Competency Based Learning Material (CBLM)

Android Mobile Application Development

Level-4

Module: Working with Kotlin Basics

Code: CBLM-OU-ICT-AMAD-01-L4-V1

National Skills Development Authority
Prime Minister's Office
Government of the People's Republic of Bangladesh

Copyright

Approved by __ th Authority Meeting of NSDA Held on --------------

How to use this Competency Based Learning Material (CBLM)

The module, Work with Kotlin Basics contains training materials and activities for you to complete. These activities may be completed as part of structured classroom activities or you may be required you to work at your own pace. These activities will ask you to complete associated learning and practice activities in order to gain knowledge and skills you need to achieve the learning outcomes.

1. Review the **Learning Activity** page to understand the sequence of learning activities you will undergo. This page will serve as your road map towards the achievement of competence.

2. Read the **Information Sheets.** This will give you an understanding of the jobs or tasks you are going to learn how to do. Once you have finished reading the **Information Sheets** complete the questions in the **Self-Check.**

3. **Self-**Checks are found after each **Information Sheet**. **Self-Checks** are designed to help you know how you are progressing. If you are unable to answer the questions in the **Self-Check** you will need to re-read the relevant **Information Sheet**. Once you have completed all the questions check your answers by reading the relevant **Answer Keys** found at the end of this module.

4. Next move on to the **Job Sheets. Job Sheets** provide detailed information about *how to do the job* you are being trained in. Some **Job Sheets** will also have a series of **Activity Sheets**. These sheets have been designed to introduce you to the job step by step. This is where you will apply the new knowledge you gained by reading the Information Sheets. This is your opportunity to practice the job. You may need to practice the job or activity several times before you become competent.

5. Specification **sheets**, specifying the details of the job to be performed will be provided where appropriate.

6. A review of competency is provided on the last page to help remind if all the required assessment criteria have been met. This record is for your own information and guidance and is not an official record of competency

When working though this Module always be aware of your safety and the safety of others in the training room. Should you require assistance or clarification please consult your trainer or facilitator.

When you have satisfactorily completed all the Jobs and/or Activities outlined in this module, an assessment event will be scheduled to assess if you have achieved competency in the specified learning outcomes. You will then be ready to move onto the next Unit of Competency or Module

Table of Contents

Module Content

Unit of Competency	**Work with Kotlin Basics**
Unit Code	**OU-ICT-AMAD-01-L4-V1**
Module Title	**Working with Kotlin Basics**
Module Descriptor	This unit covers the knowledge, skills and attitudes required of a worker to work with Kotlin basics. It specifically includes the tasks of creating Kotlin data types and using Kotlin.
Nominal Hours	4**0** Hours
Lerning Outcome	After completing the practice of the module, the trainees will be able to perform the following jobs: 1. Create Kotlin data types 2. Use Kotlin

Assessment Criteria

1. Basic anatomy of a Kotlin program for android application is described.
2. Kotlin variables are presented.
3. Input and output syntax are applied
4. Conditional statement is explained.
5. Conditional loops are used.
6. Data types in Kotlin is created.
7. An application is created.
8. The numbers are seen in display.
9. Methods in Kotlin is implemented.

Learning Outcome 1: Create Kotlin data types

Assessment Criteria	1. Basic anatomy of a Kotlin program for android application is described. 2. Kotlin variables are presented. 3. Input and output syntax are applied 4. Conditional statement is explained. 5. Conditional loops are used. 6. Data types in Kotlin is created.
Conditions and Resources	• Actual workplace or training environment • CBLM • Handouts • Job related tools, equipment, and materials • Multimedia Projector • Paper, Pen, Pencil, and Eraser • Internet Facilities • Whiteboard and Marker
Contents	1. Basic anatomy of a Kotlin program for android application 2. Kotlin variables ▪ Integer ▪ String ▪ Double ▪ Float ▪ Date Time 3. Access modifier ▪ Public ▪ Private ▪ Protected 4. Input and output syntax 5. Conditional statement 6. Conditional loops 7. Data types in Kotlin
Activities/job/Task	1. Declare and initialize variables of different types (e.g., integer, string, boolean) in Kotlin. Print their values to the console. 2. Write a Kotlin program that takes user input for them name and age. Then, output a personalized message based on the input. 3. Implement a Kotlin program that checks if a given number is positive, negative, or zero using an if-else statement. 4. Write a Kotlin program that uses a while loop to print the first 5 even numbers (starting from 2).

	5. Define a data class in Kotlin representing a "Person" with attributes such as name, age, and email. Create an instance of this class and print its details.
Training Methods	• Blended • Discussion • Presentation • Demonstration • Guided Practice • Individual Practice • Project Work • Problem Solving • Brainstorming
Assessment Methods	Assessment methods may include but not limited to • Written Test • Demonstration • Oral Questioning

Learning Experience 1: Create Kotlin data types

In order to achieve the objectives stated in this learning guide, you must perform the learning steps below. Beside each step are the resources or special instructions you will use to accomplish the corresponding activity.

Learning Activities	Recourses/Special Instructions
1. Trainee will ask the instructor about the learning materials	1. Instructor will provide the learning materials Create Kotlin data types
2. Read the Information sheet and complete the Self Checks & Check answer sheets on "Create Kotlin data types"	2. Read Information sheet 1: Create Kotlin data types 3. Answer Self-check 1: Create Kotlin data types 4. Check your answer with Answer key 1: Create Kotlin data types
3. Read the Job/Task Sheet and Specification Sheet and perform job/Task	5. Job/Task Sheet and Specification Sheet **Job Sheet 1.1:** Declare and initialize variables of different types (e.g. integer, string, boolean) in Kotlin. Print their values to the console. **Specification Sheet 1.1:** Declare and initialize variables of different types (e.g. integer, string, boolean) in Kotlin. Print their values to the console. **Job Sheet 1.2:** Write a Kotlin program that takes user input for their name and age. Then, output a personalized message based on the input. **Specification Sheet 1.2:** Write a Kotlin program that takes user input for their name and age. Then, output a personalized message based on the input. **Job Sheet 1.3:** Implement a Kotlin program that checks if a given number is positive, negative, or zero using an if-else statement. **Specification Sheet 1.3:** Implement a Kotlin program that checks if a given number is positive, negative, or zero using an if-else statement.

	Job Sheet 1.4: Write a Kotlin program that uses a while loop to print the first 5 even numbers (starting from 2). **Specification Sheet 1.4**: Write a Kotlin program that uses a while loop to print the first 5 even numbers (starting from 2). **Job Sheet 1.5:** Define a data class in Kotlin representing a "Person" with attributes such as name, age, and email. Create an instance of this class and print its details. **Specification Sheet 1.5**: Define a data class in Kotlin representing a "Person" with attributes such as name, age, and email. Create an instance of this class and print its details.

Information Sheet 1: Create Kotlin data types

Learning Objective: After completion of this information sheet, the learners will be able to explain, define and interpret the following contents

1.1 Basic anatomy of a Kotlin program for android application
1.2 Kotlin variables
1.3 Input and output syntax
1.4 Conditional statement
1.5 Conditional loops
1.6 Data types in Kotlin

1.1 Basic anatomy of a Kotlin program for android application

A Kotlin program for an Android application typically consists of several components that work together to create a functional and user-friendly app. Here is a basic overview of the anatomy of such a program:

a. Project Structure: A Kotlin Android application is structured as an Android Studio project, which includes various folders and files necessary for the app's development. Key folders are 'java' for Java code, 'res' for resources like layouts, drawable, and strings, 'manifests' for AndroidManifest.xml, and 'kotlin' for Kotlin files.

b. Main Activity: The main entry point of the application is the 'MainActivity' class, which is written in Kotlin. This class extends the `AppCompatActivity’ and overrides its onCreate() method. In this method, you set the content view (activity_main.xml layout) and add any necessary components, such as buttons or text views.

c. Layout Files: Android applications use XML layout files to define the visual structure and user interface elements of the app. These files are located in the 'res/layout' folder. The main layout file is usually named 'activity_main.xml,' and it contains views like buttons, text fields, and images.

d. Kotlin Code: Kotlin code files have the .kt extension and are placed in the 'kotlin' folder. These files contain the logic and functionality of the app, such as event handling, data manipulation, and interactions with other app components.

e. Resources: The 'res' folder contains various resources like strings, colors, dimensions, and drawable. These resources can be accessed and used in the Kotlin code to provide localized content, customize the app's appearance, and add graphics.

f. Manifest File: The AndroidManifest.xml file, located in the 'manifests' folder, is a crucial part of the app. It declares the app's components, permissions, activities, services, and other configurations required by Android. This file is essential for the app to function properly and be recognized by the Android system.

g. Build. Gradle Files: There are two builds. Gradle files in an Android project: 'build. Gradle (Module: app)' and 'build. Gradle (Project: your project name)'. These files contain configuration settings for app dependencies, plugins, and build types. They help manage the app's development process and ensure proper compilation and packaging.

h. **Dependencies and Libraries:** Android applications often rely on external libraries and dependencies for features like networking, data storage, or UI components. These are managed through the build. Gradle files and can be added using the 'dependencies' block.

i. **Testing:** Kotlin allows for unit testing and code verification using the built-in JUnit and Mockito libraries. Test files are usually placed in the 'test' folder and help ensure the app's functionality and stability.

j. **Version Control and Collaboration:** Most Android development projects use version control systems like Git, which allows developers to track changes, collaborate, and manage releases. Git repositories are often hosted on platforms like GitHub or Bitbucket.

1.2 Kotlin variables

Variables are an important part of any programming. They are the names you give to computer memory locations which are used to store values in a computer program and later you use those names to retrieve the stored values and use them in your program.

Kotlin variables are created using either **var** or **val** keywords and then an equal sign **=** is used to assign a value to those created variables.

Syntax

Following is a simple syntax to create two variables and then assign them different values:

```
var name = "Zara Ali"
var age = 19
var height = 5.2
```

a. Integer Types

Byte
The Byte data type can store whole numbers from -128 to 127
Example

```
val myNum: Byte = 100
println(myNum)
```

Short

The Short data type can store whole numbers from -32768 to 32767:
Example

```
val myNum: Short = 5000
println(myNum)
```

Int

The Int data type can store whole numbers from -2147483648 to 2147483647:

Example

```
val myNum: Int = 100000
println(myNum)
```

Long

The Long data type can store whole numbers from -9223372036854775807 to 9223372036854775807. This is used when Int is not large enough to store the value. Optionally, you can end the value with an "L":

Example

```
val myNum: Long = 15000000000L
println(myNum)
```

Int VS Long

A whole number is an Int as long as it is up to 2147483647. If it goes beyond that, it is defined as **Long:**

Example

```
val myNum1 = 2147483647  // Int
val myNum2 = 2147483648  // Long
```

b. Floating Point Types

Floating point types represent numbers with a decimal, such as 9.99 or 3.14515.

The Float and Double data types can store fractional numbers

Float Example

```
val myNum: Float = 5.75F
println(myNum)
```

Double Example

```
val myNum: Double = 19.99
println(myNum)
```

c. Booleans

The Boolean data type and can only take the values true or false:

Example

```
val isKotlinFun: Boolean = true
val isFishTasty: Boolean = false
println(isKotlinFun)   // Outputs true
println(isFishTasty)
```

d. Characters

The Char data type is used to store a single character. A char value must be surrounded by single quotes, like 'A' or 'c':

Example

```
val myGrade: Char = 'B'
println(myGrade)
```

e. Strings

The String data type is used to store a sequence of characters (text). String values must be surrounded by double quotes:

Example

```
val myText: String = "Hello World"
println(myText)
```

f. **Date and Time:**

Kotlin provides the 'Long' data type to store dates and time values in milliseconds since the Unix epoch (January 1, 1970, at 00:00:00 UTC). You can also use the 'java.util.Date' or 'java.time.Instant' classes for more advanced date and time handling.

Example using 'Long':

```
val currentTimeInMillis: Long = System.currentTimeMillis() // Declare and initialize a variable storing current time in milliseconds
```

1.3 Access modifier

The Kotlin visibility modifiers are the keywords that set the visibility of classes, objects, interface, constructors, functions as well as properties and their setters. Though getters always have the same visibility as their properties, so we cannot set their visibility.

a. Public Modifier

Public modifier is accessible from anywhere in the project workspace. If no access modifier is specified, then by default it will be in the public scope. In all our previous examples, we have not mentioned any modifier, hence, all of them are in the public scope. Following is an example to understand more on how to declare a public variable or method.

```
class publicExample {
  val i = 1
    fun doSomething() {
  }
}
```

In the above example, we have not mentioned any modifier, so the method and variable defined here, are by default public. Though above example can be written with public modifier explicitly as follows:

b. Private Modifier

The classes, methods, packages and other properties can be declared with a private modifier. This modifier has almost the exact opposite meaning of public which means a private member can not be accessed outside of its scope. Once anything is declared as private, then it can be accessible within its immediate scope only. For instance, a private package can be accessible within that specific file. A private class or interface can be accessible only by its data members, etc.

```
        private class privateExample {
  private val i = 1
    private val doSomething() {
  }
}
```

In the above example, the class privateExample is only accessible from within the same source file and the variable i and method doSomething can only be accessed from inside of class privateExample.

Example

Let's check a simple example showing the usage of private members:

```
open class A() {
  private val i = 1

  fun doSomething(){
    println("Inside doSomething" )
    println("Value of i is $i" )
  }
}
class B : A() {
  fun printValue(){
      doSomething()
      // println("Value of i is $i" )
  }
}

fun main(args: Array<String>) {
  val a = A()
  val b = B()

  b.printValue()
}
```

When you run the above Kotlin program, it will generate the following output:

```
Inside doSomething
Value of i is 1
```

Here we can not access variable i inside class B because it has been defined as private which means it can be accessed inside the class itself and nowhere else.

c. Internal Modifier

Internal is a newly added modifier in Kotlin. If anything is marked as internal, then the specific field will marked as the internal field. An Internal package is visible only inside the module under which it is implemented. An internal class interface is visible only by other class present inside the same package or the module. In the following example, we will see how to implement an internal method.

```
package one
internal class InternalExample {
}
class publicExample{
   internal val i = 1
   internal fun doSomething() {
   }
}
```

In the above example, class **InternalExample** is only accessible from inside the same module, similarly variable **i** and function **doSomething()** are also accessible from inside the same module only, even though the class **publicExample** can be accessed from anywhere because this class has **public** visibility by default.

Example

Let's check a simple example showing the usage of internal members:

```
package com.tutorialspoint.modifiers
open class A() {
  internal val i = 1
    internal fun doSomething(){
    println("Inside doSomething" )
    println("Value of i is $i" )
  }
}
class B : A() {
  fun printValue(){
     doSomething()
     println("Value of i is $i" )
  }
}
fun main(args: Array<String>) {
  val a = A()
  val b = B()
    a.doSomething()
    b.printValue()
}
```

When you run the above Kotlin program, it will generate the following output:

```
Inside doSomething
Value of i is 1
Inside doSomething
Value of i is 1
Value of i is 1
```

d. Protected Modifier

Protected is another access modifier for Kotlin, which is currently not available for top level declaration like any package cannot be protected. A protected class or interface or properties or function is visible to the class itself and it's subclasses only.

```
package one;
class A() {
  protected val i = 1
}
class B : A() {
  fun getValue() : Int {
    return i
  }
}
```

In the above example, the variable i is declared as protected, hence, it is only visible to class itself and it's subclasses.

Example

Let's check a simple example showing the usage of protected members:

```
open class A() {
  protected val i = 1
    protected fun doSomething(){
    println("Inside doSomething" )
    println("Value of i is $i" )
  }
}
class B : A() {
  fun printValue(){
     doSomething()
     println("Value of i is $i" )
  }
}
fun main(args: Array<String>) {
  val a = A()
  val b = B()
    //a.doSomething()
  b.printValue()
}
```

When you run the above Kotlin program, it will generate the following output:

```
Inside doSomething
Value of i is 1
Value of i is 1
```

Here we can not call doSomething() even using an object of class A because it has been defined as protected which means it can be accessed inside the class itself or in its subclasses only.

1.4 Input and output syntax

In Kotlin, the syntax for input and output is quite similar to other programming languages. Here's a basic example of how you can input and output data in Kotlin:

a. Input (Reading data from the user or a file)

To read data from the user (console input), you can use the `readLine()` function provided by Kotlin's `ReadLine` class. Here's an example:

```
import java.lang.Integer.parseInt
fun main() {
   println("Please enter an integer:")
   val inputString = readLine()
   val inputNumber = parseInt(inputString!!)
   println("You entered: $inputNumber")
}
```

In this example, the `readLine()` function reads a line of text from the console, and the `parseInt()` function converts the input string to an integer.

To read data from a file, you can use Kotlin's `File` and `BufferedReader` classes. Here's an example:

```
import java.io.File
import java.io.BufferedReader
import java.io.FileReader
fun main() {
   val file = File("data.txt")
   val reader = BufferedReader(FileReader(file))
   reader.use {
      reader.readText().also { println(it) }
   }
}
```

In this example, the `FileReader` class is used to create a `BufferedReader` that reads the contents of the "data.txt" file.

b. Output (Writing data to the console or a file)

To output data to the console, you can use the `println()` function. Here's an example:

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

In this example, the `println()` function outputs the string "Hello, World!" to the console. To write data to a file, you can use Kotlin's `File` and `PrintWriter` classes. Here's an example:

```
import java.io.File
import java.io.PrintWriter
fun main() {
   val file = File("output.txt")
   val writer = PrintWriter(file)
   writer.use {
      it.println("This is some text written to a file.")
   }
}
```

In this example, the `PrintWriter

1.5 Conditional statement

Kotlin if...else expressions works like an if...else expression in any other Modern Computer Programming. So let's start with our traditional if...else statement available in Kotlin.

Syntax

The syntax of a traditional if...else expression is as follows:

```
if (condition) {
  // code block A to be executed if condition is true
} else {
 // code block B to be executed if condition is false
}
```

Here if statement is executed and the given condition is checked. If this condition is evaluated to true then code block A is executed, otherwise program goes into else part and code block B is executed.

Example

```
fun main(args: Array<String>) {
    val age:Int = 10

    if (age > 18) {
        print("Adult")
    } else {
        print("Minor")
    }
}
```

Output: Minor

if...else Expression

Kotlin if...else can also be used as an expression which returns a value and this value can be assigned to a variable. Below is a simple syntax of Kotlin if...else expression:

Syntax

```
val result = if (condition) {
  // code block A to be executed if condition is true
} else {
 // code block B to be executed if condition is false
}
```

Examples

```
fun main(args: Array<String>) {
    val age:Int = 10

    val result = if (age > 18) {
        "Adult"
    } else {
        "Minor"
```

```
  }
  println(result)
}
```
Output: Minor

You can ommit the curly braces {} when if has only one statement:
```
fun main(args: Array<String>) {
   val age:Int = 10
   val result = if (age > 18) "Adult" else  "Minor"
   println(result)
}
```

Output: Minor
You can include multiple statements in if...else block, in this case the last expression is returned as the value of the block. Try the following example:
```
fun main(args: Array<String>) {
   val age:Int = 10
   val result = if (age > 18) {
      println("Given condition is true")
      "Adult"
   } else {
      println("Given condition is false")
      "Minor"
   }
   print("The value of result : ")
   println(result)
}
```
Output:
```
Given condition is false
The value of result : Minor
```

if...else...if Ladder
You can use else if condition to specify a new condition if the first condition is false.
Syntax

```
if (condition1) {
  // code block A to be executed if condition1 is true
} else if (condition2) {
  // code block B to be executed if condition2 is true
} else {
  // code block C to be executed if condition1 and condition2 are false
}
```

Example

```
fun main(args: Array<String>) {
   val age:Int = 13

   val result = if (age > 19) {
      "Adult"
   } else if ( age > 12 && age  < 20 ){
      "Teen"
   } else {
      "Minor"
   }
   print("The value of result : ")
   println(result)
}
```

Output: The value of result : Teen

Nested if Expression

Kotlin allows to put an if expression inside another if expression. This is called **nested if** expression

Syntax

```
if(condition1) {
  // code block A to be executed if condition1 is true
  if( (condition2) {
    // code block B to be executed if condition2 is true
  }else{
    // code block C to be executed if condition2 is fals
  }
} else {
 // code block D to be executed if condition1 is false
}
```

Example

```
fun main(args: Array<String>) {
   val age:Int = 20

   val result = if (age > 12) {
     if ( age > 12 && age  < 20 ){
        "Teen"
     }else{
        "Adult"
     }
```

```
    } else {
        "Minor"
    }
    print("The value of result : ")
    println(result)
}
```

Output: The value of result : Adult

1.6 Conditional loops

In Kotlin, conditional loops are used to iterate over a collection or perform an action based on a condition. The most common conditional loops in Kotlin are "for" loops and "while" loops.

a. For Loop

The for loop is used to iterate over a collection like a list, array, or range of numbers. Here's a simple example of a for loop in Kotlin:

```
for (i in 1..5) {
    println("Number: $i")
}
```

In this example, the loop will iterate from 1 to 5 and print the numbers.

b. While Loop:

The while loop is used when the number of iterations is not known beforehand. It continues to execute the loop body as long as the condition is true. Here's an example of a while loop in Kotlin:

```
var i = 1
while (i <= 5) {
    println("Number: $i")
    i++
}
```

In this example, the loop will continue to print the numbers until the value of 'i' becomes greater than 5.

c. Do-While Loop:

The do-while loop is similar to the while loop, but it executes the loop body at least once before checking the condition. Here's an example of a do-while loop in Kotlin:

```
var i = 1
```

```
do {
   println("Number: $i")
   i++
} while (i <= 5)
```

In this example, the loop will print the numbers from 1 to 5, just like the previous examples.

d. Using Conditional Statements Inside Loops:

You can also use conditional statements (if, else, or when) inside loops to control the flow of execution. For example:

```
for (i in 1..5) {
   if (i % 2 == 0) {
      println("Number: $i is even")
   } else {
      println("Number: $i is odd")
   }
}
```

In this example, the loop will print whether each number is even or odd based on the condition inside the if-else statement.

1.7 Data types in Kotlin

Kotlin, a statically-typed programming language, supports various data types. These data types can be categorized into three main groups: primitive, non-primitive, and user-defined types.

a. Primitive Data Types

Kotlin has five primitive data types, which are similar to those in Java:

a. Int: Represents signed 32-bit integers.
b. Long: Represents signed 64-bit integers.
c. Float: Represents 32-bit floating-point numbers.
d. Double: Represents 64-bit floating-point numbers.
e. Boolean: Represents true or false values.

b. Non-Primitive Data Types

These are object-based types, which are instances of classes or interfaces. Some common non-primitive data types in Kotlin include:

- String: Represents a sequence of characters.
- Char: Represents a single Unicode character.

- Byte: Represents an 8-bit signed integer.
- Short: Represents a 16-bit signed integer.
- Char Array: Represents an array of characters.
- Byte Array: Represents an array of bytes.
- Any: The root of the Kotlin class hierarchy.
- Nothing: Represents the absence of a value.

c. **User-Defined Data Types**
These are custom data types created by developers, such as classes, objects, interfaces, and Enums.

d. **Collections**
Kotlin also provides various collection data types, like List, Set, and Map, which are used to store multiple values. These collections are implemented as classes and can be nested.

e. **Nullable and Non-Nullable Types**
In Kotlin, every variable is non-nullable by default, meaning it cannot have a null value. To represent a variable that can hold a null value, you need to append a '?' to the type declaration. For example, `var x: Int?` declares a

Self-Check Sheet - 1: Create Kotlin data types

Questionnaire

1. What is the basic structure of a Kotlin program for an Android application?

   Answer:

2. How do I declare an integer variable in Kotlin?

   Answer:

3. How do I declare a string variable in Kotlin?

   Answer:

4. How do I declare a double variable in Kotlin?

   Answer:

5. What are the access modifiers in Kotlin?

   Answer:

6. How do I input user data in a Kotlin Android application?

   Answer:

7. How do I output data in a Kotlin Android application?

   Answer:

8. Explain a conditional statement in Kotlin.

   Answer:

9. What is a conditional loop in Kotlin?

   Answer:

10. Give an example of a non-primitive data type in Kotlin.

    Answer:

Answer Key - 1: Create Kotlin data types

1. What is the basic structure of a Kotlin program for an Android application?
   Answer: A Kotlin program for an Android application typically consists of activities, fragments, and other components. It starts with the main function, which serves as the entry point, and includes layout files, resource files, and other necessary components to build the user interface and handle user interactions.

2. How do I declare an integer variable in Kotlin?
   Answer: You can declare an integer variable in Kotlin using the "var" or "val" keyword followed by the variable name and the data type, like this: `var myInteger: Int`.

3. How do I declare a string variable in Kotlin?
   Answer: To declare a string variable in Kotlin, use the "var" or "val" keyword, followed by the variable name and the data type, like this: `var myString: String`.

4. How do I declare a double variable in Kotlin?
   Answer: To declare a double variable in Kotlin, use the "var" or "val" keyword, followed by the variable name and the data type, like this: `var myDouble: Double`.

5. What are the access modifiers in Kotlin?
   Answer: The access modifiers in Kotlin are Public, Private, and Protected. Public allows access from any other class, Private restricts access to the same class, and Protected allows access within the same class or its subclasses.

6. How do I input user data in a Kotlin Android application?
   Answer: You can input user data in a Kotlin Android application using various UI components like EditText, TextField, or custom dialogs. You can then retrieve the inputted data using their respective methods or properties.

7. How do I output data in a Kotlin Android application?
   Answer: You can output data in a Kotlin Android application by updating the UI components like TextViews, TextFields, or custom views with the desired data. You can also use Toast messages or Log statements for simple output.

8. Explain a conditional statement in Kotlin.
   Answer: A conditional statement in Kotlin is used to execute different code blocks based on a condition. It includes if, else, and when statements. For example:

```
if (condition) {
   // Code block to execute if the condition is true
} else {
   // Code block to execute if the condition is false
}
```

9. What is a conditional loop in Kotlin?
   Answer: A conditional loop in Kotlin is used to iterate over a collection or perform an action based on a condition. Examples include for loops, while loops, and do-while loops. For example:

```
for (i in 1..5) {
   // Code block to execute for each iteration
}
```

Job Sheet-1.1: Declare and initialize variables of different types (e.g., integer, string, Boolean) in Kotlin. Print their values to the console.

Working Procedure/ Steps:

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Create a folder on the desktop with your registration number and name.
5. Collect the resources from your trainer as per the job requirement.
6. Open IntelliJ IDEA or android studio
7. Set Up Your IntelliJ IDEA Environment
8. Write Kotlin program use different types of variables
9. Ensure you have a Kotlin compiler or an IDE like IntelliJ IDEA that supports Kotlin.
10. Declare and Initialize Variables:
11. In Kotlin, you can use var for mutable (changeable) variables and val for immutable (read-only) variables.
12. Run the Kotlin program
13. Print the Values to the Console:
14. Finish your work as per specification sheet
15. Show your work to the trainer.
16. Close IntelliJ IDEA, Android Studio as per standard procedure.
17. Clean your Work Place as per standard procedure.
18. Turn off the computer and clean your workplace.

Specification Sheet-1.1: Declare and initialize variables of different types (e.g., integer, string, Boolean) in Kotlin. Print their values to the console.

Conditions for the job: Work must be carried out in a safe manner and according to relevant competency standards.

List of required PPE's

S/N	Name of PPE	Specification	Unit	Required Quantity
01	Ergonomic Chair	Wood and foam	No	1
02	Eye protective glass	Metal and Glass	No	1
03	Mask	Surgical mask	1	1

List of required Software

S/N	Name of Materials	Specification	Unit	Required Quantity
01	IntelliJ IDEA	Latest version	…	1
02	Android Studio	Latest version	…	1

List of required Tools & Equipment's

S/N	Name	Specification	Unit	Required Quantity
01	Personal Computer or Laptop	Minimum Core i5 7th gen Processor	No	1
04	Internet connection		…	1

Job Sheet-1.2: Write a Kotlin program that takes user input for their name and age. Then, output a personalized message based on the input.

Working Procedure/ Steps:

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Create a folder on the desktop with your registration number and name.
5. Collect the resources from your trainer as per the job requirement.
6. Open IntelliJ IDEA or android studio
7. Set Up Your IntelliJ IDEA Environment
8. Write Kotlin program user input for their name and age
9. Ensure you have a Kotlin compiler or an IDE like IntelliJ IDEA that supports Kotlin.
10. Create a Kotlin file: Create a new Kotlin file named "PersonalizedMessage.kt".
11. Read user input:
12. Use readLine() to get user input.
13. Convert the input to the appropriate type (String for name, Int for age).
14. Generate a personalized message: Combine the name and age into a message.
15. Output the message: Print the message to the console.
16. Print the Values to the Console:
17. Show your work to the trainer.
18. Close IntelliJ IDEA, Android Studio as per standard procedure.
19. Clean your Work Place as per standard procedure.
20. Turn off the computer and clean your workplace.

Specification Sheet-1.2: Write a Kotlin program that takes user input for their name and age. Then, output a personalized message based on the input.

Conditions for the job: Work must be carried out in a safe manner and according to relevant competency standards.

List of required PPE's

S/N	Name of PPE	Specification	Unit	Required Quantity
01	Ergonomic Chair	Wood and foam	No	1
02	Eye protective glass	Metal and Glass	No	1
03	Mask	Surgical mask	1	1

List of required Software

S/N	Name of Materials	Specification	Unit	Required Quantity
01	IntelliJ IDEA	Latest version	…	1
02	Android Studio	Latest version	…	1

List of required Tools & Equipment's

S/N	Name	Specification	Unit	Required Quantity
01	Personal Computer or Laptop	Minimum Core i5 7th gen Processor	No	1
04	Internet connection		…	1

Job Sheet-1.3: Implement a Kotlin program that checks if a given number is positive, negative, or zero using an if-else statement.

Working Procedure/ Steps:

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Create a folder on the desktop with your registration number and name.
5. Collect the resources from your trainer as per the job requirement.
6. Open IntelliJ IDEA or android studio
7. Set Up Your IntelliJ IDEA Environment
8. Ensure you have a Kotlin compiler or an IDE like IntelliJ IDEA that supports Kotlin.
9. Write a Kotlin program given number is positive, negative, or zero using an if-else statement
10. Declare a variable num to store the input number.
11. Use an if-else statement to check the value of num.
12. Check if num is greater than 0. If true, print "Positive".
13. Check if num is less than 0. If true, print "Negative".
14. If neither condition is true (i.e., num is equal to 0), print "Zero".
15. Print the Values to the Console:
16. Show your work to the trainer.
17. Close IntelliJ IDEA, Android Studio as per standard procedure.
18. Clean your Work Place as per standard procedure.
19. Turn off the computer and clean your workplace.

Specification Sheet-1.3: Implement a Kotlin program that checks if a given number is positive, negative, or zero using an if-else statement.

Conditions for the job: Work must be carried out in a safe manner and according to relevant competency standards.

List of required PPE's

S/N	Name of PPE	Specification	Unit	Required Quantity
01	Ergonomic Chair	Wood and foam	No	1
02	Eye protective glass	Metal and Glass	No	1
03	Mask	Surgical mask	1	1

List of required Software

S/N	Name of Materials	Specification	Unit	Required Quantity
01	IntelliJ IDEA	Latest version	…	1
02	Android Studio	Latest version	…	1

List of required Tools & Equipment's

S/N	Name	Specification	Unit	Required Quantity
01	Personal Computer or Laptop	Minimum Core i5 7th gen Processor	No	1
04	Internet connection		…	1

Job Sheet-1.4: Write a Kotlin program that uses a while loop to print the first 5 even numbers (starting from 2)

Working Procedure/ Steps:

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Create a folder on the desktop with your registration number and name.
5. Collect the resources from your trainer as per the job requirement.
6. Open IntelliJ IDEA or android studio
7. Set Up Your IntelliJ IDEA Environment
8. Ensure you have a Kotlin compiler or an IDE like IntelliJ IDEA that supports Kotlin.
9. Write a Kotlin program uses a while loop to print the first 5 even numbers statement
10. Declare a variable i to store the starting number (2).
11. Initialize a counter count to 0 to keep track of the number of even numbers printed.
12. Use a while loop to iterate until count reaches 5.
13. Inside the loop, check if i is even (using the modulus operator %).
14. If i is even, print the number and increment count.
15. Increment i by 1.
16. Print the Values to the Console:
17. Show your work to the trainer.
18. Close IntelliJ IDEA, Android Studio as per standard procedure.
19. Clean your Work Place as per standard procedure.
20. Turn off the computer and clean your workplace.

Specification Sheet-1.4: Write a Kotlin program that uses a while loop to print the first 5 even numbers (starting from 2)

Conditions for the job: Work must be carried out in a safe manner and according to relevant competency standards.

List of required PPE's

S/N	Name of PPE	Specification	Unit	Required Quantity
01	Ergonomic Chair	Wood and foam	No	1
02	Eye protective glass	Metal and Glass	No	1
03	Mask	Surgical mask	1	1

List of required Software

S/N	Name of Materials	Specification	Unit	Required Quantity
01	IntelliJ IDEA	Latest version	…	1
02	Android Studio	Latest version	…	1

List of required Tools & Equipment's

S/N	Name	Specification	Unit	Required Quantity
01	Personal Computer or Laptop	Minimum Core i5 7th gen Processor	No	1
04	Internet connection		…	1

Job Sheet-1.5: Define a data class in Kotlin representing a "Person" with attributes such as name, age, and email. Create an instance of this class and print its details.

Working Procedure/ Steps:

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Create a folder on the desktop with your registration number and name.
5. Collect the resources from your trainer as per the job requirement.
6. Open IntelliJ IDEA or android studio
7. Set Up Your IntelliJ IDEA Environment
8. Ensure you have a Kotlin compiler or an IDE like IntelliJ IDEA that supports Kotlin.
9. Write a Kotlin program Create an instance of this class and print its details
10. Define a data class named Person with attributes name, age, and email.
11. Create an instance of the Person class using the constructor.
12. Assign values to the attributes of the instance.
13. Print the details of the instance using property accessors.
14. Print the Values to the Console:
15. Show your work to the trainer.
16. Close IntelliJ IDEA, Android Studio as per standard procedure.
17. Clean your Work Place as per standard procedure.
18. Turn off the computer and clean your workplace.

Specification Sheet-1.5: Define a data class in Kotlin representing a "Person" with attributes such as name, age, and email. Create an instance of this class and print its details

Conditions for the job: Work must be carried out in a safe manner and according to relevant competency standards.

List of required PPE's

S/N	Name of PPE	Specification	Unit	Required Quantity
01	Ergonomic Chair	Wood and foam	No	1
02	Eye protective glass	Metal and Glass	No	1
03	Mask	Surgical mask	1	1

List of required Materials

S/N	Name of Materials	Specification	Unit	Required Quantity
01	IntelliJ IDEA	Latest version	…	1
02	Android Studio	Latest version	…	1

List of required Tools & Equipment's

S/N	Name	Specification	Unit	Required Quantity
01	Personal Computer or Laptop	Minimum Core i5 7th gen Processor	No	1
04	Internet connection		…	1

Learning Outcome 2: Use Kotlin

Assessment Criteria	1. An application is created. 2. The numbers are seen in display. 3. Methods in Kotlin is implemented.
Conditions and Resources	• Actual workplace or training environment • CBLM • Handouts • Job related tools, equipment, and materials • Multimedia Projector • Paper, Pen, Pencil, and Eraser • Internet Facilities • Whiteboard and Marker Whiteboard and Marker
Contents	1. Kotlin application 2. Display number ▪ List of number ▪ Even number ▪ odd number 3. Implementation method of Kotlin 4. Conditional statement 5. Conditional loops
Activities/job/Task	1. Create a Kotlin console application that greets the user. Prompt the user to enter their name and then print a personalized greeting. 2. Write a Kotlin program that displays numbers from 1 to 10 in the console. Use a loop to iterate through the numbers and print each one. 3. Define a Kotlin class named Calculator with methods for basic arithmetic operations (addition, subtraction, multiplication, division). Create an instance of the class and demonstrate the use of these methods.
Training Methods	• Blended • Discussion • Presentation • Demonstration • Guided Practice • Individual Practice • Project Work • Problem Solving • Brainstorming

Assessment Methods	Assessment methods may include but not limited to • Written Test • Demonstration • Oral Questioning

Learning Experience 2: Use Kotlin

In order to achieve the objectives stated in this learning guide, you must perform the learning steps below. Beside each step are the resources or special instructions you will use to accomplish the corresponding activity.

Learning Activities	Recourses/Special Instructions
1. Trainee will ask the instructor about the learning materials	1. Instructor will provide the learning materials Use Kotlin
2. Read the Information sheet and complete the Self Checks & Check answer sheets on "Apply encapsulation"	2. Read Information sheet 2: Use Kotlin 3. Answer Self-check 2: Use Kotlin 4. Check your answer with Answer key 2: Use Kotlin
3. Read the Job/Task Sheet and Specification Sheet and perform job/Task	4. Job/Task Sheet and Specification Sheet **Job Sheet 2.1:** Create a Kotlin console application that greets the user. Prompt the user to enter their name and then print a personalized greeting. **Specification Sheet 2.1:** Create a Kotlin console application that greets the user. Prompt the user to enter their name and then print a personalized greeting **Job Sheet 2.2:** Write a Kotlin program that displays numbers from 1 to 10 in the console. Use a loop to iterate through the numbers and print each one. **Specification Sheet 2.1:** Write a Kotlin program that displays numbers from 1 to 10 in the console. Use a loop to iterate through the numbers and print each one. **Job Sheet 2.3:** Define a Kotlin class named Calculator with methods for basic arithmetic operations (addition, subtraction, multiplication, division). Create an instance of the class and demonstrate the use of these methods. **Specification Sheet 2.3:** Define a Kotlin class named Calculator with methods for basic arithmetic operations (addition, subtraction, multiplication, division). Create an instance of the class and demonstrate the use of these methods.

Information Sheet 2: Use Kotlin

Learning Objective: After completion of this information sheet, the learners will be able to explain, define and interpret the following contents:

2.1 Kotlin application
2.2 Display number
2.3 Implementation method of Kotlin
2.4 Conditional statement
2.5 Conditional loops

2.1 Kotlin application

A Kotlin application refers to a software program or mobile app developed using Kotlin, a modern programming language created by JetBrains. Kotlin is known for its concise and expressive syntax, which makes it easier to write and maintain code. It is an ideal choice for developing Android applications, as Kotlin has been officially supported by Google since 2017.

Sure! Here's a simple Kotlin application that demonstrates basic console input and output:

```
import java.util.Scanner
fun main() {
    val scanner = Scanner(System.`in`)
    // Prompt the user to enter their name
    print("Enter your name: ")
    val name = scanner.nextLine()
    // Greet the user
    println("Hello, $name! Welcome to our Kotlin application.")
    // Prompt the user to enter their age
    print("Enter your age: ")
    val age = scanner.nextInt()
    // Display a message based on the user's age
    val message = if (age < 18) {
        "You are under 18 years old."
    } else {
        "You are $age years old."
    }
    println(message)
    // Close the scanner
    scanner.close()
}
```

This application prompts the user to enter their name and age, then displays a greeting message along with information about their age. Here's how it works:

- We import `Scanner` from `java.util` to handle user input.
- In the `main` function, we create a `Scanner` object to read input from the console.
- We prompt the user to enter their name and age using `print` statements.
- We use `scanner.nextLine()` to read the entire line of text entered by the user as their name.
- We use `scanner.nextInt()` to read the next integer entered by the user as their age.
- Depending on the age entered, we generate a message and display it to the user.
- Finally, we close the scanner to release system resources.

2.2 Display number

To display a number in Kotlin, you can create a simple Kotlin application that prints the desired number to the console or display it on a user interface. Here's an example of a Kotlin script that prints a number to the console:

```
fun main() {
   val numberToDisplay = 42
   println("The number is: $numberToDisplay")
}
```

In this example, we declare a variable `numberToDisplay` and assign it the value 42. Then, we use the `println()` function to print the number to the console along with a descriptive message.

a. List of number

In Kotlin, you can create a list of numbers using the `List` data structure. Here's an example of creating a list containing integers:

```
val numbersList: List<Int> = listOf(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

In this code snippet, we declare a variable `numbersList` of type `List<Int>`, which means it's a list of integers. We initialize it with the `listOf()` function, passing in the numbers we want to include in the list.
You can also create an empty list and add elements to it later:

```
val numbersList: MutableList<Int> = mutableListOf<Int>()
// Add elements to the list
numbersList.add(1)
numbersList.add(2)
numbersList.add(3)
// ... continue adding more numbers
```

In this example, we declare `numbersList` as a mutable list, which allows us to modify its contents. We initialize it using the `mutableListOf()` function, and then we can add elements to the list using the `add()` function.
You can access individual elements in the list using the index, like this:

```
println("The third number is: ${numbersList[2]}")
```

This code snippet prints the third number in the list, which in this case is 3. Remember that list indices start at 0, so the first element is at index 0.

b. Even number

To find even numbers in Kotlin, you can create a function that checks if a given number is even and then apply that function to a list of numbers. Here's an example:

```
fun isEven(number: Int): Boolean {
   return number % 2 == 0
}
val numbersList: List<Int> = listOf(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
val evenNumbers = numbersList.filter { isEven(it) }
println("The even numbers in the list are: $evenNumbers")
```

In this code, we define a function `isEven()` that takes an integer as input and returns `true` if the number is even (i.e., divisible by 2), and `false` otherwise.

We then create a list of numbers called `numbersList` containing the numbers 1 to 10. Using the `filter()` function, we apply the `isEven()` function to each element of the list. The `filter()` function returns a new list containing only the elements for which the given function returns `true`.

Finally, we print the even numbers in the list using string interpolation. In this case, the output will be: "The even numbers in the list are: [2, 4, 6, 8, 10]".

c. Odd number

To find odd numbers in Kotlin, you can create a function that checks if a given number is odd and then apply that function to a list of numbers. Here's an example:

```
fun isOdd(number: Int): Boolean {
   return number % 2 != 0
}
val numbersList: List<Int> = listOf(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
val oddNumbers = numbersList.filter { isOdd(it) }
println("The odd numbers in the list are: $oddNumbers")
```

In this code, we define a function `isOdd()` that takes an integer as input and returns `true` if the number is odd (i.e., not divisible by 2), and `false` otherwise.

We then create a list of numbers called `numbersList` containing the numbers 1 to 10. Using the `filter()` function, we apply the `isOdd()` function to each element of the list. The `filter()` function returns a new list containing only the elements for which the given function returns `true`.

Finally, we print the odd numbers in the list using string interpolation. In this case, the output will be: "The odd numbers in the list are: [1, 3, 5, 7, 9]".

2.3 Implementation method of Kotlin

Interfaces in Kotlin can contain declarations of abstract methods, as well as method implementations. What makes them different from abstract classes is that interfaces cannot store a state. They can have properties, but these need to be abstract or provide accessor implementations.

a. Basic Interface

A Kotlin interface contains declarations of abstract methods, and default method implementations although they cannot store state.

```
interface MyInterface {
   fun bar()
}
```

This interface can now be implemented by a class as follows:

```
class Child : MyInterface {
   override fun bar() {
      print("bar() was called")
   }
}
```

b. Interface with Default Implementations

An interface in Kotlin can have default implementations for functions:

```
interface MyInterface {
   fun withImplementation() {
      print("withImplementation() was called")
   }
}
```

Classes implementing such interfaces will be able to use those functions without reimplementing

```
class MyClass: MyInterface {
   // No need to reimplement here
}
   val instance = MyClass()
   instance.withImplementation()
```

c. Properties

Default implementations also work for property getters and setters:

```
interface MyInterface2 {
 val helloWorld
 get() = "Hello World!"
}
```

Interface accessors implementations can't use backing fields

```
interface MyInterface3 {
   // this property won't compile!
   var helloWorld: Int
   get() = field
   set(value) { field = value }
}
```

d. Multiple Implementations

When multiple interfaces implement the same function, or all of them define with one or more implementing, the derived class needs to manually resolve proper call

```
interface A {
  fun notImplemented()
  fun implementedOnlyInA() { print("only A") }
  fun implementedInBoth() { print("both, A") }
  fun implementedInOne() { print("implemented in A") }
}

interface B {
  fun implementedInBoth() { print("both, B") }
  // only defined
  fun implementedInOne()
}
```

```
class MyClass: A, B {
  override fun notImplemented() { print("Normal implementation") }
  // implementedOnlyInA() can by normally used in instances
  // class needs to define how to use interface functions
  override fun implementedInBoth() {
  super<B>.implementedInBoth()
  super<A>.implementedInBoth()
 }

// even if there's only one implementation,
// there multiple definitions
override fun implementedInOne() {
  super<A>.implementedInOne()
  print("implementedInOne class implementation")
 }
        }
```

e. Properties in Interfaces

You can declare properties in interfaces. Since an interface cannot have stated you can only declare a property as abstract or by providing default implementation for the accessors.

```
interface MyInterface {
 // abstract
 val property: Int
 val propertyWithImplementation: String
    get() = "foo"
 fun foo() {
   print(property)
 }
}

class Child : MyInterface {
 override val property: Int = 29
}
```

Conflicts when Implementing Multiple Interfaces with Default Implementations

When implementing more than one interface that has methods of the same name that include default implementations, it is ambiguous to the compiler which implementation should be used. In the case of a conflict, the developer must override the conflicting method and provide a custom implementation. That implementation may choose to delegate to the default implementations or not.

```
interface FirstTrait {
 fun foo() { print("first") }
 fun bar()
}

interface SecondTrait {
 fun foo() { print("second") }
 fun bar() { print("bar") }
}

class ClassWithConflict : FirstTrait, SecondTrait {
 override fun foo() {
    // delegate to the default
    // implementation of FirstTrait
    super<FirstTrait>.foo()
    // delegate to the default
    // implementation of SecondTrait
    super<SecondTrait>.foo()
 }
 // function bar() only has a default implementation
 // in one interface and therefore is ok.
}
```

f. super keyword

```
interface MyInterface {
 fun funcOne() {
    // optional body
    print("Function with default implementation")
 }
}
```

2.4 Conditional statement

Kotlin if...else expressions work like an if...else expression in any other Modern Computer Programming. So, let us start with our traditional if...else statement available in Kotlin.

Syntax

The syntax of a traditional if...else expression is as follows:

```
if (condition) {
  // code block A to be executed if condition is true
} else {
  // code block B to be executed if condition is false
}
```

Here if statement is executed and the given condition is checked. If this condition is evaluated to true then code block A is executed, otherwise program goes into else part and code block B is executed.

Example

```
fun main(args: Array<String>) {
   val age:Int = 10

   if (age > 18) {
      print("Adult")
   } else {
      print("Minor")
   }
}
```

Output: Minor

if...else Expression

Kotlin if...else can also be used as an expression which returns a value and this value can be assigned to a variable. Below is a simple syntax of Kotlin if...else expression:

Syntax

```
val result = if (condition) {
  // code block A to be executed if condition is true
} else {
  // code block B to be executed if condition is false
}
```

Examples

```
fun main(args: Array<String>) {
   val age:Int = 10
   val result = if (age > 18) {
      "Adult"
   } else {
      "Minor"
   }
   println(result)
}
```

Output: Minor

You can ommit the curly braces {} when if has only one statement:

```
fun main(args: Array<String>) {
   val age:Int = 10
   val result = if (age > 18) "Adult" else  "Minor"
   println(result)
}
```

Output: Minor

You can include multiple statements in if...else block, in this case the last expression is returned as the value of the block. Try the following example:

```
fun main(args: Array<String>) {
   val age:Int = 10
   val result = if (age > 18) {
      println("Given condition is true")
      "Adult"
   } else {
      println("Given condition is false")
      "Minor"
   }
   print("The value of result : ")
   println(result)
}
```

Output:

```
Given condition is false
The value of result : Minor
```

if...else...if Ladder

You can use else if condition to specify a new condition if the first condition is false.

Syntax

```
if (condition1) {
  // code block A to be executed if condition1 is true
} else if (condition2) {
  // code block B to be executed if condition2 is true
} else {
  // code block C to be executed if condition1 and condition2 are false
}
```

Example

```
fun main(args: Array<String>) {
    val age:Int = 13

    val result = if (age > 19) {
        "Adult"
    } else if ( age > 12 && age  < 20 ){
        "Teen"
    } else {
        "Minor"
    }
    print("The value of result : ")
    println(result)
}
```

Output: The value of result : Teen

Nested if Expression

Kotlin allows to put an if expression inside another if expression. This is called **nested if** expression

Syntax

```
if(condition1) {
   // code block A to be executed if condition1 is true
   if( (condition2) {
      // code block B to be executed if condition2 is true
   }else{
      // code block C to be executed if condition2 is fals
   }
} else {
  // code block D to be executed if condition1 is false
}
```

Example

```
fun main(args: Array<String>) {
   val age:Int = 20

   val result = if (age > 12) {
     if ( age > 12 && age  < 20 ){
        "Teen"
     }else{
        "Adult"
     }
   } else {
      "Minor"
   }
   print("The value of result : ")
   println(result)
}
```

Output: The value of result : Adult

2.5 Conditional loops

In Kotlin, conditional loops are used to iterate over a collection or perform an action based on a condition. The most common conditional loops in Kotlin are "for" loops and "while" loops.

a. For Loop

The for loop is used to iterate over a collection like a list, array, or range of numbers. Here's a simple example of a for loop in Kotlin:

```
for (i in 1..5) {
   println("Number: $i")
}
```

In this example, the loop will iterate from 1 to 5 and print the numbers.

b. While Loop:

The while loop is used when the number of iterations is not known beforehand. It continues to execute the loop body if the condition is true. Here is an example of a while loop in Kotlin:

```
var i = 1
while (i <= 5) {
   println("Number: $i")
   i++
}
```

In this example, the loop will continue to print the numbers until the value of 'i' becomes greater than 5.

c. **Do-While Loop:**
The do-while loop is like the while loop, but it executes the loop body at least once before checking the condition. Here is an example of a do-while loop in Kotlin:

```
var i = 1
do {
   println("Number: $i")
   i++
} while (i <= 5)
```

In this example, the loop will print the numbers from 1 to 5, just like the previous examples.

d. **Using Conditional Statements Inside Loops:**
You can also use conditional statements (if, else, or when) inside loops to control the flow of execution. For example,

```
for (i in 1..5) {
   if (i % 2 == 0) {
      println("Number: $i is even")
   } else {
      println("Number: $i is odd")
   }
}
```

In this example, the loop will print whether each number is even or odd based on the condition inside the if-else statement.

Self-Check - 2: Use Kotlin

Questionnaire

1. What is Kotlin?

   Answer

2. How do you create a Kotlin application?

   Answer:

3. How do you display a number in Kotlin?

   Answer:

4. How do you create a list of numbers in Kotlin?

   Answer:

5. How do you check if a number is even in Kotlin?

   Answer:

6. How do you identify even numbers in a list of numbers in Kotlin?

   Answer:

7. How do you implement methods in Kotlin?

   Answer:

8. What are conditional statements in Kotlin?

   Answer:

9. What are conditional loops in Kotlin?

   Answer:

10. How do you write a loop to display both even and odd numbers in Kotlin?

    Answer:

Answer Key - 2: Use Kotlin

1. What is Kotlin?

   Answer: Kotlin is a statically-typed programming language developed by JetBrains. It is designed to be fully interoperable with Java and is often used for developing Android applications.

2. How do you create a Kotlin application?

   Answer: You can create a Kotlin application by writing Kotlin code in a `.kt` file and then compiling and running it using a Kotlin compiler or an integrated development environment (IDE) like IntelliJ IDEA or Android Studio.

3. How do you display a number in Kotlin?

   Answer: You can display a number in Kotlin using the `println()` function or by interpolating the number into a string using the `$` symbol.

4. How do you create a list of numbers in Kotlin?

   Answer: You can create a list of numbers in Kotlin using the `listOf()` function, specifying the numbers inside the parentheses.

5. How do you check if a number is even in Kotlin?

   Answer: You can check if a number is even in Kotlin by using the modulo operator `%`. If the number modulo 2 equals 0, then the number is even.

6. How do you identify even numbers in a list of numbers in Kotlin?

   Answer: You can iterate over the list of numbers and use the `isEven()` function to check if each number is even. Add the even numbers to a new list for further processing.

7. How do you implement methods in Kotlin?

   Answer: In Kotlin, you can define functions using the `fun` keyword followed by the function name, parameters, and return type (if any).

8. What are conditional statements in Kotlin?

   Answer: Conditional statements in Kotlin, like in many programming languages, are used to execute different blocks of code based on certain conditions. Kotlin supports `if`, `if-else`, and `when` expressions for conditional branching.

9. What are conditional loops in Kotlin?

   Answer: Conditional loops, also known as iterative statements, are used to repeatedly execute a block of code until a specified condition is met. Kotlin supports `while` and `do-while` loops for conditional iteration.

10. How do you write a loop to display both even and odd numbers in Kotlin?

Answer: You can use a loop to iterate through a range of numbers and check if each number is even or odd using conditional statements. Then, display the numbers accordingly.

Job Sheet-2.1 Create a Kotlin console application that greets the user. Prompt the user to enter their name and then print a personalized greeting.

Working Procedure/ Steps:

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Create a folder on the desktop with your registration number and name.
5. Collect the resources from your trainer as per the job requirement.
6. Open IntelliJ IDEA or android studio
7. Set Up Your IntelliJ IDEA Environment
8. Ensure you have a Kotlin compiler or an IDE like IntelliJ IDEA that supports Kotlin.
9. Write a Kotlin program enter their name and then print a personalized greeting
10. Define a variable name to store the user's input.
11. Use readLine() function to read a line of input from the console and store it in the name variable.
12. Use a string template to create a personalized greeting message by concatenating the name variable with a predefined greeting phrase.
13. Print the greeting message to the console using println().
14. Print the Values to the Console:
15. Show your work to the trainer.
16. Close IntelliJ IDEA, Android Studio as per standard procedure.
17. Clean your Work Place as per standard procedure.
18. Turn off the computer and clean your workplace.

Specification Sheet-2.1: Create a Kotlin console application that greets the user. Prompt the user to enter their name and then print a personalized greeting.

Conditions for the job: Work must be carried out in a safe manner and according to relevant competency standards.

List of required PPE's

S/N	Name of PPE	Specification	Unit	Required Quantity
01	Ergonomic Chair	Wood and foam	No	1
02	Eye protective glass	Metal and Glass	No	1
03	Mask	Surgical mask	1	1

List of required Software

S/N	Name of Materials	Specification	Unit	Required Quantity
01	IntelliJ IDEA	Latest version	…	1
02	Android Studio	Latest version	…	1

List of required Tools & Equipment's

S/N	Name	Specification	Unit	Required Quantity
01	Personal Computer or Laptop	Minimum Core i5 7th gen Processor	No	1
04	Internet connection		…	1

Job Sheet-2.2: Write a Kotlin program that displays numbers from 1 to 10 in the console. Use a loop to iterate through the numbers and print each one.

Working Procedure/ Steps:

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Create a folder on the desktop with your registration number and name.
5. Collect the resources from your trainer as per the job requirement.
6. Open IntelliJ IDEA or android studio
7. Set Up Your IntelliJ IDEA Environment
8. Ensure you have a Kotlin compiler or an IDE like IntelliJ IDEA that supports Kotlin.
9. Write a Kotlin program Use a loop to iterate through the numbers and print each one
10. Define a variable i to store the iteration counter.
11. Use a for loop to iterate from 1 to 10 (inclusive).
12. Inside the loop, print the current value of i using println().
13. The loop will run 10 times, printing numbers 1 to 10.Print the Values to the Console:
14. Show your work to the trainer.
15. Close IntelliJ IDEA, Android Studio as per standard procedure.
16. Clean your Work Place as per standard procedure.
17. Turn off the computer and clean your workplace.

Specification Sheet-2.2: Write a Kotlin program that displays numbers from 1 to 10 in the console. Use a loop to iterate through the numbers and print each one.

Conditions for the job: Work must be carried out in a safe manner and according to relevant competency standards.

List of required PPE's

S/N	Name of PPE	Specification	Unit	Required Quantity
01	Ergonomic Chair	Wood and foam	No	1
02	Eye protective glass	Metal and Glass	No	1
03	Mask	Surgical mask	1	1

List of required Software

S/N	Name of Materials	Specification	Unit	Required Quantity
01	IntelliJ IDEA	Latest version	…	1
02	Android Studio	Latest version	…	1

List of required Tools & Equipment's

S/N	Name	Specification	Unit	Required Quantity
01	Personal Computer or Laptop	Minimum Core i5 7th gen Processor	No	1
04	Internet connection		…	1

Job Sheet-2.3: Define a Kotlin class named Calculator with methods for basic arithmetic operations (addition, subtraction, multiplication, division). Create an instance of the class and demonstrate the use of these methods.

Working Procedure/ Steps:

1. Follow OSH and use Personal Protective Equipment (PPE).
2. Check Electricity & Internet Connections to your Computer.
3. Start the Computer.
4. Create a folder on the desktop with your registration number and name.
5. Collect the resources from your trainer as per the job requirement.
6. Open IntelliJ IDEA or android studio
7. Set Up Your IntelliJ IDEA Environment
8. Ensure you have a Kotlin compiler or an IDE like IntelliJ IDEA that supports Kotlin.
9. Write a Kotlin program Create an instance of the class and demonstrate the use of these methods
10. Create the class: Define a class named Calculator.
11. Add methods: Implement methods for addition, subtraction, multiplication, and division
12. Create an instance: Instantiate the Calculator class.
13. Call the methods: Use the instance to call the methods and perform operations.
14. Show your work to the trainer.
15. Close IntelliJ IDEA, Android Studio as per standard procedure.
16. Clean your Work Place as per standard procedure.
17. Turn off the computer and clean your workplace.

Specification Sheet-2.3: Define a Kotlin class named Calculator with methods for basic arithmetic operations (addition, subtraction, multiplication, division). Create an instance of the class and demonstrate the use of these methods.

Conditions for the job: Work must be carried out in a safe manner and according to relevant competency standards.

List of required PPE's

S/N	Name of PPE	Specification	Unit	Required Quantity
01	Ergonomic Chair	Wood and foam	No	1
02	Eye protective glass	Metal and Glass	No	1
03	Mask	Surgical mask	1	1

List of required Software

S/N	Name of Materials	Specification	Unit	Required Quantity
01	IntelliJ IDEA	Latest version	…	1
02	Android Studio	Latest version	…	1

List of required Tools & Equipment's

S/N	Name	Specification	Unit	Required Quantity
01	Personal Computer or Laptop	Minimum Core i5 7th gen Processor	No	1
04	Internet connection		…	1

Review of Competency

Below is yourself assessment rating for module “Working with Kotlin Basics

Assessment of performance Criteria	Yes	No
Basic anatomy of a Kotlin program for android application is described.		
Kotlin variables are presented.		
Input and output syntax are applied		
Conditional statement is explained.		
Conditional loops are used.		
Data types in Kotlin is created.		
An application is created.		
The numbers are seen in display.		
Methods in Kotlin is implemented.		

I now feel ready to undertake my formal competency assessment.

Signed:

Date:

Development of CBLM

The Competency based Learning Material (CBLM) of '**Working with Kotlin Basics' (Occupation: Android Mobile Application Development, Level-4)** for National Skills Certificate is developed by NSDA with the assistance of SIMEC System Ltd., ECF Consultancy & SIMEC Institute of Technology JV (Joint Venture Firm) in the month of July, 2024 under the contract number of package SD-9B dated 15th January 2024.

SL No.	Name & Address	Designation	Contact Number
1	Engr. Md. Zuwel Parves	Writer	01737-278906
2	Md. Abdul Al Hossain	Editor	01778-926438
3	Engr Md. Zuwel Parves	Co-Ordinator	01737-278906
4	Md. Abdur Razzaque	Reviewer	01713-304824

Reference

1. https://reflectoring.io/kotlin-design-patterns/
2. https://www.w3schools.com/kotlin/index.php
3. https://www.tutorialspoint.com/kotlin/index.htm
4. https://kotlinlang.org/docs/control-flow.html#for-loops
5. https://www.programiz.com/kotlin-programming/examples/factors-number