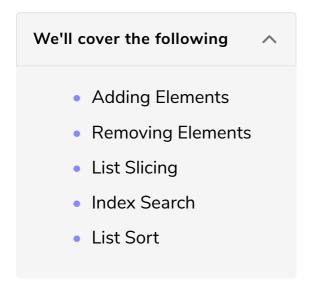
Common List Operations

In this lesson, we'll take a look at some of the properties and utilities that come with the list data structure.



Adding Elements

All the elements of a list cannot always be specified beforehand and there's a strong possibility that we'll want to add more elements during runtime.

The append() method can be used to add a new element at the end of a list. The following template must be followed:

```
a_list.append(newElement)
```

Here's an example:

```
num_list = [] # Empty list
num_list.append(1)
num_list.append(2)
num_list.append(3)
print(num_list)
```

Note: In the code above, we create an empty list at **line 1**. This can always be done by simply using empty square brackets [].

To add an element at a particular index in the list, we can use the insert()
method. We'll use it in the following format:

```
aList.insert(index, newElement)
```

If a value already exists at that index, the whole list from that value onwards will be shifted one step to the right:

```
num_list = [1, 2, 3, 5, 6]
num_list.insert(3, 4) # Inserting 4 at the 3rd index. 5 and 6 shifted ahead
print(num_list)
```

Removing Elements

Deleting elements is as easy as adding them. The counterpart of append() is the pop() operation which removes the last element from the list.

We can store this popped element in a variable:

```
houses = ["Gryffindor", "Hufflepuff", "Ravenclaw", "Slytherin"]
last_house = houses.pop()
print(last_house)
print(houses)
```

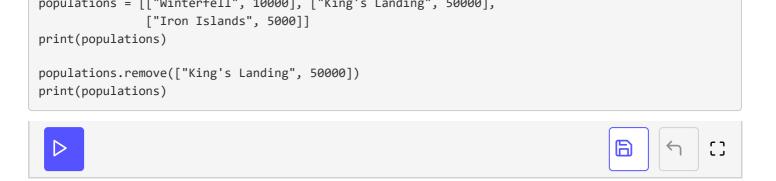
If we need to delete a particular value from a list, we can use the remove() method
by following this template:

```
aList.remove(element_to_be_deleted)
```

Let's see it in action:

```
houses = ["Gryffindor", "Hufflepuff", "Ravenclaw", "Slytherin"]
print(houses)
houses.remove("Ravenclaw")
print(houses)

# For nested lists
```



List Slicing

A list obeys all the rules of slicing that we studied earlier.

Slicing a list gives us a sublist:

```
num_list = [1, 2, 3, 4, 5, 6, 7, 8]
print(num_list[2:5])
print(num_list[0::2])
```

Index Search

With lists its really easy to access a value through its index. However, the opposite operation is also possible where we can find the index of a given value.

For this, we'll use the index() method:

```
cities = ["London", "Paris", "Los Angeles", "Beirut"]
print(cities.index("Los Angeles")) # It is at the 2nd index
```

If we just want to verify the existence of an element in a list, we can use the in operator:

```
cities = ["London", "Paris", "Los Angeles", "Beirut"]
print("London" in cities)
print("Moscow" not in cities)
```

List Sort

A list can be sorted in ascending order using the sort() method. Sorting can be done alphabetically or numerically depending on the content of the list:

```
num_list = [20, 40, 10, 50.4, 30, 100, 5]
num_list.sort()
print(num_list)
```

There are several other list methods which we haven't explored. They can be found in the official documentation for Python3.

Next, we'll examine the key features of the **tuple** data structure.