**Academic Year 2024-25 Even**

**19CSE313 – Principles of Programming Language**

**B.Tech CSE 2022-26 F Section**

**Practice Set 7 – Scala Introduction**

1. Install Scala in your System
   1. Use the following URL for appropriate versions - <https://www.scala-lang.org/download/>
   2. Go to the command prompt, type scala, and press enter (to install any missing files)
   3. To come out of the Scala prompt, use **:q**
   4. Check the version - typing ‘scala -version’ in the command prompt
2. Writing and executing a program
   1. Create a file with extension .scala
   2. Type the program
   3. In the command prompt, type ‘scala pgmname.scala’
3. First program – “Hello Scala”
   1. Code 1 – scala1.scala

object scala1

{

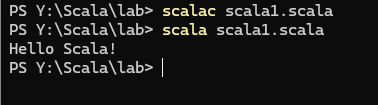
 def main(args:Array[String])=

{

println("Hello Scala!")

}

}

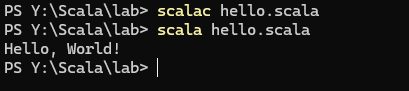


* 1. Code 2 - hello.scala

@main

def hello(): Unit =

println("Hello, World!")



* 1. **Your Code**

1. A program with two functions

object factorial{

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

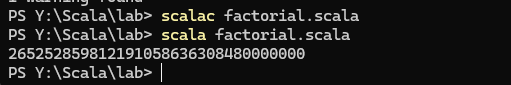
println(factorial(30));

}

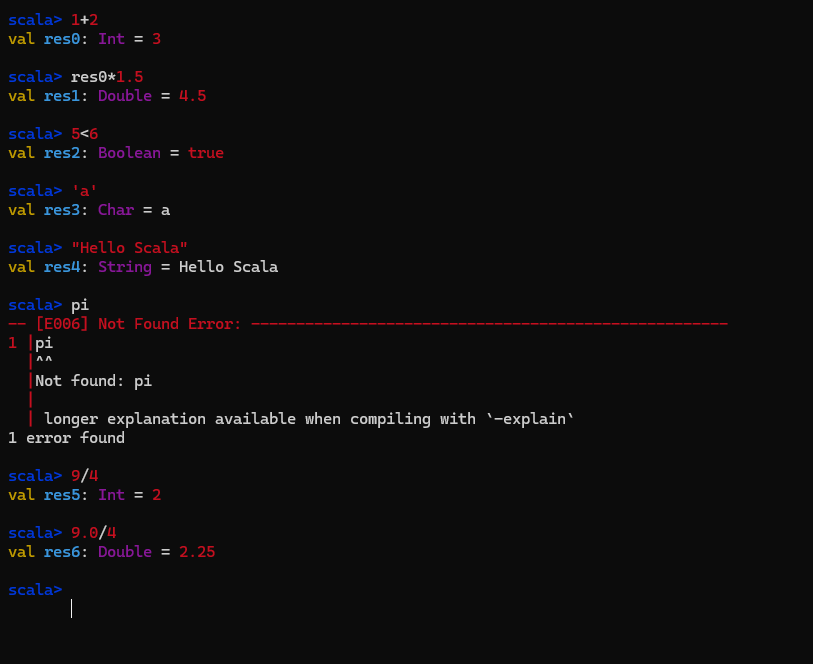
def factorial(x: BigInt): BigInt =

if (x == 0) 1 else x \* factorial(x - 1)

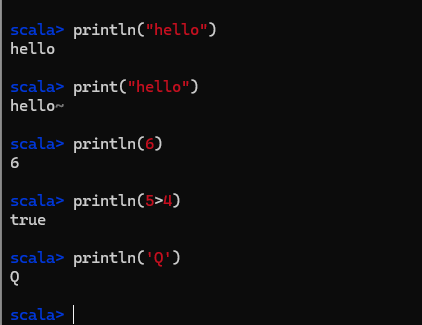
}



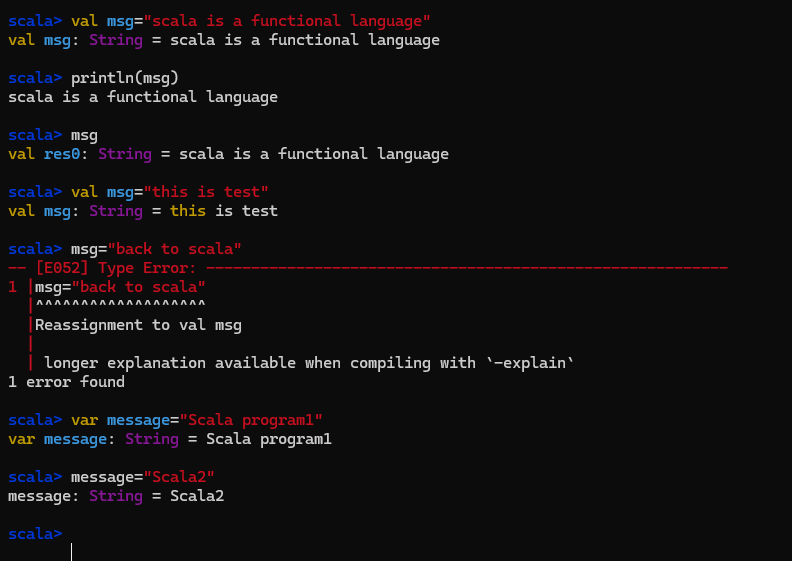
1. Using Scala prompt – type ‘scala’ in command prompt and press enter
   1. scala>1+2
   2. scala>res0\*1.5
   3. scala>5<6
   4. scala> ‘a’
   5. scala> “Hello Scala”
   6. scala>pi
   7. scala>9/4
   8. scala>9.0/4



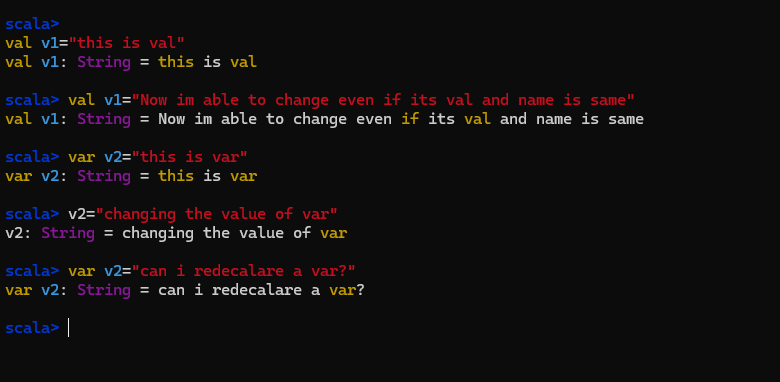
1. What are the data types that you encountered?
2. Printing on screen
   1. println(“Hello”)
   2. print(“Hello”)
   3. println(6)
   4. println(5>4)
   5. println(‘Q’)



1. val and var
   1. val msg = "Scala is a functional language"
   2. println(msg)
   3. scala>msg
   4. val msg="This is test"
   5. msg = “Back to Scala”
   6. var message = “ Scala Program 1”
   7. message = “Scala 2”



1. Try these statements:
   1. scala> val v1 = "This is a val"
   2. scala>val v1 = "Now I am able to change even if its val and name is same"
   3. scala>val v1 = "Now I am able to change even if its val and name is same"
   4. scala> var v2 = "This is a var"
   5. scala>v2 = "Changing the value of var"
   6. var v2 = "Can I redeclare a var?"



1. **What are your observations about ‘val’ and ‘var’?**

**Val is immutable but can be changed while redeclaring**

**Var is mutable and re-declare**

1. **Typing simply the val/var variable names on the scala prompt leads to value of variable being printed**

**x:Int=10**

1. Defining non-recursive functions
   1. Defining

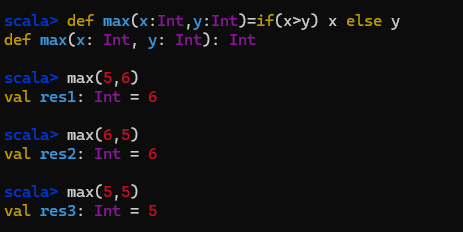
scala>def max(x: Int, y: Int) = if (x > y) x else y

* 1. Calling

scala>max(5,6)

scala>max(6,5)

scala>max(5,5)



1. Another way of writing function definitions in multiple lines (press enter key after each line)

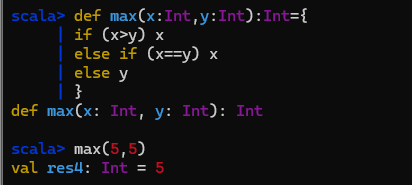
scala> def max(x:Int,y:Int):Int={

| if (x>y) x

| else if (x==y) x

| else y

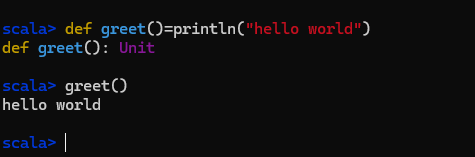
| }



1. A function that does not return anything

scala> def greet() = println("Hello, world!")

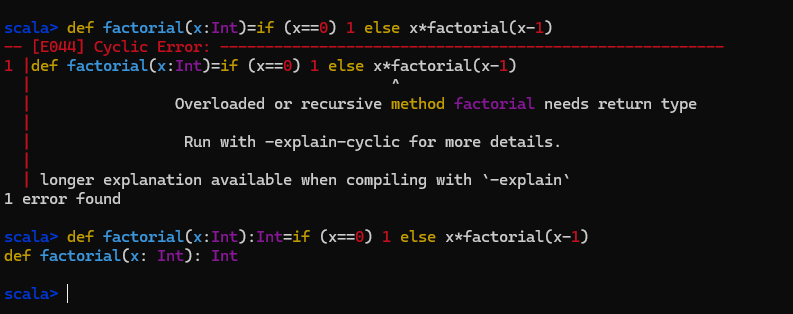
scala> greet()



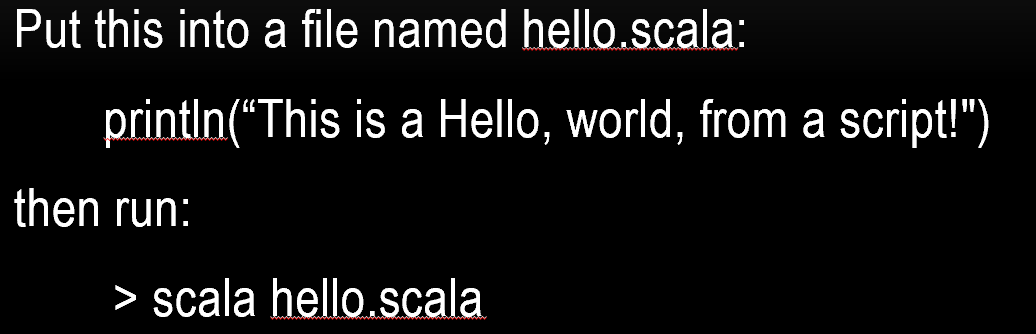
1. What are your observations about the following two function definitions?

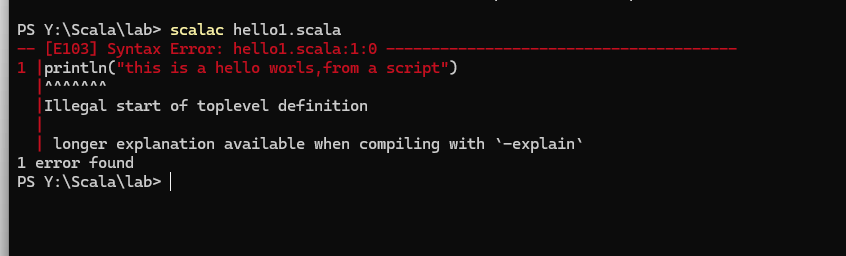
scala>def factorial(x:Int) = if (x == 0) 1 else x \* factorial(x - 1)

scala>def factorial(x:Int):Int = if (x == 0) 1 else x \* factorial(x - 1)



1. Try this





1. Loop – printing 1 to N

object cmdCheck{

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

check(5);

}

def check(args:Int): Unit = {

var i=1;

while (i <= args) {

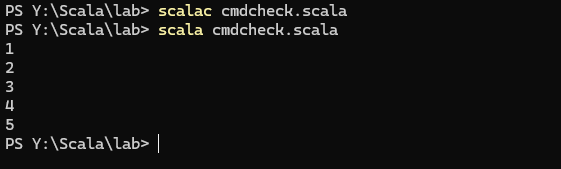
println(i)

i += 1

}

}

}



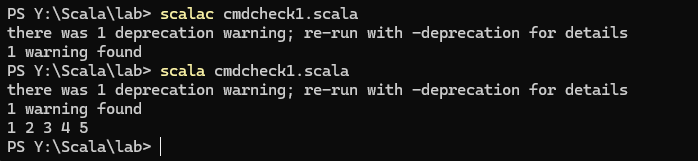
1. Change the while part of previous program as follows. What do you observe?

while (i <= args) {

print(i+" ")

i += 1

}



1. Explore the ‘for’ loop in Scala and re-write 17 using ‘for’ construct

object cmdcheck{

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

check(5);

}

def check(args:Int):Unit={

var i=1;

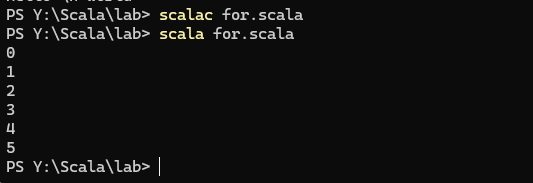
for(i<-0 to args){

println(i)

}

}

}



**FOREACH:**

object cmdCheck{

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

check(5)

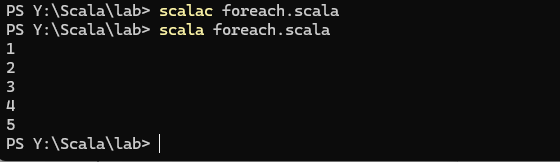
}

def check(args:Int): Unit = {

(1 to args).foreach(i => println(i))

}

}



1. CASE

def main(args:Array[String])=

{

val age1=20;

val age="50";

age1 match {

case 20 => println(age1);

case 18 => println(age1);

case 30 => println(age1);

case 40 => println(age1);

case 50 => println(age1);

case \_ => println("Default");

}

val result = age match {

case "20" => age;

case "18" => age;

case "30" => age;

case "40" => age;

case "50" => age;

case \_ => println("Default"); }

println("result=" + result);

val i=7;

i match {

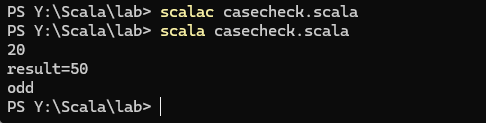
 case 1 | 3 | 5 | 7 | 9 => println("odd");

 case 2 | 4 | 6 | 8 | 10 => println("even");

}

}

}



1. Try the following code and note down your observations

object strintrp{

 def main(args:Array[String])={

val name = "john"

val age = 21

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

  println(s"$name is $age years old")

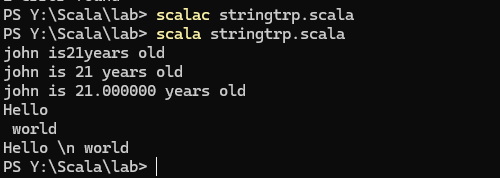
println(f"$name%s is $age%f years old")

println(s"Hello \n world")

println(raw"Hello \n world")

}

}



22.write a function in scala with on main function and others keep performing operations in a calculator (+,-,/,\*) until the user gives NO-and it exits(using a while loop)

import scala.io.StdIn.\_

object CalculatorApp {

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

println("Welcome to the Scala calculator!")

// Calling the function that will keep asking for operations

performOperations()

}

// Function to keep performing operations until "no" is entered

def performOperations(): Unit = {

var continue = true

while (continue) {

println("\nSelect an operation:")

println("1. Add")

println("2. Subtract")

println("3. Multiply")

println("4. Divide")

println("Enter 'no' to exit.")

val choice = readLine().toLowerCase

// Perform the selected operation

choice match {

case "1" => addNumbers()

case "2" => subtractNumbers()

case "3" => multiplyNumbers()

case "4" => divideNumbers()

case "no" =>

println("Exiting the calculator.")

continue = false // Stop the loop if "no" is entered

case \_ => println("Invalid option, please try again.")

}

}

}

// Addition function

def addNumbers(): Unit = {

println("Enter the first number:")

val num1 = readDouble()

println("Enter the second number:")

val num2 = readDouble()

println(s"Result: ${num1 + num2}")

}

// Subtraction function

def subtractNumbers(): Unit = {

println("Enter the first number:")

val num1 = readDouble()

println("Enter the second number:")

val num2 = readDouble()

println(s"Result: ${num1 - num2}")

}

// Multiplication function

def multiplyNumbers(): Unit = {

println("Enter the first number:")

val num1 = readDouble()

println("Enter the second number:")

val num2 = readDouble()

println(s"Result: ${num1 \* num2}")

}

// Division function

def divideNumbers(): Unit = {

println("Enter the first number:")

val num1 = readDouble()

println("Enter the second number:")

val num2 = readDouble()

if (num2 != 0) {

println(s"Result: ${num1 / num2}")

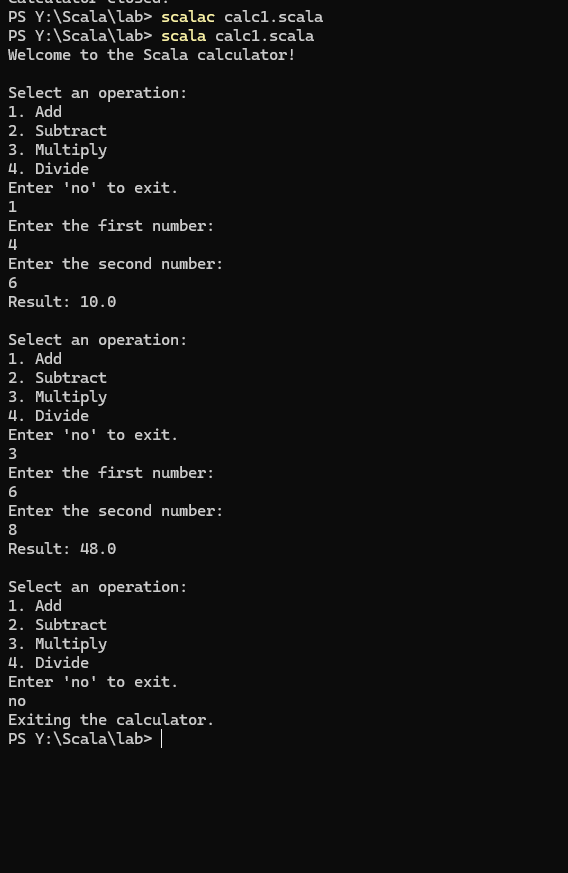
} else {

println("Error: Cannot divide by zero.")

}

}

}



import scala.io.StdIn.readLine

object Calculator {

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

var continue = true

while (continue) {

println("Enter the first number:")

val num1 = readLine().toDouble

println("Enter the operation (+, -, \*, /):")

val operation = readLine()

println("Enter the second number:")

val num2 = readLine().toDouble

operation match {

case "+" => println(s"Result: ${num1 + num2}")

case "-" => println(s"Result: ${num1 - num2}")

case "\*" => println(s"Result: ${num1 \* num2}")

case "/" =>

if (num2 != 0) {

println(s"Result: ${num1 / num2}")

} else {

println("Error: Division by zero is not allowed.")

}

case \_ => println("Invalid operation. Please try again.")

}

println("Do you want to perform another operation? (YES/NO)")

val userInput = readLine().toUpperCase

if (userInput == "NO") {

continue = false

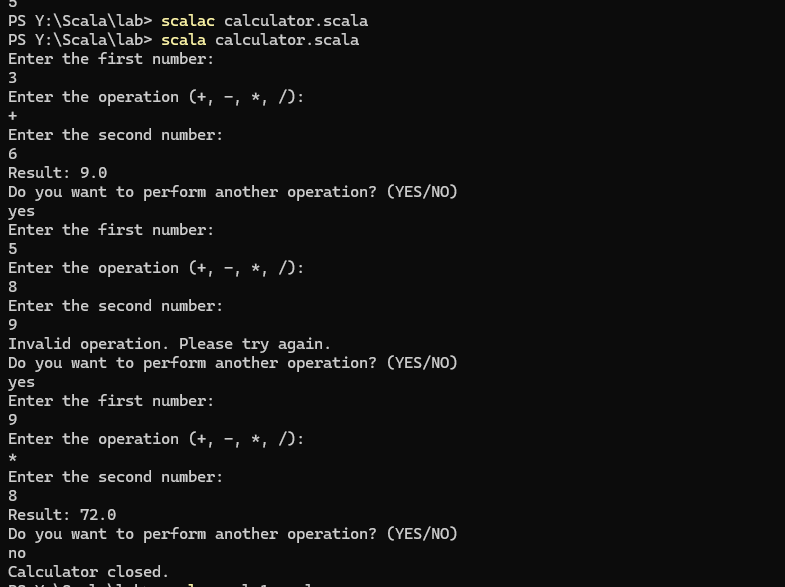
}

}

println("Calculator closed.")

}

}



(UPDATED CALCULATOR CODE)

import scala.io.StdIn.readLine

object Calculator {

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

var continue = true

while (continue) {

println("Enter an expression (e.g., 5+6):")

val expression = readLine()

// Use regular expression to split the input based on operator

val pattern = "([0-9.]+)([+\\-\*/])([0-9.]+)".r

expression match {

case pattern(num1Str, operator, num2Str) =>

val num1 = num1Str.toDouble

val num2 = num2Str.toDouble

operator match {

case "+" => println(s"Result: ${num1 + num2}")

case "-" => println(s"Result: ${num1 - num2}")

case "\*" => println(s"Result: ${num1 \* num2}")

case "/" =>

if (num2 != 0) {

println(s"Result: ${num1 / num2}")

} else {

println("Error: Division by zero is not allowed.")

}

}

case \_ => println("Invalid expression. Please try again.")

}

println("Do you want to perform another operation? (YES/NO)")

val userInput = readLine().toUpperCase

if (userInput == "NO") {

continue = false

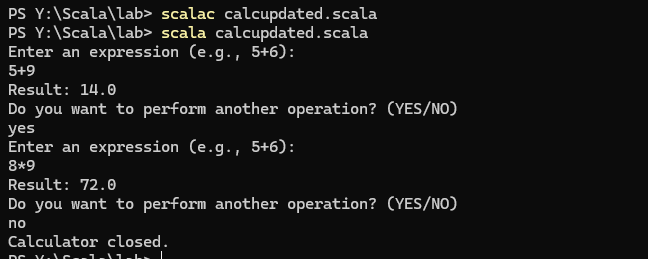
}

}

println("Calculator closed.")

}

}



DNA

object Distance {

def compute(strand1: String, strand2: String): Int = {

require(strand1.length == strand2.length, "DNA lengths must be the same")

strand1.zip(strand2).count { case (char1, char2) => char1 != char2 }

}

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

val dna1 = "GAGCCTACTAACGGGAT"

val dna2 = "CATCGTAATGACGGCCT"

println(s"Distance: ${compute(dna1, dna2)}")

}

}

