Python Data Types Explained:

☑ Data Types in Python (Explained Deeply)

In Python, **data types** refer to the classification of **data items**. They define what type of **value** a variable holds and what operations can be performed on it. Python is a **dynamically typed language**, meaning you don't need to specify the data type explicitly—it automatically detects it based on the value assigned.

🔘 1. Built-in Data Types in Python

Python has the following standard or built-in data types:

- **Numeric Types** → `int`, `float`, `complex`
- **Sequence Types** → `str`, `list`, `tuple`
- **Mapping Type** → `dict`
- **Set Types** → `set`, `frozenset`
- **Boolean Type** → `bool`
- **Binary Types** → `bytes`, `bytearray`, `memoryview`
- **None Type** → `NoneType`

💡 2. Numeric Data Types

Numeric data types store **numbers**. Python has three numeric types:

◇ A) Integer (`int`)

- It stores **whole numbers** (both positive and negative) without a decimal point.
- **No size limit** for integers in Python (unlike C or Java). Can be of
- any length.

☑ Example:

```
python


x = 10          # Positive integer y = -200        # Negative integer z =
9999999999999999  # Large integer print(type(x))   # <class 'int'>
```

◇ B) Float (`float`)

- It represents **decimal numbers** or **floating-point** numbers.
- It supports **scientific notation** (e.g., `2.5e3` → `2.5 × 10³`).

☑ Example:

```
python


x = 10.5         # Decimal value
y = -4.75        # Negative float
z = 3.5e2        # Scientific notation (350.0) print(type(x))
# <class 'float'>
```

◇ **C) Complex (`complex`)** Used to represent

complex numbers.

- Complex numbers have the form `a + bj`, where:
  - `a` → Real part (float or int)
  - `b` → Imaginary part (float or int)
  - `j` → Represents √-1 ☑

Example:

```
python


x = 3 + 4j         # Complex number print(x.real)       # Real part: 3.0
print(x.imag)       # Imaginary part: 4.0 print(type(x))      # <class
'complex'>
```

🔥 3. Sequence Data Types

Sequence types are **ordered collections** of items.

◇ **A) String (`str`)**

- A string is a **sequence of characters** enclosed in **single**, **double**, or **triple quotes**.
- Strings are **immutable** (cannot be changed after creation). Supports **slicing**,
- **indexing**, and many string methods.

☑ **Example:**

```
python


# String declaration
name = "Python" msg =
'Hello, World!'
multi_line = """This is
a multi-line string"""

# String operations
print(name[0])          # First character → P
print(name[-1])         # Last character → n
print(name[0:3])        # Slice → Pyt print(name.upper())
# Uppercase → PYTHON print(type(name))       # <class
'str'>
```

◇ **B) List (`list`)**

- A list is a **mutable collection** of items. Can
- store **mixed data types**.
- Defined using **square brackets** `[ ]`.
- Supports **slicing**, **indexing**, and various methods (`append()`, `remove()`, etc.).

☑ **Example:**

```
python


# List declaration numbers = [1,
2, 3, 4, 5] mixed = [1,
"Python", 3.5, True]

# List operations
print(numbers[0])        # Access first element → 1
print(numbers[-1])       # Last element → 5
numbers.append(6)        # Add element → [1, 2, 3, 4, 5, 6] numbers.remove(3)
# Remove 3 → [1, 2, 4, 5, 6]
print(type(numbers))     # <class 'list'>
```

◇ **C) Tuple (`tuple`)** A tuple is an **immutable**

- **collection** of items.
-
- Defined using **parentheses** `( )`.

More efficient than lists for **read-only operations**.

☑ **Example:**

```
python


# Tuple declaration t =
(10, 20, 30, "Python")
print(t[0])           # First element → 10 print(t[-1])
# Last element → "Python"

# Tuple immutability
# t[1] = 50           # ❌ Error: cannot modify tuple print(type(t))
# <class 'tuple'>
```

🛠 4. Mapping Data Type

◇ **Dictionary (`dict`)**

- A dictionary is a **collection of key-value pairs**.
- Defined using **curly braces** `{}`.
- Keys must be **unique** and **immutable** (like strings or numbers). Values can be
- **any type**.

☑ **Example:**

```
python


# Dictionary declaration
student = {
```

```
    "name": "Alice",
    "age": 22,
    "grade": "A"
}

# Accessing dictionary values print(student["name"])
# Alice print(student.get("age"))      # 22

# Adding a new key-value pair
student["city"] = "New York" print(student)

# Deleting a key-value pair
del student["grade"]
print(student)
```

---

5. Set Data Types

◇ A) Set (`set`)

- A set is an **unordered collection** of **unique elements**.
- Defined using **curly braces** `{}`.
- Supports operations like **union**, **intersection**, and **difference**.

☑ Example:

```
python


# Set declaration
fruits = {"apple", "banana", "cherry", "apple"}

# Sets contain unique items
print(fruits)            # {'apple', 'banana', 'cherry'}

# Adding elements fruits.add("orange")

# Removing elements fruits.remove("banana")

# Set operations
A = {1, 2, 3}
B = {3, 4, 5}
print(A.union(B))        # {1, 2, 3, 4, 5} print(A.intersection(B))
# {3}
```

◇ B) Frozen Set (`frozenset`)

- Similar to `set` but **immutable**.
- You cannot add or remove elements after creation.

☑ Example:

```
python


fs = frozenset([1, 2, 3, 4])
print(fs)               # frozenset({1, 2, 3, 4}) # fs.add(5)             # ❌ Error: cannot modify
frozenset
```

☑ 6. Boolean Data Type

- The boolean data type has only two values: `True` and `False`.
- Internally, `True` is equivalent to `1` and `False` is equivalent to `0`.

☑ Example:

```
python


x = True
y = False
print(x + 5)         # 6 → True is treated as 1 print(y
+ 5)         # 5 → False is treated as 0 print(type(x))
# <class 'bool'>
```

🔥 7. Binary Data Types

◇ A) Bytes (`bytes`)

- **Immutable** sequence of bytes.
- Used for **binary data** (e.g., images, audio).

☑ Example:

```
python


b = b"Hello"         # Bytes object print(b)             # b'Hello'
```

◇ B) Bytearray (`bytearray`)

- **Mutable** version of bytes.
- Used to modify binary data.

☑ Example:

```
python


ba = bytearray([65, 66, 67]) ba[0]
= 68
print(ba)            # bytearray(b'DBC')
```

🔥 8. None Type

- Represents the **absence of a value**.
- Used to **initialize** variables with no value.

☑ Example:

```
python
```

```python
x = None
print(x)              # None
print(type(x))        # <class 'NoneType'>
```

☑ 💡 Key Takeaway

Python offers a variety of **data types** that make it powerful and flexible. Understanding them is essential for **efficient programming**. 🚀