**Spark**

**Hive**

**1. External vs internal table**

**2. Load data into hive table**

**3. Hcatalog**

**4. Static vs dynamic partition**

**5. Serde**

**6. Partition**

**7. Merge: union**

**8. Save data file to hive table**

**9. Hive udf**

**10. Explode function**

**11. Index on hive table**

**12. Remove duplicate from hive table without using duplicate**

**13. File ingested using hive(engine), file format, compression technique, rc orc**

**14. Vectorization in hive, error handling**

**15. Orc: takes time to decompress data, it stores data stripe seperated and can be used to access data efficiently. Data compressed to 75%. It compress quickly but with larger file as compared to other file compression format**

**Avro: serialized data stored, schema stored along data**

**Parquet: rows of data stored adjacent to one another, data partitioned horizontally and vertically, can be useful if need to access subset of data as it can access column of data easily without accessing whole record**

**16. Beeline vs hive server**

**17. Hiveserver2**

**18. Orc with gzip is preferred**

**19. Hive mode**

**20. Hive variable**

**21. Hive query processor**

**22. Mapsidejoin and reduce side join from edureka**

**23. Map side join**

**24. Sort by and order by difference**

**25. Cluster by in DML**

**26. Distribute by**

**27. Skewed join**

**28. Update in hive**

**29. Replicated,skew merge join from educaterz YouTube**

**30. Hive.cli.log4j.properties**

**31. Internal and external table**

**32. Dynamic partition**

**33. Which column is chosen for partition**

**34. Bucketing**

**35. Different type of udf in hive**

**36. Sort and order by difference**

**37. Map side join n reduce side join**

**38. Skew join**

**39. Managed and external table**

**40. Map side n reduce side partition**

**41. SCD2 in hive**

**42. Number of bucket set to how many in hive**

**43. Mapside join table size**

**44. Vectorization in hive**

**45. mapper and reducer in hive**

**Python**

**1. Deep keywords in Python**

**2. Dict vs list vs array**

**3. Python vs scala**

**4. Anonymous function in python**

**Java**

**1. Oops concept**

**2. Abstraction and polymorphism**

**3. Overloading and overriding**

**4. Java inheritance**

**5. What is interface in Java**

**Hadoop**

**1. Block size vs input split**

**2. Secondary NN**

**3. Stand by NN**

**4. MR join**

**5. Hadoop from K R Grenier or learning journal**

**6. Input split and block size**

**7. Hdfs architecture**

**8. Hadoop 1 vs 2**

**9. Map reduce programming life cycle**

**10. Combiner**

**11. Architecture of mapreduce**

**12. Replication factor: if we copy first file with rep as 2 and then update rep as 4 then copy new file to hdfs will the previous one also get rep to 4?**

**13. High availability**

**14. Rack awareness**

**15. Yarn architecture**

**Sqoop**

**1. 97 out of 100 table import**

**2. Metastore in sqoop**

**3. How will 10TB data gets load using sqoop**

**4. Import 47 out of 50 table in sqoop**

**5. Different file format in one table via sqoop**

**6. Data compatibility in sqoop**

**7. Driver in sqoop**

**8. Data truncated in sqoop how to find**

**9. Sqoop primary key concept**

**10. No. Of mappers and reducer in sqoop**

**11. Sqoop increamental load kaise hota hai command**

**Oozie**

**1. Coordinator in oozie**

**2. Capture output in oozie**

**3. Fork in oozie**

**Zookeeper**

**1. Zookeeper role**

**AWS**

**1. AWS**

**2. EBS vs S3**

**3. EMR**

**Data warehouse**

**1. Null value, incremental load**

**2. How to insert data if field has comma delimiter**

**Scala**

**1. Closure**

A closure is a function.

Like any other scala function, a closure may be pure or impure.

It can be a named or anonymous function.

For example,

val p = 10

def getHike(salary:Double) = salary \* p/100

Here value of p is not defined in function or in a parameter. So from where the value of p is coming from? And what is p?

Well, p is a free variable for this function. The reason for calling "p" a free variable for this function is that its value is not defined in function.

So from where the compiler resolves the value for p? Well, the compiler tries to resolve the value of p from the closest lexical environment.

So actual definition for closure:

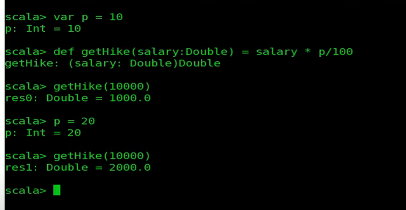
A Function that uses one or more free variables is called as a Closure.

**2. How does the change in the value of free variable impacts the Closure?**

1. **What if I keep changing the value of p, which one is used?**

When you execute a closure, it takes the most recent state of free variable (p)

For example,



1. **Is above closure is pure?**

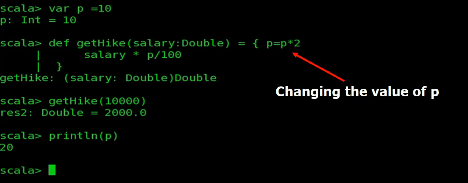
No, because p is defined as **var**, which is not a constant and hence this closure is not pure. If p

Was defined as **val** then we can say that this closure is pure.

1. **What if the closure modifies the free variable?**

The change made inside a closure for free variables are also reflected outside the closure.

For example, changing the value of free variable p from closure also reflects outside the closure.



3. Why Closure? What are the benefits?

In object oriented programming, object will be having two things, functions and data associated with it. But in functional programming language we just have function. There’s no data associated with functions. That’s where closure comes in picture. It gives a function with the state of free variables which can hold the data and it is relatable with object in object oriented programming language.

4. Monad in scala.

5. Recursive tail

6. Traits

7. Tail recursion

8. Case class kaha kaha use hota?

Case class: are immutable and are used for pattern matching

9. Options in class

10. Traits: Concept of OOP which can extend the functionality of a class using set of methods

11. Monad: Object that wraps another object in Scala

**12. Singleton Class and object:**

Class having singleton object, doesn't require instance and is the entry point for your program execution.

**13. Companion object:**

Singleton object having same name as that of class is called companion object and class is called companion class

**15. currying:**

Currying is named after Haskell Curry, an American mathematician. He is known for his work in combinatory logic.

Currying is a means of transforming a function that takes more than one argument into a chain of calls to functions, each of which takes a single argument.

**object** Currying {

**def** main(args: Array[String]) {

// Calling Currying

println(add(10)(10))

**val** addIt = add(10)\_

println(addIt(30))

}

// Currying

**def** add(a: Int)(b: Int) = {

a + b

}

}

One more example.

<https://dzone.com/articles/currying-functions-in-scala-1>

**16. Partially Applied Functions**

The Partially applied functions are the functions which are not applied on all the arguments defined by the stated function i.e, while invoking a function, we can supply some of the arguments and the left arguments are supplied when required. we call a function we can pass less arguments in it and when we pass less arguments it does not throw an exception. these arguments which are not passed to function we use underscore( \_ ) declaration as placeholder.

**def** addHundred(a: Int, b: Int): Int = {

**return** a + b

}

**val** addH = addHundred(100, \_: Int)

println(addH(20))

**17. Anonymous function:**

A function which does not contain a name is known as an anonymous function. An anonymous function provides a lightweight function definition. It is useful when we want to create an inline function.

For example,

// One Way to define an anonymous function

**val** sum = (a: Int, b: Int) => a + b

// Another Way to define an anonymous function

**val** sum2 = (\_: Int) + (\_: Int)

println(sum(20, 30))

println(sum2(1, 2))

**18. Higher order function:**

Higer order function is a function which either takes function as an argument or returns a function.

def math(a:Int, b:Int, f:(Int,Int)=>Int)=f(a,b)

**object** HigherOrderFunction {

**def** main(args: Array[*String*]) {

println("Hello World!!")

println(callMe(50, multiplyByTwo))

}

**def** multiplyByTwo(a: Int) = {

a \* 2

}

**def** callMe(a: Int, f: Int => AnyVal): Unit = {

println(f(a))

}

}

**Misc**

**1. Increase array by 2 places**

**2. Check how many places array shifted**

**3. du -h size of file**

**4. json**

**5. hash function vs round robin**

**6. Employee newly joined between 2017 and 2018**

**7. Even set of employees**

**8. Update fields in dataframe**