Read File in Scala

import scala.io.Source  
val filename = "fileopen.scala"  
for (line <- Source.fromFile(filename).getLines()) { println(line) }

**Spaghetti code** is a pejorative phrase for unstructured and difficult to maintain source **code**, broadly construed.

**Creating an Array  
val** arr = **new** Array[Int](3)  
arr(0)=1  
arr(1)=3  
arr(2)=300  
// reading values of arrays  
arr(0)  
arr(1)  
Parameterized Creation or Parameterization=> configuring an instance during the time  
of creation  
Type Parameterization means providing the Type in [ ] brackets  
Now the type of value “arr” is Array[Int]

// using a factory method  
val arr = Array(1,2,3,4)  
// the above is actually  
val arr = Array.apply(1,2,3,4)

**val** evenList = List(2,4,6,8)  
**val** anotherList = *List*.range(1,5)

**Difference between?**

Val somelist = 1 :: 2 :: Nil // cons operator adding at the head, fast

Val anotherList = somelist :+ 3 // adding 3 at the tail

1 :: 2 :: 3 // error compilation error “ operator :: not part of Int”

1 :: 2 :: 3 :: Nil // List[Int]

List1 :: 3 // error

List1 :+ 4 // list(1,2,3,4)

List1 :: 3 :: Nil // List(List(1,2,3),3)

List1 ::: list2

//take  
**val** li = *List*(1,2,3,4,5,6)  
li.head  
li.tail  
li.init  
li.last  
//generalizes init  
li.take(3)  
//generalizes tail  
li.drop(3)  
li.indices  
li.indices zip li  
li.toString  
li.mkString(“,”)

**//flatten  
val** lol = List(List(1,2,3), List(4,5,6))  
lol.flatten

// lol has to be type List[List[<BasicType like Int String>]]

// List are immutable list(i) = 56 // not possible

**Variable length parameters  
def** f1(x :Int\*)=println(x)  
**var** myIntArr = Array(1,2,3,4)  
f1( Array(1,2,3,4):\_\* )

**Tail recursion – Scala limitations**⮚Tail recursion optimization in scala is limited due to JVM  
⮚Not always something can be tailrec-ed  
⮚Examples –  
1. indirect recursion  
2. Partial functions  
⮚Tail call optimizations limited to situations in which a method or nested function  
calls itself directly as its last operation without going through a function value or  
some other intermediary

**tailrec annotation  
import** scala.annotation.tailrec  
@tailrec  
**def** summer(x :Int) :Int = {  
**if** (x ==0) 0  
**else** x + summer(x-1)  
}  
//Gives an error which says:  
error: could not optimize @tailrec annotated method factorial: it  
contains a recursive call not in tail position

Optimize Recursive function using local tail recursion functions

**Introducing function type**Introducing “type” of a function  
Introducing => . Can be read as “transforms”

scala> type fn = (Int) => Int

defined type alias fn

scala> def sq(x:fn, y:Int):Int = x(y) \* x(y)

sq: (x: fn, y: Int)Int

scala> sq(a=>2\*a, 3)

res19: Int = 36

**Function Literals**

**(x :Int) => x+1**// Function literal assigned to a variable. Now it beings to be useful  
**val a = (x :Int) => x+1  
a(3)**