

CS 94SI Syntax Handout #1 – Pure Functional Programming

Functions

Function definitions take the following form:

```
def functionName(arg1: Type1, arg2: Type2): ReturnType = functionDefinition
```

We'll be typing these things into the interpreter as we go along. The **code** we input will be in **bold**, the rest is the interpreter output.

The `timeTwo` function on `Ints` can be defined as:

```
scala> def timesTwo(n: Int): Int = n * 2  
timesTwo: (Int)Int
```

Function application is as in Java:

```
scala> timesTwo(10)  
res0: Int = 20
```

Thanks to local type inference, in this case we can leave off the return type:

```
scala> def timesTwo(n: Int) = n * 2  
timesTwo: (Int)Int
```

Recursive Functions

Recursive functions must always specify their return types:

```
scala> def factorial(n: Int) = if (n == 0) 1 else n * factorial(n - 1)  
<console>:5: error: recursive method factorial needs result type  
scala> def factorial(n: Int): Int = if (n == 0) 1 else n * factorial(n - 1)  
factorial: (Int)Int  
scala> factorial(10)  
res1: Int = 3628800
```

Higher-Order Functions

Functions can take other functions as arguments. The `applyFn` function takes a single-argument function (`fn`) on `Ints` that returns an `Int` (that is, of type `Int => Int`) and an `Int` argument (`arg`) and applies that function to that argument.

```
scala> def applyFn(fn: Int => Int, arg: Int) = fn(arg)  
applyFn: ((Int) => Int, Int) Int  
scala> applyFn(timesTwo, 10)  
res2: Int = 20  
scala> applyFn(factorial, 10)  
res3: Int = 3628800
```

Anonymous Functions

Anonymous functions can be defined with the syntax:

```
(arg1: Type1, arg2: Type2) => functionDefinition
```

For example, the anonymous function that multiplies a number by three might be defined as:

```
scala> (n: Int) => n * 3  
res4: (Int) => Int = <function>
```

And used like so:

```
scala> applyFn((n: Int) => n * 3, 10)  
res5: Int = 30
```

Sometimes Scala can even infer the types of the parameters:

```
scala> applyFn(n => n * 3, 10)  
res6: Int = 30
```

And there is also a convenient syntax for “anonymous parameters” using the underscore (`_`) character:

```
scala> applyFn(_ * 3, 10)  
res7: Int = 30
```

And that should be more than you need to know to complete the first assignment. Also remember that, if in doubt, the Java syntax usually works. (Hint: Like operators on Booleans.)