Lecture Thursday 9/10/15

* First assignment is available on Friday 9/11.

Block Expressions and Assignments

* + In Java and C++, a block statement is a sequence of statements enclosed in { }.
  + You use a block statement whenever you need to put multiple actions in the body of a paragraph

Assignments

* In Scala, assignments have no value
* Well, strictly speaking, they have a value of type Unit
* Unit is similar to void in Java and C++
* A block that ends in an assignment, such as
* Multiple assignments does not compile
  + Var x =2
  + Var y=4
  + X = y = 24
    - The last bullet does not compile

Input and Output

* To print a value use print or println
  + Print(“Answer: “)
  + Println(42)
* Yields the same output as,
  + Println(Answer: “ + 42)
* To read a line of input from the console use the readLine function.
* To read a numeric value use: readInt, readDouble, readByte, readShort, readLong, readFloat, readBoolean, or readChar.
* Sample program line by line (ignore capital letters at start).
  + import scala.io.StdIn.\_
  + val name = readLine(“Your name: “)
  + print(“Your age: “)
  + val age = readInt()
  + println(“Hello, $name! Next year you will be $(age+1)”)

Loops

* Scala has the same while nd do loops as Java and C++
  + while(n > 0) {
    - r = r \* n
    - n -= 1
  + }
* Scala does not have a Java/C++ for (int; test; update loops;
  + for(int j = 0; j < 10; j++);
* Rather you can use a for statement like this:
  + for( i <- 1 to n)
    - r = r \* i

Loop Patterns

* When traversing a string or array, you often need a range from 0 to n – 1. In that case, use the until method instead of the to method.
  + val x = “Hello”
  + var sum = 0;
  + for (I <- 0 until s.length
    - sum += s(i)
* In the above example, it turns out there is a simpler way:
  + var sum = 0
  + for (ch <- “Hello”) sum += ch
  + NOTE: In Scala, loops are not used as much as in other languages. As you will see later in the course, you can often process the values in a sequence by applying a function to all of them

Advanced for Loops

* For loops can have multiple generators:
  + for(I <- to 3; j<- to 3) print ((0\*i\*j) + “ “)
  + //prints 11 12 13 21 22 23 31 32 33
* Each generator can have a guard
  + for (i<- 1 to 3; j <- 1 to 3 if I != j) print ((10 \* I + j) + “ “)
  + //Prints 12 13 21 23 31 32

For Comprehensions

* When the body of the for loop starts with yield, then the loop constructs a collection of values, one for each iteration:
  + for( I <- 1 to 10) yield I % 3
* This type of loop is called a for comprehension.
* The generated collection is compatible with the first generator.
  + for( c<- “Hello”; I <- 0 to 1) yield (c+i).toChar
  + //Yields ”Hieflmlmop”
* ecause for comprehensions can get busy, there is alternate syntax using { } along with newlines to make it more clear:
  + for {
    - I <- to 3
    - from = 4 – i
    - j <- from to 3
  + } yield (19\*i+j) +””
  + //Yields a vector

Functions

* To define a function, you specify the function’s name, parameters and body like this:
  + def abs(x: Double) = if (x >= 0) x else –x
* If the body is more than one expression:
  + def fac(n: Int) = {
    - var r = 1
    - for (I <- 1 to n) r = r \* i
    - r
  + }
* There is no need to return a keyword (don’t use return)

Recursive Functions: Reread the slide

Default Arguments

* You can provide default arguments for functions that are used when you don’t specify explicit values
  + def decorate (str: String,
    - left: String = “[“,
    - right: String = “]”)
    - Reread the slide for more info

Named Arguments

* You can also specify the parameter names when you supply the arguments like this:
  + decorate (left = “<<<”, str = “Hello”, right = “>>>”

Variable arguments

* Sometimes it is convenient to implement a function that can take a variable number of arguments
  + def sum(args: Int\*) = (
    - var result = 0
    - for (arg <- args) result += arg
    - result
  + }
* You can call this function with as many arguments as you like”
  + sum(1 , 4, 9, 16, 25)

Procedures

* Scala has a special notation for a function that returns no value. If the function body is enclosed in braces without a preceding = symbol, then the return type is Unit.
* The following procedure prints a string inside a box:
  + def box(s: String) {
    - val border = “-“ \* s.length + “--\n”
    - print ln(border + “I” + s + “I\n” + border
  + }
  + Prints a boxed in hello

Scala Arrays

* Scala has arrays!
* Java and C++ programmers often choose an array or a close relative (Read the slide too fast 5 me)

Fixed- Length arrays

* If you need an array whose length doesn’t change, use the array type in scala
  + val nums = new array[Int](10)
    - //An array of 10 integers starting from 0
  + val a = new Array [String](10)
    - //An array of 10 strings starting from 0
  + val s = Array(“Hello”,”World”)
    - //An Array[String] of length 2 – the type is inferred
    - //Note: No newe when you supply initial values
  + s(0) = “Goodbye”
    - //Array(“Goodbye”, “World”)
    - //Use () instead of [] to access elements

Variable-Length Arrays

* Java has ArrayList<T> and C++ has vector that grow and shrink on demand
* The equivalent in Scala is called ArrayBuffer[T]
* To use an ArrayBuffer you need to import it from Scalas mutable (Read the rest of the slides)