Python Lists & List Comprehension – Complete Notes (Zero to Hero)

◇ 1. What is a List?

A **list** in Python is an **ordered**, **mutable**, **indexed** collection that can store **heterogeneous data**.

```
my_list = [1, 2, 3, "hello", True]
```

2. List Basics

Operation	Example	Notes
Create List	`l = [1, 2, 3]`	Use `[]` brackets
Indexing	`l[0]` → `1`	Starts at 0
Negative Index	`l[-1]` → `3`	From the end
Slicing	`l[1:3]` → `[2, 3]`	End index excluded
Nested List	`l = [[1, 2], [3, 4]]`	Matrix or 2D arrays

3. Modifying Lists

Operation	Example	Description
Append	`l.append(4)`	Adds to end
Insert	`l.insert(1, "hi")`	At specific index
Extend	`l.extend([5,6])`	Adds multiple items
Remove (by value)	`l.remove(2)`	First match only
Pop (by index)	`l.pop()` / `l.pop(1)`	Removes & returns element
Del (by index/slice)	`del l[0]` or `del l[1:3]`	Deletes element/slice
Clear all	`l.clear()`	Empties list

4. List Searching & Info

Method	Example	Description
`in`	`3 in l` → `True/False`	Membership check

Method	Example	Description
`index()`	`l.index(3)`	First index of item
`count()`	`l.count(3)`	Number of occurrences
`len()`	`len(l)`	Length of list

🖌️ 5. Sorting & Reversing

Method	Example	Description
`sort()`	`l.sort()`	Sorts in-place (ascending)
`sort(reverse=True)`	Descending order	
`sorted(l)`	Returns sorted copy	Original list unchanged
`reverse()`	`l.reverse()`	Reverses in-place
`reversed(l)`	`list(reversed(l))`	Returns reversed copy

🧠 6. Iterating Lists

```
for item in l:
    print(item)

# With index
for i, item in enumerate(l):
    print(f"Index {i} → {item}")
```

⚙️ 7. List Functions

Function	Description
`max(l)`	Largest item
`min(l)`	Smallest item
`sum(l)`	Total (only for numeric lists)
`list()`	Convert iterable to list
`range()`	Commonly used with list
`all(l)`	True if all values are truthy
`any(l)`	True if any value is truthy

🧩 8. List Copying (Shallow vs Deep) 📋

```
a = [1, 2, 3]
b = a          # Same reference 😬
c = a.copy()   # New shallow copy ☑
d = list(a)    # Also a copy
```

🔁 **Deep Copy** (for nested lists):

```
import copy
deep = copy.deepcopy(nested_list)
```

🧹 9. List vs Tuple

Feature	List	Tuple
Mutable	☑ Yes	❌ No
Syntax	`[]`	`()`
Performance	Slower	Faster
Use case	Changing data	Fixed data

💎 10. List Comprehension (💖 Best Python Feature)

> ☑ Fast, readable, and Pythonic way to create lists from iterables.

☑ Syntax:

```
[expression for item in iterable if condition]
```

🧹 Examples:

```
# Squares
[x**2 for x in range(5)] → [0, 1, 4, 9, 16]

# Even numbers
[x for x in range(10) if x % 2 == 0] → [0, 2, 4, 6, 8]

# Uppercase strings
[s.upper() for s in ["hi", "bye"]] → ['HI', 'BYE']
```

```
# Conditional expression
['Even' if x%2==0 else 'Odd' for x in range(5)]
```

Nested List Comprehension

```
matrix = [[1,2], [3,4]]
flat = [item for row in matrix for item in row]  # [1,2,3,4]
```

---

11. Real-World List Comprehension Tricks

```
# Extract numbers from string
[int(ch) for ch in "a1b2c3" if ch.isdigit()] → [1, 2, 3]

# Filter out None values
[val for val in data if val is not None]

# Find common elements
[a for a in list1 if a in list2]
```

---

12. List Interview Problems

Problem	One-Liner or Idea
Reverse list	`l[::-1]`
Remove duplicates	`list(set(l))` (order may change)
Flatten nested list	`[x for sub in nested for x in sub]`
Second largest element	`sorted(set(l))[-2]`
Pair sum = target	Use 2-pointer / hash map
Rotate list by k	`l[k:] + l[:k]`
Longest increasing sublist	Sliding window / DP

---

13. 2D Lists (Matrix)

```
matrix = [[1, 2, 3], [4, 5, 6]]

# Access
matrix[1][2] → 6
```

```python
# Row-wise loop
for row in matrix:
    print(row)

# Flattening
flat = [x for row in matrix for x in row]
```

---

14. Useful Libraries

Library	Use Case
`collections`	Counter, deque, defaultdict
`itertools`	Combinations, permutations
`numpy`	Advanced array ops (for ML/DS)

Summary Table

Concept	Example
Add	`append(), insert(), extend()`
Remove	`pop(), remove(), del`
Search/Info	`in, index(), count()`
Sort/Reverse	`sort(), reverse(), sorted()`
Copy	`copy(), deepcopy()`
Loop	`for x in list:`
List Comp	`[x for x in list]`
Flatten	`[i for sub in l for i in sub]`

Practice Problems You Should Try

- [ ] Reverse a list in-place
- [ ] Find duplicates in a list
- [ ] Group anagrams
- [ ] Merge 2 sorted lists
- [ ] Flatten a 2D list
- [ ] Shift elements left/right by `k`
- [ ] Remove falsy values (`None`, `0`, `False`, `''`)

Understanding the Confusion

In Python, when you **assign to a slice**, the behavior can differ based on **what you're assigning**:

🧪 Behavior Demo

```
tea_varieties = ['Black', 'Oolong', 'Herbal']

# ❌ Case 1: Assigning a string to a slice
tea_varieties[1:2] = "Lemon"
print(tea_varieties)
```

🔍 Output:

```
['Black', 'L', 'e', 'm', 'o', 'n', 'Herbal']
```

> ⚠ Why this happened: Because `"Lemon"` is a **string**, and strings are **iterables**, so Python treats each character as an element and inserts them **one by one**.

☑ Correct Way: Wrap it in a list to insert as a single item

```
tea_varieties = ['Black', 'Oolong', 'Herbal']

tea_varieties[1:2] = ["Lemon"]
print(tea_varieties)
```

✔ Output:

```
['Black', 'Lemon', 'Herbal']
```

🔄 Regular (non-slice) Assignment

```
tea_varieties[1] = "Lemon"
print(tea_varieties)
```

🎯 Output:

```
['Black', 'Lemon', 'Herbal']
```

> ☑ Here, `"Lemon"` replaces the value at index `1` directly, no matter if it's a string or list.

🔑 Key Takeaway

Operation	Effect
`list[i] = "Lemon"`	Replaces one element with a string
`list[i:j] = "Lemon"`	Spreads `"Lemon"` (string) into chars
`list[i:j] = ["Lemon"]`	Inserts `"Lemon"` as a single list item

🧠 Python List Slicing Assignment – Master Guide

Let's understand how Python treats **slice assignments** in lists differently than regular assignments — especially when inserting or deleting values.

◇ Example Setup

```
tea = ['Black', 'Green', 'Masala', 'White']
```

❌ Typo Example

```
>>> trea
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'trea' is not defined. Did you mean: 'tea'?
```

> ⚠ Python throws a `NameError` if you misspell a variable. It even gives a helpful suggestion!

☑ Correct Name

```
>>> tea
['Black', 'Green', 'Masala', 'White']
```

◇ Insertion Using Slice

👇 Insert items at index `1` using slicing

```
tea[1:1] = ["Test", "Test"]
print(tea)
```

📤 Output:

```
['Black', 'Test', 'Test', 'Green', 'Masala', 'White']
```

☑ This **inserts** two elements *before* index 1.

◇ Deletion Using Slice

👇 Remove items at positions 1 and 2

```
tea[1:3] = []
print(tea)
```

🖌 Output:

```
['Black', 'Green', 'Masala', 'White']
```

☑ This deletes a slice by assigning it an empty list.

◇ Special Case: Assigning a String to a Slice

```
tea = ['Black', 'Oolong', 'Herbal']
tea[1:2] = "Lemon"
print(tea)
```

😵 Output:

```
['Black', 'L', 'e', 'm', 'o', 'n', 'Herbal']
```

❗ Because `"Lemon"` is a string (which is iterable), Python **splits it into characters** and inserts them one-by-one.

☑ Correct Way: Use a List for Whole Item

```
tea = ['Black', 'Oolong', 'Herbal']
tea[1:2] = ["Lemon"]
print(tea)
```

✔ Output:

```
['Black', 'Lemon', 'Herbal']
```

◇ Regular Index Assignment (No Slicing)

```
tea[1] = "Lemon"
print(tea)
```

🎯 Output:

```
['Black', 'Lemon', 'Herbal']
```

> ☑ This directly **replaces** the value at index `1`.

🧠 Key Takeaways

Syntax	Result
`list[i] = "Lemon"`	Replaces item at `i` with `"Lemon"`
`list[i:j] = "Lemon"`	Inserts characters `'L','e','m','o','n'`
`list[i:j] = ["Lemon"]`	Inserts `"Lemon"` as a **single item**
`list[i:j] = []`	Deletes slice from `i` to `j-1`
`list[i:i] = [x, y]`	Inserts at index `i`

🔍 Python `for` Loop with `print(..., end="=>")` – Full Explanation

🧪 Example:

```
tea = ['Green', 'Masala', 'White']

for key in tea:
    print(key, end="=>")
```

🖨️ Output:

```
Green=>Masala=>White=>
```

🧠 What's Happening Here?

1. `for key in tea:`

🔁 You're looping over each element in the list `tea`.

Iteration	Value of `key`
1st	`'Green'`
2nd	`'Masala'`
3rd	`'White'`

---

2. `print(key, end="=>")`

Normally, `print()` adds a **newline (`\n`)** after each value. But when you set `end="=>"`, Python will:

- **Avoid the default newline**
- **Append the string** `=>` after each print instead

🔄 Loop Execution Flow:

Let's walk through what Python does internally:

Step	`key`	What gets printed	Final Output So Far
1	`"Green"`	`Green=>`	`Green=>`
2	`"Masala"`	`Masala=>`	`Green=>Masala=>`
3	`"White"`	`White=>`	`Green=>Masala=>White=>`

☑️ Done. No newline, everything stays on one line.

🔁 Same Example With Default `print()` (Without `end=...`)

```
for key in tea:
    print(key)
```

📤 Output:

```
Green
Masala
White
```

> Because the default `end='\n'` adds a newline after every `print()`.

🛠️ Tip: Customize Delimiters

You can use any custom symbol or string:

```
for key in tea:
    print(key, end=" 💚 ")
```

🔥 Output:

```
Green 💚 Masala 💚 White 💚
```

Or for a cleaner ending, handle last item separately:

```
for i in range(len(tea)):
    if i != len(tea) - 1:
        print(tea[i], end="=>")
    else:
        print(tea[i])
```

☑️ Output:

```
Green=>Masala=>White
```

☑️ Summary

Parameter	Effect
`end='\n'`	Default behavior (new line)
`end='=>'`	Adds `=>` after each `print()`
`end=' '`	Space-separated printing
Custom logic	For avoiding the last delimiter

☑ Python List Mastery Workbook

▧ Section 1: 30+ List Practice Problems

◇ Beginner (Level 1)

1. Create a list of integers from 1 to 10
2. Find the sum and average of a list
3. Get the maximum and minimum elements
4. Reverse a list without using `.reverse()`
5. Remove duplicates from a list
6. Access the last 3 elements of a list
7. Merge two lists
8. Count how many times a value appears in a list
9. Check if a list is empty
10. Find the index of an element

◇ Intermediate (Level 2)

11. Rotate a list by K elements
12. Flatten a nested list (2D to 1D)
13. Remove all falsy values (None, False, 0, '')
14. Get the second largest number from a list
15. Extract only numbers from a mixed list
16. Find common elements between two lists
17. Find the difference between two lists
18. Group elements into sublists of length N
19. Remove elements at even indexes
20. Convert a string to a list of words and back

◇ Advanced (Level 3)

21. Implement a stack using list
22. Implement a queue using list
23. Find all pairs with a given sum
24. Move all zeros to the end of a list
25. Find longest increasing subsequence (LIS)
26. Find the frequency of each element
27. Split a list into two halves
28. Find all duplicates in a list
29. Replace every element with product of others
30. Check if a list is a palindrome
31. Find the most frequent element in the list
32. Remove elements from list while iterating safely

🛠️ Project/Case Studies Using Lists

🎲 1. Deck of Cards Shuffler

- Create a list representing a 52-card deck
- Shuffle the deck
- Deal 5 cards to 4 players
- Ensure no card is repeated

📊 2. Student Marks Analyzer

- Take a list of student scores
- Calculate average, median, and standard deviation
- Find topper(s)
- Remove outliers (values too far from mean)

🛒 3. Shopping Cart System

- List of items added to cart
- Support quantity update, delete, total cost
- Apply coupon discount to the final total
- Generate bill with breakdown

🍕 4. Pizza Order Tracker

- Store all customer orders as lists
- Each order = [customer_name, pizza_size, toppings, quantity]
- Filter orders based on size/toppings
- Generate statistics (most popular topping)

🗳️ 5. Voting System

- List of candidates and vote counts
- Add vote, remove vote
- Sort candidates by votes
- Declare winner(s) — handle ties

❓ MCQs + Quizzes on Python Lists

🧠 Beginner

1. What is the index of the first element in a list?
    - a) 1
    - b) 0 ☑
    - c) -1
    - d) None
2. What does `my_list[-1]` return?

- a) First element
- b) Error
- c) Last element ☑
- d) Second last element

3. Which method adds an element to end of list?

- a) `add()`
- b) `insert()`
- c) `append()` ☑
- d) `push()`

4. `list('abc')` returns:

- a) `['abc']`
- b) `['a', 'b', 'c']` ☑
- c) `[a, b, c]`
- d) Error

5. How to clear a list completely?

- a) `list.clear()` ☑
- b) `list.remove()`
- c) `del list`
- d) `list.popall()`

Intermediate

6. Which method returns and removes an element?

- a) `get()`
- b) `pop()` ☑
- c) `delete()`
- d) `out()`

7. Which expression gives all but the first element?

- a) `l[:-1]`
- b) `l[1:]` ☑
- c) `l[1]`
- d) `l[0:]`

8. What does `l * 2` do?

- a) Duplicates values ☑
- b) Adds 2 to all elements
- c) Creates matrix
- d) Multiplies all items

9. How to reverse a list `l`?

- a) `l[::-1]` ☑

- b) `rev(l)`
- c) `l.reverse(l)`
- d) `l.reverse(l[::-1])`

10. `l1 = [1, 2]; l2 = l1; l2.append(3)` → value of l1?

- a) `[1, 2]`
- b) `[1, 2, 3]` ☑
- c) `[3]`
- d) Error
