**Classification of Python Data Types**

**1. Numeric Data Types**

* int: Represents integer numbers (e.g., 10, -5, 0)
* float: Represents floating-point numbers (e.g., 3.14, -2.5)
* complex: Represents complex numbers (e.g., 2+3j)

**2. Sequence Data Types**

* str: Represents sequences of characters (e.g., "Hello, world!")
* list: Represents ordered collections of items, mutable (e.g., [1, 2, 3, "apple"])
* tuple: Represents ordered collections of items, immutable (e.g., (1, 2, 3, "apple"))
* range: Represents a sequence of numbers (e.g., range(10))

**3. Mapping Data Type**

* dict: Represents key-value pairs (e.g., {"name": "Alice", "age": 30})

**4. Set Data Type**

* set: Represents unordered collections of unique items (e.g., {1, 2, 3})
* frozenset: Represents unordered collections of unique items, immutable (e.g., frozenset({1, 2, 3}))

**5. Boolean Data Type**

* bool: Represents Boolean values (True or False)

**6. Binary Data Types**

* bytes: Represents a sequence of bytes
* bytearray: Represents a mutable sequence of bytes
* memoryview: Represents a memory view of an object, often used for efficient memory access

**7. None**

**8.User Defined Datatype**

**1. Numeric Data Types**

Numeric data types in Python are used to represent numbers. They come in three main types:

**1. Integer (int):**

* Represents whole numbers without a decimal point.
* Can be positive, negative, or zero.
* Examples: 10, -5, 0

**2. Floating-point (float):**

* Represents real numbers with a decimal point.
* Used for calculations involving fractions or decimals.
* Examples: 3.14, -2.5, 1.0

**3. Complex (complex):**

* Represents complex numbers with a real and imaginary part.
* Used in advanced mathematical calculations.
* Examples: 2+3j, -1-5j

**Example:**

**#1. Numeric Data Types  
x = 10 # Integer  
y = 3.14 # Floating-point  
z = 2 + 3j # Complex  
  
print(type(x)) # Output: <class 'int'>  
print(type(y)) # Output: <class 'float'>  
print(type(z)) # Output: <class 'complex'>**

**2. Sequence Data Types**

Sequence data types in Python are used to store collections of items in a specific order. They provide a way to organize and access multiple values efficiently. There are three primary sequence data types in Python:

1. Strings (str):

* A sequence of characters enclosed in single quotes (' ') or double quotes (" ").
* Immutable: Once created, the characters within a string cannot be changed.
* Examples: 'Hello, world!', "Python is fun"

**2. Lists (list):**

* An ordered collection of items enclosed in square brackets [ ].
* Mutable: You can add, remove, or modify elements within a list.
* Can store items of different data types.
* Examples: [1, 2, 3], ['apple', 'banana', 'cherry'], [1, 'hello', 3.14]

**3. Tuples (tuple):**

* An ordered collection of items enclosed in parentheses ( ).
* Immutable: Like strings, tuples cannot be modified after creation.
* Often used to represent fixed data sets.
* Examples: (1, 2, 3), ('apple', 'banana', 'cherry')`

**Example:**

**my\_string = "Hello"**

**print(my\_string[1]) # Output: 'e'**

**my\_list = [1, 2, 3, "apple"]**

**print(my\_list[2]) # Output: 3**

**my\_tuple = (10, 20, 30)**

**print(my\_tuple[0]) # Output: 10**

**my\_list[1] = 5 # Modify an element in the list**

**print(my\_list) # Output: [1, 5, 3, 'apple']**

**3. Mapping Data Type**

In Python, the primary mapping data type is the dictionary. A dictionary is an unordered collection of key-value pairs. Each key is unique, and it's used to access the corresponding value**.**

* **Unordered:** Elements in a dictionary don't have a specific order.
* **Mutable:** You can add, remove, or modify key-value pairs after the dictionary is created.
* **Keys:** Keys must be unique and immutable (e.g., strings, numbers, tuples).
* **Values:** Values can be of any data type, including other dictionaries.
* **Range:**Represents a sequence of numbers, commonly used in loops for iteration. Iterating over a sequence of numbers in loops, such as generating a series of values or indexing elements.

**Example:**

for i in range(5):

  print(i)

**Mapping Type:**

**Dictionary**

A collection of key-value pairs, where each key is unique and maps to a value.Used when Storing data that needs to be accessed via a key, like user information, product details, or configuration settings.

**Example:**

dic={1:"Archana",2:"Rachana",3:"Chaithu"}  
print(dic)  
print(type(dic))  
print(dic.keys())  
print(dic.values()

**4. Set Data Type**

**1.Set:**

A set is a built-in data type in Python that stores an unordered collection of unique elements. Sets are mutable, meaning their content can be changed after the set is created (e.g., elements can be added or removed).Sets are useful when you need to store distinct items and do not care about their order.

**Example:**

mset = {1, 2, 3, 4, 5}

print(mset)

mset.add(6)

print(mset)

**2.Frozenset:**

A frozenset is similar to a set in that it stores an unordered collection of unique elements, but it is immutable. Once a frozenset is created, you cannot add or remove elements from it.

**Creating a frozenset**

You can create a frozenset using the frozenset () function:

**Example**:

fs = frozenset([1, 2, 3, 4])

print(fs)

print(type(fs))

**5.Boolean Type**

The Boolean data type in Python is used to represent one of two values: **True** or **False**. It is used in logical operations, conditions, and control structures to make decisions or evaluate expressions.

Example:

a = 10

b = 20

result = a < b

print(result) #False

**6.Binary Type**

Binary types in python are data types used to handle binary data, such as images ,files, or other data stored in binary form.

**Binary Types in Python**

**1.Bytes:**

* Immutable sequence of bytes.
* Used to store binary data.
* Values range from 0 to 255..
* Created using b or B as prefix.

**Example**: data = b"hello" # bytes literal

print(type(data)) # <class 'bytes'>

**2.Bytearray**

* Mutable sequence of bytes.
* Similar to bytes, but can be modified in place.
* Useful when you need to work with binary data that requires modification.

**Example**:

ba = bytearray(b"Archana")  
ba[0] = 67  
print(ba) #bytearray(b'Crchana')

**Memoryview**

A memory-efficient way to access binary data without copying it.

Allows slicing and manipulation of the underlying binary data.

**Example**:

#Memoryview  
b\_data = b"Rachana"  
# Create a memoryview  
mv = memoryview(b\_data)  
# Access the first element  
print(mv[0]) #82

**7.None Type**

The **None** data type in Python is a special constant that represents the absence of a value or a null value.It indicate that a variable has no value or that a function does not return anything explicitly.

**Example1:**

def my\_function():

pass

result = my\_function()

print(result) # Output: None

**Example2:**

if not None:

print("None is False") # Output: None is False

**8.User Defined Function**

* User-defined data types are created by the user to fit specific needs in a program.
* They allow users to define new types of data that are more complex than the built-in types like integers, strings, and lists.
* These custom data types can encapsulate both data and behavior, which is essential for modeling real-world entities or solving complex problems in a program.

Types

1.Class

2.Enumerations

3.User Defined Functions

4.Custom Data Structures