Now, let’s look at each operator closely with a different example than the one in the video.

filter(): This operation is useful to filter out the contents of an RDD based on a condition. Let’s consider the following example.

We are using a text file(document) that contains the following:

"Apache Spark™ is a unified analytics engine for large-scale data processing.
It is an open-source distributed-computing engine.
Spark provides a productive environment for data analysis because of its lightning speed and support for various libraries."

You can create a text file and upload it in a S3 bucket.
Then you can copy the file from the S3 bucket to HDFS using the following command:

```
hadoop distcp <S3 file location URI> .
```

The above command copies the S3 file to HDFS root directory.

You can then use the following command to put the text file from the root directory to HDFS:

```
hadoop fs -put "filename" /user/livy/<folder name if any>
```

For loading the data into the RDD you can use the following code:

```
rdd1 = sc.textFile(document)
rdd1.collect()
```

Now, let’s take a look at the output of this operation, as given below.

```
['Apache Spark is a unified analytics engine for large-scale data processing.',
 '',
 'It is an open-source distributed-computing engine.',
 'Spark provides a productive environment for data analysis because of its lightning speed and support for various libraries.']
```

Let’s use the filter operation to remove empty lines.

```
lines = rdd1.filter(lambda x:x!="")
lines.collect()
```

Let’s take a look at the output of this operation, as given below.

```
['Apache Spark is a unified analytics engine for large-scale data processing.', 'It is an open-source distributed-computing engine.', 'Spark provides a productive environment for data analysis because of its lightning speed and support for various libraries.']
```

You can see four elements in rdd1, where each element stores one line of the document string. You have also filtered out all the empty elements. This way, you are left with only those lines that contain at least one character.

map(): When applied to an RDD, this method will return a new RDD based on the operation performed on that RDD. Let’s consider the following example.

```
#You have already built lines RDD in the filter example.
words = lines.map(lambda x:x.split(" "))
words.collect()
```

Let's take a look at the following output of this code.

```
[['Apache', 'Spark', 'is', 'a', 'unified', 'analytics', 'engine', 'for', 'large-scale', 'data', 'processing.'],
 ['It', 'is', 'an', 'open-source', 'distributed-computing', 'engine.'],
 ['Spark', 'provides', 'a', 'productive', 'environment', 'for', 'data', 'analysis', 'because', 'of', 'its', 'lightning', 'speed', 'and', 'support', 'for', 'various', 'libraries.']]
```

On each element of an RDD, you applied an operation "split(" ")". This operation split this line on every occurrence of the " ". The number of elements in the RDD after the map() operation will always remain the same. Due to this, you now have an array of words as one element. Consider this one element of the RDD.

```
["It", "is", "an", "open-source", "distributed-computing", "engine."]
```

This is one element of the RDD ‘words’, which is an array of all the words separated after the map() operation.

flatMap(): This is another operation similar to map(), but the number of elements in the output can be different from the number of elements in the input. Let's consider the following example.

```
#You have already built lines RDD in the filter example.
words = lines.flatMap(lambda x:x.split(" "))
```

```
words.collect()
```

Let’s take a look at the output of this operation, as given below.

```
['Apache', 'Spark', 'is', 'a', 'unified', 'analytics', 'engine', 'for', 'large-scale', 'data', 'processing.', 'It', 'is', 'an', 'open-source', 'distributed-computing', 'engine.', 'Spark', 'provides', 'a', 'productive', 'environment', 'for', 'data', 'analysis', 'because', 'of', 'its', 'lightning', 'speed', 'and', 'support', 'for', 'various', 'libraries.']
```

In this RDD, every word is now a different element. In the map() the RDD word was divided into three elements, but in the case of flatmap() every word becomes a different element.

distinct(): This function will identify the unique elements in an RDD and put them in a new RDD. Let’s consider the following example.

Suppose you have to find the total number of distinct words in a document.

```
#You have already built lines RDD in the filter example.
words = lines.flatMap(lambda x:x.split(" "))
words.distinct().count()
```

Let’s take a look at the output of this code, as given below.

29

sorted(): This method is not used to perform an operation on an RDD but to sort the elements in a list. Let’s consider the following example:

```
rdd1 = sc.parallelize([1,5,1,3,2,3,5])
sorted(rdd1.distinct().collect())
```

Let's take a look at the output of this operation:

[1, 2 ,3, 5]

As you can see, you applied the sorted() method on the result of the collect() operation. Collect() returns a list of all elements of the RDD. Sorted() is applied on this list.