Lists

Introduction

- List is one of the most commonly used data structure in the Scala programming language.

List Literals

- We can create new List literals by writing the word List followed by parentheses with the list values inside.

```
...
val colors = List('yellow','brown','blue','green','black')
val numbers = List(1,20,22,12,82,8,4)
...
```

List of Lists

- We can create a List that each one of its elements is another List.

```
...
val myLists = List(
    List('rehovot','tel-aviv','haifa','jerusalem','eilat'),
    List('new york','boston','washington','san francisco'),
    List('london','yorkshair','manchester')
      )
...
```

Empty List

- We can explicitly create an empty list that doesn't include any element.

```
...
val myLists = List()
...
```

The List Type

- When creating a new list we can specify the type it uses. We should specify it within square brackets.

```
...
val myRecs:List[Rectangle] =
    List(  Rectangle(3,2),Rectangle(4,6),Rectanlge(8,2)   )
...
val myStrings:List[String] = List(“jane”,”dave”,”mosh”)
...
```

- Lists are homogeneous. The elements of each list should be of the same type.

List Operations

- List supports working with the following operations:

  `head`

  This operation returns the first element.

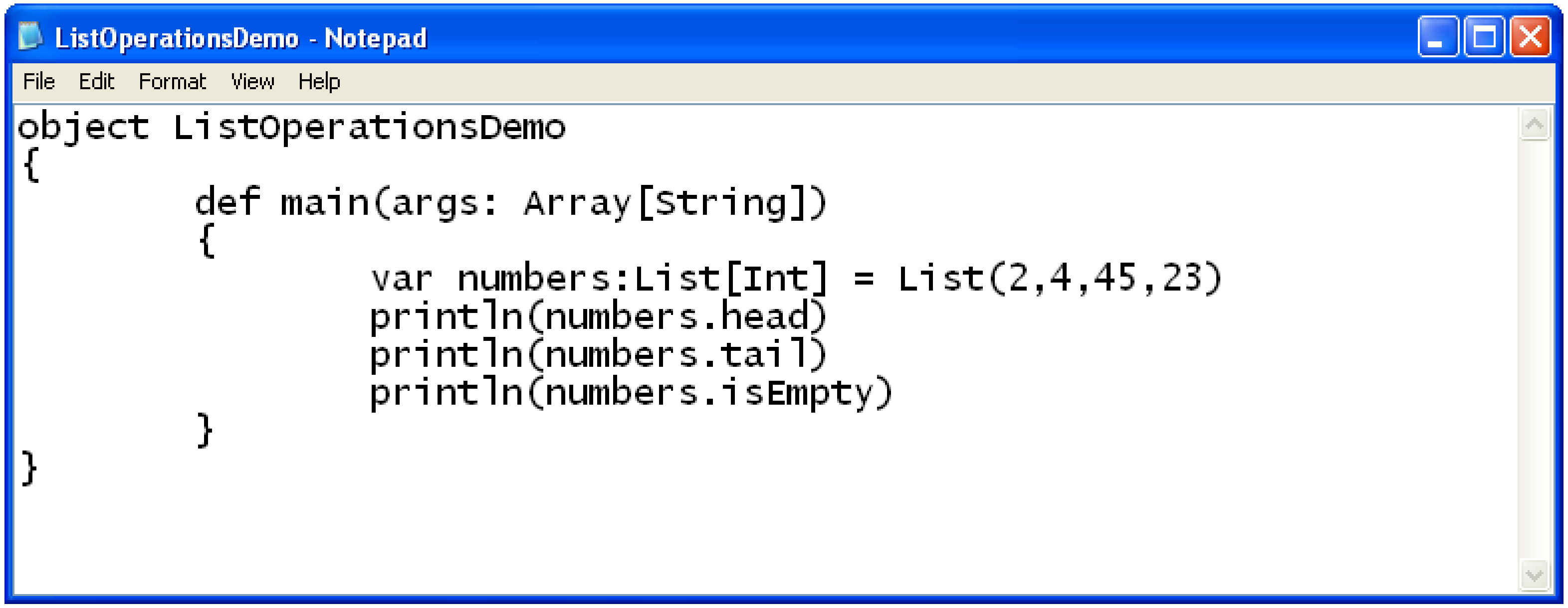
  `tail`

  This operation returns a list of the elements coming after the head.

  `isEmpty`

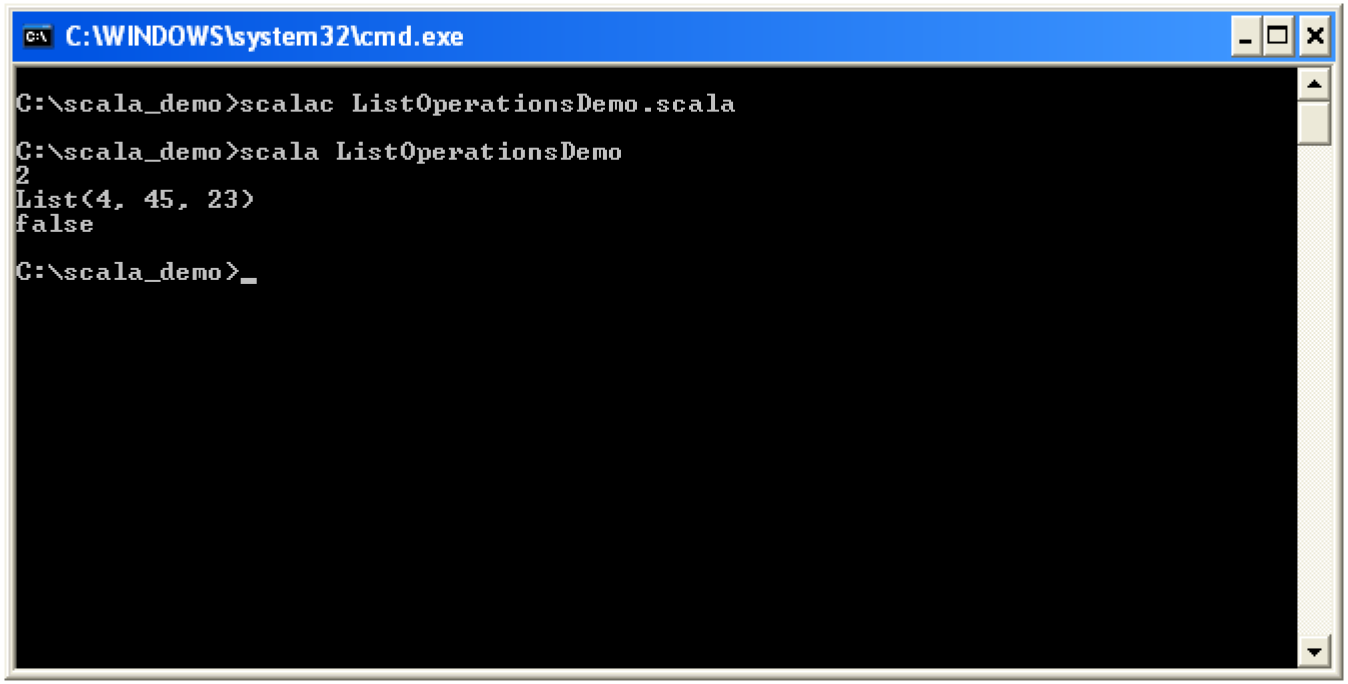
  This operation returns true if the list is empty.

List Operations



```
object ListOperationsDemo
{
        def main(args: Array[String])
        {
                var numbers:List[Int] = List(2,4,45,23)
                println(numbers.head)
                println(numbers.tail)
                println(numbers.isEmpty)
        }
}
```

List Operations



```
C:\scala_demo>scalac ListOperationsDemo.scala

C:\scala_demo>scala ListOperationsDemo
2
List(4, 45, 23)
false

C:\scala_demo>_
```

List Transformation

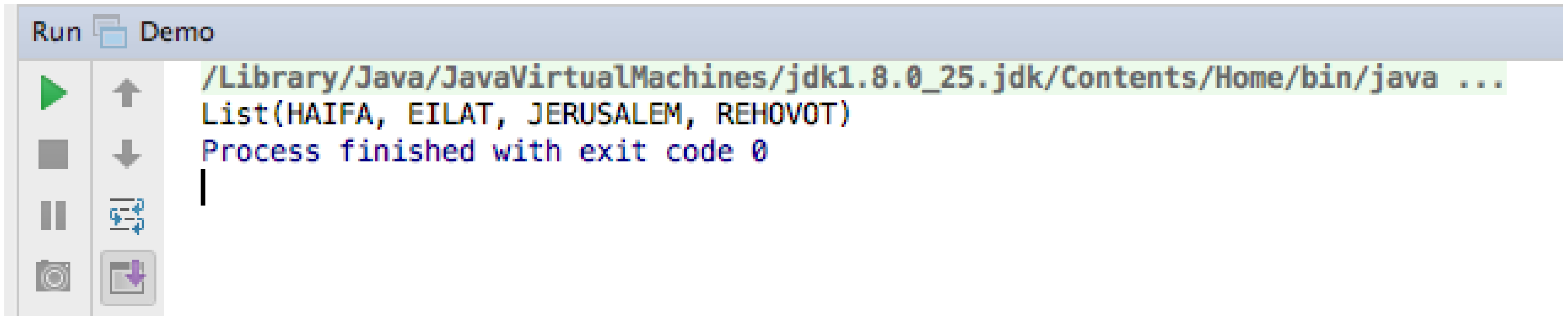
- The `map` method we can invoke on `List` and on `Seq`, transforms each and every element using the function we passed over.

List Transformation

```
package com.lifemichael.samples

object Demo
{
  def main(args:Array[String]):Unit =
  {
    val listA:List[String] = List("haifa","eilat","jerusalem","rehovot")
    val listB:List[String] = listA.map(str=>str.toUpperCase)
    print(listB)

  }
}
```



```
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java ...
List(HAIFA, EILAT, JERUSALEM, REHOVOT)
Process finished with exit code 0
```

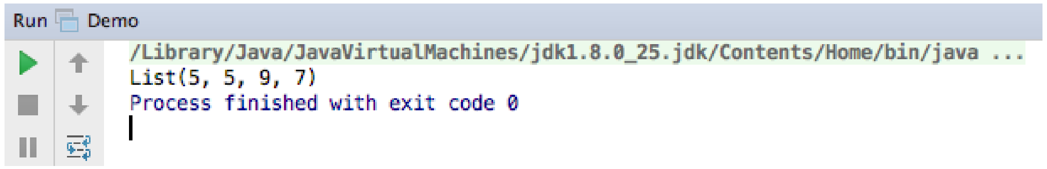
List Transformation

- The `map` method we can invoke on `List` and on `Seq`, can receive a function that transforms the elements into a new value of a different type.

List Transformation

```
package com.lifemichael.samples

object Demo
{
  def main(args:Array[String]):Unit =
  {
    val listA:List[String] = List("haifa","eilat","jerusalem","rehovot")
    val listB:List[Int] = listA.map(_.length)
    print(listB)
  }
}
```



```
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java ...
List(5, 5, 9, 7)
Process finished with exit code 0
```

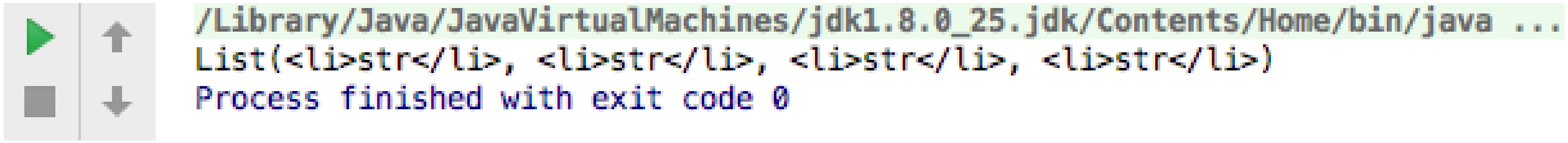
List Transformation

- We can use the `map` function in order to transform a list of strings into a list of HTML elements.

List Transformation

```scala
package com.lifemichael.samples

object Demo
{
  def main(args:Array[String]):Unit =
  {
    val listA:List[String] = List("haifa","eilat","jerusalem","rehovot")
    val listB = listA.map(str => <li>str</li>)
    print(listB)
  }
}
```



```
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java ...
List(<li>str</li>, <li>str</li>, <li>str</li>, <li>str</li>)
Process finished with exit code 0
```

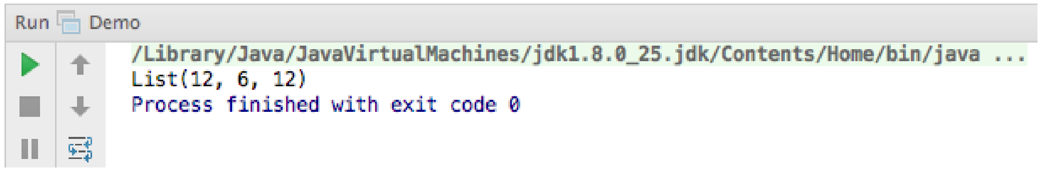
List Filtering

- The `filter` method we can invoke on `List` returns a new `List` object that includes just those elements that passed the test described by the function passed over.

List Filtering

```
package com.lifemichael.samples

object Demo
{
  def main(args:Array[String]):Unit =
  {
    val listA:List[Int] = List(12,31,41,5,6,12)
    val listB:List[Int] = listA.filter(number=>number%3==0)
    print(listB)
  }
}
```



```
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java ...
List(12, 6, 12)
Process finished with exit code 0
```

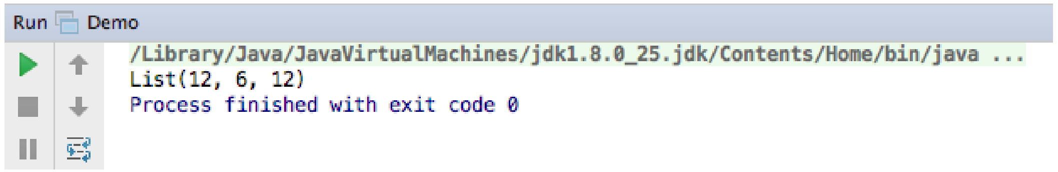
List Filtering

- The function passed over doesn't have to be an anonymous one. We can use a function we define with a meaningful name.

List Filtering

```
package com.lifemichael.samples

object Demo
{
  def isEven(num:Int):Boolean = num%2==0
  def main(args:Array[String]):Unit =
  {
    val listA:List[Int] = List(12,31,41,5,6,12)
    val listB:List[Int] = listA.filter(isEven)
    print(listB)
  }
}
```



```
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java ...
List(12, 6, 12)
Process finished with exit code 0
```

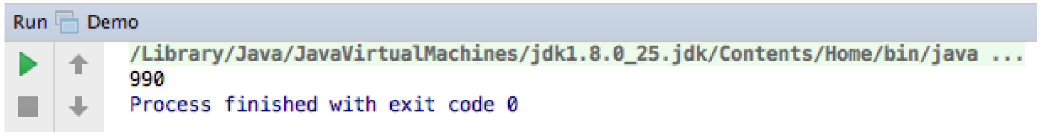
List Redeuce

- The `redeuceLeft` function we can invoke on a List object allows us to perform an operation on adjacent elements of the collection where the result of the first operation is fed into the next one.
- The `redeuceRight` function does the same. Just in the opposite direction.

List Reduce

```
package com.lifemichael.samples

object Demo
{
  def max(numA:Int,numB:Int):Int = if(numA>numB) numA else numB
  def main(args:Array[String]):Unit =
  {
    val listA:List[Int] = List(12,5,23,45,67,7,9,990,7,5,34,233,324)
    val num = listA.reduceLeft(max)
    print(num)
  }
}
```



```
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java ...
990
Process finished with exit code 0
```

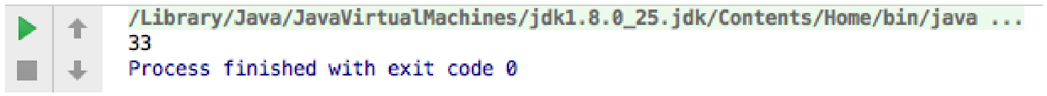
List Folding

- The `foldLeft` function works similarly to `redeuceLeft.` It starts with a seed value that is sent to the function we pass over as the second argument. The returned value is passed over as the first argument in the second invocation of our function, the second argument is the next value. Each time that function is invoked its second argument is the specific value of the list the `foldLeft` focuses on.
- The returned value is the accumulated result calculated based on the seed value.

List Folding

```
package com.lifemichael.samples

object Demo
{
  def max(numA:Int,numB:Int):Int = if(numA>numB) numA else numB
  def main(args:Array[String]):Unit =
  {
    val listA:List[Int] = List(3,4,5,6,7,8)
    val number = listA.foldLeft(0)((a,b)=>a+b)
    print(number)
  }
}
```

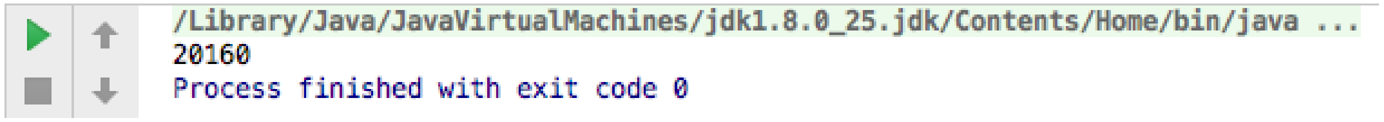


```
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java ...
33
Process finished with exit code 0
```

List Folding

```
package com.lifemichael.samples

object Demo
{
  def max(numA:Int,numB:Int):Int = if(numA>numB) numA else numB
  def main(args:Array[String]):Unit =
  {
    val listA:List[Int] = List(3,4,5,6,7,8)
    val num = listA.foldLeft(1)((a,b)=>a*b)
    print(num)
  }
}
```

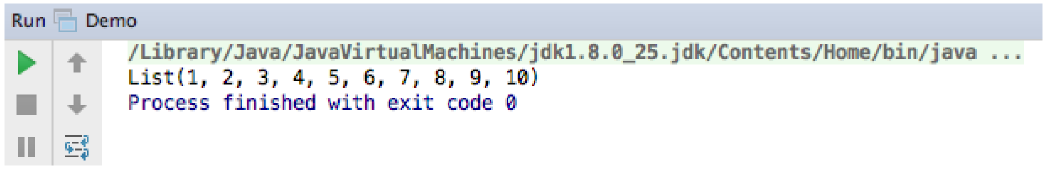


```
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java ...
20160
Process finished with exit code 0
```

Simple List Generating

- We can easily generate a list that holds numbers in a specific range.

```
object Demo {
  def main(args: Array[String]) {
    val numbers:List[Int] = (1 to 10).toList
    print(numbers)
  }
}
```



```
/Library/Java/JavaVirtualMachines/jdk1.8.0_25.jdk/Contents/Home/bin/java ...
List(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
Process finished with exit code 0
```