Period 5 - Unit 4, Lesson 1B Reference: List Operations

KEY CONCEPT: You can handle errors TWO ways - defensive checks OR try/except!

---

insert() - Add at Specific Position

The Problem

```
# append() only adds to the end
queue = ["Alice", "Charlie", "Dave"]
queue.append("Bob")
# Result: ["Alice", "Charlie", "Dave", "Bob"]
# Bob is at the end, not second!
```

The Solution

```
# insert() adds at specific position
queue = ["Alice", "Charlie", "Dave"]
queue.insert(1, "Bob")  # Insert Bob at index 1
# Result: ["Alice", "Bob", "Charlie", "Dave"]
# Bob is second!
```

More Examples

```python
playlist = ["Song A", "Song B", "Song D"]

# Insert at beginning (index 0)
playlist.insert(0, "Intro")
# Result: ["Intro", "Song A", "Song B", "Song D"]

# Insert in middle
playlist.insert(3, "Song C")
# Result: ["Intro", "Song A", "Song B", "Song C", "Song D"]
```

---

extend() - Add Multiple Items

The Annoying Way

```python
# Multiple append() calls
playlist = ["Song A", "Song B"]
playlist.append("Song C")
playlist.append("Song D")
playlist.append("Song E")
# Result: ["Song A", "Song B", "Song C", "Song D", "Song E"]
# But that's 3 lines of code!
```

The Better Way

```python
# extend() adds multiple items at once
playlist = ["Song A", "Song B"]
new_songs = ["Song C", "Song D", "Song E"]
playlist.extend(new_songs)
# Result: ["Song A", "Song B", "Song C", "Song D", "Song E"]
# One line!
```

---

append() vs extend() - THE DIFFERENCE!

append() adds the LIST itself

```python
list1 = ["A", "B"]
list1.append(["C", "D"])
print(list1)
# Output: ['A', 'B', ['C', 'D']]  # NESTED list!
```

extend() adds ITEMS from the list

```python
list2 = ["A", "B"]
list2.extend(["C", "D"])
print(list2)
# Output: ['A', 'B', 'C', 'D']  # FLAT list!
```

Remember: `append()` adds whatever you give it (even a list). `extend()` unpacks the list and adds each item.

---

remove() - Approach 1: Defensive (Check First)

```python
shopping = ["milk", "bread", "eggs"]
item_to_remove = "cheese"

# Check BEFORE removing
if item_to_remove in shopping:
    shopping.remove(item_to_remove)
    print("✅ Removed!")
else:
    print("⚠️ Item not found")

# Output: ⚠️ Item not found
```

Safe Remove Function - Defensive

```python
def safe_remove_defensive(my_list, item):
    """Remove item safely using defensive check."""
    if item in my_list:
        my_list.remove(item)
        return True
    return False

# Test it
shopping = ["milk", "bread", "eggs"]
result = safe_remove_defensive(shopping, "bread")
# Returns True, shopping is now ["milk", "eggs"]

result = safe_remove_defensive(shopping, "cheese")
# Returns False, shopping unchanged
```

---

remove() - Approach 2: Exception Handling (try/except)

```python
shopping = ["milk", "bread", "eggs"]
item_to_remove = "cheese"

# Try to remove, catch ValueError
try:
    shopping.remove(item_to_remove)
    print("✅ Removed!")
except ValueError:
    print("⚠️ Item not found")

# Output: ⚠️ Item not found
```

Safe Remove Function - Exception Handling

```
def safe_remove_exception(my_list, item):
    """Remove item safely using exception handling."""
    try:
        my_list.remove(item)
        return True
    except ValueError:
        return False

# Test it
shopping = ["milk", "bread", "eggs"]
result = safe_remove_exception(shopping, "bread")
# Returns True, shopping is now ["milk", "eggs"]

result = safe_remove_exception(shopping, "cheese")
# Returns False, shopping unchanged
```

Which Approach Should You Use?

Approach 1 - Defensive (Check First)

```
if item in my_list:
    my_list.remove(item)
```

PROS: Clear intent, no exceptions
USE WHEN: You want explicit checks

Approach 2 - Exception Handling (try/except)

```
try:
    my_list.remove(item)
except ValueError:
    pass
```

PROS: Cleaner, more Pythonic
USE WHEN: You expect success most of the time

BOTH ARE VALID! Choose what feels right!

---

pop() - Remove AND Return

```python
# pop() removes last item and returns it
tasks = ["homework", "dishes", "coding"]
last_task = tasks.pop()
print(f"Last task: {last_task}")  # "coding"
print(f"Remaining: {tasks}")      # ["homework", "dishes"]
```

pop(index) - Remove from Specific Position

```python
# pop(index) removes at specific position and returns it
tasks = ["homework", "dishes", "coding", "exercise"]
second_task = tasks.pop(1)
print(f"Second task: {second_task}")  # "dishes"
print(f"Remaining: {tasks}")          # ["homework", "coding", "exercise"]
```

Practical Example 1: Queue Management

```python
queue = ["Alice", "Charlie", "Dave"]
print(f"Original queue: {queue}")

# Bob cuts in line at position 1
queue.insert(1, "Bob")
print(f"After Bob cuts: {queue}")
# ["Alice", "Bob", "Charlie", "Dave"]

# Serve first person
first_person = queue.pop(0)
print(f"Serving: {first_person}")  # "Alice"
print(f"Queue now: {queue}")       # ["Bob", "Charlie", "Dave"]
```

Practical Example 2: Playlist Editor with Safe Remove

```python
playlist = ["Song A", "Song B", "Song C", "Song D"]

# Try to remove a song (defensive approach)
song_to_remove = "Song E"  # Doesn't exist
if song_to_remove in playlist:
    playlist.remove(song_to_remove)
    print(f"Removed {song_to_remove}")
else:
    print(f"⚠️ {song_to_remove} not in playlist")
# Output: ⚠️ Song E not in playlist

# Try to remove a song (exception approach)
song_to_remove = "Song B"  # Exists
try:
    playlist.remove(song_to_remove)
    print(f"✅ Removed {song_to_remove}")
    print(f"Playlist now: {playlist}")
except ValueError:
    print(f"⚠️ {song_to_remove} not in playlist")
# Output: ✅ Removed Song B
#         Playlist now: ['Song A', 'Song C', 'Song D']
```

Practical Example 3: Shopping List Manager

```python
shopping = ["milk", "bread"]
print(f"Starting list: {shopping}")

# Add multiple items
new_items = ["eggs", "cheese", "butter"]
shopping.extend(new_items)
print(f"After adding: {shopping}")
# ["milk", "bread", "eggs", "cheese", "butter"]

# Insert urgent item at beginning
shopping.insert(0, "coffee")
print(f"After urgent add: {shopping}")
# ["coffee", "milk", "bread", "eggs", "cheese", "butter"]

# Remove item we don't need
try:
    shopping.remove("bread")
    print(f"After removing bread: {shopping}")
except ValueError:
    print("Item not found")
# ["coffee", "milk", "eggs", "cheese", "butter"]
```

Key Patterns to Remember

Adding

```python
my_list.append(item)           # Add to end
my_list.insert(index, item)    # Add at specific position
my_list.extend(other_list)     # Add multiple items
```

Removing - Defensive

```python
if item in my_list:
    my_list.remove(item)       # Safe remove
```

Removing - Exception Handling

```
try:
    my_list.remove(item)       # Try to remove
except ValueError:
    pass                       # Handle if not found
```

Remove and Return

```
item = my_list.pop()           # Remove last, get it back
item = my_list.pop(index)      # Remove at index, get it back
```

Common Mistakes to Avoid

❌ Mistake 1: Forgetting append() vs extend()

```
my_list.append([1, 2, 3])  # Creates nested list! ['A', 'B', [1, 2, 3]]
my_list.extend([1, 2, 3])  # Adds 1, 2, 3 separately ✅ ['A', 'B', 1, 2, 3]
```

❌ Mistake 2: Not handling remove() errors

```
my_list.remove(item)  # CRASH if not found!
# Always check first OR use try/except ✅
```

❌ Mistake 3: Confusing remove() and pop()

```
my_list.remove(value)  # Removes by VALUE
my_list.pop(index)     # Removes by INDEX (and returns it)
```

✅ Correct Patterns

Safe remove (defensive):

```python
if item in my_list:
    my_list.remove(item)
```

Safe remove (exception):

```python
try:
    my_list.remove(item)
except ValueError:
    print("Not found")
```

Remove and use the value:

```python
last = my_list.pop()
```

Testing Your Code

```python
def remove_all_occurrences(my_list, item):
    """
    Remove ALL occurrences of an item.
    Shows defensive approach.
    """
    count = 0
    while item in my_list:
        my_list.remove(item)
        count += 1
    return count

if __name__ == "__main__":
    # Test remove_all_occurrences
    test_list = ["A", "B", "A", "C", "A"]
    count = remove_all_occurrences(test_list, "A")

    print(f"Test 1: {'✅ PASS' if test_list == ['B', 'C'] else '❌ FAIL'}")
    print(f"  Expected: ['B', 'C']")
    print(f"  Got: {test_list}")

    print(f"Test 2: {'✅ PASS' if count == 3 else '❌ FAIL'}")
    print(f"  Expected count: 3")
    print(f"  Got: {count}")

    print("\nYou'll see tests like this in all practice problems!")
```

Summary

What You Learned:

- ✅ `insert(index, item)` - Add at specific position
- ✅ `extend(list)` - Add multiple items from another list
- ✅ `remove(value)` - Delete by value (use defensive check OR try/except!)
- ✅ `pop()` - Remove last item AND return it
- ✅ `pop(index)` - Remove at specific position AND return it
- ✅ **TWO approaches** to error handling - BOTH are valid!

Key Takeaway: There's more than one way to solve a problem in Python. Choose what makes sense for your situation!