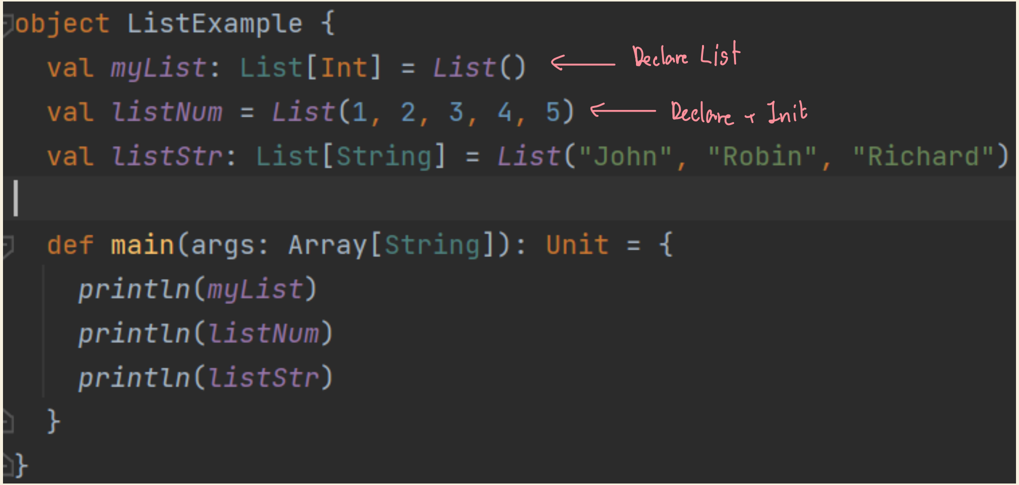
DATA STRUCTURE FOR RECURSION

LIST

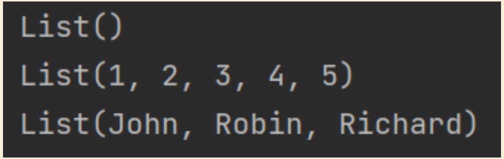
- Immutable
- Linked list



```
object ListExample {
  val myList: List[Int] = List()
  val listNum = List(1, 2, 3, 4, 5)
  val listStr: List[String] = List("John", "Robin", "Richard")

  def main(args: Array[String]): Unit = {
    println(myList)
    println(listNum)
    println(listStr)
  }
}
```

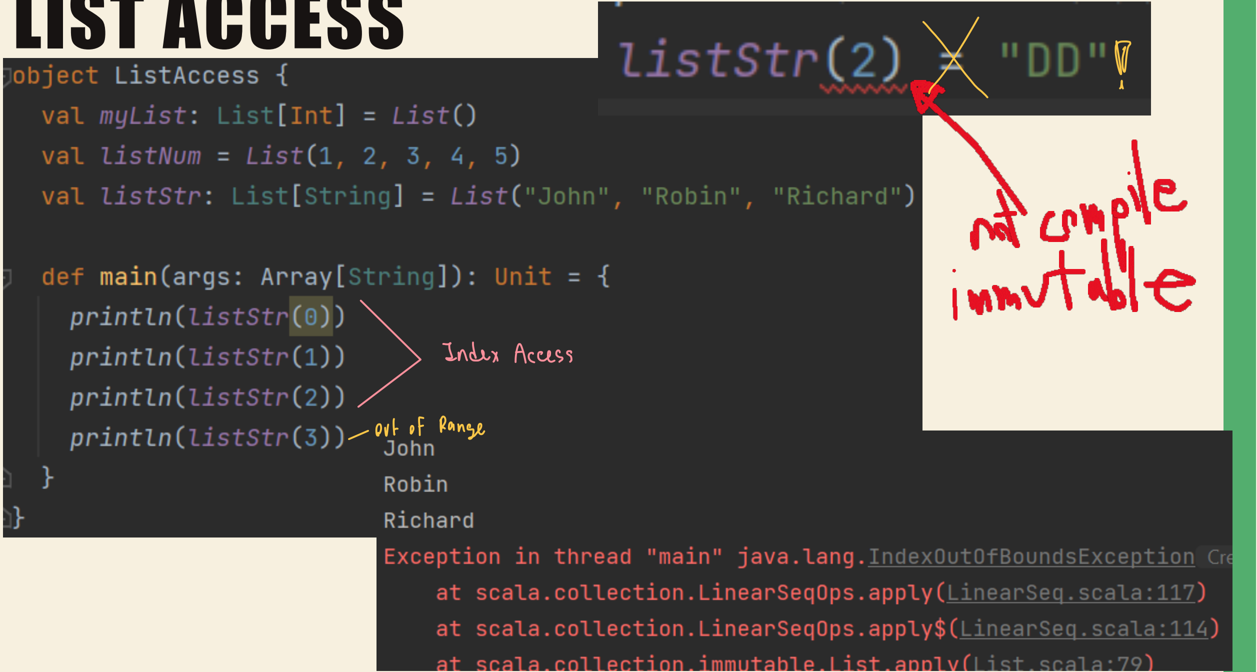
Declare List

Declare + Init



```
List()
List(1, 2, 3, 4, 5)
List(John, Robin, Richard)
```

LIST ACCESS

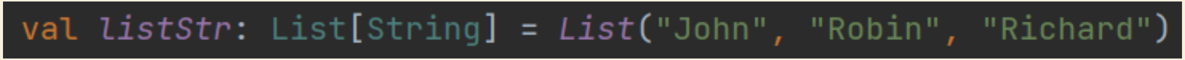


```
object ListAccess {
  val myList: List[Int] = List()
  val listNum = List(1, 2, 3, 4, 5)
  val listStr: List[String] = List("John", "Robin", "Richard")

  def main(args: Array[String]): Unit = {
    println(listStr(0))
    println(listStr(1))
    println(listStr(2))
    println(listStr(3))
  }
}
```

```
listStr(2) = "DD"
```

```
John
Robin
Richard
Exception in thread "main" java.lang.IndexOutOfBoundsException Cre
	at scala.collection.LinearSeqOps.apply(LinearSeq.scala:117)
	at scala.collection.LinearSeqOps.apply$(LinearSeq.scala:114)
	at scala.collection.immutable.List.apply(List.scala:79)
```

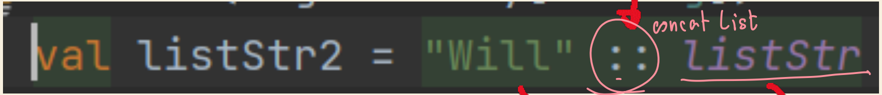
HOW TO DEFINE A LIST?



```
val listStr: List[String] = List("John", "Robin", "Richard")
```

- Use a cons

cons



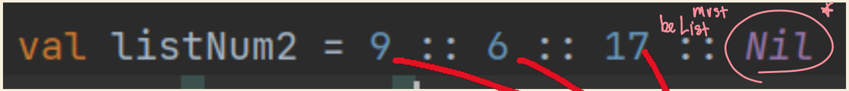
```
val listStr2 = "Will" :: listStr
```

concat List

Must be List!

First data

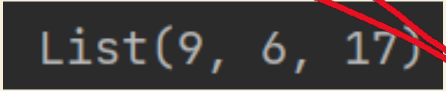
List of the rest of data



```
val listNum2 = 9 :: 6 :: 17 :: Nil
```

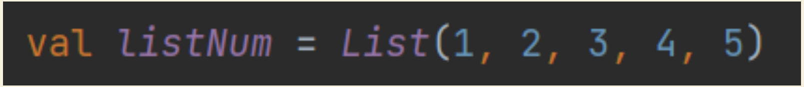
must be List

*

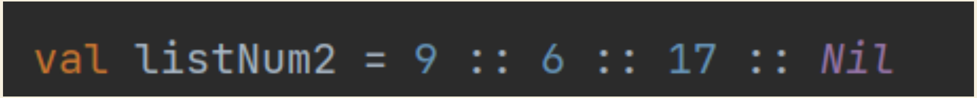


```
List(9, 6, 17)
```

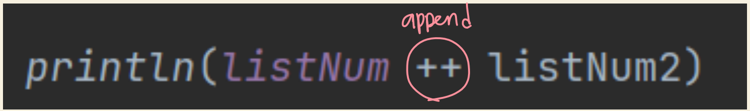
Anything in front or between it must be a data.



```scala
val listNum = List(1, 2, 3, 4, 5)
```

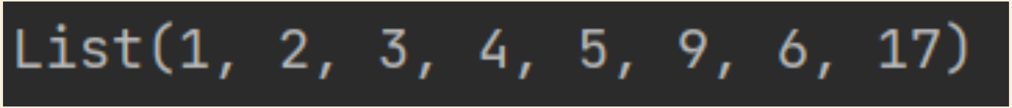


```scala
val listNum2 = 9 :: 6 :: 17 :: Nil
```



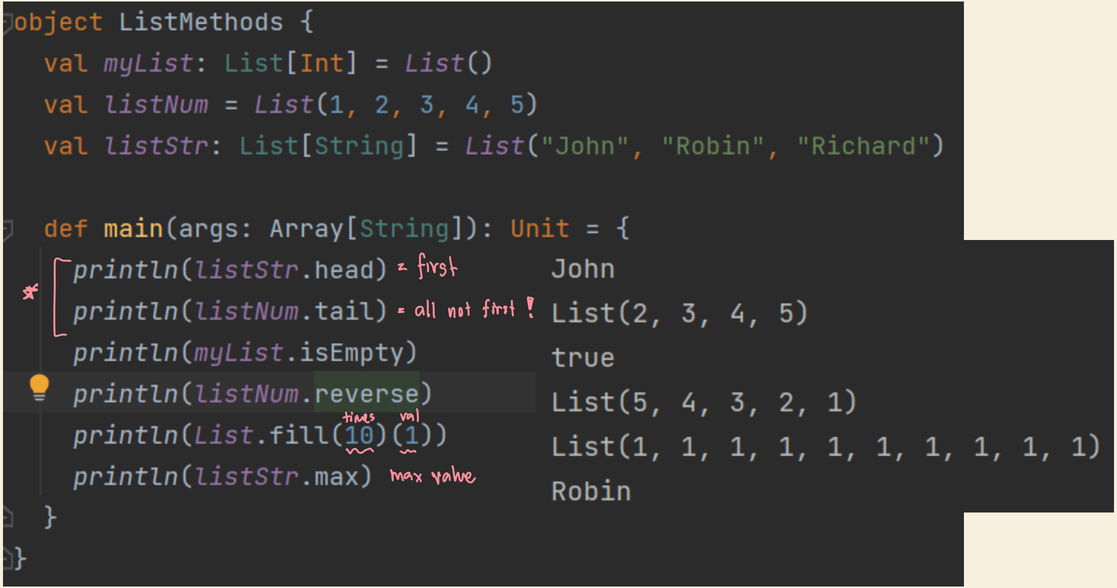
append

```scala
println(listNum ++ listNum2)
```



```
List(1, 2, 3, 4, 5, 9, 6, 17)
```

LIST METHODS

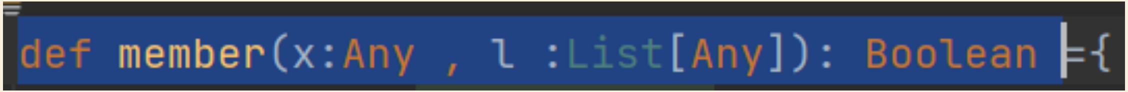


```
object ListMethods {
  val myList: List[Int] = List()
  val listNum = List(1, 2, 3, 4, 5)
  val listStr: List[String] = List("John", "Robin", "Richard")

  def main(args: Array[String]): Unit = {
*   println(listStr.head) = first
    println(listNum.tail) = all not first !
    println(myList.isEmpty)
    println(listNum.reverse)
                  times val
    println(List.fill(10)(1))
    println(listStr.max) max value
  }
}
```

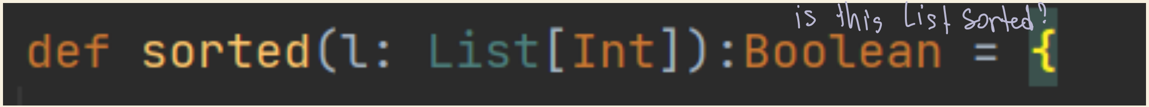
```
John
List(2, 3, 4, 5)
true
List(5, 4, 3, 2, 1)
List(1, 1, 1, 1, 1, 1, 1, 1, 1, 1)
Robin
```

EXERCISE (ONLY ISEMPTY, LENGTH, HEAD, TAIL, ::, ++ AVAILABLE)

is x in l ?

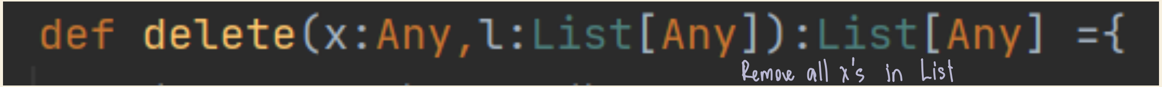


```
def member(x:Any , l :List[Any]): Boolean ={
```



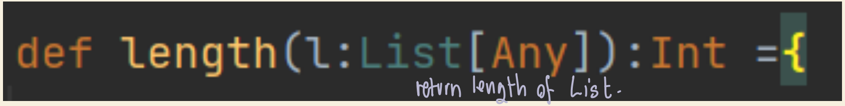
```
def sorted(l: List[Int]):Boolean = {
```

is this List sorted?



```
def delete(x:Any,l:List[Any]):List[Any] ={
```

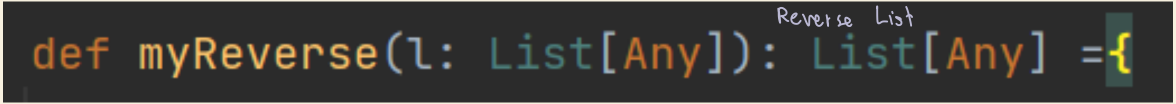
Remove all x's in List



```
def length(l:List[Any]):Int ={
```

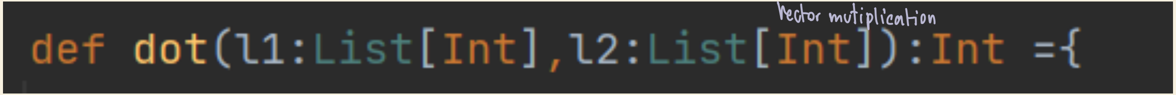
return length of List.

EXERCISE - CONT



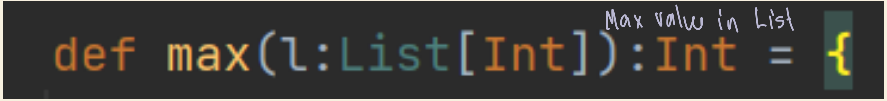
Reverse List

```
def myReverse(l: List[Any]): List[Any] ={
```



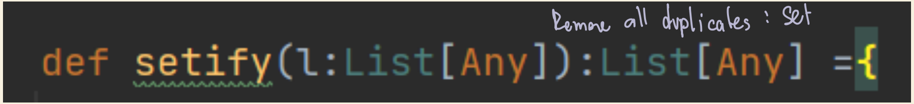
Vector mutiplication

```
def dot(l1:List[Int],l2:List[Int]):Int ={
```



Max value in List

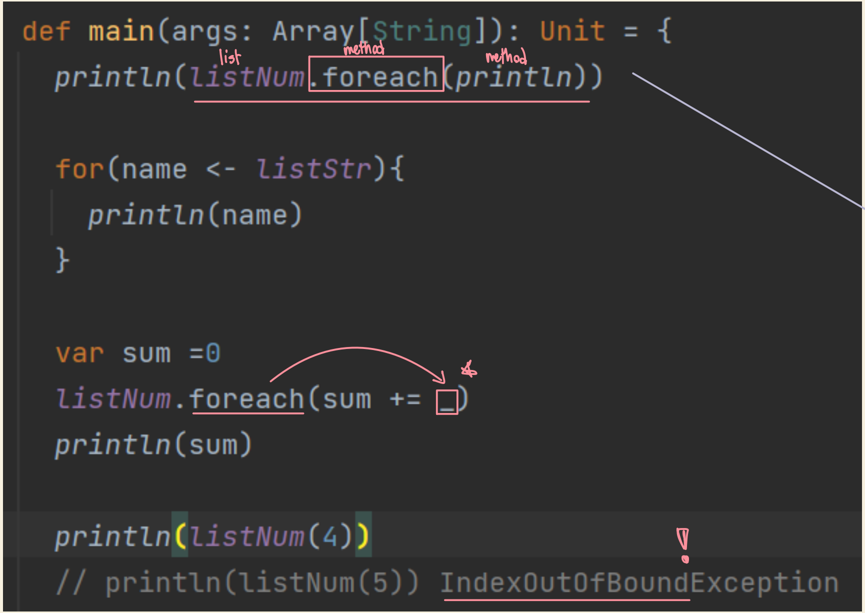
```
def max(l:List[Int]):Int = {
```



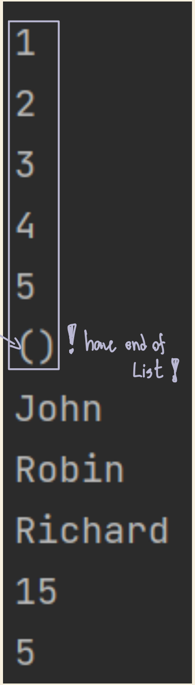
Remove all duplicates : Set

```
def setify(l:List[Any]):List[Any] ={
```

LIST ITERATION



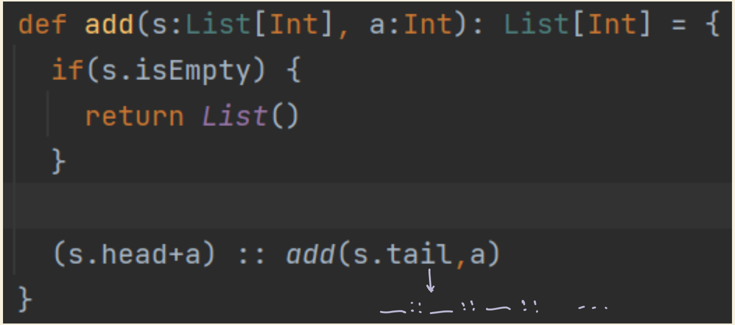
```
def main(args: Array[String]): Unit = {
  println(listNum.foreach(println))

  for(name <- listStr){
    println(name)
  }

  var sum =0
  listNum.foreach(sum += _)
  println(sum)

  println(listNum(4))
  // println(listNum(5)) IndexOutOfBoundException
```



```
1
2
3
4
5
()
John
Robin
Richard
15
5
```

ITERATE TO MODIFY A LIST?

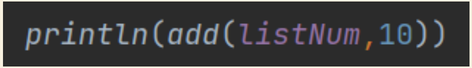
- Cannot be done because list is immutable.
- We have to produce a new list.



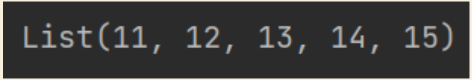
```
def add(s:List[Int], a:Int): List[Int] = {
  if(s.isEmpty) {
    return List()
  }

  (s.head+a) :: add(s.tail,a)
}
```

↓

_ :: _ :: _ :: ...



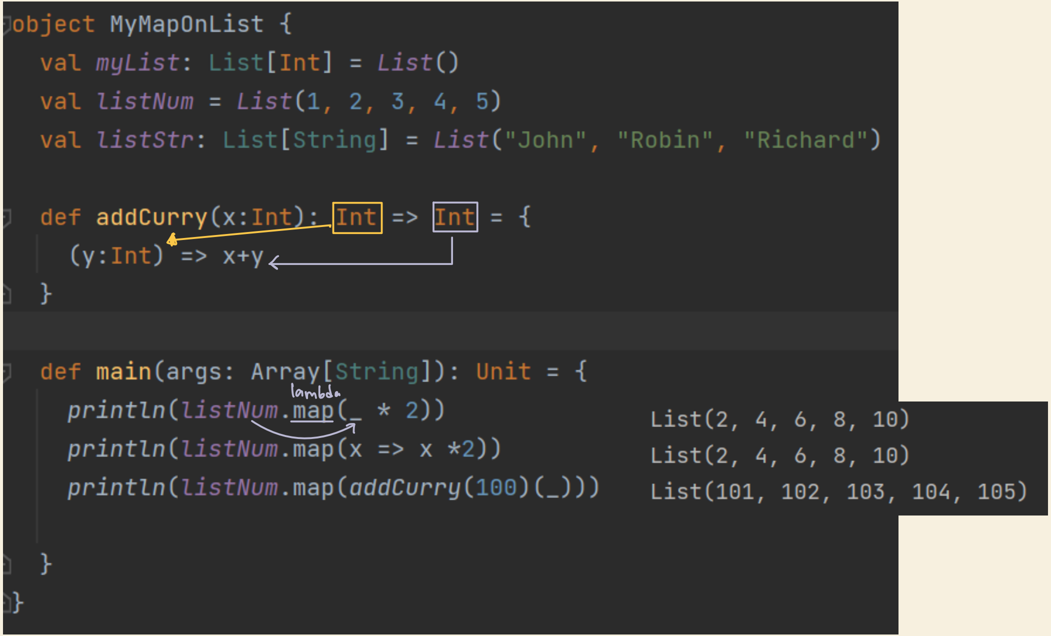
```
println(add(listNum,10))
```



```
List(11, 12, 13, 14, 15)
```

HIGHER ORDER METHODS

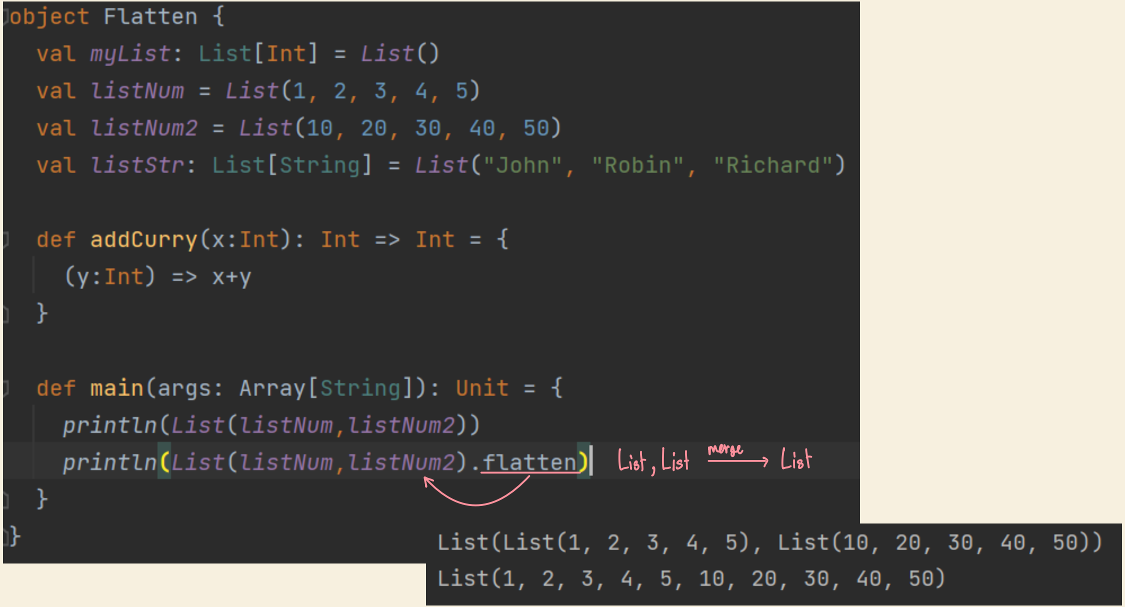
MAP



```
object MyMapOnList {
  val myList: List[Int] = List()
  val listNum = List(1, 2, 3, 4, 5)
  val listStr: List[String] = List("John", "Robin", "Richard")

  def addCurry(x:Int): Int => Int = {
    (y:Int) => x+y
  }


  def main(args: Array[String]): Unit = {
    println(listNum.map(_ * 2))
    println(listNum.map(x => x *2))
    println(listNum.map(addCurry(100)(_)))

  }
}
```

```
List(2, 4, 6, 8, 10)
List(2, 4, 6, 8, 10)
List(101, 102, 103, 104, 105)
```

FLATTEN!

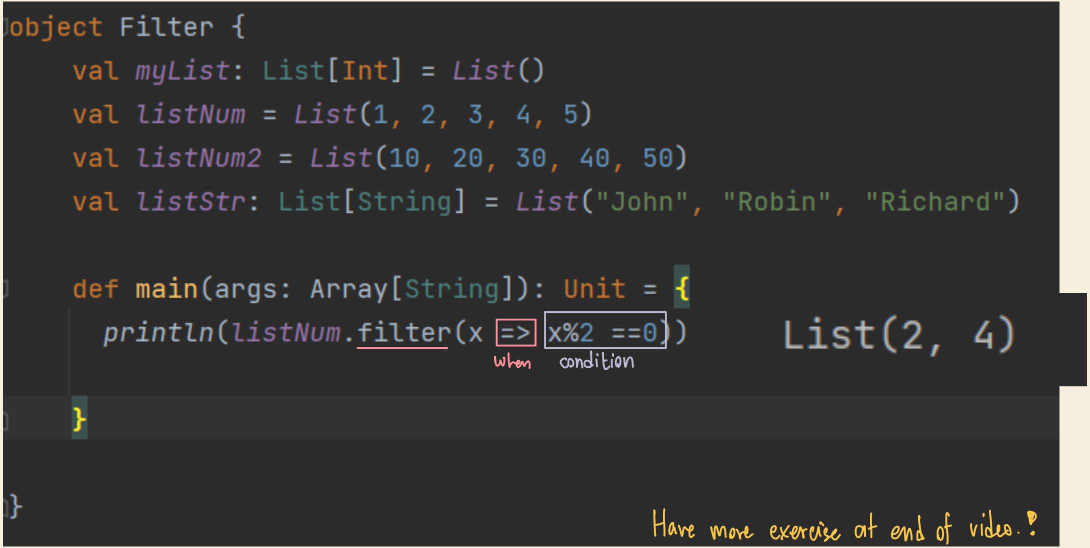


```
object Flatten {
  val myList: List[Int] = List()
  val listNum = List(1, 2, 3, 4, 5)
  val listNum2 = List(10, 20, 30, 40, 50)
  val listStr: List[String] = List("John", "Robin", "Richard")

  def addCurry(x:Int): Int => Int = {
    (y:Int) => x+y
  }

  def main(args: Array[String]): Unit = {
    println(List(listNum,listNum2))
    println(List(listNum,listNum2).flatten)
  }
}
```

List, List $\xrightarrow{merge}$ List

```
List(List(1, 2, 3, 4, 5), List(10, 20, 30, 40, 50))
List(1, 2, 3, 4, 5, 10, 20, 30, 40, 50)
```

FILTER



```scala
object Filter {
    val myList: List[Int] = List()
    val listNum = List(1, 2, 3, 4, 5)
    val listNum2 = List(10, 20, 30, 40, 50)
    val listStr: List[String] = List("John", "Robin", "Richard")

    def main(args: Array[String]): Unit = {
      println(listNum.filter(x => x%2 ==0))

    }

}
```

```
List(2, 4)
```

Have more exercise at end of video.!