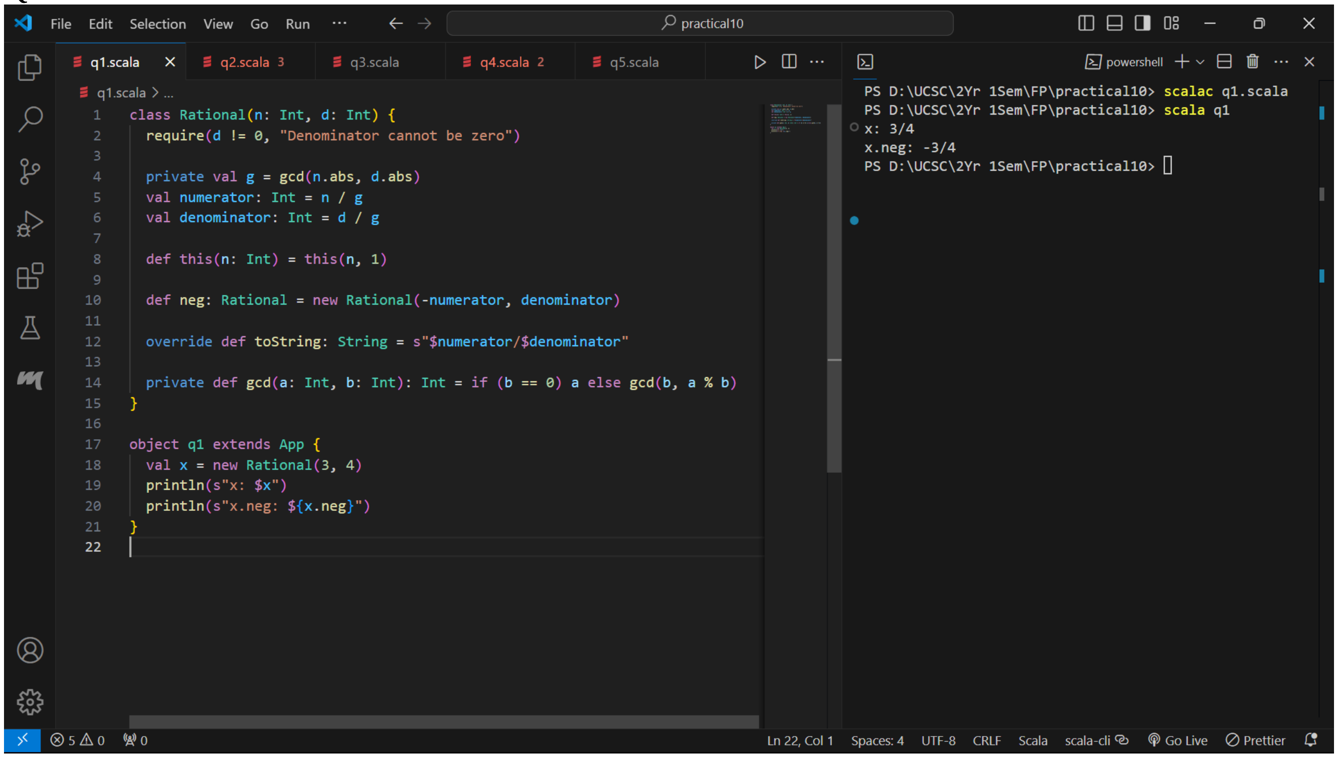
Practical 10 - 22001956

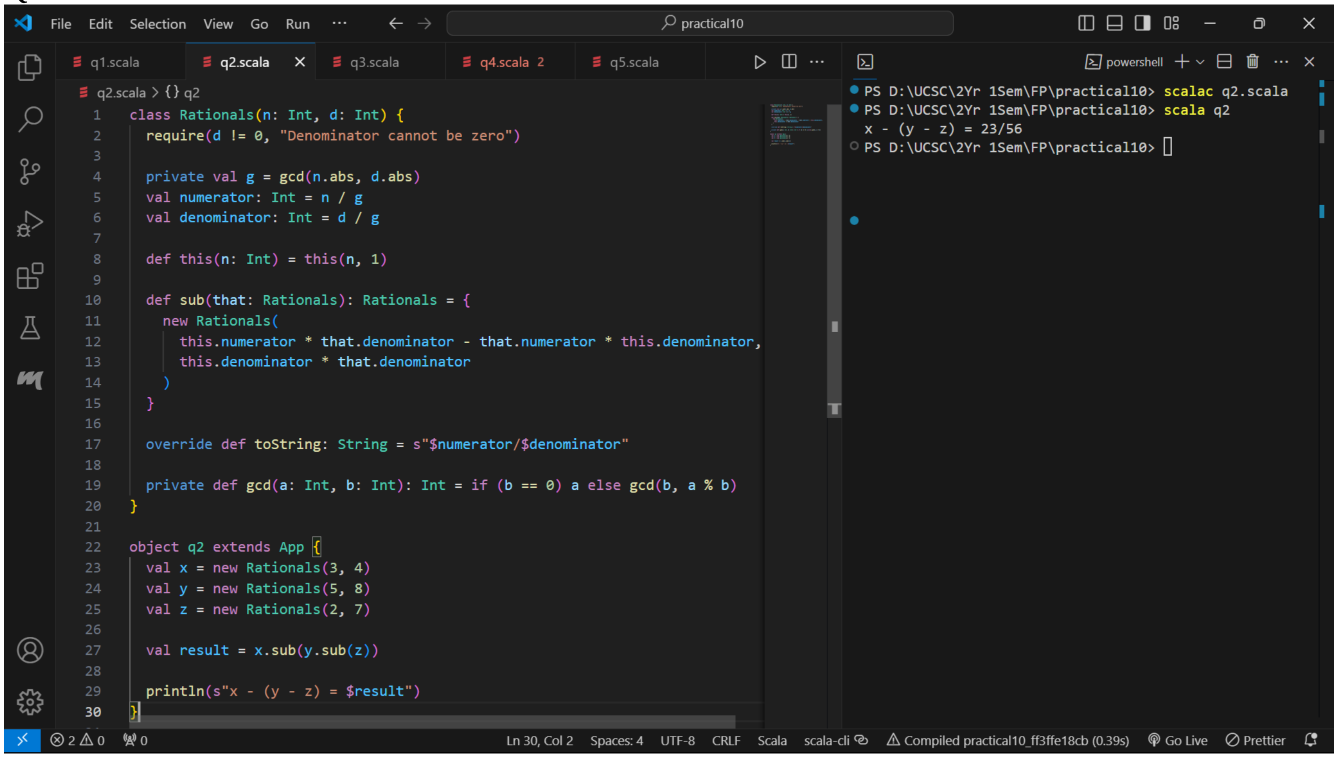
Q1



```scala
class Rational(n: Int, d: Int) {
  require(d != 0, "Denominator cannot be zero")

  private val g = gcd(n.abs, d.abs)
  val numerator: Int = n / g
  val denominator: Int = d / g

  def this(n: Int) = this(n, 1)

  def neg: Rational = new Rational(-numerator, denominator)

  override def toString: String = s"$numerator/$denominator"

  private def gcd(a: Int, b: Int): Int = if (b == 0) a else gcd(b, a % b)
}

object q1 extends App {
  val x = new Rational(3, 4)
  println(s"x: $x")
  println(s"x.neg: ${x.neg}")
}
```

```
PS D:\UCSC\2Yr 1Sem\FP\practical10> scalac q1.scala
PS D:\UCSC\2Yr 1Sem\FP\practical10> scala q1
x: 3/4
x.neg: -3/4
PS D:\UCSC\2Yr 1Sem\FP\practical10>
```

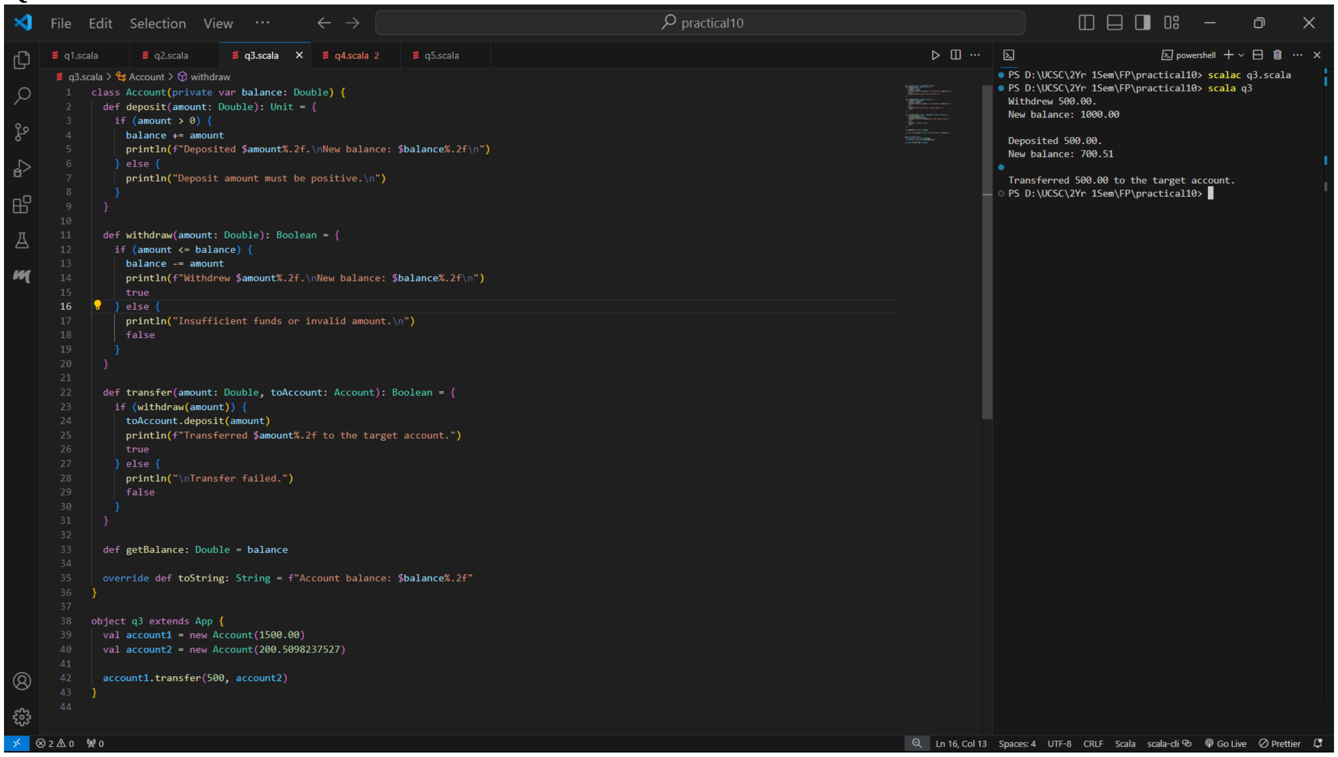
Q2



```scala
class Rationals(n: Int, d: Int) {
  require(d != 0, "Denominator cannot be zero")

  private val g = gcd(n.abs, d.abs)
  val numerator: Int = n / g
  val denominator: Int = d / g

  def this(n: Int) = this(n, 1)

  def sub(that: Rationals): Rationals = {
    new Rationals(
      this.numerator * that.denominator - that.numerator * this.denominator,
      this.denominator * that.denominator
    )
  }

  override def toString: String = s"$numerator/$denominator"

  private def gcd(a: Int, b: Int): Int = if (b == 0) a else gcd(b, a % b)
}

object q2 extends App {
  val x = new Rationals(3, 4)
  val y = new Rationals(5, 8)
  val z = new Rationals(2, 7)

  val result = x.sub(y.sub(z))

  println(s"x - (y - z) = $result")
}
```

```
PS D:\UCSC\2Yr 1Sem\FP\practical10> scalac q2.scala
PS D:\UCSC\2Yr 1Sem\FP\practical10> scala q2
x - (y - z) = 23/56
PS D:\UCSC\2Yr 1Sem\FP\practical10>
```

Q3

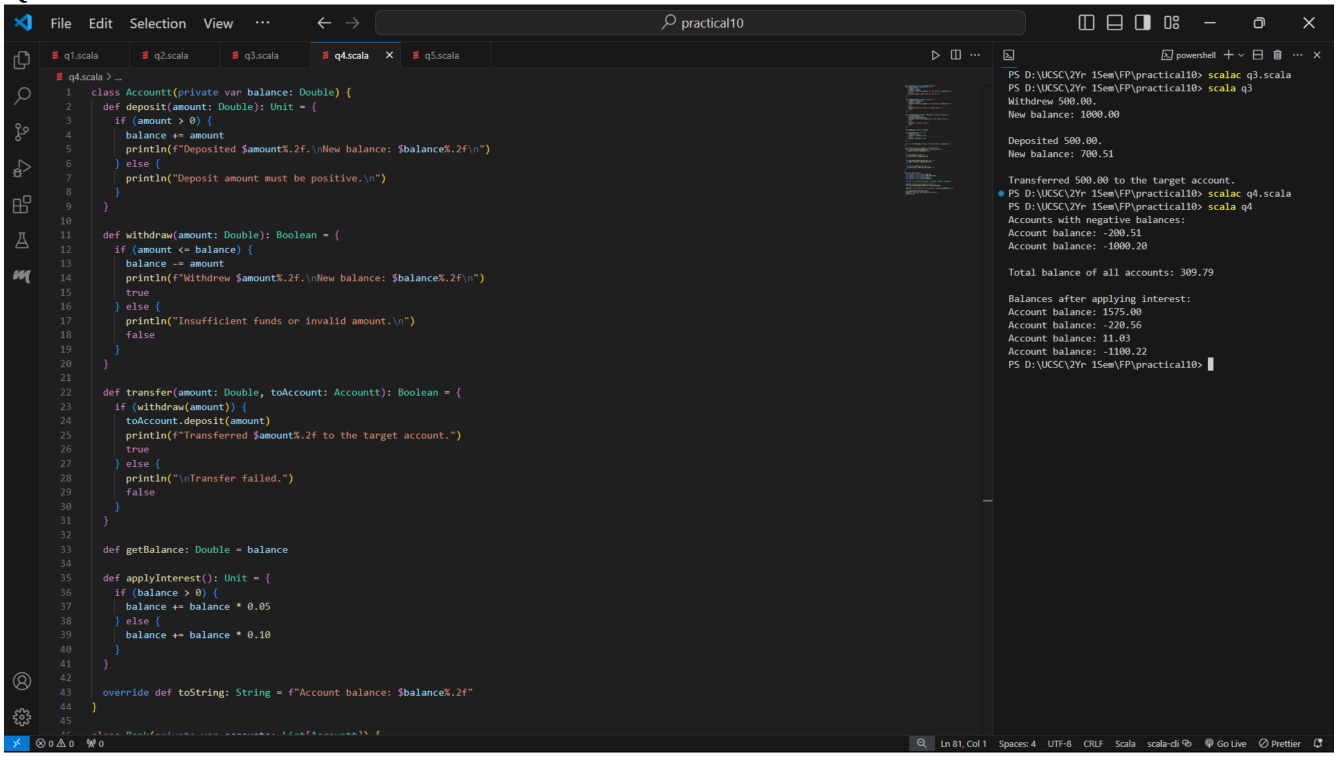


```scala
class Account(private var balance: Double) {
  def deposit(amount: Double): Unit = {
    if (amount > 0) {
      balance += amount
      println(f"Deposited $amount%.2f.\nNew balance: $balance%.2f\n")
    } else {
      println("Deposit amount must be positive.\n")
    }
  }

  def withdraw(amount: Double): Boolean = {
    if (amount <= balance) {
      balance -= amount
      println(f"Withdrew $amount%.2f.\nNew balance: $balance%.2f\n")
      true
    } else {
      println("Insufficient funds or invalid amount.\n")
      false
    }
  }

  def transfer(amount: Double, toAccount: Account): Boolean = {
    if (withdraw(amount)) {
      toAccount.deposit(amount)
      println(f"Transferred $amount%.2f to the target account.")
      true
    } else {
      println("\nTransfer failed.")
      false
    }
  }

  def getBalance: Double = balance

  override def toString: String = f"Account balance: $balance%.2f"
}

object q3 extends App {
  val account1 = new Account(1500.00)
  val account2 = new Account(200.5098237527)

  account1.transfer(500, account2)
}
```

```
PS D:\UCSC\2Yr 1Sem\FP\practical10> scalac q3.scala
PS D:\UCSC\2Yr 1Sem\FP\practical10> scala q3
Withdrew 500.00.
New balance: 1000.00

Deposited 500.00.
New balance: 700.51

Transferred 500.00 to the target account.
PS D:\UCSC\2Yr 1Sem\FP\practical10>
```

```
Ln 16, Col 13   Spaces: 4   UTF-8   CRLF   Scala   scala-cli   Go Live   Prettier
```

Q4

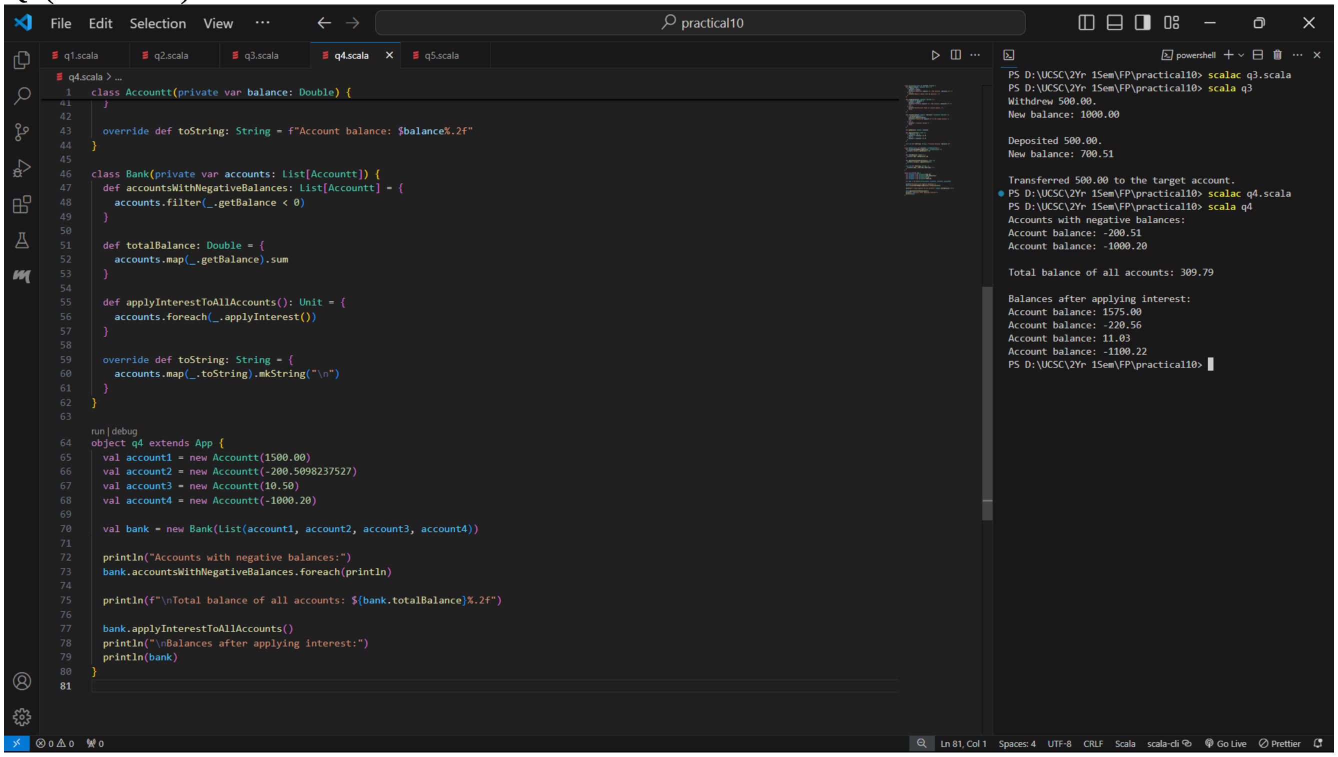


```scala
class Accountt(private var balance: Double) {
  def deposit(amount: Double): Unit = {
    if (amount > 0) {
      balance += amount
      println(f"Deposited $amount%.2f.\nNew balance: $balance%.2f\n")
    } else {
      println("Deposit amount must be positive.\n")
    }
  }

  def withdraw(amount: Double): Boolean = {
    if (amount <= balance) {
      balance -= amount
      println(f"Withdrew $amount%.2f.\nNew balance: $balance%.2f\n")
      true
    } else {
      println("Insufficient funds or invalid amount.\n")
      false
    }
  }

  def transfer(amount: Double, toAccount: Accountt): Boolean = {
    if (withdraw(amount)) {
      toAccount.deposit(amount)
      println(f"Transferred $amount%.2f to the target account.")
      true
    } else {
      println("\nTransfer failed.")
      false
    }
  }

  def getBalance: Double = balance

  def applyInterest(): Unit = {
    if (balance > 0) {
      balance += balance * 0.05
    } else {
      balance += balance * 0.10
    }
  }

  override def toString: String = f"Account balance: $balance%.2f"
}

```

```
PS D:\UCSC\2Yr 1Sem\FP\practical10> scalac q3.scala
PS D:\UCSC\2Yr 1Sem\FP\practical10> scala q3
Withdrew 500.00.
New balance: 1000.00

Deposited 500.00.
New balance: 700.51

Transferred 500.00 to the target account.
PS D:\UCSC\2Yr 1Sem\FP\practical10> scalac q4.scala
PS D:\UCSC\2Yr 1Sem\FP\practical10> scala q4
Accounts with negative balances:
Account balance: -200.51
Account balance: -1000.20

Total balance of all accounts: 309.79

Balances after applying interest:
Account balance: 1575.00
Account balance: -220.56
Account balance: 11.03
Account balance: -1100.22
PS D:\UCSC\2Yr 1Sem\FP\practical10>
```

```
Ln 81, Col 1   Spaces: 4   UTF-8   CRLF   Scala   scala-cli   Go Live   Prettier
```

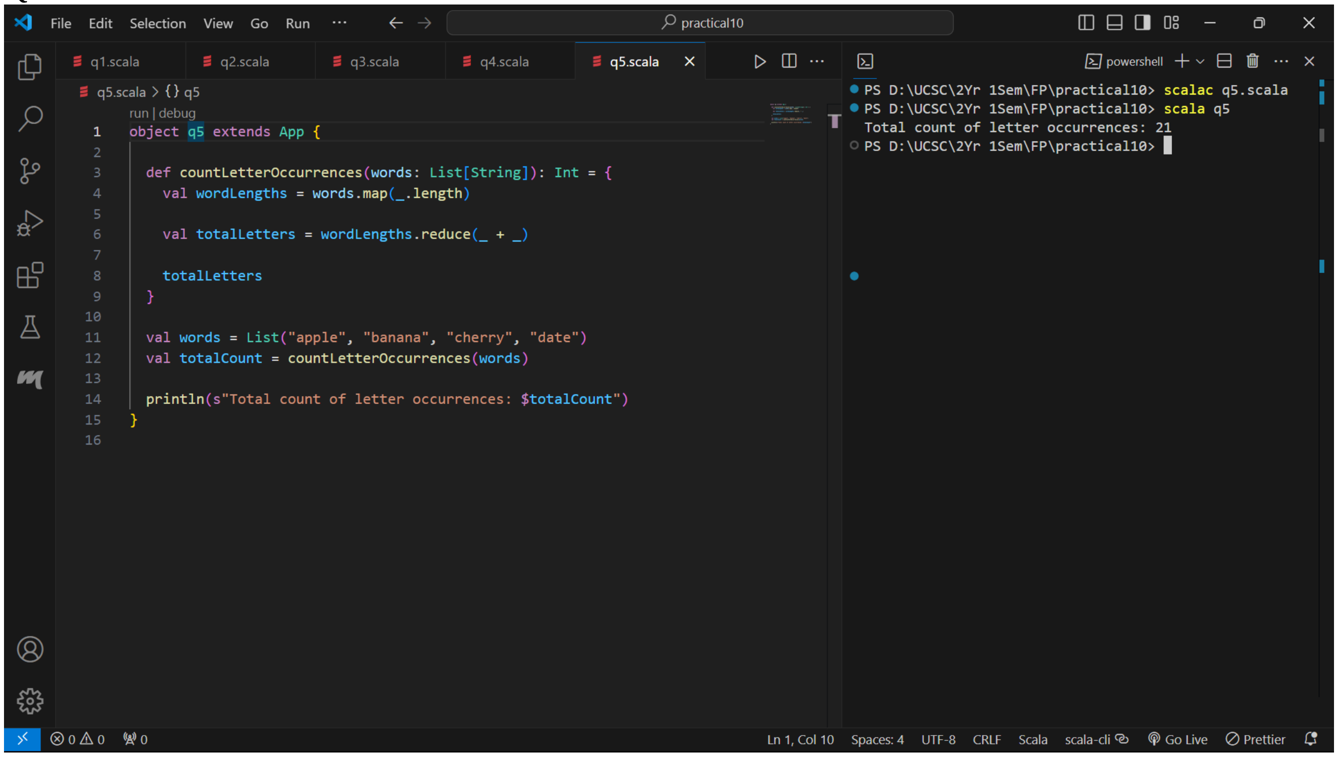
Q4(continued)



```scala
class Accountt(private var balance: Double) {
  }

  override def toString: String = f"Account balance: $balance%.2f"
}

class Bank(private var accounts: List[Accountt]) {
  def accountsWithNegativeBalances: List[Accountt] = {
    accounts.filter(_.getBalance < 0)
  }

  def totalBalance: Double = {
    accounts.map(_.getBalance).sum
  }

  def applyInterestToAllAccounts(): Unit = {
    accounts.foreach(_.applyInterest())
  }

  override def toString: String = {
    accounts.map(_.toString).mkString("\n")
  }
}

run | debug
object q4 extends App {
  val account1 = new Accountt(1500.00)
  val account2 = new Accountt(-200.5098237527)
  val account3 = new Accountt(10.50)
  val account4 = new Accountt(-1000.20)

  val bank = new Bank(List(account1, account2, account3, account4))

  println("Accounts with negative balances:")
  bank.accountsWithNegativeBalances.foreach(println)

  println(f"\nTotal balance of all accounts: ${bank.totalBalance}%.2f")

  bank.applyInterestToAllAccounts()
  println("\nBalances after applying interest:")
  println(bank)
}
```

```
PS D:\UCSC\2Yr 1Sem\FP\practical10> scalac q3.scala
PS D:\UCSC\2Yr 1Sem\FP\practical10> scala q3
Withdrew 500.00.
New balance: 1000.00

Deposited 500.00.
New balance: 700.51

Transferred 500.00 to the target account.
PS D:\UCSC\2Yr 1Sem\FP\practical10> scalac q4.scala
PS D:\UCSC\2Yr 1Sem\FP\practical10> scala q4
Accounts with negative balances:
Account balance: -200.51
Account balance: -1000.20

Total balance of all accounts: 309.79

Balances after applying interest:
Account balance: 1575.00
Account balance: -220.56
Account balance: 11.03
Account balance: -1100.22
PS D:\UCSC\2Yr 1Sem\FP\practical10>
```

Q5



```scala
run | debug
object q5 extends App {

  def countLetterOccurrences(words: List[String]): Int = {
    val wordLengths = words.map(_.length)

    val totalLetters = wordLengths.reduce(_ + _)

    totalLetters
  }

  val words = List("apple", "banana", "cherry", "date")
  val totalCount = countLetterOccurrences(words)

  println(s"Total count of letter occurrences: $totalCount")
}
```

```
PS D:\UCSC\2Yr 1Sem\FP\practical10> scalac q5.scala
PS D:\UCSC\2Yr 1Sem\FP\practical10> scala q5
Total count of letter occurrences: 21
PS D:\UCSC\2Yr 1Sem\FP\practical10>
```