* **Other built-in types**

Besides the **core built-in types** (like int, float, str, list, dict, etc.), Python includes **other built-in types** that are more advanced or used in specific contexts. These are still part of the Python language and are available without needing to import additional libraries.

1. Binary Types in Python

Binary types are used to store and manipulate **binary data** (such as files, images, or raw network streams). Python provides three main binary types:

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Mutable | Description | Example |
| bytes | No | Immutable sequence of bytes | b'hello' |
| bytearray | Yes | Mutable sequence of bytes | bytearray(b'hello') |
| memoryview | Depends | View into another binary object’s memory | memoryview(b'abc') |

## **bytes**

Bytes are an **immutable** sequence of bytes,Each byte is a number between 0 and 255.

Commonly used when reading binary files, working with byte-oriented protocols, etc.

* Similar to a string, but contains raw 8-bit values.
* Represented with a prefix b or B before the string.

### Example:

b = b'hello'

print(b[0])

print(type(b))

output:

104

<class 'bytes'>

## **bytearray**

Bytearray A **mutable** sequence of bytes and Allows modification of byte content.

Useful when you need to change binary data.

* Can be created from bytes, lists of integers, or a string (with encoding).

### Example:

ba = bytearray(b'hello')

ba[0] = 72

print(ba)

output:

bytearray(b'Hello')

## **memoryview**

memoryview Provides a memory-efficient **view** into the data of a binary object like bytes or bytearray and Allows access to parts of the data **without copying**. These Useful for large datasets and performance-critical applications.

* Acts like a "window" on the data.
* Can read and write (if the original object is mutable).

### Example:

data = bytearray(b'hello')

mv = memoryview(data)

print(mv)

mv[0] = 72

print(data)

output:

<memory at 0x000001EB0C437940>

bytearray(b'Hello')

1. Special Types in Python

Python provides some **special types** that are not commonly used in basic programming but are **essential in advanced applications**, such as operator overloading, placeholder code, class design, or memory optimization.

## **NoneType**

The type of the **None** object. It is Represents the **absence of a value** or a **null value**.

Often used as a **default return value** of functions that don’t return anything.

### Example:

print(type(None))

output:

<class 'NoneType'>

## **type**

A special type that represents the **type of all objects** and Can also be used to **dynamically create classes**.

### Example:

print(type(5))

output:

<class 'int'>

## **object**

The **base class of all classes** in Python. And All user-defined and built-in classes derive from object.

class A:

    pass

print(isinstance(A(), object))

output:

True

## **slice**

Represents a **slice object**, used in list/string slicing operations. And Can be created manually using slice(start, stop, step).

### Example:

s = slice(1, 5, 2)

print([0, 1, 2, 3, 4, 5][s])

output:

[1, 3]