PYTHON

Instructor : Nirajan Bekoju

ML Engineer, Fusemachines Inc.

Imagine a tool that's powerful enough to drive Artificial Intelligence, easy enough to Create Websites, flexible enough to Analyze Data, and handy for Scientific Research, all while being Beginner-Friendly!

That's Python...

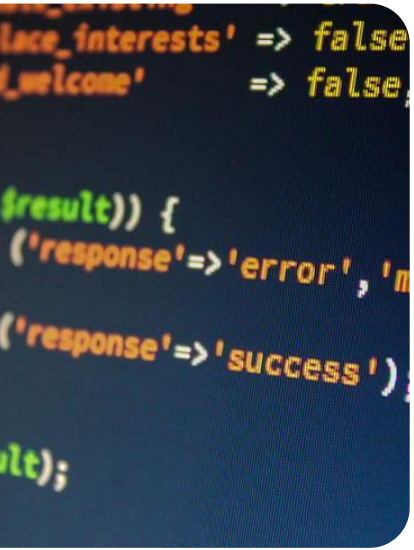
Training - Outline

- Introduction to Python Programming
- Data Types, Variables and Strings
- Basic Data Structures
- Conditionals, Loops, and Functions
- Object-oriented Programming (OOP)
- Decorators and Generators
- Error Handling, Logging and Debugging
- File Handling
- Introduction to Modules and Libraries
- Data Science and Visualization Basics
- Introduction to Machine Learning
- Coding Exams and Interview Questions

5 Reasons Why You Should Learn Python

- Beginner-Friendly
- Versatility Across Fields
- Career Opportunities
- Extensive Libraries & Community Support
- Cross-Platform & Future-Proof

print(" Welcome to Python ! Your journey into code begins here.")



Introduction

- ❏ Programming Language
- ❏ Overview of Python and its applications
- ❏ Setting up the environment (Virtualenv., Jupyter Notebooks, VSCode)
- ❏ Python Syntax, Variables, and Data Types

Programming Language

- Programming Language is a formal language that is used to write instructions that a computer can understand and execute.

Feature	High-Level Languages	Low-Level Languages
Abstraction	Closer to human language	Closer to machine language
Ease of Use	Easier to read, write, and maintain	Harder to read and write
Examples	Python, Java, C++	Assembly, Machine Code
Portability	Platform-independent	Platform-specific
Execution Speed	Slower (needs translation)	Faster (executed directly)

Hello world !

Hello World Program in 3 Different Language.

C

```c
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```

Python

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

Java

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

Overview of Python and its applications

- Python is a high-level, interpreted programming language created by Guido van Rossum and first released in 1991.
- Known for its simplicity and readability.
- Open-source and cross-platform.

Van Rossum was inspired by the British comedy series Monty Python's Flying Circus and wanted to create a language that was fun and approachable. He also wanted to create a language that was easy to understand and efficient to write, with clear syntax.

Built Python from scratch in three months using a C compiler on his home Macintosh.

Top Companies who uses Python

Netflix uses python in personalized recommendations, Data analytics, Text automation. etc.

Google uses python in web services, in data processing and in Machine Learning and AI etc.

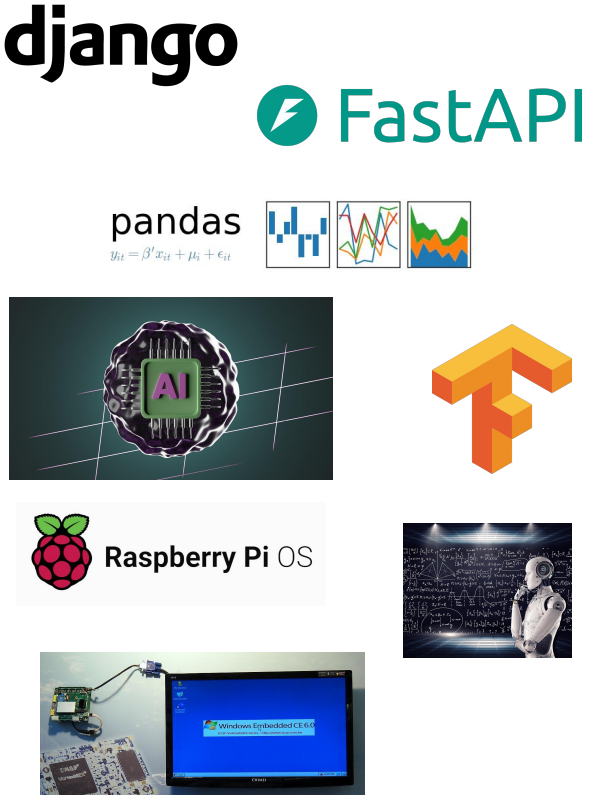
Spotify uses python for data processing and data analysis. etc.

Instagram (Meta) uses python for performance optimization and in web framework (django). etc.

Reddit uses python to handle massive traffic, web services and at the core. etc.

Applications of Python Programming

- Web Development
- Machine Learning & Artificial Intelligence
- Data Science and Analytics
- Automation/Scripting
- IoT and Embedded Systems
- Game Development, Cybersecurity etc.



SET UP

- Install Python
- Install Vs Code (IDE)
- Download Packages: Virtualenv , ipykernel.. etc.

VS Code

VS Code is a lightweight, customizable IDE with built-in Python support, debugging, and Git integration, making it perfect for both beginners and professionals.

VS Code offers excellent support for Jupyter Notebooks through its extensions

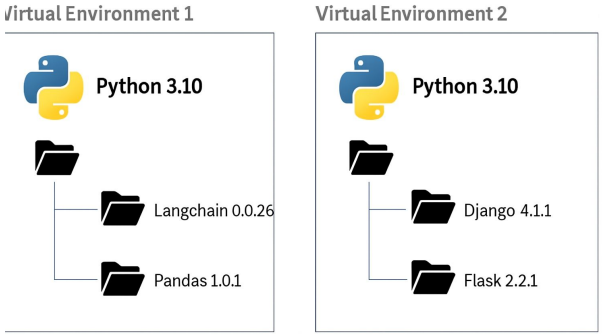
Virtualenv

Virtualenv is a tool for creating isolated Python environments, allowing developers to manage dependencies for different projects without conflicts. It's lightweight, easy to use, and ensures consistent environments across development and deployment.

Coding.....

Virtual Environment

- A virtual environment in Python is an isolated environment on your computer, where you can run and test your Python projects.
- Think of a virtual environment as a separate container for each Python project. Each container:
  - Has its own Python interpreter
  - Has its own set of installed packages
  - Is isolated from other virtual environments
  - Can have different versions of the same package



Virtualenv

Installation

python -m venv /path/to/new/virtual/environment

Creating Environment in Windows

python -m venv /path/to/new/virtual/environment

Note: On Microsoft Windows, it may be required to enable the `Activate.ps1` script by setting the execution policy for the user. You can do this by issuing the following PowerShell command:

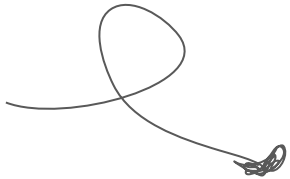
```
PS C:\> Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```

See About Execution Policies for more information.

Activate

For Linux:
source .venv/bin/activate

For Windows:
.venv\Scripts\activate



<venv>\Scripts\Activate.ps1

Deactivate

deactivate

Syntax, Variable and Data Type

Syntax :

Syntax refers to the guidelines that determine the structure of a language. In computer programming, syntax specifies the rules governing the arrangement of symbols, punctuation, and words within a programming language.

We Know:

Programming languages function on the same principles.

If the syntax of a language is not followed, the code will not be understood by a compiler or interpreter.

Key Features of Python Syntax

Indentation: Python uses indentation to define code blocks instead of braces { }.

Case Sensitivity: Python is case-sensitive.

Comments: Single line comment (#) , Multiline comment (''' ''').

Statements: Python executes the statement in line by line manner. I.e one line at a time.

Input/Output : input() is used to take user input. And print() is used to show the output.

Syntax, Variable and Data Type

Variable :

Variable is something that represent some information(value) that may be string, number or any data type. They provide a way of labeling data with a descriptive name, so our programs can be understood more clearly by the reader and ourselves.
Naming a variable is important task in programming, so make sure that the name you assign your variable is accurately descriptive and understandable to another reader.
Rules for creating the variable name:

- Must start with a letter or the underscore character
- Cannot start with a number
- Can only contain alphanumeric characters and underscores (A-z, 0-9, and _)
- Case-sensitive (mobile, Mobile and MOBILE are three different variables)
- Cannot be any of the Python keywords. (Check >> help("keywords"))

Python do not have any command for creating the variable, you can just type variable name and the value along with it. Variables do not require a specific type declaration and can change their type even after being assigned a value. Type casting is also there if you want to specify the variable type.

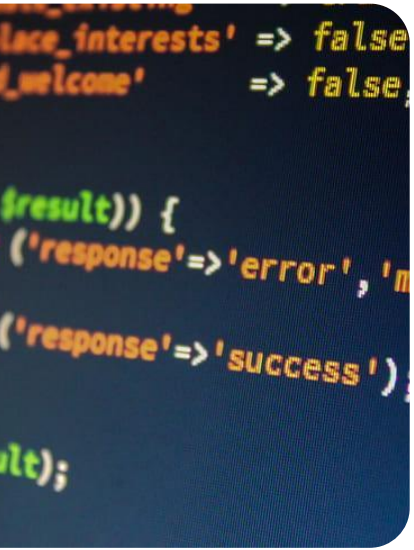
Syntax, Variable and Data Type

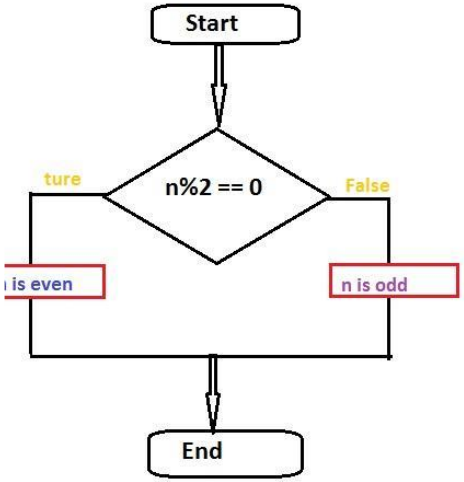
Data Type :

Data type refers to the classification or category of data that determines the kind of value a variable can hold and the operations that can be performed on it. It defines the nature of data, such as numeric, textual, or logical, and helps the compiler or interpreter understand how the data will be used. They define how memory is allocated for the data and also help to ensure program correctness by preventing invalid operations.

Data Types	Classes	Description
Numeric	int, float, complex	holds numeric values
String	str	holds sequence of characters
Sequence	list, tuple, range	holds collection of items
Mapping	dict	holds data in key-value pair form
Boolean	bool	holds either `True` or `False`
Set	set, frozenset	hold collection of unique items

Python is a dynamically typed language, meaning variables do not need explicit type declarations.





Control Structures and Functions

- Conditional statements: if, else, elif
- Loops: for, while, and control keywords (break, continue)
- Functions: Defining, calling, return values, lambda functions
- Scope and Lifetime of Variables

Control Structures and Functions

Conditional statements:

Conditional statements in programming are control flow structures that allow a program to make decisions and execute specific blocks of code based on whether a certain condition is True or False. These conditions are typically based on logical or comparison expressions.

Types of Conditional Statements

- **if**: Executes a block of code if the condition is true.
- **else**: Executes a block of code if the condition is false.
- **elif**: Checks additional conditions if the initial if condition is false

Why Are Conditional Statements Required?

- Decision-Making
- Control Flow
- Dynamic Behavior
- Error Handling
- Efficient Resource Usage

Control Structures and Functions

Loops:

Loops are a fundamental concept in programming that allow you to execute a block of code repeatedly as long as a specified condition is true. Python provides two main types of loops: **for** and **while**, along with control keywords like **break**, **continue**, and **pass** to manage loop behavior.

For Loop

Used to iterate over a sequence (like a list, tuple, string, or range). Often used when the number of iterations is known beforehand. Combined with the enumerate() function for index-value pairs:

While Loop

The while loop runs as long as a specified condition evaluates to True. Used when the number of iterations is not known beforehand and depends on a condition.

Control College

Control keywords allow you to manage the flow of loops by breaking out, skipping iterations, or using placeholders.

Break: Terminates the loop prematurely, regardless of the loop condition.

Continue: Skips the current iteration and moves to the next iteration of the loop.

Pass: Does nothing. It serves as a placeholder when a statement is syntactically required but no action is desired.

Control Structures and Functions

Function:

Functions are "self contained" modules of code that accomplish a specific task. Functions usually "take in" data, process it, and "return" a result. Once a function is written, it can be used over and over and over again. Functions can be "called" from the inside of other functions.

- **Defining Function:**
  A function is a block of reusable code that performs a specific task. In Python, functions are defined using the def keyword.
- **Calling Function**:
  To execute a function, you call it by its name and pass the required arguments (if any).
- **Return Values:**
  Functions can return a value using the return statement. The return keyword ends the function execution and sends a value back to the caller.

Lambda Functions (Anonymous Functions)

A small, one-line, anonymous function defined using the lambda keyword. It is useful for short operations or functions that are used as arguments to other functions. Lambda functions are often used with higher-order functions like map, filter, or reduce.

Control Structures and Functions

Scope:

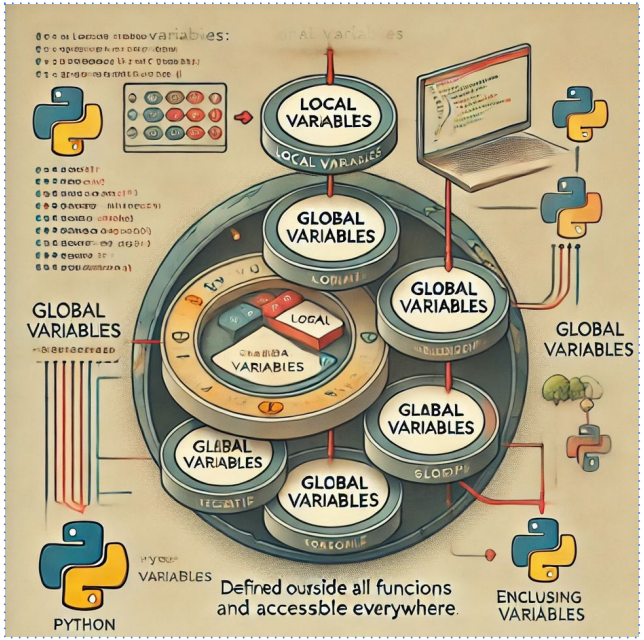
In Python, the scope of a variable refers to the area of a program where that variable can be accessed or modified. The lifetime of a variable refers to how long the variable exists in memory during the program's execution.

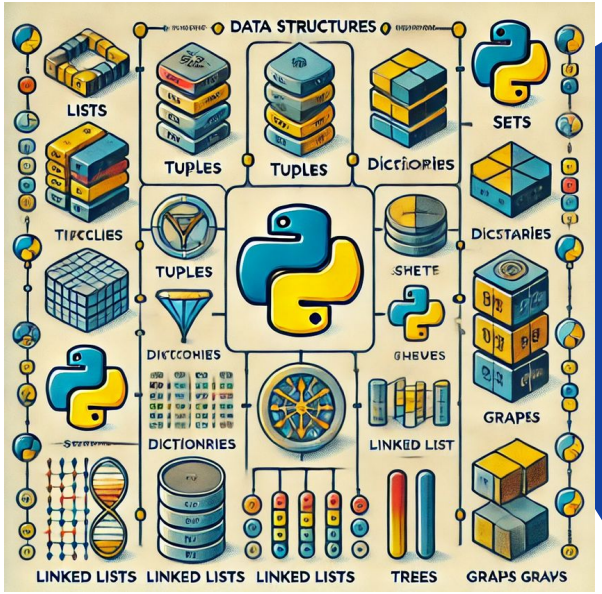
Scope of variables:

- Local
- Enclosing
- Global
- Built-in

Lifetime of Variables

- Local Variables
- Global Variables
- Enclosing Variables





Data Structures

- Lists, Tuples, Sets, Dictionaries
- List and dictionary comprehensions
- Basic operations and methods for each data structure
- Common use cases in data manipulation

Data Structures

A data structure is a method for organizing, storing, and managing data in a way that allows for easy access and modification. It specifies how data is arranged in memory and outlines how various operations—like searching, sorting, inserting, and deleting—can be carried out efficiently.

Common examples of data structures include lists, tuples, sets, dictionaries, arrays, stacks, queues, linked lists, trees, graphs, and hash tables.

Lists: Mutable, ordered, indexing, slicing, methods (append(), remove(), sort(), reverse(), etc.)
Tuples: Immutable, ordered, packing & unpacking, count(), index()
Sets: Unordered, unique elements, set operations (union, intersection, difference)
Dictionaries: Key-value pairs, accessing/modifying elements, dictionary methods (keys(), values(), items(), update(), etc.)

Advance Data Structure:

- Stack (LIFO), Queues, Priority Queues, LInked LIsts
- Hash Tables
- BInary Trees, BST
- Graph Data Structure

Data Structures

List and Dictionary Comprehensions:

In Python, List and dictionary comprehensions provide a concise way to create lists and dictionaries in Python using a single line of code.

List Comprehension

Syntax: [expression for item in iterable if condition]

Dictionary Comprehension

Syntax: {key: value for item in iterable if condition}

Why ? - Concise and readable code

- Faster than traditional Loops
- Useful while filtering and transforming

Basic Operations and Methods:

List : Accessing Elements, Slicing, Adding Elements (append, insert, extend, remove, pop, del, sort, sorted, reverse, iterate)

When it is used ?

- Sorting and processing sequential data (eg, list of numbers, user data)
- Sorting and searching
- Stacks (append, pop) and Queues (collection.deque)

Data Structures

Dictionary: Accessing Values, adding and updating values, removing keys, iterating(checking membership, merging dictionaries)

When it is used ?

- For Lookups using key-value pairs
- Storing structuring data (JSON-like objects)
- Caching results (Memoization)

Tuple: Accessing Values, Slicing, Concatenation, Repetition, Checking membership, Coughing occurrence.

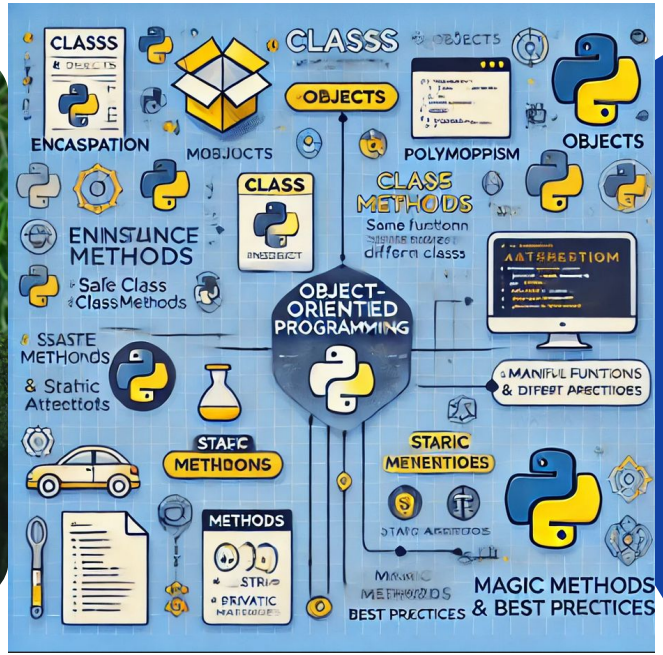
When it is used ?

- Storing fixed collection of data (coordinates, db records)
- Returning multiple values from functions
- Hashable feature.

Set: Adding elements, Removing elements, Checking membership, Set operations(union, intersection, difference, symmetric difference)

When it is used ?

- Removing duplicate value from list
- Checking fast membership
- Mathematical set operations (union, different etc)



Object-oriented Programming (OOP)

- ❑ Classes and Objects
- ❑ OOP Principles
- ❑ Methods in Classes
- ❑ Magic (Dunder) Methods & Best Practices

Object-oriented Programming (OOP)

Object-Oriented Programming (OOP) is a programming paradigm that organizes code into objects, which bundle data (attributes) and behavior (methods) together.

- It helps structure code using real-world entities.
- Examples: A car (object) has attributes (color, model) and behaviors (drive, stop).

Key Principles of OOP

1. **Encapsulation –** Binding data and methods together, Restrict access to data using private (__var) and protected (_var) attributes.
2. **Inheritance –** Reusing code from another class, Allows a child class to reuse the parent class's properties and methods.
3. **Polymorphism –** Methods behaving differently based on the class it belongs to.
4. **Abstraction –** Hiding implementation details and exposes only the essential features.

Classes and Objects

- Classes are blueprints for creating objects, and objects are instances of classes.
- Objects store attributes (variables) and methods (functions) that define their behavior.

Object-oriented Programming (OOP)

The __init__ Method (Constructor) :

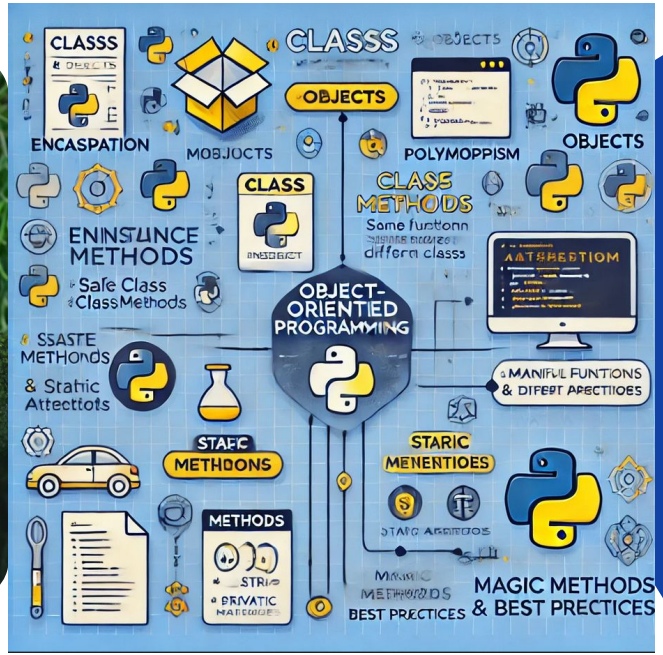
__init__ is a special method (dunder method) that initializes object attributes when an instance is created. It is automatically called when a new object is instantiated.

Self represents the instance of the class and allows access to its attributes and methods. It must be the first parameter in instance methods but is not a keyword (can be renamed, though not recommended).

ALL OOPs concepts and principles will get clearer while practicing with theories.

Best Practices in OOP Python

- Follow **naming conventions** (CamelCase for classes, snake_case for variables).
- Use **getter/setter methods** for accessing private variables.
- **Prefer Composition over Inheritance** when possible.
- Avoid **deep inheritance trees** (complex debugging).
- Use **dunder methods** to improve class usability.



Error Handling and Debugging

- ❑ Exception Handling
- ❑ Built-in Exceptions
- ❑ Debugging

Error Handling and Debugging

Exception Handling (try, except, finally)

Python uses try-except blocks to catch and handle runtime errors without crashing the program. The finally block is used for cleanup (e.g., closing files).

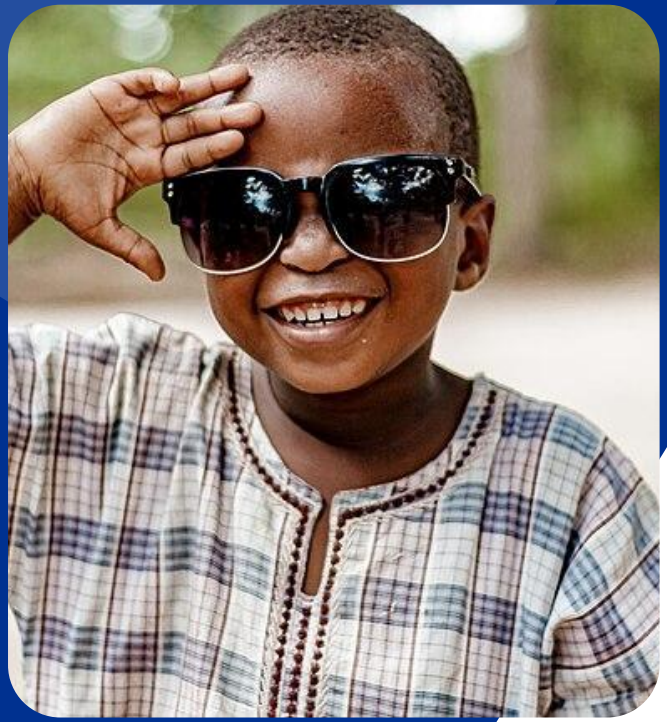
Built-in Exceptions and Custom Exceptions

- Python provides built-in exceptions (ValueError, TypeError, IndexError, etc.).
- We can create custom exceptions using raise and define them using a custom class.

Debugging with print(), Logging, and pdb (Python Debugger)

- Use print() statements for quick debugging.
- Use the logging module for structured debugging with log levels (DEBUG, INFO, ERROR).
- The pdb module allows interactive debugging with breakpoints.

Use linters (like flake8, pylint) and IDEs (like PyCharm, VSCode) for better debugging.



Thank you for your time today.