Principles of Programming Language
Unit-3
Scala Programming

1. **Word Frequency Counter (using immutable Maps and functional operations)**

```
object WordFrequencyCounter extends App {
  val text = "Scala is great and Scala is powerful and
functional"

  val frequencies = text
    .toLowerCase
    .split("\\W+")
    .groupBy(identity)
    .mapValues(_.length)

  frequencies.foreach { case (word, count) =>
    println(s"$word -> $count")
  }
}
```

Explanation: A singleton object WordFrequencyCounter is created. extends App makes this object executable without needing a main method—whatever is in the body of the object runs automatically. A string text is defined in val. This is the input sentence we'll analyze for word frequency. **toLowerCase:** Converts the entire string to lowercase. **split("\\W+"):** Splits the text into words using a **regular expression**: \\W+ matches any sequence of non-word characters (punctuation, spaces, etc.). **groupBy(identity);** Groups the words by themselves. identity just returns the word. You get a **Map[String, Array[String]]**, where each key is a word and the value is a list of all its occurrences. **mapValues(_.length):** For each group of words (the values), it computes the **length**, i.e., how many times that word appeared.
s"..." – String Interpolation (in println_). The s before the string allows you to embed variables directly into the string using $.

- **So inside the string:**
  - **$word is replaced with the actual value of the word variable.**
  - **$count is replaced with the value of the count variable.**

2. **Fibonacci with Memoization using Lazy Val**

```
object FibonacciMemoization extends App {
  def fib(n: Int): BigInt = {
    lazy val memo: Stream[BigInt] = BigInt(0) #:: BigInt(1) #::
memo.zip(memo.tail).map(t => t._1 + t._2)
    memo(n)
  }

  println((0 to 20).map(fib).mkString(", "))
}
```

Explanation: returns the **nth Fibonacci number** as a BigInt (for large number support).
What's a Stream?

- A Stream is a **lazy list**. Elements are **computed only when accessed**.
- It helps **memoize** results automatically, meaning each Fibonacci number is calculated once and reused later.
- • `BigInt(0)` is the first Fibonacci number → `0`
- • `BigInt(1)` is the second Fibonacci number → `1`
- • `memo` is the full stream: `0, 1, 1, 2, 3, 5, ...`
- • `memo.tail` is the same stream but without the first element: `1, 1, 2, 3, ...`
- • `memo.zip(memo.tail)` pairs them:
  .map(t => t._1 + t._2) adds each pair to generate the next Fibonacci number.

memo(n)
Grabs the nth Fibonacci number from the stream

 Computes fib(0) through fib(20)
 Joins them into a string with commas
 Prints the result:

3. Matrix multiplication

```
object MatrixMultiplication extends App {
  def multiply(a: Array[Array[Int]], b: Array[Array[Int]]):
Array[Array[Int]] = {
    val rows = a.length
    val cols = b(0).length
    val common = b.length

    Array.tabulate(rows, cols) { (i, j) =>
      (0 until common).map(k => a(i)(k) * b(k)(j)).sum
    }
  }

  val matA = Array(Array(1, 2), Array(3, 4))
  val matB = Array(Array(5, 6), Array(7, 8))

  val result = multiply(matA, matB)
  result.foreach(row => println(row.mkString(" ")))
}
```

Explanation: rows: number of rows in matrix a, cols: number of columns in matrix b, common: number of columns in a (same as number of rows in b)
This is the shared dimension used during multiplication.

- **Array.tabulate(rows, cols):** Creates a 2D array of size rows x cols.

- For each element at position (i, j) it calculates the dot product of the i-th row of a and the j-th column of b.
- Inner .map and .sum: Loops through k from 0 to common-1
- Multiplies a(i)(k) with b(k)(j) for each k
- Sums up the products to get the element at position (i, j) in the result matrix
- Prints each row of the result matrix with values separated by spaces.

4. Simple Actor-like Concurrency using Future

```
import scala.concurrent._
import scala.concurrent.duration._
import ExecutionContext.Implicits.global

object FutureExample extends App {
  def fetchData(id: Int): Future[String] = Future {
    Thread.sleep(1000)
    s"Data for ID: $id"
  }

  val results = (1 to 5).map(fetchData)

  val aggregated = Future.sequence(results)
  aggregated.foreach(data => println("All data fetched:\n"
+ data.mkString("\n")))

  Await.result(aggregated, 5.seconds)
}
```

Explanation: scala.concurrent._: Enables use of Future, Await, etc.

- scala.concurrent.duration._: Enables time units like 5.seconds.
- ExecutionContext.Implicits.global: Provides a thread pool for executing Futures. This is necessary because a Future needs an execution context to run in.
- The App trait allows the object to run as a main program without needing a main method.
- fetchData is an asynchronous function that: Sleeps for 1 second (Thread.sleep(1000)) to simulate a time-consuming task (like calling an API). Returns a Future[String] containing "Data for ID: X" where X is the passed id.
- Calls fetchData for IDs 1 to 5. This creates 5 Futures, each working concurrently (in parallel). The result is a Seq[Future[String]] (it's **a collection of Futures**, and each Future will eventually produce a String.).

- Future.sequence transforms Seq[Future[String]] into Future[Seq[String]] (it's **one single future** that will give you **a full sequence** of strings once it's complete). It waits for all the futures to complete and then aggregates their results into a single future.

- `foreach` is called **when the future is completed**. It prints each fetched string (`data.mkString("\n")`).

- `Await.result` blocks the main thread until the `aggregated` future completes or timeout (after 5 seconds). This is necessary because the main thread might exit before the asynchronous work is done.

5. Escape sequence

```
object EscapeSequencesExample {
  def main(args: Array[String]): Unit = {
    println("Scala Escape Sequences Examples:")

    // New line
    println("Line1\nLine2")

    // Tab
    println("Name\tAge\tCountry")

    // Backslash
    println("This is a backslash: \\")

    // Double quote
    println("He said, \"Scala is awesome!\"")

    // Single quote
    println("It\'s a beautiful day!")

    // Carriage return
    println("12345\rABCDE")

    // Backspace
    println("Oops!\b\b\b    ")

    // Form feed (might not be visible in modern consoles)
    println("Page 1\fPage 2")

    // Unicode character
    println("Unicode heart: \u2665")
  }
}
```

Explanation: \\ prints a literal backslash. \" allows you to use double quotes inside a string. \' is used to include a single quote (apostrophe) inside a string. Carriage Return (\r): moves the cursor back to the beginning of the line. So "ABCDE" overwrites "12345" character by character. \b moves the cursor one character **backward**, removing the previous character. This will erase the last 3 characters of "Oops!" before printing spaces. \f was used in old printers to advance to the next page. In modern consoles, it's often **not visible**, or it may just insert a strange character. \u2665 represents a **Unicode heart symbol**.

6. **If-Else Example**

```
object IfElseExample {
  def main(args: Array[String]): Unit = {
    val number = 42

    if (number % 2 == 0)
      println(s"$number is even")
    else
      println(s"$number is odd")
  }
}
```

7. **For Loop Example**

```
object ForLoopExample {
  def main(args: Array[String]): Unit = {
    println("Printing numbers from 1 to 5:")
    for (i <- 1 to 5) {
      println(i)
    }
  }
}
```

8. **Pattern Matching**

```
object PatternMatchExample {
  def main(args: Array[String]): Unit = {
    val day = "Sunday"

    val activity = day match {
      case "Monday" | "Tuesday" | "Wednesday" | "Thursday" |
"Friday" => "Work"
      case "Saturday" | "Sunday" => "Relax"
      case _ => "Unknown day"
    }

    println(s"Today is $day: $activity")
  }
}
```

9. While Loop Example

```
object WhileLoopExample {
  def main(args: Array[String]): Unit = {
    var count = 5

    while (count > 0) {
      println(s"Countdown: $count")
      count -= 1
    }

    println("Liftoff!")
  }
}
```

10. Array Example

```
object ArrayExample {
  def main(args: Array[String]): Unit = {
    val fruits = Array("Apple", "Banana", "Cherry")

    for (fruit <- fruits) {
      println(fruit)
    }
  }
}
```

11. Map Example

```
object MapExample {
  def main(args: Array[String]): Unit = {
    val capitals = Map("France" -> "Paris", "Japan" -> "Tokyo",
"India" -> "New Delhi")
    println("Capital of France is " + capitals("France"))
    capitals.foreach { case (country, capital) =>
      println(s"The capital of $country is $capital")
    }
  }
}
```

12. List Example

```
object ListExample {
  def main(args: Array[String]): Unit = {
    val numbers = List(1, 2, 3, 4, 5)
    val squared = numbers.map(n => n * n)
    println("Original List: " + numbers)
    println("Squared List: " + squared)
  }
}
```

13. Try Example (Safe Exception Handling)

```
import scala.util.{Try, Success, Failure}

object TryExample {
  def divide(a: Int, b: Int): Try[Int] = Try(a / b)

  def main(args: Array[String]): Unit = {
    val result = divide(10, 0)
    result match {
      case Success(value) => println(s"Result: $value")
      case Failure(exception) => println(s"Error occurred:
${exception.getMessage}")
    }
  }
}
```

14. Simple Class and Object

```
class Person(val name: String, val age: Int) {
  def greet(): Unit = {
    println(s"Hi, my name is $name and I am $age years old.")
  }
}

object ClassExample {
  def main(args: Array[String]): Unit = {
    val p = new Person("Alice", 25)
    p.greet()
  }
}
```

15. Function Example

```
object FunctionExample {
  def greet(name: String): String = {
    s"Hello, $name!"
  }

  def main(args: Array[String]): Unit = {
    val message = greet("Scala Learner")
    println(message)
  }
}
```

16. OOPS with inheritance

Animal.scala

```
class Animal(val name: String) {
  def speak(): Unit = {
```

```
    println(s"$name makes a sound.")
  }
}
```

Dog.scala

```
class Dog(name: String) extends Animal(name) {
  override def speak(): Unit = {
    println(s"$name says: Woof!")
  }
}
```

OOPInheritanceExample.scala

```
object OOPInheritanceExample {
  def main(args: Array[String]): Unit = {
    val dog = new Dog("Buddy")
    dog.speak()
  }
}
```