Chapter 10 : Loops

Exercises:

1. Load a text file in Scala and see how many times a particular term appears in it. Try implementing this exercise using a for and a while loop.
2. Try using loops in a function (e.g., one that prints odd numbers from 0 to 200) and invoke that function in your main program.
3. Try assigning a for or while loop to a variable. Understand what happens when you do so.
4. Try using two variables in the () after for. Explore the prospects of these and see how you can use them.
5. Try nesting one for loop into another.
6. Try the good old sorting algorithms (e.g., bubble sort, merge sort) using for and while loops in Scala.

Answers :

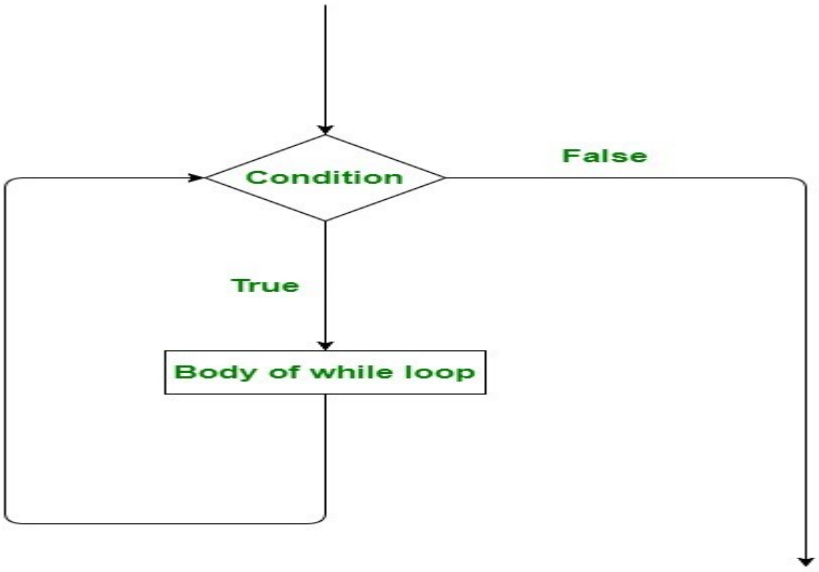
➔ **while loop**

While programming there might be situation which we need to repeat until and unless a condition is met. In these cases, while loop is used. A while loop generally takes a condition in parenthesis. If the condition is True then the code within the body of the while loop is executed. A while loop is used when we don’t know the number of times we want the loop to be executed however we know the termination condition of the loop. The condition at which loop stops is called breaking condition.

Syntax:

```
while (condition)
{
   // Code to be executed
}
```

Flowchart:



➔ **do while loop**

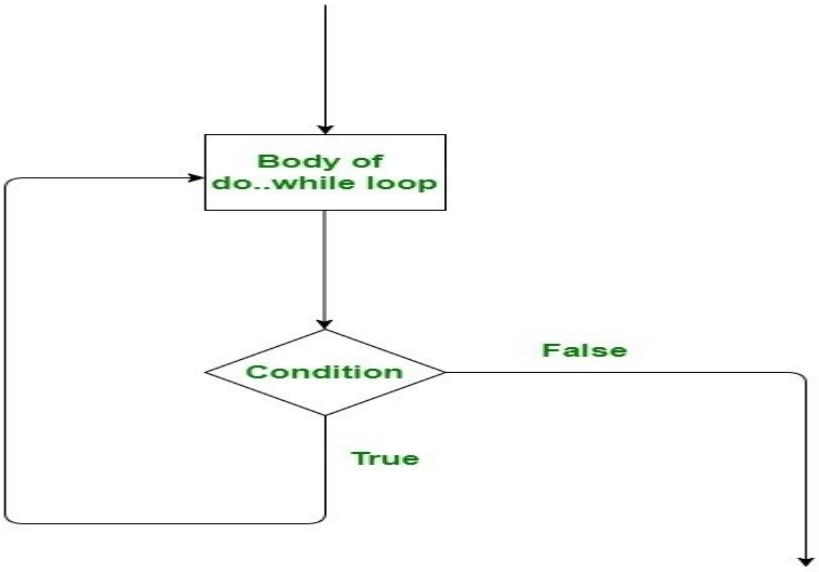
A do..while loop is almost same as a while loop. The only difference is that do..while loop runs at least one time. The condition is checked after the first execution. A do..while loop is used when we want the loop to run at least one time. It is also known as the exit controlled loop as the condition is checked after executing the loop. In while loop condition is placed at top of loop Whereas in do while loop condition is placed at end, due to this positioning of condition all statements under do while gets executes at least once.
Syntax:

```
do {

// statements to be Executed

} while(condition);
```

Flowchart:



➔ **for Loop**

for loop has similar functionality as while loop but with different syntax. for loops are preferred when the number of times loop statements are to be executed is known beforehand. There are many variations of “for loop in Scala” which we will discuss in upcoming articles. Basically, it is a repetition control structure which allows the programmer to write a loop that needs to execute a particular number of times.

Syntax:

for (i <- range){

// statements to be Executed

}

```
scala> (0 to 9) map { i => s"i = $i :: j = $i" } mkString "\n"
res44: String =
i = 0 :: j = 0
i = 1 :: j = 1
i = 2 :: j = 2
i = 3 :: j = 3
i = 4 :: j = 4
i = 5 :: j = 5
i = 6 :: j = 6
i = 7 :: j = 7
i = 8 :: j = 8
i = 9 :: j = 9

scala> for (i<- 0 to 9; j <- 0 to 9)
     | {
     |       println("i = " + i + " :: " + "j = " + j)
     |  }
i = 0 :: j = 0
i = 0 :: j = 1
i = 0 :: j = 2
i = 0 :: j = 3
i = 0 :: j = 4
i = 0 :: j = 5
```

- bubble sort

```
scala> :paste
// Entering paste mode (ctrl-D to finish)

  def bubbleSort(arr:Array[Int])={
    var temp=0
    for(i<- 1 until arr.length-1 ; j<- 1 until (arr.length-1-i)){
      if(arr(j-1)>arr(j)) {
        temp=arr(j-1)
        arr(j-1)=arr(j)
        arr(j)=temp
       }
     }
     arr
  }
var x = bubbleSort(Array(3, 60, 35, 2, 45, 320, 5))
    println(x)

// Exiting paste mode, now interpreting.

[I@55cc01ef
bubbleSort: (arr: Array[Int])Array[Int]
x: Array[Int] = Array(2, 3, 35, 45, 60, 320, 5)

scala>
```