Unit 3c: Scala Functions

1 Functions

In Java "Functions" can only be defined as methods in a class.

Scala functions can be defined as methods in a class, but can also be defined outside a class (there's more to functional programming that this, though!); we can enter and call a function in the REPL. The syntax for parameters is rather like Java syntax "reversed"; the function returns the last expression evaluated and so we don't need *return* statement.

```
scala> def square(n:Int): Int = {
     | n*n
     | }
square: (n: Int)Int

scala> square(4)
res0: Int = 16
```

Note the "=" sign before the opening {.

```
scala> def power(value:Int, power:Int):Int = {
     |   var result = 1
     | for(i <- 0 to power - 1) {
     |     result = result * value
     | }
     | result
     | }
power: (value: Int, power: Int)Int

scala> power(2, 4)
res0: Int = 16

scala>
```

1.1 Default and named parameters

In Java you need to provide values for *all* parameters when calling a method; overloaded methods can implement different parameter lists – provides support for calling with "incomplete parameter lists"

```
public int add(int a, int b) {
      return a + b;
}

public int add(int a) {
      return a + 10;
}

int result1 = calculator.add(3,4);

int result2 = calculator.add(3);
```

calculator is an instance of the class in which these methods are defined; *result1 = 7* and *result2 = 13*.

Scala supports default and named parameters in function/method calls.

```
scala> def add(a:Int=5, b:Int=10):Int = {
     | a+b
     | }
add: (a: Int, b: Int)Int
scala> add(3,4)
res1: Int = 7
scala> add(3)
res2: Int = 13
scala> add(b=4)
res3: Int = 9
```