Combining List

Table of contents

01

Copying List

Introduction

In Python, lists are **mutable** objects, meaning they can be changed after creation. When working with lists, you might want to create a separate copy to avoid unintended modifications.

There are multiple ways to **copy a list**, like:

- Using **list**() function
- Using the **copy**() method
- Using **slice** notation

Using list() function

I have a question for you…what do you think…what will be the output of the following code?

Python

```
original_list = [1, 2, 3]
copied_list = original_list
copied_list.append(4)

print(original_list)
```

Output

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

Using list() function

The original list has changed…
But **WHY**?

Reason:
Both **original_list** and **copied_list** refer to the *same* list. Any changes made to one affect the other.

If you assign one list to another using **=**, it doesn't create a new list. Instead, both variables will point to the same list in memory.

Python

```
original_list = [1, 2, 3]
copied_list = original_list
copied_list.append(4)

print(original_list)
```

Output

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

Using list() function

So is it *IMPOSSIBLE* to make a list using b = a?

Answer:
NO.
You just need to use **list()** function.

Python	**Output**
`original_list = [1, 2, 3]`	
`copied_list = list(original_list)`	
`copied_list.append(4)`	`[1, 2, 3]`
	`[1, 2, 3, 4]`
`print(original_list)`	
`print(copied_list)`	

Using .copy() method

You can also use something else called **.copy()** method.

The **copy()** method is a built-in list method and it is recommended for simple lists

Python	Output

```
original_list = [1, 2, 3]
copied_list = original_list.copy()
copied_list.append(4)

print(original_list)
print(copied_list)
```
	```
	[1, 2, 3]
	[1, 2, 3, 4]
	```

Using slice notation

Slicing a list using **[:]** creates a shallow copy of the list.

Python	Output
`original_list = [1, 2, 3]`	
`copied_list = original_list[:]`	
`copied_list.append(4)`	`[1, 2, 3]`
	`[1, 2, 3, 4]`
`print(original_list)`	
`print(copied_list)`	

Shallow Copy Vs Deep Copy

Shallow copy and deep copy can be confusing but actually, it is simple. Let me explain.

Shallow Copy	**Deep Copy**
A **shallow copy** only copies the outermost list, meaning if the list contains other lists (nested lists), they are not fully copied.	A **deep copy** creates a completely independent copy, including all nested lists.

Deep Copy

- If a list contains **nested lists**, a simple copy might not work as expected. This is because inner lists remain linked.

- To perform a deep copy, use the built-in **copy** module's **deepcopy()** function.

Python	Output

`import copy`
 	
`original_list = [[1, 2], [3, 4]]`
`copied_list = copy.deepcopy(original_list)`
`copied_list[0].append(5)`
 	
`print(original_list`
`print(copied_list)`
	`[[1, 2], [3, 4]]`
	`[[1, 2, 5], [3, 4]]`

Summary

- Assignment (**=**) doesn’t create a real copy, just a new reference.
- Use **“.copy()”**, **“= with list() function”** and **“Slicing Notation [:]”** for simple lists.
- For nested lists, use **copy.deepcopy**() to avoid unexpected changes.

02

Joining List

Introduction

- Joining lists means **merging multiple lists into one**.

→ There are multiple ways to **Join** Lists:

- **'+' Operator** → Creates a new list.
- **.extend()** → Adds elements to an existing list.
- **.append()** → Combined with loop to add element to an existing list.
- **List Comprehension** → Merges with customization.

Using ‘+’ Operator

- The simplest way to join two or more lists is by using the **+** operator.
- This method creates a new list that contains elements from all the lists.

Python	**Output**

`list1 = [1, 2, 3]`
`list2 = [4, 5, 6]`
`joined_list = list1 + list2`
`print(joined_list)`
	`[1, 2, 3, 4, 5, 6]`

Using extend() Method

- The **extend()** method adds elements from one list to the end of another list in place.

- Unlike **+**, it modifies the original list.

Python	Output

`list1 = [1, 2, 3]`
`list2 = [4, 5, 6]`
`list1.extend(list2)`
`print(list1)`
	`[1, 2, 3, 4, 5, 6]`

Using append() Method

- The **append()** method adds elements to the end of another list one by one using loop.
- This function **modifies** the original list by adding the element to the end of the list.

Python

```python
list1 = ["a", "b" , "c"]
list2 = ["d", "e" , "f"]

for x in list2:
  list1.append(x)

print(list1)
```

Output

```
['a', 'b', 'c', 'd', 'e', 'f']
```

Using List Comprehension

- You can use list comprehensions to **join** lists in a **more customizable way.**
- It is used to **generate new lists** by applying an **expression** to each item in an existing iterable.

Python	Output

`list1 = [1, 2, 3]`
`list2 = [4, 5, 6]`
 	
`merged_list = [item for sublist in [list1, list2] for item in sublist]`
 	
`print(merged_list)`
	`[1, 2, 3, 4, 5, 6]`

Using List Comprehension

- You can use list comprehensions to **join** lists in a **more customizable way.**

- It is used to **generate new lists** by applying an **expression** to each item in an existing iterable.

Python	Output

`list1 = [1, 2, 3]`
`list2 = [4, 5, 6]`
 	
`merged_list = [item for sublist in [list1, list2] for item in sublist]`
 	
`print(merged_list)`
	`[1, 2, 3, 4, 5, 6]`

- First loops through **each list** (list1 and list2).

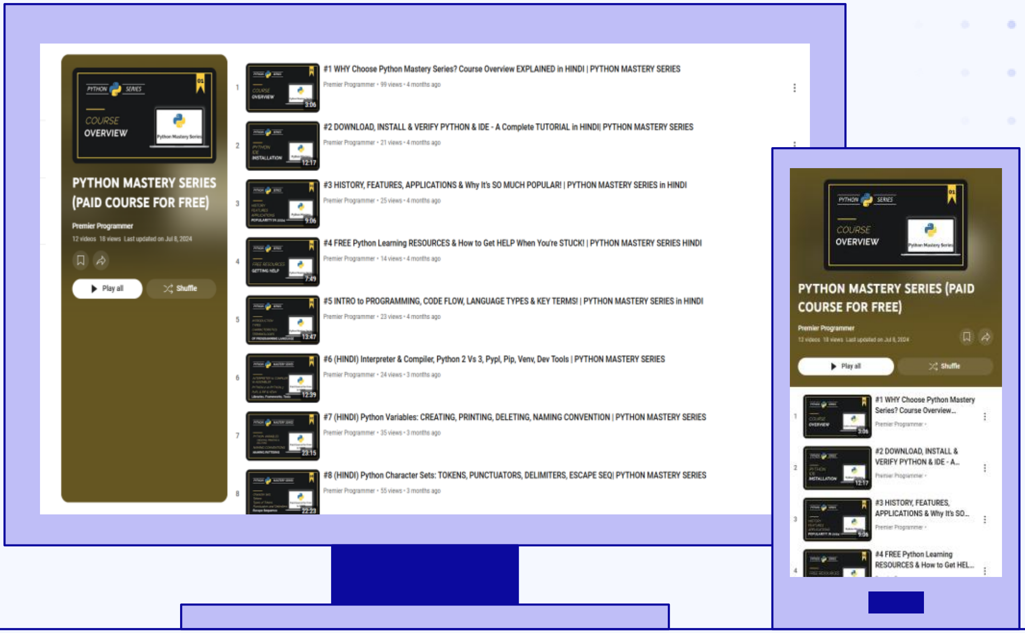
- Then extracts **each element** and places it in merged_list.

Summary

- Use the **+ operator** for simple and quick concatenation.
- Use the **extend() & append()** method when you need to modify a list in place.
- Use **list comprehensions** for more control over how lists are joined.

WATCH

Level up your coding with each episode in this focused Python series.



PYTHON MASTERY SERIES (PAID COURSE FOR FREE)

Premier Programmer

12 videos 18 views Last updated on Jul 8, 2024

1 #1 WHY Choose Python Mastery Series? Course Overview EXPLAINED in HINDI | PYTHON MASTERY SERIES
Premier Programmer • 99 views • 4 months ago

2 #2 DOWNLOAD, INSTALL & VERIFY PYTHON & IDE - A Complete TUTORIAL in HINDI| PYTHON MASTERY SERIES
Premier Programmer • 21 views • 4 months ago

3 #3 HISTORY, FEATURES, APPLICATIONS & Why It's SO MUCH POPULAR! | PYTHON MASTERY SERIES in HINDI
Premier Programmer • 25 views • 4 months ago

4 #4 FREE Python Learning RESOURCES & How to Get HELP When You're STUCK! | PYTHON MASTERY SERIES HINDI
Premier Programmer • 14 views • 4 months ago

5 #5 INTRO to PROGRAMMING, CODE FLOW, LANGUAGE TYPES & KEY TERMS! | PYTHON MASTERY SERIES in HINDI
Premier Programmer • 23 views • 4 months ago

6 #6 (HINDI) Interpreter & Compiler, Python 2 Vs 3, Pypl, Pip, Venv, Dev Tools | PYTHON MASTERY SERIES
Premier Programmer • 24 views • 3 months ago

7 #7 (HINDI) Python Variables: CREATING, PRINTING, DELETING, NAMING CONVENTION | PYTHON MASTERY SERIES
Premier Programmer • 35 views • 3 months ago

8 #8 (HINDI) Python Character Sets: TOKENS, PUNCTUATORS, DELIMITERS, ESCAPE SEQ| PYTHON MASTERY SERIES
Premier Programmer • 55 views • 3 months ago

Next Video!

List Methods