The Godfather of Talent

Futurense

Lab Manual

Program Name: CSE-AIDE

Course: Scala Programming

Semester:3

SME: Dr. Arjun Magotra

Program Name: Functional Programming		
1	Background & Use-case	
2	Problem Statement 1	Write a Scala program to print "Hello, world" and version of the Scala language.

3	Research Methodology	`object HelloWorld {`
		`  def main(args: Array[String]): Unit = {`
		`    println("Hello, world!")`
		`    println("Scala language: "+util.Properties.versionString)`
		`  }`
		`}`
		
		Scala Code-2:
		
		`object Hello extends App {`
		
		`    println("Hello, world")`
		
		`    println("Scala language: "+util.Properties.versionString)`
		
		`}`

4	**Skills Required**	

5	**Expected output**	Sample Output:
		`Hello, world!`
		`Scala language: version 2.13.3`
		Sample Output:
		`Hello, world`
		`Scala language: version 2.13.3`

Program Name:		
1	**Background & Use-case**	
2	**Problem Statement 2**	Write a Scala program to compute the sum of the two given integer values. If the two values are the same, then return triples their sum.
3	**Research Methodology**	**Sample Solution**:

<table>
<tr>
<td></td>
<td></td>
<td>

Scala Code:

```
object scala_basic {
  def test(x:Int, y:Int) : Int =
    {
        if (x == y) (x + y) * 3 else x + y
    }

   def main(args: Array[String]): Unit = {
      println("Result: " + test(1, 2));
      println("Result: " + test(2, 2));
   }
}
```

</td>
</tr>
<tr>
<td>4</td>
<td><b>Skills Required</b></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td><b>Expected output</b></td>
<td>

Sample Output:

```
Result: 3
Result: 12
```

</td>
</tr>
</table>

Program Name:		
1	**Background & Use-case**	
2	**Problem Statement 3**	Write a Scala program to get the absolute difference between n and 51. If n is greater than 51 return triple the absolute difference.

3	**Research Methodology**	**Sample Solution**:
		Scala Code:
		`object scala_basic {`
		`  def test(x:Int) : Int =`
		`    {`
		`    val abs_Diff = Math.abs(x - 51)`

```
    if (x > 51) 3 * abs_Diff else abs_Diff
    }


   def main(args: Array[String]): Unit = {
      println("Result: " + test(60));
      println("Result: " + test(40));
   }
}
```

4	**Skills Required**	
5	**Expected output**	Sample Output: Result: 27 Result: 11

Program Name:		
1	**Background & Use-case**	
2	**Problem Statement 4**	Write a Scala program to check two given integers, and return true if one of them is 30 or if their sum is 30.
3	**Research Methodology**	**Sample Solution**: **Scala Code:**

```
object scala_basic {
  def test(x:Int, y:Int) : Boolean  =
    {
     x == 30 || y == 30 || x + y == 30
    }


   def main(args: Array[String]): Unit = {
      println("Result: " + test(30, 0));
      println("Result: " + test(25, 5));
      println("Result: " + test(30, 20));
```

		`    println("Result: " + test(25, 20));`
		`  }`
		`}`

4	**Skills Required**	

5	**Expected output**	Sample Output:
		`Result: true`
		`Result: true`
		`Result: true`
		`Result: false`

Program Name:		
1	**Background & Use-case**	
2	**Problem Statement 5**	Write a Scala program to check a given integer and return true if it is within 20 of 100 or 300.
3	**Research Methodology**	**Sample Solution**: **Scala Code:** `object scala_basic {`

<table>
<tr><td></td><td></td><td><pre>
  def test(x:Int) : Boolean =
    {
     Math.abs(100 - x) <= 20 || Math.abs(300 - x) <= 20
    }

   def main(args: Array[String]): Unit = {
      println("Result: " + test(115));
      println("Result: " + test(200));
      println("Result: " + test(250));
      println("Result: " + test(70));
   }
}
</pre></td></tr>
<tr><td>4</td><td><b>Skills Required</b></td><td></td></tr>
<tr><td>5</td><td><b>Expected output</b></td><td>Sample Output:<pre>
Result: true
Result: false
Result: false
Result: false
</pre></td></tr>
</table>

		Program Name:
1	**Background & Use-case**	
2	**Problem Statement 6**	Write a Scala program to create a new string where 'if' is added to the front of a given string. If the string already begins with 'if', return the string unchanged.
3	**Research Methodology**	**Scala Code:**

```
object scala_basic {
  def test(str: String): String =
    {
     if (str.startsWith("if")) str else "if " + str
    }

   def main(args: Array[String]): Unit = {
      println("Result: " + test("if else"));
      println("Result: " + test("else"));
    }
```

		`}`
4	**Skills Required**	
5	**Expected output**	Sample Output: `Result: if else` `Result: if else`

Program Name:		
1	**Background & Use-case**	
2	**Problem Statement 7**	Write a Scala program to remove the character in a given position of a given string. The given position will be in the range 0...string length -1 inclusive.

3	**Research Methodology**	**Sample Solution**:
		Scala Code:
		`object scala_basic {`
		` def test(str: String, n: Int): String =`
		`  {`
		`   str.take(n) + str.drop(n + 1)`
		`  }`
		`  def main(args: Array[String]): Unit = {`
		`   println("Result: " + test("Scala", 1));`

		`        println("Result: " + test("Scala", 0));`
		`        println("Result: " + test("Scala", 4));`
		`    }`
		`}`

4	**Skills Required**	

5	**Expected output**	Sample Output:
		`Result: Sala`
		`Result: cala`
		`Result: Scal`

Program Name:		
1	**Background & Use-case**	
2	**Problem Statement 8**	Write a Scala program to exchange the first and last characters in a given string and return the new string.
3	**Research Methodology**	**Sample Solution**: **Scala Code:**

```
object scala_basic {
  def test(str1: String): String =
    {
    val len = str1.length
    if (len <= 1) str1
    else str1.charAt(len - 1) + str1.substring(1, len - 1) + str1.charAt(0)
    }
   def main(args: Array[String]): Unit = {
      println("Result: " + test("Scala"));
      println("Result: " + test("abcd"));
```

		`        println("Result: " + test("ab"));`
		`        println("Result: " + test("a"));`
		`    }`
		`}`

4	**Skills Required**	

5	**Expected output**	Sample Output:
		`Result: acalS`
		`Result: dbca`
		`Result: ba`
		`Result: a`

Program Name:		
1	**Background & Use-case**	
2	**Problem Statement 9**	Write a Scala program to create a new string which is n (non-negative integer) copies of a given string.
3	**Research Methodology**	**Sample Solution**: **Scala Code:**

<table>
<tr><td></td><td></td><td><pre>
object scala_basic {

   def test(str1: String, n: Int): String = {

    str1 * n;

  }

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

      println("Result: " + test("Scala", 2));

      println("Result: " + test("Python",1));

      println("Result: " + test("JS",6));

    }

  }
</pre></td></tr>
<tr><td>4</td><td><b>Skills Required</b></td><td></td></tr>
<tr><td>5</td><td><b>Expected output</b></td><td>Sample Output:<pre>
Result: ScalaScala
Result: Python
Result: JSJSJSJSJSJS
</pre></td></tr>
</table>

Program Name:		
1	**Background & Use-case**	
2	**Problem Statement 10**	Write a Scala function to check if a given list is sorted in ascending order.
3	**Research Methodology**	**Sample Solution**: **Scala Code:**

```scala
object ListSortChecker {
  def isSortedAscending[A](list: List[A])(implicit ord: Ordering[A]): Boolean = {
    list.sliding(2).forall { case List(a, b) => ord.lteq(a, b) }
  }


  def main(args: Array[String]): Unit = {
    val list1 = List(1, 2, 3, 4, 5, 6)
    val list2 = List(4, 2, 5, 1, 6, 3)
    println("List1: "+list1)
```

		`println(s"Is List1 is sorted in ascending order? ${isSortedAscending(list1)}")`
		`println("List2: "+list2)`
		`println(s"Is List2 is sorted in ascending order? ${isSortedAscending(list2)}")`
		`}`
		`}`

4	**Skills Required**	

5	**Expected output**	`List1: List(1, 2, 3, 4, 5, 6)`
		`Is List1 is sorted in ascending order? true`
		`List2: List(4, 2, 5, 1, 6, 3)`
		`Is List2 is sorted in ascending order? false`

Program Name: Object Oriented Programming		
1	**Background & Use-case**	**Explanation:** In the exercise, The "Person" class has properties like name, age, and country defined using constructor parameters. The class also includes getter and setter methods for each property. The getName, setName, getAge, setAge, getCountry, and setCountry methods allow you to retrieve and modify the properties' values. The "PersonApp" object contains the main method, which demonstrates the "Person" class. It creates a "Person" object, prints its original values, updates the properties using the setter methods, and then prints the updated values.
2	**Problem Statement 11**	Write a Scala program that creates a class called Person with properties like name, age and country. Implement methods to get and set properties.

3	**Research Methodology**	**Sample Solution**:
		Scala Code:
		`class Person(var name: String, var age: Int, var country: String) {`
		`  def getName: String = name`

```
  def setName(newName: String): Unit = {
    name = newName
  }

  def getAge: Int = age

  def setAge(newAge: Int): Unit = {
    age = newAge
  }

  def getCountry: String = country

  def setCountry(newCountry: String): Unit = {
    country = newCountry
  }
}

object PersonApp {
  def main(args: Array[String]): Unit = {
    val person = new Person("Andrey Ira", 35, "France")

    println("Original Person:")
    println(s"Name: ${person.getName}")
    println(s"Age: ${person.getAge}")
    println(s"Country: ${person.getCountry}")

    person.setName("Lior Daniela")
    person.setAge(30)
    person.setCountry("Canada")

    println("\nUpdated Person:")
```

		`    println(s"Name: ${person.getName}")`
		`    println(s"Age: ${person.getAge}")`
		`    println(s"Country: ${person.getCountry}")`
		`  }`
		`}`

4	**Skills Required**	

5	**Expected output**	Sample Output:
		`Original Person:`
		`Name: Andrey Ira`
		`Age: 35`
		`Country: France`
		
		`Updated Person:`
		`Name: Lior Daniela`
		`Age: 30`
		`Country: Canada`

Program Name: Scala OOP

1	Background & Use-case	The "Person" class defines the properties name, age, and country, along with the corresponding getter and setter methods. The "Student" class extends the Person class and adds an additional property called grade. It also includes getter and setter methods specific to the grade property. The StudentApp object contains the main method, which demonstrates the Student class usage. It creates a Student object, prints its original values, updates the properties using the setter methods inherited from the Person class and the setGrade method from the Student class. It then prints the updated values.
2	Problem Statement 12	Write a Scala program that creates a subclass Student that extends the Person class. Add a property called grade and implement methods to get and set it.

<table>
<tr>
<td>3</td>
<td><b>Research Methodology</b></td>
<td>

```
class Person(var name: String, var age: Int, var country: String) {
  def getName: String = name

  def setName(newName: String): Unit = {
    name = newName
  }

  def getAge: Int = age

  def setAge(newAge: Int): Unit = {
    age = newAge
  }

  def getCountry: String = country

  def setCountry(newCountry: String): Unit = {
    country = newCountry
  }
}

class Student(name: String, age: Int, country: String, var grade: String)
  extends Person(name, age, country) {

  def getGrade: String = grade

  def setGrade(newGrade: String): Unit = {
    grade = newGrade
  }
}

object StudentApp {
  def main(args: Array[String]): Unit = {
    val student = new Student("Saturnino Nihad", 18, "USA", "A")

    println("Original Student:")
    println(s"Name: ${student.getName}")
    println(s"Age: ${student.getAge}")
    println(s"Country: ${student.getCountry}")
    println(s"Grade: ${student.getGrade}")
```

</td>
</tr>
</table>

<table>
<tr>
<td></td>
<td></td>
<td>

```
    student.setName("Albino Ellen")
    student.setAge(20)
    student.setCountry("Canada")
    student.setGrade("B")

    println("\nUpdated Student:")
    println(s"Name: ${student.getName}")
    println(s"Age: ${student.getAge}")
    println(s"Country: ${student.getCountry}")
    println(s"Grade: ${student.getGrade}")
  }
}
```

</td>
</tr>
<tr>
<td>4</td>
<td><b>Skills Required</b></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td><b>Expected output</b></td>
<td>

Sample Output:

```
Original Student:
Name: Saturnino Nihad
Age: 18
Country: USA
Grade: A

Updated Student:
Name: Albino Ellen
Age: 20
Country: Canada
Grade: B
```

</td>
</tr>
</table>

Program Name: **Scala OOP**

1	**Background & Use-case**	
2	**Problem Statement 13**	The "MathUtils" object contains the factorial method. This method calculates the factorial of a given number using recursion. If the number is 0 or 1, it returns 1. Otherwise, it recursively calls itself with n - 1 and multiplies the result by n. The "Main" object contains the main method where you can test the factorial method. In this example, it calculates the factorial of the number 4 and 10 and prints the result.
3	**Research Methodology**	

```scala
object MathUtils {
  def factorial(n: Int): BigInt = {
    if (n == 0 || n == 1) {
      1
    } else {
      n * factorial(n - 1)
    }
  }
}

object Main {
  def main(args: Array[String]): Unit = {
    val number1 = 4
    val result1 = MathUtils.factorial(number1)
```

<table>
<tr><td></td><td></td><td>

```
    println(s"The factorial of $number1 is: $result1")
    val number2 = 10
    val result2 = MathUtils.factorial(number2)
    println(s"The factorial of $number2 is: $result2")
  }
}
```

</td></tr>
<tr><td>4</td><td><b>Skills Required</b></td><td></td></tr>
<tr><td>5</td><td><b>Expected output</b></td><td>

Sample Output:

```
The factorial of 4 is: 24
The factorial of 10 is: 3628800
```

</td></tr>
</table>

	Program Name: Scala OOP Exercise	
1	**Background & Use-case**	• he "Shape" class is an abstract class that defines the area method without implementation. • The "Rectangle" class extends the "Shape" class and provides an implementation of the area method based on the rectangle's width and height. • The "Circle" class also extends the "Shape" class and implements the area method based on the circle radius. • The "ShapeApp" object contains the main method where you can test functionality. It creates instances of Rectangle and Circle, and calls their area methods to calculate and print the respective areas.
2	**Problem Statement 14**	Write a Scala program that creates an abstract class Shape with an abstract method area. Implement subclasses Rectangle and Circle that override the area method.

3	**Research Methodology**	`abstract class Shape {`
		`  def area: Double`
		`}`
		
		`class Rectangle(width: Double, height: Double) extends Shape {`
		`  override def area: Double = width * height`
		`}`
		
		`class Circle(radius: Double) extends Shape {`
		`  override def area: Double = math.Pi * radius * radius`
		`}`
		
		`object ShapeApp {`
		`  def main(args: Array[String]): Unit = {`

		`    val rectangle = new Rectangle(7, 5)`
		`    println(s"Rectangle Area: ${rectangle.area}")`
		
		`    val circle = new Circle(4.5)`
		`    println(s"Circle Area: ${circle.area}")`
		`  }`
		`}`

4	**Skills Required**	

5	**Expected output**	Sample Output:
		`Rectangle Area: 35.0`
		`Circle Area: 63.61725123519331`

<table>
<tr><th colspan="3">Program Name:</th></tr>
<tr><td>1</td><td><b>Background & Use-case</b></td><td>
<ul>
<li>The "BankAccount" class is defined with a constructor that takes accountNumber as a parameter and initializes the balance property. The accountNumber property is defined as a val to make it read-only, while the balance property is defined as a var to make it mutable.</li>
<li>The "deposit()" method takes an amount parameter, adds it to the balance, and prints the updated balance.</li>
<li>The "withdraw()" method takes an amount parameter and checks if the withdrawal amount is less than or equal to the balance. If it is, it subtracts the amount from the balance and prints the updated balance. Otherwise, it prints "Insufficient balance."</li>
<li>The "BankAccountApp" object contains the "main()" method where you can test functionality. A BankAccount instance is created, the initial account number and balance are printed, and then deposits and withdrawals are performed.</li>
</ul>
</td></tr>
<tr><td>2</td><td><b>Problem Statement 15</b></td><td>Write a Scala program that creates a class BankAccount with properties accountNumber and balance. Implement methods to deposit and withdraw money from the account.</td></tr>
<tr><td>3</td><td><b>Research Methodology</b></td><td><pre>
class BankAccount(val accountNumber: String, var balance: Double) {
  def deposit(amount: Double): Unit = {
    balance += amount
    println(s"Deposited $amount. New balance: $balance")
  }

  def withdraw(amount: Double): Unit = {
</pre></td></tr>
</table>

```
    if (amount <= balance) {
      balance -= amount
      println(s"Withdrew $amount. New balance: $balance")
    }
    else
    {
      println(s"Want to withdraw $amount? Insufficient balance!")
    }
  }
}

object BankAccountApp {
  def main(args: Array[String]): Unit = {
    val account = new BankAccount("SB-1234", 1000.0)

    println(s"Account Number: ${account.accountNumber}")
    println(s"Initial Balance: ${account.balance}")

    account.deposit(500.0)
    account.withdraw(200.0)
    account.withdraw(2000.0)
  }
}
```

4	**Skills Required**	
5	**Expected output**	Sample Output: Account Number: SB-1234 Initial Balance: 1000.0 Deposited 500.0. New balance: 1500.0 Withdrew 200.0. New balance: 1300.0

		`Want to withdraw 2000.0? Insufficient balance!`

Program Name:		
1	**Background & Use-case**	- The "ContactInfo" class is defined with the properties name, email, and address. It has a primary constructor that takes these properties as parameters. - The "Customer" class is defined by the property contactInfo of type ContactInfo. It represents a customer and includes contact information. - The CustomerApp object contains the main method to test functionality. It creates an instance of the ContactInfo class with sample information and then creates a Customer object using the ContactInfo instance. - Finally, the program prints the customer's name, email, and address using string interpolation.
2	**Problem Statement 16**	Write a Scala program that creates a class ContactInfo with properties name, email, and address. Create a class Customer that includes a ContactInfo object.
3	**Research Methodology**	`class ContactInfo(val name: String, val email: String, val address: String)`

		`class Customer(val contactInfo: ContactInfo)`
		
		`object CustomerApp {`
		`  def main(args: Array[String]): Unit = {`
		`    val contact = new ContactInfo("Serafim Eline", "serafim@example.com", "11 Open Street")`
		`    val customer = new Customer(contact)`
		
		`    println(s"Customer Name: ${customer.contactInfo.name}")`
		`    println(s"Customer Email: ${customer.contactInfo.email}")`
		`    println(s"Customer Address: ${customer.contactInfo.address}")`
		`  }`
		`      }`

4	**Skills Required**	

5	**Expected output**	Sample Output:
		
		`Customer Name: Serafim Eline`
		`Customer Email: serafim@example.com`
		`Customer Address: 11 Open Street`

Program Name:		
1	**Background & Use-case**	• First, we define a sealed trait "Color" that serves as the base type for different color objects. Then, we define case objects "Red", Green, and Blue that extend the Color trait to represent specific colors. • The "ColorApp" object contains the “main()” method where we can test functionality. It assigns the "Red" color to the myColor variable and calls the "printColor()" method to print the color name. • The "printColor()" method uses pattern matching to determine the specific color of the object and prints a corresponding message.
2	**Problem Statement 17**	Write a Scala program that creates an enum class Color with values for different colors. Use the enum class to represent an object's color.
3	**Research Methodology**	**Sample Solution**: **Scala Code:** `sealed trait Color`

```
case object Red extends Color

case object Green extends Color

case object Blue extends Color

case object Orange extends Color


object ColorApp {

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

    val myColor: Color = Red

    //val myColor: Color = Blue

    printColor(myColor)

  }


  def printColor(color: Color): Unit = color match {

    case Red   => println("The color is Red.")

    case Green => println("The color is Green.")

    case Blue  => println("The color is Blue.")

    case Orange  => println("The color is Orange.")
```

		`    case _      => println("Unknown color.")`
		`  }`
		`}`

4	**Skills Required**	
5	**Expected output**	Sample Output: `The color is Red.`

Program Name:		
1	**Background & Use-case**	• First, we create two sets set1 and set2 with some elements. • To find the common elements between the two sets, we use the intersect method. This returns a new set containing the elements common to both sets. • In the program, we find the common elements between set1 and set2 by using the line val commonElements = set1.intersect(set2). • Finally, we print the sets and the common elements.
2	**Problem Statement 18**	Write a Scala program to create a set and find the common elements between two sets.

3	**Research Methodology**	`object CommonElementsSetExample {`
		`  def main(args: Array[String]): Unit = {`
		`    // Create two sets`
		`    val set1 = Set(1, 2, 3, 4, 5)`
		`    val set2 = Set(4, 5, 6, 7, 8)`
		
		`    // Print the sets`
		`    println("Set1: " + set1)`
		`    println("Set2: " + set2)`
		
		`    // Find the common elements between the two sets`
		`    val commonElements = set1.intersect(set2)`
		
		`    // Print the common elements`
		`    println("Common elements: " + commonElements)`
		`  }`
		`}`

4	**Skills Required**	

5	**Expected output**	Sample Output:
		`Set1: HashSet(5, 1, 2, 3, 4)`
		`Set2: HashSet(5, 6, 7, 8, 4)`
		`Common elements: HashSet(5, 4)`

Program Name:		
1	**Background & Use-case**	First, we create two maps map1 and map2 using the Map constructor and provide key-value pairs. • Here are the steps we need to follow to determine the difference between the two maps: • Retrieve the set of keys from map2 using the keySet method • Use the -- operator on map1 and specify the set of keys from map2 to remove those keys and their associated values from map1. Finally, we print the result using println, including the difference between the two maps.
2	**Problem Statement 19**	Write a Scala program to create a map and find the difference between two maps.

3	**Research Methodology**	`object FindDifferenceBetweenMapsExample {`
		`  def main(args: Array[String]): Unit = {`
		`    // Create two maps`

<table>
<tr>
<td></td>
<td></td>
<td>

```
    val map1 = Map("Red" -> 1, "Green" -> 4, "Blue" -> 2, "Orange" -> 3)
    val map2 = Map("Red" -> 5, "Green" -> 4, "Blue" -> 2, "Pink" -> 3)

    // Print the original map
    println("Original map1: " + map1)
    println("Original map2: " + map2)

    // Find the difference between the maps
    val difference = map1 -- map2.keySet

    // Print the result
    println(s"The difference between the maps is: $difference")
  }
}
```

</td>
</tr>
<tr>
<td>4</td>
<td><strong>Skills Required</strong></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td><strong>Expected output</strong></td>
<td>

Sample Output:

```
Original map1: Map(Red -> 1, Green -> 4, Blue -> 2, Orange -> 3)
Original map2: Map(Red -> 5, Green -> 4, Blue -> 2, Pink -> 3)
The difference between the maps is: Map(Orange -> 3)
```

</td>
</tr>
</table>