')
print(f'greet val : {greet}')
#--------------
multi_line = '''
Hi
This is Python
'''          # Triple quotes for multi-line strings
print(f'multi_line Type : {type(multi_line)}')
print(f'multi_line val : {multi_line}')
```

```
name Type : <class 'str'>
name val : Hello I'm double quotes !
greet Type : <class 'str'>
greet val : Hello I'm single quotes!
multi_line Type : <class 'str'>
multi_line val :
Hi
This is Python
```

Complex

Complex literals can be created by using the notation x + yj where x is the real component and y is the imaginary component.

In [16]:
```
# complex numbers: note the use of `j` to specify the imaginary part
x = 1.0 - 2.0j
type(x)
```

Out[16]:
```
complex
```

In [17]:
```
print(x)
```

```
(1-2j)
```

In [18]:
```
print(x.real, x.imag)
```

```
1.0 -2.0
```

Dynamic Typing

In Python, while the value that a variable points to has a type, the variable itself has no strict type in its definition. You can re-use the same variable to point to an object of a different type.

In [19]:
```
ten = 10
print(ten)

ten = 'ten'
print(ten)
```

```
10
ten
```

Boolean Type

In [21]:
```
is_valid = True
print(is_valid)
is_empty = False
print(is_valid)
```

```
True
False
```

In [23]:
```
# Boolean evaluation
print(True and False)
print(True or False)
print(not True)
print('a' == 'a')
```

```
False
True
False
True
```

None Type

Represents the absence of a value.

In [25]:
```
data = None
print(data)
print(type(data))
```

```
None
<class 'NoneType'>
```

Type Checking and Conversion

In [24]:
```
a = "10"
print(type(a))
a = int(a)
print(type(a))
```

```
<class 'str'>
<class 'int'>
```

In [ ]:
```

```

📚 References

1. The Python execution model details src 1
2. The Python execution model src 2