Scala

val	var
→ creates immutable variable → its like final keyword in Java	→ creates mutable variable.

Data Types

String, Int, Short, Byte
char Float Double.

```
var name : String = "Pramod"
var name = "ABC"
var (s1, s2, s3, s4) = (10, 20, 30, 40)
```

Tripple quotes for multiline string.

```
var address = """ #26, jpnagar
Bangalore
India"""
```

Interpolator

s & f are the interpolators

```
s"$name - $lname, How are you?"
f"${name * 2}, come here!"
f"$pi %.2f"
```

Pattern Gaurding.

```
var day = "Monday"
var someday = "Friday"
day match {
  case someday => if (someday == "saturday") {
    println("working")
  }
}
```

Case Classes.

```
val x : Int = Random.nextInt(10)
x match
  case 0 => "zero"
  case 1 => "one"
  case _ => "other"
```

Pattern guards

Pattern guards are boolean expressions which are used to make cases more specific.

Eg:

```
notification match
  case Email(sender, _, _) if
  importantPeopleInfo.contains(sender)
  => "You got an email from
     someone"

  case SMS(number, _) if
  importantPeopleInfo.contains
  (number) => "You got an sms from
     special someone"
```

Eg:

```
var day : Any = 10
day match {
  case day : Int =>
      println("Integer");
  case day : Double =>
      println("Double");
  case day : String =>
      println("String")
  case _ => println("default")
}
```

Passing function as Parameter

```
def sum (sqrFn : Int ⇒ Int,
         a:Int, b:Int) : Long = {
  var sum = 0
  for (i <- a to b) {
    sum += sqrFn(i)
  }
  sum
}
def sqrFn (i:Int) = i*i
println (sum (sqr, 3, 30))
```

Type casting in Scala

a.asInstanceOf[Int]
Char
Double.

To convert to string above one won't work
var s = a.toString()

Scala main topics.

May 30 2023

→ Latest scala version 3.3.0 ~~2.13.10~~ 13/Oct/22

→ Type Inference (No need to mention data type)

→ Executing Scala
- scalac test.scala
- scala test

→ For Loop

```
object MainObject {
  def main(args: Array[String]) {
    for (a <- 1 until 10) {
      println(a)
    }
  }
}
```

- for (a ← 1 to 10 if a%2 == 0) {

Yield

```
var result = for (a <- 1 to 10) yield a
for (i <- result)
  println(i)
```

O/P 1 to 10

```
var List = List(1,2,3,4,5)
→ for (i <- list)
    println(i)
→ List.foreach {
    println;
  }
→ list.foreach(print)
→ list.foreach((element:Int)
    ⇒ print(element + " "))
```

→ In Scala we dont have primitive variables

→ Scala ~~in~~ in Nutshel has seamless Java interop

→ Type Inference

→ Concurrency & Distribution

→ Traits

→ Pattern Matching

→ Higher Order Fns.

→ Twitter announced that it had switched large portions of its backend from Ruby to Scala

→ App Inc, Walmart

→ Singleton Object
- Scala has no static variables or methods.
- It uses singleton object which is essentially class with only 1 object in source file.

→ Immutability using val
- This helps to manage concurrency control.

→ Lazy Computation
• U can declare lazy variable by using lazy keyword.
• only vals can be lazy → lazy val a = 10

→ Case classes & Pattern matching

→ Concurrency control

→ String Interpolation
s, f & raw

→ if, if else,

Scala Object & Class.

Object	Class
→ It has state - data values of an object	→ Main method can't be defined here
→ static like instance of Scala	→

Singleton Object

→ object APP {
def main (args: Array[String]) {
SingletonObject.hello()
}
} // create 1 singleton object with hello method.

Companion Object

Object created to create an instance to a class.

Scala Object & Classes.

Eg: of class

```
class Student {
    var id: Int = 0;
    var name: String = null;
}
object APP {
    def main (args: Array[String]) {
        var s = new Student()
        println (s.id + " " + s.name);
    }
}
```

Eg2:

```
class Student (id: Int,
               name: String)
{ def show() {
    println (id + " " + name)
  }
}
object {
var s = new Student (100, "Nartin")
```

Anonymous Object

```
class Arithmetic {
    def add (a: Int, b: Int) {
    }
new Arithmetic().add (10, 10);
```

---

① Primary Constructor

```
class student (id: Int, name: String) {
    def show() {
    }
```

② Secondary constructor:

```
class student (id: Int, name: String)
{ def show() {
}
```

```
def this(id:Int, name:String,
        age:Int){
    this(id,name)
    this.age=age
  }
}

object App{
  def main(args: Array[String]){
  var s= new Student(101, "Rama",
                   20);
}
```

Constructor Overloading.

```
class Student(id:Int){
  def this(id:Int, name:String)={
    this(id)
    println(id+" "+name)
  }
  println(id)
}
object App {
  def main(args: Array[String]){
    new Student(101)
    new Student(100, "India")
  }
}
```

wont work

instead all this methods

def this(_,_) = this(_,_)}

↑ No paranthesis

Method Overloading

→ same as Java

This Keyword in Scala

- this is used to refer current object
- You can call instance variables, methods, constructors by using this Keyword

Inheritance

```
class child extends
              Parent()
{
}
```

Method Overriding

```
class Vehicle{
  def run(){
    println("running..")
  }
}
class Bike extends Vehicle{
  override def run(){
    println("Bike is
           running")
  }
}
```

mail >show
2tasks cmplt
prod
possible ELK

Scala Field Overriding Example

```
class Vehicle{
  Val speed:Int=60
}
class Bike extends Vehicle{
  override val speed : Int=100
  def show(){
      println(speed)
} }
```

Exception Handling.

```
class ExceptionExample {
  def divide(a:Int, b:Int) = {
    a/b
    println("Rest of the code is executing")
  }
}

object MainObject {
  def main(args:Array[String]){
    var e = new ExceptionExample()
    e.divide(100, 0)
  }
}
```

```
try{
  a/b
}
catch{
  case e: ArithmaticException => println(e)
}
```

```
try{
  a/b
  var arr = Array(1,2)
  arr(10)
}
catch {
  case e: ArithmaticException => println(e)
  case ex: Throwable => println("found unknow exception")
}
```

```
finally
try {
  a/b
  var arr = Array(1,2)
  arr(10)
}
catch {
  case e: ArithmaticException => println(e)
  case ex: Exception => println(ex)
  case th: Throwable => println("found a unknown exception"+th)
}
```

Data Defn Language

```
val df = spark.read.format("json")
  .option("inferSchema", true)
  .schema("cxDfSchema_DDL)
  .load("/filestore/...")
```

```
sc.parallelize(seq(
sc.textFile
```

Dataframes.

```
spark.read.format("").option
  .schema(
  .load(
```

```
schema = "`customerId` STRING,
  `orderID` INT, `products`
  Array<STRUCT<`pdctId`:STRING,
    `quantity`: BIGINT,
    `soldprice`: Double
```

→ STRING INT BIGINT DOUBLE

`CxId` STRING, `OrderId` BIGINT
`products` DOUBLE

"`CxId` STRING

`cxId` STRING, `OrderId` BIGINT
`products` DOUBLE
`Cxid` STRING

Scala Singleton & Companion Object

Objects
Classes
Constructor
OOPS

Scala singleton

→ declared by using object keyword.
→ There is no static concept in scala, so scala creates a singleton object to point for your program execution.
→ Methods created inside a singleton object are accessible globally.
→ A singleton object can extend classes & traits.

Companion object

→ when you have an object with same name as companion class.

Primary, Secondary constructor & Constructor Overloading.

→ Default Primary Constructor

```
class student {
    println("Hello from default constructor");
}
```

→ Primary constructor

```
class Student (id: Int, name: String) {
    def showDetails() {
        println(id name)
    }
}

main {
    var s = new Student (101, "Rama")
}
```

→ As usual primary constructor gets executed during the object creation.

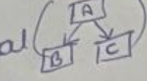
→ '=' is mandatory even if you return or wont return any value in scala def pgm.

```
def main() = {
```

→ mandatory from scala 3

→ During constructor overloading call to primary constructor shld be the 1st stmt
→ Defining the extra param of the second constructor, on top of the class is also mandatory.
→ As above stmt clears, the secondary constructor must have all params of primary along with its additional ones

Inheritance in Class

→ single, Multilevel, Heirarchical () is allowed for classes in scala
→ whereas using Traits Multiple & Hybrid (Diamond UNL) can be achieved.
→ Use 'override' keyword to override a val type variable as below

```
override val speed: Int = 150 , But var defend
```

variables can be changed just like that

Scala → left out topics.

3.2.2v as of January 30 2023.

Features of Scala

① Type Inference
② singleton object
③ Immutability
④ Lazy computation
⑤ Case classes & Pattern matching.
⑥ Concurrency control
⑦ String Interpolation
⑧ Higher order fns
Taking fn as argument & returning function.
⑨ Traits
⑩ Rich set of collections

Concurrency control — Future

English meaning of concurrent →
"to be served at the same time"
"meeting or tending towards one point"

A Future represents a value which may or may not currently be available, but will be available at some point or exception if value not available.

In single threaded world you bind the result of a method call to a variable like this.

→ def aShortRunningTask(): Int = 42
val x = aShortRunningTask()
// Here 42 is immediately bound to x.

→ def aLongRunningTask(): Future[Int] = ???
val x = aLongRunningTask()
// so here aLRTask() takes undefined amt of time to return the value of x which will be available in future

→ def aShortRunningTask(): Int =
Thread.sleep(500)
42
val x = aShortRunningTask()
println("Hello")
// println stmt is printed ~~almost immediately~~ once aShortRunningTask is completed
// If aShortRunningTask is created as Future println is printed almost immediately.

→ Akka is a toolkit for building highly concurrent, distributed, & resilient message-driven applications for Java and Scala.

println(s"before the sleep...")

Future → Handle relative slow computation code on somother thread & call me back with result when you are done

Eg: Running multiple Features & joining their results

Note: Some Future methods
→ filter
→ flatmap
→ map

And these callback methods
→ onComplete
→ andThen
→ Foreach

Transformation methods
→ fallbackTo
→ recover
→ recoverWith

```
import scala.concurrent.Future
import scala.concurrent.ExecutionContext.Implicits.global
import scala.util.{Failure, Success}
val startTime = System.currentTimeMillis()
def delta() = System.currentTimeMillis() - startTime
def sleep(millis: Long) = Thread.sleep(millis)
@main def multipleFutures1 =
    println(s"creating the futures : ${delta()}")
    val f1 = Future{sleep(500); 1}
    val f2 = Future{sleep(200); 2}
    val f3 = Future{sleep(400); 3}
    val result =
        for
            r1 <- f1          //join the futures in for
            r2 <- f2              expression
            r3 <- f3
        yield
        println(s"In the 'yield' : ${delta()}")
            (r1+r2+r3)
    //process the result
    result.onComplete {
        case Success(x) =>
            println(s"in the success case : ${delta()}")
            println(s"result= $x")
        case Failure(e) =>
            e.printStackTrace
    }
```

println (s"before the 'sleep(3000)' : ${deltaT()}ms")
Sleep(3000)

O/P

creating the futures : 1
before the 'sleep(3000)' : 2
in the 'yield' : 806
in the Success case : 806
result = 6

→ As the o/p shows the futures are created rapidly & in 2 seconds the print stmt above sleep(3000) is executed showing the full code is run on the JVM's main thread.

→ Then at 806 ms the 3 futures complt & the code in the yield block is run
→ The code goes to the sucess case in the on complete method.

→ The 806 o/p is a key to seeing that the 3 computations are run in parallel. If run sequentially it would be 1400 ms

Key Points about Futures

→ Futures to run tasks off the main thread
→ The value inside a Future is always an instance of one of the Try types: Success or Failure

b)

Scala Runing Pgm

```
Object ScalaExample{
    def main(args: Array[String]) {
        println "Hello Scala"
}
}
```

Save as ScalaExample.scala

cmd >> scalac ScalaExample.scala
scala ScalaExample

Data Types

→ Boolean Byte short Char Int Long Float Double String.
1bit 1Byte 2B 2B 4B 8B 4B 8B

→ Any

Latest version of Scala.
3.2.2 v as of January 2023

Spark latest version
3.4.0 v was released
April 14 2023

Spark supports Scala 2.12/2.13
Python 3.7+ versions (3.7 is deprecated)

• Why we use = (equal) ...

Python vs Scala

Python → Guvido Van Rossum in 1989 Netherlands

→ Python is a dynamically typed language

Eg → x = 6
// This will store 6 in the memory & binds name x to it. After it runs, type of x will be int
print(type(x))

→ x = hello
// This will store hello at some location int memory & binds the name x to it. After it runs type of x, it will be string.

→ Easy to learn
→ No need to specify objects since dynamic
→ Huge Community
→ Has such libraries ML, NLP
→ Used for small-scale projects
→ doesnt provide scalable feature support

Scala → Martin Odersky in 2002 Switzerland.

→ Statically type language.

Advantages of statically type
→ Dynamically types language is fast, But these functions dont give any info about what arguments to pass u will end up passing junk arguments

JS is dynamic so TS is added for static typing.

→ Difficult to learn, but 10 times faster than Python
→ Need objects.
→ Okay community.
→ No such tools.
→ large-scale projects.
→ provide scalable feature support.

based on ~~the~~ above Hadoop ... select winners

• Why we use = (equal) operator in scala functions

Functions

Pattern Matching in scala

→ ① In java we have switch case

```
String day = "Monday"
switch (day) {
  case "Monday":
      stmts
      break;
  case  :
      stmts
      break
  default :
}
```

- Scala

```
var a = 1
a match {
  case 1 ⇒ println("one")
  case 2 ⇒
  case _ ⇒
}
```

→ In scala there is no break statement can be done as below using breakable method.

```
import scala.util.control.Breaks._
object MainObj {
  def main(args: Array[String]) {
    breakable {
      =
      break
    else
      =
  }
```

→ Case classes in Scala

Functions in Scala

- Why we use = (equal) operator in Scala functions?
  - → We can create fn with or without equal operator
  - → If used returns values
  - → If not used it will return anything & acts as subroutine.

```
def fn(a:Int):Int = {
    var b = a+a;
    b
}
```

Note: From Scala 3 = operator on all the functions is mandatory even if it returns anything or not

- Function parameter default value in Scala

```
def sum(a:Int=0, b:Int=0):Int = {
}
//so here 0,0 is the default values if not passed it considers the value 0.
```

- Function named parameter

```
var a = sum(a=15, b=10)   //while passing params u can pass using the already defined variable names.
```

- Passing function as parameter or Higher Order Function in Scala.

```
def multiplyBy(a:Int):Int = {
    a*2
}
def main(args: Array[String]) {
    output(25, multiplyBy)
}
def output(a:Int, f: Int => AnyVal): Unit = {
    println(f(a))
}
```

- Function composition

```
var result = multiplyBy(addBy(10))   //function composition
```

- Anonymous Lambda function → can be created by using ⇒ or _

```
var result1 = (a:Int, b:Int) ⇒ a+b   //Anonymous function by using ⇒
var result2 = (_:Int)+(_:Int)   //Anonymous function by using _ wildcard
println(result1(10,10))
println(result2(10,10))
```

based on the above hadoop ... select winners

→ [illegible] classes in Scala Imp:

- Multiline expression in Scala

```
def add(a:Int, b:Int) = {
    a
} +b
```

- Scala Function currying → add(a:Int)(b:Int), add(10)(10),

```
var c = add(10)_
var r = c(3)
```

```
object MainObject{
    def add(a:Int)(b:Int) = {
        a+b
    }
    def main(args: Array[String]) = {
        var result = add(10)(10)
        println("o/p is" + result)
        var addIt = add(10)_
        var result2 = addIt(3)
        println("o/p is " + result2)
    }
```

- Nested function in Scala.

```
def add(a:Int, b:Int, c:Int) = {
    def add2(x:Int, y:Int) = {
        x+y
    }
    add2(a, add2(b,c))
}
def main(args: Array[String]) = {
    var result = add(10,10,10)
}
```

→ Case classes in Scala IMP

- regular classes which are immutable by default & decomposable through pattern matching.
- It uses equal method to compare instances structurally.
- It does not use new keyword to instantiate the object.

→ case class className(params)

Eg1:

```
case class Airport(a:Int, b:Int)
object MainObject {
  def main(args: Array[String]){
    var airp = Airport(10,10)
    print(airp.a + " " + airp.b)
  }
}
```

Eg2: IMP

```
trait Flights
case class Boeing(a:Int, b:Int) extends Flights
case class Mig(a:Int) extends Flights
case object F16 extends Flights

object MainObject {
  def main(args: Array[String]){
    callCase(Boeing(10,10))
    callCase(Mig(10))
    callCase(F16)
  }
  def callCase(f: SuperTrait) = f match {
    case Boeing(a,b) => println("a=" + a + "b=" + b)
    case Mig(a) => println(a)
    case F16 => println("No Args")
  }
```

based on

→ Scala file handling

- Handling file operations
- Predefined methods to deal with the file
- Create, open, write & read the file
- Scala has a complete package for file Handling
  Scala.io

```
import scala.io.Source
  object MainObj{
    def main(args: Array[String]){
      val filename = "ScalaFile.txt"
      val fileSource = Source.fromFile(filename)
      for (line <- fileSource.getLines){
          println(line)
      }
    fileSource.close()
  }
}
```

→ Multidimensional Array

var arr = Array.ofDim [ArrayType] (R, C) → Row, column

var arr = Array (Array (1,2,3), Array (6,7,8,9))

→ String interpolation

s, f, raw & use $ inside string sentence to show the value of variable

%.2.2f

raw → prints as it is ignore \n \t

eg var s = "ABC\t DEF \t GHI" o/p with tab seperation

var s = raw(" ,, ,, ") o/p as it is.

→ Exceptions in Scala (throw, throws TA Team

• Scala makes "checked vs unchecked" very simple. It doesn't have checked exceptions. All exceptions are unchecked in Scala.

Scala pgm without exception handling.

```
class Example {
    def divide (a:Int, b:Int) = {
        a/b
        println ("Rest of the code is executing...")
    }
}
object MainObject {
    def main (args: Array [String]) {
        var e = new Example()
        e.divide (100,0)
    }
}
```

→ wont get exec

o/p

- java.lang. Arithmetic Exception

Exception Propogation

40-50.

Collections

50-60

Companion object

- There is no static concept in scala
- companion class & companion object must be defined in same source file.

singleton object

I can call the methods inside this without creating any object.

based on above Hadoop

→ Anonymous object in Scala

```
class Arithmetic{
    def add(a:Int, b:Int){
        var sum = a+b
    } Print(sum)
}
```

```
object Main{
    def main(args: Array[String]){
        new Arithmetic.add(10, 10)
    }
}
```

→ Method overriding in Scala

Extend class & override the method.

"extends"

```
override def run(){
```

→ final

Used to prevent Inheritance

final class cant be inherited itself

→ Abstract class

- has both abstract & non abstract methods
- using extends keyword other classes can give implementation.
- if all methods are undefined define class as abstract

→ Scala Trait

- all abstract or some abstract & non abstract methods
- Trait Mixins
  - ↳ means u can extend any number of traits with a class or abstract class
  - ↳ u can extend only traits or combo of traits & class or traits & abstract class

→ Access modifiers only 3

	class	Companion	Subclass	Package	world
No modifier/Public	Yes	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No	No
Private	Yes	Yes	No	No	No

Companion is a singleton object named same as the class

→ Array (1,2,3,4,5)

var arr =

```
var arr : Array[Int] = new Array[Int](10)
var arr = new Array[Int](10)
var arr : Array[Int] = new Array(10)
var arr = Array(5,4,3,2,1)
```

based on above Hadoop package select winutils

Final Variable

```
class Vehicle {
  final val speed: Int = 60
}
class Bike extends Vehicle {
  override val speed: Int = 100
  def show() {
    println(speed)
  }
}
// Error value speed cannot override final member.
```

Final Method

```
class Vehicle {
  final def show() {
    println("vehicle is running")
  }
}
class Bike extends Vehicle {
  //override val speed: Int = 100
  override def show() {
    println("bik is running")
  }
}
// Error.
```

Final class.
→ final class can't be inherited.

Abstract class.

→ object of abstract class can't be created.
→ If a class extend abstract class it must provide implementation of its all abstract methods.

Eg:

```
abstract class Bike(a: Int)
{
  var b: Int = 20
  var c: Int = 25
  def run()
    def performance() {
  abstract method
    println("performance awesome")
  }
}
```

Scala Trait

→ same Interface, collection of abstract - non abstract methods.

```
. trait Printable {
    def print()
  }
class App extends Printable {
  def print() {
    println("Hello")
  }
}
```

```
trait Printable {
  def print()
}
abstract class A4 extends Printable {
  def printA4() {
    println("Hello this is A4 sheet")
  }
}
```

```
trait Printable {
  def print()
  def show() {
    println("ABC")
  }
}
```

---

extends	with
trait ABC trait DEF	class GHI extends ABC with DEF

Scala Access Modifier

No Access Modifier	O/side package	Package	class	subclass	Companion
	Yes	Yes	Yes	Yes	Yes
Protected	No	No	Yes	Yes	Yes
Private	No	No	Yes	No	Yes

Scala Array

```
var arrayname : Array[Type]
    = new Array[Type](size)

var arrName = new Array[Type]
    (size) or
```

Eg:

```
var arr = Array(1,2,3,4,5)
var arr = new Array[Int](5)
```

Multidimensional Array

```
var arr1 = Array(Array(1,2,3,4,5),
        Array(6,7,8,9,10))
var arr3 = Array.ofDim[Int](2,5)
```

Apache Spark

Categories of Data

① Structured Data (tabular)
② Semi structured Data (JSON)
③ Unstructure Data
↳ text, audio, video, graphs (logs)

RDMS handles ① & ②

Yr → 2002 → Big data was introduced to handle Unstructured Data

→ **Big Data Technologies**
The software tools used to manage all types of datasets & transform them into business insights.

We can categorize them into
- Data Storage → Hadoop, MongoDB, Cassandra
- Data Mining
- Data Analytics
- Data Visualization

vg61303

Hadoop

For storage uses
HDFS → Hadoop distributed File System
→ Uses commodity Hardware to store data so not so reliable
→ So always keeps 3 copies of data
→ Kerberos, 3rd party tool maintains Data security for HDFS

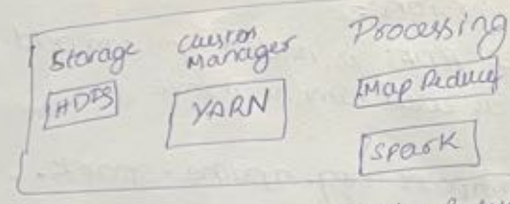
For Processing Map Reduce Framework
→ It can process all kind of data.
→ It's a 2 step process
① Map phase
② Reduce phase
Narrow Dependency → Map phase
Wide Dependency → Reduce phase

How Map Reduce F/w works

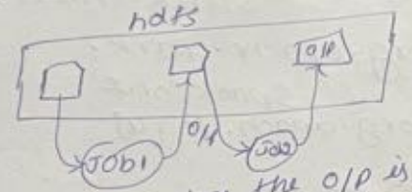
eg: Word count

Input | splitting | Mapping | shuffling | Reducing | Final result

In 2007 → YARN (Yet another resource negotiator)



Main Disadvantage of MapReduce



→ Every time the O/P is stored onto HDFS & retrieved for the next job

→ But spark jobs are executed in DAG format

Map Reduce	Spark
• Supports only Java	• Supports so many other languages Python, SQL, R, Java scala
• Slow	• Fast

Java Scala Spark Setup

① Download JDK 8
JAVA_HOME /jdk1.8
Java Path /jdk1.8/bin
check Java -version on cmd prompt

② Download Spark ← Search.
select Spark release, Package type
//Pre-built Apache Hadoop 2.7

If older versions needed go to Archived versions.

③ Download winutils from any github based on above Hadoop package ① select winutils

spark

RDD programming.

→ To write a Spark application u need to add Maven dependency.

groupId =
artifactId =
version =

→ If HDFS is needed you can add it as well same as above.

① import org.apache.spark._
import org.apache.spark.
SparkContext
import org.apache.spark.
SparkConf
import org.apach.log4j

②
Object Transformationmethods{ (any name)
def main(args: Array[String]){
Logger.getLogger("org").setLevel(Level.OFF)

val sc = new SparkContext("local[*]",
"Helloworld")

③ Taking the data in RDD

val distData = sc.parallelize
(List("alpha", "beta", "gama", "xrays"))
or Seq

val data = sc.textFile("C:\\Users
\\Pramod .txt" or .csv")

→ sc.parallelize(data, 20) ← partitions
sc.textfile("path", 5) ← partitions

→ Spark tries to set the number of partitions automatically based on ~~your~~ no of cores (executors) in your laptop [local[*]]
manually you can set as above.

→ Partitioning is a way to split the data into multiple partitions so that you can execute transformations on multiple partitions in parallel.

RDD fns

val a = sc.parallelize(List( ))
.map(i => (i, 1))

distData.first | .take(2)
.top(4) | .take(3)
.count() | .takeOrdered(4)

a.countByKey()
distData.countByValue()

1 way of creating sparkcontext
another way.

val conf = new SparkConf().setAppName
("wordcnt")
val sc = new SparkContext(conf)

→ val b = sc.parallelize(Seq
((1,2),(3,4),(1,5),(1,6)))
.collectAsMap()

o/p → (1,6) (3,4)

gives o/p of distinct key value pairs overwriting with the latest o/p i.e 1,6

④ C:// → bigdata
↳ spark (Extracted only) not tar
↳ winutils.

⑤ Set 3 Env variables (o
→ SPARK_HOME
C/bigdata/spark
→ HADOOP_HOME ← winutils path
C/bigdata/spark.
↳ Try putting winutils inside spark if this wont work

⑥ → Inside Path variable
e/bigdata/spark/bin

>> spark-shell

scala >

· Spark-Scala
· JS, Nodejs, Reactjs

---

18 HighLevel Syntax in RDD

① rdd = sc.parallelize(Seq( ), no of partitions)
Array( )
List( )
= sc.textFile(path, no of partitions)

② rdd = rdd.map(i ⇒ i, 1)
(i ⇒ (i, i.length))
(i ⇒ (i.split("\\W+")))

③ rdd = rdd.flatMap(i ⇒ (i.split("\n")))

④ rdd = rdd.take(2)
rdd.takeOrdered(4)
rdd.head() top()

⑤ rdd = rdd.countByKey()
= rdd.countByValue()

(18) = rdd.count()

⑥ rdd.sortBy(i ⇒ i._2, true)
.sortByKey(true)

⑦ rdd.filter(i ⇒ i != headerLine)
.filter(i ⇒ .startsWith("")
.equals( )
.contains
.equalsIgnoreCase( )

⑧ rdd.filterByRange(3,10)

⑨ rdd.collect

⑩ rdd.distinct().count

⑪ rdd.reduce((i,j) ⇒
if(i._2 > j._2) i else j)
rdd.reduceByKey( )

⑫ print(rdd.partitions.length)
rdd.repartition(10)
rdd.coalesce(3)

⑬ rdd.cache( )

⑭ sc.broadcast(rdd)

⑮ val df = spark.createDataFrame(rdd)
= spark.read.json(path)
df.printSchema()
df.show()
df.createOrReplaceTempView("student_table")

⑯ spark.sql("select * from student_table")

⑰ rdd.saveAsTextFile("path")

⑱ set Fns
rdd1.union(rdd2)
rdd1.intersection(rdd2)
rdd1.subtract(rdd2)

```
val rdd2 = rdd1.map(i =>
    (i, (scala.math.pow(i, 0.5))))

val names = sc.parallelize (List
("spark", "Hadoop", "Mapreduce"))
  .map(i=>(i, i.length))
  .sortBy(e=>e._2, true).collect

// .sortByKey().collect
//.sortBy Key(true).collect
//.sortBy(i=>i._2, false).collect
//.sortBy(i=>i._2, true).collect
//.filter(_._1.startsWith("s"))
    .collect
```

Filter By Range.

```
→ sc.parallelize(List(" ", " "
  " "))
  .map(i=>(i.length, i))
  .filterByRange(3, 10)
```

Filter, equals contains.

```
.filter(_.equals("spark"))
    .collect

.map(i=>(i, i.length))
    .filter(_._2>3)

.filter(i=>i.contains
    ("spark")).collect
```

→ Broadcast Cache.

Broadcast

```
val words = input.filter
    .map .filter ..
val a = sc.broadcast(words)
```

- Broadcast is used to supply a copy of small sized dataset to each & every executor.
- So whenever the executors need this data set they dont need to read them from the disk.

```
cache()
val words = input.filter.map
    .cache()
```

- Spark uses lazy evaluation for computation to enhance the performance.
- A lineage is maintained, that is the composition of steps to build the RDD
- When we perform any action on an RDD all the steps in the lineage will be processed.
- Spark by default will reprocess these steps everytime we call an action. which is in efficient.
- In order to avoid this we call cache() on an RDD which will store the processed data.

filter

```
al header = input.first()
al brandCount = input.
  .filter(i => i != (headerLine))

→ count of sedan cars.

al words = input.filter(i => i != (
(headerLine)).map(x => ((x.split
("\\w+"))(4)))).filter
(_.equalsIgnoreCase("sedan"))
  .count()

UNION
NTERSECTION
UBTRACT
IP WITH Index

→ myrdd.partitions.length
      //gives the set partition value

→ val rdd1 = sc.parallelize(List( ))
  rdd2 =

→ val y = rdd1.union(rdd2)
          .collect

→  =rdd1.intersection(rdd2)

→  =rdd1.subtract(rdd2);

→
.groupByKey()
```

Repartition & coalesce

```
val n = sc.parallelize(
          1 to 100, 5)
val n1 = n.repartition(10)
val n2 = n.coalesce(3)
```

Repartition
Increase or decrease the number of partitions, but it involves heavy data shuffling across the cluster.

coalesce
Used to decrease the partition does not involve shuffle.

Saving File TO Disk in RDD

```
System.setProperty
("hadoop.home.dir",
    "C:\\winutil\\")

val a = sc.parallelize
  (1 to 10000, 3)
a.saveAsTextFile("path")
```

S →

val input = sc.textFile (path)

val coords = input.flatMap(
x ⇒ x.split ("\\W+"))

map	flatMap
O/P is Array if used on top of split fn	O/P is converted from Array to row

① map

df. map (i ⇒ i.split(" "))

O/P

[Project, Gutenberg]

[Alice's, Adventures, in, Wonderland]

② flatMap

df flatMap (i ⇒ i.split(" "))

O/P

Project
Gutenberg
Alice's
Adventures
In
Wonderland

map	flatMap
→ only performs mapping	→ Performs mapping as well as flattening.

• SortByKey()
• countByValue()
• reduceByKey
•

→
• map (x ⇒ x.split("\\W+")(2)
.toDouble)

→ • reduce ((x,y) ⇒ if (x._2
> y._2) x else y)

→ input.map (i ⇒ {
val j = i.split(",")(j(0), j(4))
y) • reduce ((x,y) ⇒ if
(x._2 < y._2) x else y)

→ input.flatMap (x ⇒ x.split
("\n")). map (x ⇒ ((x.split
(",")(0)))).distinct().count

→ input.flatMap (x ⇒ x.split
("\n")).map (x ⇒ ((x.split
(",")(0)), (x.split("\\W+")
(1).toDouble)))
• sortByKey()
• reduceByKey(_+_)

Spark Session

```
Object JSON{
    def main(args: Array[String]){
    Logger.getLogger("org")
        .setLevel(Level.ERROR)

val spark = SparkSession
        .builder
        .appName("SparkSQL")
        .master("local[*]")
        .config("spark.sql.
                warehouse.dir",
        "file:///C:/temp")
        .getOrCreate()

val data = spark.read.json(path
                           "json")
data.printSchema
data.show()
data.createOrReplaceTempView("ppl")
spark.sql("select * from people").show
```

Big Data Tools.

Apache Hive

→ It's a query Engine. → Abstraction of MapReduce
→ It runs on Spark as well

SQL → Hive
MR | Spark
HDFS

→ Hive always stores metadata in RDBMS (DB2, Oracle, mySQL), actual data is stored in HDFS
→ by default has an embedded RDBMS called Derby.

CLI → Hive
JDBC → Hive
connected from CLI & JDBC

Eg: Read Hive Tables from Spark & perform T/R

(A)
① Install Hadoop
② Install Hive
③ Install Spark

(B)
→ set all variables in bashrc file
→ start bashrc $ source .bashrc
→ start hadoop
→ check all 5 daemons are started using "jps" cmd $jps
→ Go to Apache Hive directory & start Hive bin/hive

(C)
hive > show tables;
hive > select * from test;
hive >

(D)
→ start spark from spark folder as usual
→ Go to Hive folder → conf → hive-site.xml
→ copy paste xml in spark → conf folder
→ Go to jar of Spark find mysql connector.jar
if not there download & paste in spark jar file.

(E)
Now in spark session
scala > val a = spark.sql("select * from default.test")
a.show()

② HBase

→ HBase → HBase is a database management system
HDFS HBase runs on top of HDFS
It can store huge amt of data in tabular format for extremely fast read & writes
Used mostly where frequent inserting is required.

→ HBase is a type of NoSQL DB

Window

```
val windowSpec = Window.partitionBy("ELId").orderBy(desc("InpDate"))
df = df.withColumn("SLNO", row_number.over(windowSpec))
```

row_number → takes the serial numbers of a table or df.

partitionBy → If multiple ELId are there, they will be kept as it is but in the next consecutive row together,

Eg:

ELId	After partitionBy
1	1
2	2
3	10
4	10
10	2
9	3
10	4
9	9
	9

Cache & Persist

→ optimization techniques for iterative & interactive Spark applications to improve performance.

→ cache & persist store the intermediate computation of an RDD Dataframe & Dataset

→ Cost efficient, Time efficient & Execution time reduces.

Cache

→ Spark Dataframe & Dataset caching default saves it to storage level 'MEMORY_AND_DISK'

→ RDD.cache() → 'MEMORY_ONLY'

Persist

→ It can take storage level Argument

→ default 'MEMORY_AND_DISK'

→ df.persist(storageLevel.MEMORY_ONLY)

MEMORY_ONLY, MEMORY_AND_DISK, MEMORY_ONLY_SER, MEMORY_AND_DISK_SER, DISK_ONLY, MEMORY_ONLY_2, MEMORY_AND_DISK_2

→ import org.apache.spark.storage.StorageLevel

Unpersist

remove the storage

Storage Level	Space Used	CPU Time	In Memory Y or N	On Disk Y or N	Serialized Y or N	Recompute some partitions Y or N

All these parameters gets offered based on the storage level mentioned.

→ Create a quick dataframe.

Val schema =

→ Partitions & Coalesce

→ Val df = spark.range(0, 20)

println (~~df.partitions.size~~ df.rdd.partitions.size)
~~df.partitions.length~~ df.rdd.partitions.length

→ df.rdd.getNumPartitions. O/P → 8

→ df.repartition(9) or 5

→ df.coalesce(7) or 5

df.repartition(10, col("id"))

↳ repartition based on col("id")
limit it to 10 only.

→ coalesce doesn't involve shuffle & can only be used to decrease the number of partitions.

To check shuffle & display of partitioned dataset(~~length~~)

df.write.mode(SaveMode.Overwrite).csv("partition.csv")

read & show.

Comparision Operators.

→ •equalTo (78987)
•contains ("POSTAGE")
•where (col("InvoiceNO") === 536365)
•where ("InvoiceNO = 536365")
•where ("Invoice <> 536365").show(5) → other than 536365 it gives its above & below values.

col("UnitPrice") >
>=
<=

<=> → including that value above & below 5 values.

col(" ").isin("DOT").where(filter1.or(filter2))

•withColumn("ABC", expr("NOT UnitPrice <= 250")) •and •not

→ gea, leq
•eqNullSafe("hello")
•geq(250)

52 - 55 cmd (4) left

Spark Streaming

→ val streamingDataFrame = spark.readStream
- .schema(schema_CC)
- .option("maxFilesPerTrigger", 1)
- .format("csv")
- .option("header", "true")
- .load("/path")

//To check if DF is still streaming.

→ streamingDataFrame.isStreaming.
O/P True or False

→ val df.writeStream.format("memory") //memory = store in-memory table
- .queryName("customer_purchases") // the name of the in-memory table
- .outputMode("complete") //complete = all the counts should be in the table

Community Cloud Databricks

Notepad Shortcuts.

① %fs ls /filestore/tables/promodDemo/

→ lists all the dataset files under this path with size.

Spark with Cloud.

Reading from amazon S3 bucket

```
var df = spark.read.json("s3://apache_spark/data
                          /consumer.json")
```

Spark RDD

Resilient Distributed Datasets.

- Fundamental data structure of spark
- RDD is an immutable distributed collection of elements of your data, partitioned across nodes in your cluster, that can be operated in parallel with a low-level API that offers transformations & actions.

Spark ML

Pg 46-51 Spark definitive guide

cmd 14

→ import org.apache.spark.ml.feature.StringIndexer

Apache Spark Interview qstns.

① How is ApSp different from Map Reduce.

Apache Spark	Map Reduce
ⓐ Spark stores data in memory	ⓐ Map reduce data stored in HDFS takes longer time to retrieve.
ⓑ 100 times faster than Map Reduce	ⓑ Map Reduce is slower for large scale processing.
ⓒ provides caching & different level of storage using persist	ⓒ MR is highly disk dependent
ⓓ spark processes data in batches as well as in real time.	ⓓ Procss data in batches only.

② What are the important components of Spark ecosystem?

Language support → Java, Python, Scala, R

Spark Core → Spark SQL, Spark Streaming, Spark MLib, GraphX → Core components

Standalone Cluster, Apache Mesos, YARN → Cluster Management

③ Explain how Spark runs applications with the help of its architecture.

④ What are different cluster managers available in Apache Spark

Apache Spark Standalone Mode, Mesos, Hadoop YARN, Kubernetes.

⑤ what is lazy evaluation in Spark

Transformations in Spark are not evaluated until you perform an action which aids in optimizing the overall data processing workflow. This is called lazy evaluation.

⑥ What makes Spark good at low latency workloads like graph processing & machine learning?

→ Apache Spark stores data in-memory for faster processing & model building.

→ Machine learning algorithms require multiple iterations to create an optimal model ⇒ Graph algorithm traverse all the nodes & Graphs.

⑦ Automatic cleanups for cleaning accumulated metadata

→ spark.cleaner.ttlx

⑧ How can you connect Spark to Apache Mesos.

- Mesos is a open source cluster manager

Step1 : Install Mesos https://mesos.apache.org/

Step2 : Start Mesos Master on the cluster

./bin/mesos-master.sh --cmd.

Start Mesos Agents on all the worker nodes

./bin/mesos-agent.sh --- cmd.

Step3: Make sure Mesos is up & running with all worker Nodes configured

http://[MasterIP]@5050.

Step4: Spark binary package accessible to Mesos.

Install spark in the same location as that of Mesos.

Step5: Configure the

spark.mesos.executor.home.

pointing to spark location.

⑨ what is Parquet file & what are its advantages. what are different file formats. (parquet, avro...

→ Parquet (parke) is a columnar format that is supported by several data processing systems.

→ If you have a columned file & most of the columns are not in use we can go for parquet type of storage.

→ Spark performs both read & write operations with parquet file format

Advantages of Parke

① Able to fetch specific columns for access
② Consumes less space
③ Follows type specific encoding.
④ Limited IO operations.

csv → comma seperated values
json → javascript object Notation
ORC → Optimized Row columnar
↳ best for HIVE
avro → they include markers that can be used to split large datasets into subsets suitable for Apache MR processing.

⑩ what is shuffling in Spark? when do it occur?

→ Unless partition is set manually, Sparks sets partition based on the size of the Dataset or on available cores

→ Shuffling is the process of redistributing data across partitions that may lead to data movement across executors

→ It occurs while joining 2 tables or while performing Bykey operations

⑪ use of coalesce in Spark

Partion A: 11, 12
Partion B: 20, 40, 50
Partition C: 6, 7
Partition D: 9, 10

Remove all nos which are multiple of 10

Partition A: 11, 12
Partition B: –
Partition C: 6, 7
Partition D: 9

Coalesce

Partition A: 11, 12
Partition C: 6, 7, 9

12) How can you calculate the executor memory?

Eg: Consider Nodes = 10

Each Node has = 16 cores −1 for OS

Each Node RAM = 61GB Ram −1 for OS.

No of executors = No of cores / concurrent tasks

(multiple tasks that can be run in parallel) always considered 5

= 15/5

= 3

Total no of Executors = No of Nodes * no of executors in each node

= 10 * 3

= 30

13) What are the various functionalities supported by Spark Core

→ Spark core is the engine for parallel & distributed processing of large datasets.

ⓐ Scheduling & monitoring jobs.
ⓑ Memory management
ⓒ Fault recovery.
ⓓ Task dispatching.

14) What is the significance of RDD

→ Building blocks of Spark
→ immutable, fault-tolerant distributed collection of objects that can be operated in parallel.
→ Whenever you do transformation on RDD is getting created.

15) How do you convert RDD into Dataframe.

→ .toDf

def createDataFrame(RDD, schema: StructType)

23) Does ...

→ Yes Apache Spark provides an ...
→ checkpointing is the process of making streaming applications resilient to failures.

16) Transformations

→ Transformations are operations that are performed on an RDD to create a new RDD containing the results

Eg: map, filter, join, union

Actions.

→ Actions are operations that return a value after running a computations on an RDD

Eg reduce, first, count

18) What is a lineage graph

→ Spark does not support data replication in the memory so if any data is lost it can be rebuilt using RDD lineage

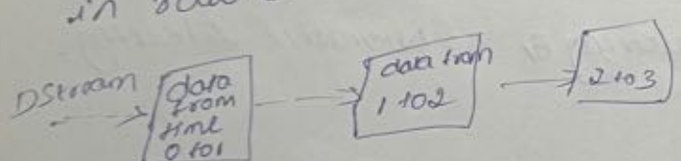
It is also called an RDD operator graph

→ DAG is a lineage graph.

19) What do you understand by DStreams in Spark?

→ Discretized Streams is the basic abstraction provided by Spark Streaming

→ It represents a continuous stream of data that is either in raw or rdd form.



21) What is the need of broadcast variables in spark?

→ Reduces shuffling

→ Increases speed

Since a chunk of data stored variable is stored in all the workers.

22) filter(func) returns a new DStream by selecting only those records

23) Does Apache Spark provide check points

→ Yes Apache Spark provides an API for adding & managing checkpoints

→ checkpointing is the process of making streaming applications resilient to failures.

25) Datatypes supported by MLlib Spark.

24) what do you mean by sliding window operation

→ controlling the transmission of data packets between multiple computer networks is done by sliding window

→ Spark Streaming library provides windowed computations where the transformation on RDDs are applied over a sliding window of data

26) Difference b/w map() and flatmap()

map()
→ Transformation applies a function to each row in a Dataframe/Dataset & returns new transformed Dataset
→ o/p has same no of rows as i/p

flatmap()
→ applies to every element
o/p will have more rows

27) what are the different MLib tools available in Spark?

ML Algorithms
→ classification, regression, clustering & collaborative filtering.

Featurization
→ feature extraction, transformation, dimensionality reduction & selection

Pipelines
→ tools for constructing evaluating & tuning ML pipelines.

Persistance
→ saving & loading algorithms models & pipelines

Utilities
→ linear algebra, statistics & data handling

28) Datatypes supported by MLLib Spark.

① Local vector
② Labeled point
③ Local matrix
④ Distributed matrix

29) What is a Sparse Vector in Spark MLib

Sparse vector is a type of local vector which is represented by an index array & a value array.

```
sparse1 = SparseVector(4, [1,3], [3.0, 4.0])
```

where

- 4 is the size of the vector
- [1,3] are the ordered indices of the vector
- [3,4] are the values

Understand
→ Spark GraphX
→ Read Spark MLib.