📋 Unit 4 Reference Sheet — Collections Mastery

📋 List Methods

Adding Elements

```
items.append(x)      # Add ONE item to end
items.insert(i, x)   # Add at index i
items.extend([a,b])  # Add MULTIPLE items
```

Removing Elements

```
items.remove(x)      # Remove first x (ValueError if missing!)
items.pop()          # Remove & return last
items.pop(i)         # Remove & return at index i
items.clear()        # Remove ALL
```

Finding Elements

```
items.index(x)       # Index of first x
items.count(x)       # Count occurrences
x in items           # True/False check
```

⚠️ Always check `if x in items` before using `remove()` or `index()`!

✂️ List Slicing

Syntax: `list[start:stop:step]`

```
nums = [0, 1, 2, 3, 4, 5]

nums[1:4]    # [1, 2, 3]  (stop excluded!)
nums[:3]     # [0, 1, 2]  (first 3)
nums[3:]     # [3, 4, 5]  (from index 3)
nums[:]      # [0, 1, 2, 3, 4, 5]  (copy!)
nums[::2]    # [0, 2, 4]  (every 2nd)
nums[::-1]   # [5, 4, 3, 2, 1, 0]  (reversed)
```

🔄 Sort vs Sorted

```
nums.sort()          # Modifies original, returns None
new = sorted(nums)   # Returns NEW list, original unchanged
```

💡 Use `sorted()` when you need to keep the original!

📋 Unit 4 Reference Sheet — Page 2

🚀 List Comprehensions

Basic Transform

```
[expression for item in list]

# Example:
[n * 2 for n in [1, 2, 3]]  # [2, 4, 6]
```

Filter (if at END) — Removes items

```
[expression for item in list if condition]

# Example:
[n for n in nums if n > 0]  # Only positives
```

Transform (if-else at START) — Changes all

```
[A if condition else B for item in list]

# Example:
["Pass" if n >= 70 else "Fail" for n in scores]
```

if at END = FILTER (shorter list) **if-else at START = TRANSFORM (same length)**

🗂️ Dictionary Operations

Basic Access

```
player = {"name": "Alex", "level": 5}

player["name"]          # "Alex"
player["hp"]            # KeyError!
player.get("hp")        # None (safe!)
player.get("hp", 100)   # 100 (default)
```

Modifying

```
player["xp"] = 500      # Add/update key
del player["level"]     # Remove key
player.update({"a": 1}) # Merge dicts
```

Iterating

```
for key in player:              # Keys only
for val in player.values():     # Values only
for k, v in player.items():     # Both!
```

💡 Use `.get(key, default)` to avoid KeyError crashes!

📋 Unit 4 Reference Sheet — Page 3

📦 Tuples — Immutable

Creating & Unpacking

```
point = (10, 20)
x, y = point          # x=10, y=20

# Multiple returns from functions
def get_stats(nums):
    return sum(nums), len(nums), sum(nums)/len(nums)

total, count, avg = get_stats([1, 2, 3])
```

⚠️ Tuples CANNOT be modified after creation!

When to Use Tuples

- Multiple return values
- Dictionary keys (lists can't be keys!)
- Data that shouldn't change

🔵 Sets — Unique Values

Creating Sets

```
colors = {"red", "blue", "red"}  # {"red", "blue"}
empty = set()                     # NOT {}!
unique = set([1, 1, 2, 2, 3])    # {1, 2, 3}
```

Set Operations

```
a = {1, 2, 3}
b = {2, 3, 4}

a | b    # Union: {1, 2, 3, 4}
a & b    # Intersection: {2, 3}
a - b    # Difference: {1}
```

Fast Membership

```
if "red" in colors:   # Very fast!
```

💡 Use sets when you need unique items or fast lookups!

📋 Unit 4 Reference Sheet — Page 4

🗂️ Nested Data (List of Dicts)

Accessing Nested Data

```
students = [
    {"name": "Alice", "grade": 92},
    {"name": "Bob", "grade": 85}
]

students[0]             # First dict
students[0]["name"]     # "Alice"
students[1]["grade"]    # 85
```

Looping Through

```
for student in students:
    print(student["name"])
```

Comprehension with Nested Data

```
# Get all names
names = [s["name"] for s in students]

# Filter by grade
passing = [s for s in students if s["grade"] >= 70]

# Get names of passing students
passing_names = [s["name"] for s in students
                 if s["grade"] >= 70]
```

⚡ Common Patterns

Safe Access with Defaults

```
# Dictionary
player.get("health", 100)

# Nested
value = data.get("outer", {}).get("inner", default)
```

Aggregation

```
total = sum(item["price"] for item in items)
count = len(items)
average = total / count if count else 0
```

Counting by Category

```
counts = {}
for item in items:
    cat = item["category"]
    counts[cat] = counts.get(cat, 0) + 1
```

Handling Empty Lists

```
def process(items):
    if not items:
        return 0  # or [] or {}
    # ... rest of code
```

🔑 Quick Syntax Reminders

Task	Syntax
Check if empty	`if not items:`
Sum values	`sum(items)`

Task	Syntax
Reverse list	`items[::-1]`
Every nth item	`items[::n]`

Task	Syntax
Count items	`len(items)`
Get dict values	`dict.values()`
Safe dict access	`dict.get(key, default)`
Copy a list	`items[:]` or `items.copy()`

Task	Syntax
First n items	`items[:n]`
Check membership	`x in items`
Unique values	`set(items)`
Dict key-values	`dict.items()`

References vs Copies:

- `new = old` → SAME list (reference)
- `new = old[:]` → COPY (independent)

Comprehension Position Rule:

- `if` at END → FILTER (removes items)
- `if-else` at START → TRANSFORM (changes all items)