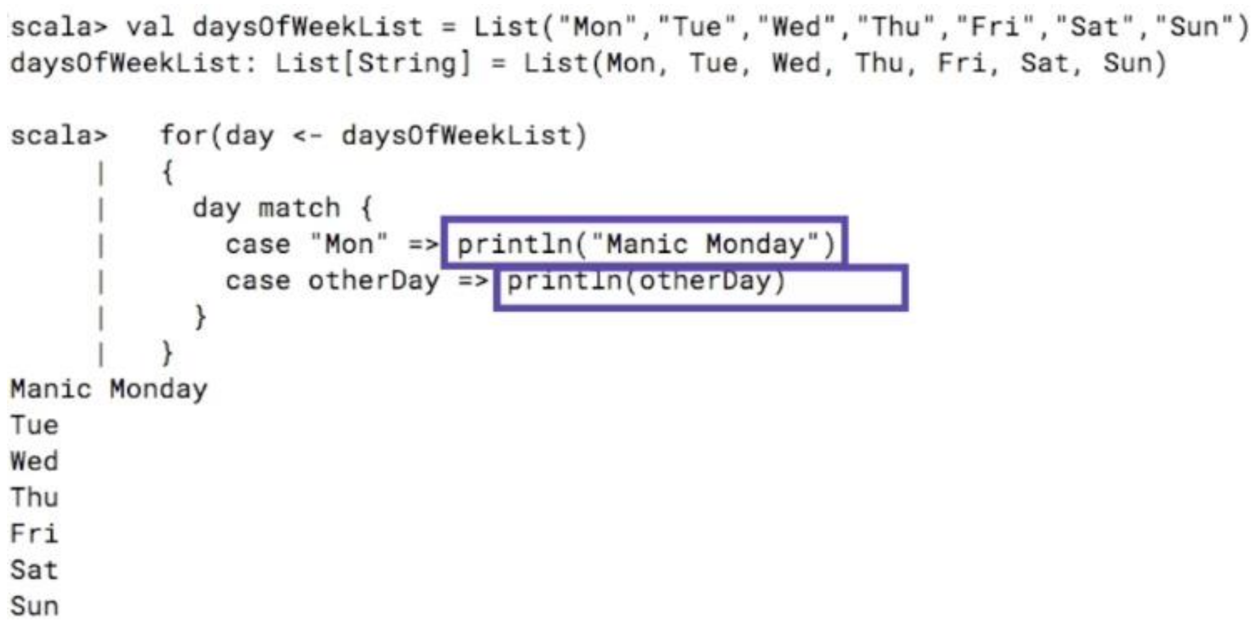
Scala

Expression doesnot return value

Statement return value

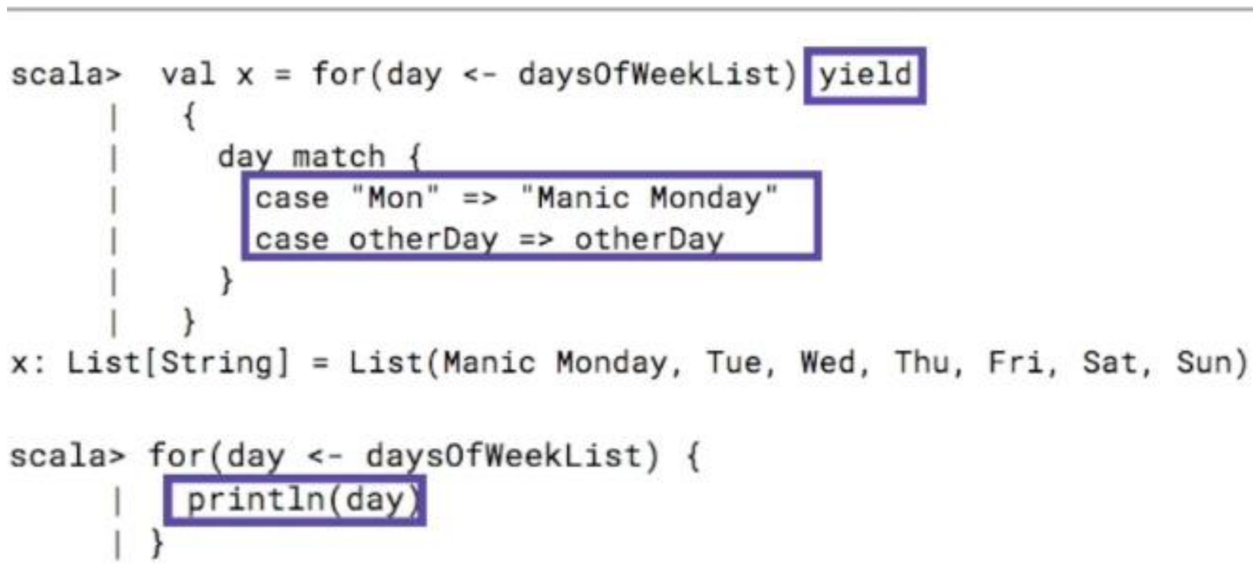
FOR LOOP EXAMPLE

Statement

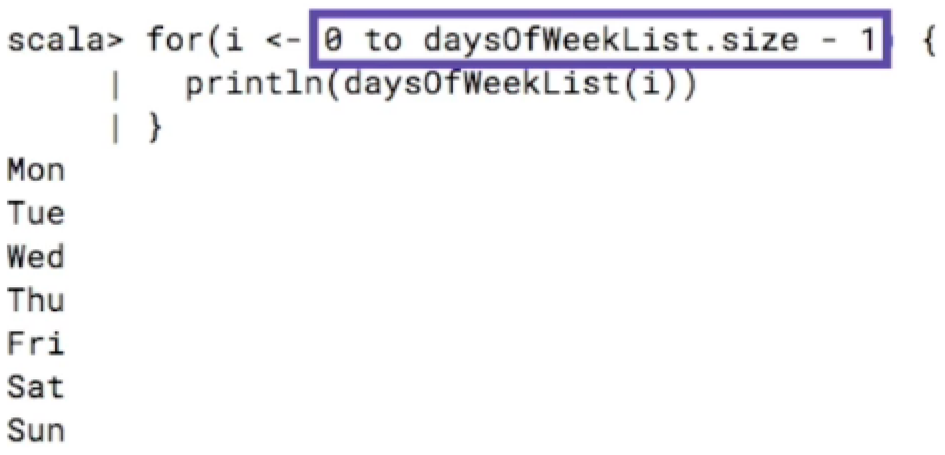


```
scala> val daysOfWeekList = List("Mon","Tue","Wed","Thu","Fri","Sat","Sun")
daysOfWeekList: List[String] = List(Mon, Tue, Wed, Thu, Fri, Sat, Sun)

scala>   for(day <- daysOfWeekList)
     |   {
     |     day match {
     |       case "Mon" => println("Manic Monday")
     |       case otherDay => println(otherDay)
     |     }
     |   }
Manic Monday
Tue
Wed
Thu
Fri
Sat
Sun
```

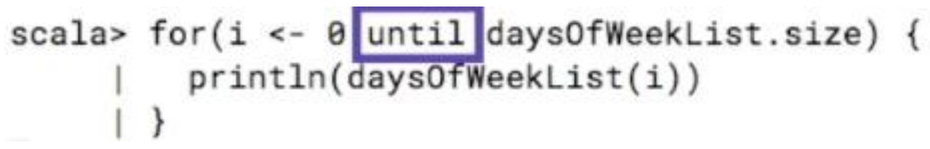
Expression



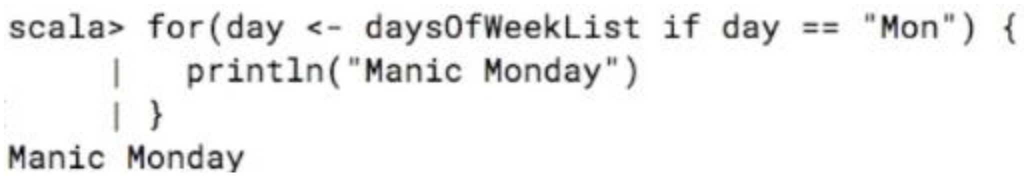
```
scala>  val x = for(day <- daysOfWeekList) yield
     |   {
     |     day match {
     |       case "Mon" => "Manic Monday"
     |       case otherDay => otherDay
     |     }
     |   }
x: List[String] = List(Manic Monday, Tue, Wed, Thu, Fri, Sat, Sun)

scala> for(day <- daysOfWeekList) {
     |   println(day)
     | }
```



```
scala> for(i <- 0 to daysOfWeekList.size - 1) {
     |   println(daysOfWeekList(i))
     | }
Mon
Tue
Wed
Thu
Fri
Sat
Sun
```

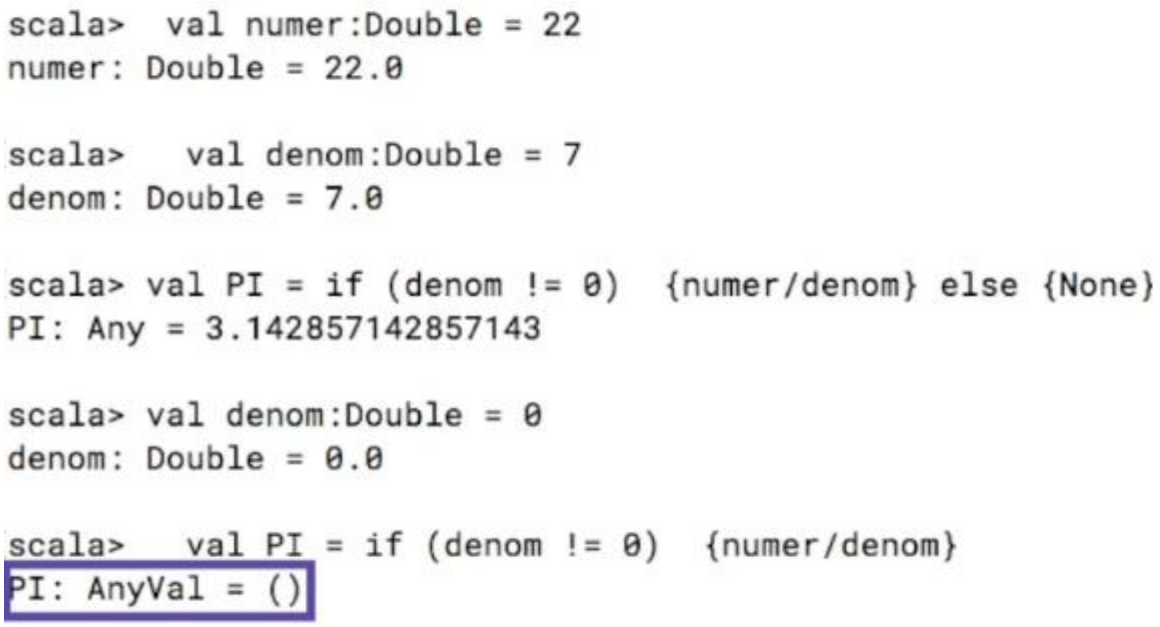


```
scala> for(i <- 0 until daysOfWeekList.size) {
     |   println(daysOfWeekList(i))
     | }
```



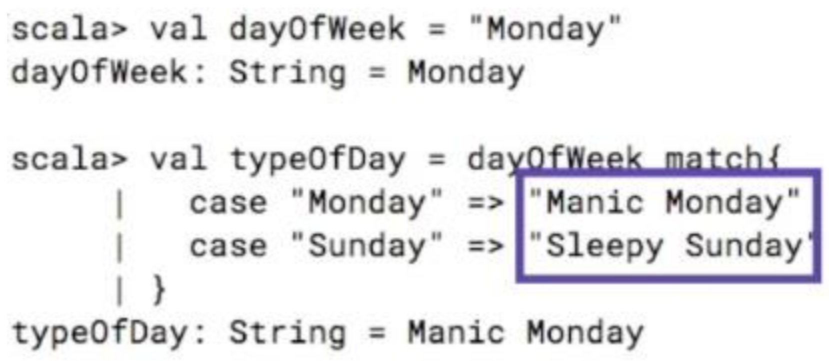
```
scala> for(day <- daysOfWeekList if day == "Mon") {
     |   println("Manic Monday")
     | }
Manic Monday
```

IF ELSE STATEMENT

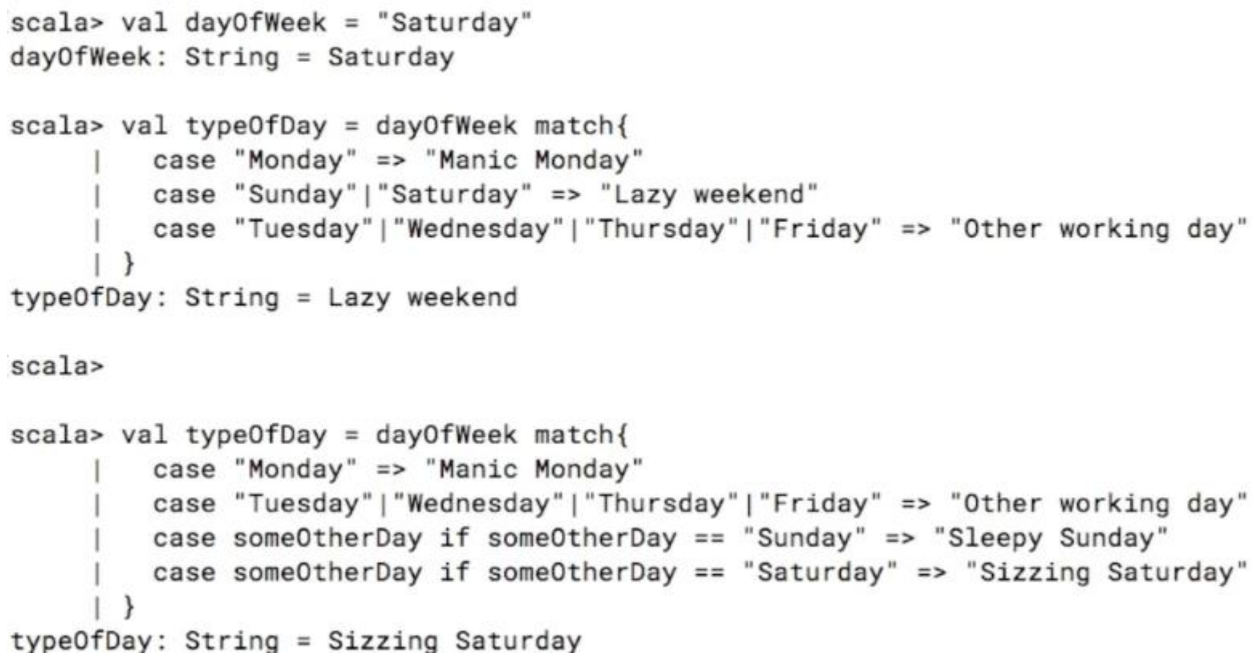


```
scala>  val numer:Double = 22
numer: Double = 22.0

scala>   val denom:Double = 7
denom: Double = 7.0

scala> val PI = if (denom != 0)  {numer/denom} else {None}
PI: Any = 3.142857142857143

scala> val denom:Double = 0
denom: Double = 0.0

scala>   val PI = if (denom != 0)  {numer/denom}
PI: AnyVal = ()
```

PATERN MATCHING

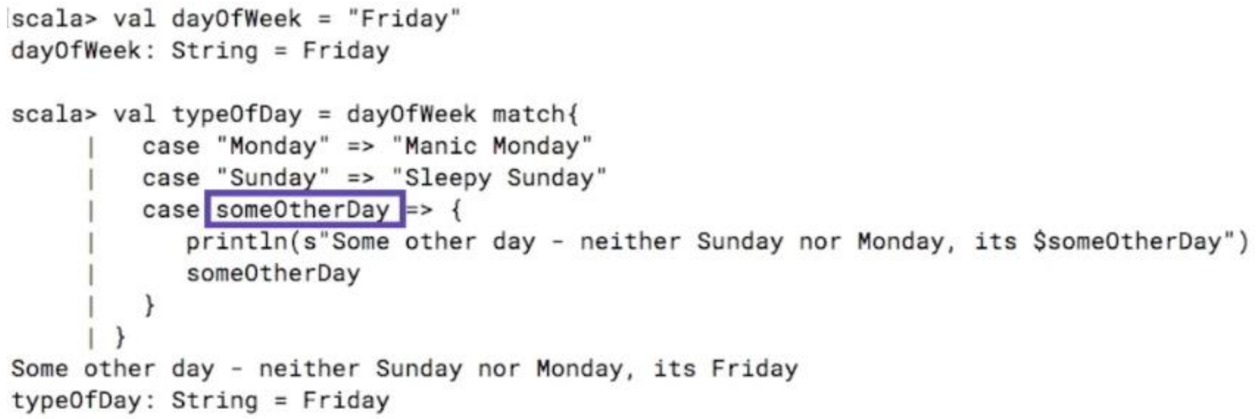


```
scala> val dayOfWeek = "Monday"
dayOfWeek: String = Monday

scala> val typeOfDay = dayOfWeek match{
     |   case "Monday" => "Manic Monday"
     |   case "Sunday" => "Sleepy Sunday"
     | }
typeOfDay: String = Manic Monday
```

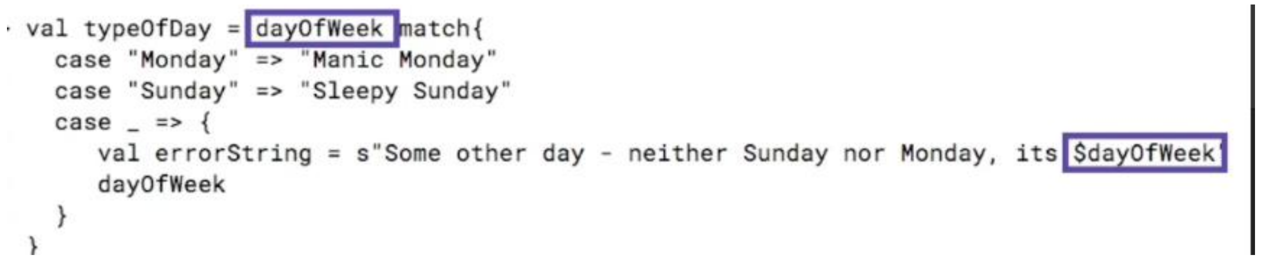


```
scala> val dayOfWeek = "Saturday"
dayOfWeek: String = Saturday

scala> val typeOfDay = dayOfWeek match{
     |   case "Monday" => "Manic Monday"
     |   case "Sunday"|"Saturday" => "Lazy weekend"
     |   case "Tuesday"|"Wednesday"|"Thursday"|"Friday" => "Other working day"
     | }
typeOfDay: String = Lazy weekend

scala>

scala> val typeOfDay = dayOfWeek match{
     |   case "Monday" => "Manic Monday"
     |   case "Tuesday"|"Wednesday"|"Thursday"|"Friday" => "Other working day"
     |   case someOtherDay if someOtherDay == "Sunday" => "Sleepy Sunday"
     |   case someOtherDay if someOtherDay == "Saturday" => "Sizzing Saturday"
     | }
typeOfDay: String = Sizzing Saturday
```

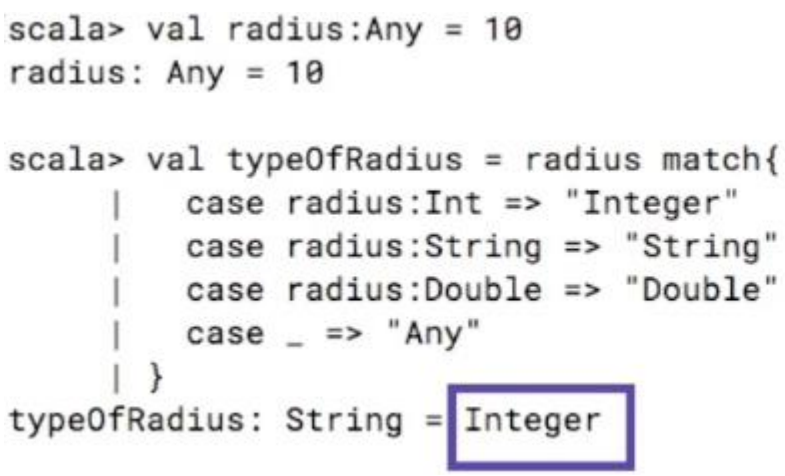
IF NONE OF THE CASE MATCHES WE CAN USE CATCH BLOCKS AS SHOWN BELOW



```
scala> val dayOfWeek = "Friday"
dayOfWeek: String = Friday

scala> val typeOfDay = dayOfWeek match{
     |   case "Monday" => "Manic Monday"
     |   case "Sunday" => "Sleepy Sunday"
     |   case someOtherDay => {
     |      println(s"Some other day - neither Sunday nor Monday, its $someOtherDay")
     |      someOtherDay
     |   }
     | }
Some other day - neither Sunday nor Monday, its Friday
typeOfDay: String = Friday
```



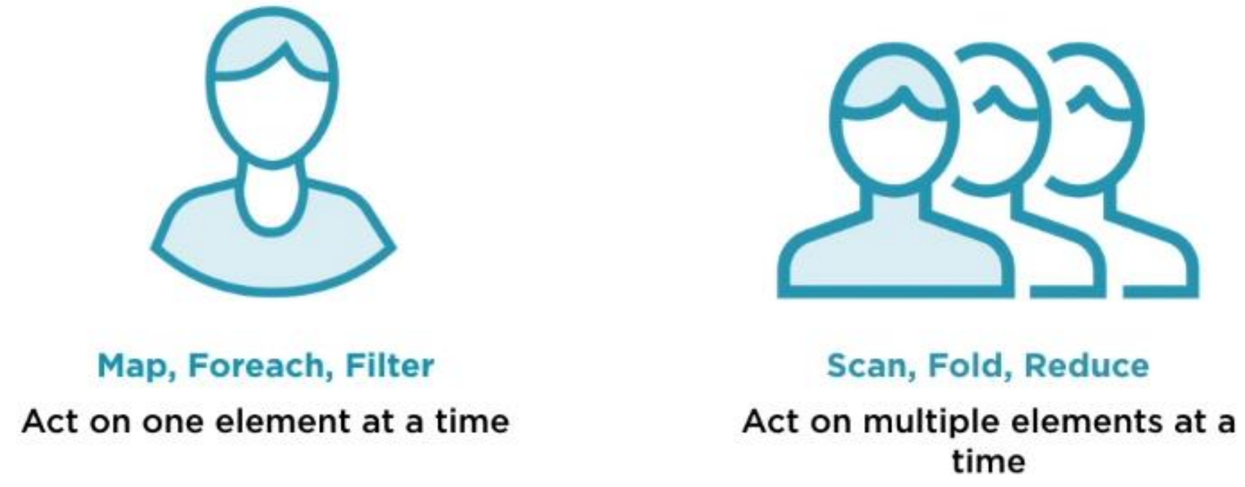
```
val typeOfDay = dayOfWeek match{
   case "Monday" => "Manic Monday"
   case "Sunday" => "Sleepy Sunday"
   case _ => {
      val errorString = s"Some other day - neither Sunday nor Monday, its $dayOfWeek
      dayOfWeek
   }
}
```



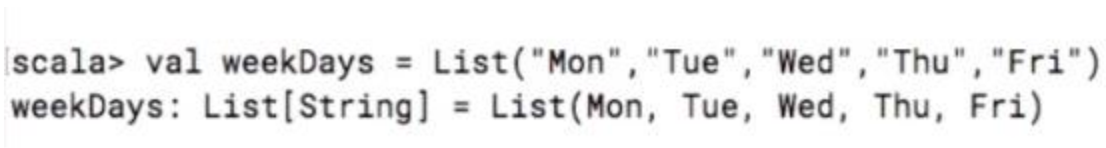
```
scala> val radius:Any = 10
radius: Any = 10

scala> val typeOfRadius = radius match{
     |   case radius:Int => "Integer"
     |   case radius:String => "String"
     |   case radius:Double => "Double"
     |   case _ => "Any"
     | }
typeOfRadius: String = Integer
```

COLLECTIONS

Higher Order METHODS

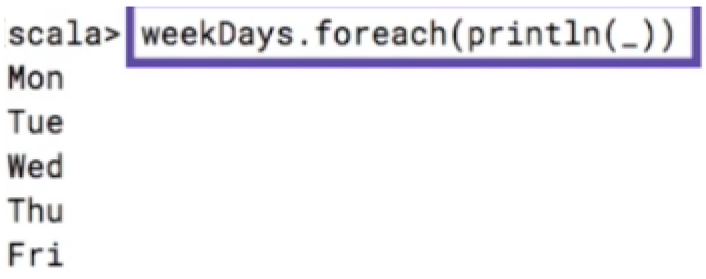

Map, Foreach, Filter
Act on one element at a time
Scan, Fold, Reduce
Act on multiple elements at a time

INPUT FOR THE ABOVE METHOD



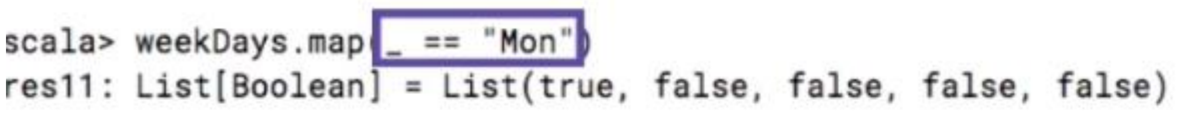
```
scala> val weekDays = List("Mon","Tue","Wed","Thu","Fri")
weekDays: List[String] = List(Mon, Tue, Wed, Thu, Fri)
```

FOREACH SYNTAX

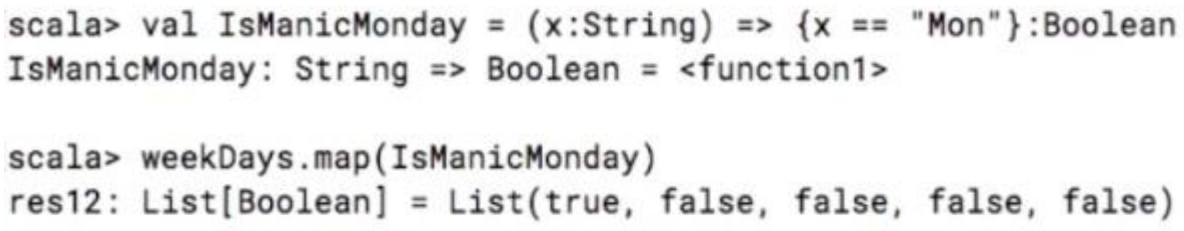


```
scala> weekDays.foreach(println(_))
Mon
Tue
Wed
Thu
Fri
```

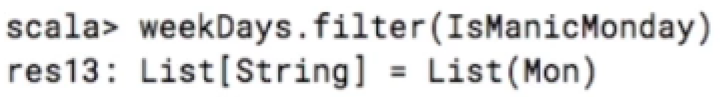
MAP METHOD



```
scala> weekDays.map(_ == "Mon")
res11: List[Boolean] = List(true, false, false, false, false)
```

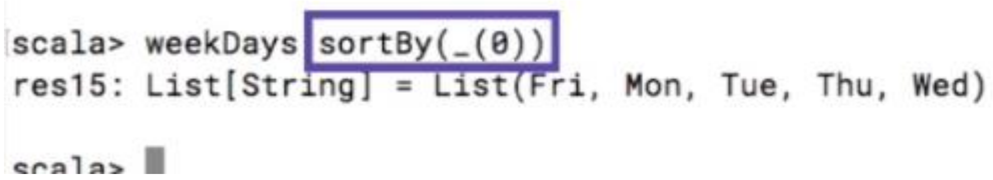


```
scala> val IsManicMonday = (x:String) => {x == "Mon"}:Boolean
IsManicMonday: String => Boolean = <function1>

scala> weekDays.map(IsManicMonday)
res12: List[Boolean] = List(true, false, false, false, false)
```



```
scala> weekDays.filter(IsManicMonday)
res13: List[String] = List(Mon)
```

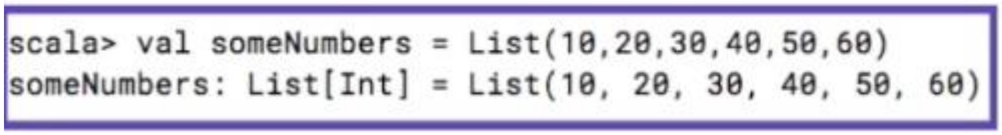
The following will sort by first character



```
scala> weekDays sortBy(_(0))
res15: List[String] = List(Fri, Mon, Tue, Thu, Wed)

scala> 
```

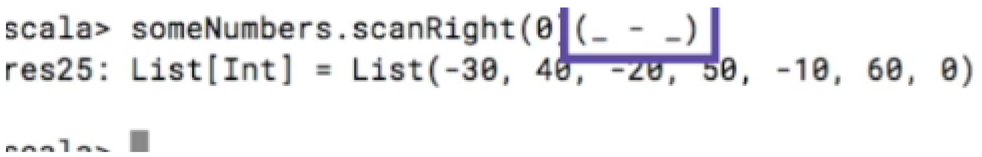
SCAN

INPUT

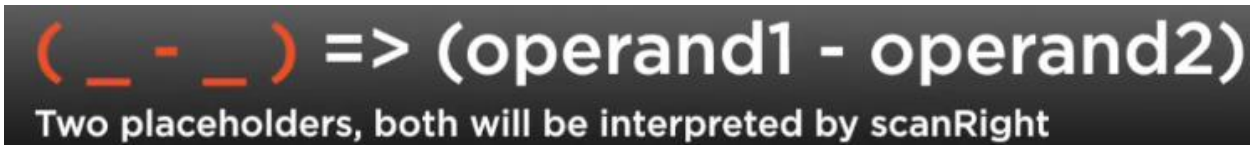


```
scala> val someNumbers = List(10,20,30,40,50,60)
someNumbers: List[Int] = List(10, 20, 30, 40, 50, 60)
```

SCAN RIGHT SYNTAX

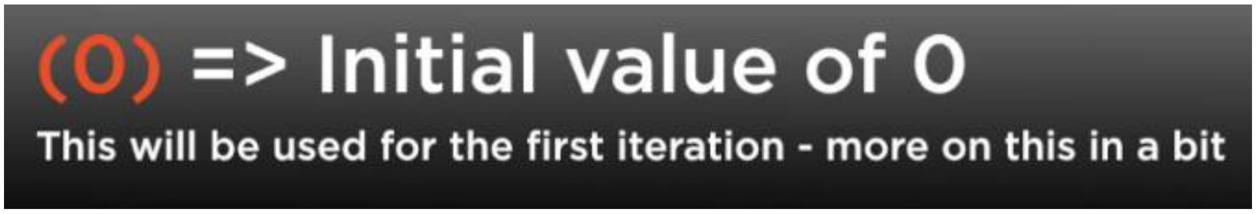


```
scala> someNumbers.scanRight(0)(_ - _)
res25: List[Int] = List(-30, 40, -20, 50, -10, 60, 0)

scala> 
```



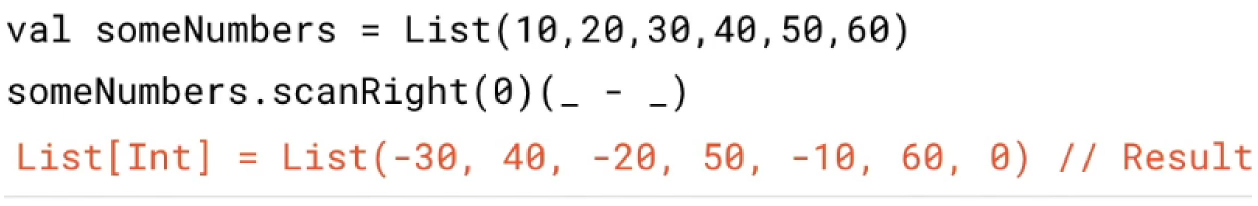
(_ - _) => (operand1 - operand2)

Two placeholders, both will be interpreted by scanRight

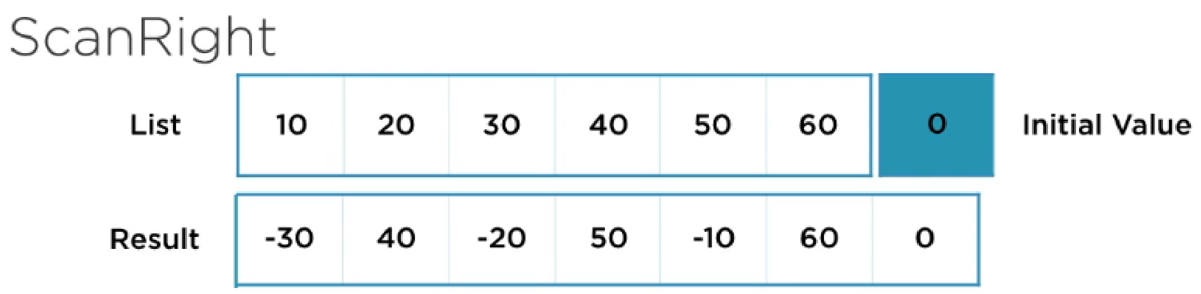


(0) => Initial value of 0

This will be used for the first iteration - more on this in a bit

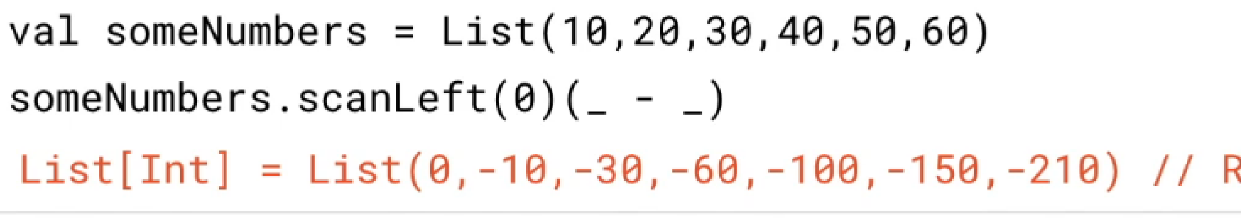


```
val someNumbers = List(10,20,30,40,50,60)
someNumbers.scanRight(0)(_ - _)
 List[Int] = List(-30, 40, -20, 50, -10, 60, 0) // Result
```

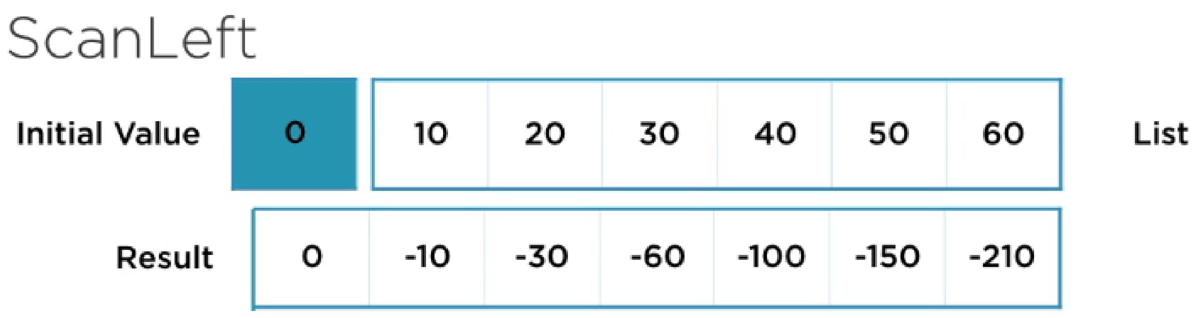


ScanRight

List	10	20	30	40	50	60	0	Initial Value
Result	-30	40	-20	50	-10	60	0	



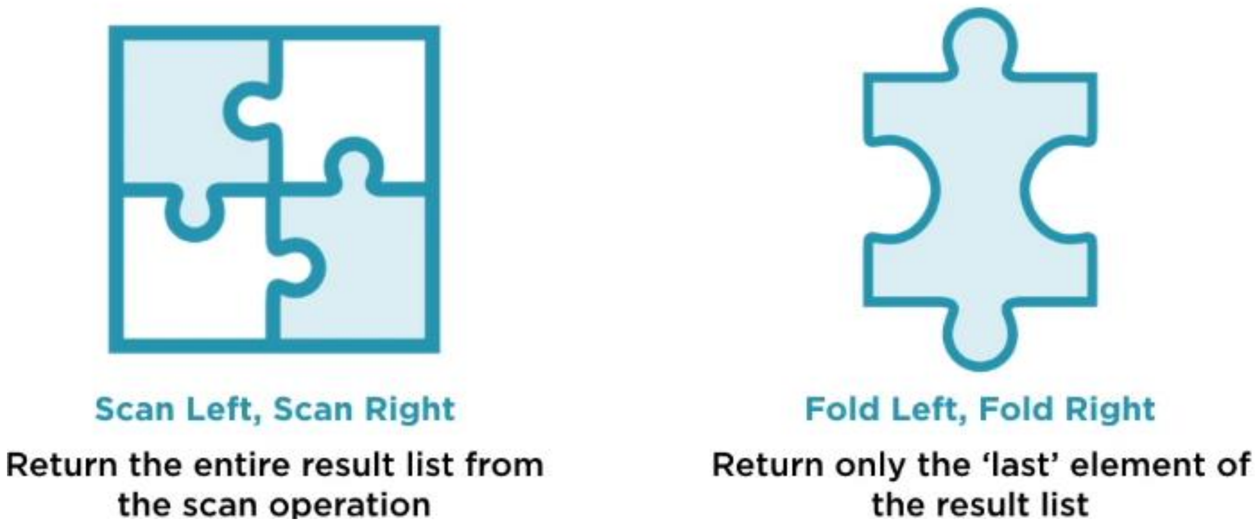
```
val someNumbers = List(10,20,30,40,50,60)
someNumbers.scanLeft(0)(_ - _)
 List[Int] = List(0,-10,-30,-60,-100,-150,-210) // R
```



ScanLeft

Initial Value	0	10	20	30	40	50	60	List
Result	0	-10	-30	-60	-100	-150	-210	

FOLD LEFT AND FOLD RIGHT



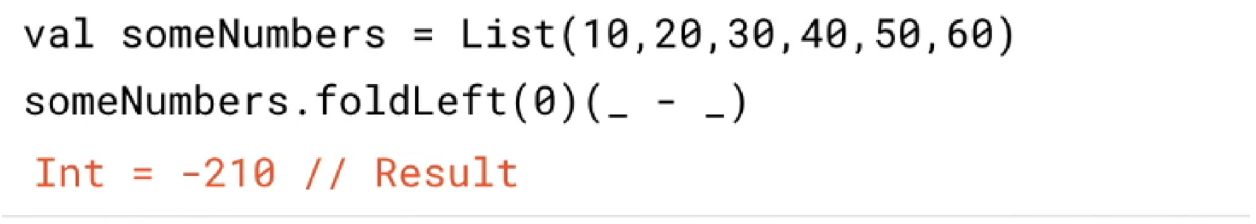
Scan Left, Scan Right

Return the entire result list from the scan operation

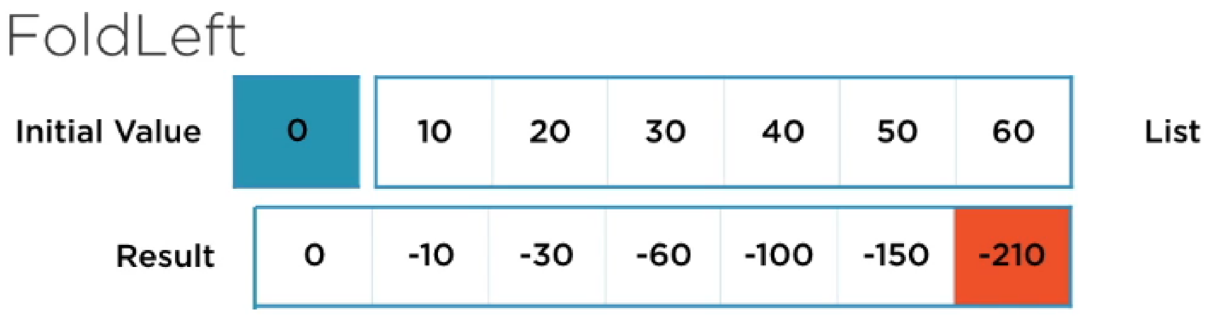
Fold Left, Fold Right

Return only the 'last' element of the result list

Fold left



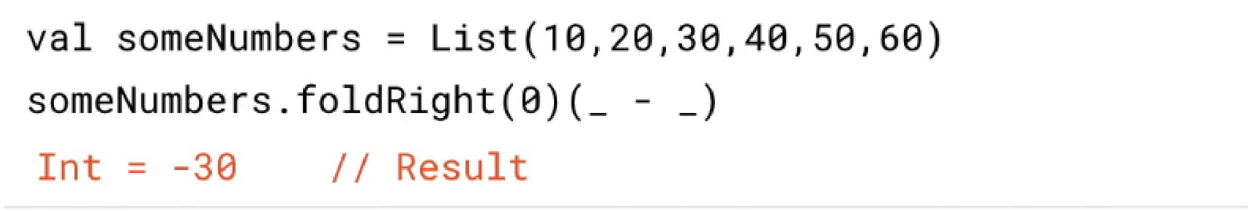
```
val someNumbers = List(10,20,30,40,50,60)
someNumbers.foldLeft(0)(_ - _)
 Int = -210 // Result
```



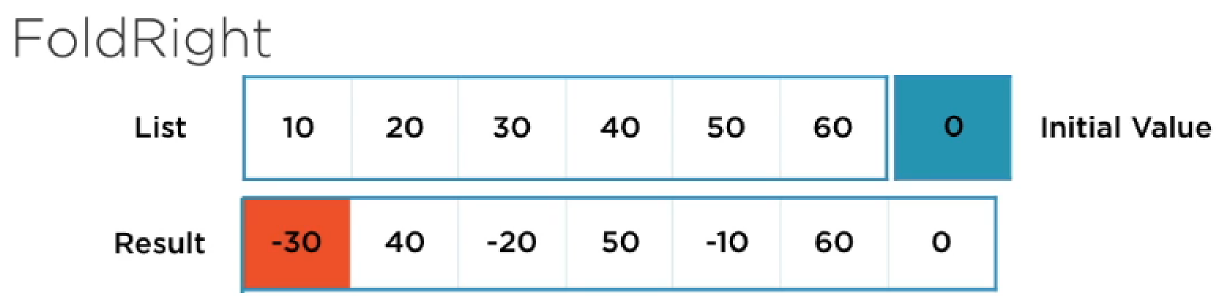
FoldLeft

Initial Value	0	10	20	30	40	50	60	List
Result	0	-10	-30	-60	-100	-150	-210	

Fold right



```
val someNumbers = List(10,20,30,40,50,60)
someNumbers.foldRight(0)(_ - _)
 Int = -30     // Result
```

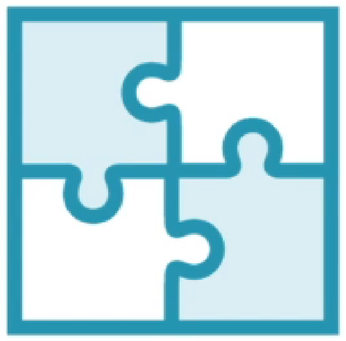


FoldRight

List	10	20	30	40	50	60	0	Initial Value
Result	-30	40	-20	50	-10	60	0	

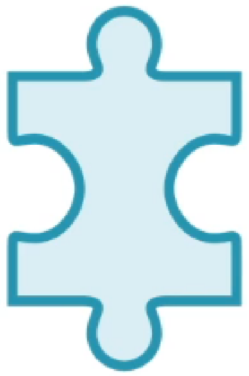
REDUCE

Scan and Reduce



Scan Left, Scan Right

Take in an initial value; use this initial value as second operand in first step



Reduce Left, Reduce Right

No initial value - first two list elements as operands in first step

```
val someNumbers = List(10,20,30,40,50,60)
someNumbers.reduceRight(_ - _)
 Int = -30     // Result
```

ReduceRight

List	10	20	30	40	50	60
Result	-30	40	-20	50	-10	