

HADOOP FOR BEGINNERS



by TAM SEL

KOTLIN FOR BEGINNERS



Kotlin

by TAM SEL

**KOTLIN
AND
HADOOP
FOR
BEGINNERS**

**LEARN TO CODE FAST
BY
TAM SEL**

[Learn Kotlin](#)

[Getting Started](#)

[Create and Run your First Kotlin Project in IntelliJ IDEA](#)

[Kotlin Hello World Program – First Kotlin Program](#)

[Kotlin Basics](#)

[Kotlin Keywords, Soft Keywords and Identifiers](#)

[Kotlin Variables and Data Types](#)

[Kotlin Type Casting with examples](#)

[Kotlin Operators – Arithmetic, Assignment, Unary, Logical and More](#)

[Kotlin – How to take Input from User](#)

[Kotlin Comments](#)

[Kotlin String](#)

[Kotlin Array](#)

[Kotlin Ranges](#)

[Kotlin Control Flow](#)

[Kotlin When Expression with examples](#)

[Kotlin for Loop with examples](#)

[Kotlin while Loop with examples](#)

[Kotlin do-while Loop with examples](#)

[Kotlin continue Expression with examples](#)

[Kotlin break Statement with examples](#)

[Kotlin Functions](#)

[Kotlin Recursion and Tail Recursion with examples](#)

[Kotlin Default and Named Argument](#)

[Kotlin Lambda Function with example](#)

[Kotlin Higher order function with example](#)

[Kotlin Exception Handling Tutorials](#)

[Kotlin Try Catch with example](#)

[Kotlin Multiple Catch Blocks with example](#)

[Kotlin Nested Try-Catch Block with example](#)

[Kotlin throw keyword with example](#)

[Kotlin Try as an expression in Exception handling](#)

[Kotlin Class and Objects – Object Oriented Programming \(OOP\)](#)

[Kotlin Constructors with examples](#)

[HADOOP](#)

[What is Big Data](#)

[What is Hadoop](#)

[Hadoop Installation](#)

[HADOOP MODULES](#)

[What is HDFS](#)

[HDFS Features and Goals](#)

[What is YARN](#)

[HADOOP MapReduce](#)

[Data Flow In MapReduce](#)

[MapReduce API](#)

[MapReduce Word Count Example](#)

[MapReduce Char Count Example](#)

[HBase](#)

[What is HBase](#)

[HBase Read](#)

[HBase Write](#)

[HBase MemStore](#)

[HBase Installation](#)

[RDBMS vs HBase](#)

[HBase Commands](#)

[HBase Example](#)

[Hive Tutorial](#)

[What is HIVE](#)

[Hive Architecture](#)

[Apache Hive Installation](#)

[HIVE Data Types](#)

[Hive - Create Database](#)

[Hive - Drop Database](#)

[Hive - Create Table](#)

[Hive - Load Data](#)

[Partitioning in Hive](#)

[Dynamic Partitioning](#)

[Hadoop Interview Questions](#)

KOTLIN FOR BEGINNERS

**LEARN TO CODE FAST
BY
TAM SEL**

Learn Kotlin

Kotlin is a statically-typed programming language, developed by JetBrains. If you have basic knowledge of Java, you will be able to learn Kotlin in no time. This **Kotlin tutorial** is designed for beginners so you would be able to understand Kotlin programming even if you have no knowledge of Java.

Kotlin and Java are interoperable which means you can use them together in a Project as well as you can re-write a Java code in Kotlin efficiently. The syntax of Kotlin is concise than Java. In this tutorial you will learn why use Kotlin, what are the advantages of it and several guides on the various topics of Kotlin.

Features of Kotlin

Concise: Kotlin is concise than Java, you would need to write approx 40% less lines of code compared to Java.

Interoperability: Kotlin is highly interoperable with Java. You would not face any difficulty using Kotlin in a Java project.

Open Source: Kotlin is a open source programming language.

Trust: You can trust kotlin as this is developed by popular and well known company JetBrains. JetBrains is known for creating several development tools. The popular Java IDE IntelliJ IDEA is developed by this same company.

Feature-rich: Kotlin provides several advanced features such as Operator overloading, Lambda expressions, string templates etc.

Easy: Kotlin is easy to learn programming language. If you have came from a Java background, you would find it easy to learn Kotlin.

Less error prone: As I mentioned in the beginning, Kotlin is a

statically-typed programming language, which makes you able to catch errors at compile-time as Statically typed programming languages do type checking at compile-time.

Getting Started

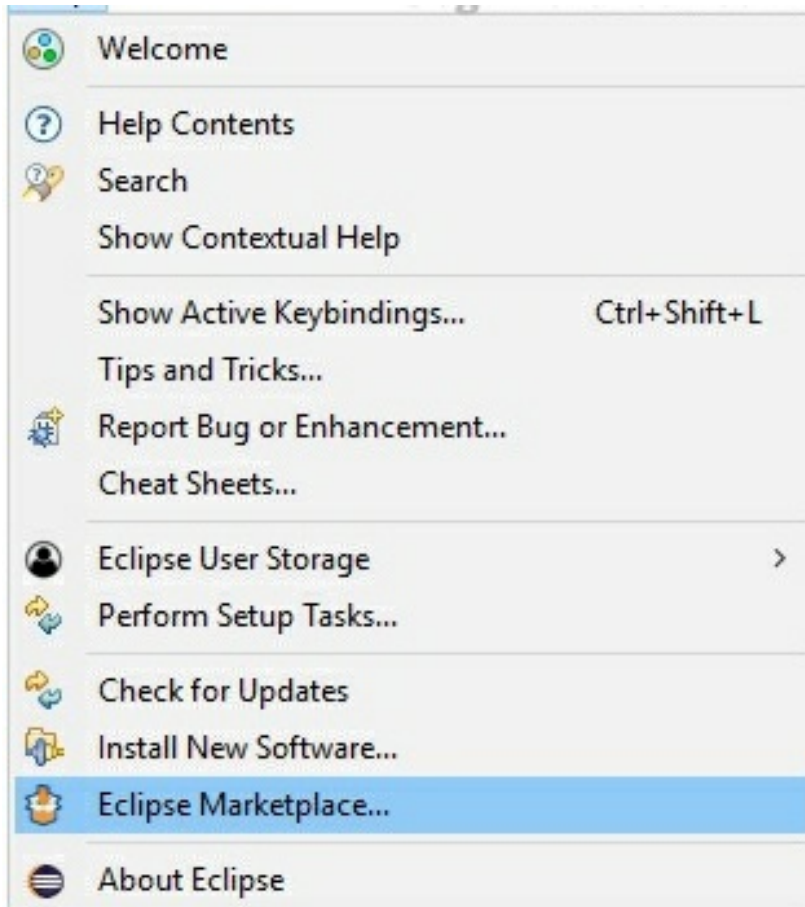
Create and Run your First Kotlin Project in Eclipse IDE

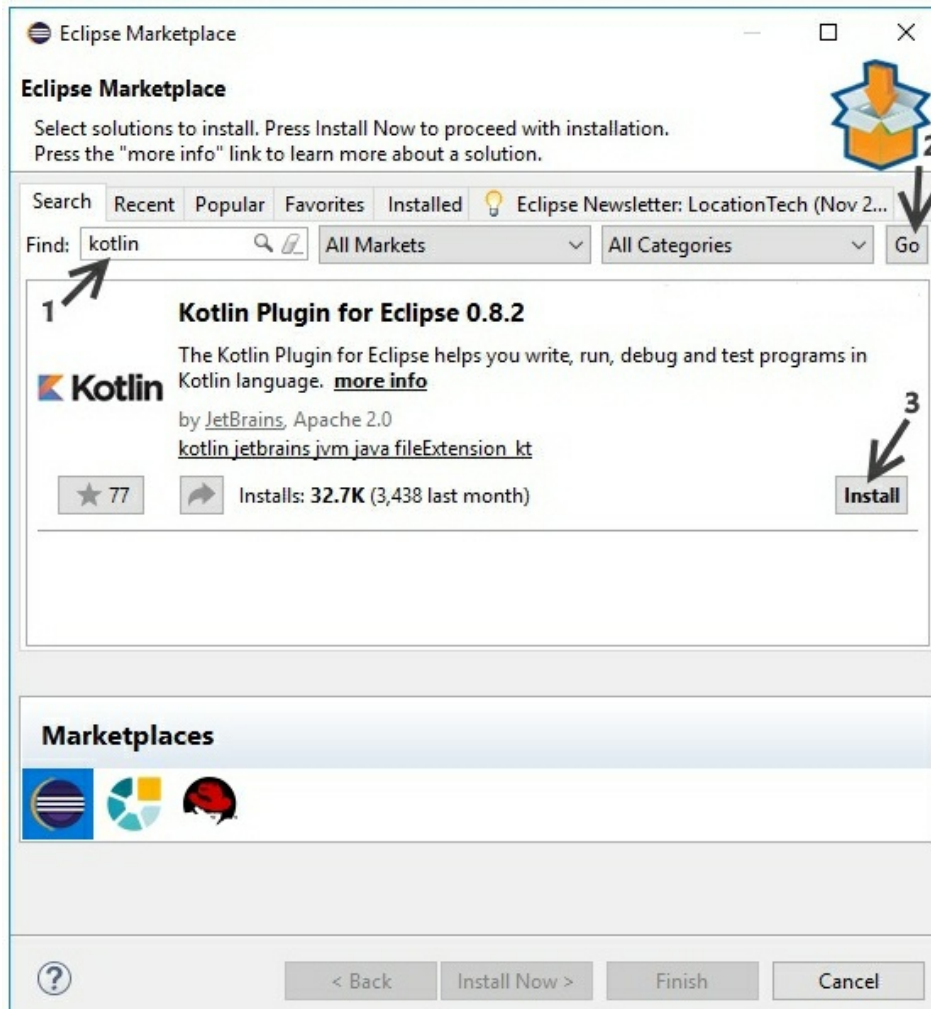
In this tutorial we will see how to install Kotlin plugin in Eclipse IDE to create and run your **first Kotlin application** in **Eclipse IDE**.

Installing Kotlin Plugin

Step 1: If you have not installed Eclipse IDE, you can download it from this link: <https://www.eclipse.org/downloads/> Be sure to grab the “Eclipse IDE for Java Developers” bundle. To run the Kotlin in Eclipse IDE, you must have the Eclipse Neon or later version of the IDE. I m using Elipse Oxygen.

Step 2: To install Kotlin plugin in Eclipse, go to Help section in Eclipse IDE menu and click on “Eclipse Marketplace”. Search for kotlin plugin by typing “Kotlin” in the search field and click go. Click install to install the Kotlin plugin as shown in the screenshot below.





Step 3: Once installation is done, accept the agreement and click “Finish”.

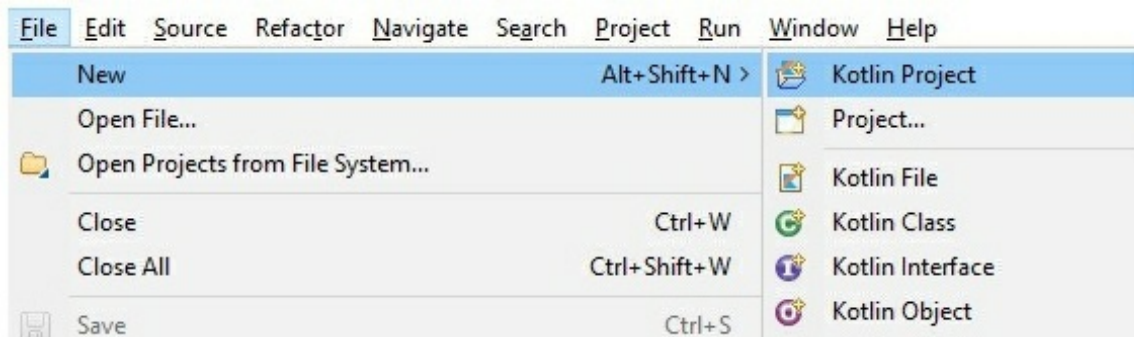
Open Kotlin Perspective in Eclipse IDE

Open the Perspective window in Eclipse IDE by clicking the icon shown in the screenshot below. You can find this icon in the top right corner of Eclipse IDE. Alternatively you can open Perspective window from menu Window -> Open Perspective -> Other. Select Kotlin and click Open.



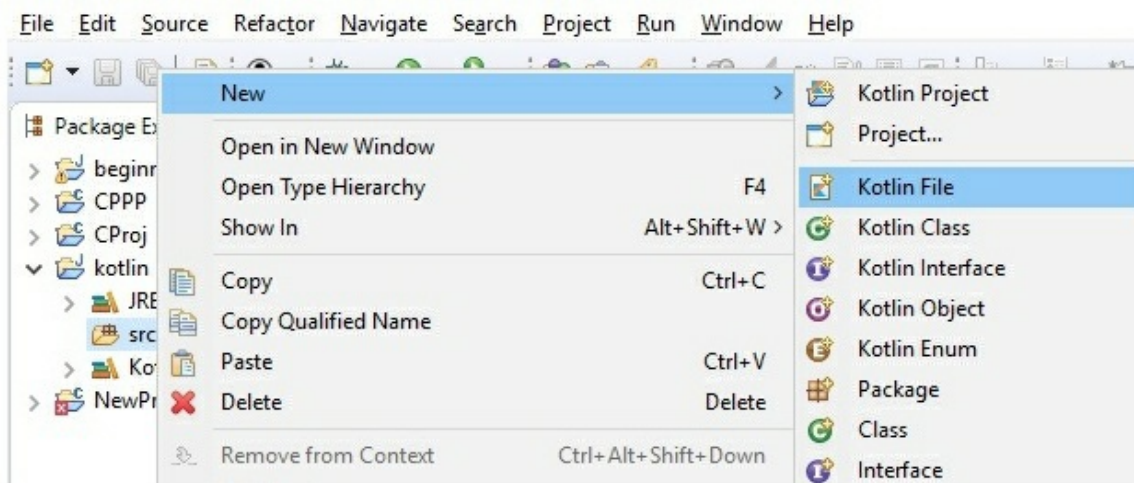
Creating First Kotlin Project in Eclipse

File -> New -> Kotlin Project. Give project name and click finish.



Creating Project File in Kotlin Project

Project->right click on src->New->Kotlin File. Give a meaningful file name and click finish. Kotlin files have .kt extension.



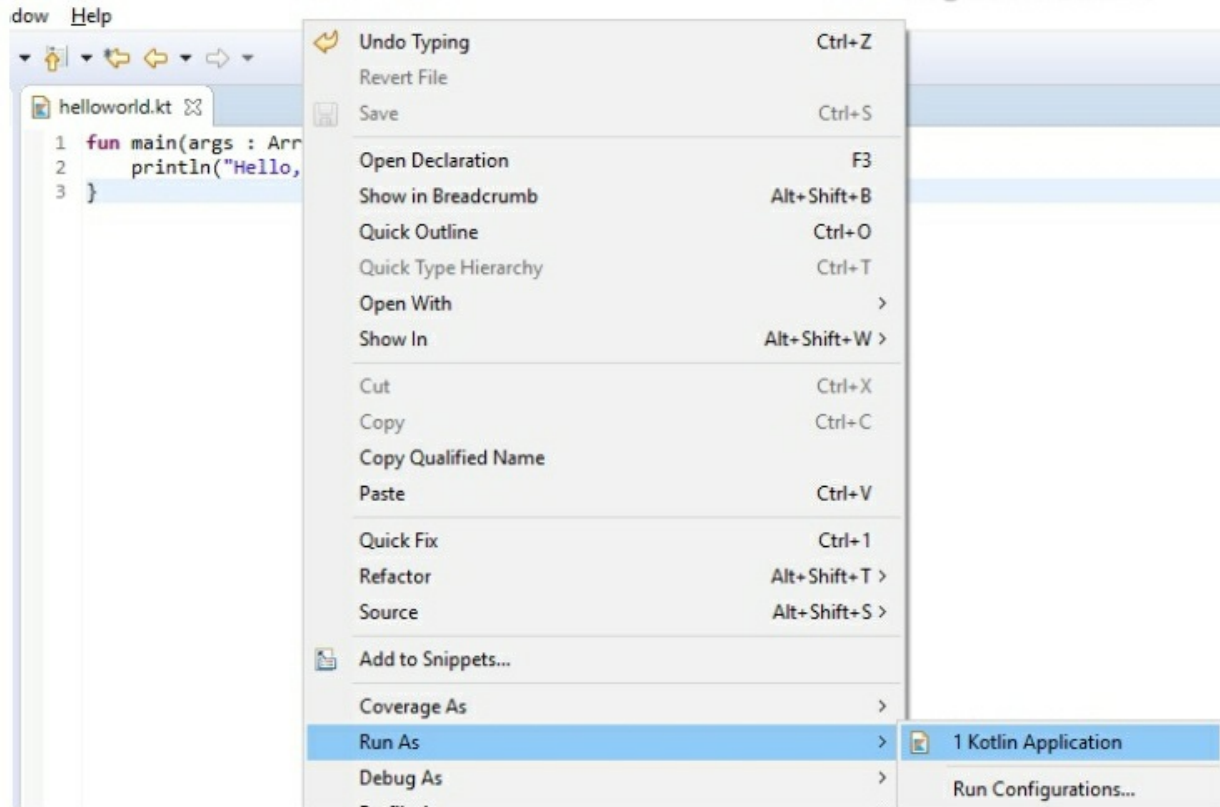
Type this code in the file:

```
fun main(args : Array) {
    println("Hello, World!")
}
```

```
}
```

Running the first Program

Right click on the file ->Run As->Kotlin Application



Output:



Create and Run your First Kotlin Project in IntelliJ IDEA

In the previous tutorial we have seen how to create and run your first Kotlin Project in Eclipse IDE. In this guide, we will see how to install **IntelliJ IDEA** which is an **IDE by JetBrains**, the same company that developed Kotlin programming language. We will also see how to create and run your **first Kotlin project** in IntelliJ IDEA.

IntelliJ IDEA Installation

Go to the download page of official JetBrains website. Choose the Operating System and download the community edition.



The screenshot shows the IntelliJ IDEA download page. On the left is the IntelliJ logo, a stylized 'IJ' inside a black square with a white underline, set against a background of colorful geometric shapes. Below the logo, it lists: Version: 2017.3.1, Build: 173.3942.27, Released: December 12, 2017. There are three links: System requirements, Installation Instructions, and Previous versions. The main heading is 'Download IntelliJ IDEA'. Below it are three tabs: Windows, macOS (selected), and Linux. There are two columns for editions: 'Ultimate' and 'Community'. The 'Ultimate' column describes it as 'For web and enterprise development' and has a blue 'DOWNLOAD' button with 'Free trial' text below it. The 'Community' column describes it as 'For JVM and Android development' and has a black 'DOWNLOAD' button with 'Free, open-source' text below it.

Download IntelliJ IDEA

Windows macOS Linux

Ultimate
For web and enterprise development
DOWNLOAD
Free trial

Community
For JVM and Android development
DOWNLOAD
Free, open-source

Version: 2017.3.1
Build: 173.3942.27
Released: December 12, 2017

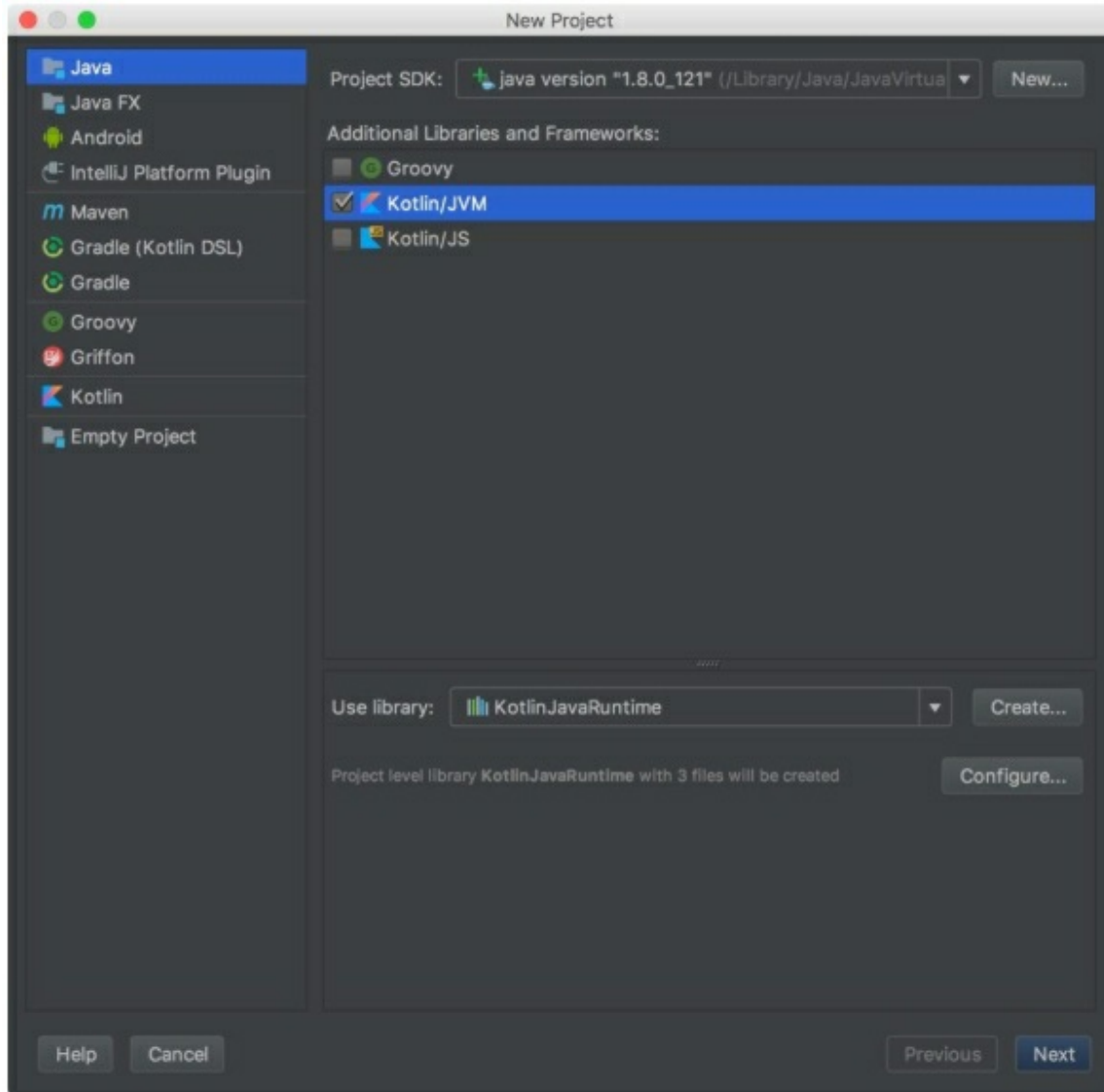
[System requirements](#)
[Installation Instructions](#)
[Previous versions](#)

Creating your First Kotlin Project in IntelliJ IDEA IDE

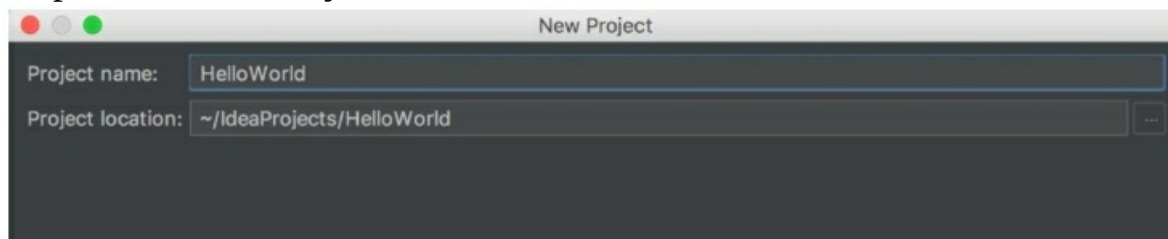
Step 1: Select “Create New Project”.



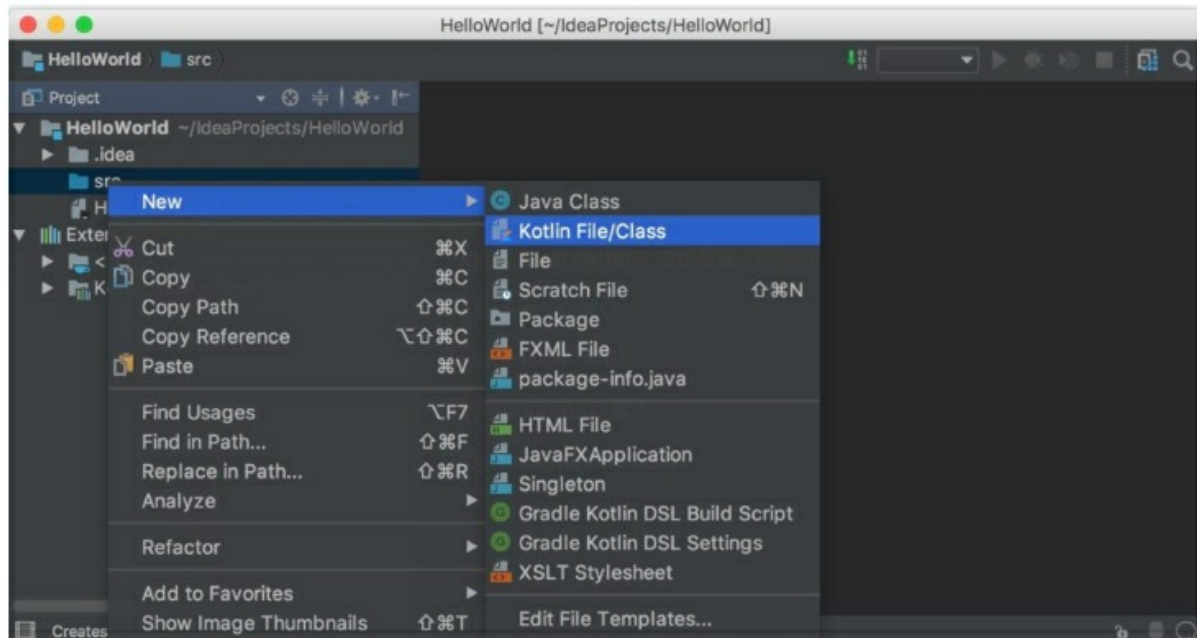
Step 2: In “Additional Libraries and Frameworks” section, select **Kotlin/JVM** and click “Next”.



Step 3: Give the Project Name and click “Finish”.

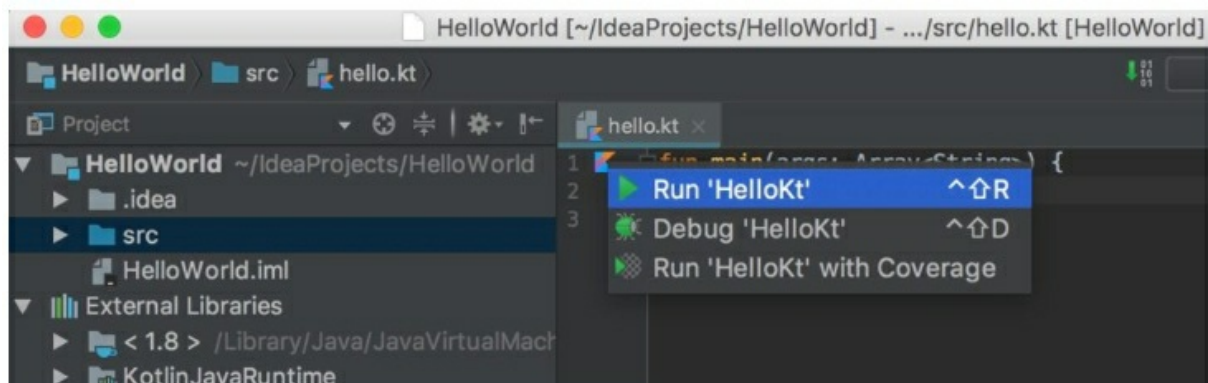


Step 4: Create a new Kotlin File in “src” folder of Kotlin Project.

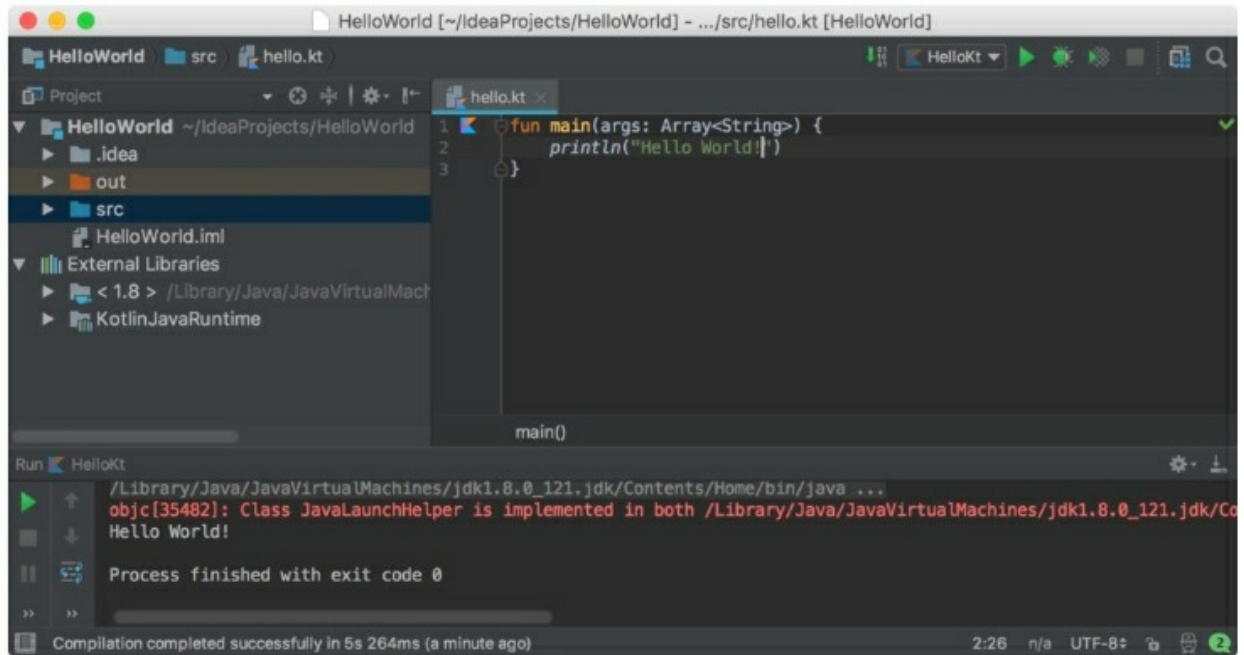


Step 5: Type the code

Step 6: To run the Kotlin file, click the **Kotlin Icon** located at the left side as shown in the screenshot below. Select Run 'HelloKt' to run the file.



Step 7: If everything went fine, you should see the output of the program at the bottom of the screen.



Kotlin Hello World Program – First Kotlin Program

Lets write a **simple Kotlin program** to display “**Hello World**” message on the screen. With the help of this simple program we will try to understand the basics of **Kotlin Programming**.

You can run Kotlin program in Eclipse IDE or the popular IntelliJ IDEA IDE. You can refer the following tutorials to learn, how to create and run your first Kotlin programs in these IDEs.

1. Create and Run Kotlin Project in Eclipse IDE
2. Create and Run Kotlin Project in IntelliJ IDEA IDE

Hello World Program in Kotlin

```
// Display Hello World! on screen
fun main(args : Array<String>) {
    println("Hello World!")
}
```

Output:

Hello World!

Lets discuss Hello World Program in detail

1. The first line of the program is:

```
// Display Hello World! on screen
```

This is a comment. You can write anything here, the compiler ignore these comments while executing the program. Comments improve the code readability so when a programmer reads them, they can easily understand the purpose of code, by just reading the comment.

2. The second line of the program is:

```
fun main(args : Array<String>) { }
```

This is the main function. Similar to [java](#), the execution of the Kotlin program starts from this function. This function is the starting point of the

Kotlin program. This is the mandatory function of the Kotlin program.

3. The third line of the program is:

```
println("Hello World!")
```

This is similar to the `System.out.println("Hello World!")` statement in java.

The purpose of this statement to display the message inside double quotes on the screen.

Kotlin Basics

Kotlin Keywords, Soft Keywords and Identifiers

There are certain words in Kotlin that have special meaning and cannot be used as identifiers(variable name, function name, class name etc). These words are called reserved words or keywords. In this guide, we will learn about **keywords and identifiers**.

Types of Keywords in Kotlin

We have two types of keywords:

1. Hard Keywords
2. Soft Keywords

1. Hard Keywords

These keywords cannot be used as identifiers. For example

This is valid:

```
//valid variable name  
val myvar = 5
```

This is invalid:

```
//error: "else" cannot be used as a variable name  
val else = 5
```

Kotlin Hard keywords Table

as	class	break	continue	do	else
for	fun	false	if	in	interface
super	return	object	package	null	is
try	throw	true	this	typeof	typealias
when	while	val	var		

2. Soft Keywords

Soft keywords are the keywords that are used only in a certain context which means we can use them as identifier. Apart from the above list of keywords, there are other keywords that can be used as identifiers. For example, “by” is a soft keyword which delegates the implementation of an interface to another object. We can use the keyword “by” as identifier as well.

//valid code

```
fun main(args: Array) {
    val by=10
    println(by+10)
}
```

There are several other soft keywords available in Kotlin such as catch, get, finally, field etc.

Kotlin Identifiers

The name that we give to a variable, class, function etc is known as identifier. For example:

```
var num = 100
```

Here num is an identifier.

Naming convention of Kotlin Identifiers

1. The identifier cannot have whitespaces.
2. Identifiers in Kotlin are case sensitive.
3. They cannot contain special characters such as @, #, % etc.
4. An identifier can start with an underscore “_”.

5. It is a best practise to give meaningful names to the identifiers. For example: add, multiply and divide are the meaningful identifier than the a, m and d.
6. If you wish to include two words in an identifier than you can start the second word with a capital letter. For example, sumOfTwo.

Kotlin Variables and Data Types

There are two types of variables – mutable and immutable. An immutable variable is one whose value cannot be changed, also known as unchangeable or read-only variable. On the other hand the value of the mutable variable can be changed.

Immutable variable: val keyword

Immutable variable is declared using `val` keyword in Kotlin. In this example, we have declared an immutable variable `myName` using `val` keyword and later displayed the value of it.

```
fun main(args : Array<String>){  
    /**  
     * This is an immutable variable  
     * also called unchangeable variable  
     * or read-only variable.  
     */  
    val myName = "TOM SEL"  
    println("My Name is: "+myName)  
}
```

Output:

My Name is: TOM SEL

What happens when we try to change the value of an immutable variable?

If we try to change the value of an immutable variable, we will get a compilation error, just like we get in the following example.

```
fun main(args : Array<String>){  
    /**  
     * This is an immutable variable  
     * also called unchangeable variable
```



```

    * or read-only variable.
    */
    val myName = "TOMSEL"
    myName="TOM SEL"
    println("My Name is: "+myName)
}

```

Output:

Error:(13, 5) Kotlin: Val cannot be reassigned

Mutable variable: var keyword

Unlike immutable variable, we can change the value of a mutable variable. In kotlin, we use var keyword to declare a mutable variable. Lets take an example to understand this.

In this example we have declared a mutable variable using `var` keyword, to demonstrate that we can change the value of it, we have reassigned a different value to `myName` variable.

```

fun main(args : Array<String>){
    /**
     * This is an mutable variable
     * we can change the value of this
     * variable
     */
    var myName = "TOMSEL"
    myName="TOM SEL"
    println("My Name is: "+myName)
}

```

Output:

My Name is: TOM SEL

What is a variable?

A variable is a name, given to a location in memory that can hold data. For example, when I declare the variable like this:

```
var website = "beginnersbook"
```

The data “beginnersbook” is stored in a memory at a particular location, named as website.

Here var is a keyword which is used for declaring a variable, website is an identifier(name of the variable), “beginnersbook” is the data(value of the variable) and the type of variable is String(we will discuss it later).

Type inference

As we mentioned earlier, we can declare and initialize the variable in a single statement like this:

```
var website = "beginnersbook"
```

In the above statement we have not specified the type of the variable, kotlin knows that the variable `website` is a string. Compiler can understand the type of the variable by looking at the values.

However if you want to explicitly mention the type of the variable then you can do that like this:

```
var website: String = "beginnersbook"
```

Here we have explicitly mentioned the type of variable “website” as String.

Declaration first and initialization later:

We can declare the variable first and then we can initialize it later in the program. The important point to note here is that we you do it like this, you must have to specify the type of the variable during declaration.

//It is mandatory to specify the type in this case

```
var website: String  
website = "beginnersbook"
```

Kotlin Data Types

1. Numbers – Byte, Short, Int, Long, Float, Double
2. Boolean – True, false
3. Characters
4. Arrays
5. Strings

Numbers

We have several data types to represent numbers in Kotlin.

1. Byte

The range of Byte data type is -128 to 127. This is used for representing the smaller integer values.

```
fun main(args : Array) {  
    val num: Byte = 99  
    println("$num")  
}
```

Output:

99

You must be aware of the range of the data type to avoid errors. For example, the following code would generate an error because the value assigned to the variable of type byte is not in the range.

```
fun main(args : Array) {  
    //Range is -128 to 127  
    val num: Byte = 300  
    println("$num")  
}
```

Output:

error

we can also know the min and max value of a data type like this:

```
fun main(args : Array<String>){  
  
    var bigByte: Byte = Byte.MAX_VALUE  
    var smallByte: Byte = Byte.MIN_VALUE  
    println("Biggest Byte value is: " + bigByte)  
  
    /**  
     * I can use String interpolation and put  
     * the variable inside quotes to display its  
     * value  
     */  
    println("Smallest Byte value is: $smallByte")  
}
```

OUTPUT

Biggest Byte value is: 127
Smallest Byte value is: -128

2. Short

The range of Short data type is -32768 to 32767.

```
fun main(args : Array) {  
    val num: Short = 29999  
    println("$num")  
}
```

Output:

29999

You may be wondering when we have a larger range in Short data type, why we use Byte data type?

This is to save the memory. The Short data type holds more memory compared to the Byte data type so if you are sure that the value would be in the limit of -128 to 127 then the Byte data type would be a better choice from the memory perspective

3. Int

The range of Int data type is -2^{31} to $2^{31}-1$

```
fun main(args : Array) {  
    val num: Int = 123456  
    println("$num")  
}
```

Output:

123456

Note: If you do not explicitly specify the type of the variable then the compiler would treat that variable as Int, if the value ranges between -2^{31} to $2^{31}-1$

4. Long

The range of Long data type is -2^{63} to $2^{63}-1$

```
fun main(args : Array) {  
    val num: Long = 12345678  
    println("$num")  
}
```

Output:

12345678

Note: If you do not explicitly specify the type of the variable then the compiler would treat that variable as Long, if the value is out of the range of -2^{31} to $2^{31}-1$ but lies in the range of -2^{63} to $2^{63}-1$

5. Double

```
fun main(args : Array) {  
    // all floating numbers are double by default  
    // unless you suffix the value with F letter  
    val num = 101.99  
    println("$num")  
}
```

Output:

101.99

6. Float

```
fun main(args : Array) {  
    // This is a float data type as we have suffixed  
    // the value with letter 'F'  
    val num = 101.99F  
    println("$num")  
}
```

Output:

101.99

Boolean

The value of boolean variable is either true or false.

```
fun main(args : Array) {  
    val boolValue = false
```

```
println("$boolValue")  
}
```

Output:

false

Char

All english alphabets(lowercase or uppercase) are included in Char data type.
You cannot assign numbers to the variable of Char data type.

```
fun main(args : Array) {  
    val ch = 'A'  
    println("$ch")  
  
    val ch2: Char  
    ch2 = 'Z'  
    println("$ch2")  
}
```

Output:

A
Z

Kotlin Type Casting with examples

Type casting is a process of converting one data type to another type, for example – converting int to long, long to double etc. In this tutorial we will learn how to do **type conversion in Kotlin**.

Type conversion in Kotlin vs Java

In java, one type is automatically converted to other type (in some cases), In Kotlin we need to explicitly convert the type.

For example:

Java:

Int is automatically converted to Long data type as long is larger than int.

```
// This code is valid in Java, even though we are converting int to long  
// because long is larger than int and it can hold integers
```

```
int num1 = 101;  
long num2 = num1;
```

Kotlin:

In Kotlin the conversion is not automatic, we need to explicitly do the type conversion.

```
// This code is invalid in Kotlin. This will cause compile time  
// error, type mismatch
```

```
val num1: Int = 101  
val num2: Long = num1
```

Correct code in Kotlin:

We use the `toLong()` function to convert int to long in Kotlin.

```
val num1: Int = 101  
val num2: Long = num1.toLong()
```

More functions for type conversion in Kotlin

In the above example, we have seen how we used `toLong()` function to convert an integer to Long type. Similarly we have other functions available in Kotlin to help us in type conversion.

1. **toChar()** – To convert a type to `Char` type.

2. **toInt()** – To convert a type to `Int` type.
3. **toLong()** – To convert a type to `Long` type.
4. **toFloat()** – To convert a type to `Float` type.
5. **toDouble()** – To convert a type to `Double` type.
6. **toByte()** – To convert a type to `Byte` type.
7. **toShort()** – To convert a type to `Short` type.

Simple Type Casting Examples

```
fun main(args : Array<String>){  
    /**  
     * Double to int type casting  
     */  
    println("4.554 to int: " + (4.554.toInt()))  
  
    /**  
     * int to Char type casting  
     */  
    println("66 to Char: " + (65.toChar()))  
  
    /**  
     * Char to int type casting  
     */  
    println("B to int: " + ('B'.toInt()))  
}
```

Output:

```
4.554 to int: 4  
66 to Char: A  
B to int: 66
```

What happens when we convert a larger type to smaller

type?

When converting a larger type to a smaller type, we have two possible outcomes. If the value that we are converting to a smaller type is outside the range of target type, the result is then truncated. However when the value is within the range of target type, it is converted without being truncated.

Example 1: When the value is outside the range of target type

In this example we are converting a long type to int type where the value of long type is outside the range of integer type.

```
fun main(args : Array<String>) {  
    val num1: Long = 5453448989999998988  
    val num2: Int = num1.toInt()  
    println("Number num1 is: $num1")  
    println("Number num2 is: $num2")  
}
```

Output:

```
Number num1 is: 5453448989999998988  
Number num2 is: 2041161740
```

Example 2: When the value is within the range of target type

In this example we are converting a long type to int type where the value of long type is within the range of integer type.

```
fun main(args : Array<String>) {  
    val num1: Long = 5453448  
    val num2: Int = num1.toInt()  
    println("Number num1 is: $num1")  
    println("Number num2 is: $num2")  
}
```

Output:

```
Number num1 is: 5453448  
Number num2 is: 5453448
```

Kotlin Operators – Arithmetic, Assignment, Unary, Logical and More

Operators works on operands and used for calculation. For example, + is an operator and when I use it in a expression like this: a+b, it performs addition on operands a & b. In this guide, we will learn types of operators available in Kotlin and how to use them.

Operators in Kotlin are grouped in following category

1. Arithmetic operators

1. Arithmetic Operators

- + Addition Operator
- Subtraction Operator
- Multiplication Operator
- / Division Operator
- % Modulus Operator

Example of Arithmetic Operators

```
fun main(args: Array<String>) {
```

```
    val num1 = 101.99
```

```
    val num2 = 100.50
```

```
    var op: Double
```

```
    op = num1 + num2
```

```
    println("Addition: $op")
```

```
    op = num1 - num2
```

```
    println("Subtraction: $op")
```

```
    op = num1 * num2
```

```
    println("Multiplication: $op")
```

```
    op = num1 / num2
    println("Division: $op")

    op = num1 % num2
    println("Modulus: $op")
}
```

Output:

Addition: 202.49

Subtraction: 1.48999999999999949

Multiplication: 10249.9949999999999

Division: 1.014825870646766

Modulus: 1.48999999999999949

The Addition operator (+) is also used for the string concatenation. You can read about it here:

[String Concatenation in Kotlin](#)

Kotlin – How to take Input from User

In this tutorial, we will see how to **take input from user in Kotlin**.

Example 1: Display String entered by user

In this example, we will take the input from user and display it in the output. Here we are using `readLine()` function to read the string entered on console.

```
fun main(args: Array<String>) {  
    print("Write anything here: ")  
  
    val enteredString = readLine()  
    println("You have entered this: $enteredString")  
}
```

Output:

Write anything here: welcome to beginner.com

You have entered this: welcome to beginner.com

Example 2: Taking input and converting it into a different type

As we have seen in the above example that the `readLine()` function reads the input as a String. If we want to take the input in a different type such as integer, long, double then we need to either explicitly convert the input to another type or use the `java Scanner` class.

Taking the input as String and converting it to an int

Here we are explicitly converting the input to an integer number.

```
fun main(args: Array<String>) {  
    print("Write any number: ")  
  
    val number = Integer.valueOf(readLine())  
    println("The entered number is: $number")  
}
```

Output:

Write any number: 101

The entered number is: 101

Taking the input other than String using Scanner class

In this example we are taking the input as an integer and float using the `nextInt()` and `nextFloat()` functions respectively. Similarly we can use `nextLong()`, `nextDouble()` and `nextBoolean()` methods to take long, double and boolean inputs respectively.

```
import java.util.Scanner
fun main(args: Array<String>) {
    //creating Scanner object
    val read = Scanner(System.`in`)

    //Taking integer input
    println("Enter an integer number: ")
    var num1 = read.nextInt()

    //Taking float input
    println("Enter a float number: ")
    var num2 = read.nextFloat()

    //Displaying input numbers
    println("First Input Number: "+num1)
    println("Second Input Number: "+num2)
}
```

Output:

Enter an integer number:

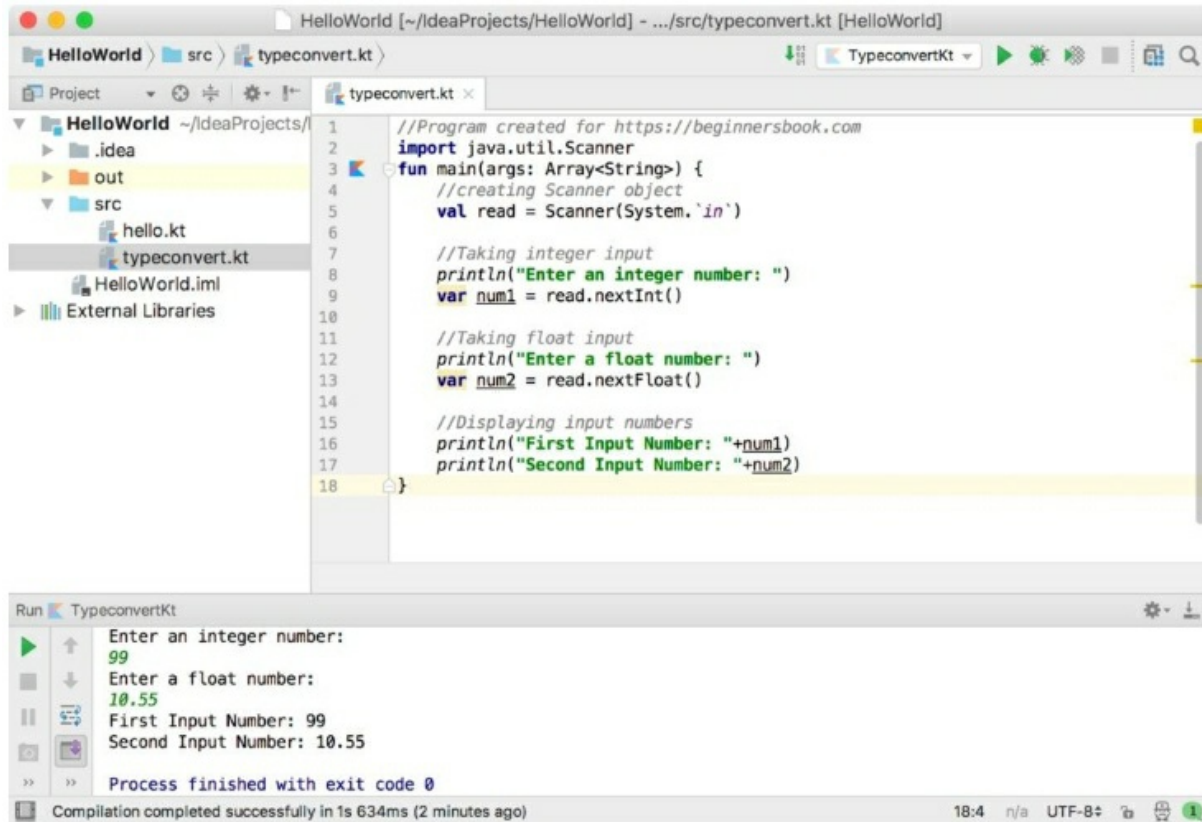
99

Enter a float number:

10.55

First Input Number: 99

Second Input Number: 10.55



Kotlin Comments

Comment is an important part of your program, it improves the readability of your code and makes it easier to understand. Writing meaningful comments in the code is considered a good programming practice. Similar to [Java](#), Kotlin also supports single line and multi-line comments. The syntax of **comments in Kotlin** is same as Java.

What is a comment?

A comment is a meaningful text written by the programmer in the various places of the code to explain the purpose and logic of the code. Comments are written to improve the readability of the code, they play no role in the compilation and execution of the program and they are completely ignored by the compiler during the compilation of the program.

Types of Comments in Kotlin

There are two types of comments in Kotlin – 1) Single line comment 2) Multiple line comment

1) Single line comment

As the name suggests, this type of comment is to write a single line of comment. Single line comments are started with the double slash, any text written after double slash is ignored by the compiler.

```
fun main(args: Array<String>) {  
    //This is a single line comment  
    println("Hello World!")  
}
```

The text after double slash “This is a single line comment” is a comment. I have written this text to demonstrate the use of comment, However the text should explain what is the purpose of following line or block of the code, so the meaningful comment should look like this:

```
fun main(args: Array<String>) {  
    //This displays the "Hello World!" message on the output screen  
    println("Hello World!")  
}
```

2) Multi-line comment

There are cases where we need to explain the piece of code in several lines, in such cases we use the multi-line comments. We start the comment with `/*` and ends the comment with `*/`.

For example:

```
fun main(args: Array<String>) {  
    /* This is a multi-line comment. I'm writing this  
       * text to show you that we can write the comments  
       * in multiple lines  
       */  
    println("Hello World!")  
}
```


Kotlin String

There are whole bunch of ways we can define a String in Kotlin. Lets have a look at the following example, here we have declare two immutable strings `website` & `longString` and we have also declared two mutable strings `name` & `lName`. This example also shows a simple way of string concatenation, to read more refer [String Concatenation in Kotlin](#) guide.

```
fun main(args : Array<String>){  
  
    /**  
     * These Strings are Immutable which  
     * means they are read-only and  
     * unchangeable  
     */  
    val website = "KOTLIN"  
  
    /**  
     * This is how we declare long strings  
     */  
    val longString = """"Welcome to  
        kotlin.com"""  
  
    /**  
     * Mutable Strings  
     */  
    var name = "Tom"  
    var lName = "Sel"  
    name = name + " " + lName  
    println("Name is: $name")  
}
```

Output:

Name is: Tom Sel

Get String Length in Kotlin

Lets see how we can get the String length in Kotlin. In the following example we have a String `firstName` and we are displaying the length of this string.

```
fun main(args : Array<String>){  
  
    var firstName = "Tom"  
  
    /**  
     * String interpolation  
     */  
    println("String Length: ${firstName.length}")  
  
    /**  
     * Or you can display like this  
     */  
    println("String Length: " + firstName.length)  
}
```

Output:

String Length: 3
String Length: 3

Compare Strings in Kotlin

Lets take an example where we compare two Strings in Kotlin. There are two ways to compare Strings, either using `equals()` method or using `compareTo()` method.

```
fun main(args : Array<String>){  
  
    var str1 = "KotlinBeginners"
```

```

var str2 = "Kotlinbeginners"

/**
 * true if equals, otherwise false
 */
println("String Equals? : ${str1.equals(str2)}")

/**
 * 0 if equals, otherwise false
 */
println("String Equals? : ${str1.compareTo(str2)}")
}

```

Output:

String Equals? : false

String Equals? : -32

Access character in a string at a specific index

We can get a character from a specific index in a string using the `get()` method which is equivalent to the `charAt()` method of Java

```

fun main(args : Array<String>){

    var str = "BeginnersBook"

    println("3rd Index: ${str.get(3)}")

    /**
     * Another way of doing the same
     * This is the recommended way
     */
    println("3rd Index: ${str[3]}")
}

```

Output:

3rd Index: i

3rd Index: i

Substring

We can display a substring in Kotlin using the `subSequence()` method. We can provide the `fromIndex` and `toIndex` in the `subSequence(fromIndex, toIndex)` method where `fromIndex` is inclusive and `toIndex` is exclusive.

```
fun main(args : Array<String>){  
  
    var str = "BeginnersBook"  
  
    /**  
     * Here fromIndex is inclusive and  
     * toIndex is exclusive which means  
     * 5th index char will not be included  
     */  
    println("Index from 2 to 5: " +  
            str.subSequence(2,5))  
}
```

Output:

Index from 2 to 5: gin

Check whether String contains another String

We can use `contains()` method to check whether the given string contains the specified string or not. Lets take an example to understand the usage of `contains()`

```
fun main(args : Array<String>){  
  
    var str = "kotlin.com"  
  
    println("Contains .com: ${str.contains(".com")}")  
}
```

Output:

Contains .com: true

Kotlin Array

Arrays in Kotlin are able to store multiple values of different data types. Of course if we want we can restrict the arrays to hold the values of a particular data types. In this guide, we will discuss about arrays in Kotlin.

Kotlin Array Declaration

Array that holds multiple different data types.

```
var arr = arrayOf(10, "BeginnersBook", 10.99, 'A')
```

Array that can only hold integers

```
var arr = arrayOf<Int>(1, 22, 55)
```

Array that can only hold strings

```
var arr2 = arrayOf<String>("ab", "bc", "cd")
```

Access Array elements in Kotlin

In the following example, we have an array `arr` that has multiple elements of different data types, we are displaying the 4th element of the array using the index (`arr[3]`). Since array is mutable, we can change the value of an element, which we have shown in the example below.

```
fun main(args : Array<String>){  
  
    /**  
     * Array that holds multiple different data types  
     */  
    var arr = arrayOf(10, "   ", 10.99, 'A')  
  
    /**  
     * Displaying 4th element  
     */  
    println(arr[3])  
  
    /**  
     * modifying 4th element  
     */  
    arr[3] = 100
```

```
/**
 * Displaying 4th element again
 */
println(arr[3])
}
```

Output:

A
100

Kotlin Array set() and get() functions

get()

In the above example we have used the following statement to access the value of 4th element of the array.

```
arr[3]
```

We can rewrite the same statement using get function like this:

```
arr.get(3)
```

set()

In the above example we have used the following statement to modify the value of 4th element in the array.

```
arr[3] = 100
```

We can rewrite the same statement using set() function like this:

```
arr.set(3,100)
```

Size of an array

We can easily find out the size of an array like this:

```
fun main(args : Array<String>){
```

```
var arr = arrayOf(1, 2, 3, 4, 5)

println("Size of Array arr is: ${arr.size}")

}
```

Output:

Size of Array arr is: 5

Check the element in an array

We can also check whether an element exists in array or not using the contains().

```
fun main(args : Array<String>){

    var arr = arrayOf(1, 22, "CPS")

    println("Array contains CPS: ${arr.contains("CPS")}")

}
```

Output:

Array contains CPS: true

Kotlin Array first() and last() functions

We can easily find out the first and last element of an array using first() and last() function.

```
fun main(args : Array<String>){
```

```
var arr = arrayOf(1, 22, "CPS")

println("First Element: ${arr.first()}")
println("Last Element: ${arr.last()}")

}
```

Output:

First Element: 1

Last Element: CPS

Finding the index of an element in an array

We can find out the index of a particular element in an array using `indexOf()` function.

```
fun main(args : Array<String>){

    var arr = arrayOf(1, 22, "CPS")

    println("Index of 22: ${arr.indexOf(22)}")

}
```

Output:

Index of 22: 1

Kotlin Ranges

In this guide, we will discuss the very cool feature of Kotlin which is **ranges**. With the help of **ranges in Kotlin** we can easily create a list of sequence by specifying starting and ending value. For example a range of 1..5 would create a series of values 1, 2, 3, 4, 5. Similarly we can create character ranges such as 'A'..'D' which will create a series of values A, B, C, D. We can also create ranges in reverse order and several other things with ranges. Lets get started.

A simple example of Kotlin Ranges

In the following example, we have created two ranges, one is an integer range and other one is a character range. We are cycling through the elements of the ranges using for loop.

```
fun main(args : Array<String>){  
    println("Number range:")  
    for(num in 1..4){  
        println(num)  
    }  
    println("Character range:")  
    for(ch in 'A'..'E'){  
        println(ch)  
    }  
}
```

Check element in Ranges

We can also check whether a particular element is present in the range or not. Lets see how to do this with the help of a simple example.

```
fun main(args : Array<String>){
```

```

val oneToTen = 1..10

println("3 in oneToTen: ${3 in oneToTen}")
println("11 in oneToTen: ${11 in oneToTen}")
}

```

Output:

```

3 in oneToTen: true
11 in oneToTen: false

```

Kotlin Range: rangeTo() and downTo() functions

Instead of .. we can use these functions rangeTo() and downTo(), rangeTo() is for increasing order and downTo() is for decreasing order.

```

fun main(args : Array<String>){

    val oneToFive = 1.rangeTo(5)
    val sixToThree = 6.downTo(3)

    println("rangeTo:")
    for(x in oneToFive){
        println(x)
    }

    println("downTo")
    for(n in sixToThree){
        println(n)
    }
}

```

Kotlin Range Step

With the help of step() function we can define the interval between the values. By default the value of step is 1 so when we create range 1..10, it is 1, 2, 3,..10. However if we want a specific interval like 3 then we can define the range like this 1..10.step(3) this way the values would be 1 4 7 10. Lets take an example.

```
fun main(args : Array<String>){  
    val oneToTen = 1..10  
    val odd = oneToTen.step(2)  
    for(n in odd){  
        println(n)  
    }  
}
```

Output:

```
1  
3  
5  
7  
9
```

Kotlin range reverse

We can reverse a range in Kotlin using reversed() function.

```
fun main(args : Array<String>){  
    val oneToFive = 1..5  
    for (n in oneToFive.reversed()){  
        println(n)  
    }  
}
```

Kotlin Control Flow

Kotlin If – Else Expression with examples

In any programming language, we need **control statements** to control the flow of the program based on the output of a condition. For example, if a number is even then display “even number” but if the number is odd then display “odd number”. To achieve this in programming, we need to use control statements that check whether a condition is satisfied or not, if it does then do this, if not then skip this. In **kotlin we use expressions to control flow** in the program. In this tutorial, we will learn several types of **expressions used in Kotlin**.

1. If expression
2. If..else expression
3. If..else if..else expression
4. Nested expressions

1. Kotlin – If expression

If expression is pretty simple to understand. Lets have a look at the syntax of if expression –

```
if(condition){  
    // Statements that need to be executed if condition is true  
    ...  
}
```

Here we have a condition in the if expression, if the condition returns true then the statements inside the body of if expression are executed, if the condition returns false then they are completely ignored. Lets take an example to understand this:

If expression example

In this example, if the given number is even then we are displaying “Even Number” in the output, else we are skipping the statements inside “if”.

```
fun main(args: Array<String>) {
```

```

val number = 100

// if expression
if (number%2 == 0)
    println("Even Number")

println("Out Of If statement")
}

```

Output:

Even Number

Out Of If statement

Lets change the value of variable “number”.

```

fun main(args: Array<String>) {

    val number = 101

    if (number%2 == 0)
        println("Even Number")

    println("Out Of If statement")
}

```

Output:

Out Of If statement

2. Kotlin – If..Else expression

If..Else expression is used when we need to perform some action if a condition is true and we need to perform a different action when a condition is false. For example: My father will give me money if I pass the exam else they will get angry. If I have to write this in programming then I would do it like this –

```

fun main(args: Array<String>) {
    // Marks out of 100
    val marks = 90
    if (marks < 30) {
        println("Father will get angry")
    }
}

```

```
    else {  
        println("Father will give me money")  
    }  
}
```

Output:

Father will give me money

Since the condition returned false, the statement inside “if” was skipped and the statement inside “else” was executed.

Syntax of if..else expression:

```
if(condition){  
    // Statements that will be executed if the condition returns true  
    ...  
}  
else{  
    // Statements that will be executed if the condition returns false  
    ...  
}
```

If..else expression example

In this example, we are checking a number to see whether it is positive or negative.

```
fun main(args: Array<String>) {  
    // Traditional usage  
    val num = -101  
    if (num < 0) {  
        println("Negative number")  
    }  
    else {  
        println("Positive number")  
    }  
    println("Out of if else statement")  
}
```

Output:

Negative number

Out of if else statement

3. Kotlin – if..else if..else ladder expression

In this expression we have one “if” block, one “else” block and one or more “else if” blocks. This is used for checking multiple conditions.

If..else if..else expression example

In this example, we have a number and we are checking whether it's a negative number, single digit number, two digit number or multiple digit number. We are checking these multiple conditions using if..else if..else expression. When none of the condition returns true then the statements inside the “else” block gets executed.

```
fun main(args: Array<String>) {  
    val num = 99  
    if(num<0)  
        println("Number is Negative")  
    else if (num>0 && num<10)  
        println("Single digit number")  
    else if (num>=10 && num <100)  
        println("Double digit number")  
    else  
        println("Number has 3 or more digits")  
}
```

Output:

Double digit number

4. Kotlin – Nested expression

When one expression is present inside another expression body then it is called nesting of expressions. For example if an “if expression” is present inside another “if” then it is called “nested if” expression.

For example:

```
fun main(args: Array<String>) {  
    val num = 101  
    if(num<0)  
        println("Negative Number")  
    else {  
        //Nested expression  
        if(num%2 == 0)
```

```
        println("Even Number")
    else
        println("Odd Number")
}
```

```
}
```

Output:

Odd Number

Kotlin When Expression with examples

The **when expression in Kotlin** works same as switch case in other programming languages such as [C](#), [C++](#) and [Java](#).

Kotlin when expression simple example

```
fun main(args : Array<String>){  
    var ch = 'A'  
    when(ch){  
        'A' -> println("A is a Vowel")  
        'E' -> println("E is a Vowel")  
        'I' -> println("I is a Vowel")  
        'O' -> println("O is a Vowel")  
        'U' -> println("U is a Vowel")  
        else -> println("$ch is a Consonant")  
    }  
}
```

Output:

A is a Vowel

We can also rewrite the same code in a more cleaner way like this:

```
fun main(args : Array<String>){  
    var ch = 'A'  
    when(ch){  
        'A', 'E', 'I', 'O', 'U' -> println("$ch is a Vowel")  
    }
```

```

        else -> println("$ch is a Consonant")
    }
}

```

Kotlin when expression with ranges

We can also use **ranges** in **when expression**. In the following example we have used multiple ranges inside the when expression to find out the digits in the given number.

```

fun main(args : Array<String>){

    var num = 78

    when(num) {
        in 1..9 -> println("$num is a single digit number")
        in 10..99 -> println("$num is a two digit number")
        in 100..999 -> println("$num is a three digit number")
        else -> println("$num has more than three digits")
    }
}

```

Arithmetic operation inside when expression

We can also perform operations on the variable that we pass in the when expression.

```

fun main(args : Array<String>){

    var age = 16

    when(age) {
        in 1..17 -> {
            val num = 18 - age
            println("You will be eligible for voting in $num years")
        }
        in 18..100 -> println("You are eligible for voting")
    }
}

```

}

Kotlin for Loop with examples

The for loop in Kotlin is used to iterate or cycle through the elements of array, ranges, collections etc. In this guide, we will learn how to use for loop in Kotlin with the help of various examples.

A simple example of for loop in Kotlin

In the following example we are iterating through an integer range using for loop.

```
fun main(args : Array<String>){  
    for(n in 10..15){  
        println("Loop: $n")  
    }  
}
```

Kotlin for loop using Array

In the following example we have declared an array `myArray` and we are displaying the elements of the array using for loop.

```
package beginnersbook
```

```
fun main(args : Array<String>){  
    val myArray = arrayOf("ab", "bc", "cd", "da")  
    for (str in myArray){  
        println(str)  
    }  
}
```

Output:

```
ab  
bc  
cd  
da
```

Kotlin for loop iterating through array indices

We can also use array indexes to iterate through the array.

```
fun main(args : Array<String>){  
  
    val myArray = arrayOf("Steve", "Robin", "Kate", "Lucy")  
    for (n in myArray.indices){  
        println("myArray[$n]: ${myArray[n]}")  
    }  
}
```

Function withIndex() in for loop

In the above example we have iterated through the array using array indices. Another way of doing the same is with the use of withIndex() function.

```
package beginnersbook
```

```
fun main(args : Array<String>){  
  
    val myArray = arrayOf("Steve", "Robin", "Kate", "Lucy")  
    for((index, value) in myArray.withIndex()){  
        println("Value at Index $index is: $value")  
    }  
}
```

Output:

```
Value at Index 0 is: Steve  
Value at Index 1 is: Robin  
Value at Index 2 is: Kate  
Value at Index 3 is: Lucy
```

Kotlin while Loop with examples

While loop is used to iterate a block of code repeatedly as long as the given condition returns true. In this guide, we will learn how to use while loop with the help of examples.

A simple while loop in Kotlin

In the following example we are displaying the value from 10 to 5 using while loop. The important point to note here is the counter, which is variable num in the following example, for ascending loop the counter value should increase to meet the given condition and for the descending loop the counter value should decrease in every iteration just like we did in the following example.

```
fun main(args : Array<String>){  
    var num = 10  
  
    while(num>=5){  
        println("Loop: $num")  
        num--  
    }  
}
```

Infinite While loop

If the condition specified in the while loop never returns false then the loop iterates infinitely and never stops such while loops are called infinite while loops. We should always avoid such situation while writing code. Lets see few examples of infinite while loop.

1. Since the condition is always true this will run infinitely.

```
while (true){  
    println("loop")  
}
```

2. In this while loop we are incrementing the counter `num`, the counter initial value is 10 and we are increasing it on every iteration, which means the specified condition `num >= 5` will always remain true and the loop will never stop.

```
var num = 10
```

```
while(num >= 5){  
    println("Loop: $num")  
    num++  
}
```

3. The following loop will also be an infinite loop because the condition will always remain true as we are decreasing the value of `num` which means the condition `num <= 10` will always be satisfied.

```
var num = 5
```

```
while(num <= 10){  
    println("Loop: $num")  
    num--  
}
```

Kotlin do-while Loop with examples

A **do-while loop** is similar to while loop except that it checks the condition at the end of iteration. A do-while loop will at least run once even if the given condition is false.

Kotlin do-while loop Example

```
fun main(args : Array<String>){  
    var num = 100  
    do {  
        println("Loop: $num")  
        num++  
    }  
    while (num<=105)  
}
```

A do-while loop at least run once

As I mentioned in the beginning of this guide, a do-while loop will at least run once even if the given condition returns false. This happens because the do-while loop checks the condition after execution of the loop body.

```
fun main(args : Array<String>){  
    var num = 100  
    do {  
        println("Loop: $num")  
        num++  
    }  
    while (false)  
}
```


Infinite do while loop in Kotlin

A do while loop that runs infinitely and never stops is called infinite do while loop. Lets look at the few examples of infinite do while loop.

Example 1:

```
var num = 100
do {
    println("Loop: $num")
    num++
}
while (true)
```

Example 2:

```
var num = 100
do {
    println("Loop: $num")
    num--
}
while (num<=105)
```

Example 3:

```
var num = 105
do {
    println("Loop: $num")
    num++
}
while (num>=100)
```

Kotlin continue Expression with examples

The **continue** construct skips the current iteration of the loop and jumps the control to end of the loop for the next iteration. The continue is usually used with if else expression to skip the current iteration of the loop for a specified condition. In this guide, we will learn Continue construct and Continue Labels.

Kotlin Continue For loop Example

```
fun main(args : Array<String>){  
    for (n in 1..5){  
        println("before continue, Loop: $n")  
        if(n==2||n==4)  
            continue  
  
        println("after continue, Loop: $n")  
    }  
}
```

OUTPUT

```
before continue, Loop: 1  
after continue, Loop: 1  
before continue, Loop: 2  
before continue, Loop: 3  
after continue, Loop: 3  
before continue, Loop: 4  
before continue, Loop: 5  
after continue, Loop: 5
```

As you can see in the output that `println("after continue, Loop: $n")` statement didn't execute for the loop iterations `n==2` and `n==4` because on these iterations we have used the continue before this statement which

skipped the iteration for these values of n.

However you can observe that `println("before continue, Loop: $n")` statement executed on every iteration because it is executed before `Continue` is encountered.

Based on this you can conclude that `continue` skips the iteration but it is not capable of skipping the statements that are encountered before `continue`. As soon as `continue` is encountered, the control jumps to the end of the loop skipping the rest of the statements.

Lets take another example

Kotlin continue example: displaying even numbers

```
fun main(args : Array<String>){
```

```
    for (n in 1..10){  
        if(n%2!=0)  
            continue
```

```
        println("$n")
```

```
    }  
}
```

OUTPUT

```
2  
4  
6  
8  
10
```

Continue Label

Till now we have learned how `continue` works. Lets learn about `continue labels`. These labels are cool basically they give us more control when we are dealing with nested loops.

Lets take an example where we first do not use **continue label** and then we will take the same example with **continue label**.

Nested loop example without continue label

In this example we have a nested for loop and we are not using label. When we do not use labels we do not have any control and as soon as

the `continue` is encountered the current iteration is skipped for the **inner loop**.

```
fun main(args : Array<String>){  
    for (x in 'A'..'D'){  
        for (n in 1..4){  
            if (n==2||n==4)  
                continue  
  
            println("$x and $n")  
        }  
    }  
}
```

Output:

```
A and 1  
A and 3  
B and 1  
B and 3  
C and 1  
C and 3  
D and 1  
D and 3
```

Nested loop example with continue label

You can see in the above example output that for n value 2 and 4 the iteration is skipped for the inner loop. Lets say we want to skip the iteration for the outer for loop, we can do so with the help of continue labels.

```
fun main(args : Array<String>){  
  
    myloop@ for (x in 'A'..'D'){  
        for (n in 1..4){  
            if (n==2||n==4)  
                continue@myloop  
        }  
    }  
}
```

```
        println("$x and $n")
    }
}
}
```

Output:

A and 1

B and 1

C and 1

D and 1

Basically what happened here is that as soon as the n value reached 2 the control jumped to the end of outer loop because of the label and it happened for the each iteration. The syntax of the continue label is pretty straight forward as you just have to give a name to label followed by @ symbol and the same name needs to be appended with the continue statement as shown in the above example.

Kotlin break Statement with examples

The **break statement** is used to terminate the loop immediately without evaluating the loop condition. As soon as the break statement is encountered inside a loop, the loop terminates immediately without executing the rest of the statements following break statement. In this guide, we will learn how break works and we will also discuss break labels.

Kotlin break example

The break statement is usually used with if else expression.

```
fun main(args : Array<String>){  
    for(n in 1..10){  
        println("before break, Loop: $n")  
        if (n==5) {  
            println("I am terminating loop")  
            break  
        }  
    }  
}
```

OUTPUT

```
before break, Loop: 1  
before break, Loop: 2  
before break, Loop: 3  
before break, Loop: 4  
before break, Loop: 5  
I am terminating loop
```

As you can observe in the output that as soon as the break is encountered the loop terminated.

Kotlin break example in nested loop

When break is used in the nested loop, it terminates the inner loop when it is encountered.

```
fun main(args : Array<String>){
```

```

for(ch in 'A'..'C'){
    for (n in 1..4){
        println("$ch and $n")
        if(n==2)
            break
    }
}

```

OUTPUT

```

A and 1
A and 2
B and 1
B and 2
C and 1
C and 2

```

As you can observe in the output that the outer loop never got terminated, however the inner loop got terminated 3 times.

Kotlin break labels

Lets talk about labels now. Similar to continue labels, the break label gives us more control over which loop is to be terminated when the break is encountered.

In the above example of nested loop, the inner loop got terminated when break encountered. Lets write a program with the help of labels to terminate the outer loop rather than inner loop.

```

fun main(args : Array<String>){

    myloop@ for(ch in 'A'..'C'){
        for (n in 1..4){
            println("$ch and $n")
            if(n==2)
                break@myloop
        }
    }
}

```

Output:

A and 1

A and 2

The syntax of label is simple we just have to use any name followed by @ in front of the loop which we want to terminate and the same name needs to be appended with the break keyword prefixed with @ as shown in the above example.

Kotlin Functions

Kotlin Function with examples

A function is a block of related statements that together perform a specific task. For example let's say we have to write three lines of code to find the average of two numbers, if we create a function to find the average then instead of writing those three lines again and again, we can just call the function we created.

Types of Function in Kotlin

There are two types of functions in Kotlin:

1. Standard Library Function
2. User-defined functions

1. Standard Library Function

The functions that are already present in the Kotlin standard library are called standard library functions or built-in functions or predefined functions. For example when we need to use the `Math.floor()` function we do not define the function because it is already present and we can directly call it in our code.

Standard Library Function Example

```
fun main(args : Array<String>){  
  
    var num = 16  
    println("Square root of $num is: ${Math.sqrt(num.toDouble())}")  
  
}
```

OUTPUT

Square root of 16 is: 4.0

2. User-defined functions

The functions that we define in our program before we call them are known as user defined functions. For example, let's say we want a function to check even or odd number in our program then we can create a function for this task and later call the function where we

need to perform the check .

User-defined functions Example

We create a function using `fun` keyword. Lets create a function that prints “Hello”.

```
//Created the function
fun sayHello(){
    println("Hello")
}
fun main(args : Array<String>){

    //Calling the function
    sayHello()

}
```

Output:

Hello

User defined function with arguments and return type

Syntax:

```
fun function_name(param1: data_type, param2: data_type, ...): return_type
```

Lets create a user defined function that accepts the arguments and has a return type. In the following program we have declared a function `sum`. This function accepts variable number of arguments thats why we have used `vararg`, those arguments are of type integers and the return type of the function is also an integer.

```
//Created the function
fun sum(vararg numbers: Int): Int
{
    var sum = 0
    numbers.forEach {num -> sum += num}
}
```

```
        return sum
    }
    fun main(args : Array<String>){

        println("Sum: ${sum(10, 20, 30, 40)}")

    }
```

OUTPUT

Sum: 100

Kotlin Inline functions

An inline function can be defined inside the main() function. Lets take an example of inline function. In the following example we have defined an inline function `sum` which accepts two integer arguments `num1` and `num2` and return type is integer.

```
fun main(args : Array<String>){

    val sum = {num1: Int, num2: Int -> num1 + num2}

    println("6 + 4 = ${sum(6,4)}")

}
```

Output:

6 + 4 = 10

Kotlin Recursion and Tail Recursion with examples

A function is called **recursive function** if it calls itself and this process is called **recursion**.

How a recursive function looks like?

Here the function `myfunction()` calls itself, this is a recursive function.

```
fun myfunction(){  
    //some code  
  
    ....  
    //myfunction() calling myfunction()  
    myfunction()  
}
```

Lets take an example to understand the recursion.

Kotlin Recursion example

This is a simple example of factorial. Here we have defined a function `fact()` to calculate the factorial of a number that it passed to this function as a parameter. In the function body we are calling this function again, this process is called recursion.

User is asked to enter a positive integer number and based on the input, program finds the factorial of the input number by passing the input number as an argument to the user defined function `fact()`.

```
fun main(args: Array<String>) {  
    print("Enter a positive integer number: ")  
    val number: Int = Integer.valueOf(readLine())  
    val factorial = fact(number)  
    println("Factorial of $number = $factorial")  
}
```

```
//recursive function  
fun fact(num: Int): Int {  
    return if(num == 1){  
        num
```

```

    }
    else{
        //function fact() calling itself
        num*fact(num-1)
    }
}

```

Tail Recursion

In recursion the computation is done after the recursive call, the example of factorial we have seen above is an example of recursion or head recursion where to calculate the factorial of n we need the factorial of $n-1$.

In **Tail recursion** the computation is done at the beginning before the recursive call. In tail recursion the call to the recursive function occurs at the end of the function. Which means the computation is done first and then passed to the next recursive call.

Lets take an example of tail recursion.

Tail Recursion Example

To declare a tail recursive function we use `tailrec` modifier before the function.

```

fun main(args: Array<String>) {
    val number = 6
    val factorial = fact(number)
    println("Factorial of $number = $factorial")
}

```

```

tailrec fun fact(n: Int, temp: Int = 1): Int {
    return if (n == 1){
        temp
    } else {
        fact(n-1, temp*n)
    }
}

```

Output:

Factorial of 6 = 720

Kotlin Default and Named Argument

In this guide, we will learn **default and named argument** that are used in Kotlin functions.

Kotlin Default Argument

We have already learned how a function can have parameters and we can pass the value of those parameters while calling the function, by using default arguments we can set the default value of those parameters while defining a function. Lets take an example to understand default arguments.

Default argument example

Not passing any value during function call

```
fun main(args: Array<String>) {  
    demo()  
}  
fun demo(number: Int = 100, ch: Char = 'A'){  
    print("Number is: $number and Character is: $ch")  
}
```

Output:

Number is: 100 and Character is: A

In the above example we have not passed any values while calling the function `demo()`, lets see what happens when we pass values to the function that already has default values set for their parameters.

Passing value of some of the parameters during function call

```
fun main(args: Array<String>) {  
    demo(99)  
}  
fun demo(number: Int = 100, ch: Char = 'A'){  
    print("Number is: $number and Character is: $ch")  
}
```

Output:

Number is: 99 and Character is: A

As you can see in the output that when we pass values while calling function then it overrides the default values. In the above example I have passed a single value while calling function that's why it overrides only first default value, however we can also pass values for all the parameters and this will override all the default values.

Passing values of all the parameters during function call

```
fun main(args: Array<String>) {  
    demo(99, 'Z')  
}  
fun demo(number: Int = 100, ch: Char = 'A'){  
    print("Number is: $number and Character is: $ch")  
}
```

Output:

Number is: 99 and Character is: Z

Kotlin Named Arguments

In the above examples we have learned how we can set default values for the parameters. In the second example we learned that we can call the function while passing few of the values, we have made the function call like this in the second example `demo(99)`, it overrides the first default value. However if we want to only override the second default value then we can do this with the help of **named arguments**.

```
fun main(args: Array<String>) {  
    demo(ch='Z')  
}  
fun demo(number: Int = 100, ch: Char = 'A'){  
    print("Number is: $number and Character is: $ch")  
}
```



```
}
```

Output:

Number is: 100 and Character is: Z

As you can see we have overridden the default value of second parameter by using the parameter name during function call `demo(ch='Z')` . If we would have done this without named parameter like this `demo('Z')` then this would have thrown error because then it would have tried to override the first integer parameter with this value.

Kotlin Lambda Function with example

Lambda function is also known as anonymous function because it has no name. Parameters are in the left side of the arrow and actual code is on the right side of the arrow. Don't worry we will take an example to explain this.

How a Lambda function looks?

```
{my_var -> actual_code_implementation}
```

Kotlin Lambda function example

In the following example we have defined a lambda function to add two integer numbers. The lambda function is defined in curly braces, the left side of the arrow is our parameters with their data type and in the right side of the arrow is the body of function.

```
fun main(args: Array<String>){  
    //lambda function  
    val sum = {num1: Int, num2: Int -> num1 + num2}  
    println("10+5: ${sum(10,5)}")  
}
```

OUTPUT

10+5: 15

Kotlin Higher order function with example

Higher order function or higher level function can have another function as a parameter or return a function or can do both. Till now we have seen how we pass integer, string etc. as parameters to a function but in this guide, we will learn how we pass a function to another function. We will also see how a function returns another function.

Kotlin Higher order function example: Passing a function to another function

In the following example we are passing a function `demo()` to another function `func()`. To pass a function as a parameter to other function we use `::` operator before function as shown in the following example.

```
fun main(args: Array<String>) {  
    func("KOTLIN", ::demo)  
}  
  
fun func(str: String, myfunc: (String) -> Unit) {  
    print("Welcome to Kotlin tutorial at ")  
    myfunc(str)  
}  
  
fun demo(str: String) {  
    println(str)  
}
```

Output:

Welcome to Kotlin tutorial at KOTLIN

Kotlin Higher order function example: function returns another function

In the following example the custom function `func` is returning another function.

To understand this code, let's look at the function `func` first, it accepts an integer parameter `num` and in the return area we have defined a function `(Int) -> Int = {num2 -> num2 + num}` so this is the other function which also accepts integer parameter and returns the sum of this parameter and `num`.

You may be wondering why we have passed the value 20 as a parameter in `sum`, well this is because the function `func` returned the function so the `sum` is the function that will accept the int parameter. This is the same function that we have defined in the return area of function `func`.

```
fun main(args: Array<String>) {  
  
    val sum = func(10)  
    println("10 + 20: ${sum(20)}")  
  
}  
  
fun func(num: Int): (Int) -> Int = {num2 -> num2 + num}
```

Output:

10 + 20: 30

Kotlin Exception Handling Tutorials

Kotlin Exception Handling with examples

Exceptions are unwanted issues that can occur at runtime of the program and terminate your program abruptly. Exception handling is a process, using which we can prevent the program from such exceptions that can break our code.

There are two types of exceptions:

1. Checked exceptions that are declared as part of method signature and are checked at the compile time, for example `IOException`
2. Unchecked exceptions do not need to be added as part of method signature and they are checked at the runtime, for example `NullPointerException`.

Note: In Kotlin all exceptions are unchecked.

Handling of exception in Kotlin is same as Java. We use try, catch and finally block to handle the exceptions in the kotlin code.

Kotlin Exception handling example

In the following example we are dividing a number with 0 (zero) which should throw `ArithmeticException`. Since this code is in `try` block, the corresponding catch block will execute.

In this case the `ArithmeticException` occurred so the catch block of `ArithmeticException` executed and “Arithmetic Exception” is printed in the output.

When an exception occurs, it ignores everything after that point and the control instantly jumps to the catch block if any. The finally block is always executed whether exception occurs or not.

```
fun main(args: Array<String>) {  
    try {  
        var num = 10/0  
        println("BeginnersBook.com")  
    }  
}
```

```

        println(num)

    } catch (e: ArithmeticException) {
        println("Arithmetic Exception")
    } catch (e: Exception) {
        println(e)
    } finally {
        println("It will print in any case.")
    }
}

```

Output:

Arithmetic Exception
It will print in any case.

What happens if we don't handle exception?

Lets say if we don't handle the exception in the above example then the program would terminate abruptly.

Here we didn't handle exception so the program terminated with an error.

```

fun main(args: Array<String>) {

    var num = 10/0
    println("Kotlin.com")
    println(num)
}

```

Output:

Exception in thread "main" java.lang.ArithmeticException: / by zero
at FileKt.main (File.kt:3)

How to throw an exception in Kotlin

We can also throw an exception using `throw` keyword. In the following

example we are throwing an exception using throw keyword. The statement before the exception got executed but the statement after the exception didn't execute because the control transferred to the catch block.

```
fun main(args: Array<String>) {  
    try{  
        println("Before exception")  
        throw Exception("Something went wrong here")  
        println("After exception")  
    }  
    catch(e: Exception){  
        println(e)  
    }  
    finally{  
        println("You can't ignore me")  
    }  
}
```

Output:

```
Before exception  
java.lang.Exception: Something went wrong here  
You can't ignore me
```

Kotlin Try Catch with example

In the last tutorial we learned what is an **exception handling**. In this guide we will see various examples of try catch block. We will also see how to use try as an expression.

Syntax of try catch block

```
try {  
    //code where an exception can occur  
}  
catch (e: SomeException) {  
    // handle the exception  
}  
finally {  
    // optional block but executes always  
}
```

A try block can be associated with more than one catch blocks, however there can be only one finally block present.

Kotlin try catch block example

In this example we have placed the code that can cause exception inside try block. Once the exception occurs inside try block, it looks for the corresponding catch block where the occurred exception is handled. Since in the code `ArithmeticException` is occurred and the same exception is handled in the catch block, the code inside catch block is executed.

The main advantage of exception handling is that the program doesn't terminate abruptly. In the following example the last `println` statement `println("Out of try catch block")` is executed after catch block. If we didn't have exception handling in place then this statement wouldn't be executed as the program would have been terminated on the line `var num = 100/0`

```
fun main(args: Array<String>) {  
    try{  
        var num = 100/0  
        println(num)  
    }
```



```

        catch(e: ArithmeticException){
            println("Arithmetic Error in the code")
        }
        println("Out of try catch block")
    }
}

```

Output:

Arithmetic Error in the code
Out of try catch block

Kotlin try block with no catch blocks

A try block can have no catch blocks but in that case a finally block must be present. **In short you can say that at least one catch or finally block should be present.** Finally block is optional but when there are no catch blocks present, it is must to have a finally block.

```

fun main(args: Array<String>) {
    try{
        var num = 10/5
        println(num)
    }
    finally{
        println("Finally block")
    }

    println("Out of try catch block")
}

```

Output:

2
Finally block
Out of try catch block

Kotlin Multiple Catch Blocks with example

A try block can have multiple catch blocks. When we are not sure what all exceptions can occur inside the try block then it is always a good idea to have multiple catch blocks for the potential exceptions and in the last catch block have the parent exception class to handle the remaining exceptions that are not specified by catch blocks.

Kotlin multiple catch blocks example

In the following example we have multiple catch blocks but when an exception occurs it looks for the handler for that particular exception. The exception occurred here is Arithmetic exception, however the first two catch blocks didn't handle the Arithmetic exception that's why the third catch block's code executed. Third block is handling all the exceptions because it is using the Exception class which is a parent of all the exception classes.

```
fun main(args: Array<String>) {  
    try{  
        var num = 10/0  
        println(num)  
    }  
    catch(e: NumberFormatException){  
        println("Number format exception")  
    }  
    catch(e: ArrayIndexOutOfBoundsException){  
        println("Array index is out of range")  
    }  
    catch(e: Exception){  
        println("Some Exception occurred")  
    }  
  
    println("Out of try catch block")  
}
```

Output:

Some Exception occurred

Out of try catch block

Another example of multiple catch blocks

Here is another example of multiple catch blocks, here

`ArrayIndexOutOfBoundsException` occurred, since there is a handler (catch block) present for this exception, the code inside the handler is executed.

```
fun main(args: Array<String>) {  
    try{  
        val a = IntArray(5)  
        a[10] = 99  
    }  
    catch(e: ArithmeticException){  
        println("ArithmeticException occurred")  
    }  
    catch(e: NumberFormatException){  
        println("Number format exception")  
    }  
    catch(e: ArrayIndexOutOfBoundsException){  
        println("Array index is out of range")  
    }  
    catch(e: Exception){  
        println("Some error occurred")  
    }  
  
    println("Out of try catch block")  
}
```

Output:

Array index is out of range

Out of try catch block

Why it is a good idea to have parent Exception

class in the last catch block

Lets take the same example that we have above but in this code, we have made a small change. Here we have the handler (catch block) of parent Exception class in the first place.

In the code `ArrayIndexOutOfBoundsException` occurred and we have the handler for this particular exception but since we have the general `Exception` class in the first place and it handles all the exceptions so it got executed instead of the catch block that handles `ArrayIndexOutOfBoundsException`. The catch blocks are checked in the sequential manner so the first catch block is executed, in fact in case of any exception the first catch will execute which is a bad programming practice because we want a specific message rather than a generalized message. So the solution is to have this default handler at the last place like we did in the above example.

```
fun main(args: Array<String>) {  
    try{  
        val a = IntArray(5)  
        a[10] = 99  
    }  
    catch(e: Exception){  
        println("Some error occurred")  
    }  
    catch(e: ArithmeticException){  
        println("ArithmeticException occurred")  
    }  
    catch(e: NumberFormatException){  
        println("Number format exception")  
    }  
    catch(e: ArrayIndexOutOfBoundsException){  
        println("Array index is out of range")  
    }  
    println("Out of try catch block")  
}
```

Output:

Some error occurred
Out of try catch block

Kotlin Nested Try-Catch Block with example

When a try catch block is present inside another try catch block then it is called nested try catch block. If any exception occurs in the inner try catch block which is not handled in the inner catch blocks then the catch blocks of the outer try catch blocks are checked for that exception.

Syntax of nested try catch block

```
try{
    //code
    try{
        //code
    }
    catch
    {
        //handler
    }
}
catch
{
    //handler
}
```

Nested Try-Catch Block example

In the following example there is an exception in the inner try block but the occurred exception (ArithmeticException) is not handled in the inner catch blocks so the outer catch blocks are checked for this exception, since outer catch block is handling this exception, the code inside outer catch block is executed for ArithmeticException.

There can be more than one try catch blocks in a try block, also there can be a try catch block inside the inner try block as well. The only thing to take note here is that if the exception is not handled in the child try catch block, then the handlers of parent try catch blocks are checked for the occurred exception.

```
fun main(args: Array<String>) {
```

```
try {  
    val num = 100 / 5  
    println(num)  
    try {  
        val num2 = 100 / 0  
        println(num2)  
    }  
    catch(e: NumberFormatException){  
        println("Number Format Exception")  
    }  
}  
catch(e: ArithmeticException){  
    println("Arithmetic Exception")  
}  
}
```

Output:

20

Arithmetic Exception

Kotlin throw keyword with example

In this short guide, we will learn how to throw an exception using throw keyword in Kotlin.

Kotlin throw keyword example

In the following example we are throwing an exception using throw keyword. Here I have thrown parent Exception class, you can throw any exception such as ArithmeticException, ArrayIndexOutOfBoundsException etc.

```
fun main(args: Array<String>) {  
    print("Enter your name: ")  
    val name = readLine()  
  
    try{  
        if (name == "Tamsel"){  
            throw Exception("You don't have access")  
        }  
        else  
        {  
            println("Welcome! You have access")  
        }  
    }  
    catch (e: Exception){  
        println(e.message)  
    }  
}
```

Output:

Enter your name: Welcome! You have access

Kotlin Try as an expression in Exception handling

The try block in Kotlin can work as an expression. It can return a value just like any other expression, the returned value can be stored in a variable. In this guide, we will learn how to use try as an expression in Kotlin.

Kotlin try as an expression example

In the following example try block is working as an expression, the value returned by try can be stored in the variable as shown in the following example.

```
fun main(args: Array<String>) {  
    var website = "Kotlin.com"  
    var num = try {  
        website.toInt()  
    }  
    catch (e: NumberFormatException)  
    {  
        "Cannot convert String to integer"  
    }  
    println(num)  
  
    var number = "100"  
    var num2 = try {  
        number.toInt()  
    }  
    catch (e: NumberFormatException)  
    {  
        "Cannot convert String to integer"  
    }  
    println(num2)  
}
```

Output:

Cannot convert String to integer
100

Kotlin Class and Objects – Object Oriented Programming (OOP)

Kotlin is an object oriented programming language just like Java. Object oriented programming (OOP) allows us to solve the complex problem by using objects. In this guide, we will learn what is a class, what is an object and several other features of Object Oriented Programming (OOP).

Kotlin Class

Classes are the main building blocks of any object oriented programming language. All objects are part of a class and share the common property and behaviour defined by the class in form of data members and member functions respectively.

A class is like a blueprint for the objects.

A class is like a prototype for objects which you can create by grouping methods and variables.

How to define a class in Kotlin

A class is defined using `class` keyword in Kotlin. For example –

```
class MyClass {  
    // variables or data members  
    // member functions  
    ..  
    ..  
}
```

Kotlin class Example

In the following example we have a class `Example` and in that class we have a data member `number` and a member function `calculateSquare()`. Data member also known as property and member function as behaviour in the object oriented programming language terms. The objects of this class share these properties and behaviours.

```
class Example {  
    // data member  
    private var number: Int = 5
```

```
// member function
fun calculateSquare(): Int {
    return number*number
}
}
```

I have not specified any access modifier in the above class. Access modifiers restrict the access. By default the access modifier is `public`. In the above example we have not specified any access modifier so by default `public` access modifier is applicable for the above class `Example`

The data member in the above class is specified as `private` which means the data member `number` is not accessible outside the class `Example`.

How to define access modifiers?

Access modifiers are specified before the class keyword.

```
private class MyClass {
}
```

Other Access modifiers:

`private` – Can be accessed inside the class only.

`public` – Can be accessed everywhere.

`protected` – Can be accessed to the class and its subclasses.

`internal` – Can be accessed inside the module.

Kotlin Object

Objects use the properties and behaviours of the class. As mentioned above, class is just a blueprint, no memory is allocated to the class. Once the objects of the class are created they take memory space and perform various actions on the data using data members and members functions of the class.

How to create object of class

Lets say my class name is `Example`, the object of this class can be

created like this :

```
Example e = Example()
```

Note: Here I have named the object as `e`, you can give any name to object except the already defined keywords in the Kotlin. Another point to note here is that unlike java where we use `new` keyword to create the object, we do not use the `new` keyword here, in fact if you use it, you will get a compilation error.

How to access the data members and member functions

To access the data member `number` and member function `calculateSquare` of the class `Example` using the object `e`.

```
//Access data member  
e.number
```

```
//Access member function  
e.calculateSquare()
```

Lets take a complete example to understand the above discussed concepts.

Kotlin object example

```
class Example {  
    // data member  
    private var number: Int = 5  
  
    // member function  
    fun calculateSquare(): Int {  
        return number*number  
    }  
}  
  
fun main(args: Array<String>) {  
    // create obj object of Example class  
    val obj = Example()
```

```
println("${obj.calculateSquare()}")  
}
```

Output:

25

Kotlin Constructors with examples

The main purpose of constructor is to initialize the properties of a class. Constructor is called when we create the object of a class. In Kotlin we have two types of constructor – primary and secondary constructor. In this guide, we will learn primary and secondary constructor with example, we will also learn about initializer blocks.

1. Primary Constructor – Initialize the properties of class
2. Secondary Constructor – Initialize the properties of class, we can have additional initialization code inside secondary constructor.

1. Primary Constructor

A primary constructor is the easiest way to initialize the class. It is declared as part of the class header. In the following example we have declared a constructor (`val name: String, var age: Int`) as part of the class header. This is our primary constructor that initializes the `name` and `age` properties (data members) of class `Student`.

```
fun main(args: Array<String>) {  
  
    //creating the object of class Student  
    val stu = Student("Tamsel", 31)  
  
    println("Student Name: ${stu.name}")  
    println("Student Age: ${stu.age}")  
}  
  
class Student(val name: String, var age: Int) {  
    //This is my class. For now I am leaving it empty  
}
```

Output:

Student Name: Tamsel
Student Age: 31

Default value in Kotlin constructor

We can also specify the default values in the Kotlin constructor like we did in

the following example. Here we have specified the default student name as “Student” and default student age is 99.

We have created three objects of this class, one with both name and age, second object with only name and third object without name and age. As you can see in the output that default values are overridden by the values passed while creating objects of the class.

```
fun main(args: Array<String>) {  
  
    //creating the object of class Student  
    val stu = Student("Tamsel", 31)  
    val stu2 = Student("Tamsel")  
    val stu3 = Student()  
  
    println("Name: ${stu.name} and Age: ${stu.age}")  
    println("Name: ${stu2.name} and Age: ${stu2.age}")  
    println("Name: ${stu3.name} and Age: ${stu3.age}")  
  
}  
  
class Student(val name: String = "Student", var age: Int = 99) {  
    //This is my class. For now I am leaving it empty  
}
```

OUTPUT

```
Name: Tamsel and Age: 31  
Name: Tamsel and Age: 99  
Name: Student and Age: 99
```

2. Kotlin Secondary Constructor

Secondary constructor in Kotlin is created using the `constructor` keyword. They play major role in inheritance. I recommend you to read the inheritance topic first before going through the secondary constructor.

Syntax of secondary constructor

```
class Student {  
    constructor(name: String) {
```

```

        // code inside constructor
    }
    constructor(name: String, age: Int) {
        // code inside constructor
    }
}

```

Example of secondary constructor

This is a simple example of how a secondary constructor is declared inside a class.

```

fun main(args: Array<String>){
    val obj = Student ("Ajeet", 30)
}

class Student{
    constructor(name: String, age: Int){
        println("Student Name: ${name.toUpperCase()}")
        println("Student Age: $age")
    }
}

```

Output:

```

Student Name: AJEET
Student Age: 30

```

Calling one secondary constructor from another

In this example we have two secondary constructors, one with one parameter and other with two parameters. We are calling a constructor from another constructor using this keyword .

```

fun main(args: Array<String>){
    val obj = Student ("Ajeet")
}

class Student{

```

```

    constructor(name: String): this(name, 0){
        println("secondary constructor with one param")
    }
    constructor(name: String, age: Int){
        println("secondary constructor with two param")
        println("Student Name: ${name.toUpperCase()}")
        println("Student Age: $age")
    }
}

```

Kotlin Secondary Constructor example with parent and child class

In the following example we have two classes `College` which is a parent class and a child class `Student`. Here the child class secondary constructor is calling the parent class secondary constructor using the `super` keyword.

```

fun main(args: Array<String>){
    val stu = Student("Harry", 24)
}

open class College{

    constructor(name: String, age: Int){
        println("parent class constructor")
        println("Student Name: ${name.toUpperCase()}")
        println("Student Age: $age")
    }
}

class Student: College{
    constructor(name: String, age: Int): super(name,age){
        println("child class constructor")
        println("Student Name: $name")
        println("Student Age: $age")
    }
}

```

Output:

parent **class** constructor

Student Name: HARRY

Student Age: 24

child **class** constructor

Student Name: Harry

Student Age: 24

HADOOP FOR BEGINNERS

**LEARN TO CODE FAST
BY
TAM SEL**

HADOOP

Hadoop tutorial provides basic and advanced concepts of Hadoop. Our Hadoop tutorial is designed for beginners and professionals.

Hadoop is an open source framework. It is provided by Apache to process and analyze very huge volume of data. It is written in Java and currently used by Google, Facebook, LinkedIn, Yahoo, Twitter etc.

Our Hadoop tutorial includes all topics of Big Data Hadoop with HDFS, MapReduce, Yarn, Hive, HBase, Pig, Sqoop etc.

What is Big Data

Data which are very large in size is called Big Data. Normally we work on data of size MB(WordDoc ,Excel) or maximum GB(Movies, Codes) but data in Peta bytes i.e. 10^{15} byte size is called Big Data. It is stated that almost 90% of today's data has been generated in the past 3 years.

Sources of Big Data

These data come from many sources like

- **Social networking sites:** Facebook, Google, LinkedIn all these sites generates huge amount of data on a day to day basis as they have billions of users worldwide.
- **E-commerce site:** Sites like Amazon, Flipkart, Alibaba generates huge amount of logs from which users buying trends can be traced.
- **Weather Station:** All the weather station and satellite gives very huge data which are stored and manipulated to forecast weather.
- **Telecom company:** Telecom giants like Airtel, Vodafone study the user trends and accordingly publish their plans and for this they store the data of its million users.
- **Share Market:** Stock exchange across the world generates huge amount of data through its daily transaction.

3V's of Big Data

1. **Velocity:** The data is increasing at a very fast rate. It is estimated that the volume of data will double in every 2 years.
2. **Variety:** Now a days data are not stored in rows and column. Data is structured as well as unstructured. Log file, CCTV footage is unstructured data. Data which can be saved in tables are structured data like the transaction data of the bank.
3. **Volume:** The amount of data which we deal with is of very large size of Peta bytes.

Use case

An e-commerce site XYZ (having 100 million users) wants to offer a gift voucher of 100\$ to its top 10 customers who have spent the most in the previous year. Moreover, they want to find the buying trend of these customers so that company can suggest more items related to them.

Issues

Huge amount of unstructured data which needs to be stored, processed and analyzed.

Solution

Storage: This huge amount of data, Hadoop uses HDFS (Hadoop Distributed File System) which uses commodity hardware to form clusters and store data in a distributed fashion. It works on Write once, read many times principle.

Processing: Map Reduce paradigm is applied to data distributed over network to find the required output.

Analyze: Pig, Hive can be used to analyze the data.

Cost: Hadoop is open source so the cost is no more an issue.

What is Hadoop

Hadoop is an open source framework from Apache and is used to store process and analyze data which are very huge in volume. Hadoop is written in Java and is not OLAP (online analytical processing). It is used for batch/offline processing. It is being used by Facebook, Yahoo, Google, Twitter, LinkedIn and many more. Moreover it can be scaled up just by adding nodes in the cluster.

Modules of Hadoop

1. **HDFS:** Hadoop Distributed File System. Google published its paper GFS and on the basis of that HDFS was developed. It states that the files will be broken into blocks and stored in nodes over the distributed architecture.
2. **Yarn:** Yet another Resource Negotiator is used for job scheduling and manage the cluster.
3. **Map Reduce:** This is a framework which helps Java programs to do the parallel computation on data using key value pair. The Map task takes input data and converts it into a data set which can be computed in Key value pair. The output of Map task is consumed by reduce task and then the out of reducer gives the desired result.
4. **Hadoop Common:** These Java libraries are used to start Hadoop and are used by other Hadoop modules.

Hadoop Architecture

The Hadoop architecture is a package of the file system, MapReduce engine and the HDFS (Hadoop Distributed File System). The MapReduce engine can be MapReduce/MR1 or YARN/MR2.

A Hadoop cluster consists of a single master and multiple slave nodes. The master node includes Job Tracker, Task Tracker, NameNode, and DataNode whereas the slave node includes DataNode and TaskTracker.

Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) is a distributed file system for Hadoop. It contains a master/slave architecture. This architecture consist of a single NameNode performs the role of master, and multiple DataNodes performs the role of a slave.

Both NameNode and DataNode are capable enough to run on commodity machines. The Java language is used to develop HDFS. So any machine that supports Java language can easily run the NameNode and DataNode software.

NameNode

- It is a single master server exist in the HDFS cluster.
- As it is a single node, it may become the reason of single point failure.
- It manages the file system namespace by executing an operation like the opening, renaming and closing the files.
- It simplifies the architecture of the system.

DataNode

- The HDFS cluster contains multiple DataNodes.
- Each DataNode contains multiple data blocks.
- These data blocks are used to store data.
- It is the responsibility of DataNode to read and write requests from the file system's clients.
- It performs block creation, deletion, and replication upon instruction from the NameNode.

Job Tracker

- The role of Job Tracker is to accept the MapReduce jobs from client and process the data by using NameNode.
- In response, NameNode provides metadata to Job Tracker.

Task Tracker

- It works as a slave node for Job Tracker.
- It receives task and code from Job Tracker and applies that code on the file. This process can also be called as a Mapper.

MapReduce Layer

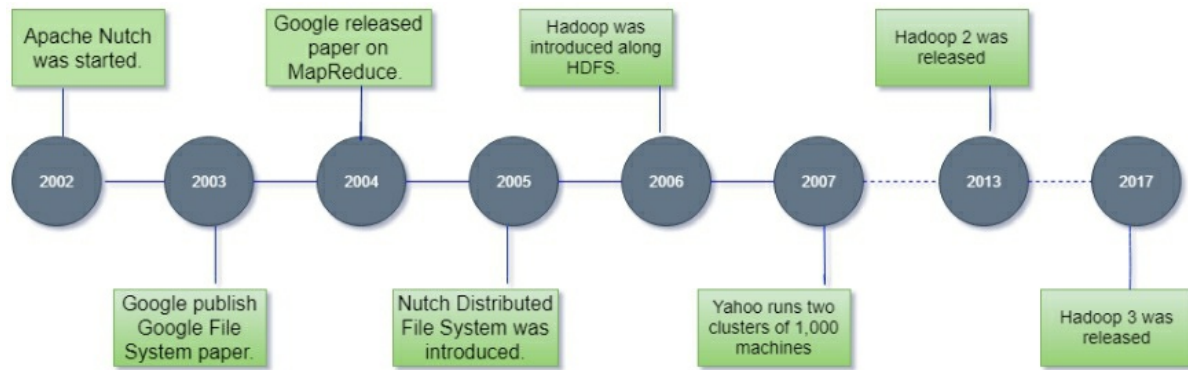
The MapReduce comes into existence when the client application submits the MapReduce job to Job Tracker. In response, the Job Tracker sends the request to the appropriate Task Trackers. Sometimes, the TaskTracker fails or time out. In such a case, that part of the job is rescheduled.

Advantages of Hadoop

- **Fast:** In HDFS the data distributed over the cluster and are mapped which helps in faster retrieval. Even the tools to process the data are often on the same servers, thus reducing the processing time. It is able to process terabytes of data in minutes and Peta bytes in hours.
- **Scalable:** Hadoop cluster can be extended by just adding nodes in the cluster.
- **Cost Effective:** Hadoop is open source and uses commodity hardware to store data so it really cost effective as compared to traditional relational database management system.
- **Resilient to failure:** HDFS has the property with which it can replicate data over the network, so if one node is down or some other network failure happens, then Hadoop takes the other copy of data and use it. Normally, data are replicated thrice but the replication factor is configurable.

History of Hadoop

The Hadoop was started by Doug Cutting and Mike Cafarella in 2002. Its origin was the Google File System paper, published by Google.



Let's focus on the history of Hadoop in the following steps: -

- In 2002, Doug Cutting and Mike Cafarella started to work on a project, **Apache Nutch**. It is an open source web crawler software project.
- While working on Apache Nutch, they were dealing with big data. To store that data they have to spend a lot of costs which becomes the consequence of that project. This problem becomes one of the important reason for the emergence of Hadoop.
- In 2003, Google introduced a file system known as GFS (Google file system). It is a proprietary distributed file system developed to provide efficient access to data.
- In 2004, Google released a white paper on Map Reduce. This technique simplifies the data processing on large clusters.
- In 2005, Doug Cutting and Mike Cafarella introduced a new file system known as NDFS (Nutch Distributed File System). This file system also includes Map reduce.
- In 2006, Doug Cutting quit Google and joined Yahoo. On the basis of the Nutch project, Dough Cutting introduces a new project Hadoop with a file system known as HDFS (Hadoop Distributed File System). Hadoop first version 0.1.0 released in this year.
- Doug Cutting gave named his project Hadoop after his son's toy elephant.
- In 2007, Yahoo runs two clusters of 1000 machines.
- In 2008, Hadoop became the fastest system to sort 1 terabyte of data on a 900 node cluster within 209 seconds.
- In 2013, Hadoop 2.2 was released.

- In 2017, Hadoop 3.0 was released.

Hadoop Installation

Environment required for Hadoop: The production environment of Hadoop is UNIX, but it can also be used in Windows using Cygwin. Java 1.6 or above is needed to run Map Reduce Programs. For Hadoop installation from tar ball on the UNIX environment you need

1. Java Installation
2. SSH installation
3. Hadoop Installation and File Configuration

1) Java Installation

Step 1. Type "java -version" in prompt to find if the java is installed or not. If not then download java from

<http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html> . The tar file `jdk-7u71-linux-x64.tar.gz` will be downloaded to your system.

Step 2. Extract the file using the below command

1. `#tar xzf jdk-7u71-linux-x64.tar.gz`

Step 3. To make java available for all the users of UNIX move the file to `/usr/local` and set the path. In the prompt switch to root user and then type the command below to move the jdk to `/usr/lib`.

1. `# mv jdk1.7.0_71 /usr/lib/`

Now in `~/.bashrc` file add the following commands to set up the path.

1. `# export JAVA_HOME=/usr/lib/jdk1.7.0_71`
2. `# export PATH=PATH:$JAVA_HOME/bin`

Now, you can check the installation by typing "java -version" in the prompt.

2) SSH Installation

SSH is used to interact with the master and slaves computer without any prompt for password. First of all create a Hadoop user on the master and slave systems

1. # useradd hadoop
2. # passwd Hadoop

To map the nodes open the hosts file present in /etc/ folder on all the machines and put the ip address along with their host name.

1. # vi /etc/hosts

Enter the lines below

1. 190.12.1.114 hadoop-master
2. 190.12.1.121 hadoop-salve-one
3. 190.12.1.143 hadoop-slave-two

Set up SSH key in every node so that they can communicate among themselves without password. Commands for the same are:

1. # su hadoop
2. \$ ssh-keygen -t rsa
3. \$ ssh-copy-id -i ~/.ssh/id_rsa.pub tutorialspoint@hadoop-master
4. \$ ssh-copy-id -i ~/.ssh/id_rsa.pub hadoop_tp1@hadoop-slave-1
5. \$ ssh-copy-id -i ~/.ssh/id_rsa.pub hadoop_tp2@hadoop-slave-2
6. \$ chmod 0600 ~/.ssh/authorized_keys
7. \$ exit

3) Hadoop Installation

Hadoop can be downloaded from
<http://developer.yahoo.com/hadoop/tutorial/module3.html>

Now extract the Hadoop and copy it to a location.

1. \$ mkdir /usr/hadoop
2. \$ sudo tar vxzf hadoop-2.2.0.tar.gz ?c /usr/hadoop

Change the ownership of Hadoop folder

1. `$sudo chown -R hadoop usr/hadoop`

Change the Hadoop configuration files:

All the files are present in `/usr/local/Hadoop/etc/hadoop`

1) In `hadoop-env.sh` file add

1. `export JAVA_HOME=/usr/lib/jvm/jdk/jdk1.7.0_71`

2) In `core-site.xml` add following between configuration tabs,

1. `<configuration>`
2. `<property>`
3. `<name>fs.default.name</name>`
4. `<value>hdfs://hadoop-master:9000</value>`
5. `</property>`
6. `<property>`
7. `<name>dfs.permissions</name>`
8. `<value>>false</value>`
9. `</property>`
10. `</configuration>`

3) In `hdfs-site.xml` add following between configuration tabs,

1. `<configuration>`
2. `<property>`
3. `<name>dfs.data.dir</name>`
4. `<value>usr/hadoop/dfs/name/data</value>`
5. `<final>true</final>`
6. `</property>`
7. `<property>`
8. `<name>dfs.name.dir</name>`
9. `<value>usr/hadoop/dfs/name</value>`
10. `<final>true</final>`
11. `</property>`

```
12.      <property>
13.      <name>dfs.replication</name>
14.      <value>1</value>
15.      </property>
16.      </configuration>
```

4) Open the Mapred-site.xml and make the change as shown below

```
1. <configuration>
2. <property>
3. <name>mapred.job.tracker</name>
4. <value>hadoop-master:9001</value>
5. </property>
6. </configuration>
```

5) Finally, update your \$HOME/.bashrc

```
1. cd $HOME
2. vi .bashrc
3. Append following lines in the end and save and exit
4. #Hadoop variables
5. export JAVA_HOME=/usr/lib/jvm/jdk/jdk1.7.0_71
6. export HADOOP_INSTALL=/usr/hadoop
7. export PATH=$PATH:$HADOOP_INSTALL/bin
8. export PATH=$PATH:$HADOOP_INSTALL/sbin
9. export HADOOP_MAPRED_HOME=$HADOOP_INSTALL
10.      export
    HADOOP_COMMON_HOME=$HADOOP_INSTALL
11.      export
    HADOOP_HDFS_HOME=$HADOOP_INSTALL
12.      export YARN_HOME=$HADOOP_INSTALL
```

On the slave machine install Hadoop using the command below

```
1. # su hadoop
2. $ cd /opt/hadoop
3. $ scp -r hadoop hadoop-slave-one:/usr/hadoop
```

4. `$ scp -r hadoop hadoop-slave-two:/usr/Hadoop`

Configure master node and slave node

1. `$ vi etc/hadoop/masters`
2. `hadoop-master`
- 3.
4. `$ vi etc/hadoop/slaves`
5. `hadoop-slave-one`
6. `hadoop-slave-two`

After this format the name node and start all the deamons

1. `# su hadoop`
2. `$ cd /usr/hadoop`
3. `$ bin/hadoop namenode -format`
- 4.
5. `$ cd $HADOOP_HOME/sbin`
6. `$ start-all.sh`

The easiest step is the usage of cloudera as it comes with all the stuffs pre-installed which can be downloaded from <http://content.udacity-data.com/courses/ud617/Cloudera-Udacity-Training-VM-4.1.1.c.zip>

HADOOP MODULES

What is HDFS

Hadoop comes with a distributed file system called HDFS. In HDFS data is distributed over several machines and replicated to ensure their durability to failure and high availability to parallel application.

It is cost effective as it uses commodity hardware. It involves the concept of blocks, data nodes and node name.

Where to use HDFS

- **Very Large Files:** Files should be of hundreds of megabytes, gigabytes or more.
- **Streaming Data Access:** The time to read whole data set is more important than latency in reading the first. HDFS is built on write-once and read-many-times pattern.
- **Commodity Hardware:** It works on low cost hardware.

Where not to use HDFS

- **Low Latency data access:** Applications that require very less time to access the first data should not use HDFS as it is giving importance to whole data rather than time to fetch the first record.
- **Lots Of Small Files:** The name node contains the metadata of files in memory and if the files are small in size it takes a lot of memory for name node's memory which is not feasible.
- **Multiple Writes:** It should not be used when we have to write multiple times.

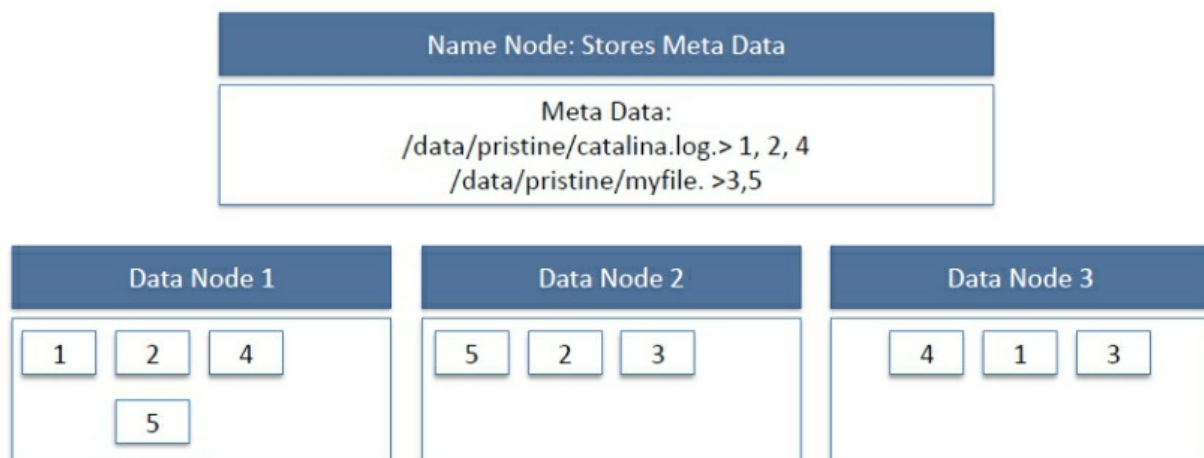
HDFS Concepts

1. **Blocks:** A Block is the minimum amount of data that it can read or write. HDFS blocks are 128 MB by default and this is configurable. Files in HDFS are broken into block-sized

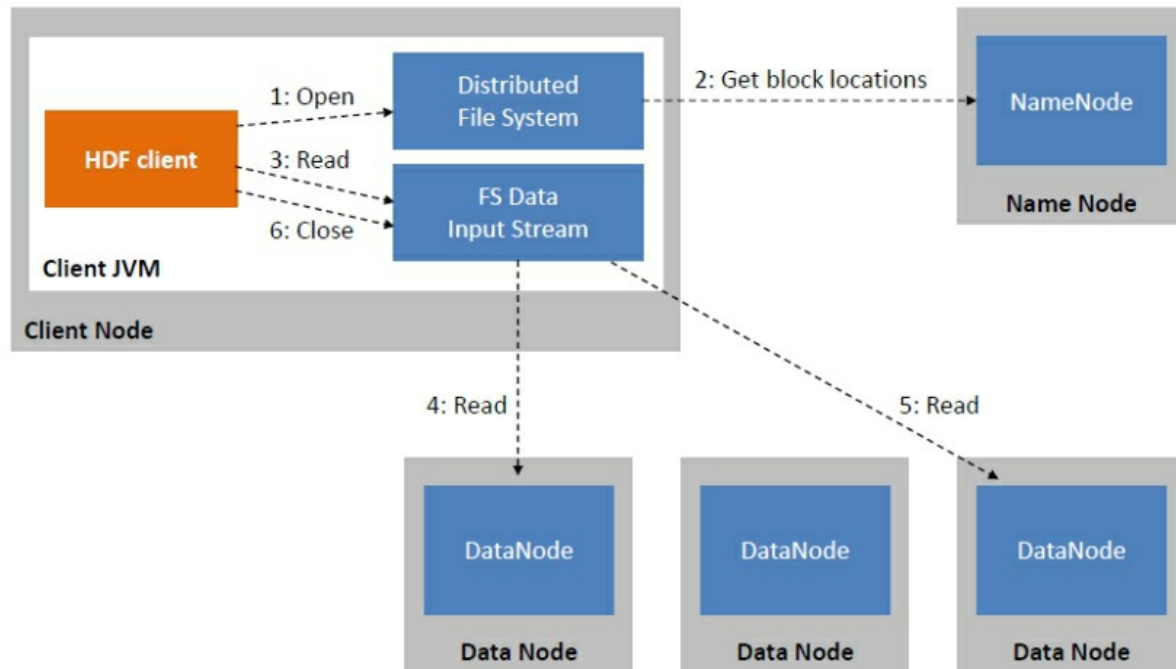
chunks, which are stored as independent units. Unlike a file system, if the file in HDFS is smaller than block size, then it does not occupy full block's size, i.e. 5 MB of file stored in HDFS of block size 128 MB takes 5MB of space only. The HDFS block size is large just to minimize the cost of seek.

2. **Name Node:** HDFS works in master-worker pattern where the name node acts as master. Name Node is controller and manager of HDFS as it knows the status and the metadata of all the files in HDFS; the metadata information being file permission, names and location of each block. The metadata are small, so it is stored in the memory of name node, allowing faster access to data. Moreover the HDFS cluster is accessed by multiple clients concurrently, so all this information is handled by a single machine. The file system operations like opening, closing, renaming etc. are executed by it.
3. **Data Node:** They store and retrieve blocks when they are told to; by client or name node. They report back to name node periodically, with list of blocks that they are storing. The data node being a commodity hardware also does the work of block creation, deletion and replication as stated by the name node.

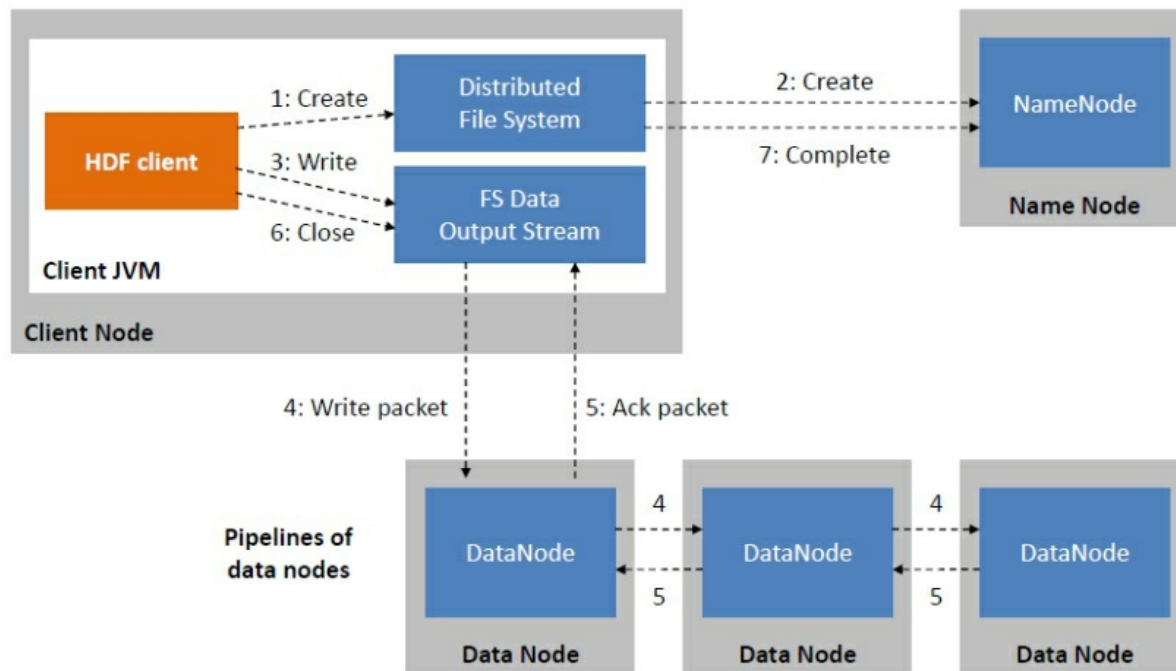
HDFS DataNode and NameNode Image:



HDFS READ IMAGE



HDFS Write Image



Since all the metadata is stored in name node, it is very important. If it fails the file system can not be used as there would be no way of knowing how to reconstruct the files from blocks present in data node. To overcome this, the

concept of secondary name node arises.

Secondary Name Node: It is a separate physical machine which acts as a helper of name node. It performs periodic check points. It communicates with the name node and take snapshot of meta data which helps minimize downtime and loss of data.

Starting HDFS

The HDFS should be formatted initially and then started in the distributed mode. Commands are given below.

To Format \$ **hadoop namenode -format**

To Start \$ **start-dfs.sh**

HDFS Basic File Operations

1. Putting data to HDFS from local file system
 - First create a folder in HDFS where data can be put from local file system.

\$ **hadoop fs -mkdir /user/test**
 - Copy the file "data.txt" from a file kept in local folder /usr/home/Desktop to HDFS folder /user/ test

\$ **hadoop fs -copyFromLocal /usr/home/Desktop/data.txt /user/test**
 - Display the content of HDFS folder

\$ **Hadoop fs -ls /user/test**
2. Copying data from HDFS to local file system
 - \$ **hadoop fs -copyToLocal /user/test/data.txt /usr/bin/data_copy.txt**
3. Compare the files and see that both are same
 - \$ **md5 /usr/bin/data_copy.txt /usr/home/Desktop/data.txt**

Recursive deleting

- `hadoop fs -rmr <arg>`

Example:

- `hadoop fs -rmr /user/sonoo/`

HDFS Other commands

The below is used in the commands

"<path>" means any file or directory name.

"<path>..." means one or more file or directory names.

"<file>" means any filename.

"<src>" and "<dest>" are path names in a directed operation.

"<localSrc>" and "<localDest>" are paths as above, but on the local file system

- `put <localSrc><dest>`

Copies the file or directory from the local file system identified by localSrc to dest within the DFS.

- `copyFromLocal <localSrc><dest>`

Identical to -put

- `copyFromLocal <localSrc><dest>`

Identical to -put

- `moveFromLocal <localSrc><dest>`

Copies the file or directory from the local file system identified by localSrc to dest within HDFS, and then deletes the local copy on success.

- `get [-crc] <src><localDest>`

Copies the file or directory in HDFS identified by src to the local file system path identified by localDest.

- `cat <filename>`

Displays the contents of filename on stdout.

- `moveToLocal <src><localDest>`

Works like `-get`, but deletes the HDFS copy on success.

- `setrep [-R] [-w] rep <path>`

Sets the target replication factor for files identified by path to rep. (The actual replication factor will move toward the target over time)

- `touchz <path>`

Creates a file at path containing the current time as a timestamp. Fails if a file already exists at path, unless the file is already size 0.

- `test -[ezd] <path>`

Returns 1 if path exists; has zero length; or is a directory or 0 otherwise.

- `stat [format] <path>`

Prints information about path. Format is a string which accepts file size in blocks (%b), filename (%n), block size (%o), replication (%r), and modification date (%y, %Y).

HDFS Features and Goals

The Hadoop Distributed File System (HDFS) is a distributed file system. It is a core part of Hadoop which is used for data storage. It is designed to run on commodity hardware.

Unlike other distributed file system, HDFS is highly fault-tolerant and can be deployed on low-cost hardware. It can easily handle the application that contains large data sets.

Let's see some of the important features and goals of HDFS.

Features of HDFS

- **Highly Scalable** - HDFS is highly scalable as it can scale hundreds of nodes in a single cluster.
- **Replication** - Due to some unfavorable conditions, the node containing the data may be loss. So, to overcome such problems, HDFS always maintains the copy of data on a different machine.
- **Fault tolerance** - In HDFS, the fault tolerance signifies the robustness of the system in the event of failure. The HDFS is highly fault-tolerant that if any machine fails, the other machine containing the copy of that data automatically become active.
- **Distributed data storage** - This is one of the most important features of HDFS that makes Hadoop very powerful. Here, data is divided into multiple blocks and stored into nodes.
- **Portable** - HDFS is designed in such a way that it can easily portable from platform to another.

Goals of HDFS

- **Handling the hardware failure** - The HDFS contains multiple server machines. Anyhow, if any machine fails, the HDFS goal is to recover it quickly.
- **Streaming data access** - The HDFS applications usually run on the general-purpose file system. This application requires streaming access to their data sets.

- **Coherence Model** - The application that runs on HDFS require to follow the write-once-ready-many approach. So, a file once created need not to be changed. However, it can be appended and truncate.

What is YARN

Yet Another Resource Manager takes programming to the next level beyond Java , and makes it interactive to let another application Hbase, Spark etc. to work on it. Different Yarn applications can co-exist on the same cluster so MapReduce, Hbase, Spark all can run at the same time bringing great benefits for manageability and cluster utilization.

Components Of YARN

- **Client:** For submitting MapReduce jobs.
- **Resource Manager:** To manage the use of resources across the cluster
- **Node Manager:** For launching and monitoring the computer containers on machines in the cluster.
- **Map Reduce Application Master:** Checks tasks running the MapReduce job. The application master and the MapReduce tasks run in containers that are scheduled by the resource manager, and managed by the node managers.

Jobtracker & Tasktracker were used in previous version of Hadoop, which were responsible for handling resources and checking progress management. However, Hadoop 2.0 has Resource manager and NodeManager to overcome the shortfall of Jobtracker & Tasktracker.

Benefits of YARN

- **Scalability:** Map Reduce 1 hits scalability bottleneck at 4000 nodes and 40000 task, but Yarn is designed for 10,000 nodes and 1 lakh tasks.
- **Utilization:** Node Manager manages a pool of resources, rather than a fixed number of the designated slots thus increasing the utilization.
- **Multitenancy:** Different version of MapReduce can run on YARN, which makes the process of upgrading MapReduce more manageable.

HADOOP MapReduce

MapReduce tutorial provides basic and advanced concepts of MapReduce. Our MapReduce tutorial is designed for beginners and professionals.

Our MapReduce tutorial includes all topics of MapReduce such as Data Flow in MapReduce, Map Reduce API, Word Count Example, Character Count Example, etc.

What is MapReduce?

A MapReduce is a data processing tool which is used to process the data parallelly in a distributed form. It was developed in 2004, on the basis of paper titled as "MapReduce: Simplified Data Processing on Large Clusters," published by Google.

The MapReduce is a paradigm which has two phases, the mapper phase, and the reducer phase. In the Mapper, the input is given in the form of a key-value pair. The output of the Mapper is fed to the reducer as input. The reducer runs only after the Mapper is over. The reducer too takes input in key-value format, and the output of reducer is the final output.

Steps in Map Reduce

- The map takes data in the form of pairs and returns a list of <key, value> pairs. The keys will not be unique in this case.
- Using the output of Map, sort and shuffle are applied by the Hadoop architecture. This sort and shuffle acts on these list of <key, value> pairs and sends out unique keys and a list of values associated with this unique key <key, list(values)>.
- An output of sort and shuffle sent to the reducer phase. The reducer performs a defined function on a list of values for unique keys, and Final output <key, value> will be stored/displayed.

Sort and Shuffle

The sort and shuffle occur on the output of Mapper and before the reducer.

When the Mapper task is complete, the results are sorted by key, partitioned if there are multiple reducers, and then written to disk. Using the input from each Mapper $\langle k_2, v_2 \rangle$, we collect all the values for each unique key k_2 . This output from the shuffle phase in the form of $\langle k_2, \text{list}(v_2) \rangle$ is sent as input to reducer phase.

Usage of MapReduce

- It can be used in various application like document clustering, distributed sorting, and web link-graph reversal.
- It can be used for distributed pattern-based searching.
- We can also use MapReduce in machine learning.
- It was used by Google to regenerate Google's index of the World Wide Web.
- It can be used in multiple computing environments such as multi-cluster, multi-core, and mobile environment.

Prerequisite

Before learning MapReduce, you must have the basic knowledge of Big Data.

Audience

Our MapReduce tutorial is designed to help beginners and professionals.

Problem

We assure that you will not find any problem in this MapReduce tutorial. But if there is any mistake, please post the problem in contact form.

Data Flow In MapReduce

MapReduce is used to compute the huge amount of data . To handle the upcoming data in a parallel and distributed form, the data has to flow from various phases.

Phases of MapReduce data flow

Input reader

The input reader reads the upcoming data and splits it into the data blocks of the appropriate size (64 MB to 128 MB). Each data block is associated with a Map function.

Once input reads the data, it generates the corresponding key-value pairs. The input files reside in HDFS.

Map function

The map function process the upcoming key-value pairs and generated the corresponding output key-value pairs. The map input and output type may be different from each other.

Partition function

The partition function assigns the output of each Map function to the appropriate reducer. The available key and value provide this function. It returns the index of reducers.

Shuffling and Sorting

The data are shuffled between/within nodes so that it moves out from the map and get ready to process for reduce function. Sometimes, the shuffling of data can take much computation time.

The sorting operation is performed on input data for Reduce function. Here, the data is compared using comparison function and arranged in a sorted form.

Reduce function

The Reduce function is assigned to each unique key. These keys are already arranged in sorted order. The values associated with the keys can iterate the Reduce and generates the corresponding output.

Output writer

Once the data flow from all the above phases, Output writer executes. The role of Output writer is to write the Reduce output to the stable storage.

MapReduce API

In this section, we focus on MapReduce APIs. Here, we learn about the classes and methods used in MapReduce programming.

MapReduce Mapper Class

In MapReduce, the role of the Mapper class is to map the input key-value pairs to a set of intermediate key-value pairs. It transforms the input records into intermediate records.

These intermediate records associated with a given output key and passed to Reducer for the final output.

Methods of Mapper Class

`void cleanup(Context context)`-This method called only once at the end of the task.

`void map(KEYIN key, VALUEIN value, Context context)`-This method can be called only once for each key-value in the input split.

`void run(Context context)`-This method can be override to control the execution of the Mapper.

`void setup(Context context)`-This method called only once at the beginning of the task.

MapReduce Reducer Class

In MapReduce, the role of the Reducer class is to reduce the set of intermediate values. Its implementations can access the Configuration for the job via the `JobContext.getConfiguration()` method.

Methods of Reducer Class

`void cleanup(Context context)`-This method called only once at the end of the task.

`void map(KEYIN key, Iterable<VALUEIN> values, Context context)`-
This method called only once for each key.

void run(Context context)-This method can be used to control the tasks of the Reducer.

void setup(Context context)-This method called only once at the beginning of the task.

MapReduce Job Class

The Job class is used to configure the job and submits it. It also controls the execution and query the state. Once the job is submitted, the set method throws IllegalStateException.

Methods of Job Class

Methods-Description

Counters getCounters()-This method is used to get the counters for the job.

long getFinishTime()-This method is used to get the finish time for the job.

Job getInstance()-This method is used to generate a new Job without any cluster.

Job getInstance(Configuration conf)-This method is used to generate a new Job without any cluster and provided configuration.

Job getInstance(Configuration conf, String jobName)-This method is used to generate a new Job without any cluster and provided configuration and job name.

String getJobFile()-This method is used to get the path of the submitted job configuration.

String getJobName()-This method is used to get the user-specified job name.

JobPriority getPriority()-This method is used to get the scheduling function of the job.

void setJarByClass(Class<?> c)-This method is used to set the jar by providing the class name with .class extension.

void setJobName(String name)-This method is used to set the user-specified job name.

void setMapOutputKeyClass(Class<?> class)-This method is used to set the key class for the map output data.

`void setMapOutputValueClass(Class<?> class)`-This method is used to set the value class for the map output data.

`void setMapperClass(Class<? extends Mapper> class)`-This method is used to set the Mapper for the job.

`void setNumReduceTasks(int tasks)`-This method is used to set the number of reduce tasks for the job

`void setReducerClass(Class<? extends Reducer> class)`-This method is used to set the Reducer for the job.

MapReduce Word Count Example

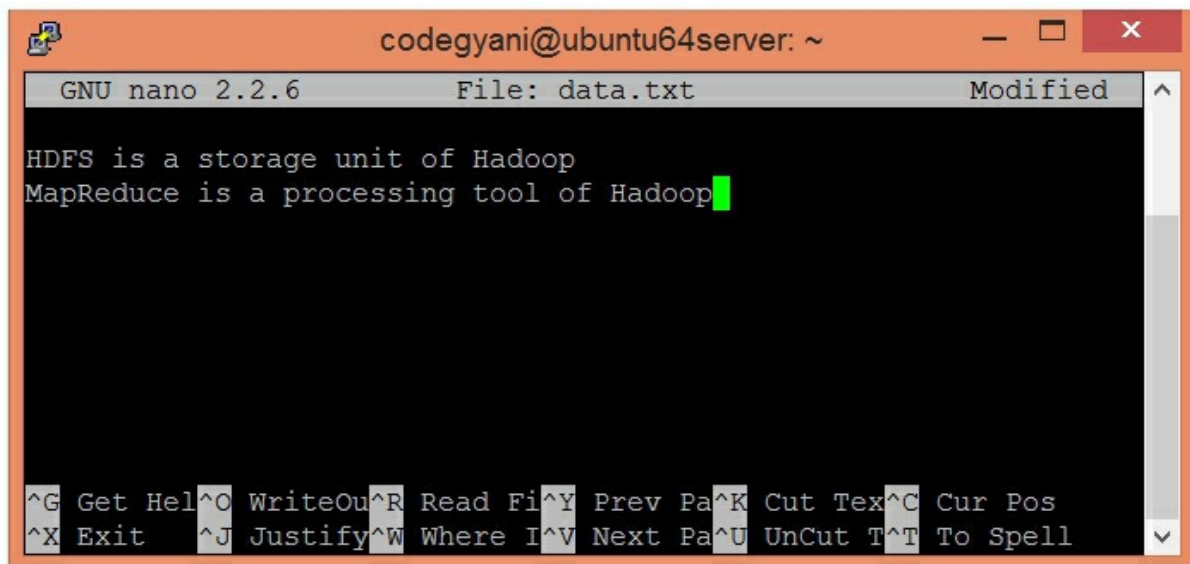
In MapReduce word count example, we find out the frequency of each word. Here, the role of Mapper is to map the keys to the existing values and the role of Reducer is to aggregate the keys of common values. So, everything is represented in the form of Key-value pair.

Pre-requisite

- **Java Installation** - Check whether the Java is installed or not using the following command.
`java -version`
- **Hadoop Installation** - Check whether the Hadoop is installed or not using the following command.
`hadoop version`

Steps to execute MapReduce word count example

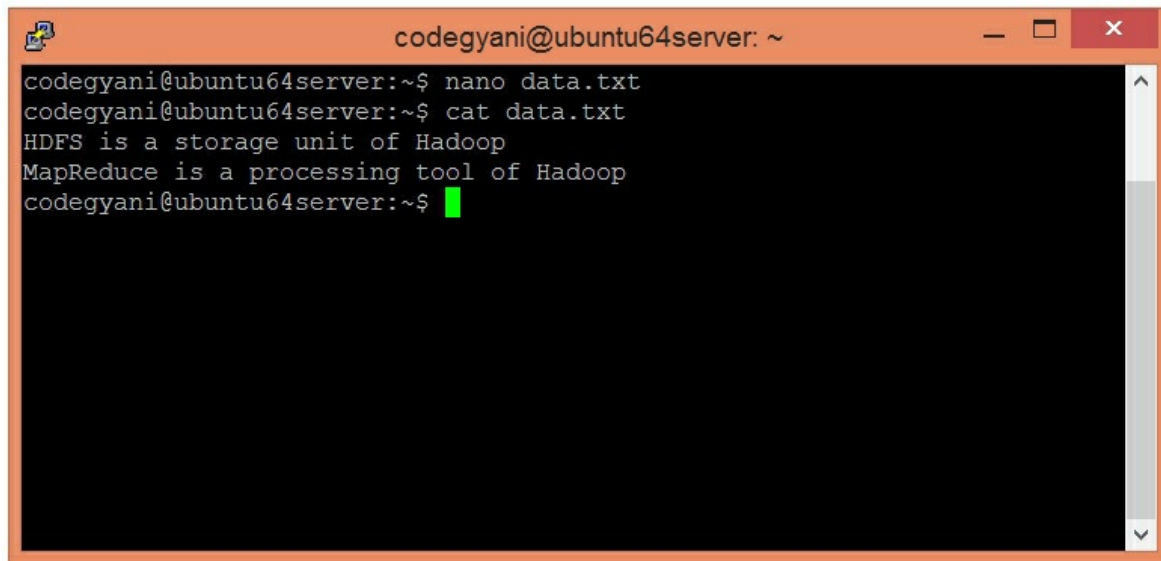
- Create a text file in your local machine and write some text into it.
`$ nano data.txt`



```
codegyani@ubuntu64server: ~
GNU nano 2.2.6      File: data.txt      Modified
HDFS is a storage unit of Hadoop
MapReduce is a processing tool of Hadoop
^G Get Hel^O WriteOu^R Read Fi^Y Prev Pa^K Cut Tex^C Cur Pos
^X Exit      ^J Justify^W Where I^V Next Pa^U UnCut T^T To Spell
```

Check the text written in the data.txt file.

\$ cat data.txt

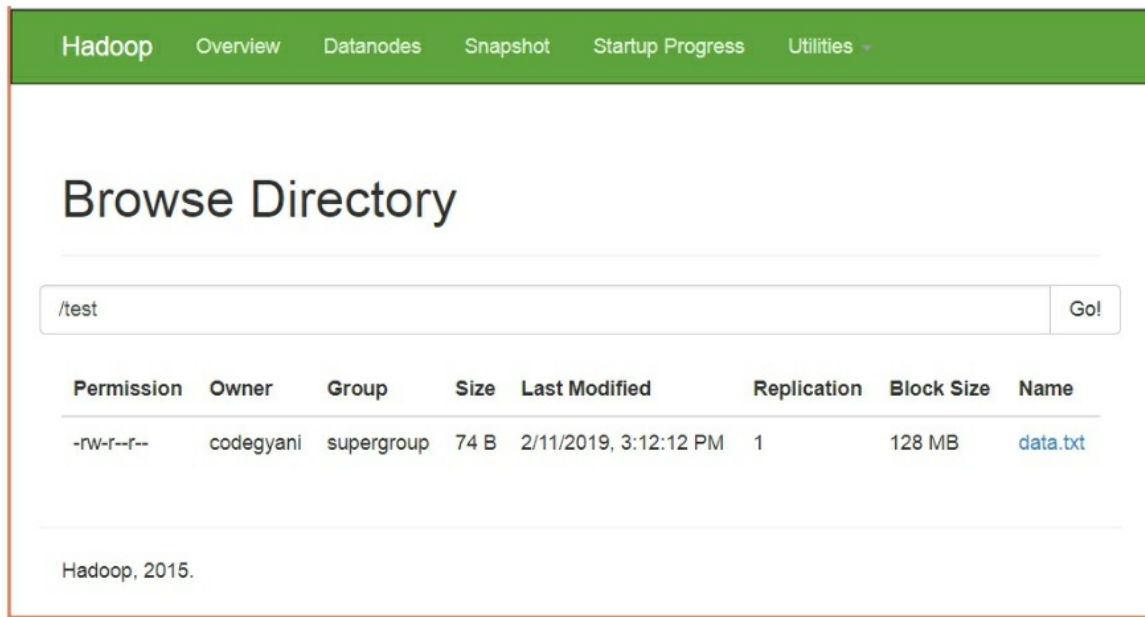
A terminal window titled 'codegyani@ubuntu64server: ~' with standard window controls. The terminal shows the following commands and output:

```
codegyani@ubuntu64server:~$ nano data.txt
codegyani@ubuntu64server:~$ cat data.txt
HDFS is a storage unit of Hadoop
MapReduce is a processing tool of Hadoop
codegyani@ubuntu64server:~$
```

A green cursor is visible at the end of the last command line.

In this example, we find out the frequency of each word exists in this text file.

- Create a directory in HDFS, where to kept text file.
\$ hdfs dfs -mkdir /test
- Upload the data.txt file on HDFS in the specific directory.
\$ hdfs dfs -put /home/codegyani/data.txt /test



- Write the MapReduce program using eclipse.

File: WC_Mapper.java

```
1. package com.javatpoint;
2.
3. import java.io.IOException;
4. import java.util.StringTokenizer;
5. import org.apache.hadoop.io.IntWritable;
6. import org.apache.hadoop.io.LongWritable;
7. import org.apache.hadoop.io.Text;
8. import org.apache.hadoop.mapred.MapReduceBase;
9. import org.apache.hadoop.mapred.Mapper;
10.     import org.apache.hadoop.mapred.OutputCollector;
11.     import org.apache.hadoop.mapred.Reporter;
12.     public class WC_Mapper extends MapReduceBase
13. implements Mapper<LongWritable,Text,Text,IntWritable>{
14.     private final static IntWritable one = new
15. IntWritable(1);
16.     private Text word = new Text();
17.     public void map(LongWritable key, Text
18. value,OutputCollector<Text,IntWritable> output,
```

```

16.         Reporter reporter) throws IOException{
17.         String line = value.toString();
18.         StringTokenizer tokenizer = new
StringTokenizer(line);
19.         while (tokenizer.hasMoreTokens()){
20.             word.set(tokenizer.nextToken());
21.             output.collect(word, one);
22.         }
23.     }
24.
25. }

```

File: WC_Reducer.java

```

1. package com.javatpoint;
2.   import java.io.IOException;
3.   import java.util.Iterator;
4.   import org.apache.hadoop.io.IntWritable;
5.   import org.apache.hadoop.io.Text;
6.   import org.apache.hadoop.mapred.MapReduceBase;
7.   import org.apache.hadoop.mapred.OutputCollector;
8.   import org.apache.hadoop.mapred.Reducer;
9.   import org.apache.hadoop.mapred.Reporter;
10.
11.       public class WC_Reducer extends MapReduceBase
implements Reducer<Text,IntWritable,Text,IntWritable> {
12.       public void reduce(Text key, Iterator<IntWritable>
values,OutputCollector<Text,IntWritable> output,
13.       Reporter reporter) throws IOException {
14.           int sum=0;
15.           while (values.hasNext()) {
16.               sum+=values.next().get();
17.           }
18.           output.collect(key,new IntWritable(sum));
19.       }

```

20. }

File: WC_Runner.java

```
1. package com.javatpoint;
2.
3.     import java.io.IOException;
4.     import org.apache.hadoop.fs.Path;
5.     import org.apache.hadoop.io.IntWritable;
6.     import org.apache.hadoop.io.Text;
7.     import org.apache.hadoop.mapred.FileInputFormat;
8.     import org.apache.hadoop.mapred.FileOutputFormat;
9.     import org.apache.hadoop.mapred.JobClient;
10.         import org.apache.hadoop.mapred.JobConf;
11.         import org.apache.hadoop.mapred.TextInputFormat;
12.         import
org.apache.hadoop.mapred.TextOutputFormat;
13.         public class WC_Runner {
14.             public static void main(String[] args) throws
IOException{
15.                 JobConf conf = new
JobConf(WC_Runner.class);
16.                 conf.setJobName("WordCount");
17.                 conf.setOutputKeyClass(Text.class);
18.
conf.setOutputValueClass(IntWritable.class);
19.                 conf.setMapperClass(WC_Mapper.class);
20.                 conf.setCombinerClass(WC_Reducer.class);
21.                 conf.setReducerClass(WC_Reducer.class);
22.                 conf.setInputFormat(TextInputFormat.class);
23.
conf.setOutputFormat(TextOutputFormat.class);
24.                 FileInputFormat.setInputPaths(conf,new
Path(args[0]));
25.                 FileOutputFormat.setOutputPath(conf,new
```

```

Path(args[1]));
26.          JobClient.runJob(conf);
27.      }
28.  }

```

Download the source code.

- Create the jar file of this program and name it **countworddemo.jar**.
- Run the jar file
`hadoop jar /home/codegyani/wordcountdemo.jar com.javatpoint.WC_Runner /test/data.txt /r_output`
- The output is stored in /r_output/part-00000

Hadoop Overview Datanodes Snapshot Startup Progress Utilities							
Browse Directory							
/r_output							Go!
Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	codegyani	supergroup	0 B	2/11/2019, 3:52:27 PM	1	128 MB	_SUCCESS
-rw-r--r--	codegyani	supergroup	79 B	2/11/2019, 3:52:23 PM	1	128 MB	part-00000

Now execute the command to see the output.

```
hdfs dfs -cat /r_output/part-00000
```

```
codegyani@ubuntu64server: ~  
codegyani@ubuntu64server:~$ hdfs dfs -cat /r_output/part-00000  
HDFS      1  
Hadoop    2  
MapReduce      1  
a         2  
is        2  
of        2  
processing    1  
storage 1  
tool       1  
unit       1  
codegyani@ubuntu64server:~$
```

MapReduce Char Count Example

In MapReduce char count example, we find out the frequency of each character. Here, the role of Mapper is to map the keys to the existing values and the role of Reducer is to aggregate the keys of common values. So, everything is represented in the form of Key-value pair.

Pre-requisite

- **Java Installation** - Check whether the Java is installed or not using the following command.
`java -version`
- **Hadoop Installation** - Check whether the Hadoop is installed or not using the following command.
`hadoop version`

Steps to execute MapReduce char count example

- Create a text file in your local machine and write some text into it.
`$ nano info.txt`
- Check the text written in the info.txt file.
`$ cat info.txt`

In this example, we find out the frequency of each char value exists in this text file.

- Create a directory in HDFS, where to kept text file.
`$ hdfs dfs -mkdir /count`
- Upload the info.txt file on HDFS in the specific directory.
`$ hdfs dfs -put /home/codegyani/info.txt /count`
- Write the MapReduce program using eclipse.

File: WC_Mapper.java

```

1. package com.javatpoint;
2.
3. import java.io.IOException;
4. import org.apache.hadoop.io.IntWritable;
5. import org.apache.hadoop.io.LongWritable;
6. import org.apache.hadoop.io.Text;
7. import org.apache.hadoop.mapred.MapReduceBase;
8. import org.apache.hadoop.mapred.Mapper;
9. import org.apache.hadoop.mapred.OutputCollector;
10.     import org.apache.hadoop.mapred.Reporter;
11.     public class WC_Mapper extends MapReduceBase
implements Mapper<LongWritable,Text,Text,IntWritable>{
12.         public void map(LongWritable key, Text
value,OutputCollector<Text,IntWritable> output,
13.             Reporter reporter) throws IOException{
14.             String line = value.toString();
15.             String tokenizer[] = line.split("");
16.             for(String SingleChar : tokenizer)
17.             {
18.                 Text charKey = new Text(SingleChar);
19.                 IntWritable One = new IntWritable(1);
20.                 output.collect(charKey, One);
21.             }
22.         }
23.
24.     }

```

File: WC_Reducer.java

```

1. package com.javatpoint;
2.     import java.io.IOException;
3.     import java.util.Iterator;
4.     import org.apache.hadoop.io.IntWritable;
5.     import org.apache.hadoop.io.Text;
6.     import org.apache.hadoop.mapred.MapReduceBase;

```

```

7.   import org.apache.hadoop.mapred.OutputCollector;
8.   import org.apache.hadoop.mapred.Reducer;
9.   import org.apache.hadoop.mapred.Reporter;
10.
11.       public class WC_Reducer extends MapReduceBase
implements Reducer<Text,IntWritable,Text,IntWritable> {
12.       public void reduce(Text key, Iterator<IntWritable>
values,OutputCollector<Text,IntWritable> output,
13.       Reporter reporter) throws IOException {
14.           int sum=0;
15.           while (values.hasNext()) {
16.               sum+=values.next().get();
17.           }
18.           output.collect(key,new IntWritable(sum));
19.       }
20.   }

```

File: WC_Runner.java

```

1. package com.javatpoint;
2.
3.   import java.io.IOException;
4.   import org.apache.hadoop.fs.Path;
5.   import org.apache.hadoop.io.IntWritable;
6.   import org.apache.hadoop.io.Text;
7.   import org.apache.hadoop.mapred.FileInputFormat;
8.   import org.apache.hadoop.mapred.FileOutputFormat;
9.   import org.apache.hadoop.mapred.JobClient;
10.      import org.apache.hadoop.mapred.JobConf;
11.      import org.apache.hadoop.mapred.TextInputFormat;
12.      import
org.apache.hadoop.mapred.TextOutputFormat;
13.      public class WC_Runner {
14.          public static void main(String[] args) throws
IOException{

```



```

15.          JobConf conf = new
JobConf(WC_Runner.class);
16.          conf.setJobName("CharCount");
17.          conf.setOutputKeyClass(Text.class);
18.
           conf.setOutputValueClass(IntWritable.class);
19.          conf.setMapperClass(WC_Mapper.class);
20.          conf.setCombinerClass(WC_Reducer.class);
21.          conf.setReducerClass(WC_Reducer.class);
22.          conf.setInputFormat(TextInputFormat.class);
23.
           conf.setOutputFormat(TextOutputFormat.class);
24.          FileInputFormat.setInputPaths(conf,new
Path(args[0]));
25.          FileOutputFormat.setOutputPath(conf,new
Path(args[1]));
26.          JobClient.runJob(conf);
27.      }
28.  }

```

Download the source code.

- Create the jar file of this program and name it **charcountdemo.jar**.
- Run the jar file
`hadoop jar /home/codegyani/charcountdemo.jar com.javatpoint.WC_Runner /count/info.txt /char_output`
- The output is stored in `/char_output/part-00000`
- Now execute the command to see the output.
`hdfs dfs -cat /r_output/part-00000`

HBase

HBase tutorial provides basic and advanced concepts of HBase. Our HBase tutorial is designed for beginners and professionals.

Hbase is an open source framework provided by Apache. It is a sorted map data built on Hadoop. It is column oriented and horizontally scalable.

Our HBase tutorial includes all topics of Apache HBase with HBase Data model, HBase Read, HBase Write, HBase MemStore, HBase Installation, RDBMS vs HBase, HBase Commands, HBase Example etc.

Prerequisite

Before learning HBase, you must have the knowledge of Hadoop and Java.

Audience

Our HBase tutorial is designed to help beginners and professionals.

Problem

We assure that you will not find any problem in this HBase tutorial. But if there is any mistake, please post the problem in contact form.

What is HBase

Hbase is an open source and sorted map data built on Hadoop. It is column oriented and horizontally scalable.

It is based on Google's Big Table. It has set of tables which keep data in key value format. Hbase is well suited for sparse data sets which are very common in big data use cases. Hbase provides APIs enabling development in practically any programming language. It is a part of the Hadoop ecosystem that provides random real-time read/write access to data in the Hadoop File System.

Why HBase

- RDBMS get exponentially slow as the data becomes large
- Expects data to be highly structured, i.e. ability to fit in a well-defined schema
- Any change in schema might require a downtime
- For sparse datasets, too much of overhead of maintaining NULL values

Features of Hbase

- Horizontally scalable: You can add any number of columns anytime.
- Automatic Failover: Automatic failover is a resource that allows a system administrator to automatically switch data handling to a standby system in the event of system compromise
- Integrations with Map/Reduce framework: All the commands and java codes internally implement Map/ Reduce to do the task and it is built over Hadoop Distributed File System.
- sparse, distributed, persistent, multidimensional sorted map, which is indexed by rowkey, column key, and timestamp.
- Often referred as a key value store or column family-oriented database, or storing versioned maps of maps.

- fundamentally, it's a platform for storing and retrieving data with random access.
- It doesn't care about datatypes(storing an integer in one row and a string in another for the same column).
- It doesn't enforce relationships within your data.
- It is designed to run on a cluster of computers, built using commodity hardware.

HBase Read

A read against HBase must be reconciled between the HFiles, MemStore & BLOCKCACHE. The BlockCache is designed to keep frequently accessed data from the HFiles in memory so as to avoid disk reads. Each column family has its own BlockCache. BlockCache contains data in form of 'block', as unit of data that HBase reads from disk in a single pass. The HFile is physically laid out as a sequence of blocks plus an index over those blocks. This means reading a block from HBase requires only looking up that block's location in the index and retrieving it from disk.

Block: It is the smallest indexed unit of data and is the smallest unit of data that can be read from disk. default size 64KB.

Scenario, when smaller block size is preferred: To perform random lookups. Having smaller blocks creates a larger index and thereby consumes more memory.

Scenario, when larger block size is preferred: To perform sequential scans frequently. This allows you to save on memory because larger blocks mean fewer index entries and thus a smaller index.

Reading a row from HBase requires first checking the MemStore, then the BlockCache, Finally, HFiles on disk are accessed.

HBase Write

When a write is made, by default, it goes into two places:

- write-ahead log (WAL), HLog, and
- in-memory write buffer, MemStore.

Clients don't interact directly with the underlying HFiles during writes, rather writes goes to WAL & MemStore in parallel. Every write to HBase requires confirmation from both the WAL and the MemStore.

HBase MemStore

- The MemStore is a write buffer where HBase accumulates data in memory before a permanent write.
- Its contents are flushed to disk to form an HFile when the MemStore fills up.
- It doesn't write to an existing HFile but instead forms a new file on every flush.
- The HFile is the underlying storage format for HBase.
- HFiles belong to a column family(one MemStore per column family). A column family can have multiple HFiles, but the reverse isn't true.
- size of the MemStore is defined in hbase-site.xml called `hbase.hregion.memstore.flush.size`.

What happens, when the server hosting a MemStore that has not yet been flushed crashes?

Every server in HBase cluster keeps a WAL to record changes as they happen. The WAL is a file on the underlying file system. A write isn't considered successful until the new WAL entry is successfully written, this guarantees durability.

If HBase goes down, the data that was not yet flushed from the MemStore to the HFile can be recovered by replaying the WAL, taken care by Hbase framework.

HBase Installation

The prerequisite for HBase installation are Java and Hadoop installed on your Linux machine.

Hbase can be installed in three modes: standalone, Pseudo Distributed mode and Fully Distributed mode.

Download the Hbase package from <http://www.interior-dsgn.com/apache/hbase/stable/> and unzip it with the below commands.

1. `$cd usr/local/$wget http://www.interior-dsgn.com/apache/hbase/stable/hbase-0.98.8-hadoop2-bin.tar.gz`
2. `$tar -zxvf hbase-0.98.8-hadoop2-bin.tar.gz`

Login as super user as shown below

1. `$su`
2. `$password: enter your password here`
3. `mv hbase-0.99.1/* Hbase/`

Configuring HBase in Standalone Mode

Set the java Home for HBase and open hbase-env.sh file from the conf folder. Edit JAVA_HOME environment variable and change the existing path to your current JAVA_HOME variable as shown below.

1. `cd /usr/local/Hbase/conf`
2. `gedit hbase-env.sh`

Replace the existing JAVA_HOME value with your current value as shown below.

1. `export JAVA_HOME=/usr/lib/jvm/java-1.7.0`

Inside /usr/local/Hbase you will find hbase-site.xml. Open it and within configuration add the below code.

1. **<configuration>**
2. //Here you have to set the path where you want HBase to store its files.
3. **<property>**
4. **<name>**hbase.rootdir**</name>**
5. **<value>**file:/home/hadoop/HBase/HFiles**</value>**
6. **</property>**
- 7.
8. //Here you have to set the path where you want HBase to store its built in zookeeper files.
9. **<property>**
10. **<name>**hbase.zookeeper.property.dataDir**</name>**
11. **<value>**/home/hadoop/zookeeper**</value>**
12. **</property>**
13. **</configuration>**

Now start the Hbase by running the start-hbase.sh present in the bin folder of Hbase.

1. \$cd /usr/local/HBase/bin
2. \$./start-hbase.sh

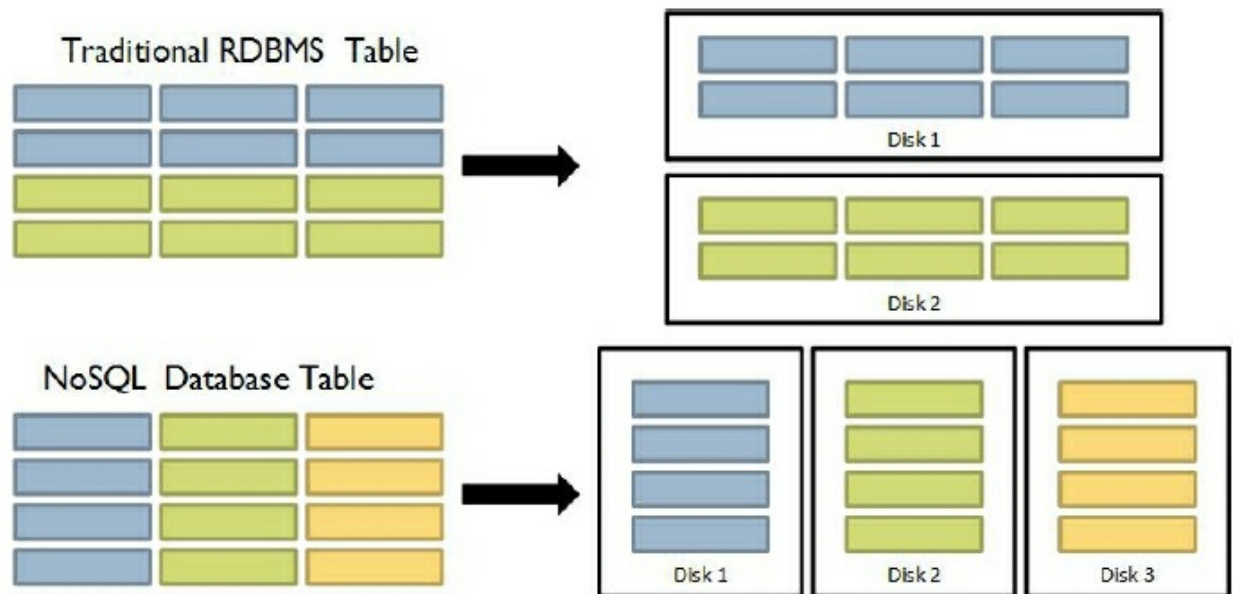
Cloudera VM is recommended as it has Hbase preinstalled on it.

Starting Hbase: Type Hbase shell in terminal to start the hbase.

RDBMS vs HBase

There differences between RDBMS and HBase are given below.

- Schema/Database in RDBMS can be compared to namespace in Hbase.
- A table in RDBMS can be compared to column family in Hbase.
- A record (after table joins) in RDBMS can be compared to a record in Hbase.
- A collection of tables in RDBMS can be compared to a table in Hbase



HBase Commands

A list of HBase commands are given below.

- **Create:** Creates a new table identified by 'table1' and Column Family identified by 'colf'.
- **Put:** Inserts a new record into the table with row identified by 'row..'
- **Scan:** returns the data stored in table
- **Get:** Returns the records matching the row identifier provided in the table
- **Help:** Get a list of commands

1. create 'table1', 'colf'
2. list 'table1'
3. put 'table1', 'row1', 'colf:a', 'value1'
4. put 'table1', 'row1', 'colf:b', 'value2'
5. put 'table1', 'row2', 'colf:a', 'value3'
6. scan 'table1'
7. get 'table1', 'row1'

HBase Example

Let's see a HBase example to import data of a file in HBase table.

Use Case

We have to import data present in the file into an HBase table by creating it through Java API.

Data_file.txt contains the below data

1. 1,India,Bihar,Champan,2009,April,P1,1,5
2. 2,India, Bihar,Patna,2009,May,P1,2,10
3. 3,India, Bihar,Bhagalpur,2010,June,P2,3,15
4. 4,United States,California,Fresno,2009,April,P2,2,5
5. 5,United States,California,Long Beach,2010,July,P2,4,10
6. 6,United States,California,San Francisco,2011,August,P1,6,20

The Java code is shown below

This data has to be inputted into a new HBase table to be created through JAVA API. Following column families have to be created

1. "sample,region,time.product,sale,profit".

Column family region has three column qualifiers: country, state, city

Column family Time has two column qualifiers: year, month

Jar Files

Make sure that the following jars are present while writing the code as they are required by the HBase.

- a. commons-logging-1.0.4
- b. commons-logging-api-1.0.4
- c. hadoop-core-0.20.2-cdh3u2
- d. hbase-0.90.4-cdh3u2
- e. log4j-1.2.15

f. zookeeper-3.3.3-cdh3u0

Program Code

```
1. import java.io.BufferedReader;
2. import java.io.File;
3. import java.io.FileReader;
4. import java.io.IOException;
5. import java.util.StringTokenizer;
6.
7. import org.apache.hadoop.conf.Configuration;
8. import org.apache.hadoop.hbase.HBaseConfiguration;
9. import org.apache.hadoop.hbase.HColumnDescriptor;
10.     import org.apache.hadoop.hbase.HTableDescriptor;
11.     import org.apache.hadoop.hbase.client.HBaseAdmin;
12.     import org.apache.hadoop.hbase.client.HTable;
13.     import org.apache.hadoop.hbase.client.Put;
14.     import org.apache.hadoop.hbase.util.Bytes;
15.
16.
17.     public class readFromFile {
18.         public static void main(String[] args) throws
IOException{
19.             if(args.length==1)
20.             {
21.                 Configuration conf =
HBaseConfiguration.create(new Configuration());
22.                 HBaseAdmin hba = new HBaseAdmin(conf);
23.                 if(!hba.tableExists(args[0])){
24.                     HTableDescriptor ht = new
HTableDescriptor(args[0]);
25.                     ht.addFamily(new
HColumnDescriptor("sample"));
26.                     ht.addFamily(new
HColumnDescriptor("region"));
```

```

27.            ht.addFamily(new
HColumnDescriptor("time"));
28.            ht.addFamily(new
HColumnDescriptor("product"));
29.            ht.addFamily(new
HColumnDescriptor("sale"));
30.            ht.addFamily(new
HColumnDescriptor("profit"));
31.            hba.createTable(ht);
32.            System.out.println("New Table Created");
33.
34.            HTable table = new HTable(conf,args[0]);
35.
36.            File f = new
File("/home/training/Desktop/data");
37.            BufferedReader br = new BufferedReader(new
FileReader(f));
38.            String line = br.readLine();
39.            int i =1;
40.            String rowname="row";
41.            while(line!=null && line.length()!=0){
42.                System.out.println("Ok till here");
43.                StringTokenizer tokens = new
StringTokenizer(line,",");
44.                rowname = "row"+i;
45.                Put p = new Put(Bytes.toBytes(rowname));
46.
47.                p.add(Bytes.toBytes("sample"),Bytes.toBytes("sar
Bytes.toBytes(Integer.parseInt(tokens.nextToken()))));
48.
49.                p.add(Bytes.toBytes("region"),Bytes.toBytes("cou
50.                p.add(Bytes.toBytes("region"),Bytes.toBytes("stat
51.                p.add(Bytes.toBytes("region"),Bytes.toBytes("city

```

```
52.         p.add(Bytes.toBytes("time"),Bytes.toBytes("year"));
53.         p.add(Bytes.toBytes("time"),Bytes.toBytes("month"));
54.         p.add(Bytes.toBytes("product"),Bytes.toBytes("product"));
55.         p.add(Bytes.toBytes("sale"),Bytes.toBytes("quantity"));
56.         p.add(Bytes.toBytes("profit"),Bytes.toBytes("earnings"));
57.         i++;
58.         table.put(p);
59.         line = br.readLine();
60.     }
61.     br.close();
62.     table.close();
63. }
64. else
65.     System.out.println("Table Already exists.Please
enter another table name");
66. }
67. else
68.     System.out.println("Please Enter the table name
through command line");
69. }
```

Hive Tutorial

Hive tutorial provides basic and advanced concepts of Hive. Our Hive tutorial is designed for beginners and professionals.

Apache Hive is a data ware house system for Hadoop that runs SQL like queries called HQL (Hive query language) which gets internally converted to map reduce jobs. Hive was developed by Facebook. It supports Data definition Language, Data Manipulation Language and user defined functions.

Our Hive tutorial includes all topics of Apache Hive with Hive Installation, Hive Data Types, Hive Table partitioning, Hive DDL commands, Hive DML commands, Hive sort by vs order by, Hive Joining tables etc.

Prerequisite

Before learning Hive, you must have the knowledge of Hadoop and Java.

Audience

Our Hive tutorial is designed to help beginners and professionals.

Problem

We assure that you will not find any problem in this Hive tutorial. But if there is any mistake, please post the problem in contact form.

What is HIVE

Hive is a data warehouse system which is used to analyze structured data. It is built on the top of Hadoop. It was developed by Facebook.

Hive provides the functionality of reading, writing, and managing large datasets residing in distributed storage. It runs SQL like queries called HQL (Hive query language) which gets internally converted to MapReduce jobs.

Using Hive, we can skip the requirement of the traditional approach of writing complex MapReduce programs. Hive supports Data Definition Language (DDL), Data Manipulation Language (DML), and User Defined Functions (UDF).

Features of Hive

These are the following features of Hive:

- Hive is fast and scalable.
- It provides SQL-like queries (i.e., HQL) that are implicitly transformed to MapReduce or Spark jobs.
- It is capable of analyzing large datasets stored in HDFS.
- It allows different storage types such as plain text, RCFile, and HBase.
- It uses indexing to accelerate queries.
- It can operate on compressed data stored in the Hadoop ecosystem.
- It supports user-defined functions (UDFs) where user can provide its functionality.

Limitations of Hive

- Hive is not capable of handling real-time data.
- It is not designed for online transaction processing.
- Hive queries contain high latency.

Differences between Hive and Pig

Hive-Pig

Hive is commonly used by Data Analysts.-Pig is commonly used by programmers.

It follows SQL-like queries.-It follows the data-flow language.

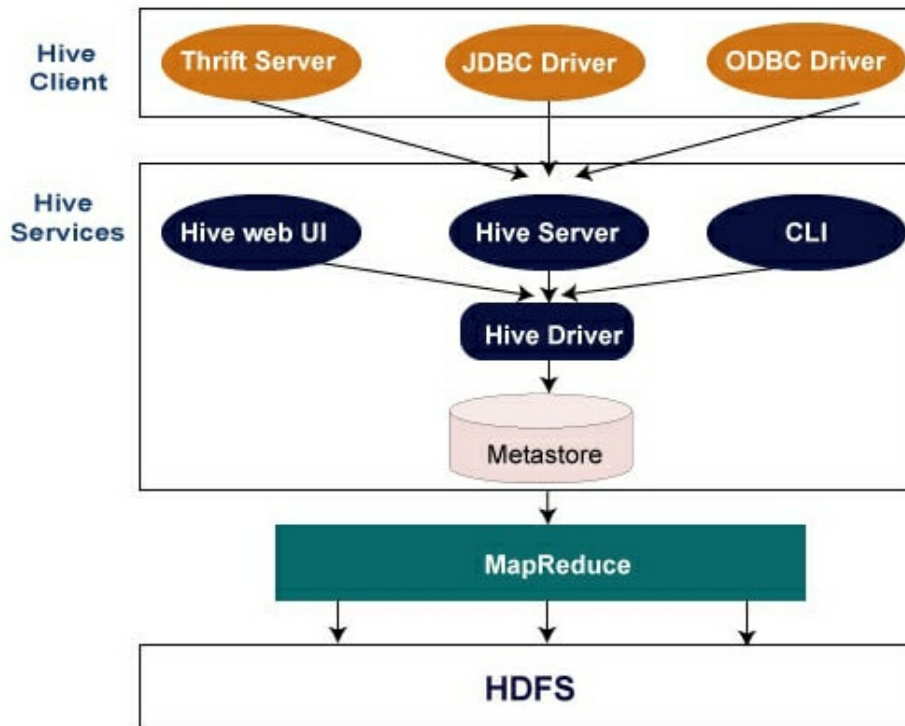
It can handle structured data.-It can handle semi-structured data.

It works on server-side of HDFS cluster.-It works on client-side of HDFS cluster.

Hive is slower than Pig.-Pig is comparatively faster than Hive.

Hive Architecture

The following architecture explains the flow of submission of query into Hive.



Hive Client

Hive allows writing applications in various languages, including Java, Python, and C++. It supports different types of clients such as:-

- **Thrift Server** - It is a cross-language service provider platform that serves the request from all those programming languages that supports Thrift.
- **JDBC Driver** - It is used to establish a connection between hive and Java applications. The JDBC Driver is present in the class `org.apache.hadoop.hive.jdbc.HiveDriver`.
- **ODBC Driver** - It allows the applications that support the ODBC protocol to connect to Hive.

Hive Services

The following are the services provided by Hive:-

- Hive CLI - The Hive CLI (Command Line Interface) is a shell where we can execute Hive queries and commands.
- Hive Web User Interface - The Hive Web UI is just an alternative of Hive CLI. It provides a web-based GUI for executing Hive queries and commands.
- Hive MetaStore - It is a central repository that stores all the structure information of various tables and partitions in the warehouse. It also includes metadata of column and its type information, the serializers and deserializers which is used to read and write data and the corresponding HDFS files where the data is stored.
- Hive Server - It is referred to as Apache Thrift Server. It accepts the request from different clients and provides it to Hive Driver.
- Hive Driver - It receives queries from different sources like web UI, CLI, Thrift, and JDBC/ODBC driver. It transfers the queries to the compiler.
- Hive Compiler - The purpose of the compiler is to parse the query and perform semantic analysis on the different query blocks and expressions. It converts HiveQL statements into MapReduce jobs.
- Hive Execution Engine - Optimizer generates the logical plan in the form of DAG of map-reduce tasks and HDFS tasks. In the end, the execution engine executes the incoming tasks in the order of their dependencies.

Apache Hive Installation

In this section, we will perform the Hive installation.

Pre-requisite

- **Java Installation** - Check whether the Java is installed or not using the following command.

1. `$ java -version`

- **Hadoop Installation** - Check whether the Hadoop is installed or not using the following command.

1. `$hadoop version`

If any of them is not installed in your system, follow the below link to install [Click Here to Install](#).

Steps to install Apache Hive

- Download the Apache Hive tar file.

<http://mirrors.estointernet.in/apache/hive/hive-1.2.2/>

- DUnzip the downloaded tar file.

1. `tar -xvf apache-hive-1.2.2-bin.tar.gz`

- DOpen the bashrc file.

1. `$ sudo nano ~/.bashrc`

- DNow, provide the following HIVE_HOME path.

1. `export HIVE_HOME=/home/codegyani/apache-hive-1.2.2-bin`

2. `export PATH=$PATH:/home/codegyani/apache-hive-1.2.2-bin/bin`

- DUpdate the environment variable.

1. `$ source ~/.bashrc`

- DLet's start the hive by providing the following command.

1. `$ hive`

HIVE Data Types

Hive data types are categorized in numeric types, string types, misc types, and complex types. A list of Hive data types is given below.

Integer Types

Type-Size-Range

TINYINT-1-byte signed integer--128 to 127

SMALLINT-2-byte signed integer-32,768 to 32,767

INT-4-byte signed integer-2,147,483,648 to 2,147,483,647

BIGINT-8-byte signed integer--9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Decimal Type

Type-Size-Range

FLOAT-4-byte-Single precision floating point number

DOUBLE-8-byte-Double precision floating point number

Date/Time Types

TIMESTAMP

- It supports traditional UNIX timestamp with optional nanosecond precision.
- As Integer numeric type, it is interpreted as UNIX timestamp in seconds.
- As Floating point numeric type, it is interpreted as UNIX timestamp in seconds with decimal precision.
- As string, it follows java.sql.Timestamp format "YYYY-MM-DD HH:MM:SS.ffffffff" (9 decimal place precision)

DATES

The Date value is used to specify a particular year, month and day, in the form YYYY--MM--DD. However, it didn't provide the time of the day. The range of Date type lies between 0000--01--01 to 9999--12--31.

String Types

STRING

The string is a sequence of characters. Its values can be enclosed within single quotes (') or double quotes (").

Varchar

The varchar is a variable length type whose range lies between 1 and 65535, which specifies that the maximum number of characters allowed in the character string.

CHAR

The char is a fixed-length type whose maximum length is fixed at 255.

Complex Type

Type-Size-Range

Struct-It is similar to C struct or an object where fields are accessed using the "dot" notation.-struct('James','Roy')

Map-It contains the key-value tuples where the fields are accessed using array notation.-map('first','James','last','Roy')

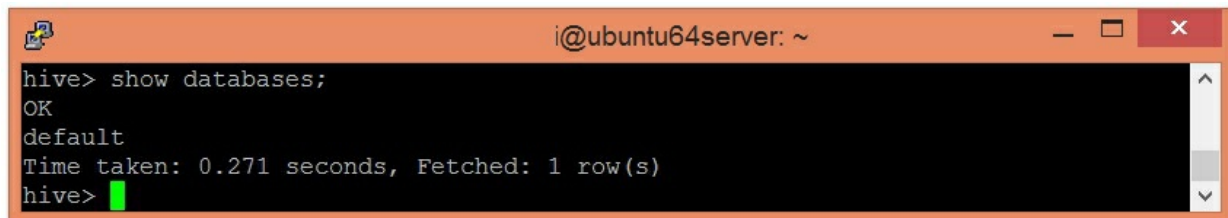
Array-It is a collection of similar type of values that is indexable using zero-based integers.-array('James','Roy')

Hive - Create Database

In Hive, the database is considered as a catalog or namespace of tables. So, we can maintain multiple tables within a database where a unique name is assigned to each table. Hive also provides a default database with a name **default**.

- Initially, we check the default database provided by Hive. So, to check the list of existing databases, follow the below command: -

1. hive> show databases;

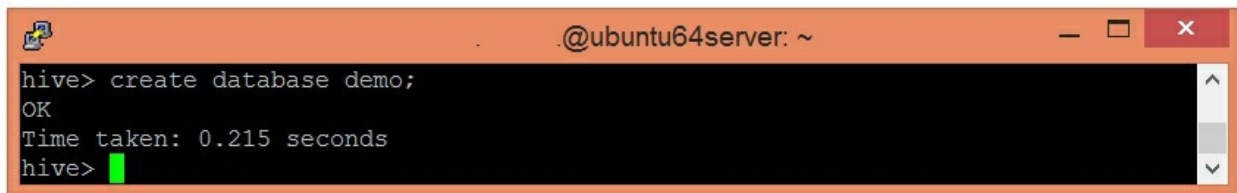


```
i@ubuntu64server: ~  
hive> show databases;  
OK  
default  
Time taken: 0.271 seconds, Fetched: 1 row(s)  
hive>
```

Here, we can see the existence of a default database provided by Hive.

- Let's create a new database by using the following command: -

1. hive> create database demo;



```
@ubuntu64server: ~  
hive> create database demo;  
OK  
Time taken: 0.215 seconds  
hive>
```

So, a new database is created.

- Let's check the existence of a newly created database.

1. hive> show databases;


```
@ubuntu64server: ~
hive> show databases;
OK
default
demo
Time taken: 0.177 seconds, Fetched: 2 row(s)
hive>
```

Each database must contain a unique name. If we create two databases with the same name, the following error generates:

```
@ubuntu64server: ~
hive> create database demo;
FAILED: Execution Error, return code 1 from org.apache.hadoop.hive.ql.exec.DDLTask. Database demo already exists
hive>
```

- If we want to suppress the warning generated by Hive on creating the database with the same name, follow the below command: -

1. hive> create a database if not exists demo;

```
@ubuntu64server: ~
hive> create database if not exists demo;
OK
Time taken: 0.107 seconds
```

- Hive also allows assigning properties with the database in the form of key-value pair.

1. hive> create the database demo
2. >WITH DBPROPERTIES ('creator' = 'Gaurav Chawla', 'date' = '2019-06-03');

```
i@ubuntu64server: ~
hive> create database demo
> WITH DBPROPERTIES ('creator' = 'Gaurav Chawla', 'date' = '2019-06-03');
OK
Time taken: 2.389 seconds
hive>
```

- Let's retrieve the information associated with the database.

1. hive> describe database extended demo;

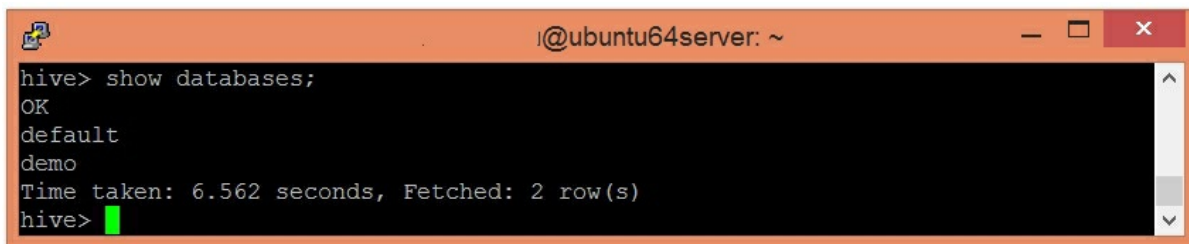
```
@ubuntu64server: ~
hive> describe database extended demo;
OK
demo          hdfs://192.168.56.123:8020/user/hive/warehouse/demo.db  codegyan
i      USER    {date=2019-06-03, creator=Gaurav Chawla}
Time taken: 0.228 seconds, Fetched: 1 row(s)
hive>
```

Hive - Drop Database

In this section, we will see various ways to drop the existing database.

- Let's check the list of existing databases by using the following command: -

1. hive> show databases;



```
@ubuntu64server: ~  
hive> show databases;  
OK  
default  
demo  
Time taken: 6.562 seconds, Fetched: 2 row(s)  
hive>
```

- Now, drop the database by using the following command.

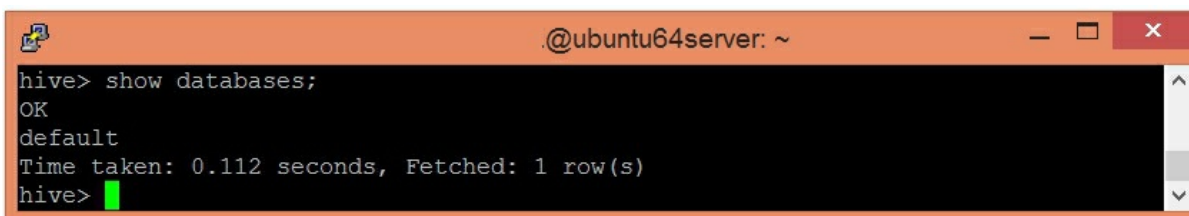
1. hive> drop database demo;



```
@ubuntu64server: ~  
hive> drop database demo;  
OK  
Time taken: 2.354 seconds  
hive>
```

- Let's check whether the database is dropped or not.

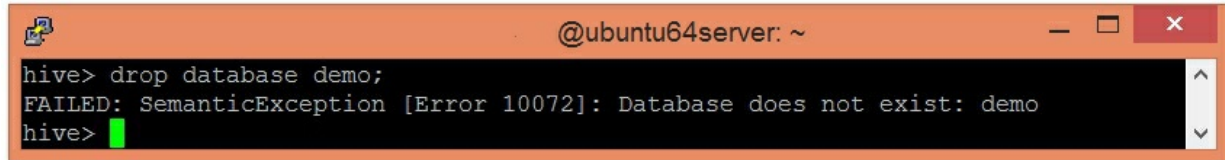
1. hive> show databases;



```
@ubuntu64server: ~  
hive> show databases;  
OK  
default  
Time taken: 0.112 seconds, Fetched: 1 row(s)  
hive>
```

As we can see, the database **demo** is not present in the list. Hence, the database is dropped successfully.

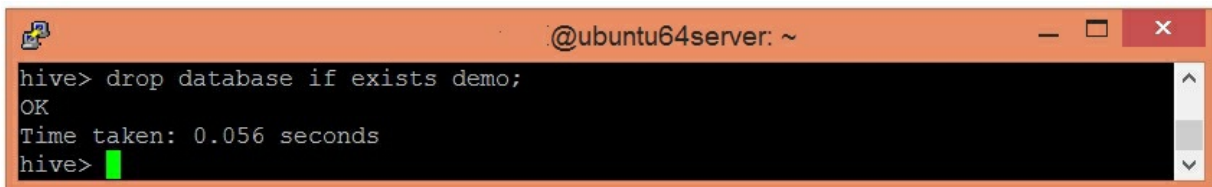
- If we try to drop the database that doesn't exist, the following error generates:

A terminal window titled "@ubuntu64server: ~" with standard window controls. The command "hive> drop database demo;" has been entered. The output is "FAILED: SemanticException [Error 10072]: Database does not exist: demo", followed by a new prompt "hive>".

```
@ubuntu64server: ~
hive> drop database demo;
FAILED: SemanticException [Error 10072]: Database does not exist: demo
hive>
```

- However, if we want to suppress the warning generated by Hive on creating the database with the same name, follow the below command:-

1. hive> drop database if exists demo;

A terminal window titled "@ubuntu64server: ~" with standard window controls. The command "hive> drop database if exists demo;" has been entered. The output is "OK" followed by "Time taken: 0.056 seconds", and a new prompt "hive>".

```
@ubuntu64server: ~
hive> drop database if exists demo;
OK
Time taken: 0.056 seconds
hive>
```

- In Hive, it is not allowed to drop the database that contains the tables directly. In such a case, we can drop the database either by dropping tables first or use Cascade keyword with the command.
- Let's see the cascade command used to drop the database:-

1. hive> drop database if exists demo cascade;

This command automatically drops the tables present in the database first.

Hive - Create Table

In Hive, we can create a table by using the conventions similar to the SQL. It supports a wide range of flexibility where the data files for tables are stored. It provides two types of table: -

- Internal table
- External table

Internal Table

The internal tables are also called managed tables as the lifecycle of their data is controlled by the Hive. By default, these tables are stored in a subdirectory under the directory defined by `hive.metastore.warehouse.dir` (i.e. `/user/hive/warehouse`). The internal tables are not flexible enough to share with other tools like Pig. If we try to drop the internal table, Hive deletes both table schema and data.

- Let's create an internal table by using the following command:-
 1. `hive> create table demo.employee (Id int, Name string , Salary float)`
 2. `row format delimited`
 3. `fields terminated by ',' ;`

Here, the command also includes the information that the data is separated by ','.

- Let's see the metadata of the created table by using the following command:-
 1. `hive> describe demo.employee`

External Table

The external table allows us to create and access a table and a data externally.

The **external** keyword is used to specify the external table, whereas the **location** keyword is used to determine the location of loaded data.

As the table is external, the data is not present in the Hive directory.

Therefore, if we try to drop the table, the metadata of the table will be deleted, but the data still exists.

To create an external table, follow the below steps: -

- Let's create a directory on HDFS by using the following command: -
 1. `hdfs dfs -mkdir /HiveDirectory`
- Now, store the file on the created directory.
 1. `hdfs dfs -put hive/emp_details /HiveDirectory`
- Let's create an external table using the following command: -
 1. `hive> create external table emplist (Id int, Name string , Salary float)`
 2. `row format delimited`
 3. `fields terminated by ','`
 4. `location '/HiveDirectory';`

Hive - Load Data

Once the internal table has been created, the next step is to load the data into it. So, in Hive, we can easily load data from any file to the database.

- Let's load the data of the file into the database by using the following command: -

1. load data local inpath '/home/codegyani/hive/emp_details' into table demo.employee;

Here, **emp_details** is the file name that contains the data.

- Now, we can use the following command to retrieve the data from the database.

1. select * from demo.employee;

- If we want to add more data into the current database, execute the same query again by just updating the new file name.

1. load data local inpath '/home/codegyani/hive/emp_details1' into table demo.employee;

- In Hive, if we try to load unmatched data (i.e., one or more column data doesn't match the data type of specified table columns), it will not throw any exception. However, it stores the Null value at the position of unmatched tuple.
- Let's add one more file to the current table. This file contains the unmatched data.

Here, the third column contains the data of string type, and the table allows the float type data. So, this condition arises in an unmatched data situation.

- Now, load the data into the table.

1. load data local inpath '/home/codegyani/hive/emp_details2' into table

```
demo.employee;
```

Here, data loaded successfully.

- Let's fetch the records of the table.

1. `select * from demo.employee`

Here, we can see the Null values at the position of unmatched data.

Partitioning in Hive

The partitioning in Hive means dividing the table into some parts based on the values of a particular column like date, course, city or country. The advantage of partitioning is that since the data is stored in slices, the query response time becomes faster.

As we know that Hadoop is used to handle the huge amount of data, it is always required to use the best approach to deal with it. The partitioning in Hive is the best example of it.

Let's assume we have a data of 10 million students studying in an institute. Now, we have to fetch the students of a particular course. If we use a traditional approach, we have to go through the entire data. This leads to performance degradation. In such a case, we can adopt the better approach i.e., partitioning in Hive and divide the data among the different datasets based on particular columns.

The partitioning in Hive can be executed in two ways -

- [Static partitioning](#)
- [Dynamic partitioning](#)

Static Partitioning

In static or manual partitioning, it is required to pass the values of partitioned columns manually while loading the data into the table. Hence, the data file doesn't contain the partitioned columns.

Example of Static Partitioning

- First, select the database in which we want to create a table.
1. `hive> use test;`
 - Create the table and provide the partitioned columns by using the following command: -
 1. `hive> create table student (id int, name string, age int, institute string)`

2. partitioned by (course string)
3. row format delimited
4. fields terminated by ',';

- Let's retrieve the information associated with the table.

1. hive> describe student;

- Load the data into the table and pass the values of partition columns with it by using the following command: -

1. hive> load data local inpath '/home/codegyani/hive/student_details1' into table student
2. partition(course= "java");

Here, we are partitioning the students of an institute based on courses.

- Load the data of another file into the same table and pass the values of partition columns with it by using the following command: -

1. hive> load data local inpath '/home/codegyani/hive/student_details2' into table student
2. partition(course= "hadoop");

- Let's retrieve the entire data of the table by using the following command: -

1. hive> select * from student;

- Now, try to retrieve the data based on partitioned columns by using the following command: -

1. hive> select * from student where course="java";

In this case, we are not examining the entire data. Hence, this approach

improves query response time.

- Let's also retrieve the data of another partitioned dataset by using the following command: -

1. `hive> select * from student where course= "hadoop";`

Dynamic Partitioning

In dynamic partitioning, the values of partitioned columns exist within the table. So, it is not required to pass the values of partitioned columns manually.

- First, select the database in which we want to create a table.

1. `hive> use show;`

- Enable the dynamic partition by using the following commands: -

1. `hive> set hive.exec.dynamic.partition=true;`
2. `hive> set hive.exec.dynamic.partition.mode=nonstrict;`

- Create a dummy table to store the data.

1. `hive> create table stud_demo(id int, name string, age int, institute string, course string)`
2. `row format delimited`
3. `fields terminated by ',';`

- Now, load the data into the table.

1. `hive> load data local inpath '/home/codegyani/hive/student_details' into table stud_demo;`

- Create a partition table by using the following command: -

1. `hive> create table student_part (id int, name string, age int, institute string)`
2. `partitioned by (course string)`
3. `row format delimited`
4. `fields terminated by ',';`

- Now, insert the data of dummy table into the partition table.

1. hive> insert into student_part
2. partition(course)
3. select id, name, age, institute, course
4. from stud_demo;

Hadoop Interview Questions

There is given Hadoop interview questions and answers that have been asked in many companies. Let's see the list of top Hadoop interview questions.

1) What is Hadoop?

Hadoop is a distributed computing platform. It is written in Java. It consists of the features like Google File System and MapReduce.

2) What platform and Java version are required to run Hadoop?

Java 1.6.x or higher versions are good for Hadoop, preferably from Sun. Linux and Windows are the supported operating system for Hadoop, but BSD, Mac OS/X, and Solaris are more famous for working.

3) What kind of Hardware is best for Hadoop?

Hadoop can run on a dual processor/ dual core machines with 4-8 GB RAM using ECC memory. It depends on the workflow needs.

4) What are the most common input formats defined in Hadoop?

These are the most common input formats defined in Hadoop:

1. TextInputFormat
2. KeyValueInputFormat
3. SequenceFileInputFormat

TextInputFormat is a by default input format.

5) How do you categorize a big data?

The big data can be categorized using the following features:

- Volume
- Velocity
- Variety

6) Explain the use of .media class?

For the floating of media objects from one side to another, we use this class.

7) Give the use of the bootstrap panel.

We use panels in bootstrap from the boxing of DOM components.

8) What is the purpose of button groups?

Button groups are used for the placement of more than one buttons in the same line.

9) Name the various types of lists supported by Bootstrap.

- Ordered list
- Unordered list
- Definition list

10) Which command is used for the retrieval of the status of daemons running the Hadoop cluster?

The 'jps' command is used for the retrieval of the status of daemons running the Hadoop cluster.

11) What is InputSplit in Hadoop? Explain.

When a Hadoop job runs, it splits input files into chunks and assigns each split to a mapper for processing. It is called the InputSplit.

12) What is TextInputFormat?

In TextInputFormat, each line in the text file is a record. Value is the content of the line while Key is the byte offset of the line. For instance, Key: longWritable, Value: text

13) What is the SequenceFileInputFormat in Hadoop?

In Hadoop, SequenceFileInputFormat is used to read files in sequence. It is a specific compressed binary file format which passes data between the output of one MapReduce job to the input of some other MapReduce job.

14) How many InputSplits is made by a Hadoop Framework?

Hadoop makes 5 splits as follows:

- One split for 64K files
- Two splits for 65MB files, and
- Two splits for 127MB files

15) What is the use of RecordReader in Hadoop?

InputSplit is assigned with a work but doesn't know how to access it. The record holder class is totally responsible for loading the data from its source and convert it into keys pair suitable for reading by the Mapper. The RecordReader's instance can be defined by the Input Format.

16) What is JobTracker in Hadoop?

JobTracker is a service within Hadoop which runs MapReduce jobs on the cluster.

17) What is WebDAV in Hadoop?

WebDAV is a set of extension to HTTP which is used to support editing and uploading files. On most operating system WebDAV shares can be mounted as filesystems, so it is possible to access HDFS as a standard filesystem by exposing HDFS over WebDAV.

18) What is Sqoop in Hadoop?

Sqoop is a tool used to transfer data between the Relational Database Management System (RDBMS) and Hadoop HDFS. By using Sqoop, you can transfer data from RDBMS like MySQL or Oracle into HDFS as well as exporting data from HDFS file to RDBMS.

19) What are the functionalities of JobTracker?

These are the main tasks of JobTracker:

- To accept jobs from the client.
- To communicate with the NameNode to determine the location of the data.
- To locate TaskTracker Nodes with available slots.
- To submit the work to the chosen TaskTracker node and monitors the progress of each task.

20) Define TaskTracker.

TaskTracker is a node in the cluster that accepts tasks like MapReduce and Shuffle operations from a JobTracker.

21) What is Map/Reduce job in Hadoop?

Map/Reduce job is a programming paradigm which is used to allow massive scalability across the thousands of server.

MapReduce refers to two different and distinct tasks that Hadoop performs.

In the first step maps jobs which takes the set of data and converts it into another set of data and in the second step, Reduce job. It takes the output from the map as input and compresses those data tuples into the smaller set of tuples.

22) What is "map" and what is "reducer" in Hadoop?

Map: In Hadoop, a map is a phase in HDFS query solving. A map reads data from an input location and outputs a key-value pair according to the input type.

Reducer: In Hadoop, a reducer collects the output generated by the mapper, processes it, and creates a final output of its own.

23) What is shuffling in MapReduce?

Shuffling is a process which is used to perform the sorting and transfer the map outputs to the reducer as input.

24) What is NameNode in Hadoop?

NameNode is a node, where Hadoop stores all the file location information in HDFS (Hadoop Distributed File System). We can say that NameNode is the centerpiece of an HDFS file system which is responsible for keeping the record of all the files in the file system, and tracks the file data across the cluster or multiple machines.

25) What is heartbeat in HDFS?

Heartbeat is a signal which is used between a data node and name node, and between task tracker and job tracker. If the name node or job tracker doesn't respond to the signal then it is considered that there is some issue with data node or task tracker.

26) How is indexing done in HDFS?

There is a very unique way of indexing in Hadoop. Once the data is stored as per the block size, the HDFS will keep on storing the last part of the data which specifies the location of the next part of the data.

27) What happens when a data node fails?

If a data node fails the job tracker and name node will detect the failure. After that, all tasks are re-scheduled on the failed node and then name node will replicate the user data to another node.

28) What is Hadoop Streaming?

Hadoop streaming is a utility which allows you to create and run map/reduce job. It is a generic API that allows programs written in any languages to be used as Hadoop mapper.

29) What is a combiner in Hadoop?

A Combiner is a mini-reduce process which operates only on data generated by a Mapper. When Mapper emits the data, combiner receives it as input and sends the output to a reducer.

30) What are the Hadoop's three configuration files?

Following are the three configuration files in Hadoop:

- core-site.xml
- mapred-site.xml
- hdfs-site.xml

31) What are the network requirements for using Hadoop?

Following are the network requirement for using Hadoop:

- Password-less SSH connection.

- Secure Shell (SSH) for launching server processes.

32) What do you know by storage and compute node?

Storage node: Storage Node is the machine or computer where your file system resides to store the processing data.

Compute Node: Compute Node is a machine or computer where your actual business logic will be executed.

33) Is it necessary to know Java to learn Hadoop?

If you have a background in any programming language like C, C++, PHP, Python, Java, etc. It may be really helpful, but if you are nil in java, it is necessary to learn Java and also get the basic knowledge of SQL.

34) How to debug Hadoop code?

There are many ways to debug Hadoop codes but the most popular methods are:

- By using Counters.
- By web interface provided by the Hadoop framework.

35) Is it possible to provide multiple inputs to Hadoop? If yes, explain.

Yes, It is possible. The input format class provides methods to insert multiple directories as input to a Hadoop job.

36) What is the relation between job and task in Hadoop?

In Hadoop, A job is divided into multiple small parts known as the task.

37) What is the difference between Input Split and HDFS Block?

The Logical division of data is called Input Split and physical division of data is called HDFS Block.

38) What is the difference between RDBMS and Hadoop?

RDBMS-Hadoop

RDBMS is a relational database management system.- Hadoop is a node based flat structure.

RDBMS is used for OLTP processing.- Hadoop is used for analytical and for big data processing.

In RDBMS, the database cluster uses the same data files stored in shared storage.- In Hadoop, the storage data can be stored independently in each processing node.

In RDBMS, preprocessing of data is required before storing it.- In Hadoop, you don't need to preprocess data before storing it.

39) What is the difference between HDFS and NAS?

HDFS data blocks are distributed across local drives of all machines in a cluster whereas, NAS data is stored on dedicated hardware.

40) What is the difference between Hadoop and other data processing tools?

Hadoop facilitates you to increase or decrease the number of mappers without worrying about the volume of data to be processed.