

## Assignment - 26.1

### Task – 1:

Read a stream of Strings, fetch the words which can be converted to numbers. Filter out the rows, where the sum of numbers in that line is odd. Provide the sum of all the remaining numbers in that batch.

### Pre requisite

In this assignment we are going to read the strings which is captured in a port using **netcat** and hence we are installing the **netcat** in our server,

Command,

 **sudo yum install nc**

The below screen shot shows the successful installation of **netcat** using **yum**.

```
[acadgild@localhost ~]$ sudo yum install nc
Loaded plugins: fastestmirror, refresh-packagekit, security
Setting up Install Process
Loading mirror speeds from cached hostfile
 * base: ftp.iitm.ac.in
 * extras: ftp.iitm.ac.in
 * updates: ftp.iitm.ac.in
Resolving Dependencies
--> Running transaction check
--> Package nc.i686 0:1.84-24.el6 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====================================================================================================================================
 Package                Arch                Version              Size
=====================================================================================================================================
Installing:
 nc                     i686                1.84-24.el6          57 k
Transaction Summary
=====================================================================================================================================
Install      1 Package(s)

Total download size: 57 k
Installed size: 107 k
Is this ok [y/N]: y
Downloading Packages:
nc-1.84-24.el6.i686.rpm                                | 57 kB    00:00
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : nc-1.84-24.el6.i686                                1/1
  Verifying  : nc-1.84-24.el6.i686                                1/1

Installed:
 nc.i686 0:1.84-24.el6

Complete!
[acadgild@localhost ~]$ nc -lk 9999
```

Start listening the port **9999** using the below command,

Command,

 **nc -lk 9999**

```
^C
[acadgild@localhost ~]$ nc -lk 9999
HT
```




Start Spark shell with multi threads, in this case we are taking 4 multi threads.

Command,

```
 /home/acadgild/spark-2.2.1-bin-hadoop2.7/bin/spark-shell --master local[4]
```

## Answer – 1:


Step – 1 - Import all the spark streaming packages

```
 import org.apache.spark._  
 import org.apache.spark.streaming._  
 import org.apache.spark.streaming.StreamingContext._
```

```
scala> import org.apache.spark._  
import org.apache.spark._  
  
scala> import org.apache.spark.streaming._  
import org.apache.spark.streaming._  
  
scala> import org.apache.spark.streaming.StreamingContext._  
import org.apache.spark.streaming.StreamingContext._
```

Step – 2 - Define an Accumulator

Defining an accumulator “**EvenLines**” which will keep track of sum of number of word numbers in lines so far,



```
 val EvenLines = sc.accumulator(0)
```

```
scala> val EvenLines = sc.accumulator(0)  
warning: there were two deprecation warnings; re-run with -deprecation for details  
EvenLines: org.apache.spark.Accumulator[Int] = 0
```

Step – 3 - convert words to numbers

We are creating a RDD which maps the strings into the corresponding numbers, if we provide any word in the port **9999** which is not mapped, the numerical 0 will be returned.

Broadcast the newly created map,

```
 val wordstonumbers = map("Hi"->1, "This"->2, "is"->3, "Assignment"->4, "number"->5,  
"Twenty"->6,"it"->7, "about"->8, "spark"->9, "Streaming"->10)  
 val wordstonumbersbroadcast = sc.broadcast(wordstonumbers)
```

```
scala> val wordstonumbers = Map("Hi" -> 1, "This" -> 2, "is" -> 3, "Assignment"->4, "number"->5, "Twenty"->6, "it"->7, "about"->8, "spark"->9, "  
Streaming"->10)  
wordstonumbers: scala.collection.immutable.Map[String,Int] = Map(number -> 5, is -> 3, This -> 2, Streaming -> 10, it -> 7, Twenty -> 6, spark  
-> 9, Hi -> 1, Assignment -> 4, about -> 8)  
  
scala> val wordstonumbersbroadcast = sc.broadcast(wordstonumbers)  
18/01/15 12:54:49 WARN SizeEstimator: Failed to check whether UseCompressedOups is set; assuming yes  
wordstonumbersbroadcast: org.apache.spark.broadcast.Broadcast[scala.collection.immutable.Map[String,Int]] = Broadcast(0)
```

Step – 4 - create a function to return sum of word converted to number in a line

Create a function “**lineWordNumberSum**” where we are splitting a line based on blank space to get all the words in next. In the lookup value, we are determining corresponding numbers for a word in the **wordstonumbersbroadcast** and we adding the all the numbers.

Please see the code below

```
def lineWordNumberSum(line:String):Int = {  
var sum:Int = 0  
var words = line.split(" ")  
for (word <- words) sum += wordstonumbersbroadcast.value.get(word).getOrElse(0)  
sum  
}
```

```
scala> def lineWordNumberSum(line:String):Int = {  
|     var sum:Int = 0  
|     var words = line.split(" ")  
|     for (word <- words) sum += wordstonumbersbroadcast.value.get(word).getOrElse(0)  
|     sum  
| }  
lineWordNumberSum: (line: String)Int
```

### Step – 5 - Text Streaming

In this step, we are streaming the data as a string in a 5 seconds interval and return the stream. The streams are reading in a port 9999 which is listened

```
val ssc = new StreamingContext(sc, Seconds(5))  
val stream = ssc.socketTextStream("localhost", 9999)
```

```
scala> val ssc = new StreamingContext(sc, Seconds(5))  
ssc: org.apache.spark.streaming.StreamingContext = org.apache.spark.streaming.StreamingContext@a1ce8  
  
scala> val stream = ssc.socketTextStream("localhost", 9999)  
stream: org.apache.spark.streaming.dstream.ReceiverInputDStream[String] = org.apache.spark.streaming.dstream.SocketInputDStream@93161a
```

### Step – 6 - Processing the each RDD

Process each RDD in stream, we are converting the RDD to string. Consider the below scenarios, If it is not blank calculate corresponding word's number and sum them using the function

**lineWordNumberSum** and put as variable **numTotal**.

If **numTotal** is odd, print the provided line in the output, else add **numTotal** to accumulator **EvenLines** and print the sum.

Code,

```
stream.foreachRDD(line => {val lineStr = line.collect().toList.mkString("")  
if (lineStr != "") {var numTotal = lineWordNumberSum(lineStr) if (numTotal % 2 == 1) println(lineStr)  
else  
{EvenLines += numTotal  
println("Sum of lines with even word number so far =" + EvenLines.value.toInt)}}  
})
```

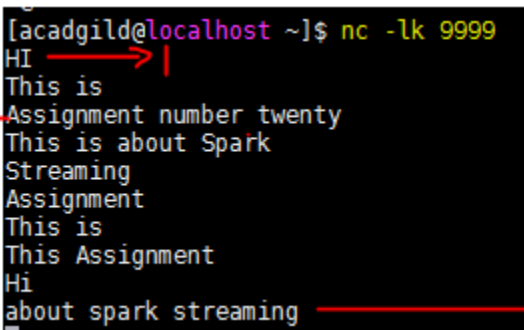
```
scala> stream.foreachRDD(line => {  
|     val lineStr = line.collect().toList.mkString("")  
|     if (lineStr != "") {  
|         var numTotal = lineWordNumberSum(lineStr)  
|         if (numTotal % 2 == 1) println(lineStr)  
|         else {  
|             EvenLines += numTotal  
|             println("Sum of lines with even word number so far =" + EvenLines.value.toInt)  
|         }  
|     }  
| } )
```

## Step – 7 - Spark Streaming

Now, Start the streams and wait till its termination

- `ssc.start()`
- `ssc.awaitTermination()`

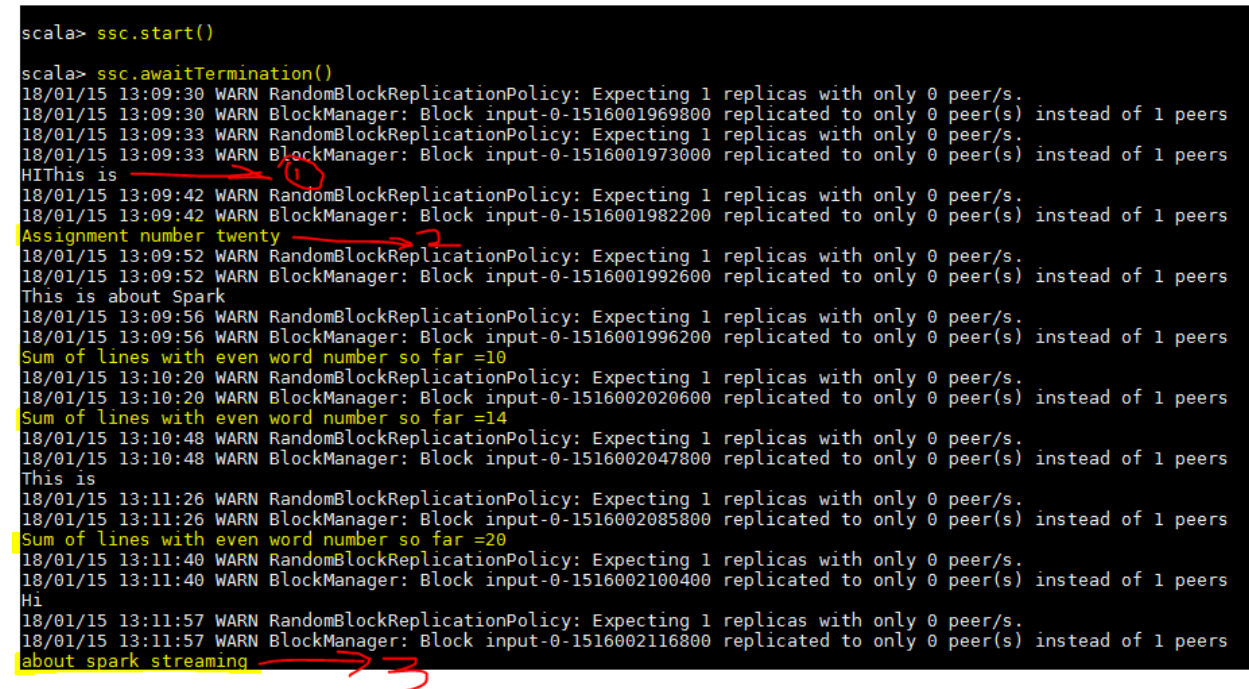
Start **netcat** in the Linux terminal, provide the texts which we determined in the map.



A screenshot of a netcat listener terminal window. The prompt is `[acadgild@localhost ~]$ nc -lk 9999`. The received messages are: `HI`, `This is`, `Assignment number twenty`, `This is about Spark`, `Streaming`, `Assignment`, `This is`, `This Assignment`, `Hi`, and `about spark streaming`. Red arrows and numbers are used for annotation: a red arrow points from the number '1' to the word 'HI'; a red arrow points from the number '2' to the word 'Assignment' in the third line; a red arrow points from the number '3' to the end of the last line.

1. We know that the string “Hi” has a number 1, when we type “Hi” in the port which is ODD, hence it will be displayed.
2. Same as, “Assignment number twenty” which has value of 15 which is again ODD and hence it is displayed,
3. For lines with even numbered word number, the **summation done so far** will be displayed. Please see the below screen shot,

## Expected Output



A screenshot of a Scala Spark Streaming application output. The output shows the execution of `ssc.start()` and `ssc.awaitTermination()`. It displays several warning messages from the BlockManager. The received messages are: `HI`, `This is`, `Assignment number twenty`, `This is about Spark`, `Sum of lines with even word number so far =10`, `Sum of lines with even word number so far =14`, `This is`, `Sum of lines with even word number so far =20`, `Hi`, and `about spark streaming`. Red arrows and numbers are used for annotation: a red arrow points from the number '1' to the word 'HI'; a red arrow points from the number '2' to the word 'Assignment' in the third line; a red arrow points from the number '3' to the end of the last line.