Recall Week 6 - Lists and Tuples

You should be able to:

- Discuss the key characteristics of lists and tuples
- Index and slice list and tuple elements
- Apply Python methods associated with lists and tuples
- Perform standard and nested loops on lists and tuples
- Assess membership and perform operations on lists and tuples

Week 7 - Sets and Strings

You should be able to:

- Discuss the key characteristics of sets & strings & hashability
- Index and slice strings
- Apply Python methods associated with sets and strings
- Perform standard loops on sets and strings
- Assess membership and perform operations on lists and tuples

Hashable Objects

An object in Python is **hashable** if:

Its contents don't change so its identity can be stored as a hash value (int) and quickly looked up in data structures like **sets** and **dictionary keys**.

Most immutable objects are hashable.

Hashable Objects

Data structures like **sets** and **dict keys** are implemented as **hash tables**.

They work by:

- Taking each element's **hash value** (an integer)
- Using that value to decide **where** to store it in memory
- Using the hash again later to **find** it quickly

Hashable Objects

Only immutable objects are hashable but not all immutable objects are hashable.

Why Not?

Hashable Objects

Only immutable objects are hashable

Type	Mutable?	Hashable?	Can be a dict key or set member?
`int`	❌ No	✅ Yes	✅ Yes
`float`	❌ No	✅ Yes	✅ Yes
`str`	❌ No	✅ Yes	✅ Yes
`tuple`	❌ No (if elements hashable)	✅ Yes	✅ Yes
`frozenset`	❌ No	✅ Yes	✅ Yes
`list`	✅ Yes	❌ No	❌ No
`dict`	✅ Yes	❌ No	❌ No
`set`	✅ Yes	❌ No	❌ No

Sets

- Mutable
- Unordered - cannot be indexed and sliced
- Unique - no duplicate items
- Immutable items (hashable - int, float, str, tuple)
- Iterable - can loop over
- 17 associated built-in methods

Frozen Sets

- Immutable (hashable)
- Unordered - cannot be indexed and sliced
- Unique - no duplicate items
- Only include immutable items (not list, dict, set)
- Iterable - can loop over
- 17 associated built-in methods

Sets vs Frozen Sets

- Use a **regular set** if you need a dynamic collection of unique elements that may change over time.

- Use a **frozenset** when you need a fixed, unchangeable set of unique values — especially when you want to use it as a key or store it inside another set.

Sets vs Frozensets

Use Cases:

- Remove duplicates
- Dataset manipulation & comparisons
- Membership testing - data validation, search
- Unique counts
- Frozensets - quick look up, use as dictionary keys, etc.

Strings

Sequences of characters:

- Immutable & hashable
- Ordered - can be indexed and sliced
- Iterable
- Support multiple languages and symbols
- 47 built-in dot methods

HW7 Q2 - sorting

```
for i in range(n - 1):  # loop will be sorted after n-1 passes

    for j in range(n - 1 - i): # compare and swap adjacent elements\
```

Here's a visualization if `n = 5`:

Outer `i`	Inner loop range (`range(n - 1 - i)`)	Comparisons made
0	`range(4)` → j = 0,1,2,3	Compare pairs [0-1], [1-2], [2-3], [3-4]
1	`range(3)` → j = 0,1,2	Compare pairs [0-1], [1-2], [2-3]
2	`range(2)` → j = 0,1	Compare pairs [0-1], [1-2]
3	`range(1)` → j = 0	Compare pair [0-1]

HW7 Q2 - sorting

Sort [5,2,4,1]

Pass 1 (i=0, range(3))

```
(5,2) → swap → [2,5,4,1]
(5,4) → swap → [2,4,5,1]
(5,1) → swap → [2,4,1,5]
```

Pass 2 (i = 1, range = 2))

```
(2,4) → OK
(4,1) → swap → [2,1,4,5]
```

Pass 3 (i=2, range(1))

```
(2,1) → swap → [1,2,4,5]
```

HW7 Q2 - sorting

```
for i in range(n - 1):  # loop will be sorted after n-1 passes
    for j in range(n - 1 - i): # compare and swap adjacent elements\
```

Concept	Explanation
Outer loop (`i`)	Counts how many passes to make through the list.
Inner loop (`j`)	Compares neighboring elements and swaps if needed.
`n - 1`	Prevents `j + 1` from going past the end.
`- i`	Avoids rechecking sorted elements at the end.
Result	Each pass "bubbles" the largest unsorted element to the end.