Week 7: Collections

Objectives

- To describe the difference between scalar and collection value types
- To be able to create list literals
- To be able to create and manage lists dynamically

Lesson - Collections of things

Up until now all of the variable values we've looked at have been `scalars`. That means that each name we create to represent a value can only represent a single value at a time. There are many situations where using a single name to represent multiple values at once can be quite useful. We call the broad class of types that can hold multiple values at once `collections`. Python has several built-in collection types, we'll be looking specifically at the `list` type.

In Python we can create a list literally by providing the initial values upfront:

```
month_temperatures = [47, 49, 52, 57, 63, 66, 72, 72, 67, 59, 51, 46]

print(type(month_temperatures))
print(month_temperatures)
```

Notice that the values in the list are in a particular order. We can read a value in a particular position by using square brackets. Notice that the first position is `0` not `1`.

```
print(month_temperatures[0])
print(month_temperatures[6])
```

We can find out how many items are in a list with the `len()` function:

```
count = len(month_temperatures)
print(f'There are {count} items')
```

In Python, a list can contain values of different types:

```
heterogeneous_list = [6, 4.5, 'cat', True]
heterogeneous_list[1] = 'dog'

print(heterogeneous_list)
```

The real power of lists is when they are not static or literal, but dynamic. When we don't know how many things will be in the list, then we have to build it dynamically. We can add a new value to the end of the list with `append()`.

```
my_list = []
my_list.append(5)
my_list.append(10)

print(my_list)
```

Many times we will want to go through the items in a list. We've already learned about loops, and here is one way to walk through a list:

```
my_list = [1, 2, 3, 4, 5]

current_index = 0
length_of_list = len(my_list)

while current_index < length_of_list:
  print(my_list[current_index] * 2)
  current_index = current_index + 1
```

This is very cumbersome, and something that occurs very frequently. Luckily there is a better way. When we want to loop through a list (or other collection) we can use a `for` loop. This is a `determinate` loop:

```
my_list = [2, 4, 6, 8]
for item in my_list:
  print(item * 2)
```

Reverse the list

Prompt the user to enter the names of their brothers, sisters, and cousins. When they are done (by pressing `enter` without a string) print out the names in reverse order.

Excercise: Find the average

Prompt the user to enter numbers, collecting each number as it is typed. When the user presses enter without a number, find the average of the numbers they entered.