Development scenario 1: Personal Finance Tracker

Assignment-1 introduction and setup and variables and control structures.

Task 1: Install Kotlin and configure IntelliJ IDEA

1. Download and install IntelliJ IDEA from the JetBrains website. The Community Edition is free and sufficient for Kotlin development.

2. During installation, make sure to select the Kotlin plugin.

3. Once installed, open IntelliJ IDEA and create a new Kotlin project:

- Click "New Project"
- Select "Kotlin" from the left panel
- Choose "JVM | IDEA" as the project template
- Name your project and click "Create".

4. In the src folder, create a new Kotlin file named "HelloWorld.kt"

5. Add the following code:

```
fun main() {
    println("Hello, World!")
}
```

6. Run the program by clicking the green play button next to the main function.

If you see "Hello, World!" printed in the console, your Kotlin setup is working correctly.

Task 2: Explore Kotlin REPL

1. In IntelliJ IDEA, go to Tools > Kotlin > Kotlin REPL

2. In the REPL, you can type Kotlin code and see immediate results. Try these examples:

```
// Variable declaration
val name = "Amulya"
println(name)
// Simple arithmetic
```

```
val sum = 5 + 3
println(sum)

// String template
println("The sum is $sum")

// Function definition and call
fun greet(name: String) = "Hello, $name!"
println(greet("virat"))
```

Task 3: Create a Transaction class

Create a new Kotlin file named "Transaction.kt" and add the following code:

```
import java.time.LocalDate
data class Transaction(
    val amount: Double,
    val date: LocalDate,
    val category: String
)
```

This creates a Transaction class with three properties: amount, date, and category.

Task 4: Implement control structures to categorize transactions

Create a new Kotlin file named "TransactionCategorizer.kt" and add the following code:

```
fun categorizeTransaction(transaction: Transaction): String {
    return when (transaction.category.toLowerCase()) {
        "groceries", "restaurant" -> "Food"
```

```
        "electricity", "water", "gas" -> "Utilities"
        "movies", "games", "sports" -> "Entertainment"
        else -> "Other"
    }
}
fun main() {
    val transactions = listOf(
        Transaction(50.0, LocalDate.now(), "Groceries"),
        Transaction(30.0, LocalDate.now(), "Movies"),
        Transaction(100.0, LocalDate.now(), "Electricity"),
        Transaction(20.0, LocalDate.now(), "Books")
    )
    for (transaction in transactions) {
        val category = categorizeTransaction(transaction)
        println("Transaction of ${transaction.amount} on ${transaction.date} categorized as: $category")
    }
}
```

To run this code, make sure to import the Transaction class at the top of the file:

```
import java.time.LocalDate
```

Run the main function in TransactionCategorizer.kt to see the results.

Assignment-2

Certainly! I'll guide you through each of these tasks for setting up and starting to work with Kotlin. Let's go through them step by step.

Task 1: Set up the Kotlin development environment and write a simple Kotlin script

1. Install IntelliJ IDEA

2. Create a new Kotlin project in IntelliJ IDEA.

3. Create a new Kotlin file named "Setup.kt" and add the following code:

```
fun main() {
    println("Kotlin setup successful!")
}
```

Run this script. If you see "Kotlin setup successful!" in the output, your environment is correctly set up.

Task 2: Experiment with Kotlin's string templates

Create a new file named "WelcomeMessages.kt" and add the following code:

```
fun main() {
    val name = "Ammu"
    val eventName = "Tech Conference"
    val date = "July 20, 2024"

    println("Welcome, $name!")
    println("You're invited to the $eventName on $date.")
    println("We're excited to have you join us, ${name.uppercase()}!")

    val daysUntilEvent = 5
    println("Only $daysUntilEvent days left until the ${eventName.lowercase()}!")
}
```

This demonstrates various ways to use string templates in Kotlin.

Task 3: Define data types for event details

Create a file named "EventTypes.kt" and add the following code:

```
import java.time.LocalDate

data class Event(
    val name: String,
    val date: LocalDate,
    val attendeeCount: Int
)

fun main() {
    val techConference = Event("Tech Conference", LocalDate.of(2024, 7, 20), 500)
    println("Event: ${techConference.name}")
    println("Date: ${techConference.date}")
    println("Expected Attendees: ${techConference.attendeeCount}")
}
```

This defines a data class to represent an event and demonstrates its usage.

Task 4: Implement a basic user input flow for creating events

Create a file named "EventCreator.kt" and add the following code:

```
import java.time.LocalDate
import java.time.format.DateTimeFormatter

fun main() {
    println("Let's create a new event!")

    print("Enter event name: ")
    val name = readLine() ?: ""
```

```
print("Enter event date (YYYY-MM-DD): ")
val dateString = readLine() ?: ""
val date = LocalDate.parse(dateString, DateTimeFormatter.ISO_DATE)

print("Enter expected attendee count: ")
val attendeeCountString = readLine() ?: "0"
val attendeeCount = attendeeCountString.toIntOrNull() ?: 0
val event = Event(name, date, attendeeCount)

println("\nEvent created successfully!")
println("Event details:")
println("Name: ${event.name}")
println("Date: ${event.date}")
println("Expected Attendees: ${event.attendeeCount}")
when {
    event.attendeeCount < 50 -> println("This is a small event.")
    event.attendeeCount < 200 -> println("This is a medium-sized event.")
    else -> println("This is a large event!")
}
if (event.date.isAfter(LocalDate.now().plusMonths(1))) {
    println("You have plenty of time to prepare for this event.")
} else {
    println("The event is coming up soon! Make sure you're prepared.")
}
```

```
}
```

This script demonstrates user input, creating an Event object, and using if and when statements to provide additional information about the event.