Modifying, Adding, Removing & Searching Lists

AiCE Software Engineering Foundations — Python Programming

A list that never changes isn't using its most important property — mutability. This lesson covers updating items in place, the three ways to add new items, the four ways to remove them, and how to search a list for what it already contains.

1. Modifying List Items

Because lists are mutable, you can update any item by assigning a new value to its index:

```
students = ["Amara", "Kofi", "Ngozi", "Fatima", "Kwame"]
print(f"Before: {students}")

# Update the item at index 2
students[2] = "Zainab"
print(f"After : {students}")

# Update the last item
students[-1] = "Diallo"
print(f"Final : {students}")
```

Updating a list item is as simple as treating the indexed position like a variable — students[2] = "Zainab" replaces whatever was at index 2 with the new value.

2. Adding Items to a List

.append() — Adding to the End

.append() adds a single item to the end of a list, modifying it in place — no new list is created:

```
cohort = []    # Start with an empty list

cohort.append("Amara Diallo")
cohort.append("Kofi Mensah")
cohort.append("Ngozi Adeyemi")
```

```
print(cohort)
print(f"Cohort size: {len(cohort)}")
```

A closely related pattern collects items from user input inside a loop:

```
african_languages = []
print("Enter 3 African languages:")

for i in range(3):
    language = input(f"Language {i+1}: ").strip().title()
    african_languages.append(language)

print(f"Languages entered: {african_languages}")
```

> **The Loop-Plus-Append Pattern**
> *Starting with an empty list, then appending inside a loop, is one of the most frequently used patterns in all of Python — it's how lists get built from user input, files, databases, and calculated results in nearly every real program.*

.insert() — Adding at a Specific Position

.insert(index, value) adds an item at a specified position, shifting existing items to the right:

```
ranking = ["Nairobi", "Lagos", "Kigali"]

# Insert "Accra" at position 1 (second place)
ranking.insert(1, "Accra")
print(ranking)
# Output: ["Nairobi", "Accra", "Lagos", "Kigali"]
```

.extend() — Adding Multiple Items at Once

.extend() adds all items from another list (or any iterable) to the end of the current list:

```
west_africa = ["Lagos", "Accra", "Dakar"]
east_africa = ["Nairobi", "Kigali", "Addis Ababa"]

west_africa.extend(east_africa)
print(west_africa)
# Output: ["Lagos", "Accra", "Dakar", "Nairobi", "Kigali",
"Addis Ababa"]
```

3. Removing Items from a List

Method	Removes By	Returns the Item?
.remove(value)	First matching value found	No
.pop(index)	Index (last item if omitted)	Yes
del list[index]	Index or slice	No
.clear()	Everything — empties the list	No

.remove() finds the first occurrence of a value and removes it. If the value doesn't exist, it raises a ValueError:

```
cities = ["Lagos", "Accra", "Nairobi", "Accra", "Kigali"]
cities.remove("Accra")    # Removes the first "Accra"
print(cities)
```

.pop() removes the item at a specified index and returns it. If no index is given, it removes and returns the last item:

```
queue = ["Amara", "Kofi", "Ngozi", "Fatima", "Kwame"]

served = queue.pop(0)
print(f"Now serving: {served}")
print(f"Queue      : {queue}")

last = queue.pop()
print(f"Last removed: {last}")
```

> **.pop() Powers Real Queues and Stacks**
> *.pop() is the correct tool for queue and stack implementations — data structures where you retrieve and remove items as they're processed. This is exactly how task queues, job schedulers, and message queues work in real systems.*

del removes an item or a slice without returning the removed value, and .clear() empties the entire list at once:

```
scores = [78, 85, 92, 67, 95, 88]
del scores[2]         # Remove item at index 2
del scores[1:3]       # Remove a slice

temp_data = ["record1", "record2", "record3"]
temp_data.clear()
print(temp_data)      # []
```

4. Searching and Checking Lists

The in keyword, first introduced with strings, checks whether a value exists in a list:

```
enrolled = ["Amara", "Kofi", "Ngozi", "Fatima"]

print("Kofi" in enrolled)       # True
print("Kwame" in enrolled)      # False
print("Kwame" not in enrolled)  # True
```

.index() returns the position of the first occurrence of a value, and .count() counts how many times a value appears:

```
cities = ["Lagos", "Accra", "Nairobi", "Kigali"]
position = cities.index("Nairobi")
print(f"Nairobi is at index {position}")

grades = ["A", "B", "A", "C", "A", "B", "F", "A"]
print(f"Number of A grades: {grades.count("A")}")
```

Key Takeaways

List items update in place through direct index assignment. .append() adds to the end, .insert() adds at a chosen position, and .extend() merges in another list entirely. .remove() deletes by value, .pop() deletes by index and hands the item back, del removes without returning anything, and .clear() wipes the list clean. in, .index(), and .count() let you check what a list already contains without modifying it. The next lesson covers sorting lists, combining them with loops for real data processing, and building lists dynamically from user input.