<https://www.dezyre.com/apache-spark-tutorial/pyspark-tutorial>

[Python](https://www.dezyre.com/data-science-in-python-tutorial/python-list-tutorial)is a powerful programming language for handling complex data analysis and data munging tasks. It has several in-built libraries and frameworks to do data mining tasks efficiently. However, no programming language alone can handle big data processing efficiently. There is always need for a distributed computing framework like [Hadoop or Spark](http://www.dezyre.com/article/hadoop-mapreduce-vs-apache-spark-who-wins-the-battle/83).

PySpark helps data scientists interface with Resilient Distributed Datasets in apache spark and python.

Py4J is a popularly library integrated within PySpark that lets python interface dynamically with JVM objects (RDD’s).

**conf**  
Holds all the necessary configuration files to run any spark application

**ec2**  
Holds the scripts to launch a cluster on amazon cloud space with multiple ec2 instances

Spark Context allows the users to handle the managed spark cluster resources so that users can read, tune and configure the spark cluster. Spark Content is used to initialize the driver program but since PySpark has Spark Context available as sc, PySpark itself acts as the driver program.

When performing collect action on a larger file the data is pulled from multiples nodes and there is a probability that the driver node could run out of memory

**Transformation and Actions in Apache Spark**

**Spark Transformations**

* **map():**applies changes on each line of the RDD and returns the transformed RDD as iterable of iterables i.e. each line is equivalent to a iterable and the entire RDD is itself a list
* **flatMap()**: applies changes to each line same as map but the return is not a iterable of iterables but it is only an iterable holding entire RDD contents.
* map(lambda line : line.split(" ")) retourne une liste de liste : la première liste représente le RDD // le fichier où chaque entrée est une ligne. Maintenant, chaque ligne est aussi une liste de mots. En utilisant flatMap, on aura une seule liste de chaines. La séparation en lignes sera éliminée.
* **filter() :** reduces the old RDD based on some condition

example: changesRDD.filter(lambda line : "ankurdave@gmail.com" in line)

* **Sample (withReplacement, fraction, seed):** picks sample RDD from a larger RDD. It is frequently used in Machine learning operations where a sample of the dataset needs to be taken. The fraction means percentage of the total data you want to take the sample from.
* **RDD.union(RDD):** merge two RDDs together if they have the same structure => une union des structure, comme on colle deux objets
* **RDD.intersection(RDD):** gives the common terms or objects from the two RDDs.
* **distinct():** helps getting rid of any ambiguities by picking out the lines from the RDD that are unique.

**Example:** Suppose that there are various movie nominations in different categories. We want to find out, how many movies are nominated overall

total\_nomination\_rdd = story\_rdd.union(direction\_rdd).union(screen\_rdd)

total\_nomination\_rdd.collect()

* **RDD.join(RDD):** joins two RDDs based on a common key => retourne toutes les valeurs de la même clé à partir des différents RDD invoqués.

**Spark Actions**

* reduce()
* collect()
* count()
* first()
* takeSample(withReplacement, num, [seed])

## ****RDD Partitions****

Parallelism is the key feature of any distributed system where operations are done by dividing the data into multiple parallel partitions. The same operation is performed on the partitions simultaneously which helps achieve fast data processing with spark. Map and Reduce operations can be effectively applied in parallel in apache spark by dividing the data into multiple partitions. A copy of each partition within an RDD is distributed across several workers running on different nodes of a cluster so that in case of failure of a single worker the RDD still remains available.

Degree of parallelism of each operation on RDD depends on the fixed number of partitions that an RDD has. We can specify the degree of parallelism or the number of partitions when creating it or later on using the repartition () and coalesce() methods.

When processing data with reduceByKey operation, Spark will form as many number of output partitions based on the default parallelism which depends on the numbers of nodes and cores available on each node.

**Accumulators** in spark are the global variable that can be shared across tasks. The scope of normal variables is just limited to a specific task so they can’t hold any update that needs to be aggregated from all other tasks. Under such circumstances, accumulators are used. They are write only variables which can be updated by each task and the aggregated result is propagated to the driver program. Example: var = sc.accumulator(0)

<https://stackoverflow.com/questions/37933023/how-to-call-python-script-in-spark>