Python Programming
Unit 02 – Lecture 01 Notes
Lists Fundamentals and Comprehensions

Tofik Ali

February 14, 2026

Contents

1 Lecture Overview **2**

2 Core Concepts **2**
2.1 What Is a List? . . . 2
2.2 Creating Lists (Common Patterns) . . . 2
2.3 Indexing and Negative Indexing . . . 2
2.4 Slicing (start:stop:step) . . . 3
2.5 List Methods: Mutating vs Returning New . . . 3
2.6 List Comprehensions (Filter + Transform) . . . 3
2.7 Nested Lists (2D Data) . . . 4

3 Common Pitfalls (and How to Avoid Them) **4**
3.1 Aliasing vs Copy . . . 4
3.2 Modifying While Iterating . . . 4

4 Demo Walkthrough: Student Marks Analyzer **4**
4.1 What the Demo Is Teaching . . . 4
4.2 Suggested Run Steps . . . 5
4.3 Extensions (Optional) . . . 5

5 Quiz and Solutions **5**
5.1 Checkpoint 1 (Mutating vs Returning New) . . . 5
5.2 Checkpoint 2 (Comprehension) . . . 5
5.3 Think-Pair-Share (Readability) . . . 5
5.4 Exit Question (Slicing) . . . 5

6 Practice Exercises (With Solutions) **6**
6.1 Exercise 1: Remove Duplicates (Keep Order) . . . 6
6.2 Exercise 2: Top-3 Without Mutating the Input . . . 6
6.3 Exercise 3: 2D Sum . . . 6

7 Further Reading **6**

1 Lecture Overview

Lists are the first *workhorse* data structure most Python learners use for storing many values in one variable. This lecture focuses on:

- how to create lists and access elements (indexing and slicing),
- how mutation works (methods that change the list in place),
- how to build new lists using list comprehensions, and
- how to avoid common beginner bugs (aliasing and editing while iterating).

The demo script uses a list of marks to compute summary statistics and shows a realistic use of sorting and comprehensions.

2 Core Concepts

2.1 What Is a List?

A **list** is an **ordered** and **mutable** collection:

- **Ordered** means items have positions (index 0, 1, 2, ...).
- **Mutable** means the contents can change after creation (append, insert, remove, sort, etc.).
- Lists allow duplicates and can store mixed types. In practice, keep a list *conceptually uniform* (e.g., a list of marks is all integers).

2.2 Creating Lists (Common Patterns)

```
numbers = [2, 4, 6] # literal
letters = list("python") # from an iterable (string)
nums = list(range(5)) # from a range
items = ["A", 3.5, True] # mixed types (allowed)
empty = [] # empty list
```

The constructor `list(...)` works with any iterable. For example, `list(range(5))` produces `[0, 1, 2, 3, 4]`.

2.3 Indexing and Negative Indexing

Python uses **zero-based indexing**:

```
scores = [80, 92, 75, 88]
first = scores[0] # 80
last = scores[-1] # 88 (negative index counts from the end)
```

If you try to access an index that does not exist, Python raises an `IndexError`. This is good: it prevents silent wrong answers.

2.4 Slicing (start:stop:step)

Slicing returns a **new list** (a copy of the selected region):

```
nums = [1, 2, 3, 4, 5, 6]
nums[1:4] # [2, 3, 4] (stop index 4 is excluded)
nums[::2] # [1, 3, 5] (step = 2)
nums[-3:] # last three elements
```

Typical uses:

- Take the first/last k elements: `data[:3]`, `data[-3:]`.
- Sample regularly (every second item): `data[::2]`.
- Create a shallow copy: `copy = a[:]`.

2.5 List Methods: Mutating vs Returning New

Many list methods **mutate** (modify) the same list object:

Method	Effect
`append(x)`	add one item at the end
`extend(iter)`	add many items from an iterable
`insert(i, x)`	insert at a position
`remove(x)`	remove first occurrence of a value
`pop(i)`	remove and return item at index
`sort()`	sort the list in place

In contrast, `sorted(...)` is a **function** that returns a **new list**:

```
nums = [3, 1, 2]
nums.sort() # nums becomes [1, 2, 3]
new_nums = sorted(nums, reverse=True) # returns a new list
```

Rule of thumb: if you need the original ordering later, prefer `sorted`. If you are done with the old order, `sort()` is fine.

2.6 List Comprehensions (Filter + Transform)

Comprehensions let you build a new list concisely:

```
squares = [x*x for x in range(6)]
even_squares = [n*n for n in nums if n % 2 == 0]
```

Reading tip: say it in English.

> "Take $n * n$ for each n in nums, if n is even."

If a comprehension becomes hard to read (multiple conditions, nested loops), switch to a normal `for` loop for clarity.

2.7 Nested Lists (2D Data)

A nested list is a list that contains other lists. A common pattern is a 2D matrix or table:

```
matrix = [[1, 2, 3], [4, 5, 6]]
matrix[1][2] # 6 (row 1, col 2)
```

Important pitfall: avoid creating a matrix with list multiplication like `[[0]*3]*2`. That creates aliases of the same inner list. Prefer:

```
rows, cols = 2, 3
matrix = [[0 for _ in range(cols)] for _ in range(rows)]
```

3 Common Pitfalls (and How to Avoid Them)

3.1 Aliasing vs Copy

Assignment does **not** copy a list; it creates a new name pointing to the same list object:

```
a = [1, 2, 3]
b = a # alias (same object)
b.append(99)
a # [1, 2, 3, 99]
```

To create an independent list:

```
b = a.copy() # or: b = a[:]
```

3.2 Modifying While Iterating

Removing items while iterating can skip elements because indices shift. Two safe patterns are:

- build a new list using a filter (comprehension), or
- iterate over a copy (`for x in a[:]`) when appropriate.

4 Demo Walkthrough: Student Marks Analyzer

Script: `demo/list_marks_analyzer.py`

4.1 What the Demo Is Teaching

The demo ties together list fundamentals in a realistic task:

- compute summary values (average),
- rank values (top-3 using sorting and slicing),
- filter values (passing marks using a comprehension).

4.2 Suggested Run Steps

```
python demo/list_marks_analyzer.py
```

Ask students to predict outputs *before* running the script. Prediction forces them to reason about list operations, not just watch results.

4.3 Extensions (Optional)

- Compute the median mark (hint: sort a copy first).
- Count pass/fail and print percentages.
- Print the top-3 marks with ranks (1st, 2nd, 3rd).

5 Quiz and Solutions

This section contains the interactive checkpoints from the lecture, plus clear solutions.

5.1 Checkpoint 1 (Mutating vs Returning New)

Question. Which of these returns a new list instead of mutating? `append`, `sort`, `sorted`.
Solution. `sorted` returns a *new* list. Both `append` and `sort` mutate the existing list.

5.2 Checkpoint 2 (Comprehension)

Question. Write a list comprehension to square only even numbers from `nums = [1,2,3,4,5,6]`
Solution.

```
nums = [1, 2, 3, 4, 5, 6]
result = [n*n for n in nums if n % 2 == 0]
# result = [4, 16, 36]
```

5.3 Think-Pair-Share (Readability)

Prompt. When is a **for-loop** clearer than a list comprehension?
Sample response. A loop is clearer when the logic has multiple steps, needs debugging prints, or has side effects. For example, validating inputs and collecting errors is usually clearer as a loop than as a complex comprehension.

5.4 Exit Question (Slicing)

Question. What slice returns the last three items of a list `data`?
Solution. `data[-3:]`.

6 Practice Exercises (With Solutions)

6.1 Exercise 1: Remove Duplicates (Keep Order)

Task. Given `data`, create a new list that removes duplicates but keeps the first-occurrence order.
Solution.

```
def unique_keep_order(data):
    seen = set()
    out = []
    for x in data:
        if x not in seen:
            seen.add(x)
            out.append(x)
    return out
```

6.2 Exercise 2: Top-3 Without Mutating the Input

Task. Write a function that returns the top-3 values of a list without changing the original list.
Solution.

```
def top3(values):
    return sorted(values, reverse=True)[:3]
```

6.3 Exercise 3: 2D Sum

Task. Given a matrix (list of lists), compute the sum of all elements.
Solution.

```
def sum_matrix(matrix):
    total = 0
    for row in matrix:
        for x in row:
            total += x
    return total
```

7 Further Reading

Official Python docs are short and practical:

- `https://docs.python.org/3/tutorial/datastructures.html`