WELCOME

Python  Series

AGENDA

Data Types

Python  Series

DATA TYPES

- Data types define the type of a variable, indicating the kind of data stored in memory.

- To know the type of data type, use **type() or isinstance(variable_name, datatype)**

Python  Series

TYPES OF DATA TYPES

Data Type
Numeric
String
Sequence
Mapping
Boolean
Set
Binary
None

Python Series

NUMERIC DATA TYPES

Types:

- **Int**: Represent whole number.
- **Float**: Represent decimal number.
- **Complex**: consists of real and imaginary number.

Features:

- **Int:** can grow as big as available memory allows
- **Float:** Precision is limited by system but usually up to 15 decimal places

Use Case:

- **Int**: indexing, counting, use of exact result
- **Float**: measurement, scientific calculation, use of precise result
- **Complex**: advanced calculation, physics calculation, engineering

STRING DATA TYPES

Overview:

- Sequence of Unicode characters enclosed in single quotes (' ') or double quotes (" ") or triple quote (''' ''')
- String is immutable, meaning, once created, content cannot be changed.

Key Features:

- Accessibility
- Concatenation
- Repetition
- String methods

Use Cases:

- User Input
- Text Processing
- File I/O

Python  Series

SEQUENCE DATA TYPES

- Sequence data types in Python are a category of data types that **store collections of items**, ordered by their position.
- Each item in a sequence is assigned a number - its position or **index**.
- The first index is zero, the second index is one, and so forth.

Common Sequence Data Types:

- **List**: Mutable sequence of items.
- **Tuple**: Immutable sequence of items.
- **Range**: mutable sequence of numbers.

Use Case:

- **List**: where order matters and might change, eg: product items.
- **Tuple**: where order matters but does not change, eg: coordinates
- **Range**: For efficient looping with numbers.

We will discuss them in detail in their respective units.

Python  Series

MAPPING DATA TYPES

- **Mapping** refers to the concept of associating or linking a set of keys to the set of values.

- This concept is implemented using **`dict`** (dictionary) data type, which is a collection of **key : value pair**.

Use Cases:

- Storing and retrieving data based on unique keys
- Grouping data together in structured manner.
- Counting items (keys for items and values for counts)

- More on dict in its specific unit (later)

BOOLEAN DATA TYPES

- A fundamental data type that takes 2 values only: **True or False**.

Characteristics:

- **Binary Values**: 'True' and 'False', essentially 1 and 0.
- **Derived from integers**: It can participate in arithmetic operations as '1' and '0'. Eg: True + 5 will give 6 as (1 + 5).
- **Automatic Evaluation of non-Boolean object**: In case of 'if' and 'while' condition.

Common Use Case:

- Conditional Statements
- Loop Control
- Logical Operations

Creation:

- Direct Assignment: a = True
- Result from comparison: 5 > 2 (result True)

BOOLEAN DATA TYPES

What are Truthy and Falsy Values?

- **Truthy** and **Falsy** are concepts used to evaluate the **truthiness or falseness** of an object in Boolean context, such as conditional statements or loops.

- While Python has 2 explicit Boolean values (True and False), all values are inherently "truthy" or "falsy", meaning that they can be evaluated as 'True' or 'False' in context that require a Boolean result.

Falsy Value:

- Certain objects in python are considered 'Falsy' (False).
- The falsy values are:
  - **None**: A singleton object represent absence of value.
  - **Zero in any numeric type**: 0, 0.0, 0j
  - **Empty sequence or collection**: '' "" (empty string), '( )' empty tuple, '[ ]' empty list, '{ }' empty dictionary, ' set() ' empty set.
  - **Instance of custom class**: __bool__() if it returns False or __len__() if it returns 0.

BOOLEAN DATA TYPES

Truthy Values:

- Exact oppotsite of Falsy, meaning truthy are evaluated as true in Boolean context.

Truthy Values are:

- Non zero numbers
- Non empty strings
- Non empty lists, tuples, sets, dictionaries.

Use bool() to explicitly evaluate truthiness of values

- Example:
  - Bool(None) -> False
  - Bool(0) -> False
  - Bool("") -> False
  - Bool('Python') -> True
  - Bool([1,2,3]) -> True

Python  Series

SET DATA TYPES

Sets:

- A **set** is a mutable collection of distinct (unique) immutable objects, just like the sets in mathematics.

Characteristics:

- **Unique**: Automatically removes duplicate elements.
- **Mutability**: Set is mutable but its elements are immutable (number, string, tuple)
- **Unordered**: Elements do not have fixed order, hence no indexing or slicing.

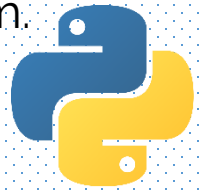
frozenset:

- Similar to set except it is immutable, meaning once it is created, it cannot be altered (no adding or removing elements).

- Can be created using **frozenset()** constructor.

Use Case:

- **Sets**: When you need collection of unique elements which will change over time.
- **frozensets**: When you want your set to remain constant throughout the program.

Python  Series

BINARY DATA TYPES

- **A way of representing series of binary digits (0s and 1s).**
- They are used for dealing with files or image or anything that can be represented using only binary digits.
- Python has 3 ways to represent binary data:
- **Bytes:**
  - **Immutable sequence**: Represent data as sequence of bytes (Integer between 0-255)
  - **Usage**: Ideal for storing binary data files like images, files and network packets.
  - **Creation**:
    - Literal notation with prefix **`b`** as **b"Hello"**
    - Using **byte()** function with sequence of integers as **bytes([65,66])**

- **Bytearray:**
  - **Mutable Sequence**: Similar to byte, but allow modification.
  - **Usage**: Useful for binary data that needs to be changed or updated.
  - **Creation**:
    - **bytearray([65,66])**
    - **bytearray("hello","utf-8")**

Python  Series

BINARY DATA TYPES

- **Memoryview:**

- **Efficient Data Access**: Provide memory-efficient view into another object's buffer, without copying data.
- **Usage**: Manipulating large datasets or binary data efficiently
- **Creation**:
  - Directly from a bytes or bytearray using **`memoryview()`**
  - memoryview(bytearray("Hello"))

- **Key Point:**
  - Use **`byte`** for **immutable data**, **`bytearray`** for **mutable data** and **`memoryview`** for **efficient access and manipulation**

NONE DATA TYPES

- None represent absence of value or a null value.
- It is a special object of **NoneType**.
- There is only one None value in python.

Key Characteristics:

- **Represent Null**: It signifies absence of value or null state.
- **Singleton**: Only one instance of `None` exists, means all reference to None will point to same memory address.
- **Comparison**: None is **NOT** equivalent to "False" or "0", an empty string or other falsey values in python.

Use Cases:

- Default argument in function
- Optional Return value
- Initialization

Python  Series

TYPES OF DATA TYPES

Data Type	Examples
Numeric	int, float, complex
String	str (text sequence type)
Sequence	list, tuple, range
Mapping	dict
Boolean	bool
Set	set, frozenset
Binary	bytes, bytearray, memoryview
None	NoneType

Python  Series

NEXT VIDEO...

TYPE CASTING

Python  Series