LISTS AND TUPLES

Lists allow you to store multiple pieces of information—as many as you want—in one place. Once you've defined a list, you can access and work with individual items, a section of the list, or all the information in the list at once.

Understanding how lists work gives you a solid foundation for working with more complex data types. These cards cover lists and tuples, which are similar to lists.

LISTS

- What is a list?
- How do you define a list?
- How do you add items to a list?

A *list* is a collection of items in a specific order. Square brackets indicate a list.

You access list items through their position in the list, or *index*. The first item is always index zero; a negative index accesses the list from the other end:

```
>>> vowels = ['e', 'i', 'o']
>>> vowels[0]
'e'
>>> vowels[-1]
'o'
```

Use `append()` to add single items to the end of a list:

```
>>> vowels.append('u')
>>> vowels
['e', 'i', 'o', 'u']
```

Use `insert()` to insert items at any position using the index of that position:

```
>>> vowels.insert(0, 'a')
>>> vowels
['a', 'e', 'i', 'o', 'u']
```

Modify an element in a list using its index:

```
>>> vowels[-1] = 'y'
>>> vowels
['a', 'e', 'i', 'o', 'y']
```

This replaces the last item with `y`.

REMOVING ITEMS FROM LISTS

- How do you remove an item from a list?
- How do you remove an item at a certain position from a list?

The `remove()` method removes a specific item from a list:

```
>>> airports = ['ANC', 'JNU', 'SEA', 'SIT']
>>> airports.remove('SEA')
>>> airports
['ANC', 'JNU', 'SIT']
```

If an item appears more than once in the list, only its first appearance is removed.

The `del` keyword deletes an item at a specific position in the list using its index:

```
>>> airports = ['ANC', 'JNU', 'SEA', 'SIT']
>>> del airports[2]
>>> airports
['ANC', 'JNU', 'SIT']
```

The `pop()` method removes and returns the last item in a list. The `pop()` method can also accept an index and will remove and return the item at that index:

```
>>> airports = ['ANC', 'JNU', 'SEA', 'SIT']
>>> airport = airports.pop()
>>> airport
'SIT'
>>> airport = airports.pop(0)
>>> airport
'ANC'
```

SLICING A LIST

- What is a slice?
- How do you make a slice from the beginning of a list?
- How do you make a slice at the end of a list?

A *slice* is part of a list that begins with the item at the first index specified and stops with the item before the last index specified:

```
>>> states = ['CA', 'CO', 'CT', 'DE', 'FL', 'GA']
>>> states[2:4]
['CT', 'DE']
```

Omit the first index, and the slice starts with the first item in the list:

```
>>> states[:2]
['CA', 'CO']
```

Omit the second index, and the slice goes to the end of the list:

```
>>> states[3:]
['DE', 'FL', 'GA']
```

You can construct a slice that returns any part of a list:

First two items	`states[:2]`
Last two items	`states[-2:]`
Range of items	`states[2:4]`

COPYING A LIST

- What can you do with a copy of a list?
- How do you make a copy of a list?

A copy of a list lets you work with the contents of the copied list without affecting the original list.

Make a copy of a list by slicing while omitting both indexes. This generates a slice from the start to the end of the list.

```
>>> states = ['CA', 'CO', 'CT', 'DE']
>>> my_states = states[:]
>>> my_states
['CA', 'CO', 'CT', 'DE']
```

Changes you make to the new list don't affect the original list:

```
>>> my_states.append('HI')
>>> my_states
['CA', 'CO', 'CT', 'DE', 'HI']
>>> states
['CA', 'CO', 'CT', 'DE']
```

LOOPING THROUGH LISTS

- What is a `for` loop?
- How do you loop through a list?
- How do you loop through a section of a list?

A for loop lets you work with each item in a list, one item at a time.

Use a for loop to loop through all items in a list:

```
countries = ['CAN', 'CHL', 'IND', 'AUS']
for country in countries:
    print(f"I'm flying to {country}.")
```

```
I'm flying to CAN.
I'm flying to CHL.
I'm flying to IND.
I'm flying to AUS.
```

Loop through part of a list using a slice:

```
for country in countries[:2]:
    print(f"I've been to {country}.")
```

```
I've been to CAN.
I've been to CHL.
```

SORTING LISTS

- How do you sort a list temporarily?
- How do you sort a list permanently?

The `sorted()` function returns a copy of a list in a natural sorted order. The order of the original list is not affected:

```
>>> vowels = ['a', 'i', 'e', 'u', 'o']
>>> sorted(vowels)
['a', 'e', 'i', 'o', 'u']
>>> vowels
['a', 'i', 'e', 'u', 'o']
```

To sort a list, use the `sort()` method:

```
>>> vowels.sort()
>>> vowels
['a', 'e', 'i', 'o', 'u']
```

To put a numerical list in order, use any sorting function:

```
>>> nums = [1, 3, 2, 5]
>>> sorted(nums)
[1, 2, 3, 5]
```

REVERSE SORTING LISTS

- How do you sort a list in reverse alphabetical order?
- How do you reverse a list's order?

To sort a list in reverse natural order, use the `reverse` argument set to `True`:

```
>>> vowels = ['a', 'i', 'e', 'u', 'o']
>>> sorted(vowels, reverse=True)
['u', 'o', 'i', 'e', 'a']
```

The `reverse` argument works with `sort()` and `sorted()`.

To reverse the original order of a list, use the `reverse()` method:

```
>>> vowels = ['a', 'i', 'e', 'u', 'o']
>>> vowels.reverse()
>>> vowels
['o', 'u', 'e', 'i', 'a']
```

The `reverse` argument to `sort()` or `sorted()` results in reverse natural order; the `reverse()` method reverses the original order of the list.

NUMERICAL LISTS

- How do you make a simple list of numbers?
- How do you make a list of numbers that follows a specific pattern?

The `range()` function generates a series of numbers. Giving `range()` one argument returns a series of numbers from 0 up to, not including, the number:

```
>>> nums = list(range(5))
>>> nums
[0, 1, 2, 3, 4]
```

Pass two values to create a range that starts with the first value and ends at one less than the second value:

```
>>> high_nums = list(range(95, 100))
>>> high_nums
[95, 96, 97, 98, 99]
```

Use a third argument to specify a distance to skip between numbers in a list:

```
>>> odds = list(range(1, 10, 2))
>>> odds
[1, 3, 5, 7, 9]
```

To define a pattern, make an empty list and then modify each number in a range. This lists the powers of 10:

```
nums = []
for exp in range(5):
    nums.append(10**exp)
print(nums)
```

```
[1, 10, 100, 1000, 10000]
```

LIST COMPREHENSIONS

- What is a list comprehension?
- How do you use a list comprehension?

A *list comprehension* is a compact way of defining a list in one line:

```
>>> nums = [10**exp for exp in range(5)]
>>> nums
[1, 10, 100, 1000, 10000]
```

A list comprehension uses square brackets, an expression that generates each item in the list, and a `for` loop as the basis for the list.

A list comprehension can modify each item in an existing list:

```
>>> states = ['CA', 'CO', 'CT']
>>> lower_states = [state.lower()
...   for state in states]
>>> lower_states
['ca', 'co', 'ct']
```

TUPLES

- What is a tuple?
- How do you define a tuple?
- How do you work with a tuple?

A *tuple* is an ordered collection of items that can't be modified. It is usually indicated by parentheses:

```
>>> elementary_grades = (2, 3, 4)
```

Elements in a tuple are accessed using indexes:

```
>>> elementary_grades[0]
2
>>> elementary_grades[-1]
4
>>> elementary_grades[:2]
(2, 3)
```

You can loop through a tuple using a `for` loop:

```
elementary_grades = (2, 3, 4)
for grade in elementary_grades:
    print(f"Welcome to grade {grade}!")
```

```
Welcome to grade 2!
Welcome to grade 3!
Welcome to grade 4!
```

If you try to modify an element in a tuple, you'll get an error. Tuples are useful when you have a collection of items that shouldn't change throughout the life of your program.