

SULEIMAN DAHIRU

FCP/CSE/22/2002

MOBILE APPLICATION DEVELOPMENT

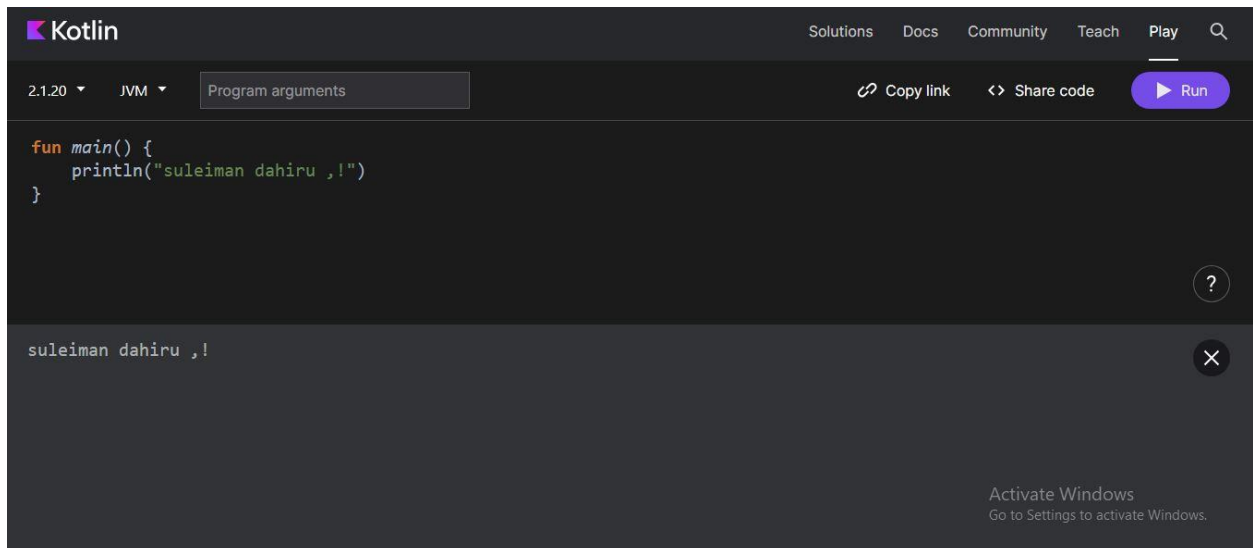
ASSIGNMENT

PRACTICAL PART

Task 1: Hello World

Write a Kotlin program that prints Hello, [Your Name]! to the console.

Expected Output: Hello, Suleiman dahiru,

The image shows a screenshot of the Kotlin Playground web application. At the top, there's a header with the Kotlin logo and navigation links: Solutions, Docs, Community, Teach, Play, and a search icon. Below the header, there's a toolbar with version '2.1.20', JVM target, a 'Program arguments' input field, and buttons for 'Copy link', 'Share code', and 'Run'. The main area contains a Kotlin code snippet:

```
fun main() {  
    println("suleiman dahiru ,!")  
}
```

 Below the code editor, the output console shows the result:

```
suleiman dahiru ,!
```

 There are also help and close icons for the output console. At the bottom right, there's a 'Activate Windows' watermark.

Explanation:

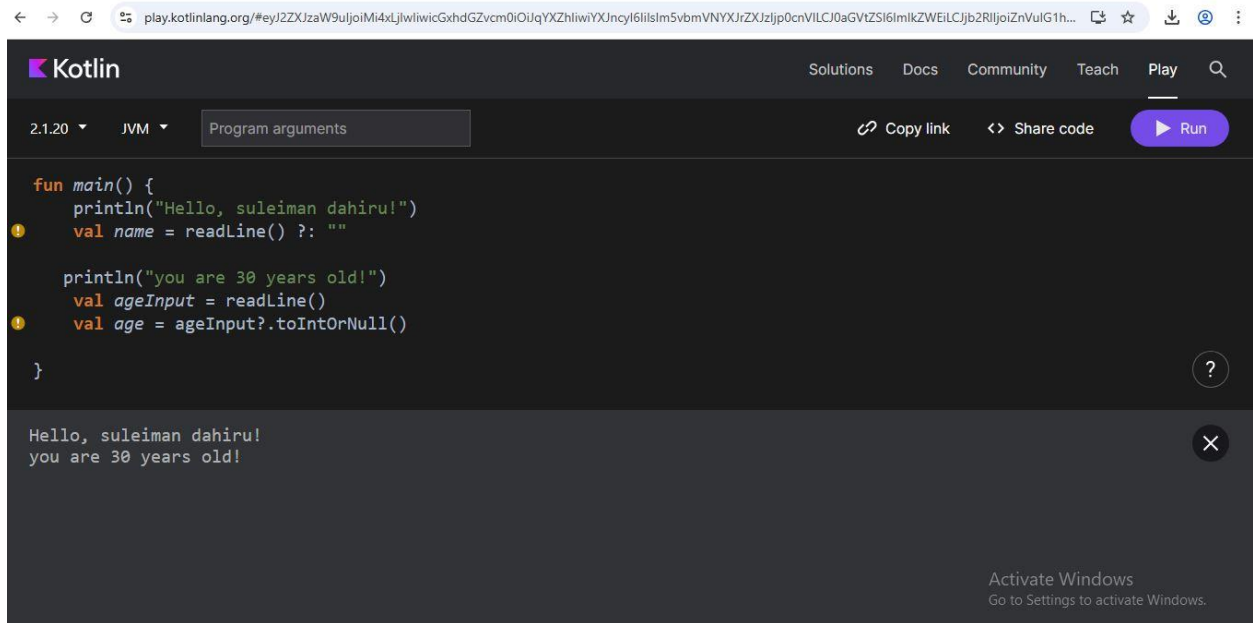
fun main() defines the main function where the program starts executing.

println() prints the text inside the parentheses to the console, followed by a newline.

Task 2: User Input

Create a Kotlin program that asks the user to input their name and age, then prints a greeting like:

Hello Suleiman dahiru!, you are 30 years old!



The screenshot shows the Kotlin Playground interface. At the top, there's a navigation bar with 'Kotlin' logo and links for Solutions, Docs, Community, Teach, and Play. Below this, there's a dropdown menu for '2.1.20' and 'JVM', and a text input field containing 'Program arguments'. To the right of the input field are buttons for 'Copy link', 'Share code', and a 'Run' button. The main area contains a Kotlin code snippet:

```
fun main() {  
    println("Hello, suleiman dahiru!")  
    val name = readLine() ?: ""  
  
    println("you are 30 years old!")  
    val ageInput = readLine()  
    val age = ageInput?.toIntOrNull()  
  
}
```

Below the code, the output is displayed: 'Hello, suleiman dahiru!' and 'you are 30 years old!'. There are also icons for help (?) and close (X) next to the output. At the bottom right, there's a 'Activate Windows' watermark.

Explanation:

- The program prompts the user to enter their name.
- It then asks for the age and tries to convert it to an integer.
- If the age is valid, it prints: "Hello, Mustapha Lawal Daura, you are 21 years old!"
- If the age is invalid, it notifies the user.
- `readLine()` reads user input from the console.
- `toIntOrNull()` safely converts the string to an integer (returns null if it fails).

Task 3: Conditional Statements

Write a program that checks if a given number is even or odd.

Sample Output:

Enter a number: 10

10 is an even number

The screenshot shows the Kotlin Playground web interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, and Play. Below this, the Kotlin logo is on the left, and version 2.1.20 and JVM target are selected. A text box contains 'Program arguments'. On the right, there are buttons for 'Copy link', 'Share code', and 'Run'. The main area contains a Kotlin code snippet:

```
fun main() {
    print("Enter a number: ")
    val number = readLine()?.toIntOrNull() ?: 0

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

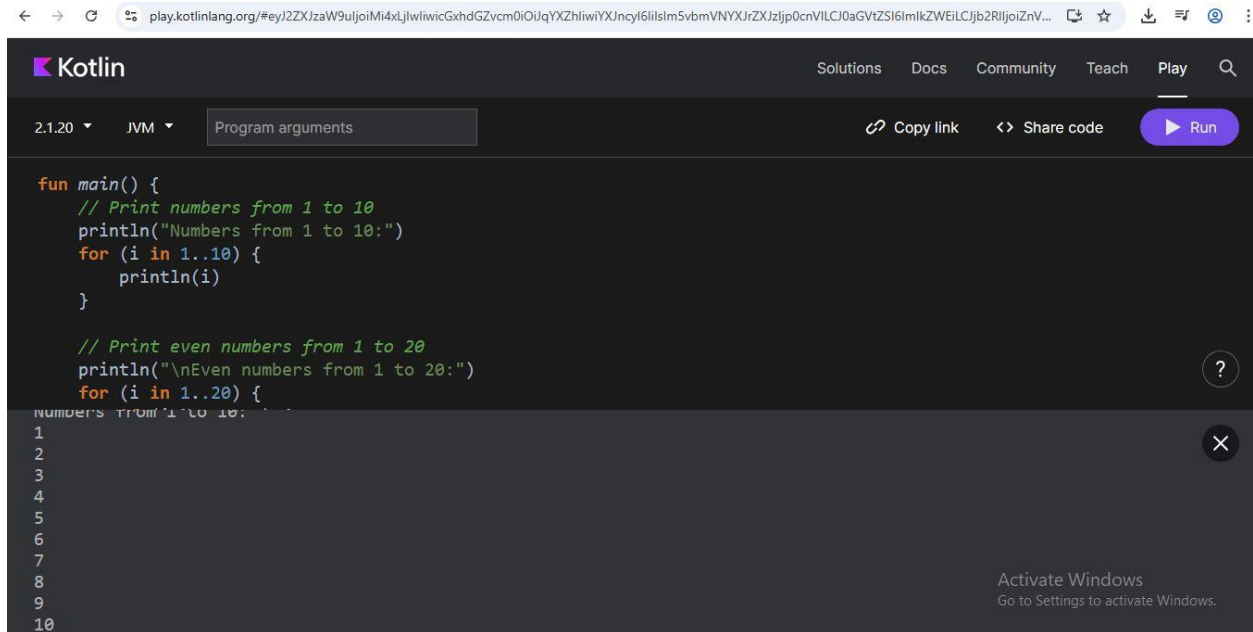
 Below the code, the output shows 'Enter a number: 0 is even.'. A help icon (?) is on the right of the code editor, and a close icon (X) is on the right of the output area. At the bottom right, there's a 'Activate Windows' watermark.

Explanation:

- The program asks the user to input a number using `readLine()`.
- It attempts to convert the input to an integer using `toIntOrNull()`. If the input is not a valid integer, it defaults to 0.
- It checks if the number is even by using the modulo operator (%). If the remainder of the division by 2 is 0, the number is even.
- It prints whether the number is even or odd.

Task 4: Loops and Ranges

Print numbers from 1 to 10 using a for loop. Then, print only even numbers from 1 to 20



```
fun main() {  
    // Print numbers from 1 to 10  
    println("Numbers from 1 to 10:")  
    for (i in 1..10) {  
        println(i)  
    }  
  
    // Print even numbers from 1 to 20  
    println("\nEven numbers from 1 to 20:")  
    for (i in 1..20) {  
        if (i % 2 == 0) {  
            println(i)  
        }  
    }  
}
```

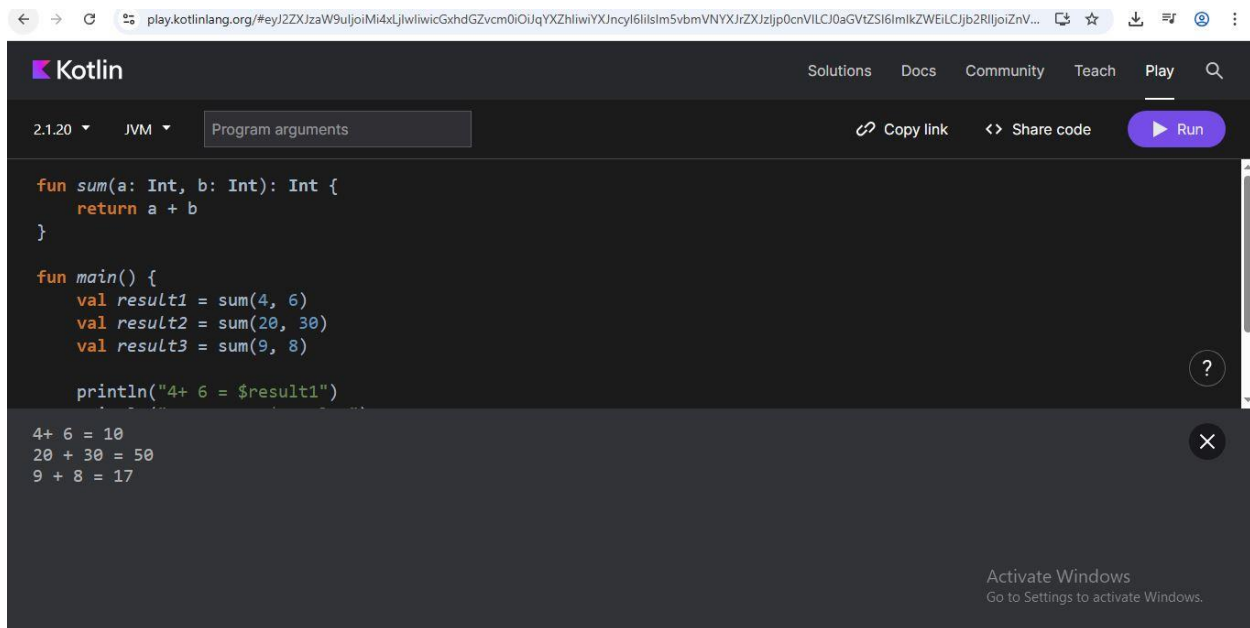
Numbers from 1 to 10:
1
2
3
4
5
6
7
8
9
10

Explanation:

- The first loop iterates from 1 to 10 using the `..` operator and prints each number.
- The second loop iterates from 1 to 20 and checks if each number is even using the modulo operator (`%`). If the number is even, it prints the number.

Task 5: Functions

Write a function `sum(a: Int, b: Int): Int` that returns the sum of two numbers. Call it with different values and display the result.



The screenshot shows the Kotlin Playground interface. At the top, there's a navigation bar with 'Kotlin' logo and links for Solutions, Docs, Community, Teach, and Play. Below this, there's a header with '2.1.20', 'JVM', and a text input field containing 'Program arguments'. To the right of the input field are buttons for 'Copy link', 'Share code', and a 'Run' button. The main area contains a Kotlin code snippet:

```
fun sum(a: Int, b: Int): Int {  
    return a + b  
}  
  
fun main() {  
    val result1 = sum(4, 6)  
    val result2 = sum(20, 30)  
    val result3 = sum(9, 8)  
  
    println("4+ 6 = $result1")  
}
```

Below the code, the output is displayed:

```
4+ 6 = 10  
20 + 30 = 50  
9 + 8 = 17
```

At the bottom right, there's a watermark that says 'Activate Windows Go to Settings to activate Windows.'

Explanation:

- The sum function takes two Int parameters, a and b, and returns their sum.
- In the main function, we call sum with different values and store the results variables.
- We then print the results using string templates.

Task 6: Arrays

Create an array of 5 names. Loop through the array and print each name with a greeting.

Sample Output:

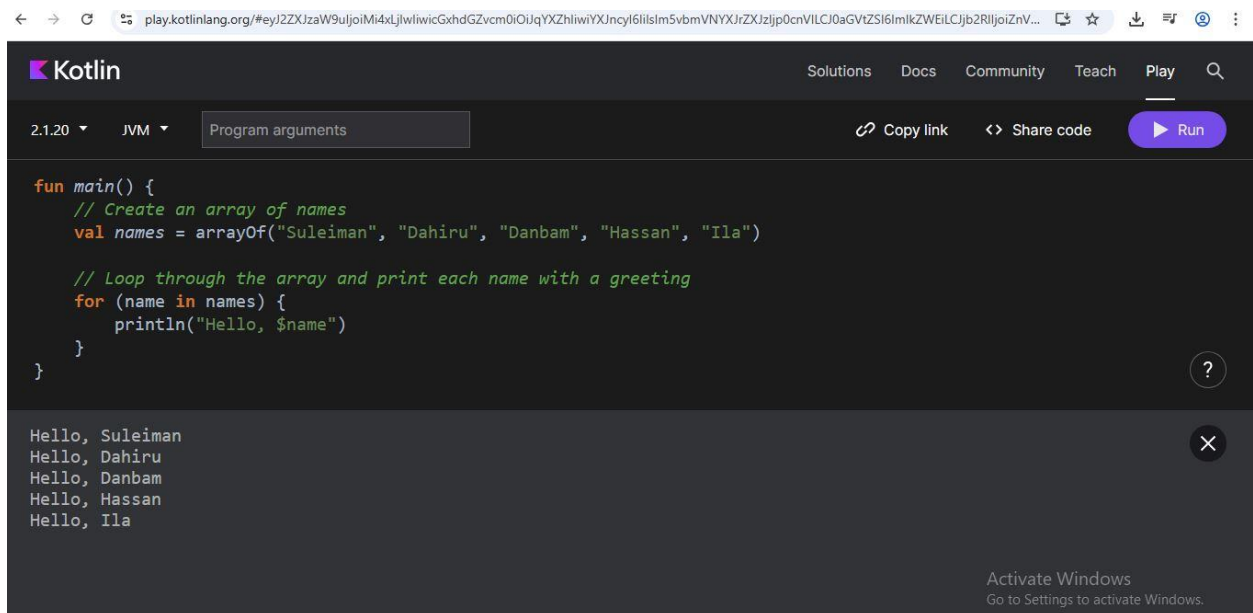
Hello, suleiman

Hello, dahiru

Hello, dambam

Hello, hassan

Hello, ila



The screenshot shows the Kotlin Playground web application. The browser address bar displays the URL: `play.kotlinlang.org/#eyJ2ZXJzaW9uIjoIbW9uIiwicGxhdGZvcml0IjoiYXZlbiwiYXNcyI6IiIsIm5vbmVNYXJrZXIzIjp0cnVILCJ0aGVtZSI6ImlkZWElLCJjb2RlIjoIZnV...`. The Kotlin Playground interface includes a header with the Kotlin logo and navigation links (Solutions, Docs, Community, Teach, Play, Search). Below the header, the version (2.1.20) and JVM target are selected, and the program arguments are set to "Program arguments". The code editor contains the following Kotlin code:

```
fun main() {  
    // Create an array of names  
    val names = arrayOf("Suleiman", "Dahiru", "Danbam", "Hassan", "Ila")  
  
    // Loop through the array and print each name with a greeting  
    for (name in names) {  
        println("Hello, $name")  
    }  
}
```

The output window at the bottom displays the results of the program execution:

```
Hello, Suleiman  
Hello, Dahiru  
Hello, Danbam  
Hello, Hassan  
Hello, Ila
```

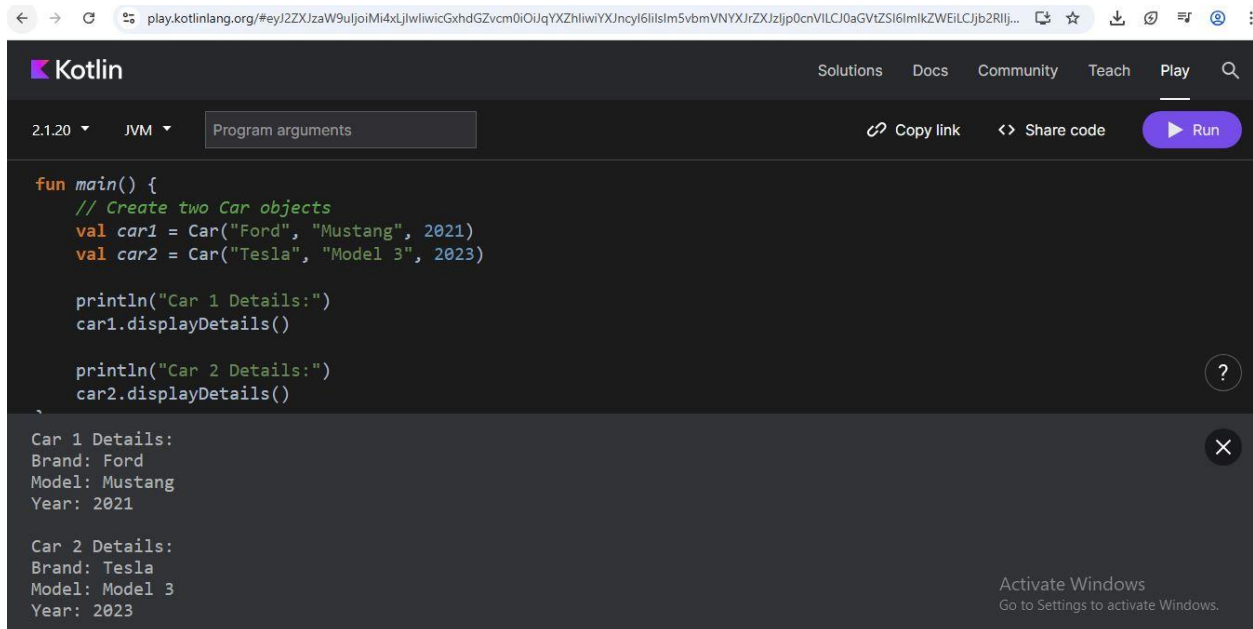
An "Activate Windows" watermark is visible in the bottom right corner of the screenshot.

Explanation:

- Create an array of names
- Loop through the array and print each name with a greeting

Task 7: Classes and Objects

Define a class `Car` with properties `brand`, `model`, and `year`. Add a function `displayDetails()` that prints the car details. Create at least two objects of this class



The screenshot shows the Kotlin Playground interface. The top bar includes the Kotlin logo, navigation links (Solutions, Docs, Community, Teach, Play), and a search icon. Below the bar, there are tabs for '2.1.20', 'JVM', and 'Program arguments'. A 'Run' button is visible on the right. The main editor area contains the following Kotlin code:

```
fun main() {  
    // Create two Car objects  
    val car1 = Car("Ford", "Mustang", 2021)  
    val car2 = Car("Tesla", "Model 3", 2023)  
  
    println("Car 1 Details:")  
    car1.displayDetails()  
  
    println("Car 2 Details:")  
    car2.displayDetails()  
}
```

The output area at the bottom shows the results of running the code:

```
Car 1 Details:  
Brand: Ford  
Model: Mustang  
Year: 2021  
  
Car 2 Details:  
Brand: Tesla  
Model: Model 3  
Year: 2023
```

An 'Activate Windows' watermark is visible in the bottom right corner of the output area.

Explanation:

- We define a `Car` class with properties `brand`, `model`, and `year`.
- The `displayDetails` function prints the car's details.
- In the `main` function, we create two `Car` objects, `car1` and `car2`.
- We call the `displayDetails` function on each object to print their details.

Task 8: Inheritance

Create a base class `Person` with properties `name` and `age`. Create a subclass `Student` that adds a property `grade`. Add methods to print each detail.

The screenshot shows the Kotlin Playground interface. At the top, there's a navigation bar with 'Kotlin' logo and links for Solutions, Docs, Community, Teach, and Play. Below this, a dropdown menu shows '2.1.20' and 'JVM', and a text input field contains 'Program arguments'. To the right are buttons for 'Copy link', 'Share code', and a 'Run' button. The main area contains the following Kotlin code:

```
fun main() {  
    val person = Person("Suleiman Dahiru", 22)  
    person.printPersonDetails()  
  
    println()  
  
    val student = Student("Suleiman Dahiru", 22, 90)  
    student.printStudentDetails()  
}
```

Below the code, the output is displayed in a dark-themed box:

```
Name: Suleiman Dahiru  
Age: 22  
  
Name: Suleiman Dahiru  
Age: 22  
Grade: 90
```

At the bottom right of the output box, there's a message: 'Activate Windows Go to Settings to activate Windows.' The Windows taskbar is visible at the very bottom, showing the search bar, task view button, and several application icons.

Explanation:

- We define a base class **Person** with properties **name** and **age**, and a method **printDetails** to print these properties. The **open** keyword is used to allow inheritance.
- The **Student** class is a subclass of **Person** and adds a **grade** property. It also defines a **printStudentDetails** method to print all properties, including grade.
- In the **Student** class, we use **super** to access the properties and methods of the **Person** class.
- We override the **toString** method to provide a string representation of the **Student** object, including the grade property.
- In the example usage section, we create instances of **Person** and **Student**, and demonstrate how to print their details using the **printDetails** and **printStudentDetails** methods.

Task 9: Collections and Map

Create a map with student names as keys and their scores as values. Print students who scored above 70.

The screenshot shows the Kotlin Playground interface. At the top, there's a navigation bar with 'Kotlin' logo and links for Solutions, Docs, Community, Teach, and Play. Below this, the version '2.1.20' and 'JVM' are selected, along with a 'Program arguments' field. On the right, there are buttons for 'Copy link', 'Share code', and 'Run'. The main code editor contains the following Kotlin code:

```
run main() {  
    val studentScores = mapOf(  
        "Suleiman" to 85,  
        "Muhammad" to 90,  
        "Dahiru" to 60,  
        "Danbam" to 78,  
        "Ila" to 92  
    )  
  
    println("Students who scored above 70:")  
    studentScores.filter { it.value > 70 }  
        .forEach { println("${it.key} : ${it.value}") }  
}
```

Below the code editor, the output is displayed:

```
Students who scored above 70:  
Suleiman: 85  
Muhammad: 90  
Danbam: 78  
Ila: 92
```

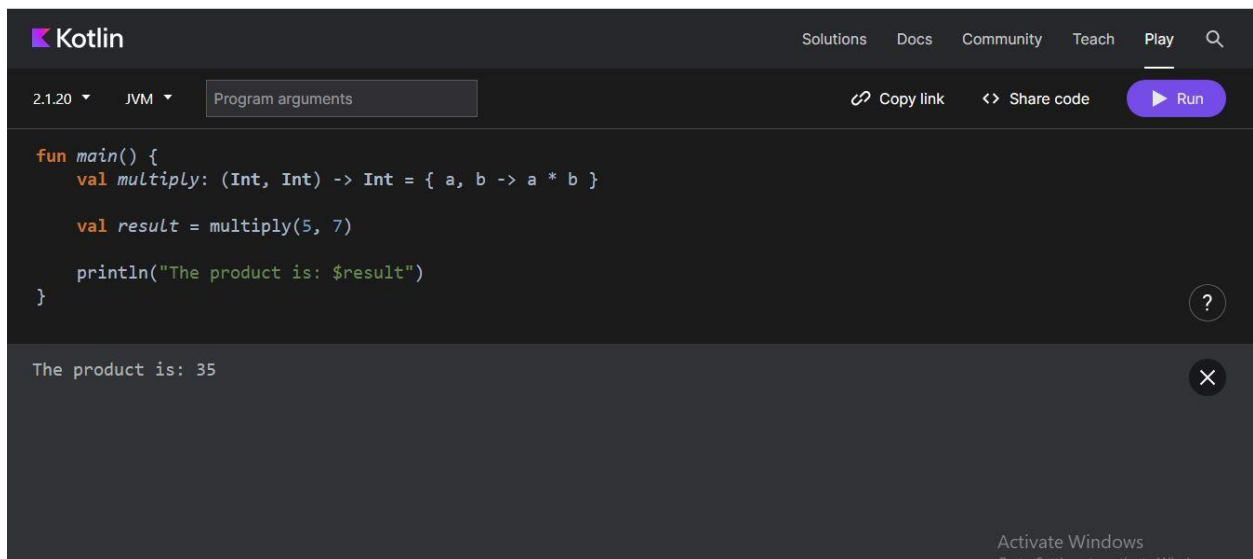
An 'Activate Windows' watermark is visible in the bottom right corner of the screenshot.

Explanation:

- This creates an immutable map (mapOf) named studentScores.
- The map contains key-value pairs where the key is the student's name (a string) and the value is their score (an integer).
- The to keyword is used to create these key-value pairs.
- This uses the filter function to create a new map that only includes the students who scored above 70.
- The lambda expression { it.value > 70 } is the condition for filtering. it refers to each key-value pair in the map, and it.value refers to the score.
- The resulting map will only include the pairs where the score is greater than 70.
- This uses the forEach function to iterate over the filtered map and print each student's name and score.

- The lambda expression `{ println("${it.key}: ${it.value}") }` is executed for each pair in the filtered map.
- `${it.key}` refers to the student's name, and `${it.value}` refers to their score.

Task 10: Lambda Expression Write a lambda expression that takes two integers and returns their product. Call it and print the result.



The screenshot shows the Kotlin Playground web interface. At the top, there's a navigation bar with links for Solutions, Docs, Community, Teach, and Play. Below this, the Kotlin version (2.1.20) and JVM target are selected. A text input field for 'Program arguments' is present. On the right, there are buttons for 'Copy link', 'Share code', and 'Run'. The main code editor contains the following Kotlin code:

```
fun main() {  
    val multiply: (Int, Int) -> Int = { a, b -> a * b }  
  
    val result = multiply(5, 7)  
  
    println("The product is: $result")  
}
```

Below the code editor, the output console shows the result: "The product is: 35". A help icon (?) is visible next to the code editor, and a close icon (X) is next to the output console. At the bottom right, there is a watermark for "Activate Windows" with a link to "Go to Settings to activate Windows".

Explanation:

- **(Int, Int) -> Int** defines the type of the lambda expression, which takes two integers as input and returns an integer.
- **{ a, b -> a * b }** is the lambda expression itself, where a and b are the input parameters, and a * b is the expression that gets evaluated and returned.
- **multiply(5, 7)** calls the lambda expression with the arguments 5 and 7, and the result is stored in the result variable.
- **Finally, the result is printed to the console.**