UNIT-2

List: Introduction to List, List Operations, Traversing a List, List Methods and Built in Functions. **Tuples and Dictionaries:** Introduction to Tuples, Tuple Operations, Tuple Methods and Built in Functions, Nested Tuples. Introduction to Dictionaries, Dictionaries are Mutable, Dictionary Operations, Traversing a Dictionary, Dictionary Methods and Built-in functions.

List in python:-

a list in python is a built-in data structure used to store a collection of elements in a single variables.

lists are ordered, mutable, and allow duplicate values. the elements of a list can be of different data types such as integers, strings, floats and Booleans. lists are created using square brackets [] and elements are separated by commas.

→ python list supports indexing and slicing, which helps in accessing individual elements or a group of elements. since lists are mutable, their contents can be modified after creation.

how to create a python list:-

the list can be created using either the list constructor or using square brackets [].

1. using list() constructor:-
2. using square brackets [].

Ex:-

```
my_list=list((1,2,3,4))
print(my_list, type(my_list))
```

Ex:-

```
my_list=[10,"raja",99.63,True]
print(my_list, type(my_list))
```

Properties of a List or key features of lists:-

1. ordered collection:-

lists maintain the order in which elements are inserted. each element has a fixed position called an index, starting from 0.

Ex:-

```
colors = ["red", "green", "blue"]
print(colors[0])   # red
print(colors[1])   # green
print(colors[2])   # blue
```

2. Mutable (changeable):-

lists are mutable, meaning their elements can be modified, added, or removed after creation.

Ex:-

```
numbers = [1, 2, 3]
numbers[1] = 20
print(numbers)
```

3. Allows Duplicate Elements:-

a list contain duplicate values. This means the same value can appear more than once in a list.

Ex:-

```
data = [10, 20, 10, 30, 20]
print(data)
```

4. can store different data types :-

a single list can store multiple data types such as integers, strings, floats, and Booleans.

Ex:-

```
mixed_list = [1, "Python", 3.14, True]
print(mixed_list)
```

5. supporting indexing and slicing:-

indexing:-

used to access a single element.

Ex:-

```
marks = [70, 80, 90]
print(marks[2])
```

slicing:-

used to access a range of elements.

Ex:-

```
marks = [70, 80, 90, 85, 75]
print(marks[1:4])
```

Ordered and Unordered in Python:-

In Python, ordered means elements are stored in a fixed sequence (the same order in which they are added).

Features of Ordered

-Order of elements is maintained
-Indexing is possible (for list, tuple)
-Slicing is possible (for list, tuple)
-Elements can be accessed predictably
-Examples of Ordered is List, Tuple , Dictionary (Python 3.7+)

Unordered

In Python, unordered means elements do not have a fixed order. The order may change every time you run the program.

Features of Unordered

No fixed sequence
Indexing is not possible
Slicing is not possible
Mainly used for unique values
Example of Unordered:- Set

List Operation in Python:-

List operations in Python refer to the various actions that can be performed on a list to access, modify, add, remove, or combine elements stored in it. These operations help in managing and manipulating list data efficiently.

1).Indexing – (Accessing elements using index)

Indexing in a list refers to the process of accessing individual elements of a list using their position number, called an index .lists, indexing starts from 0 for the first element, while negative indexing starts from -1 for the last element.

Ex:-

```
a = [10, 20, 30]
print(a[1])
```

2).Slicing – Accessing a range of elements

Slicing in Python means cutting a part of a list to get multiple elements at once. Instead of taking one element (indexing), slicing helps us take a group of elements.

Ex:-

```
a = [10, 20, 30]
print(a[0:2])
```

3).Concatenation (+) – Combining two lists

Concatenation in Python lists is the process of joining two or more lists using the + operator. It returns a new list containing all elements of the original lists in order and does not change the original lists.

Ex:-

```
a = [10, 20, 30]
b = [40, 50]
print(a + b)
```

4).Repetition (*) – Repeating list elements

Repetition in lists is performed using * (astrick) operator to create a new list where the original list elements are repeated n times. It does not modify the original list and is useful for pattern generation.

Ex:-

```
a = [10, 20, 30]
b = [40, 50]
print(a * 2)
```

5).Membership (in, not in) –

Membership operators in and not in are used to check whether an element exists in a list. The in operator returns True if element is present, while not in returns True if element is absent.

Ex:-

```
a = [10, 20, 30]
b = [40, 50]
print(20 in a)
```

6).Length (len()) – Finding number of elements

The len() function in Python is used to find the total number of elements in a list. It counts each element as one item, including nested lists. Empty lists have length 0.

Ex:-

```
a = [10, 20, 30]
print(len(a))
```

Traversing a list:-

Traversing a list is the process of accessing each element of the list sequentially.
In Python, list traversal can be done using for loops, while loops, or built-in functions like enumerate().
Traversal is used to display elements, perform operations, or apply conditions on list items.
Since lists are ordered and index-based, elements can be accessed easily during traversal.

Methods of Traversing a List:-

1. Using for loop

Ex:-

```
lst = [10, 20, 30, 40]
for i in lst:
    print(i)
```

2.Using while loop

```
lst = [10, 20, 30, 40]
i = 0
while i <= 3:
    print(lst[i])
    i += 1
```

3. using enumerate()

enumerate() is a built-in Python function used to traverse a list (or any iterable) while getting both index (position) and value at the same time.
enumerate() = index + element together
Ex:-

```
languages = ['Python', 'Java', 'C']
for i, lang in enumerate(languages):
    print(i, lang)
```

Meaning:

i → stores index
lang → stores value

List Methods:-

List methods are built-in functions that work on list objects to add, remove, search, and modify elements in a list.

1. append():-

The append() method is used to add a single element at the end of a list in Python.
It modifies the original list and does not return any value.
The element added can be of any data type, including another list.
It is commonly used to insert new elements dynamically.

```
a1 = [10, 20]
a1.append(30)
print(a1)
```

2. extend():-

The extend() method in Python is used to add multiple elements from an iterable (list, tuple, string, etc.) to the end of an existing list. It modifies the original list without creating a new one. It differs from append() which adds the argument as a single element.

```
a1 = [1, 2]
a1.extend([3, 4,'SGDC'])
print(a1)
```

3.insert():-

insert() is a **built-in list method** that adds an element at a **given index** without replacing existing elements.

```
a1 = [10, 30]
a1.insert(1, 20)
print(a1)
```

4.remove():-

remove() is a **built-in list method** that deletes the **first occurrence** of a given element from the list.

```
a1 = [10, 20, 30]
a1.remove(20)
print(a1)
```

5.pop():-

pop() is a **built-in list method** used to **remove an element by its index** and **return that element**.

If index is given → removes element at that index

If index is not given → removes the last element by default

```
a1 = [10, 20, 30]
a1.pop()
print(a1)
```

6.clear():-

clear() is a built-in method in Python that is used to **remove all elements from a list**, leaving it as an **empty list**.

```
lst = [1, 2, 3]
lst.clear()
print(lst)
```

7.index():-

index() is a built-in list method used to **return the index (position)** of the **first occurrence** of a given element in the list.

```
lst = [10, 20, 30]
print(lst.index(30))
```

8.count():-

count() is a built-in list method that is used to **return the number of times a specific element appears** in the list.

```
lst = [10, 10, 20]
print(lst.count(10))
```

9.sort():- (Works only on lists)

Sorts the list in ascending order.(arranging elements from smallest to largest.)

```
lst = [30, 10, 20]
lst.sort()
print(lst)
```

Ex:-

```
languages = ['Python', 'Java', 'C']
languages.sort(reverse=True)
print(languages)
```

Ex:- using sorted function

```
numbers = [5, 2, 9, 1]
new_list = sorted(numbers, reverse=True)
print(new_list)
```

Descending order refers to arranging elements from the highest value to the lowest value.

In Python, this can be done using the sort() method with the parameter reverse=True or using the sorted() function with reverse=True.

10.reverse():-

reverse() is a built-in list method that **reverses the elements of the list in place**, i.e., it **changes the original list** and places elements in **reverse order**.

```
lst = [1, 2, 3]
lst.reverse()
print(lst)
```

11.copy():-

copy() is a built-in list method that is used to **create a shallow copy** of the list. It means it creates a **new list with the same elements** as the original.

```
lst1 = [1, 2]
lst2 = lst1.copy()
print(lst2)
```

List Built-in Functions in Python:-

Python built-in functions are predefined functions that can be used with lists to perform common operations like finding length, maximum, minimum, sum, sorting, checking conditions, and iterating with index.

1. len():-

len() is a built-in Python function used to **return the total number of elements** present in a list (or any iterable like tuple, string, dictionary, etc.).

Ex:-

```
lst = [10, 20, 30]
print(len(lst))  # Output: 3
```

2. max():-

max() is a built-in Python function that returns the **largest (maximum) element** from the given list or iterable.

```
lst = [10, 50, 20]
print(max(lst))
```

3. min():-
min() is a built-in Python function that returns the smallest (minimum) element from the given list or iterable.

```
lst = [10, 50, 20]
print(min(lst))
```

4. sum():-
sum() is a built-in Python function that is used to **calculate the total (sum) of all numeric elements** in an iterable like list or tuple.

```
lst = [1, 2, 3]
print(sum(lst))
```

5. sorted():-
Returns a new sorted list in ascending or descending order.
sorted() is a built-in Python function that **returns a new sorted list** from the given iterable **without modifying the original data**.

```
lst = [3, 1, 2]
print(sorted(lst))         # [1, 2, 3]
print(sorted(lst, reverse=True))  # [3, 2, 1]
```

6.enumerate():-
enumerate() is a built-in Python function that is used to **add counter/index numbers** while iterating over an iterable such as list, tuple, or string.

```
lst = ['a', 'b', 'c']
for i, val in enumerate(lst, start=1):
    print(i, val)
```

7.zip():-
zip() is a built-in Python function that is used to **combine two or more iterables** (like lists, tuples, strings) **element-wise into pairs (or tuples)**.

```
a = [1, 2]
b = ['x', 'y']
print(list(zip(a, b)))  # [(1, 'x'), (2, 'y')]
```

Tuples :-

A tuple is an ordered and immutable data type in Python used to store multiple values.
It is enclosed in parentheses and supports indexing, slicing, and traversal.
Tuples allow duplicate values and can store different data types.
Since tuples are immutable, their elements cannot be modified once created.
They are faster and more memory efficient than lists.

how to create a python tuple:-
1.using parenthesis()
2). using tuple() constructor

Properties of Tuple in Python:-

1. Ordered:-
Elements are stored in a fixed order
Supports indexing and slicing

```
t = (10, 20, 30)
print(t[0])
```

2.Immutable:-

Once created, elements cannot be modified

```
t = (10, 20, 30)
t[0] = 100
```

3. Allows Duplicate Values

Same value can appear more than once

```
t = (1, 2, 2, 3)
print(t)
```

4. Can Store Multiple Data Types

Can hold heterogeneous elements

```
t = (1, "Python", 3.14, True)
print(t)
```

Ex:-

```
t = (10, 20, 30)
print(t)
```

Ex:-

```
t = (10, 20, 30)
print(t[0])
print(t[1:3])
```

Tuple Operations in Python –

Tuple operations are the actions we can perform on tuples.

Since tuples are immutable, we cannot modify elements, but we can perform many operations on them.

1. Indexing :-

Indexing means **accessing elements of a tuple using their position (index number)**.

Index in Python always starts from **0**.

```
t = (10, 20, 30, 40)
print(t[0])
print(t[-1])
```

2. Slicing

Accessing a part of the tuple. Slicing means **extracting a portion (part) of a tuple** using **Indexing_range.**

It returns a **new tuple** containing selected elements.

```
t = (10, 20, 30, 40, 50)
print(t[1:4])
print(t[:3])
```

3. Concatenation (+)

Concatenation means **joining two or more tuples** to form a **new tuple** using the + operator.

```
t1 = (1, 2)
t2 = (3, 4)
print(t1 + t2)
```

4. Repetition (*)

Repetition means **repeating the elements of a tuple multiple times** using the * operator.

```
t = (1, 2)
print(t * 3)
```

5. Membership (in, not in)

Membership operators are used to **check whether an element exists in a tuple** or not.

- in → returns True if element **exists** in the tuple
- not in → returns True if element **does NOT exist** in the tuple

```
t = (10, 20, 30)
print(20 in t)
print(40 not in t)
```

6. Traversing:-

Traversing means **accessing each element of a tuple one by one**. Since tuples are **iterable**, we can traverse them using **loops**.

```
t = (10, 20, 30)
for x in t:
   print(x)
```

7. Length (len())

len() is a built-in Python function used to **find the total number of elements in a tuple**.

```
t = (10, 20, 30)
print(len(t))
```

8. Max & Min

max() → Returns the **largest element** in the tuple
min() → Returns the **smallest element** in the tuple

```
t = (5, 2, 9)
print(max(t))
print(min(t))
```

Tuple Methods in Python –

1. count() Method

count() is a built-in tuple method that is used to **count the number of times a specific element occurs** in a tuple.

```
t = (1, 2, 2, 3, 2)
print(t.count(2))
```

2. index() Method

index() is a built-in tuple method that is used to **find the index (position) of the first occurrence of a specified element** in a tuple.

```
t = (10, 20, 30, 20)
print(t.index(20))
```

Tuple Built-in Functions in Python –

Built-in functions are predefined Python functions that can be used with tuples to perform common operations like finding length, maximum, minimum, etc.

1. len()

len() is a built-in Python function used to **find the total number of elements in a tuple**.

```
t = (10, 20, 30)
print(len(t))
```

2. max()

max() is a built-in Python function that **returns the largest element** in a tuple.

```
t = (5, 2, 9, 1)
print(max(t))
```

3. min()

min() is a built-in Python function that **returns the smallest element** in a tuple.

```
t = (5, 2, 9, 1)
print(min(t))
```

4. sum()

sum() is a built-in Python function that **calculates the total sum of all numeric elements** in a tuple.

```
t = (1, 2, 3, 4)
print(sum(t))
```

5. sorted() (Works on list, tuple, set, string, dict)

sorted() is a built-in Python function that **returns a new sorted list** from any iterable (tuple, list, set, string, or dictionary keys) **without modifying the original data**.

```
t = (3, 1, 2)
print(sorted(t))
```

6. tuple()

tuple() is a built-in Python function used to **convert an iterable (like list, string, set, or dictionary) into a tuple**.

```
lst = [1, 2, 3]
print(tuple(lst))
```

7. enumerate()

enumerate() is a built-in Python function that **adds an index (counter) to an iterable** and returns it as an **enumerate object**.

- Useful when you want **both index and value** while looping.

```
t = ('a', 'b', 'c')
for i, v in enumerate(t):
   print(i, v)
```

Nested Tuple in Python –

A nested tuple is a tuple that contains another tuple as one of its elements.
(Tuple inside another tuple is called a nested tuple.)

Example of Nested Tuple:-

```
t = (1, 2, (3, 4))
print(t)
```

Ex:

```
nested_tuple = (10, 20, (30, 40, 50))
print(nested_tuple[2])
print(nested_tuple[2][1])
```

Ex:-

```
student = ("Rahul", 21, ("Maths", "Science", "English"))
print(student[2])
print(student[2][0])
```

Dictionary in Python:-

A dictionary in Python is a built-in data structure used to store data in the form of key–value pairs. It is written inside curly braces { } where each key is followed by a colon : and then its value.

In a dictionary, keys must be unique and immutable (e.g., strings, numbers, tuples), while values can be of any data type, including lists, numbers, strings, or even another dictionary. Dictionaries are mutable, so elements can be added, modified, or deleted after creation.

Creating a Dictionary

Using curly braces { }

```
d1 = {1: 'Shri', 2: 'Gnanambica', 3: 'Degree'}
print(d1)
```

Using dict() constructor

```
p1=dict({'name':'Raja','country':'India','phone':9090123123})
print(p1)
```

Ex:-

```
student = {"name": "Rahul", "age": 20}
print(student)
```

Properties of Dictionary in Python:-

1. Stores Data in Key–Value Pairs

A dictionary stores each item as a pair, where:

Key → identifies the item

Value → holds the data

2.Mutable (Changeable) Data Structure

Dictionaries can be modified after creation.

You can:

✓ Add new items

✓ Update values

✓ Delete items

```
student={"name":"Ravi",'age':21}
student['age']=25
student['city']='Madanapalle'
print(student)
```

3.Allows Heterogeneous Data

Dictionary can store different types of values in one place

```
info={
"name": "Ravi",
"age": 21,
"marks": 87.89,
"languages": ["python","java"],
"address":{'city':'madanapalle','state':'AP'}
```

```
}
print(info)
print(type(info))
```

Dictionary Operations in Python:-

1. Accessing Dictionary Elements:-

We can access values using keys.
Using [] (square brackets)
Ex:-

```
student = {"name": "Ravi", "age": 21}
print(student["name"])
```

Using get()

```
print(student.get("age"))
```

2. Adding New Elements

We can insert new key: value pairs at any time.

```
student = {"name": "Ravi"}
student["age"] = 21
student["city"] = "Kadapa"
print(student)
```

3. Updating Elements

In Python, a dictionary stores data as key : value pairs. Sometimes we need to change or update the value of an existing key or add a new key-value pair if it doesn't exist. This process is called updating elements.

-Method 1: Direct assignment

```
student["age"] = 22
```

-Method 2: Using update()

```
student.update({"city": "Kadapa"})
```

Both modify the dictionary.

4. Deleting Elements

In Python, you can remove (delete) elements from a dictionary in multiple ways. This includes deleting a single key-value pair, deleting the last inserted item, clearing all items, or deleting the whole dictionary.

a). Using del Keyword

Used to delete a specific key-value pair or delete the entire dictionary.
Ex:- (Delete a single item)

```
student = {"name": "Ravi", "age": 20, "city": "Hyderabad"}
del student["age"]
print(student)
```

Ex:- (Delete entire dictionary)

```
student = {"name": "Ravi", "age": 20}
del student
```

Now student does not exist.

b). Using pop() Method

Deletes a specific key and returns its value.

Ex:-

```
student = {"name": "Ravi", "age": 20, "city": "Hyderabad"}
value = student.pop("city")
print(value)
print(student)
```

c). Using popitem() Method

Deletes and returns the last inserted key-value pair.

Example:

```
student = {"name": "Ravi", "age": 20, "city": "Hyderabad"}
item = student.popitem()
print(item)
print(student)
```

d). Using clear() Method

Removes all items from the dictionary and makes it empty.

Example:

```
student = {"name": "Ravi", "age": 20, "city": "Hyderabad"}
student.clear()
print(student)
```

5. Traversing (Looping)

We can loop through keys, values, or both.

Loop keys:-

```
student = {"name": "Ravi", "age": 21}
for key in student:
    print(key)
```

Loop values:-

```
student = {"name": "Ravi", "age": 21}
for value in student.values():
    print(value)
```

Loop key–value pairs:-

```
student = {"name": "Ravi", "age": 21}
for key, value in student.items():
    print(key, value)
```

6. Membership Testing (in & not in)

Membership operators in and not in are used to check if a key exists in a dictionary.

```
student = {"name": "Ravi", "age": 21}
print("name" in student)   # True
print("city" in student)   # False
```

Note: This checks keys, not values.

7. Length of Dictionary

Use len() to count number of key-value pairs.

Ex:-

```
student = {"name": "Rahul", "age": 21, "marks": 85}
print(len(student))
```

8. Copying

```
student = {"name": "Rahul", "age": 21}
new_student = student.copy()
print(new_student)
```

9. Sorting a Dictionary

Sorting happens based on keys.

```
d1={'b':2,'z':3,'a':5,'d':6}
print(sorted(d1))
```

Ex:-

```
d1={'b':2,'z':3,'a':5,'d':6}
print(sorted(d1.keys()))
```

Ex:-

```
d1={'b':2,'z':3,'a':5,'d':6}
print(sorted(d1.values()))
```

Ex:-

```
d1={'b':2,'z':3,'a':5,'d':6}
print(sorted(d1.items()))
```

Ex:-

```
d1={'b':2,'z':3,'a':5,'d':6}
print(sorted(d1, reverse=True))
```

10. Nested Dictionary Operations

Dictionaries can contain dictionaries.

```
students = {
   1: {"name": "Ravi", "marks": 85},
   2: {"name": "Sita", "marks": 90}
}
print(students[1]["name"])
```

Ex:-

```
students = {
   "student1": {"name": "Ravi", "age": 20, "marks": 85},
   "student2": {"name": "Kiran", "age": 19, "marks": 90},
   "student3": {"name": "Asha", "age": 21, "marks": 78}
}
print(students)
```

Accessing Nested Dictionary Values:-

```
print(students["student1"]["name"])
```

Updating Nested Dictionary:-

```
students["student3"]["marks"] = 80
print(students["student3"])
```

Traversing a Dictionary in Python:-

Traversing a dictionary means visiting each key, value or key-value pair in the dictionary one by one using a loop (mostly for loop).

Ex:- (Traversing Only Keys)

```
student = {"name": "Rahul", "age": 21, "course": "B.Sc"}
for key in student:
   print(key)
```

Ex:- (Traversing Only Values)

```
student = {"name": "Rahul", "age": 21, "course": "B.Sc"}
for value in student.values():
   print(value)
```

Ex:- (Traversing Keys & Values Together)

```
s1={'name':'rahul','age':21,'course':'BCA AI'}
for a1,b1 in s1.items():
   print(a1,":",b1)
```

DICTIONARY METHODS IN PYTHON:-

Dictionary methods are built-in functions used to access, modify and manage dictionary data.

1) .keys():-

The .keys() method is a built-in dictionary method in Python used to return a view object that contains all the keys of the dictionary. Since a dictionary stores data in the form of key-value pairs, the .keys() method helps in accessing only the keys without the values.

Ex:-

```
student = {"name": "Ravi", "age": 20, "city": "Hyderabad"}
print(student.keys())
```

2) .values() :-

In Python, a dictionary stores data in key : value pairs.
Sometimes we need to access only the values without the keys.
For this purpose, Python provides a built-in dictionary method called .values().
Ex:-

```
student = {"name": "Ravi", "age": 20, "city": "Hyderabad"}
print(student.values())
```

3) .items() :-

In Python, a dictionary stores data as key : value pairs.
Sometimes we need both the key and the value together while processing.
For this purpose, Python provides a built-in dictionary method called .items().
Ex:-

```
student = {"name": "Ravi", "age": 20, "city": "Hyderabad"}
print(student.items())
```

4. get(key)

Returns the value of the given key. If key not found, returns None (no error).
Example:

```
student = {"name": "Rahul", "age": 21}
print(student.get("name"))
print(student.get("marks"))
```

Output:
Rahul
None

5. update(dictionary2)

update() is a dictionary method that is used to **add new key-value pairs** or **update existing keys** in a dictionary.

Example:

```
d1 = {"name": "Rahul", "age": 21}
d2 = {"marks": 85}
d1.update(d2)
print(d1)
```

Output:

{'name': 'Rahul', 'age': 21, 'marks': 85}

6. pop(key)

pop() is a dictionary method that **removes the item with the specified key** and **returns its value**.

Example:-

```
student = {"name": "Rahul", "age": 21}
```

```
print(student.pop("age"))
print(student)
```

Output:
21
{'name': 'Rahul'}

7. popitem()

popitem() is a dictionary method that **removes and returns the last inserted key-value pair** as a **tuple**.

- In Python **3.7+**, dictionaries **preserve insertion order**, so popitem() removes the **most recently added item**.

Example:

```
student = {"name": "Rahul", "age": 21}
print(student.popitem())
print(student)
```

8.clear()

clear() is a dictionary method that **removes all items** from the dictionary, making it **empty**.

Example:

```
student = {"name": "Rahul", "age": 21}
student.clear()
print(student)
```

9. copy()

copy() is a dictionary method that **returns a shallow copy of the dictionary**.

- Shallow copy means it **creates a new dictionary** with the same key-value pairs, but changes to **nested objects** will still affect the original.

Example:

```
student = {"name": "Rahul", "age": 21}
new_dict = student.copy()
print(new_dict)
```

Dictionary Built-in Functions:-

In Python, a dictionary is a data structure that stores data in the form of **key–value pairs**. To perform different operations on dictionaries, Python provides several **built-in functions**. These built-in functions help in tasks such as finding the number of elements, checking the type of dictionary, converting it into a string form, and sorting keys.

1. len()

- The len() function returns the **total number of key–value pairs** in a dictionary.
- It counts each pair as **one item**.
- It does **not** count keys and values separately.
- Useful for finding the **size** of a dictionary.

Example:

```
student = {"name": "Rahul", "age": 21, "marks": 85}
print(len(student))
```

2. type()

- The type() function returns the **data type** of the given object.
- When used with a dictionary, it returns **<class 'dict'>**.
- Useful for debugging and verification during coding.

Example:

```
student = {"name": "Rahul", "age": 21}
print(type(student))
```

3. str()

- The str() function converts the dictionary into a **string representation**.
- After conversion, the dictionary can be printed, stored, or displayed as text.
- Useful for logging, file writing, and user display.

Example:

```
student = {"name": "Rahul", "age": 21}
print(str(student))
```

4. sorted()

- The sorted() function returns a **sorted list of dictionary keys**.
- Sorting is done in **ascending order** by default.
- It does **not** sort values unless specified manually.
- Does **not modify the original dictionary**.

Example:

```
student = {"b": 2, "a": 1, "c": 3}
print(sorted(student))
```

Output:

```
['a', 'b', 'c']
```

NOTE:
sorted() does NOT sort values, it sorts only keys
It returns a new list, does not modify original dictionary

5. max()

- the max() function returns the **maximum key** in the dictionary.
- Comparison is performed **between keys only**, not values.
- Keys are compared based on ASCII/Unicode order (Case-sensitive).
- Example: 'b' > 'a' and 'Z' < 'a'.

Example:
data = {5: "a", 2: "b", 9: "c"}
print(max(data))
If keys are strings:

info = {"apple": 1, "banana": 2, "grapes": 3}
print(max(info))

Output:
grapes

6. min()

- The min() function returns the **minimum key** in a dictionary.
- Comparison rules are same as max().
- Useful for finding the smallest key in sorted order.

Example:
data = {5: "a", 2: "b", 9: "c"}
print(min(data))

7. any()

any() returns **True** if **at least one element** in the given iterable (list, tuple, set, dictionary, etc.) is **True**.

✓ How it Works with Dictionary:

- When used with a dictionary, any() checks **keys** by default.
- If at least one key is **truthy**, it returns **True**.
- 0, False, None, and "" (empty string) are **Falsey** values.

Example 1:
d = {0: "a", 0: "b"}
print(any(d))

Output:
False
Example 2:
d = {1: "a", 0: "b"}
print(any(d))

Output:
True
Example 3:

```
d = {}
print(any(d))
```

Output:
False

8. all()

all() returns **True** only if **every element** in the iterable is **True**.

✓ **How it Works with Dictionary:**

- all() checks **all keys**.
- If **all keys are truthy**, returns **True**
- If **any key is false**, it returns **False**

Example 1:

```
d = {1: "a", 2: "b", 3: "c"}
print(all(d))
```

Output:
True
Example 2:

```
d = {1: "a", 0: "b", 3: "c"}
print(all(d))
```

Output:
False

B. Devendra Msc CS
Dept of Computers
Shri Gnanambica Degree College(A).