Lists

This lesson highlights the key features of the list data structure.

We'll cover the following

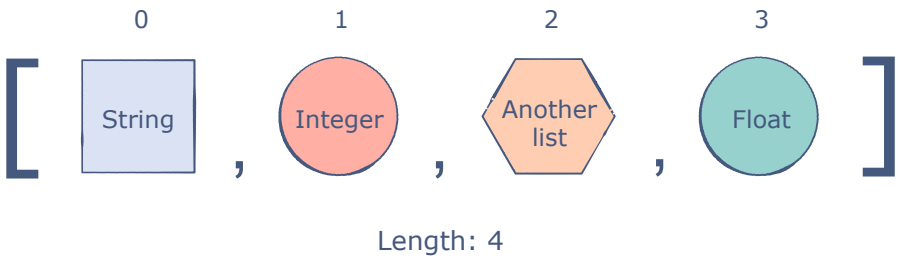
- Structure
- Creating a List
  - Using range()
- List-Ception!
  - Sequential Indexing
- Merging Lists

Structure

The list is perhaps the most commonly used data structure in Python. It allows us to store elements of different data types in one container.

The contents of a list are enclosed by square brackets, `[]`.

As we've already seen, lists are *ordered*, like strings. Elements are stored linearly at a specific **index**.



We can see from the illustration above that a list bears resemblance to a string.

A string was a collection of characters indexed in a linear fashion. A list is the same except that it can contain any type of data, even another list!

Creating a List

We already know how to create a list using square brackets, but here's a refresher.

```
jon_snow = ["Jon Snow", "Winterfell", 30]
print(jon_snow)

# Indexing
print(jon_snow[0])

# Length
print(len(jon_snow))
```

The beauty of lists lies in the fact that we are not bound to one type of data.

Lists are mutable, which further expands their functionality:

```
jon_snow = ["Jon Snow", "Winterfell", 30]
print(jon_snow[2])
jon_snow[2] += 3
print(jon_snow[2])
```

Using `range()`

A range can further be converted into a list by using the `list()` casting, which we'll explore further in the future.

Here's an example of a range being used to create a list of numbers:

```
num_seq = range(0, 10)  # A sequence from 0 to 9
num_list = list(num_seq)  # The list() method casts the sequence into a list
print(num_list)

num_seq = range(3, 20, 3)  # A sequence from 3 to 19 with a step of 3
print(list(num_seq))
```

List-Ception!

Here's an example of lists inside another list:

```
world_cup_winners = [[2006, "Italy"], [2010, "Spain"],
                     [2014, "Germany"], [2018, "France"]]
```

```
print(world_cup_winners)
```

The nested lists do not even need to be of the same size! This is not something we can find in many other languages.

Sequential Indexing

To access the elements of a list or a string which exists inside another string, we can use the concept of sequential indexing.

Each level of indexing takes us one step deeper into the list, allowing us to access any element in a complex list.

All we have to do is specify all the indices in a sequence:

```
world_cup_winners = [[2006, "Italy"], [2010, "Spain"],
                     [2014, "Germany"], [2018, "France"]]
print(world_cup_winners[1])
print(world_cup_winners[1][1])  # Accessing 'Spain'
print(world_cup_winners[1][1][0])  # Accessing 'S'
```

Merging Lists

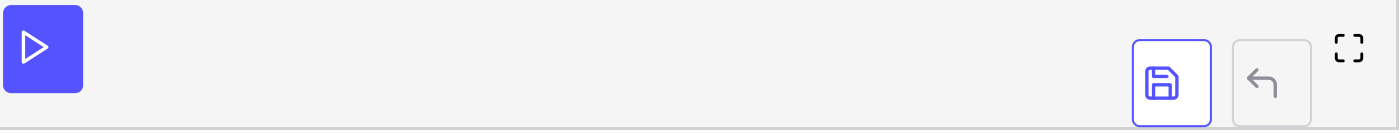
Python makes it really easy to merge lists together. The simplest way is to use the `+` operator like we did for strings:

```
part_A = [1, 2, 3, 4, 5]
part_B = [6, 7, 8, 9, 10]
merged_list = part_A + part_B
print(merged_list)
```

Alternatively, we could use the `extend()` property of a list to add the elements of one list at the end of another:

```
part_A = [1, 2, 3, 4, 5]
part_B = [6, 7, 8, 9, 10]
part_A.extend(part_B)
print(part_A)
```

```
print(part_A)
```



We've just scratched the surface of the list data structure.

In the next lesson, we'll explore other properties of lists and some common list operations.