**Python**

**Complex data type**

A complex number consists of a real and an imaginary part, written as a + bj, where a is the real part, b is the imaginary part, and j represents the square root of -1.

In Python, complex numbers are represented using the complex( ) function or by directly writing them as (a + bj)

We can access the real part of a complex number using .real and the imaginary part using .imag.

z = 3 + 4j

real part = z.real (Output: 3.0) imag\_part = z.imag (Output: 4.0)

We can do the basic arithmetic operations like addition, subtraction, division and multiplication of complex number.

We can also find the conjugate (flips the sign of the imaginary part) and magnitude (absolute value of a complex number) of the complex number as z.conjugate() and abs(z).

**Bool data type**

The bool data type in Python represents boolean values. It can only take two possible values: True and False.

True is denoted as positive and also known as 1, and False is denoted as negative and also known as 0.

False values: 0, None, [], '', {} (empty collections or zero values) denotes False

True values: Non-zero numbers, non-empty strings, and collections denotes True.

We can also perform arithmetic operation such as add, subtract and multiply on these values as:

A= True B= False print(A+B) (output= 1)

**String data type(str)**

In Python, the **str** data type represents a sequence of characters (a string). Strings are used to store and manipulate text.

* We can create strings by enclosing characters in either single quotes '...', double quotes "...", or triple quotes '''...''' , """..."""(document string). There is no char data type in python as in C.
* Strings are indexed, so that we can access individual characters using square brackets [ ] and zero-based indexing.
* We can also concatenate the string as A= “hello” + “world” print(A) (output= helloworld)
* We can multiply the string as we want as a= “#”\*3 print(a) (output= ###)
* We can get the length of the string as :len("bhaskar") (output = 7)

**Fundamental data types Vs Immutability**

All fundamental data types are immutable. Once we create an object, we cannot perform any changes in that object. If we try to change, then with those changes a new object is created.

This concept is known as immutability of data type. When performing operations on an integer, float, or string, a new object is created rather than modifying the original one. This immutability ensures data consistency, preventing accidental changes, and enhances performance by allowing Python to optimize memory usage for small and frequently used objects like integers and strings.

For example let’s suppose an integer variable a=10, it has some space on the memory. When the operation (a+2) is done then it will not be in the same memory location, a new object is created. It also ensure that when a same value numerical object is initialized then it will have the same id. For eg when a=10 and b=10 then the id of 10 will be same just a and b are pointing towards the same id.