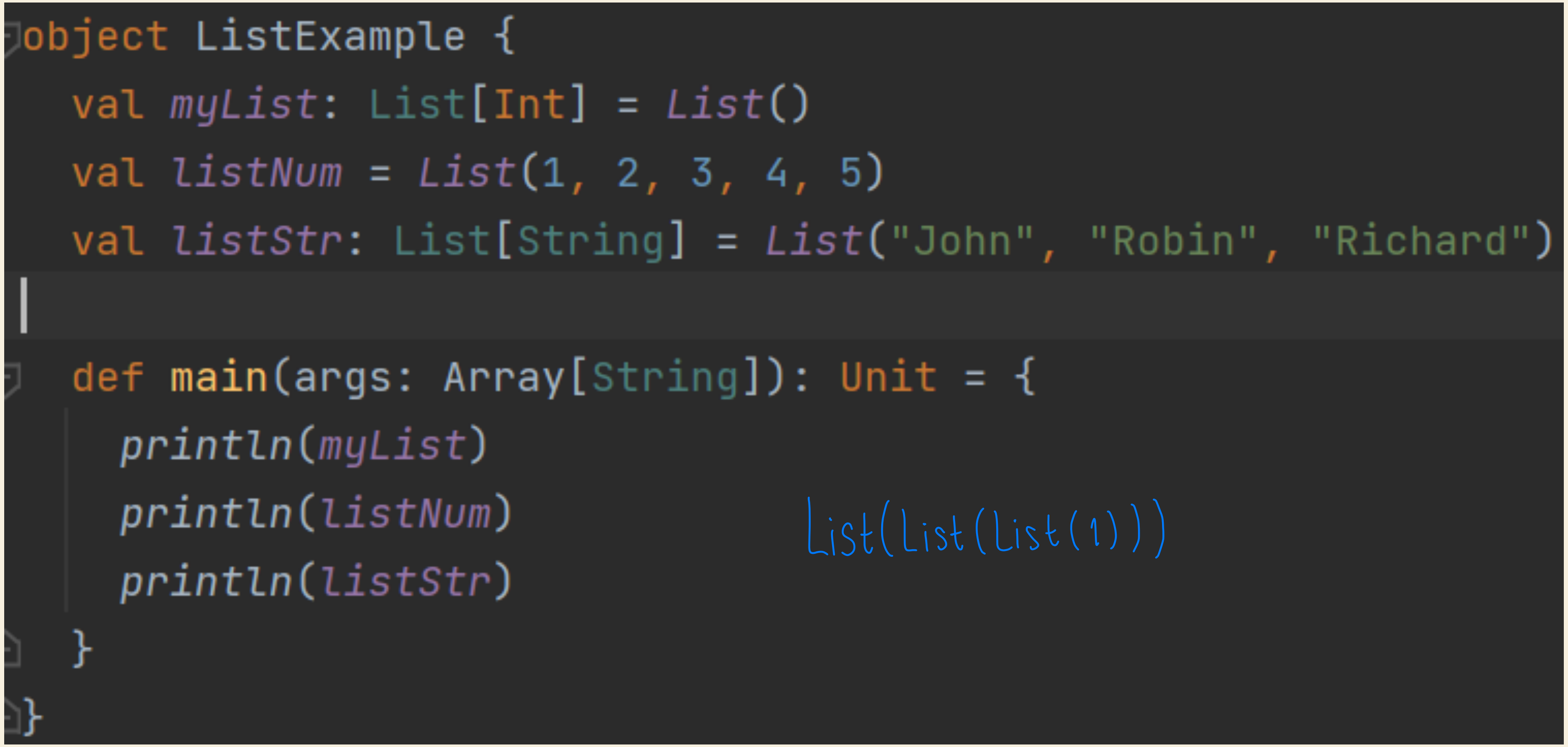
DATA STRUCTURE FOR RECURSION

LIST

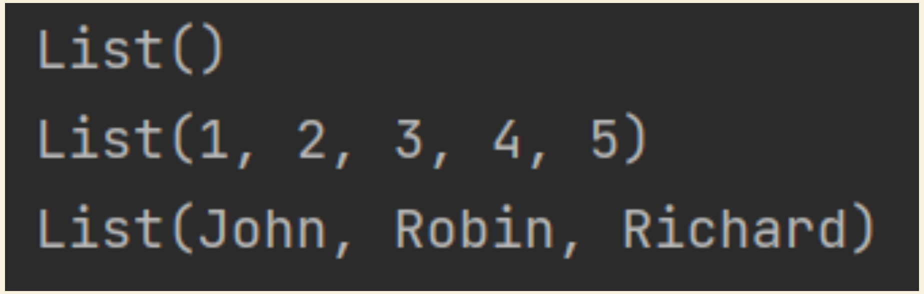
ของข้างในเปลี่ยนไม่ได้

- Immutable
- Linked list



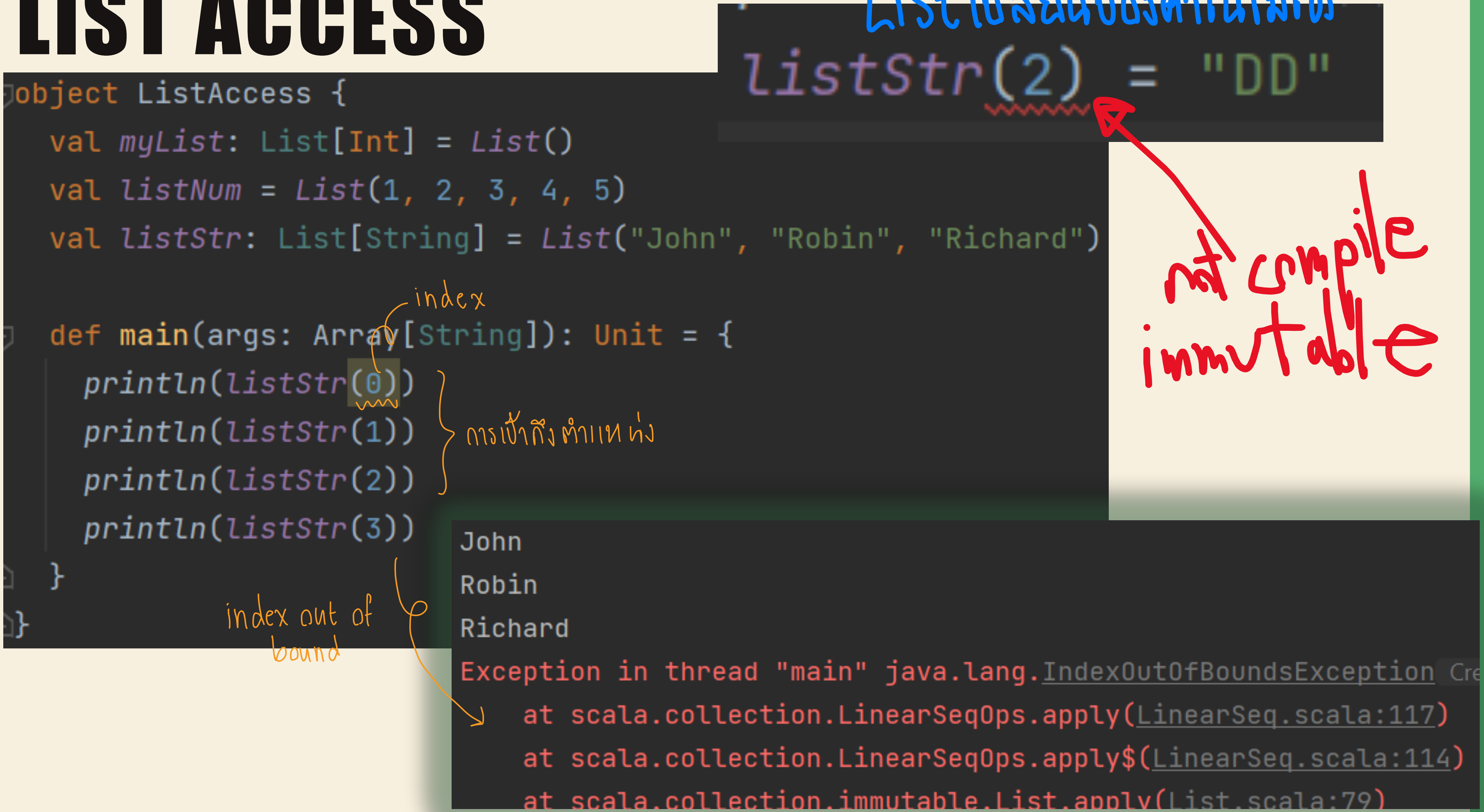
```
object ListExample {
  val myList: List[Int] = List()
  val listNum = List(1, 2, 3, 4, 5)
  val listStr: List[String] = List("John", "Robin", "Richard")

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

List(List(List(1)))



```
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))
  }
}
```

index

การเข้าถึง ตำแหน่ง

index out of bound

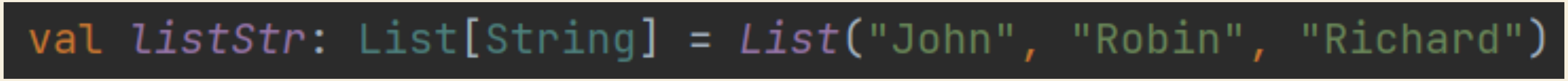
```
listStr(2) = "DD"
```

List เปลี่ยนของค่าในไม่ได้

กด compile immutable

```
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)
```

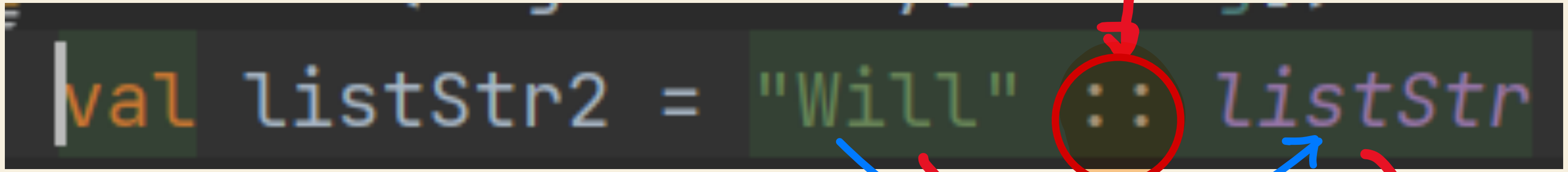
HOW TO DEFINE A LIST?



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

- Use a cons

cons

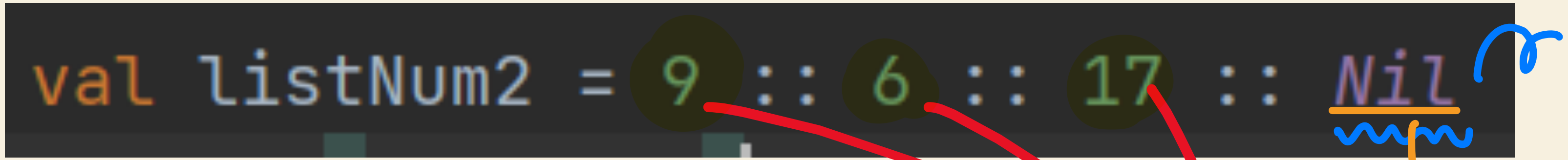


```
val listStr2 = "Will" :: listStr
```

append เข้าไป

First data

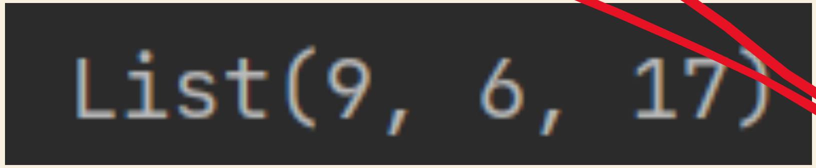
List of the rest of data



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

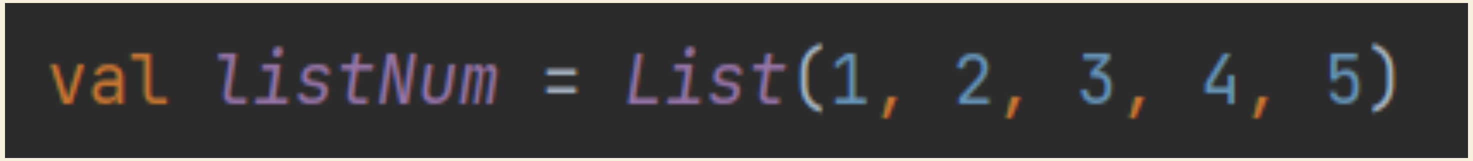
ตัวท้ายเป็น list

empty list

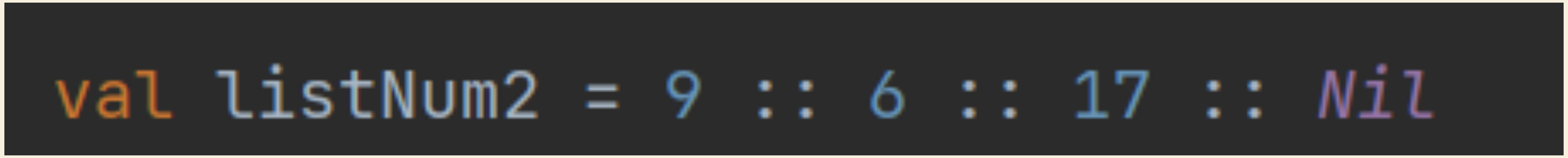


```
List(9, 6, 17)
```

Anything in front or between it must be a data.

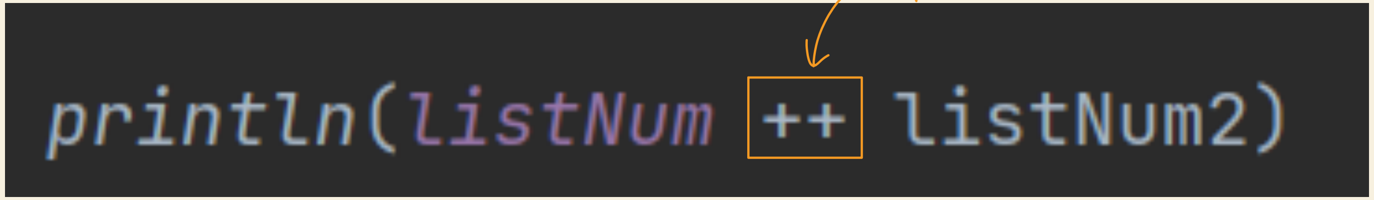


```
val listNum = List(1, 2, 3, 4, 5)
```

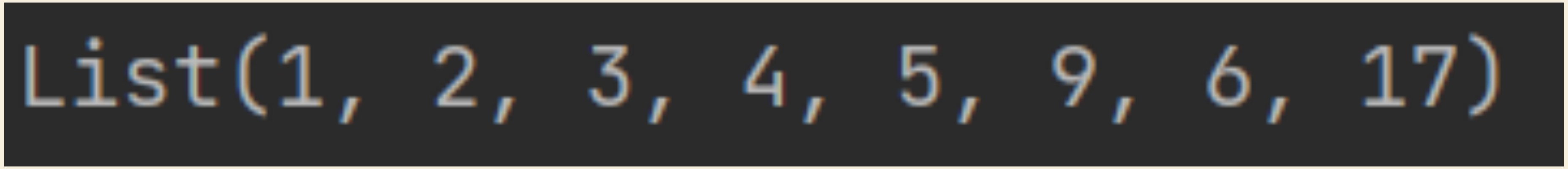


```
val listNum2 = 9 :: 6 :: 17 :: Nil
```

append

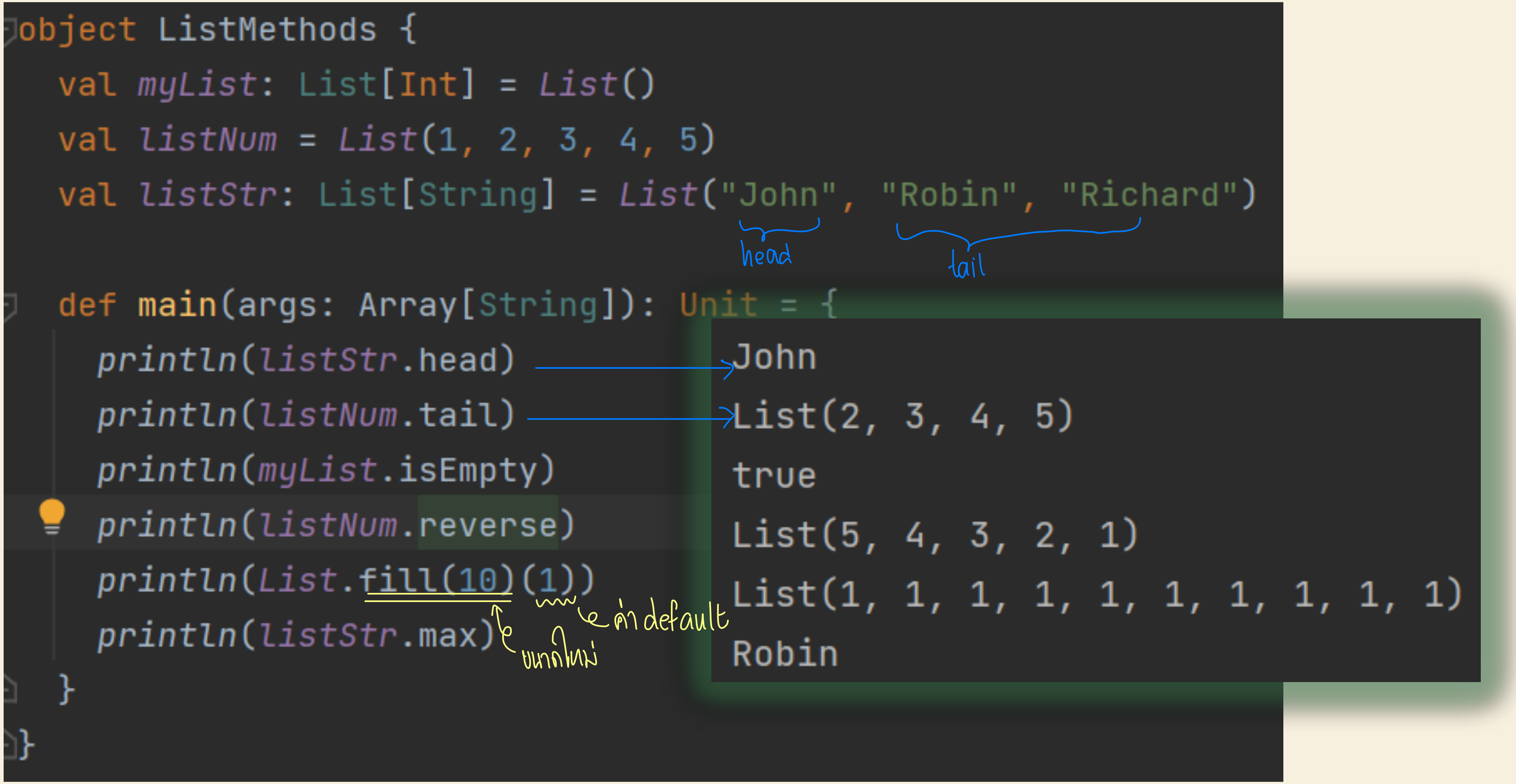


```
println(listNum ++ listNum2)
```



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

LIST METHODS

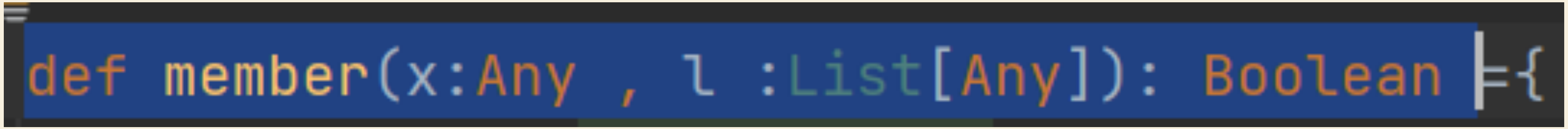


```
object ListMethods {
  val myList: List[Int] = List()
  val listNum = List(1, 2, 3, 4, 5)
  val listStr: List[String] = List("John", "Robin", "Richard")

  def main(args: Array[String]): Unit = {
    println(listStr.head)
    println(listNum.tail)
    println(myList.isEmpty)
    println(listNum.reverse)
    println(List.fill(10)(1))
    println(listStr.max)
  }
}
```

```
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
```

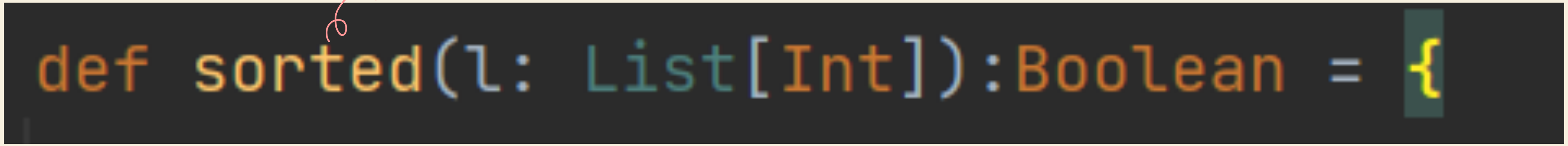
EXERCISE (ONLY ISEMPTY, LENGTH, HEAD, TAIL, ::, ++ AVAILABLE)

Recursive

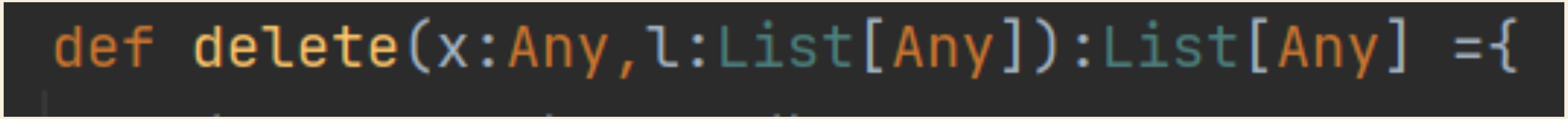


```
def member(x:Any , l :List[Any]): Boolean ={
```

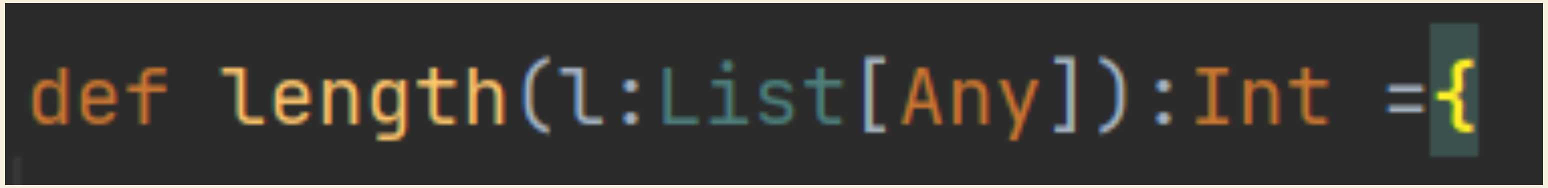
น้อยไปมาก



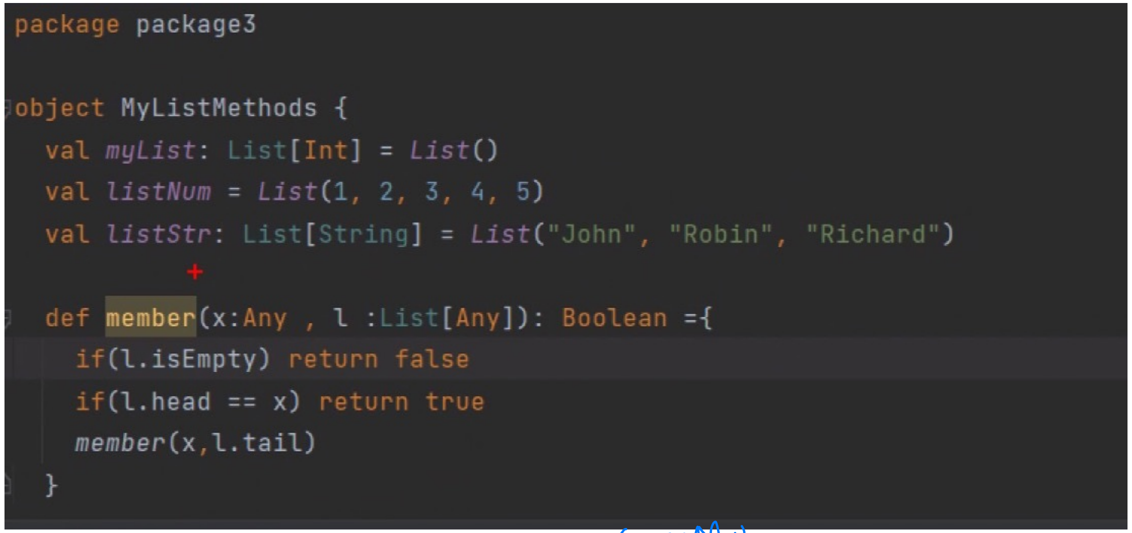
```
def sorted(l: List[Int]):Boolean = {
```



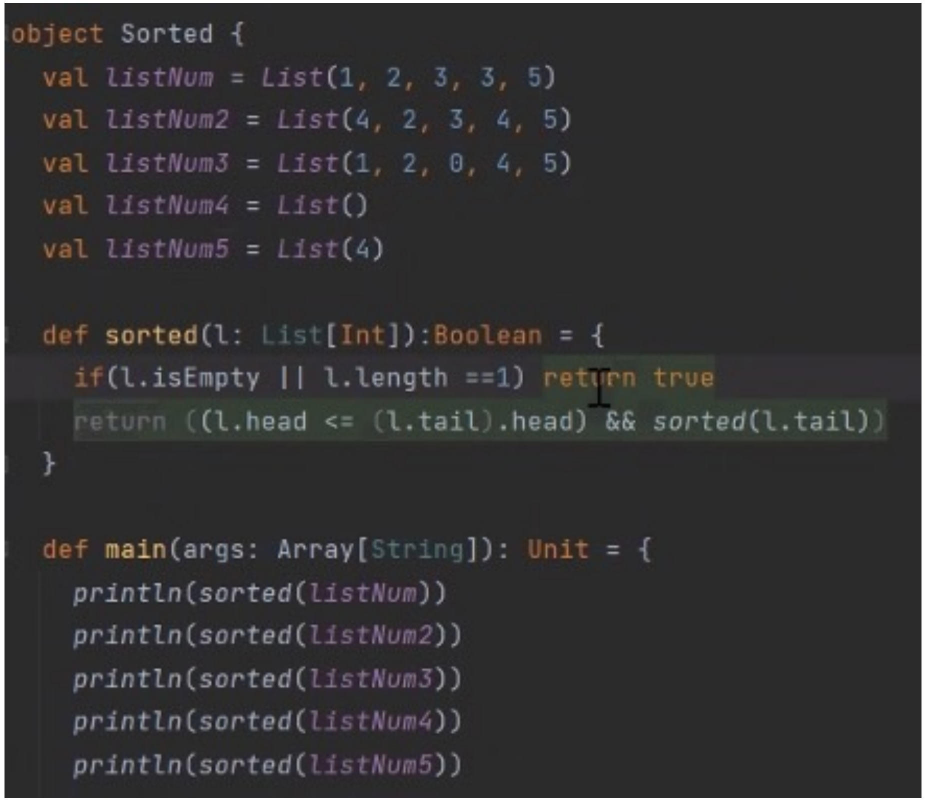
```
 def delete(x:Any,l:List[Any]):List[Any] ={
```



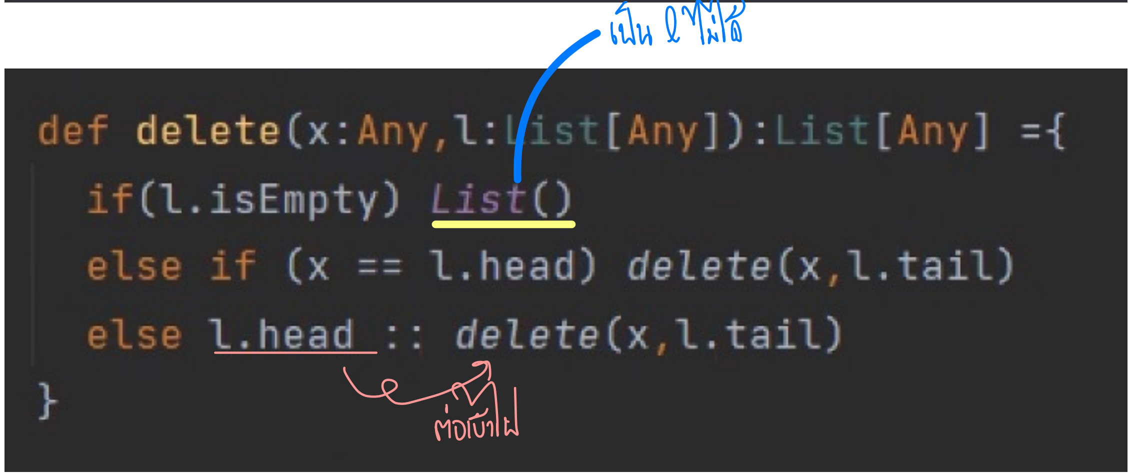
```
def length(l:List[Any]):Int ={
```



```
package package3

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

  def member(x:Any , l :List[Any]): Boolean ={
    if(l.isEmpty) return false
    if(l.head == x) return true
    member(x,l.tail)
  }
```



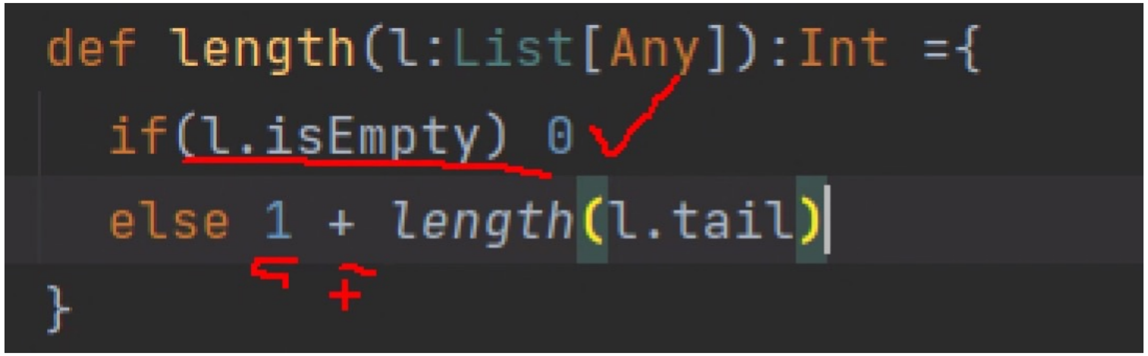
```
object Sorted {
  val listNum = List(1, 2, 3, 3, 5)
  val listNum2 = List(4, 2, 3, 4, 5)
  val listNum3 = List(1, 2, 0, 4, 5)
  val listNum4 = List()
  val listNum5 = List(4)

  def sorted(l: List[Int]):Boolean = {
    if(l.isEmpty || l.length ==1) return true
    return ((l.head <= (l.tail).head) && sorted(l.tail))
  }

  def main(args: Array[String]): Unit = {
    println(sorted(listNum))
    println(sorted(listNum2))
    println(sorted(listNum3))
    println(sorted(listNum4))
    println(sorted(listNum5))
```



```
def delete(x:Any,l:List[Any]):List[Any] ={
  if(l.isEmpty) List()
  else if (x == l.head) delete(x,l.tail)
  else l.head :: delete(x,l.tail)
}
```

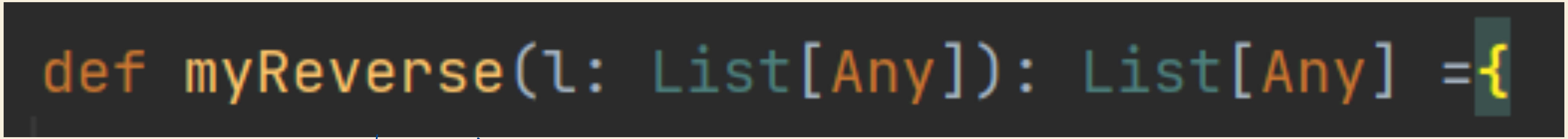
เป็น l ไม่ได้

ต่อเข้าไป



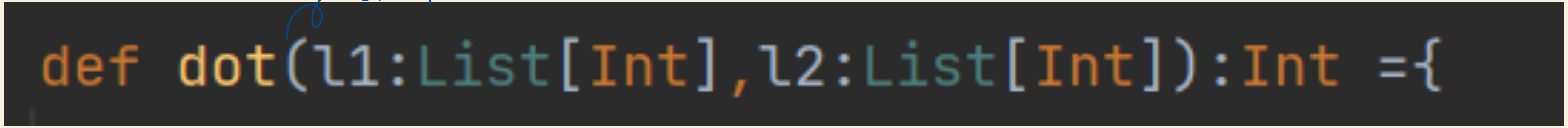
```
def length(l:List[Any]):Int ={
  if(l.isEmpty) 0
  else 1 + length(l.tail)
}
```

EXERCISE - CONT

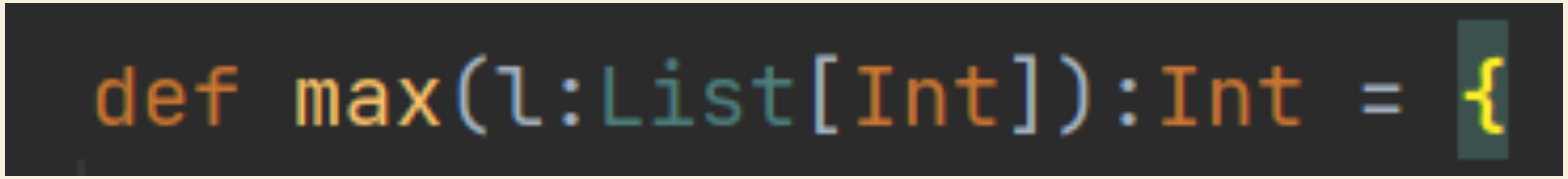


```
def myReverse(l: List[Any]): List[Any] ={
```

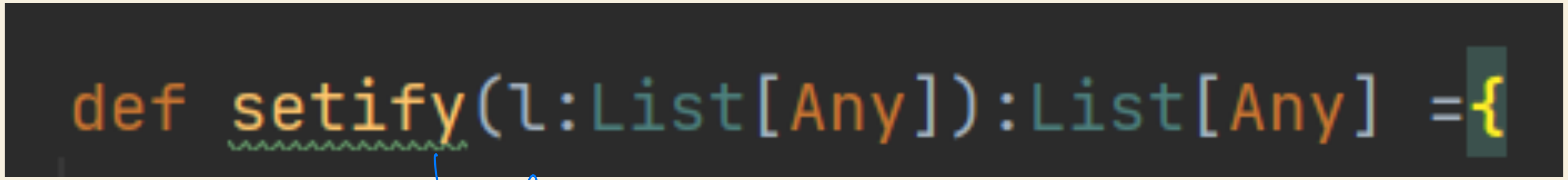
dot product



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



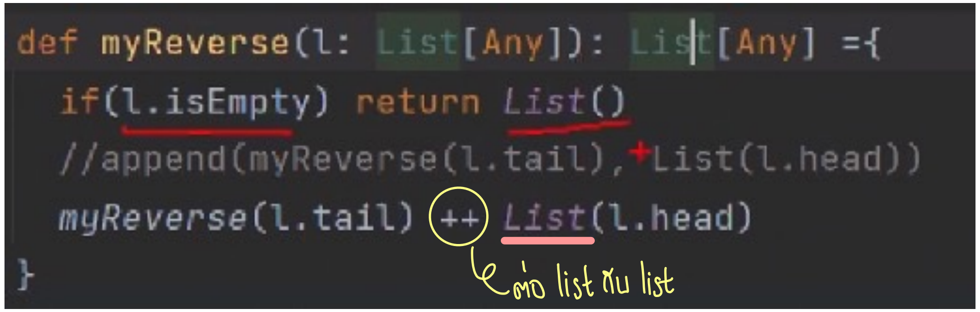
```
 def max(l:List[Int]):Int = {
```



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

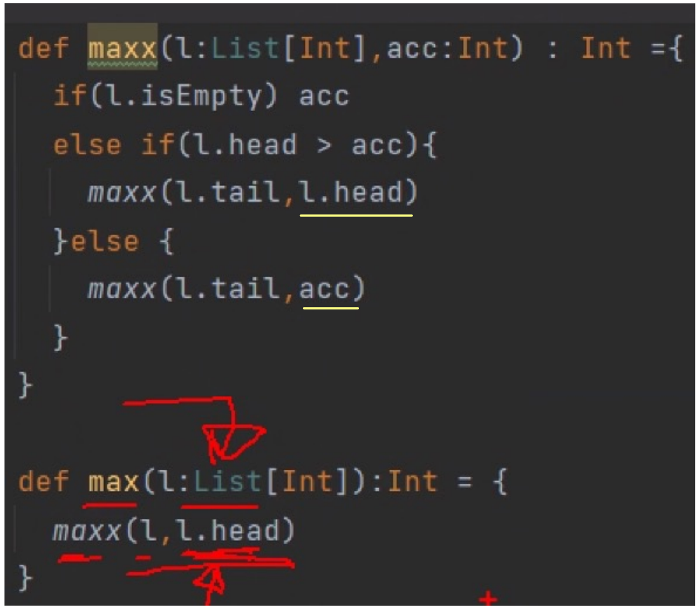
ทำให้เป็น set

(กำจัดตัวซ้ำออก)



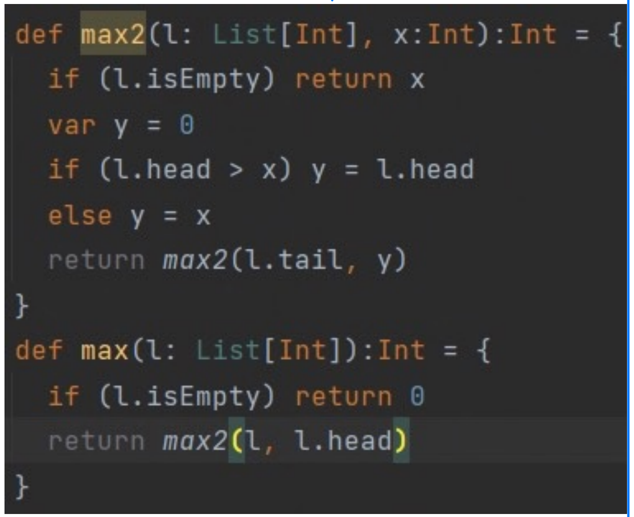
```
def myReverse(l: List[Any]): List[Any] ={
  if(l.isEmpty) return List()
  //append(myReverse(l.tail), List(l.head))
  myReverse(l.tail) ++ List(l.head)
}
```

ต่อ list กับ list

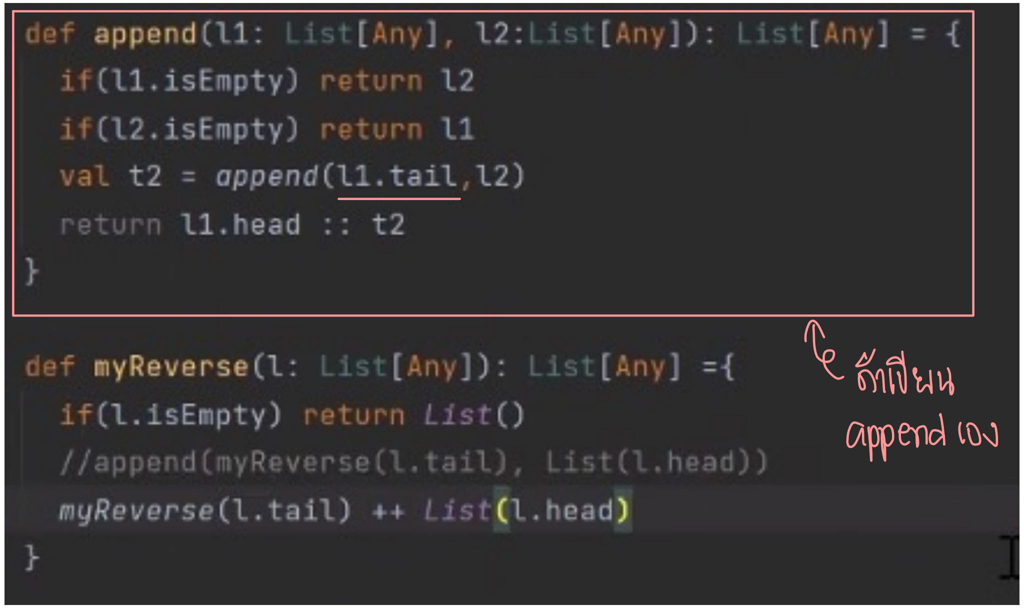


```
def maxx(l:List[Int],acc:Int) : Int ={
  if(l.isEmpty) acc
  else if(l.head > acc){
    maxx(l.tail,l.head)
  }else {
    maxx(l.tail,acc)
  }
}

def max(l:List[Int]):Int = {
  maxx(l,l.head)
}
```

(ทำเอง)

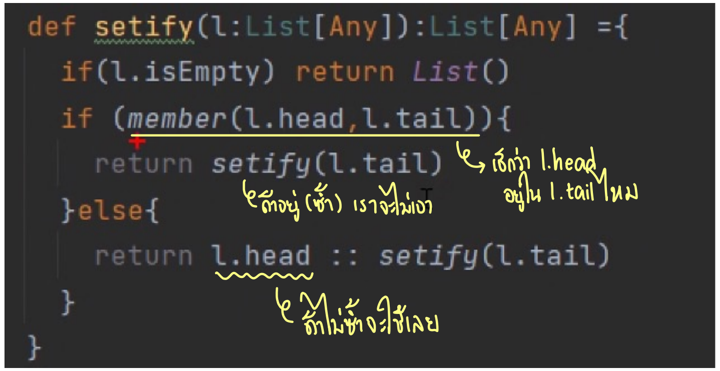


```
def max2(l: List[Int], x:Int):Int = {
  if (l.isEmpty) return x
  var y = 0
  if (l.head > x) y = l.head
  else y = x
  return max2(l.tail, y)
}
def max(l: List[Int]):Int = {
  if (l.isEmpty) return 0
  return max2(l, l.head)
}
```



```
def append(l1: List[Any], l2:List[Any]): List[Any] = {
  if(l1.isEmpty) return l2
  if(l2.isEmpty) return l1
  val t2 = append(l1.tail,l2)
  return l1.head :: t2
}

def myReverse(l: List[Any]): List[Any] ={
  if(l.isEmpty) return List()
  //append(myReverse(l.tail), List(l.head))
  myReverse(l.tail) ++ List(l.head)
}
```

ถ้าเขียน append เอง

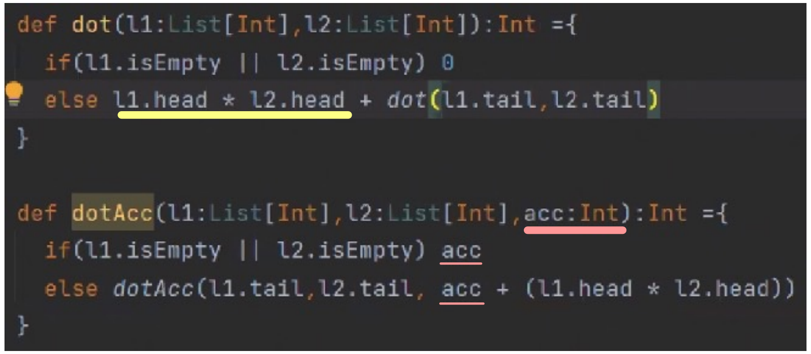


```
def setify(l:List[Any]):List[Any] ={
  if(l.isEmpty) return List()
  if (member(l.head,l.tail)){
    return setify(l.tail)
  }else{
    return l.head :: setify(l.tail)
  }
}
```

เช็คว่า l.head อยู่ใน l.tail ไหม

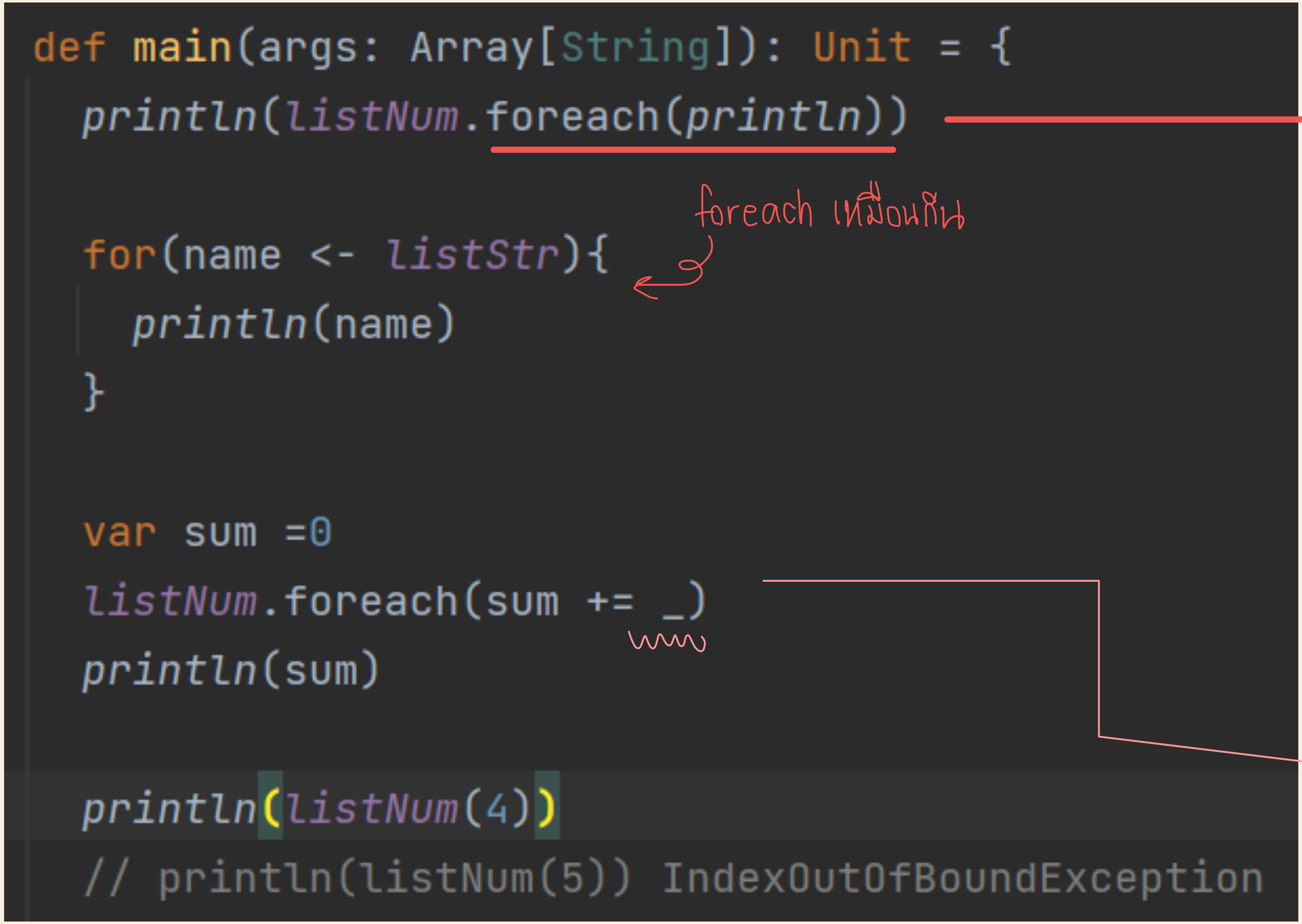
ถ้าอยู่ (ซ้ำ) เราจะไม่เอา

ถ้าไม่ซ้ำจะใช้เลย



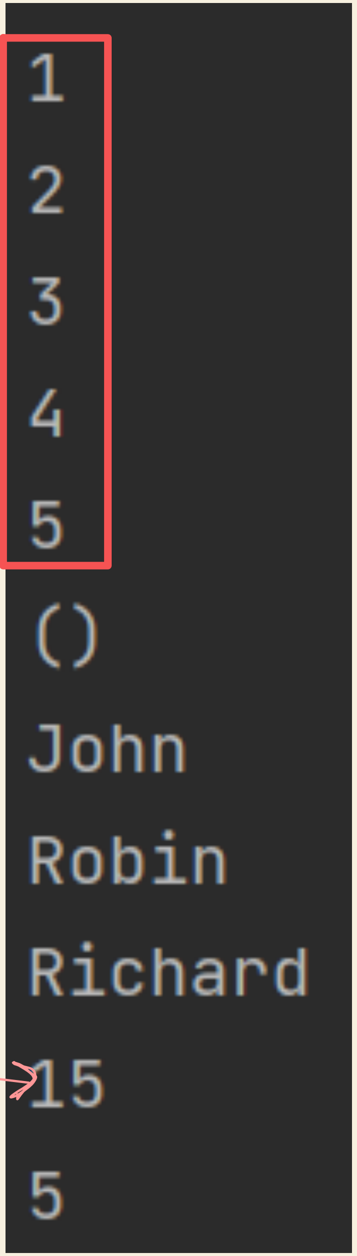
```
def dot(l1:List[Int],l2:List[Int]):Int ={
  if(l1.isEmpty || l2.isEmpty) 0
  else l1.head * l2.head + dot(l1.tail,l2.tail)
}

def dotAcc(l1:List[Int],l2:List[Int],acc:Int):Int ={
  if(l1.isEmpty || l2.isEmpty) acc
  else dotAcc(l1.tail,l2.tail, acc + (l1.head * l2.head))
}
```

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
```

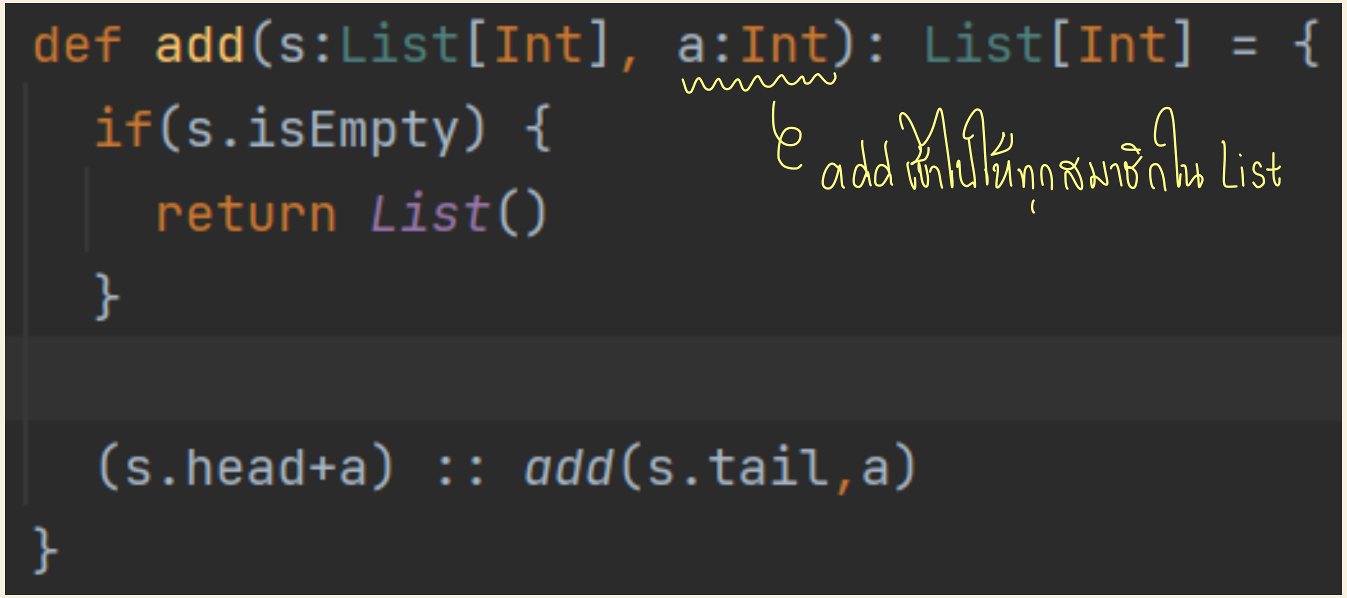
foreach เหมือนกัน



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

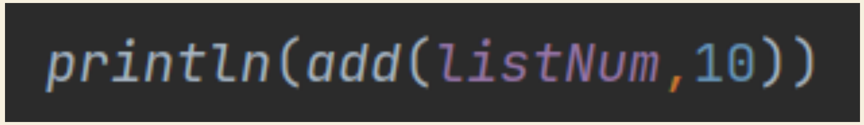
ITERATE TO MODIFY A LIST?

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

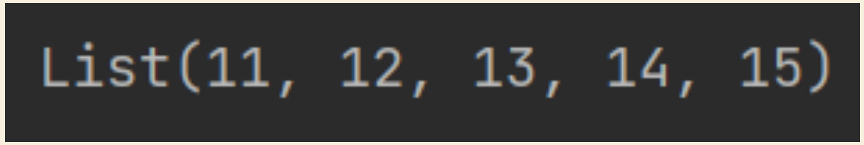


```
def add(s:List[Int], a:Int): List[Int] = {
  if(s.isEmpty) {
    return List()
  }

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

add เข้าไปให้ทุกสมาชิกใน List

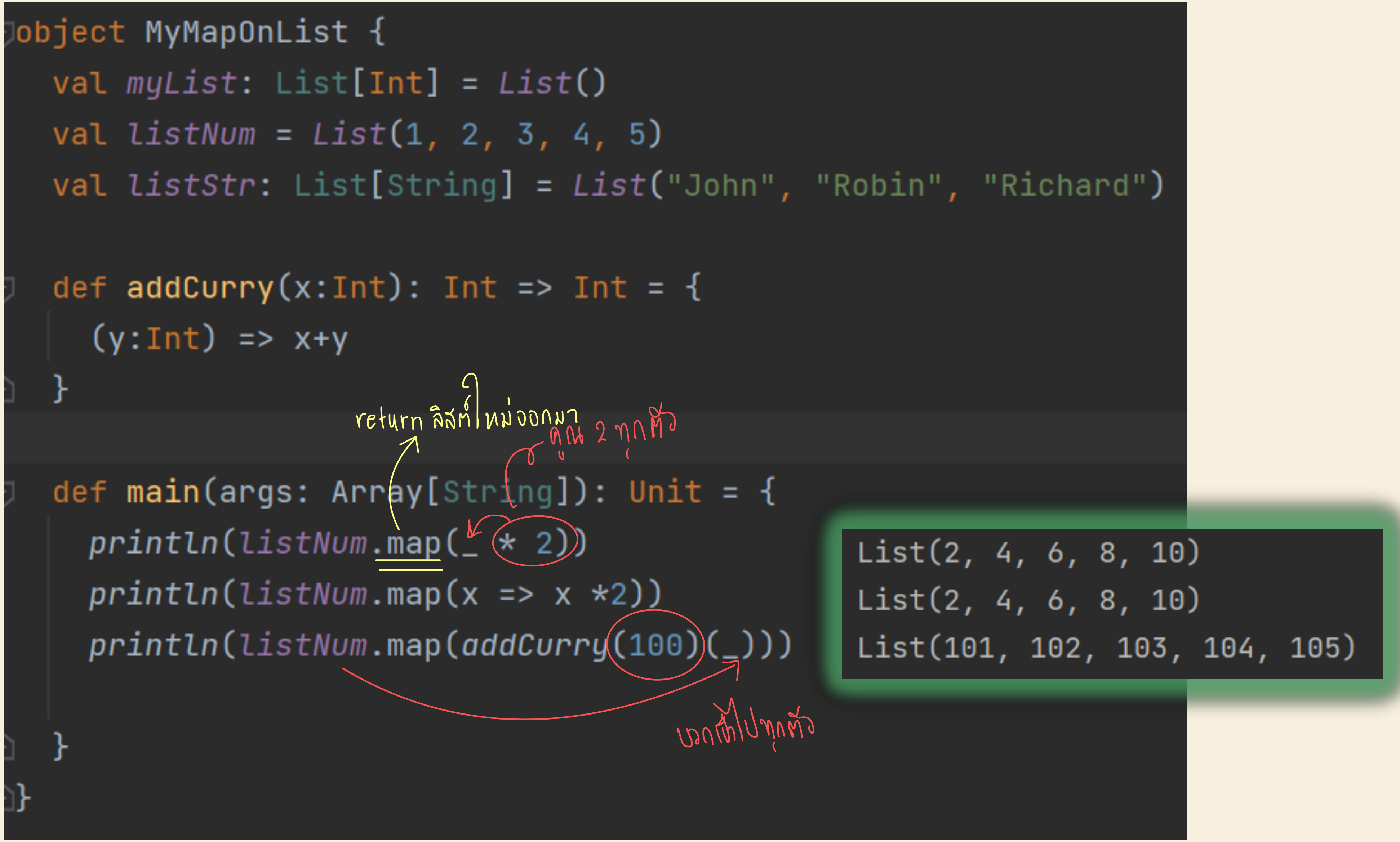


```
println(add(listNum,10))
```



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

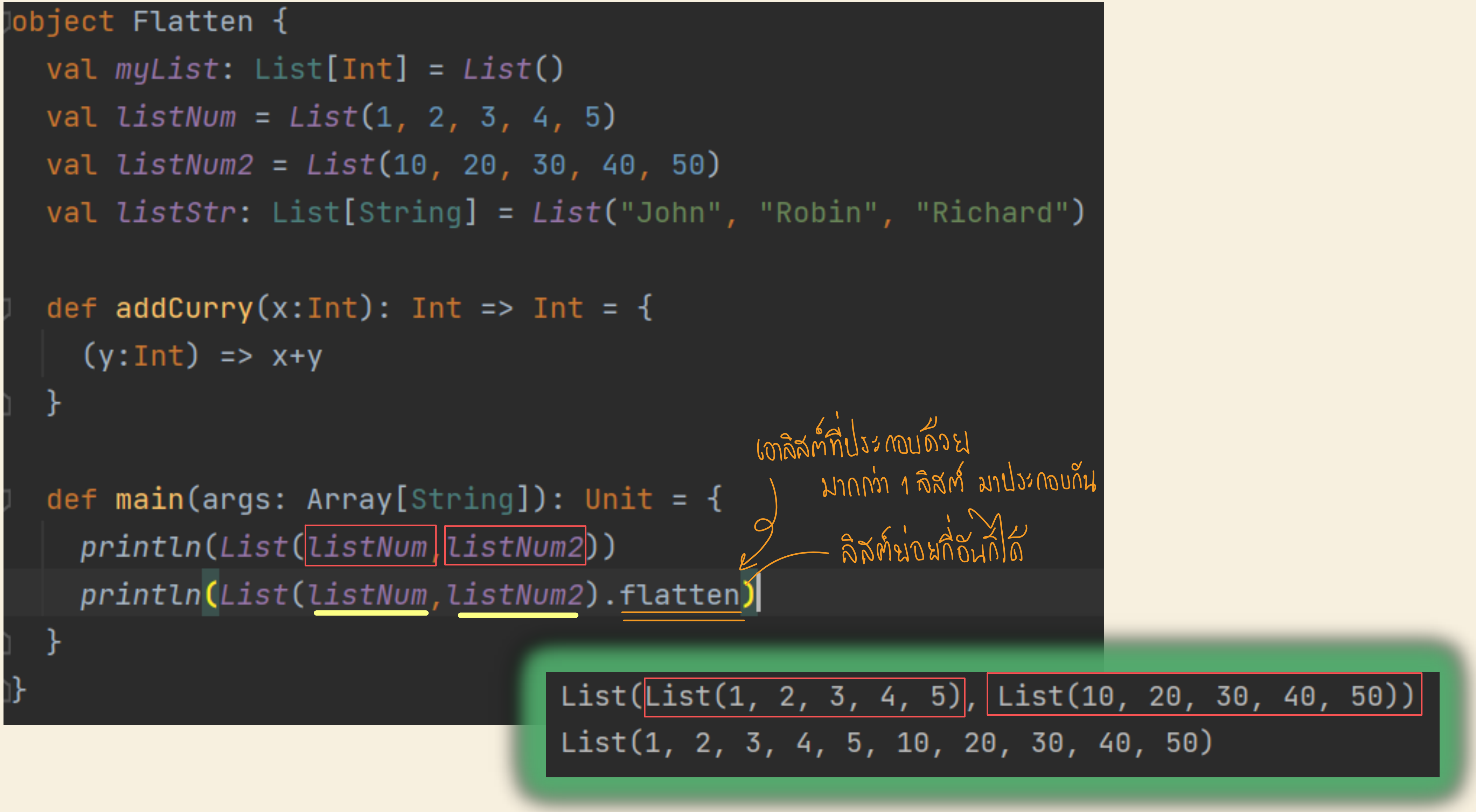
HIGHER ORDER METHODS
MAP



```
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



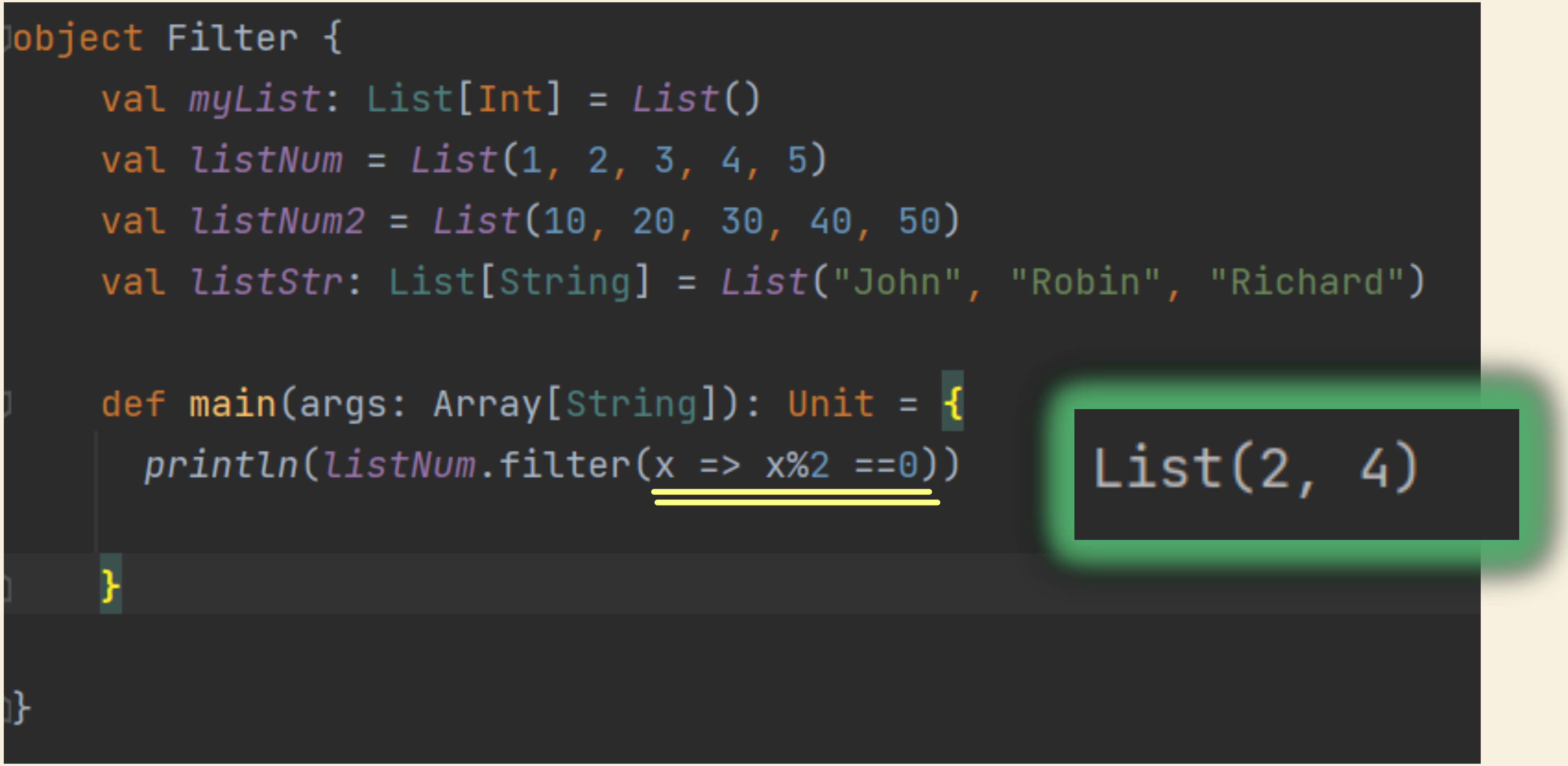
```scala
object Flatten {
  val myList: List[Int] = List()
  val listNum = List(1, 2, 3, 4, 5)
  val listNum2 = List(10, 20, 30, 40, 50)
  val listStr: List[String] = List("John", "Robin", "Richard")

  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)
  }
}
```

เอาลิสต์ที่ประกอบด้วย
มากกว่า 1 ลิสต์ มาประกอบกัน

ลิสต์ย่อยกี่อันก็ได้

```
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 = กรองเอาเฉพาะที่ตรงตามเงื่อนไข



```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)
```