**Classification of Python Data Types**

Python is a high-level, interpreted programming language known for its easy-to-read syntax . It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. It is widely used for web development, data analysis, machine learning, data science, Desktop GUI applications and more.

**Key Features of Python:**

1. **Simple and Easy to Learn:**  
    Python's syntax is clear and easy to understand, making it beginner-friendly.
2. **Interpreted Language:**  
    Python is executed line-by-line, which simplifies debugging and development.
3. **Dynamically Typed:**  
    You don't need to explicitly declare variable types. The type is inferred during runtime.
4. **Cross-Platform:**  
    Python code runs on different operating systems, including Windows, macOS, and Linux, without modification.

**Introduction to Data Types**

In Python, a data typ**e** specifies the type of data that a variable can hold. Python is a dynamically typed language, meaning you don't need to declare the datatype explicitly when defining a variable; the interpreter determines it based on the value assigned. The following are the standard or built-in data types in Python:

* **Numeric** – int, float, complex
* **Sequence Type** – string, list, tuple
* **Mapping Type** – Dictionary
* **Boolean** – bool
* **Set Type** – set, frozen set
* **Binary Types** – bytes, memory view, byte array
* **None**
* **User Defined Datatype**

1.**Numeric Datatype**

The numeric data type in Python represents the data that has a numeric value. A numeric value can be an integer, a floating number, or even a complex number.

**Integers** – This value is represented by int class. It contains positive or negative whole numbers (without fractions or decimals). In Python, there is no limit to how long an integer value can be.

**Example**:a=10

* **Float** – This value is represented by the float class. It is a real number with a floating-point representation. It is specified by a decimal point. Character e or E followed by a positive or negative integer may be appended to specify scientific notation.

**Example**:3.14

* **Complex Numbers** – A complex number is represented by a complex class. It is specified as *(real part) + (imaginary part)j* .

**Example**:– 2+3j

Example:

a=10

b=12.3

d=123e3

c=1+2j

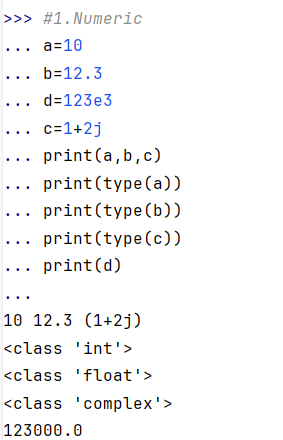
print(a,b,c)

print(type(a))

print(type(b))

print(type(c))

print(d)



**Type Conversion**

Type conversion refers to the process of converting one data type into another. Python provides mechanisms to convert between different data types, which can be either implicit or explicit.

**Implicit Type Conversion**

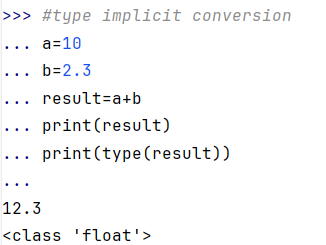
Python automatically converts a smaller data type to a larger data type to prevent data loss.

x = 5 # Integer

y = 2.5 # Float

result=x+y

print(type(result)



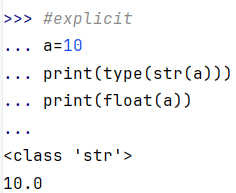
**Explicit Type Conversion**

* Also known as **typecasting**.
* It is done using predefined functions like int(),float(),etc.

**Example**:

num = 10

print(float(num)) # Output: 10.0



**2.Sequence type**

The sequence Data Type in Python is the ordered collection of similar or different Python data types. Sequences allow storing of multiple values in an organized and efficient fashion.

There are several sequence data types:

**1.String:**A string is a collection of one or more characters put in a single quote, double-quote, or triple-quote. It is represented by str class.

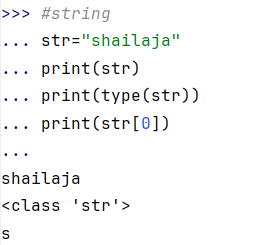
**Example:** **#string**

str="shailaja"

print(str)

print(type(str))

print(str[0])



**2.List:** An ordered collection of items, which can be of any data type, and is mutable (can be changed).The items stored in the list are separated with a comma (,) and enclosed within square brackets [].

**Example:**

#List

list=[1,2.3,1+2j,"shailaja"]

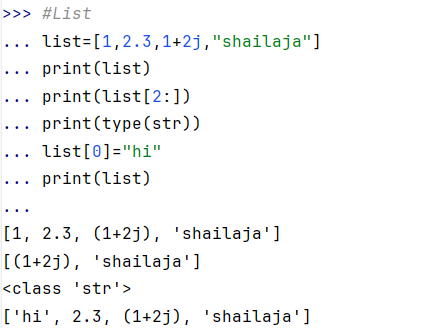
print(list)

print(list[2:])

print(type(str))

list[0]="hi"

print(list)



**3.Tuple**:A tuple is similar to the list in many ways. Like lists, tuples also contain the collection of the items of different data types. The items of the tuple are separated with a comma (,) and enclosed in parentheses ().A tuple is a read-only data structure as we can't modify the size and value of the items of a tuple.

**Example**:

tuple=(1,2,3,4.3,3+2j)

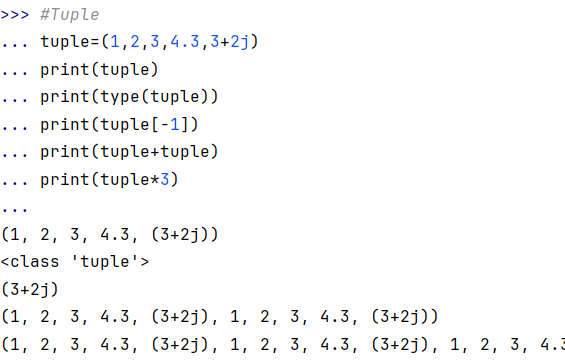
print(tuple)

print(type(tuple))

print(tuple[-1])

print(tuple+tuple)

print(tuple\*3)

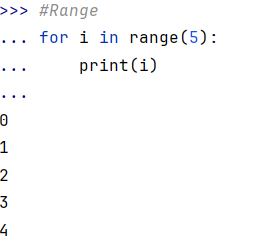


**4.Range:**Represents a sequence of numbers, commonly used in loops for iteration. Iterating over a sequence of numbers in loops, such as generating a series of values or indexing elements.

**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**:

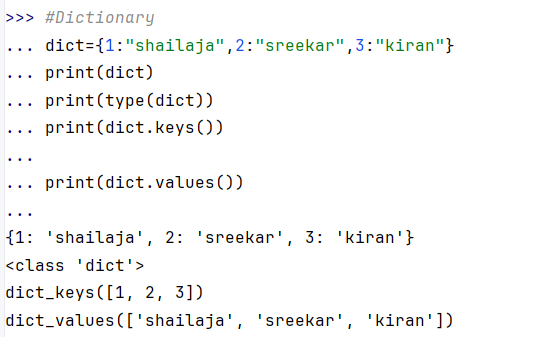
dic={1:’shailaja’,2:’sreekar’,3:’kiran’}

print(dic)

print(type(dic))

print(dic.keys())

print(dic.values())



4**.Set Types**

**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.

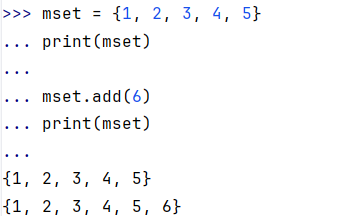
**Example:**

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**

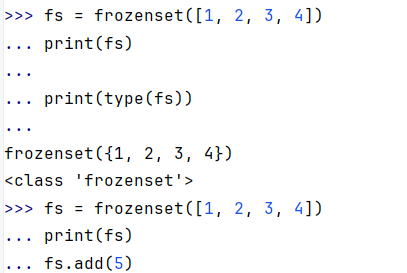
You can create a frozenset using the frozenset () function:

**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.

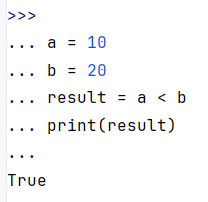
Example:

a = 10

b = 20

result = a < b

print(result) # True



**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.

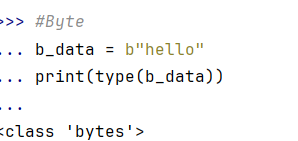
**Binary Types in Python**

**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**

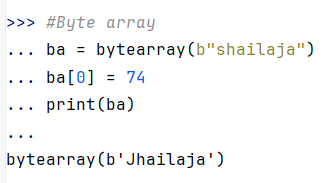
* Mutable sequence of bytes.
* Similar to bytes, but can be modified in place.
* Useful when you need to work with binary data that requires modification.

**Example**:

ba = bytearray(b"shailaja")

ba[0] = 74

print(ba) #Jhailaja



**Memoryview**

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

Allows slicing and manipulation of the underlying binary data.

**Example**:

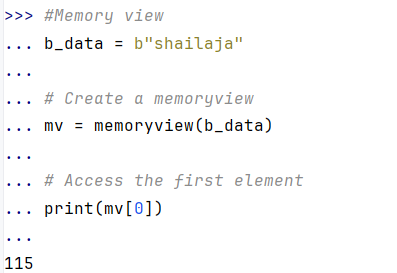
b\_data = b”Shailaja"

# Create a memoryview

mv = memoryview(b\_data)

# Access the first element

print(mv[0]) #115



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.

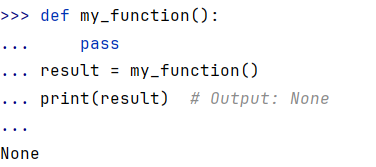
**Example1:**

def my\_function():

pass

result = my\_function()

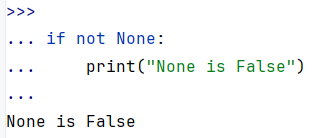
print(result) # Output: None



**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