Day 1: Python Basics – Part 1

Trainer Name: Upendra

Today's session started with a short Q&A, where the trainer tried to understand how much the audience already knows about Python.

Questions Asked:

1. How familiar are you with Python programming?
2. How and where have you used Python in real-life work situations?
3. Have you ever created or worked with models in real-time?

Main Objective of the Training: To build **Agentic AI**

Models.

Q1: What is an AI Model?

An AI model is a computer program trained to think and make decisions like humans by learning from data and finding patterns in it.

Simple Example:

- You show a computer many pictures of cats and dogs and tell it which is which.
- The computer learns from those examples.
- Later, when you show a new picture, it can guess whether it's a cat or a dog. That is how an AI model works.

In short:

- **Model:** A relationship between the input we give and the output we get.
- **AI Model:** A trained system that learns from data (input) to make predictions or decisions (output).

Q2: What are the Requirements for Model Building?

1. A clear problem to solve.
2. Enough good-quality data related to the problem.
3. Clean and prepare the data (fix errors, remove missing values, etc.).
4. Choose the right algorithm or model type.

5. Train the model so it can learn patterns.
6. Test the model on new data to check accuracy.
7. Tune and improve the model for better results.
8. Use the trained model in real-life situations.

Trainer's Explanation:
The main requirements to create a model are **data** and an **algorithm**.

- The data is used to train the model.
- The algorithm defines what type of model is being built.
  Both are given as inputs to the machine to create the final model.

Q3: What are the Different Types of Algorithms?

Supervised Algorithms

- Learn from **labeled data** (data that already has correct answers).
- Example: Showing a model pictures of apples and bananas with names so it learns to tell them apart.
- Examples include linear regression, logistic regression, decision trees, random forest, support vector machine, k-nearest neighbors, and neural networks.

Unsupervised Algorithms

- Learn from **unlabeled data** (data without correct answers).
- The model finds patterns or groups on its own.
- Example: Giving fruit pictures without names, and the model groups similar ones together.
- Examples include k-means clustering, hierarchical clustering, principal component analysis (PCA), and association rule learning.

Other Types:

- **Semi-supervised learning:** A mix of both labeled and unlabeled data.
- **Reinforcement learning:** The model learns through trial and error, like how a robot or game AI improves over time.

Q4: Another Way to Define a Model?

A model is the result of a trained machine that can recognize patterns using large datasets and algorithms.

Q5: How Can We Instruct a Machine?

To instruct or communicate with a machine, we use **programming languages**.
Machines only understand bits (0s and 1s), so we use programming languages like **Python** to give instructions in a way the machine can understand.

That's why we are learning Python — to communicate with the machine and build intelligent models effectively.

Topic : Small Discussion on Python Libraries:(Helps in model building)

NumPy: Used for working with numbers and arrays. It helps perform fast mathematical and logical operations on large data.

Pandas: Used for handling and analyzing data easily. It provides tables (DataFrames) to work with rows and columns like Excel.

SciPy: Built on top of NumPy. It is used for scientific and technical calculations such as integration, optimization, and statistics.

Matplotlib: Used for creating different types of charts and graphs to visualize data.

Seaborn: Built on top of Matplotlib. It makes it easier to create more attractive and detailed data visualizations.

PyTorch: A deep learning library used to build and train AI and machine learning models.

Scikit-learn (sklearn): Used for building machine learning models like classification, regression, and clustering.

TensorFlow: Another popular deep learning framework used to create and train AI models.

Keras:A high-level library that works on top of TensorFlow, making it easier and faster to build deep learning models.

Topic : Two levels of learning Python:

1. **Core Level Python Programming:** This is the basic level and is important for everyone who wants to learn Python, no matter the purpose or use case. (3 days session)

2. **Advanced Python Programming:** This level focuses on applying Python to real-world projects and specific applications.

Core Level Python Programming:

Topics Covered:
Day 1 : Variables, Data Types, Operators.
Day 2 : Programming Constructs (Conditional Statements and Loops), Functions.
Day 3 : Python Modules, and Object-Oriented Programming (OOP) Concepts.

IDE (Integrated Development Environment):
Common tools used for Python programming include Jupyter Notebooks, PyCharm, VS Code, Sublime Text, and Google Colab.

Note: Since our main focus is on model building, most of the upcoming sessions will use **Google Colab** and **VS Code** for hands-on practice.

Variables

The power of a programming language depends on the type and amount of data it can handle. Python can manage large amounts of data efficiently, which is why it is widely used for creating machine learning models.

Data can be represented using Variables. To work with variables we have to follow these steps:

Step 1: Choose a Variable Name
When choosing a variable name, keep these points in mind:

1. The name should clearly describe what the variable stores (for example, age, total_marks, student_name).
2. The name should start with a letter or underscore, not a number.
3. You can use letters, numbers, and underscores, but no spaces or special characters.
4. Variable names are case-sensitive, so score and Score are different.
5. Avoid using Python keywords like if, while, or class.
6. Keep the name short, meaningful, and easy to understand.

Step 2: Assign a Value
When assigning a value to a variable, follow these rules:

1. Use the equal sign (=) to assign a value. Example: x = 10
2. The value can be a number, text, list, or any other data type.
3. You don't need to declare the variable type; Python decides it automatically based on the value. Example: a = 5 (integer), name = "John" (string)
4. You can change the value of a variable anytime by reassigning it. Example: x = 10, later x = 20
5. You can also assign the same value to multiple variables at once.
   Example: a = b = c = 0

6. You can assign multiple values to multiple variables in one line.
   Example: x, y, z = 10, 20, 30

Syntax: Variable_name = Value

Topic : Print() Function:

The `print()` function in Python is used to **display information on the screen**.

It shows the output of your program, such as text, numbers, or the value of variables.

Example :

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

Output: Hello, World!

HandsOn Questions : Variables

1. Create a variable called `name` and store your name in it.
2. Create a variable `age` and store your age in it.
3. Create three variables: city, `country`, and `pincode` and store suitable values.
4. Change the value of a variable `score` from 50 to 80 and print it.
5. Assign the same value `100` to three variables `x`, `y`, and `z`.
6. Assign three values `10`, `20`, `30` to variables `a`, `b`, and `c` in one line.
7. Create variables of different data types (string, integer, float, boolean) and print their values.
8. Swap the values of two variables `x` and `y` without using a third variable.
9. Write a program to add two numbers using variables and print the result.
10. Write a program to store your favorite color, food, and movie in variables and print them.

Topic: Attributes of Python Variables

In Python, every variable is an object.Each object (or variable) has three main attributes:

1. **Identity** : It shows the unique location in memory where the object is stored. The identity of an object never changes during its lifetime.

   You can check it using the **id()** function.

   Example:
   x = 10

print(id(x))
Output: A unique number showing the memory address of x.

2. **Data Type (Type)** : It tells what kind of data the variable holds (for example, integer, string, float, list, etc.).The data type helps Python understand what operations can be done with that variable.

   You can check it using the **type()** function.

   Example:
   x = 10
   print(type(x))
   Output: <class 'int'> meaning x is an integer.

3. **Value**: It is the actual data stored in the variable.
   You can see it by printing the variable.The value of a variable can change, but its identity and data type depend on the new value assigned.

   Example:

   x = 10
   print(x)
   Output: 10

Key Points

- Every variable in Python refers to an object that has an identity, type, and value.
- The identity is unique for each object.
- The type defines what kind of data it holds.
- The value is the actual content stored in the variable.

HandsOn Questions : Variable Attributes

1. Create a variable `x = 15` and print its identity, type, and value using `id()`, `type()`, and `print()`.
2. Assign `name = "Venu"` and print its type and identity.
3. Create two variables `a = 10` and `b = 10`. Print their identities and check if both are the same.
4. Assign `x = 100`, print its identity, then change `x = 200` and print the new identity. See if it changes.

5. Create a list `nums = [1, 2, 3]`. Print its identity and type. Add one element using `nums.append(4)` and check if the identity changes.
6. Assign `a = 5` and `b = a`. Print both their identities. Then change `b = 6` and check if `a` is affected.
7. Create variables of different types:
   `a = 10`, `b = 10.5`, `c = "Python"`, `d = [1, 2, 3]`
   Print the identity and type of each variable.
8. Assign two identical strings:
   `str1 = "hello"`, `str2 = "hello"`
   Print their identities and see if they are the same.
9. Assign `x = 50`, print its identity and type, then change `x = "Fifty"` and print again.
10. Take user input and store it in a variable called `data`. Print the identity, type, and value of `data`.

Question 3 Create two variables `a = 10` and `b = 10`. Print their identities and check if both are the same. Help me to learn the concept below.

Topic : Python Interning

What is Interning

Interning is a memory optimization technique in Python.When Python finds that multiple variables hold the same immutable value, it may store that value only once and make all variables point to the same memory location.This helps Python save memory and improve performance.

Why Interning is Used

Python uses interning to: Save memory by reusing identical objects, Improve performance when comparing objects, Speed up access to commonly used small values.

Types of Objects That Are Interned

1. **Small Integers**

   Python automatically interns integer objects from -5 to 256.
   All variables assigned values within this range point to the same object.

   Example:

```
a = 10
b = 10
print(id(a) == id(b))  # True
```

2. S**hort Strings**

   Python automatically interns short strings that contain only letters, digits, or underscores. These are usually short (around 20 characters or fewer).
   Strings that look like identifiers (e.g., "name", "user123") are usually interned.

   Example:

   ```
   a = "hello"
   b = "hello"
   print(id(a) == id(b))  # True
   ```

3. **Common Constants and Identifiers**

   Python automatically stores commonly used string values like variable names, function names, and keywords in an internal table.This makes repeated lookups faster.

When Python Does Not Automatically Intern

Python will not automatically intern objects in the following cases:

1. Large integers (outside -5 to 256 range)
2. Long strings (more than about 20 characters)
3. Strings containing spaces, punctuation, or special characters
4. Dynamically created strings (e.g., using concatenation at runtime)

Example:

```
a = "hello world"
b = "hello world"
print(id(a) == id(b))  # False
```

Manual Interning

You can manually force interning using the `sys` module. Using sys.intern(text).

Example :

```
import sys
a = sys.intern("hello world")
```

```
b = sys.intern("hello world")
print(id(a) == id(b))  # True
```

Benefits of Interning

- Reduces memory usage by storing only one copy of a value.
- Makes equality comparison faster since identical objects share the same address.
- Improves performance in programs that process many repeated small strings.

Key Points

- Interning applies only to immutable objects (like integers and strings).
- Mutable objects such as lists or dictionaries are never interned.
- Interning happens at compile time for literals and identifiers.
- You can use `sys.intern()` to control it manually when needed.

Python Data Types

Python supports different types of data. Each datatype defines the kind of value a variable can hold and the operations that can be performed on it.

Number Types: Used to store numeric values.

1. int – Whole numbers (example: 10, -5, 100)
2. float – Decimal numbers (example: 10.5, -3.2, 0.99)
3. complex – Numbers with real and imaginary parts (example: 3+5j)

Text Type: Used to store text or string data.

str – Sequence of characters enclosed in quotes (example: "Hello", 'Python')

Boolean Type: Used to represent True or False values.

bool – Holds only two possible values: True or False

Sequence Types: Used to store multiple items in a single variable.

1. list – Ordered, changeable, allows duplicate items (example: [1, 2, 3])
2. tuple – Ordered, unchangeable, allows duplicates (example: (1, 2, 3))

Mapping Type: Used to store data in key-value pairs.

dict – Example: {"name": "Venu", "age": 25}

Set Types : Used to store multiple unique items.

1. set – Unordered collection of unique items (example: {1, 2, 3})
2. frozenset – Similar to set but cannot be changed after creation

Binary Types: Used to store binary data like files or images.

bytes, bytearray, memoryview – Example: b"Hello"

None Type: Represents the absence of a value.

None – Used to show that a variable has no value assigned (example: x = None)

Notes: Easy to memorize the types of data types

1. Fundamental Data types :
   Numerics - int, float, complex
   String (String will be both fundamental and sequential data type)

2. Sequential Data Types:
   String,
   List and
   Tuple

3. Mapping Data Types :
   Dictionaries

4. Random Data Types :
   Sets

5. Derived Data Types : (Not so important)
   Frozen Set,
   Byte/Byte Array,
   Boolean,
   None

Hands on for 1-4 is available in Github - click here

Topic : Type Casting in Python:

Type casting is the process of converting one data type into another. In Python, it allows you to **explicitly** or **implicitly** change the type of a variable

1. Implicit conversion is something which python does automatically.Happens when **no data loss** occurs during conversion. Usually occurs during **arithmetic operations** between compatible data types.
2. Explicit conversion is when users try to convert one data type to another manually using built-in functions.You **explicitly tell Python** to convert one type into another. If the conversion is not compatible, you'll get an **error**.

🧩 Python Type Conversion Compatibility Table

Data Type	✅ Can be Converted To	❌ Cannot be Converted To	Examples (Valid / Invalid)
`int`	`float`, `complex`, `str`, `bool`	`list`, `tuple`, `set`, `dict`, `bytes`, `bytearray`, `memoryview`	✅ `float(10)` → `10.0` ✅ `str(5)` → `'5'` ❌ `list(5)` → `TypeError`
`float`	`int`, `complex`, `str`, `bool`	`list`, `tuple`, `set`, `dict`, `bytes`, `bytearray`, `memoryview`	✅ `int(5.8)` → `5` ✅ `str(3.14)` → `'3.14'` ❌ `set(2.5)` → `TypeError`
`complex`	`str`, `bool`	`int`, `float`, `list`, `tuple`, `set`, `dict`, `bytes`, `bytearray`, `memoryview`	✅ `str(2+3j)` → `'(2+3j)'` ❌ `int(2+3j)` → `TypeError`
`str`	`int`, `float`, `complex`, `bool`, `list`, `tuple`, `set` *(if iterable)*	`dict` *(unless formatted as pairs)*, `bytes`, `bytearray`, `memoryview` *(without encoding)*	✅ `int("10")` → `10` ✅ `list("abc")` → `['a','b','c']` ❌ `int("abc")` → `ValueError`
`bool`	`int`, `float`, `complex`, `str`	`list`, `tuple`, `set`, `dict`, `bytes`, `bytearray`, `memoryview`	✅ `int(True)` → `1` ✅ `str(False)` → `'False'`

`list`	`tuple`, `set`, `frozenset`, `str` *(via join, if all strings)*	`int`, `float`, `complex`, `dict`, `bytes`, `bytearray`, `memoryview`	✅ `tuple([1,2,3])` → `(1,2,3)` ✅ `set([1,2,2])` → `{1,2}` ❌ `int([1,2])` → `TypeError`
`tuple`	`list`, `set`, `frozenset`, `str` *(via join, if all strings)*	`int`, `float`, `complex`, `dict`, `bytes`, `bytearray`, `memoryview`	✅ `list((1,2,3))` → `[1,2,3]` ✅ `set((1,2,2))` → `{1,2}`
`set`	`list`, `tuple`, `frozenset`, `str` *(via join, if all strings)*	`int`, `float`, `complex`, `dict`, `bytes`, `bytearray`, `memoryview`	✅ `list({1,2,3})` → `[1,2,3]` ❌ `int({1,2})` → `TypeError`
`frozenset`	`list`, `tuple`, `set`, `str` *(via join, if all strings)*	`int`, `float`, `complex`, `dict`, `bytes`, `bytearray`, `memoryview`	✅ `list(frozenset({1,2}))` → `[1,2]`
`dict`	`str`, `list`, `tuple`, `set` *(of keys)*	`int`, `float`, `complex`, `bytes`, `bytearray`, `memoryview`	✅ `list({"a":1,"b":2})` → `['a','b']` ✅ `str({"x":1})` → `"{'x':1}"`
`bytes`	`bytearray`, `memoryview`, `str` *(via decode)*	`int`, `float`, `complex`, `list`, `tuple`, `set`, `dict`	✅ `bytearray(b"Hi")` → `bytearray(b'Hi')` ✅ `b"Hi".decode()` → `'Hi'`
`bytearray`	`bytes`, `memoryview`, `str` *(via decode)*	`int`, `float`, `complex`, `list`, `tuple`, `set`, `dict`	✅ `bytes(bytearray(b"Hi"))` → `b'Hi'`
`NoneType`	`str`, `bool`	`int`, `float`, `complex`, `list`, `tuple`, `set`, `dict`, `bytes`, `bytearray`, `memoryview`	✅ `str(None)` → `'None'` ✅ `bool(None)` → `False`