Working with Lists

Lists are probably the most common data structure used in Scala. In the following lesson, you will learn how to create a List.

We'll cover the following

Introduction

A **List** in Scala is a collection which comes under the `Seq` class. It is an immutable collection and hence when modified the original List does not get updated, rather, a new List is created.

Like Arrays and ArrayBuffers, Lists store elements of the same type.

Creating a List

Lists can be created and populated in multiple different ways, most of which are identical to the approaches we went over when discussing Arrays. Let's quickly go over them.

constructor | range | fill

```
val fruitList = List("orange", "banana", "apple", "grape")

// Driver Code
fruitList.foreach(println)
```

Constructing Lists Using `::` and `nil`

`::` is a method, known as **cons**, which takes two arguments. The first argument is the head and is a single element. The second argument is a tail which is another List. `nil` is used to represent an empty List and is always used when constructing a List with `::`.

```
element_x::element_xs
```

This represents a List whose first element is `x` and the rest of the elements are the elements of another List `xs`. Hence, the `fruitList` in the code above could have also been defined as follows:

```
orange::(banana::(apple::(grape::Nil)))
```

When using this method in code, we can actually remove the parenthesis `()`.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
val fruitList = "orange"::"banana"::"apple"::"grape"::Nil

// Driver Code
fruitList.foreach(println)
```

We can also combine multiple methods to create a single List. The example below will use a List constructor combined with `::`.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
val fruitList = "orange"::List("banana","apple","grape")

// Driver Code
fruitList.foreach(println)
```

> Remembers, as Lists are immutable, each time we need to modify them, we need to create a new List.

Appending Elements

While appending elements to the end of a List is not common practice, it is still allowed and can be done using the `:+` method. Let's add another fruit to our `fruitList`.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
val fruitList = "orange"::"banana"::"apple"::"grape"::Nil
val fruitList2 = fruitList :+ "peach"

// Driver Code
fruitList2.foreach(println)
```

Prepending Elements

There are multiple ways to prepend an element to the start of a List. The most common method is using *cons* `::`.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
val fruitList = "orange"::"banana"::"apple"::"grape"::Nil
val fruitList2 = fruitList :+ "peach"
val fruitList3 = "watermelon"::fruitList2

// Driver Code
fruitList3.foreach(println)
```

Prepending an element to the start of a List can also be done using the `+:` method.

This code requires the following environment variables to execute:

LANG C.UTF-8

```
val fruitList = "orange"::"banana"::"apple"::"grape"::Nil
val fruitList2 = fruitList :+ "peach"
val fruitList3 = "watermelon"::fruitList2
val fruitList4 = "mango" +: fruitList3

// Driver Code
fruitList4.foreach(println)
```

List Concatenation

List concatenation is the merging of two Lists. This is great if you want to add multiple elements to a list. You can create a new List of the elements to be added and simply merge the new List with the old one.

List concatenation is done using the `:::` method.

This code requires the following environment variables to execute:

LANG C.UTF-8

```
val fruitList = "orange"::"banana"::"apple"::"grape"::Nil
val fruitList2 = fruitList :+ "peach"
val fruitList3 = "watermelon"::fruitList2
val fruitList4 = "mango" +: fruitList3
val twoFruits = "pear"::"apricot"::Nil
val fruitList5 = twoFruits ::: fruitList4

// Driver Code
fruitList5.foreach(println)
```

Head & Tail

To get the head and tail of a list or any collection for that matter, we use the `head` and `tail` method.

> The head and tail methods don't have parameters. When a method has zero parameters, it is called using `method()` syntax. But when a method doesn't have parameters, it is called using `method` syntax.

This code requires the following environment variables to execute:

LANG C.UTF-8

```
val fruitList = "orange"::"banana"::"apple"::"grape"::Nil
val fruitList2 = fruitList :+ "peach"
val fruitList3 = "watermelon"::fruitList2
val fruitList4 = "mango" +: fruitList3
val twoFruits = "pear"::"apricot"::Nil
val fruitList5 = twoFruits ::: fruitList4

val getHead = fruitList5.head
val getTail = fruitList5.tail

// Driver Code
println(getHead)
println(getTail)
```

In Scala, the tail does not refer to the last element, rather, it refers to the remaining list after the head. This is why when we called the tail method above, we got the complete list excluding the head element, i.e., `pear`.

ListBuffer

List's mutable counterpart collection is **ListBuffer**. It works exactly like a List but is used when multiple modifications are required as it updates the existing List instead of creating a new one. It is common practice to convert a ListBuffer into a List when you are done manipulating the elements.

In the next lesson, you will be challenged to append an element to a List.