What are List Functions?

List functions are **ready-made tools in Python** that help you:

- Organize your list
- Count items
- Copy a list
- Remove or reverse values

They make your life much easier when working with list data.

Most Common List Functions

`sort()` → Sorts list in ascending order

```
numbers = [3, 1, 5, 2]
numbers.sort()
print(numbers)
```

Output:

```
[1, 2, 3, 5]
```

It sorts the original list itself (permanently changes it).

`reverse()` → Reverses the order of the list

```
names = ["Anjali", "Ravi", "Tina"]
```

```
names.reverse()
print(names)
```

Output:

```
['Tina', 'Ravi', 'Anjali']
```

`copy()` → Makes a copy of the list

```
list1 = [10, 20, 30]
list2 = list1.copy()
print(list2)
```

Output:

```
[10, 20, 30]
```

Now changing `list2` won't affect `list1`.

`count(value)` → Counts how many times a value appears

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

Output:

```
2
```

`index(value)` → Finds the first position of a value

```
colors = ["red", "blue", "green"]
print(colors.index("blue"))
```

Output:

```
1
```

`extend()` → Joins two lists

```
a = [1, 2]
b = [3, 4]
a.extend(b)
print(a)
```

Output:

```
[1, 2, 3, 4]
```

Practice Questions

1. Create a list of 5 numbers and sort them using `sort()`
2. Reverse the list `["cat", "dog", "lion"]`
3. Count how many times `"apple"` appears in this list: `["apple", "mango", "apple", "grape"]`
4. Find the index of `"banana"` in `["apple", "banana", "cherry"]`
5. Copy a list of 3 colors to a new variable
6. Combine two lists: one of fruits and one of vegetables using `extend()`
7. Try this: `list1 = [10, 20]; list2 = list1.copy(); list2.append(30)` — what is in both lists now?

8. Create a list of marks, and sort them in descending order (hint: sort + reverse)

What is a Tuple?

A **tuple** is like a **list**, but with one big difference:

> **You cannot change a tuple after creating it.**

That means:

- No adding items
- No removing items
- No updating items

This makes tuples **safe** and **fast** for data you don't want to change.

---

2. Creating a Tuple

Use **round brackets** `()` instead of square brackets `[]`.

```
fruits = ("apple", "banana", "mango")
print(fruits)
```

Output:

```
('apple', 'banana', 'mango')
```

---

3. Accessing Tuple Items (Same as List)

```
print(fruits[0])    # 'apple'
print(fruits[-1])   # 'mango'
```

---

4. Tuples are Immutable (Not Changeable)

Trying to change a value will cause an error:

```
fruits[1] = "orange"   # ❌ Error
```

Output:

```
TypeError: 'tuple' object does not support item assignment
```

Practice Questions

1. Create a tuple of 3 favorite colors and print it
2. Try accessing the 2nd color from the tuple
3. Try changing a value and see what error you get
4. Create a tuple of 5 numbers. Use a loop to print each one
5. Create a tuple of weekdays
6. Create a tuple of `("Python", "Java", "C++")`, then print the last language

Functions That Work with Tuples

---

1. `len()` → Returns the number of items

```
fruits = ("apple", "banana", "mango")
print(len(fruits))
```

Output:

```
3
```

---

2. `count(value)` → Counts how many times a value appears

```
numbers = (1, 2, 3, 2, 2, 4)
print(numbers.count(2))
```

Output:

```
3
```

---

3. `index(value)` → Finds the first position of the value

```
colors = ("red", "blue", "green", "blue")
print(colors.index("blue"))
```

Output:

```
1
```

⚠️ If the value is not found, it gives an error.

---

4. `sum()` → Adds all values (numbers only)

```
marks = (80, 75, 60)
print(sum(marks))
```

Output:

```
215
```

---

5. `max()` and `min()` → Get the biggest/smallest value

```
nums = (10, 50, 20, 5)
print(max(nums))  # 50
print(min(nums))  # 5
```

Output:

```
50
5
```

⚠️ Works only with **numbers** or **comparable items** (like all strings).

---

6. `sorted()` → Returns a sorted list version of the tuple

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

Output:

```
[10, 20, 30]
```

Note: `sorted()` doesn't change the tuple. It returns a **new list**.

What is a Dictionary?

A **dictionary** in Python stores data in **key-value** format.

You use a **key** to access the **value** — just like a real dictionary where you search a word (key) to get its meaning (value).

Example:

```
student = {
    "name": "Anjali",
    "age": 20,
    "course": "Python"
}
```

Here:

- `"name"` is a key → `"Anjali"` is the value
- `"age"` is a key → `20` is the value
- `"course"` is a key → `"Python"` is the value

---

2. Creating a Dictionary

```
info = {"brand": "Nike", "price": 2999}
print(info)
```

Output:

```
{'brand': 'Nike', 'price': 2999}
```

---

3. Accessing Values

```
print(info["brand"])      # Nike
```

Using `.get()` (avoids error if key doesn't exist):

```
print(info.get("color", "Not found"))
```

Output:

```
Not found
```

---

4. Changing & Adding Items

Change value:

```
info["price"] = 2499
```

Add new key-value pair:

```
info["category"] = "Shoes"
```

---

5. Removing Items

```
info.pop("price")     # Removes 'price'
del info["brand"]     # Removes 'brand'
info.clear()          # Empties the dictionary
```

---

6. Looping Through Dictionary

```
student = {"name": "Amit", "age": 19}

for key, value in student.items():
    print(key, "→", value)
```

Output:

```
name → Amit
age → 19
```

---

7. Useful Dictionary Functions

Function	Description
`.keys()`	Returns all keys
`.values()`	Returns all values
`.items()`	Returns key-value pairs (tuple format)

`.get(key)`	Safer way to access a value
`.update()`	Adds or updates multiple key-values
`.pop(key)`	Removes the key-value pair
`.clear()`	Empties the dictionary

---

1. `get()` – To Access Values

```
student = {"name": "Anjali", "age": 20}
print(student.get("name"))       # Anjali
print(student.get("email", "N/A"))  # N/A (default)
```

Use this instead of `dict["key"]` to **avoid errors** if the key is missing.

---

2. `keys()` – Returns All Keys

```
print(student.keys())
```

Output:

```
dict_keys(['name', 'age'])
```

Use in loops:

```
for key in student.keys():
    print(key)
```

---

3. `values()` – Returns All Values

```
print(student.values())
```

Output:

```
dict_values(['Anjali', 20])
```

---

4. `items()` – Returns All Key-Value Pairs

```
for k, v in student.items():
    print(k, "→", v)
```

Output:

```
name → Anjali
age → 20
```

---

5. `update()` – Add or Modify Key-Value Pairs

```
student.update({"course": "Python", "city": "Delhi"})
print(student)
```

Output:

```
{'name': 'Anjali', 'age': 20, 'course': 'Python', 'city': 'Delhi'}
```

---

6. `pop(key)` – Remove Item by Key

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

Output:

```
{'name': 'Anjali', 'course': 'Python', 'city': 'Delhi'}
```

⚠️ Gives an error if the key doesn't exist (unless you give a default value).

---

7. `clear()` – Removes All Items

```
student.clear()
print(student)
```

Output:

CopyEdit
```
{}
```

---

8. `copy()` – Makes a Copy of Dictionary

```
original = {"a": 1, "b": 2}
duplicate = original.copy()
print(duplicate)
```

🔒 Safe copy — changes to `duplicate` won't affect `original`.

Practice Questions

1. Create a dictionary of a student with name, age, course

2. Print the course using the key
3. Add a new key `"marks"` with value 95
4. Change the student's name
5. Use `.get()` to access `"email"` key — handle the missing case
6. Loop through the dictionary and print all key-value pairs
7. Create a dictionary of 3 countries with their capitals
8. Use `.update()` to add `"city": "Delhi"` and `"college": "XYZ"`
9. Remove `"age"` key from the student dictionary
10. Use `.keys()` and `.values()` to print them separately