Pyspark week-9

Few Python Fundamentals
==========

#include

import sys

These modules consists of functions, classes, methods and variables.

Functions are not bound to any classname or object.

methods are bound to classes or its objects..

A module is a python file which holds functions, classes, methods and variables.

import time

print(time.time())

import time as t

print(t.time())

#below will import all the things

import datetime

print(datetime.date.today())

#below is to import specific things and not all

from datetime import date

print(date.today())

so when we are importing specific functions then we can call the function directly.

Module

How to import it

How to call the functions within that module

How to import full module, Partial module

__name__ Global Variable

__main__

if we run the python file directly then the global variable __name__ is set to __main__

but if we have it indirectly then the value of __name__ is set to the name of the file

file1
======
```
print("name of this module is ", __name__)
```

file2
=======
```
import module1

print("in the second file the module name is ",__name__)

if __name__ == "__main__":
    print("Executed when invoked directly")
else:
    print("executed when imported")
```

```
# python comment

// scala comment
```

a = 5

In python, unlike statically typed languages like c or java, there is no need to specifically declare the data type of the variable. In dynamically typed languages like python, the interpreter itself predicts the datatype.

Named Function
==========

```
def sum(a,b):
    return a+b

total = sum(3,4)

print(total)
```

Anonymous functions are referred to as lambda functions in python.

Few more differences between scala & python
==========

1. Case

```
var totalCount = 10

//camel case in scala

total_count

//snake case
```

2. in scala single line comment is done using //

and for multiline we use
```
/*

*/
```

but in python we use # to comment

```
# this is a comment
```

3. in scala we were using foreach

but in python foreach is not allowed.

in scala

```
import org.apache.spark.SparkContext
```

```
object First extends App {

  val arr = Array(1,2,3,4,5)

  arr.foreach(println)


}
```

in python

```
arr = [1,2,3,4,5]

for a in arr:
    print(a)
```

4. scala

```
(x => x.split(" ")) //anonymous function
```

python

```
(lambda x : x.split(" ")) //lambda function
```

word count problem that we wrote in scala
==========

```
import org.apache.spark.SparkContext

object First extends App {

//Common lines
  val sc = new SparkContext("local[*]","wordcount")
  val input = sc.textFile("/Users/trendytech/Desktop/data/input/file.txt")

 //one input row will give multiple output rows
 val words = input.flatMap(x => x.split(" "))

 //one input row will give one output row only
```

```
 val wordCounts = words.map(x=> (x, 1))

 //take two rows , and does aggregation and returns one row
 val finalCount = wordCounts.reduceByKey((x,y) => x+y)

 //action
 finalCount.collect.foreach(println)

}
```

Pyspark code
==========

```
from pyspark import SparkContext

# common lines
sc = SparkContext("local[*]", "wordcount")
input = sc.textFile("/Users/trendytech/Desktop/data/input/file.txt")

# one input row will give multiple output rows
words = input.flatMap(lambda x: x.split(" "))

# one input row will be giving one output row
word_counts= words.map(lambda x: (x, 1))

final_count = word_counts.reduceByKey(lambda x, y: x + y)

result = final_count.collect()

for a in result:
    print(a)
```

==========

1. Change the logging level

```
sc.setLogLevel("ERROR")
```

2. __name__

__main__

3. Holding the program

from sys import stdin

stdin.readline()

4. DAG

localhost:4040

pyspark uses api library

scala was connnecting to spark core directly.

hence scala dag matches to our code but pyspark dag does not.

```
from pyspark import SparkContext

from sys import stdin

if __name__ == "__main__":
    # common lines
    sc = SparkContext("local[*]", "wordcount")
    # sc.setLogLevel("ERROR")
    input = sc.textFile("/Users/trendytech/Desktop/data/search_data.txt")

    # one input row will give multiple output rows
    words = input.flatMap(lambda x: x.split(" "))

    # one input row will be giving one output row
    word_counts= words.map(lambda x: (x, 1))

    final_count = word_counts.reduceByKey(lambda x, y: x + y)

    result = final_count.collect()
```

```
    for a in result:
        print(a)

else:
    print("Not executed directly")

stdin.readline()
```

========

1. Lowercase

```
word_counts= words.map(lambda x: (x.lower(), 1))
```

2. countByValue

```
from pyspark import SparkContext

from sys import stdin

if __name__ == "__main__":
    # common lines
    sc = SparkContext("local[*]", "wordcount")
    # sc.setLogLevel("ERROR")
    input = sc.textFile("/Users/trendytech/Desktop/data/search_data.txt")

    # one input row will give multiple output rows
    words = input.flatMap(lambda x: x.split(" "))

    # one input row will be giving one output row
    word_counts= words.map(lambda x: (x.lower()))

    final_count = word_counts.countByValue()

    print(final_count)

else:
    print("Not executed directly")
```

3. sortByKey

```
from pyspark import SparkContext
```

```
from sys import stdin

if __name__ == "__main__":
    # common lines
    sc = SparkContext("local[*]", "wordcount")
    # sc.setLogLevel("ERROR")
    input = sc.textFile("/Users/trendytech/Desktop/data/search_data.txt")

    # one input row will give multiple output rows
    words = input.flatMap(lambda x: x.split(" "))

    # one input row will be giving one output row
    word_counts= words.map(lambda x: (x.lower(), 1))

    final_count = word_counts.reduceByKey(lambda x, y: x + y).map(lambda x: (x[1],x[0]))

    result = final_count.sortByKey(False).map(lambda x: (x[1],x[0])).collect()

    for a in result:
        print(a)

else:
    print("Not executed directly")
```

4. sortBy

```
from pyspark import SparkContext

from sys import stdin

if __name__ == "__main__":
    # common lines
    sc = SparkContext("local[*]", "wordcount")
    # sc.setLogLevel("ERROR")
    input = sc.textFile("/Users/trendytech/Desktop/data/search_data.txt")

    # one input row will give multiple output rows
    words = input.flatMap(lambda x: x.split(" "))

    # one input row will be giving one output row
    word_counts= words.map(lambda x: (x.lower(), 1))
```

```
    final_count = word_counts.reduceByKey(lambda x, y: x + y)

    result = final_count.sortBy(lambda x: x[1], False).collect()

    for a in result:
        print(a)

else:
    print("Not executed directly")

stdin.readline()
```

1. Customers who have spent the most
==========

```
from pyspark import SparkContext

sc = SparkContext("local[*]","customer-orders")

rdd1 = sc.textFile("/Users/trendytech/Desktop/data/customer-orders.csv")

rdd2 = rdd1.map(lambda x: (x.split(",")[0],float(x.split(",")[2])))

rdd3 = rdd2.reduceByKey(lambda x,y: x+y)

rdd4 = rdd3.sortBy(lambda x: x[1],False)

result = rdd4.collect()

for a in result:
    print(a)
```

2. Movie Rating
==========

```
from pyspark import SparkContext
```

```
sc = SparkContext("local[*]","movie-data")

lines = sc.textFile("/Users/trendytech/Desktop/data/movie-data.data")

ratings = lines.map(lambda x: (x.split("\t")[2],1))

result = ratings.reduceByKey(lambda x,y: x+y).collect()

for a in result:
    print(a)
```

3. Average number of Friends
==========

```
def parseLine(line):
    fields = line.split(",")
    age = int(fields[2])
    numFriends = int(fields[3])
    return (age,numFriends)


from pyspark import SparkContext

sc = SparkContext("local[*]","FriendsByAge")

lines = sc.textFile("/Users/trendytech/Desktop/data/friends-data.csv")

rdd = lines.map(parseLine)
# (33,385) input

#(33,(385,1)) output

#(33,(3000,5))

#in scala we used to access the elements of tuple using x._1 , x._2

#in python we access the elements of tuple using x[0],x[2]

totalsByAge = rdd.mapValues(lambda x: (x,1)).reduceByKey(lambda x,y: (x[0]+y[0], x[1]+y[1]))

averagesByAge = totalsByAge.mapValues(lambda x:x[0]/x[1])

result = averagesByAge.collect()
```

```
for a in result:
    print(a)
```