**Academic Year 2024-25 Even**

**19CSE313 – Principles of Programming Language**

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

**Practice Set 11 – Recursion and Pattern Matching in Scala**

1. **Factroial**

object Demo {

   def main(args: Array[String]) = {

      for (i <- 1 to 10)

         println( "Factorial of " + i + ": = " + factorial(i) )

   }

   def factorial(n: BigInt): BigInt = {

      if (n <= 1)

         1

      else

      n \* factorial(n - 1)

   }

}

1. **GCD**

scala> def gcd(x:Long,y:Long):Long=if(y==0) x else gcd(y,x%y)

scala> gcd(206,40)

scala> gcd(8,40)

scala> gcd(5,7)

1. **Stack overflow**

scala> def sum(num:Int):Int={

| if (num==1) 1

| else sum(num-1)+num

| }

scala> sum(9)

scala> sum(99)

scala> sum(999)

scala> sum(9999)

scala> sum(99999)

1. **Tail Recursion**

scala> def sum1(num:Int,res:Int):Int={

| if (num==1) res

| else sum1(num-1, res+num)

| }

scala> sum1(99999,1)

1. **Checking if the recursion is tail-recursive**

scala> import scala.annotation.\_

scala> @tailrec

| def sum(num:Int):Int={

| if (num==1) 1

| else sum(num-1)+num

| }

scala> @tailrec

| def sum1(num:Int,res:Int):Int={

| if (num==1) res

| else sum1(num-1, res+num)

| }

1. **Factorial using pattern matching**

scala> val fact:(Int=>Int)=(n)=>{n match{

| case 0 =>1

| case m => m\*fact(m-1)

| }}

scala> fact(5)

1. **Wild Card Pattern**

scala> def wildCardPattern(value:Any):Unit=

| value match{

| case \_ => println("Wildcard Pattern can match anything")

| }

scala> wildCardPattern(value = "Who am I?")

scala> wildCardPattern(value = 12345)

1. **Constant value pattern**

scala> def describe(x:Any) = x match{

| case 5 => "Five"

| case true => "True"

| case "Hello" => "Hello"

| case Nil => "Empty List"

| case \_ => "Something else"

| }

scala> describe(5)

scala> describe(true)

scala> describe("Hello")

scala> describe()

scala> describe([])

scala> describe(Nil)

scala> describe(false)

1. **Variable Pattern**

scala> def variablePattern(name:String):Unit=

| name match{

| case n => println(s"My name is $n")

| }

scala> variablePattern("Scala")

1. **Matching a sequence**

scala> def seqPattern(expr:Any)= expr match{

| case List(0,\_,\_)=> println("Found a list with 3 elements whose first value is a Zero")

| case \_ => println("The value is something else!")

| }

scala> seqPattern(List(1,2,3))

scala> seqPattern(List(0,2,3))

scala> seqPattern(List(0,2,3,4))

1. **Tuple Pattern**

scala> def TuplePattern(variable:Any) = variable match{

| case (\_,"India")=>println("Somebody or Somthing from India")

| case ("Camel",\_)=>println("Camel from somewhere"

)

| case \_ => println("Unknown data")

| }

scala> TuplePattern("Tiger","India")

scala> TuplePattern("Camel","UAE")

scala> TuplePattern("Camel","India")

1. **Typed Pattern**

scala> def generalSize(x: Any) = x match {

| case s: String => s.length

| case m: Map[\_, \_] => m.size

| case l: List[\_] => l.length

| case \_ => -1

| }

scala> generalSize(List(1,2,3,4))

scala> generalSize(math.Pi)

scala> generalSize(65)

scala> generalSize(Map(1 -> 'a', 2 -> 'b'))

scala> generalSize("Scala Pattern Matching")

1. **Alternate cases – Multiple cases mapping to a common result**

scala> def patternAlternatives(param:Any) = param match{

| case Seq(1,2,3,4) | Seq(1,2) => println("Sequence type")

| case "Scala" | "Python" | "Haskel" => println("Programming language")

| case \_ => println("Niether Sequence nor Programming language")

| }

scala> patternAlternatives(Seq(1,2,3,4))

scala> patternAlternatives("Haskel")

scala> patternAlternatives("C")

1. **Pattern Matching on Lists – Length of a list**

scala> val listLength: (List[Any]=>Int) = (l) => l match{

| case Nil => 0

| case head::tail => 1+listLength(tail)

| }

scala> var l1 = List(1,2,3)

scala> listLength(l1)

scala> val l3 = List("Hello","Scala","Lists")

scala> listLength(l3)

1. **Pattern Matching on Lists – Sum of elements**

scala> val listSum: (List[Int]=>Int) = (l) => l match{

| case Nil => 0

| case head::tail => head + listSum(tail)

| }

scala> listSum(l1)

1. **Pattern Matching on Lists – Product of elements**

scala> val listProduct: (List[Int]=>Int) = (l) => l match{

| case Nil => 1

| case 0::\_ => 0

| case head::tail => head \* listProduct(tail)

| }

scala> listProduct(l1)

scala> l1 = List(1,3,2,0)

scala> listProduct(l1)

1. **Another Example**

scala> def matchMonth(i:Int) = i match{

| case 1|3|5|7|8|10|12 => println("31 days")

| case 4|6|9|11 => println("30 days")

| case 2 => println("28 days")

| case \_ => println("Invalid month number")

| }

scala> var month =5

scala> matchMonth(month)

scala> var month =13

scala> matchMonth(month)

scala> var month =2

scala> matchMonth(month)

scala> var month =11

scala> matchMonth(month)

1. **Use of if as a guard**

scala> def matchRange(i:Int) = i match{

| case a if 0 to 9 contains a => println(a + " is in the range 0-9")

| case b if 10 to 19 contains b => println(b + " is in the range 10-19"

| case c if 20 to 29 contains c => println(c + " is in the range 20-29")

| case \_ => println("The entered integer is either less than 0 or greater than 29")

| }

scala> matchRange(-1)

scala> matchRange(23)

scala> matchRange(5)

scala> matchRange(37)

scala> matchRange(17)

1. **Another example for if**

scala> def digitCheck(num:Int) = num match{

| case p if (p==2 || p==3 ||p==5||p==7) => println(p + " is a prime digit")

| case c if (c==4||c==6||c==8||c==9) => println(c + " is a composit digit")

| case n if (n==0 || n==1) => println( n + " is neither prime nor composite")

| case \_ => println("The value is either a negative intger or a positive intger with more than 1 digit")

| }

scala> digitCheck(0)

scala> digitCheck(1)

scala> digitCheck(2)

scala> digitCheck(6)

scala> digitCheck(-6)

scala> digitCheck(16)