From, First Principle: Python

Aryan Singh

25/08/2025

What is a List?

A **list** is an ordered, changeable (mutable) collection of items. It can hold numbers, strings, booleans, or even other lists.

- Syntax: `[1][2][3]`, `["a", "b"]`, `[1, "two", True]`
- Keeps data together (scores, names, tasks), and supports adding, removing, searching, and sorting.

Creating Lists

- Empty: `a = []`
- With items: `nums = [10]`
- From iterable: `list("hey")  # ['h','e','y']`
- Repeated values: `zeros =  * 5  #`
- Nested (independent rows):

Listing 1: Creating independent inner lists

```
rows = [[0 for _ in range(3)] for _ in range(2)]
# Avoid: rows = [*3]*2  # inner lists are shared (bug-prone)
```

Accessing Items

- Indexing: `a` (first), `a[8]` (second)
- Negative indexing: `a[-1]` (last), `a[-2]` (second to last)
- Slicing (returns a new list): `a[start:stop:step]`
- Examples: `a[1:4]`, `a[:3]`, `a[3:]`, `a[::2]`

Mutability (Changing Lists)

Listing 2: Replacing, inserting, deleting

```
a =[1][2][3][4]
a = 99                  # replace one item[9]
a[1:3] =        # replace a slice (size may change)[7][8][9]
del a                   # delete by index
del a[::2]                 # delete a slice (every 2nd element)
```

Common List Methods

Adding

- `append(x)` add to end
- `extend(iterable)` add many at once
- `insert(i, x)` insert at position `i`

Removing

- `pop([i])` remove & return item (default last)
- `remove(x)` delete first matching value
- `clear()` remove everything

Finding/Counting

- `index(x[, start[, end]])` position of first `x` (error if missing)
- `count(x)` number of occurrences

Ordering

- `sort(key=None, reverse=False)` in-place sort (stable)
- `reverse()` in-place reverse

Copy

- `copy()` shallow copy (top level only)
- Also: `a[:]` or `list(a)`

Important: Many mutating methods return `None`. Do not do `b = a.sort()`. Use `a.sort()`, or `b = sorted(a)` for a new list.

Built-in Functions That Work With Lists

`len(a)`, `sum(a)`, `min(a)`, `max(a)`, `sorted(a)`, `reversed(a)`, `any(a)`, `all(a)`, `enumerate(a)`, `zip(a, b)`.

Loops With Lists

Listing 3: Common loop patterns

```
a =[10]
for x in a:
    print(x)                        # values

for i, x in enumerate(a):
    print(i, x)                     # index + value

for x in reversed(a):
    print(x)                        # reverse order (does not modify a)
```

List Comprehensions

Use when transforming or filtering to produce a new list.

Listing 4: Comprehension patterns

```
squares = [x*x for x in range(6)]
evens   = [x for x in range(10) if x % 2 == 0]
matrix  = [, , ][2][4][5][6][1][10]
flat    = [y for row in matrix for y in row]  # flatten one level
```

Sorting

Listing 5: In-place vs copy; custom keys

```
words = ["Banana", "apple", "cherry"]
words.sort()                          # modifies words
b = sorted(words)                     # new sorted list

words.sort(key=str.lower)             # case-insensitive
words.sort(key=len, reverse=True) # by length, descending

# Stable: ties keep original order
people = [("Ann", 30), ("Bob", 25), ("Ann", 20)]
people.sort(key=lambda t: (t, t))  # multi-key[8]
```

Copying Safely

Listing 6: Shallow vs deep copy

```
a = [, ][8][9]
b = a.copy()         # shallow; shares inner lists
a.append(9)       # b sees the change

import copy
c = copy.deepcopy(a)  # deep; fully independent
```

Searching and Membership

Listing 7: Membership and safe index

```
data =[3][4][5][1]
print(3 in data)             # True

try:
    pos = data.index(2)
except ValueError:
    pos = -1
```

Performance Quick Facts

- `append()` and `pop()` at the end: fast (average O(1))
- Insert/remove in middle or front: slower (O(n))
- `sort()`: about O(n log n), optimized for real data (stable)
- Need fast front operations? Consider `collections.deque`

Common Pitfalls

- `b = a.sort()` gives `None`; use `a.sort()` or `sorted(a)`
- Modifying a list while iterating can skip items; build a new list instead:

Listing 8: Safe in-place filtering with slicing

```
a =[4][5][6][1][2][3]
a[:] = [x for x in a if x % 2 == 0]   # keep evens
```

Mini Practice (AP CS Style)

1. Return the 3 largest numbers from a list without changing the original.
2. Given a list of words, keep only those that start with a vowel, sorted by length.
3. Flatten a list of lists and remove duplicates while keeping the first occurrence.
4. Rotate a list right by `k` positions.
5. Given test scores, drop the lowest and return the average of the rest.

Sample Solutions (Short)

Listing 9: Top-3 without modifying original

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

Listing 10: Words starting with vowels, sorted by length

```
def vowel_words(words):
    vowels = set("aeiouAEIOU")
    keep = [w for w in words if w and w in vowels]
    return sorted(keep, key=len)
```

Listing 11: Flatten and deduplicate (keep first)

```
def flatten_dedup(lst_of_lsts):
    flat = [y for row in lst_of_lsts for y in row]
    seen, out = set(), []
    for x in flat:
        if x not in seen:
            seen.add(x)
            out.append(x)
    return out
```

Listing 12: Rotate right by k

```
def rotate_right(a, k):
    n = len(a)
    if n == 0: return a
    k %= n
    return a[-k:] + a[:-k]
```

Listing 13: Average after dropping the lowest

```
def avg_drop_lowest(scores):
    if len(scores) <= 1:
        return 0.0 if not scores else float(scores)
    s = sorted(scores)
    kept = s[1:]
    return sum(kept) / len(kept)
```