Unit 4 Lesson 1C: Finding & Organizing Lists

Quick Reference Guide

🔍 Finding Elements

The `in` Keyword (Review)

Checks IF something exists in a list. Returns `True` or `False`.

```
playlist = ["Blinding Lights", "Bad Guy", "Levitating"]

if "Bad Guy" in playlist:
    print("Found it!")    # This prints!
```

`index()` - Find the Position

Purpose: Returns the index (position) of an element.

```
playlist = ["Blinding Lights", "Bad Guy", "Levitating"]

position = playlist.index("Bad Guy")
print(position)    # 1
```

Key Points:

- Returns the **FIRST** occurrence if duplicates exist
- **CRASHES** with `ValueError` if item not found!

```
scores = [85, 90, 85, 92, 85]
print(scores.index(85))    # 0 (first occurrence)
```

⚠️ Safe Pattern for `index()`

Always check with `in` before using `index()`!

```
# ✅ SAFE - Check first!
if song in playlist:
    position = playlist.index(song)
    print(f"Found at: {position}")
else:
    print("Not found")

# ❌ DANGEROUS - Could crash!
position = playlist.index(song)    # ValueError if not found!
```

🔢 Counting Elements

`count()` - Count Occurrences

Purpose: Returns how many times a value appears in the list.

```
votes = ["Pizza", "Tacos", "Pizza", "Sushi", "Pizza"]

pizza_votes = votes.count("Pizza")
print(pizza_votes)    # 3
```

Key Point: `count()` is SAFE - returns `0` if not found (never crashes!)

```
burger_votes = votes.count("Burger")
print(burger_votes)    # 0 (no error!)
```

`index()` vs `count()` Safety Comparison

Method	If Not Found	Safe?
`index()`	**CRASHES** with ValueError	❌ Need `in` check
`count()`	Returns `0`	✅ Always safe

📊 Sorting Lists

`sort()` Method - Modifies Original

Purpose: Sorts the list in place (ascending order by default).

```
scores = [85, 92, 78, 95, 88]
scores.sort()
print(scores)    # [78, 85, 88, 92, 95]
```

Reverse order:

```
scores.sort(reverse=True)
print(scores)    # [95, 92, 88, 85, 78]
```

⚠️ WARNING:

- `sort()` **modifies the original list**
- `sort()` **returns** `None` - don't assign it!

```
# ❌ WRONG - result will be None!
result = scores.sort()
print(result)    # None

# ✅ CORRECT - just call sort(), then use the list
scores.sort()
print(scores)    # [78, 85, 88, 92, 95]
```

`sorted()` Function - Creates New List

Purpose: Returns a **new** sorted list. Original stays unchanged!

```
scores = [85, 92, 78, 95, 88]
sorted_scores = sorted(scores)

print(scores)         # [85, 92, 78, 95, 88] - unchanged!
print(sorted_scores)  # [78, 85, 88, 92, 95] - new list!
```

Reverse order:

```
sorted_scores = sorted(scores, reverse=True)
```

`sort()` vs `sorted()` Comparison

Feature	`.sort()` method	`sorted()` function
Modifies original?	✅ Yes	❌ No
Returns	`None`	New sorted list
Speed	Faster (no copy)	Slightly slower
Use when	Don't need original order	Need to keep original

When in doubt, use `sorted()` - it's safer!

Reversing Lists

`reverse()` Method - Modifies Original

Purpose: Flips the list backwards in place.

```
playlist = ["Song A", "Song B", "Song C"]
playlist.reverse()
print(playlist)    # ['Song C', 'Song B', 'Song A']
```

⚠️ **WARNING:** Like `sort()`, `reverse()` returns `None`!

```
# ❌ WRONG
result = playlist.reverse()    # result is None!

# ✅ CORRECT
playlist.reverse()
print(playlist)
```

`[::-1]` Slice - Creates New Reversed List

Purpose: Creates a **new** reversed list. Original unchanged!

```
playlist = ["Song A", "Song B", "Song C"]
reversed_playlist = playlist[::-1]

print(playlist)           # ['Song A', 'Song B', 'Song C'] - unchanged!
print(reversed_playlist)  # ['Song C', 'Song B', 'Song A'] - new list!
```

📋 Master Comparison Table

Methods That MODIFY (Return `None`)

```
my_list.sort()        # Sorts in place
my_list.reverse()     # Reverses in place
my_list.append(x)     # Adds to end
my_list.remove(x)     # Removes item
my_list.insert(i, x)  # Inserts at position
my_list.pop()         # Removes and returns last
my_list.clear()       # Removes all items
```

❌ **Don't assign these to a variable!**

Functions/Methods That RETURN Values

```
sorted(my_list)       # Returns new sorted list
my_list[::-1]         # Returns new reversed list
my_list.copy()        # Returns copy of list
my_list.index(x)      # Returns position (int)
my_list.count(x)      # Returns count (int)
len(my_list)          # Returns length (int)
```

✅ **Capture these in a variable!**

🎯 The Big Question

Before using ANY list method, ask yourself:

> "Does this **modify** the original, or **return** something new?"

Modifies (returns None)	Returns New Value
`.sort()`	`sorted()`
`.reverse()`	`[::-1]`
`.append()`	`.copy()`
`.remove()`	`.index()`, `.count()`

💡 Quick Tips

1. **Use `in` before `index()`** - Prevents crashes!
2. **Use `count()` for safe searching** - Never crashes, returns 0
3. **Use `sorted()` when in doubt** - Safer than `sort()`
4. **Never assign `sort()` or `reverse()`** - They return `None`

5. **Strings sort alphabetically** - `['Alice', 'Beth', 'Zara']`
6. **Numbers sort numerically** - `[1, 2, 10, 20]` (not `[1, 10, 2, 20]`)