# Python Data Types:

## Numeric Type:

### Integers (int):

* Represents whole numbers, both positive and negative
* No fractional or decimal parts
* Example: 5, -12, 100, 50, 104

### What builtin functions can I use with Integers?

* abs(): Returns the absolute value of the integer

print (abs(-10)) # Output = 10

* pow(): Raises an integer to the power of another number. It also allows modulo operations as an optional third argument.

print (pow(2,3)) # Output: 8

print (pow(2,3,5) # Output: 3 #2^3 % 5

* divmod(): Returns a tuple containing the quotient and the remainder of the integer division.

print(divmod(10,3)) # Output: (3,1)

* round(): Rounds an integer (or float) to a specified number of decimal places. Its useful when integers are mixed with other types during calculations.

print(round(5.98) # Output: 6

* bin(): converts an Integer to its binary representation

print(bin(10)) # Output: 0b1010

* hex(): Converts an integer to its hexadecimal representation

print(hex(255)) # Output: ‘0xff’

* oct(): Converts an integer to its Octadecimal representation

print(oct(8)) # Output: ‘0010’

* isinstance: Checks if the variable belongs to the int class

print(isinstance(42,int)) # Output: True

### Floating-Point Numbers (float):

* Represents real numbers that can have decimal points
* Can also be expressed using scientific notations (e.g. 1.2e3 for 1200)
* Example: 5.74, 90.12, -5.3, 1.34e4

### Complex Numbers (complex):

* Represents numbers with a real and imaginary part
* Written in the form a + bj, where a is the real part and b is the imaginary part
* Example: 3+4j, 1-2j

Each of these numeric types is a class in Python, and variables of these types are instances of their respective classes. You can use the **type()** function to check the data type of a value. For instance:

a = 10 # Integer

b = 3.14 # Float

c = 1 + 2j # Complex

print(type(a)) # Output: <class 'int'>

print(type(b)) # Output: <class 'float'>

print(type(c)) # Output: <class 'complex'>

## Sequence Type:

Sequence data types in Python are used to store collections of items in an ordered manner, allowing for efficient organization and retrieval of elements. Here are the main sequence data types

### String (str):

* Strings are arrays of Unicode characters, used for text data
* Immutable, meaning you cannot change a string after creation
* Example: s = “Hello World”

#### Key characteristics of Strings in Python

1. **Immutable**
   1. Strings cannot be changed after they are created. Any modification creates a new string

Example: s = “Hello World”

s = s + “ Of Python” # Creates a new string

print (s) # Output: Hello World Of Python

1. **Single or Double Quotes**
   1. Strings can be created using either single or double quotes
2. **Triple Quotes for multi-line strings**
   1. Strings spanning multiple lines can be created using triple quotes (‘’’ or “””)
3. **Accessing Characters:**
   1. Strings are indexed and support slicing. Index starts at 0
   2. Negative index starts from the end. (-1 is the last character of the string)

Example:

S = “Python”

S[0] # Output: P

S[-1] # Output: n

### Common String Methods

* **.lower() :** Converts string to a lowercase. Example: “Hello”.lower() # Output: hello
* **.upper():** Converts string to uppercase. Example: “Hello”.upper() # Output : HELLO
* **.strip():** Removes leading or trailing white spaces or characters. Example: “ Hello “.strip -> Output: Hello
* **.split():** Splits string into lists based on the delimiter (default: Space). Example: “a,b,c”.split(“,”) -> # Output: [‘a’, ’ b’, ’ c’]
* **.join():** Joins elements of a list into a single string. Example: “,”.join([‘a’, ‘b’, ‘c’]) -> #Output: abc
* **.replace(old,new):** Replaces occurrences of a substring with another substring. Example: “Python”.replace(“Py”,”Cy”) -> # Output: Cython
* **.startswith():** Checks if a string starts with a given substring. Example: “Python”.startswith(“Py”) -> # Output: True
* **.endswith():** Check if a string ends with a given substring.

Example: “Python”.endswith(“on”) - > True

### String Formatting

1. **Using f-strings (Python 3.6+):**
   1. Embed variables directly into string using {}

name = “Allen”

print (f“Name is {name}”) # Output: Name is Allen

1. **Using format():**
   1. Another way to insert variables into strings

print (“I love {}”.format(“Python”)) # Output: I love Python

1. **Old-style(% Operator):**
   1. Still supported but less commonly used

print (“I scored %d out of %d” % (95, 100)) # Output: I scored 95 out of 100

### Escape Characters

Special Characters can be included using escape sequences:

* \n : Newline
* \t : Tab
* \\ : Backslash
* \’ : Single quote
* \” : Double quote

### List (list):

* A mutable collection that can hold items of various data types
* Allows insertion, deletion, and modification of elements
* Example: l = [1, “hundred”, 5.9, “7”]

### Tuple (tuple):

* Similar to lists, but immutable – values cannot be altered after creation
* Useful for storing fixed data
* Example: t = (1, “two”, 3/0)

**Common Features:**

All sequence types support slicing and indexing to access individual items or ranges of items.

You can iterate over sequences using loops

Example: l = [1,2,3,4]

Print(l[1:3]) # output [2,3]

## Mapping Type:

In Python, the Mapping Data type is represented by dictionaries, which are an unordered collection of key-value pairs. Here’s a deeper look at it:

### Dictionary (dict):

* A dictionary maps key to values. Each key is unique and immutable, while the values can be of any data type and are not required to be unique.
* Dictionaries are mutable, meaning you can add, remove, or update key-value pairs

#### Creation of Dictionary

Dictionaries can be created using curly braces {} or the dict() constructor:

Example:

# Using Curly Braces

my\_dict = {‘name’: ‘Alice’, ‘age’ : 23, ‘Place’: ‘Bangalore’}

# Using dict contructor

my\_dict = dict(name=”Bob”, Age=23)

#### Accessing Element

You can access values in a dictionary by their keys:

print (my\_dict[‘name’]) # output: Bob

Alternatively you can use get() method to avoid errors if a key does not exist

print(my\_dict.get(‘age’)) # Output: 23

print(mydict.get(‘name’,’age’,’place’) # Output: Alice, 23, Bangalore

#### Updating Dictionary

You can add, update or remove key-value pairs

# Adding a new key-value pair

my\_dict[‘country’] = ‘India’

# Updating an existing key

my\_dict['name’] = ‘Allen’

# Removing key-value pair

del my\_dict[‘country’]

### Key Features

* Keys must be unique and immutable (e.g., strings, numbers or tuples)
* Values can be of any data type
* Keys are case-sensitive (‘Name’ and ‘name’ are different)

### Common Methods

.keys(): Returns all keys in the dictionary

My\_dict.keys() -> [‘name’, ‘age’]

.values(): Returns all values in the dictionary

My\_dict.values() - > [‘Alice’, 23]

.items(): Returns key-value pairs as tuples

My\_dict.items() -> [(‘name’ ,’Alice’, ‘age’,23)]

.pop(key): Removes a key and returns its value

My\_dict.pop(‘age’) -> 23

.update(other\_dict): Updates the dictionary with another

My\_dict.update(‘Country’: ‘USA’)

Boolean Type:

Bool

Set Type:

Set

Frozenset

Binary Type:

Bytes

Bytearray

Memoryview