Getting Started with Vectors

In the following lesson, you will be introduced to Vectors.

We'll cover the following

Introduction

Like Lists, Vectors are another immutable collection of sequence type. They differ in the fact that Lists are linear while Vectors are indexed. What this means is that if you want to access a central element in a List, the compiler has to start from the beginning and linearly progress through the List until it finds the element you are trying to access. However, when you want to access a central element in a Vector, the compiler can directly go to that index and access the element in constant time.

While Lists should be used when you only plan to manipulate the head element (first element), Vectors provide efficient access when you want to manipulate elements beyond the head.

Creating a Vector

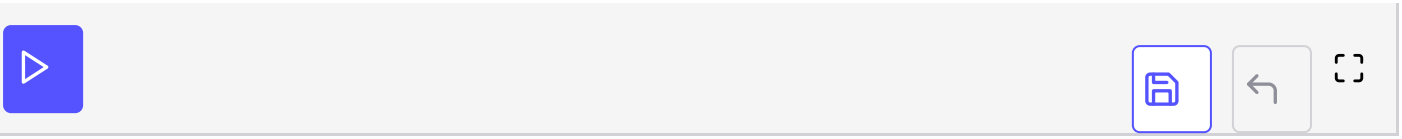
Vectors can be created using the `Vector` keyword followed by `()`.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
val numVector = Vector(1,2,3,4,5)
```

```
numVector.foreach(println)
```



Initializing an Empty Vector

If you don't want to populate your vector during declaration, you can initialize an empty vector using the `empty` method. The `empty` method doesn't take any arguments.

```
val emptyVector = Vector.empty
```

Accessing an Element

You can access an element by mentioning the Vector name followed by the index of the element in `()`.

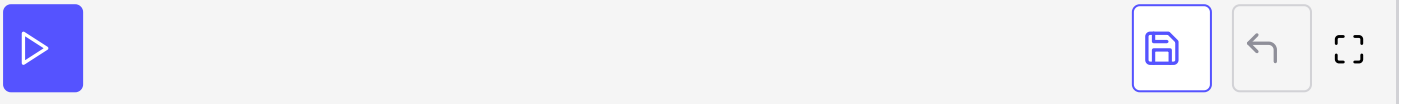
`vectorName(index of element)`

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
val patternVector = Vector("a~a", "b~b", "c~c")
val pattern = patternVector(1)

println(pattern)
```



Appending an Element

Like Lists, you can append elements to a Vector using the `:+` method.

Let's add more patterns to `patternVector`.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
val patternVector = Vector("a~a", "b~b", "c~c")
val patternVector2 = patternVector :+ "d~d"
```

```
// Driver Code
patternVector2.foreach(println)
```

Prepending an Element

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

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
val patternVector = Vector("a~a", "b~b", "c~c")
val patternVector2 = patternVector :+ "d~d"
val patternVector3 = "1~1" +: patternVector2

// Driver Code
patternVector3.foreach(println)
```

Vector Concatenation

Vector concatenation is the merging of two Vectors. Like Lists, this is great if you want to add multiple elements to a Vector. You can create a new Vector of the elements to be added and simply merge the new Vector with the old one.

Vector concatenation is done using the `++` method.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
val patternVector = Vector("a~a", "b~b", "c~c")
val patternVector2 = patternVector :+ "d~d"
val patternVector3 = "1~1" +: patternVector2
val tempVector = Vector("e~e","f~f")
val patternVector4 = patternVector3 ++ tempVector

// Driver Code
patternVector4.foreach(println)
```

Let's now move on to the collection `Range`.