Lists

Lesson Duration: 45 minutes

Learning Objectives

- Explain what lists are
- Create a list
- Access an element in a list
- Add items to a list
- Call list methods

Collections

We've been dealing a lot with singular data - a person's name, a number, a boolean value. However, once we start working with data we realise that the majority of the cases will involve working with larger sets of data - and with more complexity on top of that.

In our next sessions we will introduce you to two important data structures - Lists and Dicts. Let's start with Lists!

A list is part of a subset of objects known as "collections". It is essentially a storage container similar to a row of lockers - items in it are ordered and assigned an index number to identify them by. We will talk about these a bit later!

Just like lockers, we might find different things in these collections - like strings, booleans, numbers, or any combinations of these. Python is quite liberal when it comes to datatypes in collections, however, it does not mean it's a good idea to mix and match - in an ideal world, we will try to keep identical datatypes in a single List!

What are lists for

If we can store large collections of data in a single variable, we can use these to pass them around and simplify the running of our programs. Apart from storing values, soon we will see that we can run codeblocks with each individual item - as long as they're in a list, effectively automating the process.

What Are They Not For

Lists are fairly basic and simple - essentially all they know is that they have a certain amount of items inside them. This means that the list is not going to know what value is inside itself, so the only way to find an item inside a list is to go through each individual one and check if it is the one we were looking for.

This is not as much of a problem as it sounds like - the purpose of the list is to act as a storage for items, and we can check individual items with loops.

Accessing Lists

Each item in a list have a corresponding key - or `index`. These indexes are integers, starting from zero. Certain languages limit how big a list or array can be, but Python automatically assigns new memory space if the number of items in our list grows.

Let's see in practice how they work! Create a new folder in our `couch_to_coder` directory called `session_2`. In there, we will create a new file called `lists.py`.

We are going to work with some fruits for now.

```
fruits = ['apple', 'mango', 'pomegranate', 'melon']
print(fruits)
```

Lists are characterised by the square brackets around the elements. Each item needs to be separated by a `,`.

List/array indexes in most programming languages start at 0. More info here for the curious.

In order to access items from a list, we need to add square brackets at the end of the list or the variable holding the list, and pass in an index number. Due to indexes starting at 0, each item will have an index corresponding with its position minus 1.

```
# 1st item
print(fruits[0])

# 3rd item
print(fruits[2])
```

If we try to access an element that doesn't exist, we'll get an `IndexError`.

```
print(fruits[1000])
# Resulst in IndexError
```

One of the coolest features of Python is allowing us to use negative indexes - this will start accessing items from the end of the list, `-1` giving us the last item.

```
print(fruits[-1])
print(fruits[-2])
```

Updating an Item

We can update an item in a list by accessing an element at a particular index and giving it a new value, just like re-assigning a variable.

```
fruits[1] = 'dragonfruit'
print(fruits)
```

Be aware that this replaces the original value.

Creating an Empty List

There will be times where you want to initialize an empty list. We can do this by directly instantiating a list using `[]` or by using the `list()` function.

```
# lists.py

my_list = []
my_list = list()
```

Getting the Number of Items in a List

We often want to know how many items are there in a list. We can check this using the built-in `len()` function, and passing in the list's variable.

```
length_of_fruits = len(fruits)
print(length_of_fruits)
```

Adding and Removing Items

There are a number of ways to add or remove items from a list.

If we take a look at Python's List documentation, we can see what methods there are for these actions.

Most commonly, we will add or remove items directly from the end of the list - remember, we do not care about the order of items, we should treat lists as a black box with no visibility.

```
fruits.append('pear')
print(fruits)
```

Removing an item from the end is also fairly easy.

```
fruits.pop()
print(fruits)
```

This can also give us the removed item, in case we want to store it in a variable for later use.

```
removed_fruit = fruits.pop()
print(removed_fruit)
```

List Elements

Elements can be *any* type of object; literal values or variables. Unlike other languages, Python allows us to put a mixture of different kind of objects in an list.

```
fruits_and_numbers = [ 'apple', 1, 'grape', 2 ]
print(fruits_and_numbers)
```

We can even put lists inside of lists.

```
nested_list = [ 1, 2, 3, 4, [5, 6, 7] ]
print(nested_list)
```

This should be avoided - if there is no consistency in data, we will struggle to treat each item the same.

Conclusion

We have seen how we can store collections of objects using lists and that we can add or remove items to and from lists.