**Values and Variables**

**1. Create 2 val types with x as 100 & y as 10 respectively and find the Multiplication and division**

**of both and store in some val as z and z1.**

scala> :paste

val x=100;

val y=10;

val z=x\*y;

val z1=x/y;

println("Multiplication:",z);

println("Division:",z1);

// Exiting paste mode, now interpreting.

(Multiplication:,1000)

(Division:,10)

**2. Create a as 2000 and find the division of a by y created in step 1 and reassign a with the divided**

**result (200).**

scala> :paste

// Entering paste mode (ctrl-D to finish)

var a=2000;

a=a/y;

println("Value of a:",a);

// Exiting paste mode, now interpreting.

(Value of a:,200)

a: Int = 200

a: Int = 200

**3. Create a val type with x:Int=100, then assign the x to val y, but the datatype of y has to be String.(think about using some function like toString)**

scala> :paste

// Entering paste mode (ctrl-D to finish)

val x:Int=100;

val y:String=x.toString();

println("Value of y:",y);

// Exiting paste mode, now interpreting.

(Value of y:,100)

x: Int = 100

y: String = 100

**4. Try only in REPL for now - Create a val type sc1 and assign sc into it and also try assigning sc1**

**defined as AnyRef/Any and check the type of the sc1 using getClass function.**

The **getClass()** method is utilized to return the class of the given number.

**Conditional Structures**

**6. Write a program to find the greatest of 3 numbers**

scala> :paste

// Entering paste mode (ctrl-D to finish)

println("Enter the value of x:")

val x = scala.io.StdIn.readLine()

println("Enter the value of y:")

val y = scala.io.StdIn.readLine()

println("Enter the value of z:")

val z = scala.io.StdIn.readLine()

if(x>y)

{

if(x>z)

{

println("X is greater")

}

else

{

println("Z is greater")

}

}

else if(y>z)

{

println("Y is greater")

}

else

{

println("Z is greater")

}

// Exiting paste mode, now interpreting.

Enter the value of x:

Enter the value of y:

Enter the value of z:

Y is greater

x: String = 101880

y: String = 80

z: String = 20

**7. Write a nested if then else to print the course fees of if the student chooses bigdata then check if bigdata then fees is 25000, if spark then fees is 15000, if the student chooses datascience then check if machinelearning then 35000, if deep learning then 45000.**

var course:String = "";

var fee:Int = 0;

println("Enter the course:");

course = scala.io.StdIn.readLine();

if(course.toUpperCase == "BIG DATA")

{

fee=25000;

}

else if(course.toUpperCase == "SPARK")

{

fee=15000;

}

else if(course.toUpperCase == "DATA SCIENCE")

{

course = scala.io.StdIn.readLine();

if(course.toUpperCase == "MACHINE LEARNING")

{

fee=35000;

}

else

if(course.toUpperCase == "DEEP LEARNING")

{

fee=45000;

}

}

println("Fees:"+fee);

// Exiting paste mode, now interpreting.

Enter the course:

Fees:45000

course: String = deep learning

fee: Int = 45000

course: String = deep learning

**8. Check whether the given string is palindrome or not (try to use some function like reverse). For**

**eg: val x="madam" then print as "palindrome" else "non palindrome".**

scala> :paste

// Entering paste mode (ctrl-D to finish)

println("Enter the string:")

var str1:String = scala.io.StdIn.readLine()

var str2:String = **str1.reverse**

if (**str1.toUpperCase == str2.toUpperCase**)

println("Palindrome")

else

println("Not palindrome")

// Exiting paste mode, now interpreting.

Enter the string:

Not palindrome

str1: String = muthu

str2: String = uhtum

**9. Check whether the val x=100 is an integer or string. (try to use some functions like toString,**

**toUpperCase etc to execute this use case)**

def chkDatatype(value:Any) =

{

var res = value.**asInstanceOf**[Any].**getClass.getSimpleName**;

if(res == "Integer")

{

println("Datatype returned is Integer")

}

else if(res == "String")

{

println("Datatype returned is string")

}

else

{

println("Datatype returned is NOT an Integer/String")

}

}

// Exiting paste mode, now interpreting.

chkDatatype: (value: Any)Unit

scala> chkDatatype(10);

Datatype returned is Integer

scala> chkDatatype(10.00);

Datatype returned is NOT an Integer/String

scala> chkDatatype("10.00");

Datatype returned is string

**11. For loop to increment from 0 till 21 with the increment of 3, the result should be exactly**

0,3,6,9,12,15,18

scala> :paste

// Entering paste mode (ctrl-D to finish)

for(i <- 0 until 21 if i%3==0)

{

println(i)

}

// Exiting paste mode, now interpreting.

0

3

6

9

12

15

18

**12. Write a for or while loop to print the cube of 4, result should be 4\*4\*4=64 (think of using some**

**var type initiated outside the loop)**

scala> :paste

// Entering paste mode (ctrl-D to finish)

def cubeOfNo(x:Int) = {

var i=0; var j=0; var k=0;

var result=0;

for (I <- 1 to x ;j <- 1 to x; k <- 1 to x){

if (i==x && j==x && k==x) {

result = i\*j\*k;

}

}

println(result)

}

// Exiting paste mode, now interpreting.

cubeOfNo: (x: Int)Unit

scala> cubeOfNo(10)

1000

**13. Write for/while loop for printing only the values in the range of 1 to 20 which**

**are divisible by 4(don’t use by 4 in the for loop) rather use if condition to check the % of 4 for every element in the loop achieve this.**

**Result should be exactly like this 4,8,12,16,20.**

scala> :paste

// Entering paste mode (ctrl-D to finish)

for(i <- 1 to 20 if i%4 == 0)

{

println(i)

}

// Exiting paste mode, now interpreting.

4

8

12

16

20

**15. Write a method to create a calculator accepts 3 arguments and return type of any , first 2 of**

**integer and 3rd one is String, based on the 3rd argument value as add/sub/div/mul perform either**

**addition or subraction or multiplication or division of values and return the result to the calling**

**environment. (for division think of using.toFloat or .toDouble conversion).**

def calculator(a:Int,b:Int,op:String):Any =

{

**op match**

{

**case "add" | "addition" =>** {

println("Addition")

a+b

}

**case "sub" | "subtract" =>** {

println("Subtract")

a-b

}

**case "mul" | "multiplication" =>** {

println("Multiply")

a\*b

}

**case "div" | "division" =>** {

println("Division")

a.toFloat/b.toFloat

}

**case \_ =>** {

println("Operation not match")

"Not match"

}

}

**Output:**

scala> calculator(5,2,"sub");

Subtract

res2: Any = 3

scala> :paste

val bonusfunc = (salary:Int) => salary \* 0.03

def wage(salary:Int):Double =

{

salary+bonusfunc(salary)

}

bonusfunc: Int => Double = <function1>

wage: (salary: Int)Double

**Output:**

scala> wage(100)

res3: Double = 103.0

**16. Try multiple return statements in a method and identify which one is really returning and what**

**are the returns are ignored.**

def returnCheck(rent:Int, maint:Double):Double = {

val fullrent = rent+(rent\*maint).toDouble

val fixedmain:Double = 2500.00

return fullrent

fixedmain

}

scala> returnCheck(10000,0.10)

res4: Double = 11000.0

**17. Try creating a method with multiple return types.**

scala> :paste

// Entering paste mode (ctrl-D to finish)

def retrunCheck2(rent:Int,maint:Double):(Int,Double)= {

val fullrent = rent+(rent\*maint).toDouble

val fixedmain:Double = 2500.00

return(rent,fullrent)

}

// Exiting paste mode, now interpreting.

retrunCheck2: (rent: Int, maint: Double)(Int, Double)

scala> retrunCheck2(10000,0.10)

res5: (Int, Double) = (10000,11000.0)

**Pattern matching**

**18. Write a program using case using pattern matching to find the datatype o f a given value and**

**return either Float or string or Boolean or Char etc..**

def chkDataType(input:Any) =

{

var res = input.asInstanceOf[Any].getClass.getSimpleName;

**res match** {

**case "Integer" =>** { println("Datatype returned is Integer") }

**case "String" =>** { println("Datatype returned is String") }

**case "Float" =>** { println("Datatype returned is Float") }

**case "Double" =>** { println("Datatype returned is Double") }

**case "Boolean" =>** {println("Datatype returned is Boolean") }

**case \_ =>** { println("Unknown Datatype") }

}

}// Exiting paste mode, now interpreting.

chkDataType: (input: Any)Unit

**def finddatatype[T](v: T**) =

v match {

case \_: Int => println(v + " is Integer")

case \_: Float => println(v + " is Float")

case \_: String => println(v + " is String")

case \_: Char => println(v + " is Char")

case \_ => println("Its Invalid/Unknown")

}

**Output**

scala> chkDataType(100.5);

Datatype returned is Double

scala> chkDataType(100);

Datatype returned is Integer

scala> chkDataType(true);

Datatype returned is Boolean

scala> chkDataType(false);

Datatype returned is Boolean

scala> chkDataType(0);

Datatype returned is Integer

**Collections: Seq, Array, List, Map and Set**

**20. Create an array, list and prove mutability and immutability and non resizable properties.**

scala> :paste

var ar1= Array[Int](10,20,30,40)

ar1(4)=50

ar1(0)=100 - Mutable

//ar1.add(200)->Non-resizable

println("Not accessible:"+ar1(4))

println("Mutable:"+ar1(0))

val lst = List(10,20,30,40)

lst.add(333) lst(0) = 100

**21. Create arraybuffer from scala.collection.mutable package and prove mutability and**

**immutability and resizable properties.**

var arrbuf=scala.collection.mutable.ArrayBuffer(10,20,30)

arrbuf(0)=100

arrbuf=arrbuf+=40

arrbuf=arrbuf-30

var lstbuf=scala.collection.mutable.ListBuffer(10,20,30)

lstbuf(0)=100

lstbuf=lstbuf+=40

lstbuf=lstbuf-30

**22. Create a tuple of 4 fields and access the 2nd and 4 th fields and store in another tuple.**

scala> :paste

val emp = (101,"Muthu",200.90,("New street","chennai","TN"))

val empTup = (emp.\_2,emp.\_4)

println(emp)

println(empTup)

**output:**

(101,Muthu,200.9,(New street,chennai,TN))

(Muthu,(New street,chennai,TN))

emp: (Int, String, Double, (String, String, String)) = (101,Muthu,200.9,(New street,chennai,TN))

empTup: (String, (String, String, String)) = (Muthu,(New street,chennai,TN))

**23. Find the maximum value out of (2,3,1,5,4) elements in the array**

scala> :paste

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

println(arr.max)

**output:**

5

**24. Find the max and min value of (2,3,1,5,4) elements in the array and store these 2 values in another array.**

scala> :paste

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

val minarr = **arr.min**

val maxarr = **arr.max**

println("Minimum="+minarr)

println("Maximum="+maxarr)

val newarr = Array(minarr,maxarr)

newarr.foreach(println)

**output:**

Minimum=1

Maximum=5

Newarray=[I@622314c2

arr: Array[Int] = Array(2, 3, 1, 5, 4)

minarr: Int = 1

maxarr: Int = 5

newarr: Array[Int] = Array(1, 5)

**25. Create a method to find the highest value in the given array if the array is non empty and print it, you must pass array as an argument to the method.**

scala> :paste

// Entering paste mode (ctrl-D to finish)

def BigNoInArray(x:Int\*):Int = {

if(x.isEmpty){

return 0

}else{

return x.max

}

}

// Exiting paste mode, now interpreting.

BigNoInArray: (x: Int\*)Int

scala> BigNoInArray()

res6: Int = 0

scala> BigNoInArray(3,5,6,8,1)

res7: Int = 8

**26. Write a program to create an Int List with 5 different values using range and sum all the values**

scala> :paste

var lst = **List.range(1,10)**

lst.foreach(println)

val sum = **lst.sum**

println("Sum="+sum)

**output:**

1

2

3

4

5

6

7

8

9

Sum=45

lst: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8, 9)

sum: Int = 45

**27. Write a program to create string list to store the values of Spark,Scala,Python,Java,Hadoop and count the number of elements in the List**

scala> :paste

var lang = List("Spark","Scala","Python","Java","Hadoop")

val count = **lang.size**

lang.foreach(println)

println("Count="+count)

**output:**

Spark

Scala

Python

Java

Hadoop

Count=5

lang: List[String] = List(Spark, Scala, Python, Java, Hadoop)

count: Int = 5

**28. Write a program to store (China,Beijing),(India,New Delhi),(USA,Washington),(UK,London) using Map**

scala> :paste

var cap = Map("China"->"Beijing","India"->"New Delhi","USA"->"Washington","UK"->"London")

println(cap**)**

**output:**

Map(China -> Beijing, India -> New Delhi, USA -> Washington, UK -> London)

cap: scala.collection.immutable.Map[String,String] = Map(China -> Beijing, India -> New Delhi, USA -> Washington, UK -> London)

**29. Find the capital of India**

scala> cap("India")

**output:**

res1: String = New Delhi

**30. Take only countries and store in an array and use foreach and println to print line by line of elements.**

scala> :paste

var cap = scala.collection.mutable.Map("China"->"Beijing","India"->"New Delhi","USA"->"Washington","UK"->"London")

var country = **cap.keys.toArray**

country.foreach(println)

**output:**

China

India

UK

USA

cap: scala.collection.mutable.Map[String,String] = Map(China -> Beijing, India -> New Delhi, UK -> London, USA -> Washington)

country: Array[String] = Array(China, India, UK, USA)

**31. Take only countries and store in an set and use foreach and println to print line by line of elements**.

scala> :paste

var cap = scala.collection.mutable.Map("China"->"Beijing","India"->"New Delhi","USA"->"Washington","UK"->"London")

**var country = cap.keys.toSet**

**country.foreach(x=>println(x+"\n"))**

**output:**

China

India

UK

USA

cap: scala.collection.mutable.Map[String,String] = Map(China -> Beijing, India -> New Delhi, UK -> London, USA -> Washington)

country: Iterable[String] = Set(China, India, UK, USA)

**OOPS**

**35. Inside the class mask create a private val as addhash=100 and a method hashMask(str:String):Int={return the hashcode of str+addhash value}**

package com.scala.we35.workhome.datasecurity

class mask {

private val addhash = 100;

def hashMask(str: String): Int =

{

return str.**hashCode()** + 100;

}

}

**36. Inside the class mask create a private val as prefixstr=prefixstr=”aix” and a method**

**revEncode (str: String ={return the prefixstr+reverse of str value}**

package com.scala.we35.workhome.datasecurity

class endecode {

private val prefixstr = "aix";

def revEncode(str: String): String =

{

return prefixstr + str.reverse;

}

def revDecode(str: String): String =

{

return prefixstr + str.reverse;

}

}

**36 to 40:**

package com.scala.we35.workhome.datasecurity

object singleobject {

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

{

val objmask = new com.scala.we35.workhome.datasecurity.mask;

val objencode = new com.scala.we35.workhome.datasecurity.endecode;

val arrname = Array("Krish", "Rengnathan", "REYAANSH", "murthy")

arrname.foreach(println);

//loop the array elements, apply hashMask(name) for all 3 elements and println of the masked values.

println("Masked values.");

for (name <- arrname)

println(objmask.hashMask(name));

println("Loop the array created the revEncode(name) for all 3 elements");

val revEncodearrbf = scala.collection.mutable.ArrayBuffer();

for (name <- arrname)

println(objencode.revEncode(name))

println("Loop the array created the revDecode(name) for all 3 elements");

for (name <- arrname)

println(objencode.revDecode(objencode.revEncode(name)));

}

}

**Counting words starts with ‘h’ ,’H’**

scala> :paste

// Entering paste mode (ctrl-D to finish)

def countwords(in:String) {

val input = in

val arrhw = new scala.collection.mutable.ArrayBuffer[String]()

for(i<-input.split(" "))

{

var res = i.charAt(0)

if(res.equals('h') || res.equals('H')) {

arrhw += i

}

}

arrhw.foreach(println)

}

// Exiting paste mode, now interpreting.

countwords: (in: String)Unit

scala> countwords("Hello hi you do how")

Hello

hi

how