http://data-flair.training/blogs/hadoop-tutorial-for-beginners/

http://hadooptutorial.info/introduction-to-business-intelligence-and-data-warehousing/

http://www.hdfstutorial.com/hdfs-file-processing/

Extermely large dataset that is hard to deal with using Relational data base
  -stroge/cost
  -Search/Performance
  -Analytics and visualization

Traditional system find to diffcult to store and process large data in the cost-effective manner

3Vs or V3s stands for Volume, Velocity and Variety

A maximum of 4500 machines can be connected together using Hadoop (as per article on 2014)
A maximum of 25 Petabyte (1 PB = 1000 TB) data can be processed using Hadoop (as per article on 2014)

DataInput(I) : Reads bytes from the byte stream and converts the bytes data into java primitive types.
For example : readInt(): reads 4 bytes and returns int value. Similliary we have readXXX methods for all java primitive types.
readLine(): Reads the next line text from the stream.

DataOutput(I) : Converts java primitive types data to series of bytes and writes them to binary stream.
For example : write(int b) or write(String s)

Writable(I) : A serializable object which implements a simple, efficient, serialization protocol, based on DataInput and DataOutput.
Used Serialize the fields of this object to DataOutput stream. And Deserialize the fields of this object from DataInput stream.
Any key or value type in the Hadoop Map-Reduce framework implements this interface.

```
Methods : void write(DataOutput out) throws IOException
          void readFields(DataInput in) throws IOException
```

We have xxxWriters for all the java primitive types and also
MapWritable, SortedMapWritable,ArrayWritable and so on

```
Example : public class MyWritable implements Writable {
       // Some data
       private int counter;
       private long timestamp;

       public void write(DataOutput out) throws IOException {
         out.writeInt(counter);
         out.writeLong(timestamp);
       }

       public void readFields(DataInput in) throws IOException {
         counter = in.readInt();
         timestamp = in.readLong();
       }

       public static MyWritable read(DataInput in) throws IOException {
         MyWritable w = new MyWritable();
         w.readFields(in);
         return w;
       }
     }
```

WritableComparable(I) :

A Writable which is also Comparable.

WritableComparables can be compared to each other, typically via Comparators. Any type which is to be used as a key in the Hadoop Map-Reduce framework should implement this interface.

```
public interface WritableComparable<T> extends Writable, Comparable<T>
```

Example:

```
     public class MyWritableComparable implements WritableComparable {
       // Some data
       private int counter;
       private long timestamp;

       public void write(DataOutput out) throws IOException {
         out.writeInt(counter);
         out.writeLong(timestamp);
       }

       public void readFields(DataInput in) throws IOException {
         counter = in.readInt();
         timestamp = in.readLong();
       }

       public int compareTo(MyWritableComparable w) {
         int thisValue = this.value;
         int thatValue = ((IntWritable)o).value;
         return (thisValue < thatValue ? -1 : (thisValue==thatValue ? 0 : 1));
       }
     }
```

--Data redundancy is a condition created within a database or data storage technology in which the same piece of data is held in two separate places.