livelessons
video instruction from technology experts

Lesson 1: Introduction to the Spark Environment

1.9 Introduction to RDDs: Functions, Transformations, and Actions

What is a RDD?

- **Resilient:** if the data in memory (or on a node) is lost, it can be recreated

- **Distributed:** data is chunked into partitions and stored in memory across the cluster

- **Dataset:** initial data can come from a file or be created programmatically

Note: RDDs are read-only and immutable, we will come back to this later…

Functions Deconstructed

```
import random
flips = 1000000

# lazy eval
coins = xrange(flips)

# lazy eval, nothing executed
heads = sc.parallelize(coins) \
          .map(lambda i: random.random()) \
          .filter(lambda r: r < 0.51) \
          .count()
```

Python Generator

Create RDD

Transformations

Action (materialize result)

Functions Deconstructed

```
import random
flips = 1000000

# lazy eval
coins = xrange(flips)

# lazy eval, nothing executed
heads = sc.parallelize(coins) \
            .map(lambda i: random.random()) \
            .filter(lambda r: r < 0.51) \
            .count()

 # create a closure with the lambda function
 # apply function to data
```

Closures

Spark Functions

Transformations

Lazy Evaluation
(does not immediately evaluate)

Returns new RDD

Actions

Materialize Data
(evaluates RDD lineage)

Returns final value
(on driver)

Transformations

```
# Every Spark application requires a Spark Context
# Spark shell provides a preconfigured Spark Context called `sc`
nums = sc.parallelize([1,2,3])

# Pass each element through a function
squared = nums.map(lambda x: x*x) # => {1, 4, 9}

# Keep elements passing a predicate
even = squared.filter(lambda x: x % 2 == 0) # => [4]

# Map each element to zero or more others
nums.flatMap(lambda x: range(x)) # => {0, 0, 1, 0, 1, 2}
```

Actions

```
nums = sc.parallelize([1, 2, 3])

# Retrieves RDD contents as a local collection
nums.collect() # => [1, 2, 3]

# Returms first K elements
nums.take(2) # => [1, 2]

# Count number of elements
nums.count() # => 3

# Merge elements with an associative function
nums.reduce(lambda: x, y: x + y) # => 6

# Write elements to a text file
nums.saveAsTextFile("hdfs://file.txt")
```

Functions Revisited

```
import random
flips = 1000000

# lazy eval
coins = xrange(flips)
```

nothing runs here

```
# lazy eval, nothing executed
heads_rdd = sc.parallelize(coins) \
               .map(lambda i: random.random()) \
               .filter(lambda r: r < 0.51)
```

```
head_count = heads_rdd.count()
```

Everything runs here

Functions Revisited

```
import random
flips = 1000000

# local sequence
coins = xrange(flips)

# distributed sequence
coin_rdd = sc.parallelize(coins)
flips_rdd = coin_rdd.map(lambda i: random.random())
heads_rdd = flips_rdd.filter(lambda r: r < 0.51)

# local value
head_count = heads_rdd.count()
```

RDD Lineage