**The Scala Programming Language**

Scala is an acronym for **Sca**lable **La**nguage.

Scala is a modern **multi-paradigm programming language** designed to **express common programming patterns** in a concise, elegant & type-safe way.

Scala is written by **Martin Odersky at EPFL**.

* Statically Typed
* Runs on JVM, Full inter-op with Java
* Object Oriented
* Functional
* Dynamic Features
* Scala **blends object oriented & functional programming in a statically typed language**.

**Scala is Practical**

1. Can be used as drop-in replacement for Java
2. Mixed Scala/ Java Projects
3. Use existing **Java Libraries**
4. Use **existing Java tools** (Ant, Maven, Junit, etc...)
5. Decent **IDE Support** (NetBeans, IntelliJ, Eclipse or Scala IDE)

**What is SBT?**

**SBT** (Scala Build Tool, Formerly Simple Build Tool) is an **open source build tool** for Scala & Java Projects, Similar to Java’s Maven & Ant.

**SBT is a modern build tool**. While it is written in Scala & provides many Scala conveniences, it is a **general purpose build tool**.

SBT is the de facto Build tool in the Scala

**Why SBT?**

1. **Native Support** for compiling Scala Code.
2. **Uses Apache Ivy** for dependency management.
3. **Only-update-on-request** model
4. **Full Scala language** Support for creating tasks
5. Support for **mixed Java/Scala Projects**
6. **Launch REPL (Real-Eval-Print Loop) in Project Context.**

**How to Install SBT in to Windows 10**

1. **Download JDK 8 & Install**

Check `javac` command in your CMD

Navigate -> Program File -> Java -> JDK 8 -> bin -> Copy Path

Go -> Environmental Variable -> Path -> Edit -> New

Navigate -> Program File -> Java -> JRE 8 -> bin -> Copy Path

Go -> Environmental Variable -> Path -> Edit -> New

Make Environmental Variable -> JAVA\_HOME -> JDK 8 Path -> OK

1. **Download Scala** -> [www.scala-lang.org](http://www.scala-lang.org)

Choose Download SBT option

Select MSI File and download it & Install it

Navigate -> program file(x86) -> SBT -> bin older -> Copy Path

Go -> Environmental Variable -> Path -> Edit -> New

If path exist you don’t need to change it

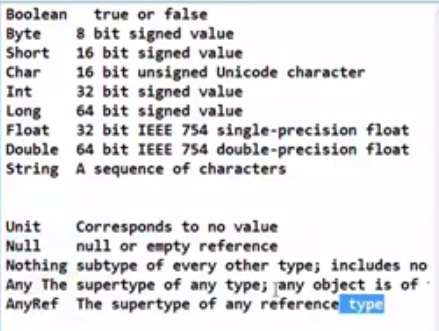
Open Command Prompt -> Make a Directory name `sbt\_project`

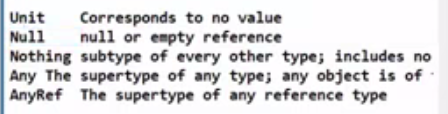
Navigate project folder -> Execute SBT command

First time you can it download big amount of files

**How to use Variables & Data Types in Scala**

Command Prompt -> **sbt console** and Enter Then You can enter in to SBT console





In Scala there are two type of variables

1. **var** -> This can be reassigned, SUM, Minas, You can totally change this to any value (ex:- var a : Int = 12)

In this case:

var a : Int = 12

**a : Int = 12**

-----------------------------------------------------------

a+20

**res0 : Int = 32**

-----------------------------------------------------------

a-6

**res1 : Int = 6**

-----------------------------------------------------------

a= 30

**a : Int = 30**

1. **val** -> this is a constant (So this can’t be changed (ex:- val b : int = 20)

val b : Int = 30

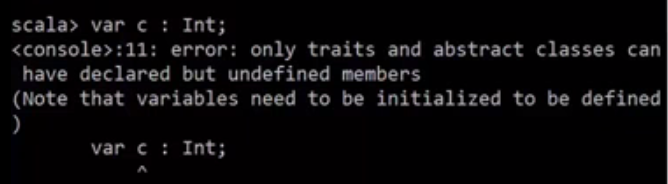
**b : Int = 30**

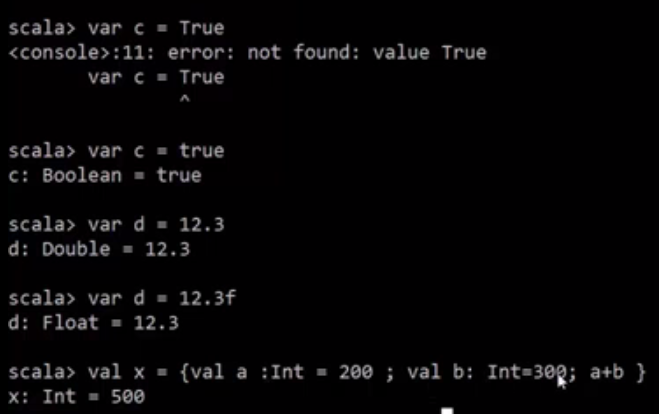
b= 40

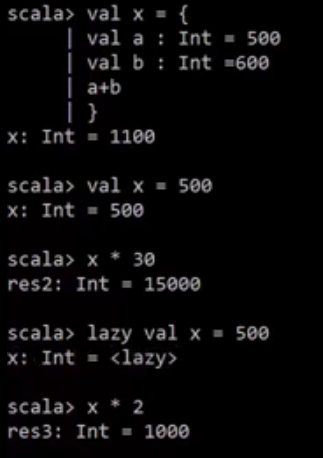
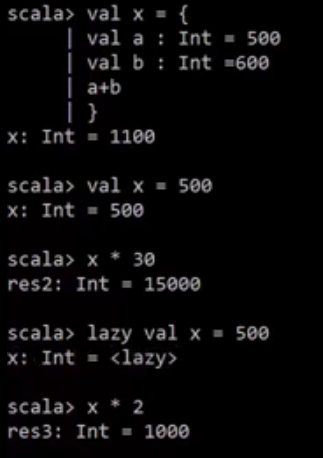
<console> **:30: error: reassignment to val**

**b=40**

**^**

**Scala is a Compile Language not interpreted Language.**

* So we can’t just declare variables in Scala



After creating first Scala-Project -> go to -> [www.docs.scala-lang.org](http://www.docs.scala-lang.org)

Select Option Scala for Java Programmers

Then scroll down a little bit and u can see the first java code example

object HelloWorld {

    def main (args: Array[String]):Unit ={

        println ("Hello World!")

    }

}

Output🡪

Hello World!

Go to you created Scala project -> right click on source (./src)

Select New -> Scala Object

Give **HelloWorld** to Scala object

**How to change preferences of Scala Project**

Go to top menu Bar -> Windows -> Preferences-> General -> Editors -> text editors -> Colors & Fonts ->

Change Text fonts -> Select Text Font

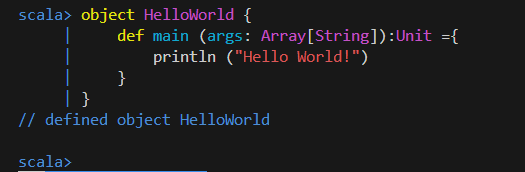
Copy the same method and paste it inside object class

Right Click -> Run as -> Scala Project

To open Scala Interpreter -> Right Click on the project -> Select Scala

Choose 2nd option (Create Scala interpreter in Scala sample)

Now it open the Scala Interpreter



**Scala String Interpolation**

object `String\_interpolation-first` {

  def main(args: Array[String]) {

    val name = "Tharindu"

    val age = 24

    val height = 5.7

    println(name + " is " + age + " years old.")

    println(s"$name is $age years old")

    println(f"$name%s is $age%d years old & $height%f feet")

    println(s"Hello \nWorld")

    println(raw"Hello \nWorld")

    println(raw"$name%s is $age%d years old & $height%f feet")

    println(raw"$name is $age years old & $height feet")

  }

}

Output🡪

Tharindu is 24 years old.

Tharindu is 24 years old

Tharindu is 24 years old & 5.700000 feet

Hello

World

Hello \nWorld

Tharindu%s is 24%d years old & 5.7%f feet

Tharindu is 24 years old & 5.7 feet

**Scala if/else**

object if\_else\_scala {

  def main(args: Array[String]) {

    val x = 40;

    var res = "";

    if (x == 20) {

      println("This matches");

      res = "This response Matches"

    } else {

      println("Not Matching");

      res = s"This response is not matching because x is equal to $x"

    }

    println(res)

    val res2 = if (x == 20) "X is equal to 20" else s"x is equal to $x";

    println(res2);

  }

}

Output🡪

Not Matching

This response is not matching because x is equal to 40

x is equal to 40

object if\_else\_scala\_2nd {

  def main(args: Array[String]) {

    val x = 20;

    val y = 30;

    var res = "";

    if (x == 20 || y == 30) {

      res = s"Matching x is equal to $x & y is equal to $y";

    } else {

      res = s"Not Matching because x is equal to $x & y is equal to $y";

    }

    println(res);

  }

}

Matching x is equal to 20 & y is equal to 30

object if\_else\_scala\_2nd {

  def main(args: Array[String]) {

    val x = 20;

    val y = 30;

    var res = "";

    if (x < 20 || y == 30) {

      res = s"Matching x is equal to $x & y is equal to $y";

    } else {

      res = s"Not Matching because x is equal to $x & y is equal to $y";

    }

    println(res);

  }}

Matching x is equal to 20 & y is equal to 30

object if\_else\_scala\_2nd {

  def main(args: Array[String]) {

    val x = 20;

    val y = 30;

    var res = "";

    if (x < 20 && y == 30) {

      res = s"Matching x is equal to $x & y is equal to $y";

    } else {

      res = s"Not Matching because x is equal to $x & y is equal to $y";

    }

    println(res);

  }

}

Not Matching because x is equal to 20 & y is equal to 30

**Scala Loops (While/do-while)**

object scala\_loops {

  def main(args: Array[String]) {

    var x = 0;

    while (x < 10) {

      println("x= " + x);

      x += 1; // x = x+1

    }

  }}

object scala\_loops {

  def main(args: Array[String]) {

    var x = 0;

    while (x < 10) {

      println("x= " + x);

      x += 1; // x = x+1

    }

    var y = 0;

    do {

      println("y = " + y);

      y += 1;

    } while (y < 0); } }

**Scala for Loop (Ranges)**

object for\_loop {

  def main(args: Array[String]) {

    for (i <- 1 to 5) {

      println("i using to " + i);

    }

    println("Next Example of For Loop");

    for (i <- 1.to(5)) {

      println("i using to " + i);

    }

    println("Next Example of For Loop");

    for (i <- 1.until(6)) {

      println("I using until " + i);

    }

    println("Next Example of For Loop");

    for (i <- 1 until 6) {

      println("I using until " + i);

    }

    println("Next Example of For Loop");

    for (i <- 1 to 9; j <- 1 to 3) { // multiple Ranges

      println("i using multiple ranges " + i + " " + j);

    }

    val list = List(1, 2, 3, 5, 4, 6, 78, 9, 6, 8)

    for (i <- list) {

      println("i using list " + i);

    }

    println("Next Example of For Loop");

    val list2 = List(1, 2, 3, 5, 4, 6, 78, 9, 6, 8)

    for (i <- list2; if i < 6) {

      println("i using Filters " + i);

    }

    val list3 = List(1, 2, 3, 5, 4, 6, 78, 9, 6, 8)

   val result = for {

      i <- list3

      if i < 6

    } yield  {

      i \* i

    }

    println("result =" + result);

  }

}

Output🡪

i using to 1

i using to 2

i using to 3

i using to 4

i using to 5

Next Example of For Loop

i using to 1

i using to 2

i using to 3

i using to 4

i using to 5

Next Example of For Loop

I using until 1

I using until 2

I using until 3

I using until 4

I using until 5

Next Example of For Loop

I using until 1

I using until 2

I using until 3

I using until 4

I using until 5

Next Example of For Loop

i using multiple ranges 1 1

i using multiple ranges 1 2

i using multiple ranges 1 3

i using multiple ranges 2 1

i using multiple ranges 2 2

i using multiple ranges 2 3

i using multiple ranges 3 1

i using multiple ranges 3 2

i using multiple ranges 3 3

i using multiple ranges 4 1

i using multiple ranges 4 2

i using multiple ranges 4 3

i using multiple ranges 5 1

i using multiple ranges 5 2

i using multiple ranges 5 3

i using multiple ranges 6 1

i using multiple ranges 6 2

i using multiple ranges 6 3

i using multiple ranges 7 1

i using multiple ranges 7 2

i using multiple ranges 7 3

i using multiple ranges 8 1

i using multiple ranges 8 2

i using multiple ranges 8 3

i using multiple ranges 9 1

i using multiple ranges 9 2

i using multiple ranges 9 3

i using list 1

i using list 2

i using list 3

i using list 5

i using list 4

i using list 6

i using list 78

i using list 9

i using list 6

i using list 8

Next Example of For Loop

i using Filters 1

i using Filters 2

i using Filters 3

i using Filters 5

i using Filters 4

result =List(1, 4, 9, 25, 16)

**Scala Match Expressions**

object Match\_expression {

  def main(args: Array[String]) {

    val age = 30;

    val age1 = "30";

    age match {

      case 20 => println(age);

      case 18 => println(age);

      case 30 => println(age);

      case 50 => println(age);

      case \_ => println("default"); // default

    }

    age1 match {

      case "20" => println(age1);

      case "18" => println(age1);

      case "50" => println(age1);

      case \_ => println("default"); // default

    }

    val result = age1 match {

      case "20" => age1;

      case "30" => age1;

      case "15" => age1;

      case \_ => "default"; // default

    }

    println("result = " + result);

    val i = 7;

    i match {

      case 1 | 3 | 5 | 7 | 9  => println(s"$i is odd number");

      case 2 | 4 | 6 | 8 | 10 => println(s"$i is even number");

    }

  }

}

Output🡪

30

default

result = 30

7 is odd number

**Scala Functions in Scala**

object Demo {

  object Math {

    def add(x: Int, y: Int): Int = {

      return x + y;

    }

    def squre(x: Int) = x \* x;

  }

  def add(x: Int, y: Int): Int = {

    return x + y;

  }

  def substarct(x: Int, y: Int): Int = {

    x - y;

  }

  def multiply(x: Int, y: Int): Int = x \* y;

  def divide(x: Int, y: Int) = x / y;

  def main(args: Array[String]) {

    println(Math.squre(9));

    println(Math.add(45, 15));

    println(Math squre 9); // syntactical suger

    println(add(45, 15));

    println(substarct(45, 15));

    println(multiply(45, 15));

    println(divide(45, 15));

  }

}

Output🡪

81

60

81

60

30

675

3

object Demo2 {

  object Math {

    def add(x: Int = 45, y: Int = 15): Int = {

      return x + y;

    }

    def squre(x: Int) = x \* x;

    def +(x: Int, y: Int): Int = {

      return x + y;

    }

    def \*\*(x: Int) = x \* x;

  }

  def main(args: Array[String]) {

    println(

      Math.add(60)

    ); // When do this this 60 parameter assign to the first parameter

    println(Math squre 9);

    var add = (x: Int, y: Int) => x + y;

    println(add(300, 50));

    val sum = 10 + 20;

    println(sum);

    println(Math.+(60, 15));

    println(Math \*\* 9);

  }

}

Output🡪

75

81

350

30

75

81

**Scala Higher Order Functions**

object higher\_order\_func {

  def math(x: Double, y: Double, f: (Double, Double) => Double): Double =

    f(x, y);

  def math(

      x: Double,

      y: Double,

      z: Double,

      f: (Double, Double) => Double

  ): Double = f(f(x, y), z);

  def main(args: Array[String]) {

    val result\_sum = math(51.67, 60.45, (x, y) => x + y); // sum

    println(result\_sum);

    val result\_min = math(25.89, 20.90, (x, y) => x min y); // min

    println(result\_min);

    val result\_max = math(50, 20, (x, y) => x max y); // max

    println(result\_max);

    val result\_multiply =

      math(50.001, 20.98, 10.8, (x, y) => x \* y); // multiply

    println(result\_multiply);

    val result\_def\_multiply = math(50, 20, 10, \_ + \_); // sum

    println(result\_def\_multiply);

    val result\_de\_min = math(50, 20, 10, \_ min \_); // sum

    println(result\_de\_min);

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

    val f = sum(10, 20, \_: Int)

    println(f(30));

    val h = sum(10, \_: Int, \_: Int)

    println(h(30, 100));

  }

}

Output🡪

112.12

20.9

50.0

11329.426584

80.0

10.0

60

140

**Scala Higher Order Functions - Date**

import java.util.Date

object date {

  var number = 10

  val add = (x: Int) => x + number

  def main(args: Array[String]) {

    val now = new Date()

    println("Current Date: " + now)

    println("Add result: " + add(20))

  }

}

Output🡪

Current Date: Tue Apr 08 19:14:42 IST 2025

Add result: 30

**Scala Closures**

A closure is a function which uses one or more variables declared outside this function

object closures {

  var number = 10;

  val add = (x: Int) => x + number;

  val substract = (x: Int) => {

    number = x -number;

    number;

}

val mulitiply = (x: Int) => {

    x \*number;

}

  def main(args: Array[String]) {

    //number = 100;  //reassign the variable number

    println(add(20));

    println(substract(30));

    println(mulitiply(45));

    println (number);

  }

}

Output🡪

30

20

900

20

**Scala Currying**

Currying is the technique of transforming a function that takes multiple arguments into a function that takes a single argument

object currying {

  def add(x: Int, y: Int) = x + y

  def add2(x: Int) = (y: Int) => x + y;

  def add3(x: Int)(y: Int) = x + y;

  def main(args: Array[String]) {

    println(add(20, 10));

    println(add2(20)(30));

    val sum40 = add2(40);

    // println(sum40(50)(200))  // Wrong! sum40(50) = 90 → now you try to call (90)(200)? 🤯

    println(sum40(100));

    val sum50 = add3(50) \_;

    println(sum50(400));

  }

}

Output🡪

30

50

140

450

**Scala String**

object scala\_strings {

  val num1 = 75;

  val num2 = 100.25;

  val str1: String = "Hello World";

  val str2: String = " Scala!";

  def main(args: Array[String]) {

    println(str1.concat(str2)+" (with concatination)");

    println("The String Length is "+str1.length());

    println(str1 + str2 +" (without concat() method)");

    printf("(%d -- %f -- %s)", num1, num2, str1);

    val result = printf("(%d -- %f -- %s)", num1, num2, str1);

    println(result);

    println("(%d -- %f -- %s)".format(num1, num2, str1));

  }

}

Output🡪

Hello World Scala! (with concatination)

The String Length is 11

Hello World Scala! (without concat() method)

(75 -- 100.250000 -- Hello World)(75 -- 100.250000 -- Hello World)()

(75 -- 100.250000 -- Hello World)

**Scala Arrays**

import Array.\_

object scala\_arrays {

  val myArray: Array[Int] = new Array[Int](4);

  val myArray2 = new Array[Boolean](5);

  val myArray3 = Array(1, 2, 3, 4, 5, 6, 8);

  def main(args: Array[String]) {

    myArray(0) = 20;

    myArray(1) = 50;

    myArray(2) = 10;

    myArray(3) = 30;

    println(myArray); // u can get array object location

    println(myArray3.length); // Length of the Array

    concat(myArray, myArray3);

    for (x <- myArray2) {

      println(x); // Array2 Results Loop -> Default Boolean value is False

    }

    for (i <- 0 to (myArray.length - 1)) {

      println(myArray(i))

    }

  }

}

Output🡪

[I@4cdbe50f

7

false

false

false

false

false

20

50

10

30

import Array.\_

import scala.util.Random

object scala\_arrays\_second {

  val myArray = Array(20, 50, 10, 30)

  val myArray2 = Array.fill(5)(scala.util.Random.nextBoolean())

  val myArray3 = Array(1, 2, 3, 4, 5, 6, 8)

  def main(args: Array[String]) {

    // Fill myArray

    myArray(0) = 20

    myArray(1) = 50

    myArray(2) = 10

    myArray(3) = 30

    println("myArray3 length: " + myArray3.length)

    // Fill myArray2 with random true/false

    for (i <- myArray2.indices) {

      myArray2(i) = Random.nextBoolean()

    }

    println("Random Boolean Array:")

    for (x <- myArray2) {

      println(x)

    }

    println("Indexed Integer Array:")

    for (i <- 0 until myArray.length) {

      println(myArray(i))

    }

  }

}

Output🡪

myArray3 length: 7

Random Boolean Array:

false

true

false

false

true

Indexed Integer Array:

20

50

10

30

**Scala Lists**

object scala\_lists {

  val myList: List[Int] = List(1, 3, 5, 7, 9);

  val names: List[String] = List("Max", "Tom", "John", "Hardy");

  def main(args: Array[String]) {

    println(0 :: myList);

    println(myList);

    println(names);

    println(1 :: 5 :: 9 :: Nil);

    println(myList.head);

    println(names.tail);

    println(names.isEmpty);

    println(myList.reverse);

    println(List.fill(5)(2));

    println(myList.max);

    myList.foreach(println)

    var sum: Int = 0;

    myList.foreach(sum += \_);

    println(sum);

    for (name <- names) {

      println(name);

    }

    println(names(0));

  }

}

Output🡪

List(0, 1, 3, 5, 7, 9)

List(1, 3, 5, 7, 9)

List(Max, Tom, John, Hardy)

List(1, 5, 9)

1

List(Tom, John, Hardy)

false

List(9, 7, 5, 3, 1)

List(2, 2, 2, 2, 2)

9

1

3

5

7

9

25

Max

Tom

John

Hardy

Max

**Scala Sets**

object scala\_sets {

  val myset: Set[Int] = Set(1, 2, 5, 8, 9, 0, 67);

  val mutable\_myset = scala.collection.mutable.Set(3, 4, 6, 7, 9, 19, 67);

  val names: Set[String] = Set("Max", "Tom", "John");

  def main(args: Array[String]) {

    println(myset); // Default is immutable

    println(myset + 10); // Default is immutable and I need to add 10

    println(myset (5)); // In this Immutable set I need to check if 5 is included

    println(mutable\_myset); // this set is mutable

    println(mutable\_myset + 100); //if we add any value there.. it will not added to set

    println(mutable\_myset.head);

    println(mutable\_myset.tail);

    println(mutable\_myset.isEmpty);

    println(myset++mutable\_myset); //Shows all values ( If duplicate values in both sets it prevents)

    println(myset.&(mutable\_myset)); //get set intersect

    println(myset.++(mutable\_myset));

    println(myset.intersect(mutable\_myset)); //get set intersect

    println(myset.max);

    println(myset.min);

    println(names);

    println(names("Tharindu")); //In this string set I need to check if the name Tharindu is included

    myset.foreach(println)

    for(name <- names){

        println(name);

    }

  }

}

Output🡪

HashSet(0, 5, 1, 9, 2, 67, 8)

HashSet(0, 5, 10, 1, 9, 2, 67, 8)

true

HashSet(3, 19, 67, 4, 6, 7, 9)

HashSet(3, 19, 67, 4, 100, 6, 7, 9)

3

HashSet(19, 67, 4, 6, 7, 9)

false

HashSet(0, 5, 1, 6, 9, 2, 7, 3, 67, 8, 19, 4)

HashSet(9, 67)

HashSet(0, 5, 1, 6, 9, 2, 7, 3, 67, 8, 19, 4)

HashSet(9, 67)

67

0

0

5

1

9

2

67

8

Max

Tom

John

Set(Max, Tom, John)

False

**Scala Maps**

Example 1:-

object scala\_maps {

  val myMap: Map[Int, String] =

    Map(801 -> "max", 802 -> "Tom", 803 -> "Calvin", 806 -> "Misha");

  def main(args: Array[String]) {

    println(myMap);

    println(myMap(803))

    println(myMap(805))

  }

}

Output🡪

Map(801 -> max, 802 -> Tom, 803 -> Calvin, 806 -> Misha)

Exception in thread "main" java.util.NoSuchElementException: key not found: 805

Calvin

at scala.collection.immutable.Map$Map4.apply(Map.scala:535) Map.scala:535

at scala\_maps$.main(scala\_maps.scala:8) scala\_maps.scala:8

at scala\_maps.main(scala\_maps.scala)

Example 2:-

object scala\_maps {

  val myMap: Map[Int, String] =

    Map(801 -> "Max", 802 -> "Tom", 803 -> "Calvin", 806 -> "Misha");

  val myMap2: Map[Int, String] =

    Map(804 -> "Hardy", 805 -> "Julia", 807 -> "Dante");

  def main(args: Array[String]) {

    println(myMap);

    println(myMap(803))

    // println(myMap(805)) //Searching not included Int will give u errors

    println(myMap.keySet); // Defines as a set

    println(myMap.values);

    println(myMap.keys); // Defines as a set

    println(myMap.isEmpty);

    myMap.keys.foreach { key =>

      println("For this Key -> " + key + " Value is " + myMap(key));

    }

    println(myMap.contains(802))

    println(myMap ++ myMap2); //Concat 2 maps

    println(myMap.size); //size of the map

  }

}

Output🡪

Map(801 -> Max, 802 -> Tom, 803 -> Calvin, 806 -> Misha)

Calvin

Set(801, 802, 803, 806)

Iterable(Max, Tom, Calvin, Misha)

Set(801, 802, 803, 806)

false

For this Key -> 801 Value is Max

For this Key -> 802 Value is Tom

For this Key -> 803 Value is Calvin

For this Key -> 806 Value is Misha

true

HashMap(802 -> Tom, 806 -> Misha, 805 -> Julia, 804 -> Hardy, 803 -> Calvin, 801 -> Max, 807 -> Dante)

4

**Scala Tuples**

object scala\_tuples {

  val myTuples = (1, 2, "hello", true);

  val myTuples2 = new Tuple2("hello", true); // until Tuple22 only

  val myTuples3 = new Tuple3(1, "hello", ("Tom", 22));

  def main(args: Array[String]) {

    println(myTuples.\_3); // get the third value

    println(myTuples.\_2); // get the first value

    // println(myTuples2.\_4); //value \_4 is not a member... compile error

    myTuples.productIterator.foreach { i =>

      println(i);

    }

    println(1 -> "Tom" -> true); //Define a 3 element tuple in a print statement

    println(myTuples3.\_3.\_2); //Get 3rd value’s 2nd value from a 3 elements tuple

  }

}

Output🡪

hello

2

1

2

hello

true

((1,Tom),true)

22

**Scala Options (Some or Not)**

Example :- 1

object scala\_option {

  val list = List(1, 2, 4);

  val map = Map(1 -> "Tom", 2 -> "Max", 3 -> "John");

  def main (args:Array[String]){

    println(list.find(\_>2));

    println(list.find(\_>2).get);

    println(map.get(2));

    println(map.get(2).get);

    println(map.get(7).get);

  }

}

Output🡪

Some(4)

4

Some(Max)

Max

Exception in thread "main" java.util.NoSuchElementException: None.get

at scala.None$.get(Option.scala:627) Option.scala:627

at scala.None$.get(Option.scala:626) Option.scala:626

at scala\_option$.main(scala\_option.scala:10) scala\_option.scala:10

at scala\_option.main(scala\_option.scala)

Example 2:-

object scala\_option {

  val list = List(1, 2, 4);

  val map = Map(1 -> "Tom", 2 -> "Max", 3 -> "John");

  val option : Option[Int] = None;

  val option2 : Option[Int] = Some(55);

  def main (args:Array[String]){

    println(list.find(\_>2));

    println(list.find(\_>2).get); //without getOrElse() method this will show errors if not contains value comes

    println(list.find(\_>7).getOrElse(0));

    println(map.get(2));

    println(map.get(2).get);

    println(map.get(7).getOrElse("Not Found"));

    println(option.isEmpty);

    println(option.isDefined);

    println(option2.isDefined);

    println(option2.isEmpty);

    println(option2.get);

  }

}

Output🡪

Some(4)

4

0

Some(Max)

Max

Not Found

true

false

true

false

55

**Scala Advance Map**

object `map\_&\_filters` {

  val list = List(1, 2, 4, 7, 0, 3, 5);

  val map = Map(1 -> "Tom", 2 -> "Max", 3 -> "John");

  def main(args: Array[String]) {

    println(list.map(\_ \* 2)); // This will mutiply list in 2x

    println(

      list.map(x => x + 3)

    ); // This will add 3 for every value in the list

    println(

      list.map(x => x / 0.5)

    ); // This will divide every values in 0.5 of the list

    println(

      list.map(x => "new String " + x)

    ); // add a String value to every value of the list

    println(

      list.map(x => "HI " \* x)

    ); // this will multiply the "HI String from the value of specific value of the list"

    println(list.map(x => "Hello " + x));

    println(map.map(x => "Hello " + x));

 println(map.mapValues(x => "Hello " + x)); // This is deprecated in new versions

 println(map.view.mapValues(x => "Hello " + x).toMap)

    println("Hello".map(\_.toUpper)); // Transform value into Uppercase

    println(List(List(1, 2, 3), List(3, 4, 5))); // get two Lists

    println(

      List(List(1, 2, 3), List(3, 4, 5)).flatten

    ); // get two lists into one List

    println(

      list.flatMap(x => List(x, x + 1))

    ); // Get the existing list and also get a new list with +1 added, it also flatten the map

  }

}

Output🡪

List(2, 4, 8, 14, 0, 6, 10)

List(4, 5, 7, 10, 3, 6, 8)

List(2.0, 4.0, 8.0, 14.0, 0.0, 6.0, 10.0)

List(new String 1, new String 2, new String 4, new String 7, new String 0, new String 3, new String 5)

List(HI , HI HI , HI HI HI HI , HI HI HI HI HI HI HI , , HI HI HI , HI HI HI HI HI )

List(Hello 1, Hello 2, Hello 4, Hello 7, Hello 0, Hello 3, Hello 5)

List(Hello (1,Tom), Hello (2,Max), Hello (3,John))

MapView(<not computed>)

Map(1 -> Hello Tom, 2 -> Hello Max, 3 -> Hello John)

HELLO

List(List(1, 2, 3), List(3, 4, 5))

List(1, 2, 3, 3, 4, 5)

List(1, 2, 2, 3, 4, 5, 7, 8, 0, 1, 3, 4, 5, 6)

**Scala Filters**

object scala\_filters {

  val list = List(1, 2, 4, 7, 0, 3, 5);

  val map = Map(1 -> "Tom", 2 -> "Max", 3 -> "John");

  def main(args: Array[String]) {

    println(list.filter(x => x % 2 == 0)); // even numbers

    println(list.filter(x => x % 2 != 0)); // odd numbers

// ✅ 1. Filter numbers greater than a specific value

    println(list.filter(\_ > 3)) // numbers greater than 3

// ✅ 2. Filter zero and non-zero elements

    println(list.filter(\_ == 0)) // only zeros

    println(list.filter(\_ != 0)) // non-zero elements

// ✅ 3. Filter names from map with specific conditions

    println(map.filter { case (k, v) =>

      v.startsWith("J")

    }) // names starting with J

// ✅ 4. Filter keys from the map

    println(map.filterKeys(\_ > 1)) // keys greater than 1 -> This is deprecated in new versions

    println(map.view.filterKeys(\_ > 1).toMap) // keys greater than 1

//✅ 5. Combine filters (chaining conditions)

    println(

      list.filter(x => x > 2 && x % 2 == 0)

    ) // even numbers greater than 2

//✅ 6. Use ``partition`` to split based on a condition

    val (evens, odds) = list.partition(\_ % 2 == 0)

    println("Evens: " + evens)

    println("Odds: " + odds)

// ✅ 7. Get values from ``map`` with keys present in ``list``

    val filteredNames = list.flatMap(map.get)

    println(filteredNames)

//✅ 8. Use ``collect`` to filter and transform at once

    println(list.collect { case x if x % 2 == 0 => s"Even: $x" })

//✅ 9. Filter based on index (e.g., elements at even indices)

    println(

      list.zipWithIndex.filter { case (\_, idx) => idx % 2 == 0 }.map(\_.\_1)

    )

  }

}

Output🡪

List(2, 4, 0)

List(1, 7, 3, 5)

List(4, 7, 5)

List(0)

List(1, 2, 4, 7, 3, 5)

Map(3 -> John)

MapView(<not computed>)

Map(2 -> Max, 3 -> John)

List(4)

Evens: List(2, 4, 0)

Odds: List(1, 7, 3, 5)

List(Tom, Max, John)

List(Even: 2, Even: 4, Even: 0)

List(1, 4, 0, 5)

**Scala Reduce Function**

object scala\_reduce\_fold\_scan {

  val list1 = List(1, 3, 5, 6, 9, 0, 2);

  val list2 = List("Max ", "is ", "very ", "good");

  def main(args:Array[String]){

    println(list1.reduceLeft(\_+\_));

    println(list2.reduceLeft(\_+\_));

    println(list1.reduceLeft((x,y)=>{println(x + " , "+y); x+y;}));

    println(list1.reduceLeft(\_-\_));

    println(list1.reduceRight(\_-\_));

    println(list1.reduceRight((x,y)=>{println(x + " , "+y); x-y;}));

  }

}

Output🡪

26

Max is very good

1 , 3

4 , 5

9 , 6

15 , 9

24 , 0

24 , 2

26

-24

8

0 , 2

9 , -2

6 , 11

5 , -5

3 , 10

1 , -7

8

**Scala Fold Function**

object scala\_reduce\_fold\_scan {

  val list1 = List(1, 3, 5, 6, 9, 0, 2);

  val list2 = List("Max ", "is ", "very ", "good");

  def main(args: Array[String]) {

    println(

      list1.foldLeft(0)(\_ + \_)); // this value is same as reduceLeft()...

    // But thing is we pass the initial value 0 which means 0 is added to the sum

    println(

      list1.foldLeft(100)(\_ + \_)); // Now we can see the difference

    println(

      list2.foldLeft("This ")(\_ + \_));

  }

}

Output🡪

26

126

This Max is very good

**Scala Scan Function**

object scala\_reduce\_fold\_scan {

  val list1 = List(1, 3, 5, 6, 9, 0, 2);

  val list2 = List("Max ", "is ", "very ", "good");

  def main(args: Array[String]) {

    println(list1.scanLeft(100)(\_+\_)); //scan method is giving the map of intermediate results

    println(list2.scanLeft("Scan ")(\_+\_));

  }

}

Output🡪

List(100, 101, 104, 109, 115, 124, 124, 126)

List(Scan , Scan Max , Scan Max is , Scan Max is very , Scan Max is very good)