

10B-Intro to Scala

Scala expressions

- First of all, recall from *10A-Functional Programming Scala* that Scala programs are expressions.
- So let's take a look at an expression. If you have installed the REPL, you can follow along with me.

```
scala> 1 + 1  
res0: Int = 2
```

```
scala> res0 * res0  
res1: Int = 4
```

```
scala> 2 * 2  
res2: Int = 4
```

Expressions (2)

```
scala> "Hello World!"  
res3: String = Hello World!
```

```
scala> val hw = "Hello World!"  
hw: String = Hello World!
```

```
scala> "Hello World!\n" * 3  
res4: String =  
"Hello World!  
Hello World!  
Hello World!  
"
```

```
scala> s"$res3\n" * 3  
res5: String =  
"Hello World!  
Hello World!  
Hello World!  
"
```

s means we can use & sign

Refactoring: extraction

```
scala> def square(x: Int) = x * x
```

```
square: (x: Int)Int
```

```
scala> square(2)
```

```
res6: Int = 4
```

```
scala> square("Hello World!")
```

```
<console>:13: error: type mismatch;
```

```
found   : String("Hello World!")
```

```
required: Int
```

```
    square("Hello World!")
```

^

```
scala> def square(x: Double) = x * x
```

```
square: (x: Double)Double
```

```
scala> square(2)
```

```
res8: Double = 4.0
```

Lists

```
scala> List(1,2,3)
```

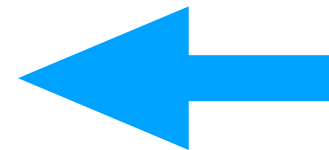
```
res9: List[Int] = List(1, 2, 3)
```

```
scala> res9 foreach println
```

```
1
```

```
2
```

```
3
```



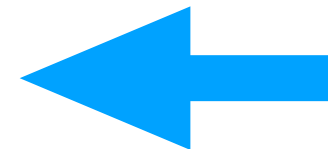
This is *not* an expression!
We are invoking a side-effect

```
scala> def square(x: Int) = x * x
```

```
square: (x: Int)Int
```

```
scala> xs map square
```

```
res16: List[Int] = List(1, 4, 9)
```



This *is* an expression!

Cuz we get back a new list

```
scala> xs sum
```

```
res18: Int = 6
```

Lists (2)

```
scala> for (x <- xs) yield square(x)  
res19: List[Int] = List(1, 4, 9)
```

This has the same effect as the map expression above



```
scala> xs :+ 4  
res20: List[Int] = List(1, 2, 3, 4)
```

Add 4 on the right hand side of list

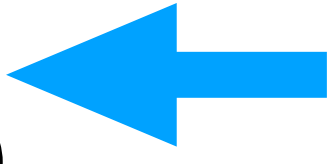
This is also an expression: the original xs doesn't change



```
scala> 0 +: xs  
res21: List[Int] = List(0, 1, 2, 3)
```

Add 4 on the left hand side of list

Note that the “:” in an operator associates left or right with the collection



Lists (3)

- Writing our own *sum* method:

```
scala> def sum(xs: Seq[Int]) = xs match { case Nil => 0; case h :: t => h + sum(t) }  
<console>:13: error: recursive method sum needs result type
```

```
def sum(xs: Seq[Int]) = xs match { case Nil => 0; case h :: t => h + sum(t) }  
                        ^
```

```
scala> def sum(xs: Seq[Int]): Int = xs match { case Nil => 0; case h :: t => h + sum(t) }  
sum: (xs: Seq[Int])Int
```

```
scala> sum(res20)
```

```
res22: Int = 10
```