Function Review

- Use "def" to define a function. "def" does not evaluate the right-hand side until it is called. "var" and "val" evaluate the right-hand side when the name is defined.

```scala
scala> val x=1
x: Int = 1
scala> var y=1
y: Int = 1
scala> def z=1
z: Int
scala> z
res5: Int = 1
```

Define a function

- Provide argument types and function return type using colon:

```scala
scala> def fac(n:Int=0):Int = if(n <=0) 1 else n *fac(n-1)
fac: (n: Int)Int
```

- Use {} for function definition that has multiple expressions.

Anonymous Functions

- We can define a function without giving a name:

```scala
scala> (n:Int) => if (n<=0) 1 else n*fac(n-1)
res8: Int => Int = <function1>
```

- The part before the arrow ‘=>’ are the parameters of the function, whereas the part that following the ‘=>’ is the body.
Higher-Order Functions

- Functions which take other functions as parameters or return them as results are called higher-order functions.
- Function takes other function as argument:
  
  scala> Array(1,2,3,4).map(x:Int) => x+1
  res10: Array[Int] = Array(2, 3, 4, 5)

- Function returns a function output:
  
  scala> def mulBy(factor:Double) = (x[Double]) => factor*x
     mulBy: (factor: Double)Double => Double
  scala> val triple=mulBy(3)
  triple: Double => Double = <function1>

Function Short-Hand

scala> Array(1,2,3,4).map((x: Int) => x+1)//Array(1,2,3,4) makes it clear that x is Int
scala> Array(1,2,3,4).map((x) => x+1)//only 1 parameter, drop ()
scala> Array(1,2,3,4).map(x => x+1)//x is only used once in the function body, replace x with _
scala> Array(1,2,3,4).map(_+1)

Currying

- Currying is the process of transforming a function that takes two or more arguments into a function that takes one argument at a time.
- The curried function takes the first argument and returns a function that takes the second argument and return a function that takes the third argument and so on until the function with one argument to implement the function details.

Examples

scala> def add3(x:Int,y:Int,z:Int)=x+y+z
add3: (x: Int, y: Int, z: Int)Int
scala> add3(10,11,12)
res16: Int = 33
scala> def add2(x:Int,y:Int)=(z:Int)=>x+y+z
add2: (x: Int, y: Int)Int => Int
scala> add2(10,11)(12)
res18: Int = 33
scala> def add1(x:Int)=(y:Int)=>(z:Int) =>x+y+z
add1: (x: Int)Int => (Int => Int)
scala> add1(10)(11)(12)
res19: Int = 33
Why Curried Function?

- Reuse: a partially applied curried function can be passed around to be combined with different inputs to carry out the defined operations.
- Late evaluation: Sometimes not all inputs are available to carry out the defined operations.