



## **Kotlin Programming**

# **"Mastering Kotlin: A Comprehensive Guide for Android Developers"**

### **BOOK 6**



**Github Link:** <https://github.com/modcomlearning/LatestKotlin>

**Website:** <https://www.modcom.co.ke>

# Table of Contents

Introduction.....	5
What You'll Learn from This Book.....	5
Where Kotlin Excels.....	6
Getting Started.....	7
Creating a Hello World Program.....	7
Code Explanation.....	7
Comments in Kotlin.....	8
1. Single-line Comments:.....	8
2. Multi-line Comments:.....	8
Chapter 1: Variables and Data Types.....	9
What You'll Learn in This Chapter:.....	9
What are variables.....	9
Syntax.....	9
Example.....	10
Differences Between val and var.....	10
val (Immutable).....	10
var (Mutable).....	10
Summary.....	10
Working with variables.....	10
Variable Naming Rules:.....	11
Example 1: Declaring and initializing Variables.....	11
Explanation:.....	12
Kotlin Data Types.....	12
Example 2: Variables with Data types.....	13
Arithmetic Operators.....	14
Example 3: Adding 2 Numbers.....	14
Assignment.....	15
Extra Work.....	15
Explanation:.....	15
Practice Assignment: Body Mass Index (BMI) Calculator.....	16
Summary.....	16
Chapter 2: Control Statements.....	17
Conditional Statements.....	17
1. if Statement.....	17
2. when Statement.....	17
Comparison/Relational Operators.....	17
Logical Operators.....	18
if Statement in Kotlin.....	18
Syntax.....	18
IF - Flow Chart.....	18
Example 1.....	19
Explanation.....	19
if-Else Statement in Kotlin.....	19
Syntax:.....	20
IF ELSE Flow Chart.....	20
Example 2.....	21
Explanation.....	21
if-Else-if Statement in Kotlin.....	21

Syntax.....	21
IF..ELSE ..IF - Flowchart.....	22
Example 3.....	23
Explanation.....	23
Example 4.....	24
Explanation.....	24
Assignment.....	25
when Statement.....	25
Syntax.....	25
Example 5.....	26
Explanation.....	26
Practice Assignment.....	26
Summary.....	27
Chapter 3: Looping/Iterative Statements and Arrays.....	28
1. for Loop.....	28
Syntax.....	28
For Loop – Flow Chart.....	28
Explanation.....	29
Assignment.....	29
2. while Loop.....	30
Syntax.....	30
Explanation.....	30
Assignment.....	30
Summary.....	31
Introduction to Arrays in Kotlin.....	31
Example 1: Creating and Accessing an Array.....	31
Explanation:.....	31
Example 2: Looping/Iterate Through an Array Using a for Loop.....	32
Explanation:.....	32
Assignment:.....	32
Summary.....	32
Chapter 4: Functions in Kotlin.....	34
Function Definition:.....	34
Syntax.....	34
Function Basic Usage.....	34
Calling a Function.....	34
Example 1.....	35
Explanation.....	35
Example 2.....	36
Explanation.....	36
Summary.....	36
Functions with Parameters.....	37
Syntax.....	37
Explanation.....	37
Example 3.....	38
Explanation.....	38
Example 4.....	39
Explanation.....	39
Assignment.....	39
Summary.....	39

Chapter 5: Introduction to Object-Oriented Programming (OOP).....	41
What is OOP?.....	41
Advantages of Object-Oriented Programming (OOP).....	41
Objects in OOP.....	42
Relate states and behaviors to Programming:.....	42
Classes in OOP.....	42
Syntax.....	42
Example 1.....	43
Explanation.....	43
Example 2.....	44
Explanation.....	44
Student Assignment:.....	45
Inheritance in OOP.....	45
Example 3.....	45
Explanation.....	46
Chapter 6: Introduction to try, catch, and Exceptions:.....	47
Key Points:.....	47
Syntax.....	47
Example 1.....	47
Explanation:.....	48
Appendix A.....	49
Reserved Keywords.....	49
Revision Questions.....	51
References.....	53

## Introduction

Kotlin has emerged as a modern, expressive, and efficient programming language, designed to enhance productivity while eliminating many pitfalls of older languages. Whether you're a beginner stepping into programming or an experienced developer looking to master Kotlin, this book provides a structured and practical approach to learning.

This book covers everything from fundamental concepts to advanced techniques, ensuring that readers not only understand Kotlin but also know how to apply it effectively in real-world scenarios. Through hands-on examples, step-by-step tutorials, and best practices, you'll gain the confidence to build robust applications using Kotlin.

Whether you're aiming to develop Android apps, backend systems, or cross-platform applications, this book equips you with the practical knowledge to make the most of Kotlin's capabilities.

## What You'll Learn from This Book

1. **Setting up Kotlin** – Installing the right tools like VS Code or IntelliJ IDEA.
2. **Understanding Syntax** – Writing your first Kotlin program.
3. **Building Real-World Applications** – Creating fun and practical projects.
4. **Core Kotlin Concepts** – Mastering variables, data types, control flow, and functions with hands-on examples.
5. **Building Interactive Programs** – Writing practical applications and solving real-world problems using Kotlin.
6. **Kotlin Best Practices** – Writing clean, efficient, and maintainable Kotlin code.

By the end of this book, you'll be confident in writing Kotlin programs and ready to explore advanced topics like Android development or backend services!

## Where Kotlin Excels

- **Android App Development** – The primary language recommended by Google for Android apps.
- **Backend Services** – Used with frameworks like Ktor and Spring Boot for server-side applications.
- **Web Solutions** – Can be compiled to JavaScript for frontend projects.
- **Data Science & AI** – Compatible with machine learning tools like KotlinDL.
- **Cross-Platform Development** – Kotlin Multiplatform allows code reuse different platforms.

## Getting Started

To set up Kotlin on your system, you can follow the detailed instructions in this [link](#).

<https://how.dev/answers/how-to-set-up-kotlin-in-linux>

It provides a step-by-step guide on installing Kotlin on Linux, including all the necessary commands and configurations.

Download Visual Studio Code from <https://code.visualstudio.com/Download>

After installing VS Code add below Extensions

1. Kotlin Language
2. Code Runner

## Creating a Hello World Program

Create a Kotlin File

First, create a new file named test.kt (the .kt extension is used for Kotlin files).

- Write the Code I Visual Studio.

In the test.kt file, write the following Kotlin code:

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

Right Click and Select Run Code.

## Code Explanation

fun main() defines the main function, which is the entry point of any Kotlin program.

println("Hello, World!") prints the text Hello, World! to the console.

You can also run Kotlin Code online on <https://play.kotlinlang.org/>

## Comments in Kotlin

### 1. Single-line Comments:

**Syntax:** Starts with `//` and continues to the end of the line.

**Use:** For adding a short description or note for a single line of code.

### 2. Multi-line Comments:

**Syntax:** Starts with `/*` and ends with `*/`.

**Use:** For adding detailed explanations or commenting out multiple lines of code.



## Chapter 1: Variables and Data Types

Every programming language revolves around handling data, and Kotlin is no exception. In this chapter, you'll learn how to store and manage data efficiently using **variables** and **data types**. Mastering these concepts is essential, as they form the foundation of every Kotlin program.

### What You'll Learn in This Chapter:

**What variables are** and how to declare them in Kotlin.

**The difference between `val` and `var`** – when to use immutable vs. mutable variables.

**Kotlin's built-in data types** – numbers, characters, booleans, and strings.

**Type inference in Kotlin** – letting the compiler determine data types automatically.

**Hands-on coding examples** to reinforce your understanding.

By the end of this chapter, you'll be able to declare and use variables confidently, ensuring your Kotlin programs efficiently handle different types of data.

### What are variables

A **variable** in programming is like a **container** that holds a value. The value can be a number, a piece of text, or any other type of data. You can think of it as a labeled box where you store something.

For example, in Kotlin, you can create a variable to store a number like this:

### Syntax

```
val name = initial value
```

## Example

```
val age = 25
```

- **val** is the keyword that tells Kotlin you're creating a variable.
- **age** is the name of the variable (the label on the box). Is an Identifier
- **25** is the value that the variable holds (the contents of the box).

In simple terms, a variable is just a way to store and keep track of data in your program!

## Differences Between val and var.

### val (Immutable)

Once you give it a value, you **can't change** that value anymore. Its like a Fixed Container.

```
val age = 25
```

```
age = 30 // Error! You can't change the value of a val variable
```

### var (Mutable)

You can **change** the value stored in it as many times as you want.

Its like a Flexible Container.

```
var age = 25
```

```
age = 30 // This is perfectly fine! You can change the value of a var.
```

## Summary

- **val**: You set the value once, and it **cannot** be changed.
- **var**: You can set and **change** the value as many times as needed.

## Working with variables

In Kotlin, when we declare a variable, we don't always need to specify the **data type**.

Instead, we can let the compiler **figure it out automatically** based on the value we assign

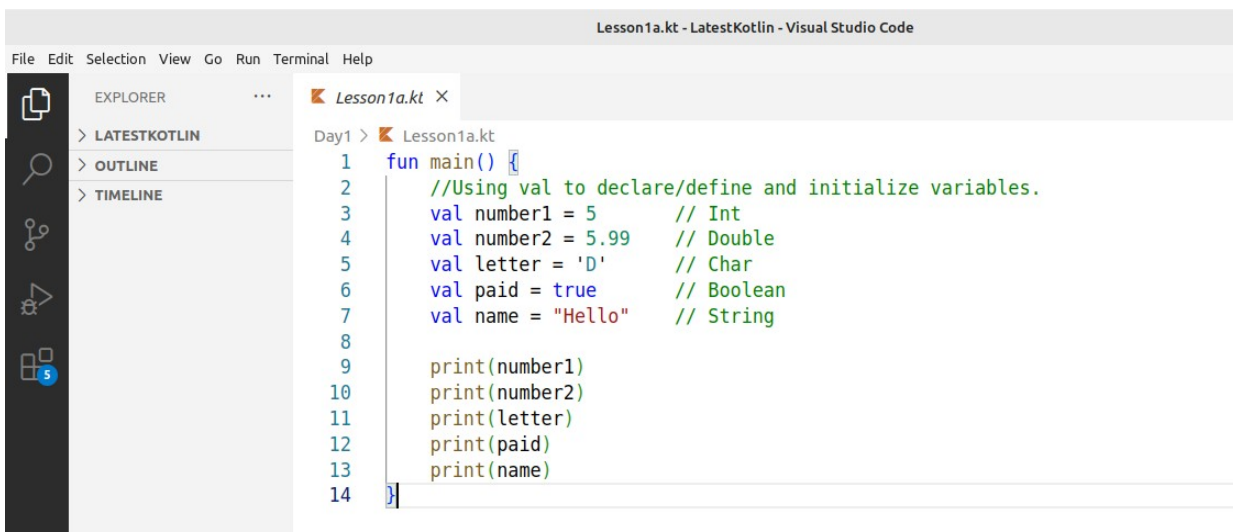
to the variable. This is known as **type inference**. The Data Type is the Type of Data Stored in a variable.

### Variable Naming Rules:

- **Start with a letter or an underscore:** The variable name must start with a letter (a-z or A-Z) or an underscore (\_).
- **Use letters, numbers, or underscores:** After the first character, you can use letters, numbers (0-9), or underscores.
- **Cannot use Kotlin keywords:** Reserved keywords like fun, val, var, etc., cannot be used as variable names. Check reserved keywords in **Appendix A**
- **Naming conventions:** By convention, variables are typically named using **camelCase** (starting with a lowercase letter and capitalizing subsequent words), e.g., userAge, productPrice, product\_price

### Example 1: Declaring and initializing Variables

Create a File named **Lesson1a.kt** and write below code.



```
Lesson1a.kt - LatestKotlin - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
> LATESTKOTLIN
> OUTLINE
> TIMELINE
Lesson1a.kt
Day1 > Lesson1a.kt
1 fun main() {
2     //Using val to declare/define and initialize variables.
3     val number1 = 5           // Int
4     val number2 = 5.99       // Double
5     val letter = 'D'         // Char
6     val paid = true          // Boolean
7     val name = "Hello"       // String
8
9     print(number1)
10    print(number2)
11    print(letter)
12    print(paid)
13    print(name)
14 }
```

### Explanation:

- **val number1 = 5:** This creates a variable number1 and assigns it the value 5. The Kotlin compiler automatically infers the data type as Int (an integer).
- **val number2 = 5.99:** This creates a variable number2 and assigns it the value 5.99. The compiler automatically recognizes it as a Double (a number with decimal points).
- **val letter = 'D':** This creates a variable letter and assigns it the value 'D'. The compiler recognizes it as a Char (a single character).
- **val paid = true:** This creates a variable paid and assigns it the value true. The compiler recognizes it as a Boolean (either true or false).
- **val name = "Hello":** This creates a variable name and assigns it the value "Hello". The compiler recognizes it as a String (a sequence of characters).

### Printing the Values:

- **print(number1):** This prints the value of number1, which is 5.
- **print(number2):** This prints the value of number2, which is 5.99.
- **print(letter):** This prints the value of letter, which is D.
- **print(paid):** This prints the value of paid, which is true.
- **print(name):** This prints the value of name, which is "Hello".

In above example, we leave kotlin to automatically identify the Data Types.

### Kotlin Data Types

In the next example we redo **Lesson1a.kt** example and specify the Data Types.

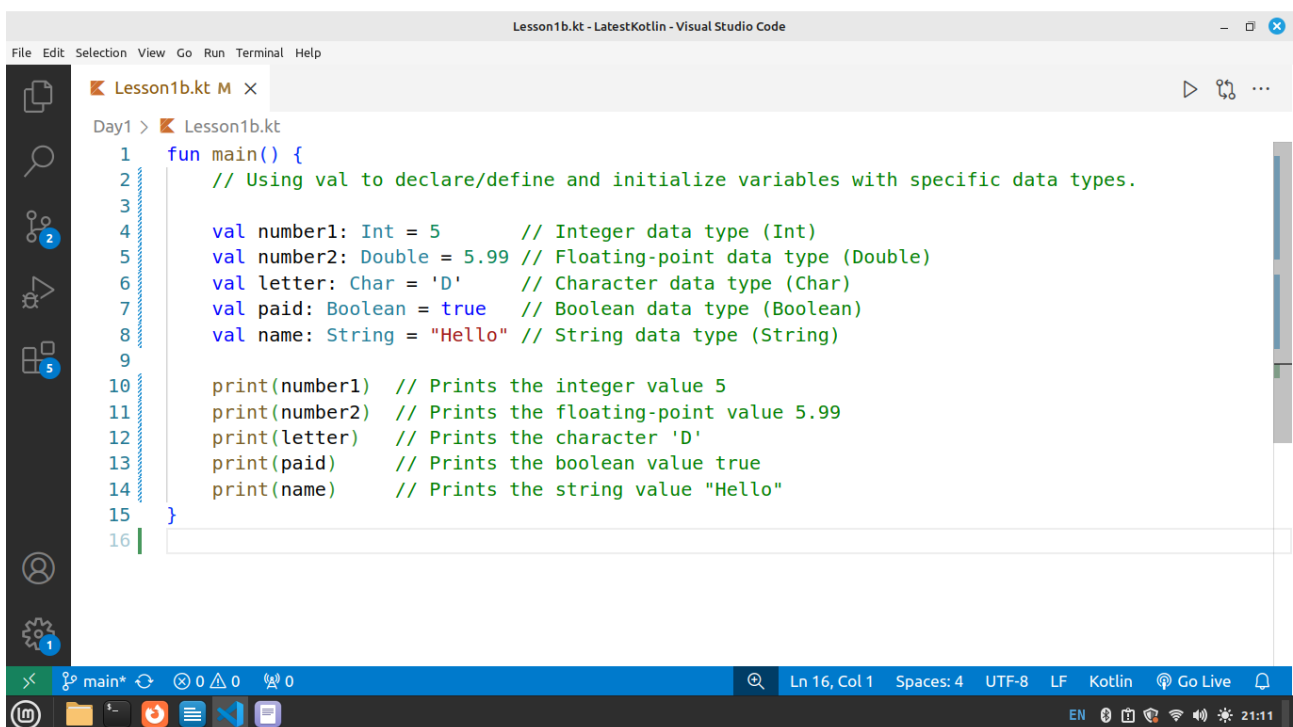
We look at below Major Data Types,

- **Int:** Used for whole numbers.
- **Double:** Used for numbers with decimals.
- **Char:** Used for a single character.
- **Boolean:** Used for true/false values.
- **String:** Used for text (sequence of characters).

Each data type serves a specific purpose, and you use the appropriate type depending on the kind of data you want to store and work with.

## Example 2: Variables with Data types

Create **Lesson2a.kt** and write below code.



```
Lesson1b.kt - LatestKotlin - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Lesson1b.kt M x
Day1 > Lesson1b.kt
1 fun main() {
2     // Using val to declare/define and initialize variables with specific data types.
3
4     val number1: Int = 5        // Integer data type (Int)
5     val number2: Double = 5.99 // Floating-point data type (Double)
6     val letter: Char = 'D'     // Character data type (Char)
7     val paid: Boolean = true   // Boolean data type (Boolean)
8     val name: String = "Hello" // String data type (String)
9
10    print(number1) // Prints the integer value 5
11    print(number2) // Prints the floating-point value 5.99
12    print(letter)  // Prints the character 'D'
13    print(paid)    // Prints the boolean value true
14    print(name)    // Prints the string value "Hello"
15 }
16
```

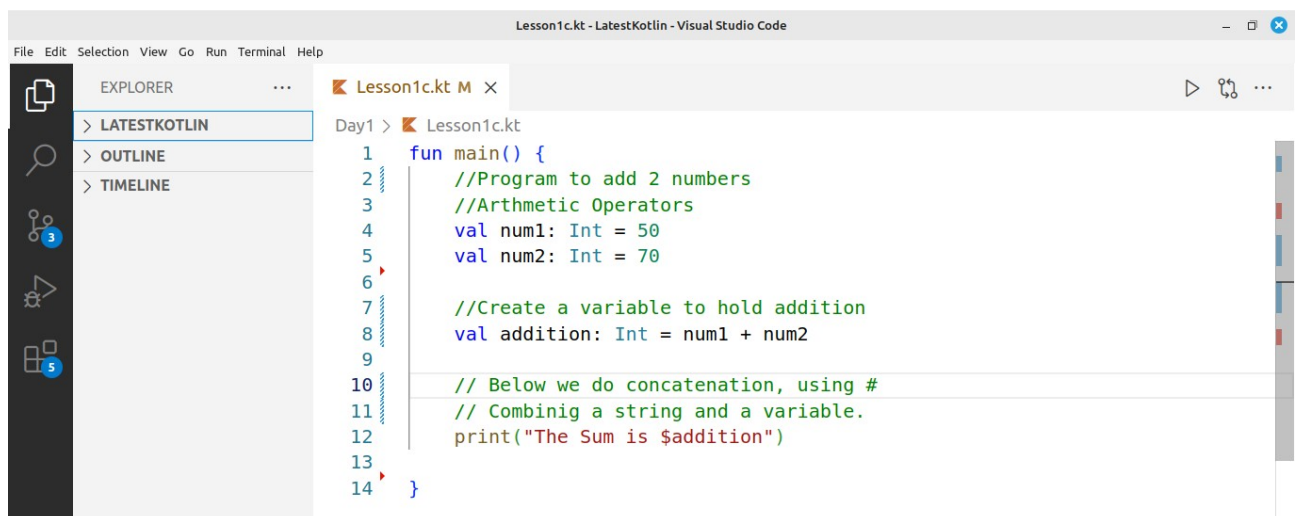
In the Next example, we create a simple calculator using Kotlin variables, First lets look at arithmetic operators to be used in our Calculations.

### Arithmetic Operators.

Operator	Name	Description	Example
+	Addition	Adds together two values	$x + y$
-	Subtraction	Subtracts one value from another	$x - y$
*	Multiplication	Multiplies two values	$x * y$
/	Division	Divides one value by another	$x / y$
%	Modulus	Returns the division remainder	$x \% y$

### Example 3: Adding 2 Numbers

Create a File named **Lesson2c.kt** and write below code

A screenshot of the Visual Studio Code editor. The title bar reads 'Lesson1c.kt - LatestKotlin - Visual Studio Code'. The Explorer sidebar on the left shows a project named 'LATESTKOTLIN' with sub-items 'OUTLINE' and 'TIMELINE'. The main editor area shows a file named 'Lesson1c.kt' with the following Kotlin code:

```
1 fun main() {
2     //Program to add 2 numbers
3     //Arithmetic Operators
4     val num1: Int = 50
5     val num2: Int = 70
6
7     //Create a variable to hold addition
8     val addition: Int = num1 + num2
9
10    // Below we do concatenation, using #
11    // Combinig a string and a variable.
12    print("The Sum is $addition")
13
14 }
```

**val num1: Int = 50:** Declares and initializes an integer variable num1 with the value 50.

**val num2: Int = 70:** Declares and initializes another integer variable num2 with the value 70.

**val addition: Int = num1 + num2:** Performs the addition of num1 and num2, stores the result (120) in the variable addition.

**print("The Sum is \$addition"):** Uses string interpolation to combine the string "The Sum is " with the value stored in addition (120) and prints the result.

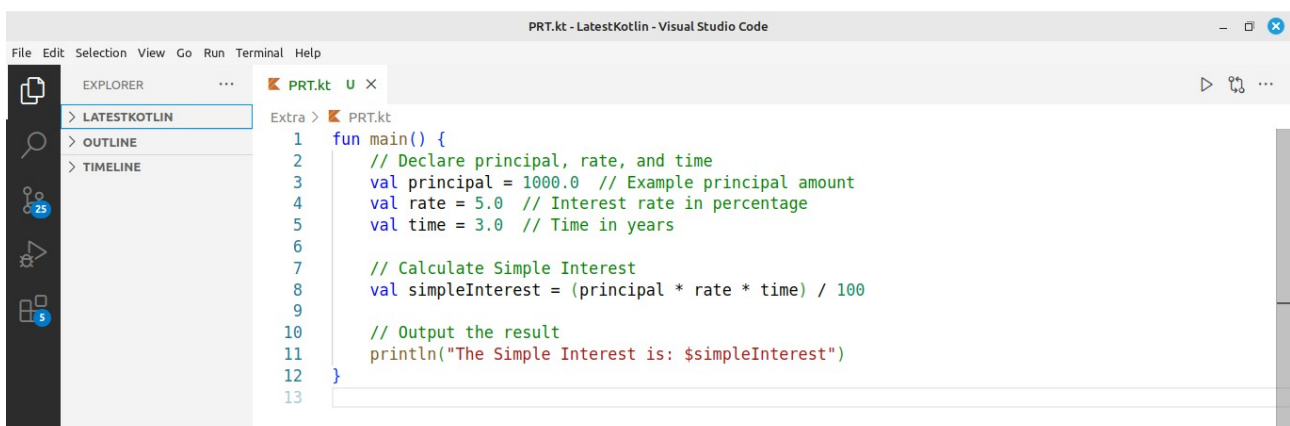
## Assignment

In this assignment, you will build up the code to perform **subtraction**, **division**, and **multiplication**.

## Extra Work

In below example we use variables and create a simple interest calculator given Formula PRT/100.

Create a File named **prrt.kt** and write below code.



```
1 fun main() {
2     // Declare principal, rate, and time
3     val principal = 1000.0 // Example principal amount
4     val rate = 5.0 // Interest rate in percentage
5     val time = 3.0 // Time in years
6
7     // Calculate Simple Interest
8     val simpleInterest = (principal * rate * time) / 100
9
10    // Output the result
11    println("The Simple Interest is: $simpleInterest")
12 }
13
```

## Explanation:

- Creates and stores values in 3 variables namely principal, rate, and time
- The program calculates the interest by multiplying the principal, rate, and time, then dividing by 100.

- Stores the answer in **simpleinterest** variable
- prints the **simpleinterest** variable

### Practice Assignment: Body Mass Index (BMI) Calculator

**Objective:** Create a program to calculate the **Body Mass Index (BMI)** using the formula:

$\text{body\_mass\_index} = \text{weight} / \text{height} * \text{height}$

Where:

- **weight** is in kilograms (kg)
- **height** is in meters (m)

### Summary

- **Variables** store data/values in a program.
- **val**: Immutable (value can't be changed).
- **var**: Mutable (value can be changed).
- **Int**: Whole numbers (e.g., 5, -10).
- **Double**: Numbers with decimals (e.g., 5.99, -2.5).
- **Char**: Single characters (e.g., 'A', 'B').
- **Boolean**: True or false values (e.g., true, false).
- **String**: Text (e.g., "Hello", "Kotlin").
- Kotlin can automatically detect the data type based on the assigned value



## Chapter 2: Control Statements

In this chapter, we will explore **control statements** in Kotlin. Control statements allow you to make decisions and repeat actions based on certain conditions. The two primary types of control statements are **Conditional Statements** and **Looping Statements**.

### Conditional Statements

Conditional statements are used to make decisions based on whether a specific condition is true or false. They allow you to execute certain blocks of code depending on the condition's result. In Kotlin, there are two common types of conditional statements:

#### 1. if Statement

#### 2. when Statement

First we need to understand Comparison and Logical Operators

### Comparison/Relational Operators

Operator	Name	Example
>	greater than	<code>x &gt; y</code>
<	less than	<code>x &lt; y</code>
>=	greater than or equal to	<code>x &gt;= y</code>
<=	less than or equal to	<code>x &lt;= y</code>
==	is equal to	<code>x == y</code>
!=	not equal to	<code>x != y</code>

## Logical Operators

Operator	Name	Description	Example
&&	Logical and	Returns true if both operands are true	x && y
	Logical or	Returns true if either of the operands is true	x    y
!	Logical not	Reverse the result, returns false if the operand is true	!x

If statement has three statements namely **IF**, **IF ELSE** and **ELSE IF**.

### if Statement in Kotlin

The if statement allows you to make decisions based on a condition. If statements runs when the condition is true. Here's the basic syntax

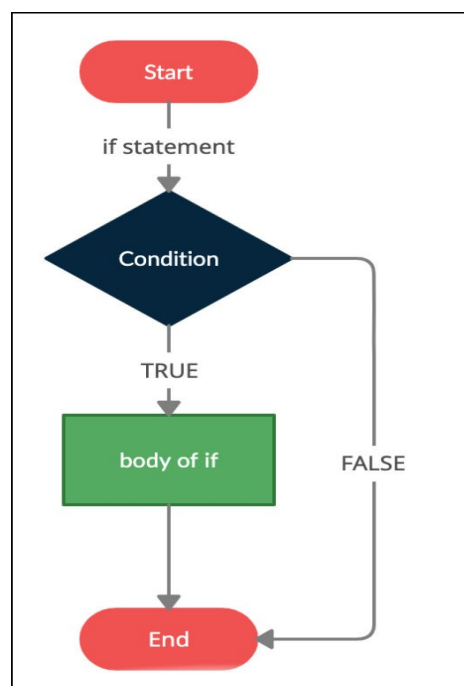
#### Syntax

```
if (condition) {
```

```
    // IF Block/Body Code to execute if condition is true
```

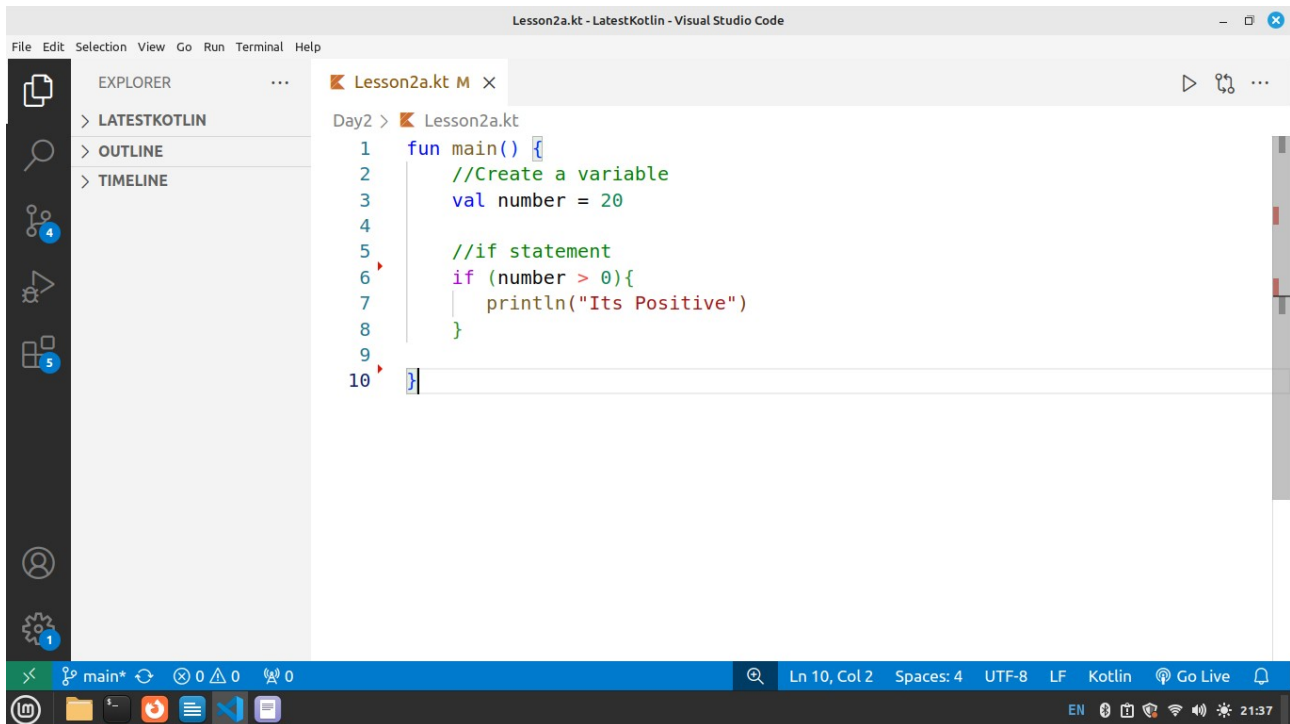
```
}
```

### IF - Flow Chart



## Example 1

Create a File named **Lesson2a.kt** and write below code.



The screenshot shows the Visual Studio Code editor with a file named Lesson2a.kt open. The code is written in Kotlin and is as follows:

```
1 fun main() {  
2     //Create a variable  
3     val number = 20  
4  
5     //if statement  
6     if (number > 0){  
7         println("Its Positive")  
8     }  
9  
10 }
```

The editor interface includes a sidebar with Explorer, Outline, and Timeline views. The status bar at the bottom shows the current line and column (Ln 10, Col 2) and the file encoding (UTF-8).

## Explanation

- Define a variable number and initialize it to an integer value (20).
- The if statement checks whether the number is greater than zero.
- If the condition (number > 0) is **true**, the code inside the if block executes.
- If the condition is **false**, the if statement exits and no further action is taken.

## if-Else Statement in Kotlin

The if-else statement allows you to check a condition and execute one block of code if the condition is **true**, and another block if the condition is **false**.

**Syntax:**

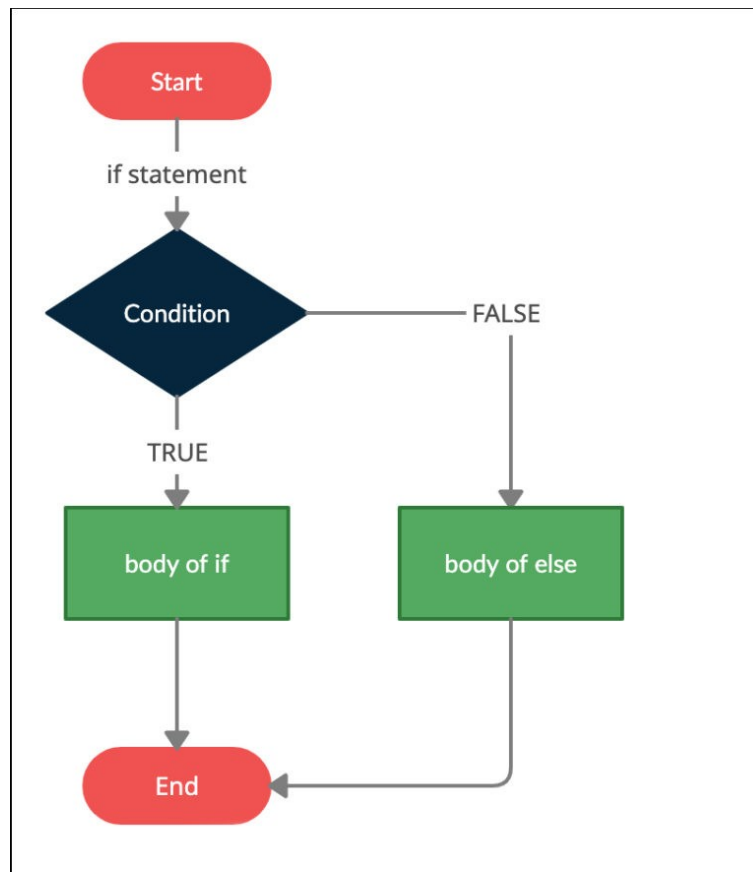
```
if (condition) {
```

```
    // IF Block/Body Code to execute if condition is true
```

```
} else {
```

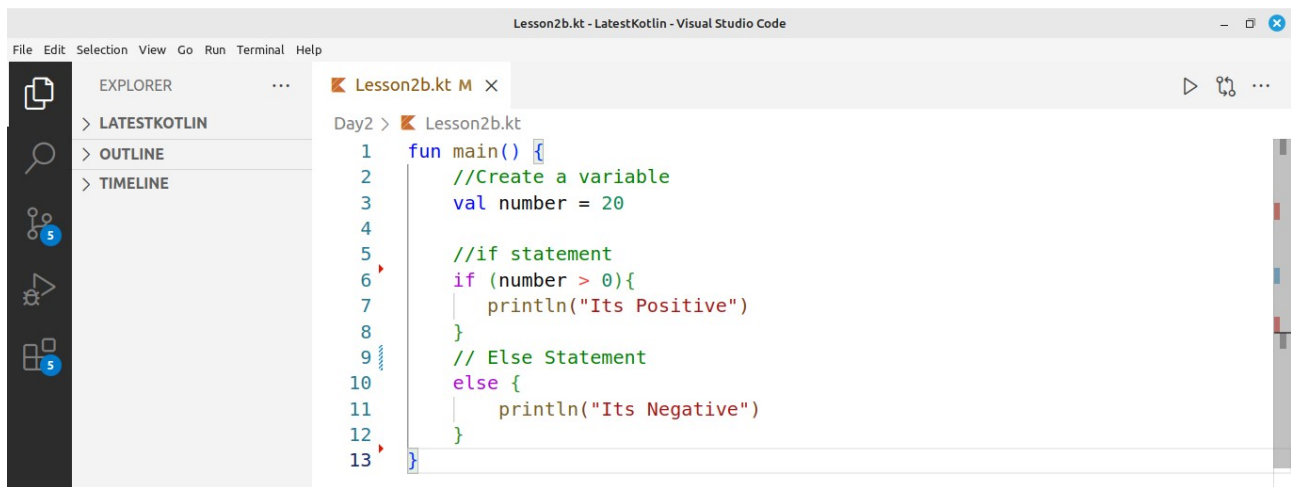
```
    // Else Block/Body Code to execute if condition is false
```

```
}
```

**IF ELSE Flow Chart**

## Example 2

Create a File named **Lesson2b.kt** and write below code.



```
Lesson2b.kt - LatestKotlin - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
> LATESTKOTLIN
> OUTLINE
> TIMELINE
Lesson2b.kt M x
Day2 > Lesson2b.kt
1 fun main() {
2     //Create a variable
3     val number = 20
4
5     //if statement
6     if (number > 0){
7         println("Its Positive")
8     }
9     // Else Statement
10    else {
11        println("Its Negative")
12    }
13 }
```

## Explanation

**if-else statement:** Used to make decisions based on a condition.

**If the condition is true,** the code inside the if block executes.

**If the condition is false,** the code inside the else block executes.

**The if block runs if the condition (`number > 0`) is true,** otherwise the else block runs.

## if-Else-if Statement in Kotlin

The if-else-if statement allows you to check multiple conditions in sequence. It evaluates each condition one by one until it finds the first true condition and executes the corresponding block of code.

## Syntax

```
if (condition1) {
```

```
    // Code to execute if condition1 is true
```

```
} else if (condition2) {
```

```

    // Code to execute if condition2 is true

} else if (condition3) {

    // Code to execute if condition3 is true

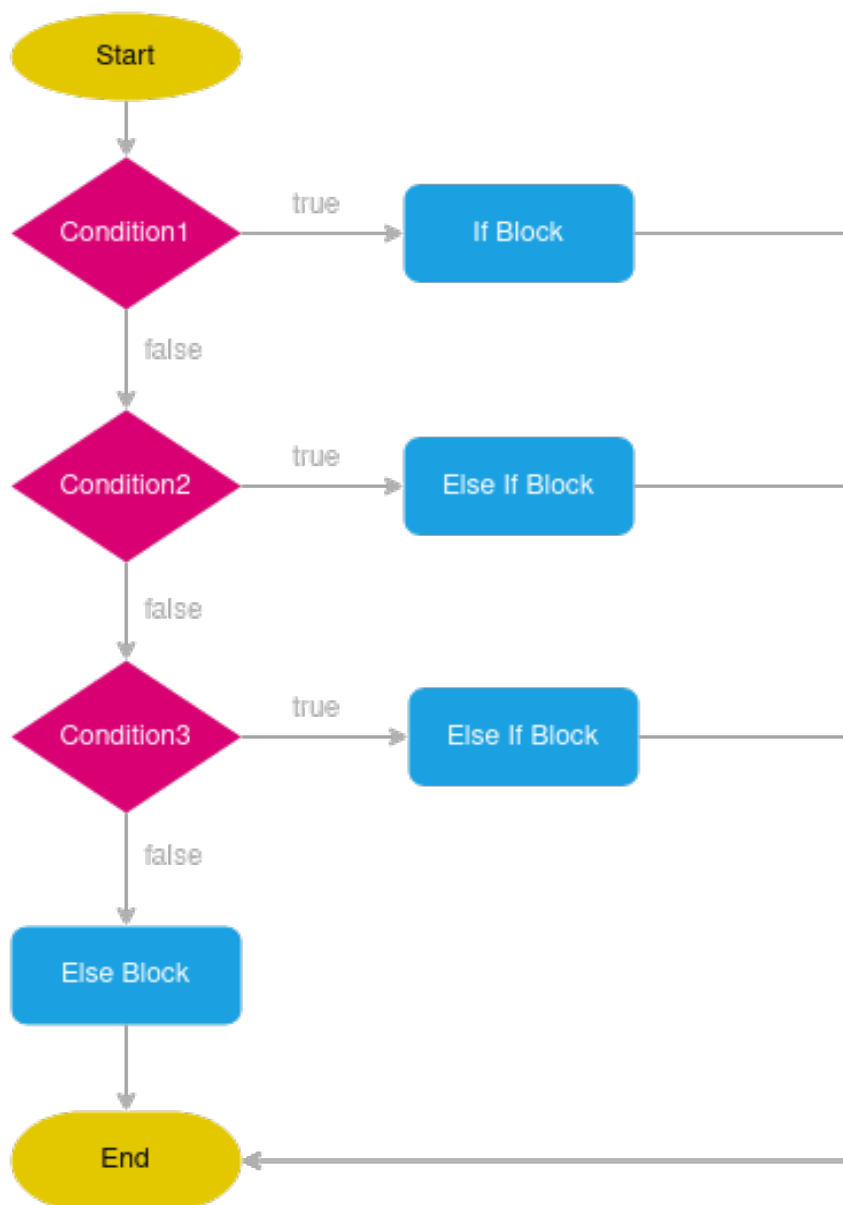
} else {

    // Code to execute if none of the conditions are true

}

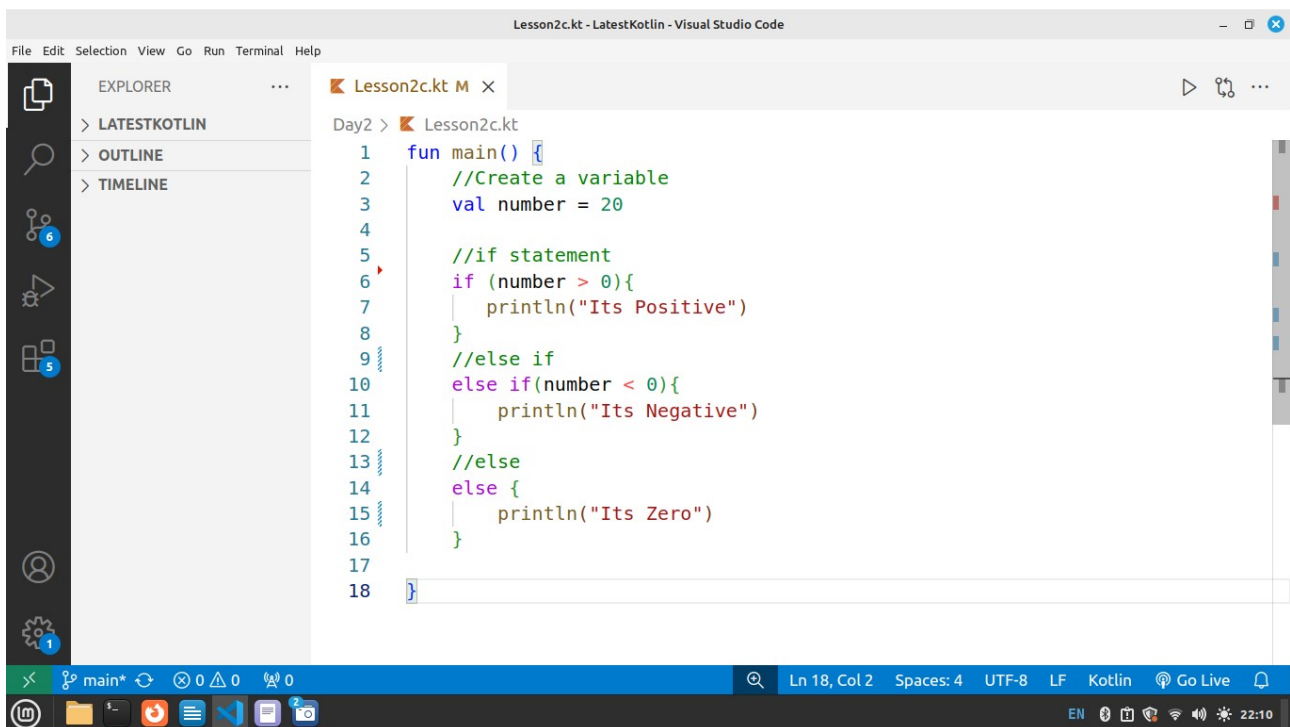
```

### IF..ELSE ..IF - Flowchart



### Example 3

Create a File named **Lesson2c.kt** and write below code.



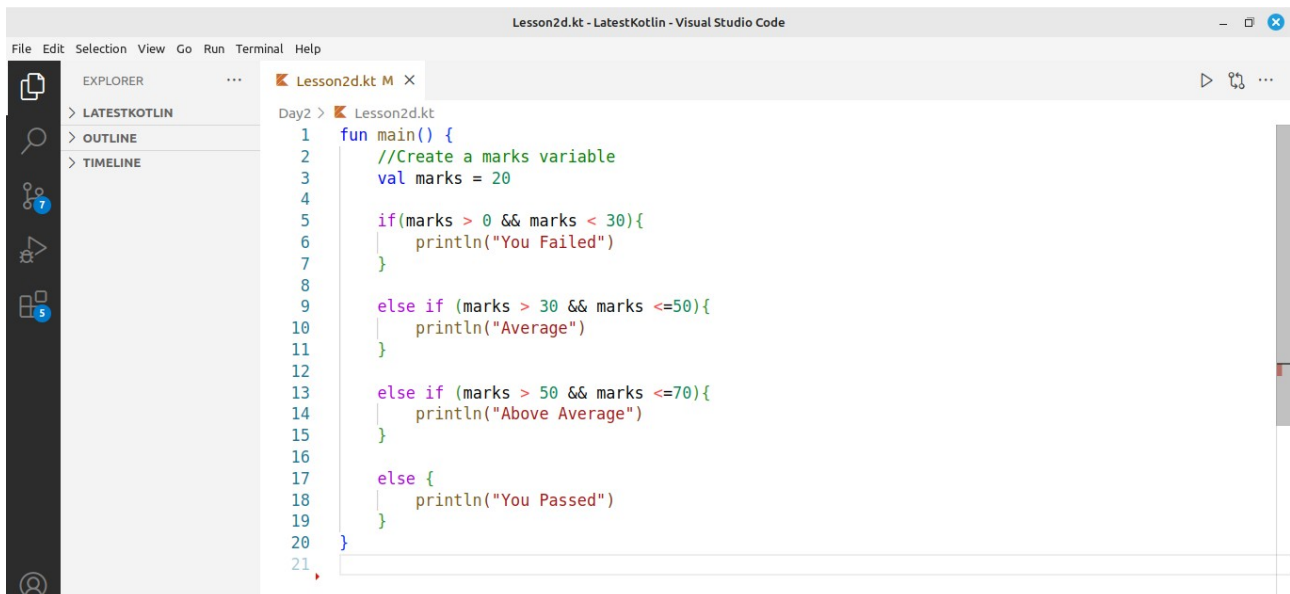
```
1 fun main() {
2     //Create a variable
3     val number = 20
4
5     //if statement
6     if (number > 0){
7         println("Its Positive")
8     }
9     //else if
10    else if (number < 0){
11        println("Its Negative")
12    }
13    //else
14    else {
15        println("Its Zero")
16    }
17 }
18 }
```

### Explanation

- **Creates a variable** number and assigns it the value 20.
- **Checks if number is greater than 0** → Prints "Its Positive".
- **If number is less than 0** → Prints "Its Negative".
- **If neither condition is met (number is 0)** → Prints "Its Zero".
- Uses **if-else-if-else** to check multiple conditions in sequence.

## Example 4

Create a File named Lesson2d.kt and write below code, here we make use of logical operator && - AND.



```
1 fun main() {
2     //Create a marks variable
3     val marks = 20
4
5     if(marks > 0 && marks < 30){
6         println("You Failed")
7     }
8
9     else if (marks > 30 && marks <=50){
10        println("Average")
11    }
12
13    else if (marks > 50 && marks <=70){
14        println("Above Average")
15    }
16
17    else {
18        println("You Passed")
19    }
20 }
21
```

## Explanation

- **Create a variable** marks and assign it the value 20.
- **if statement:** Checks if marks are between 0 and 30 (not inclusive of 30) → Prints "You Failed".
- **else if statements:**
  - Checks if marks are between 30 and 50 (inclusive of 50) → Prints "Average".
  - Checks if marks are between 50 and 70 (inclusive of 70) → Prints "Above Average".
- **Final else:** If none of the conditions match, it assumes the marks are above 70 → Prints "You Passed".



- **&& (AND):** Combines two conditions, making sure both must be true for the statement to execute

## Assignment

Given the table below, Create a Kotlin program to check the price to WIN given the points a customer has acquired

Points	Price to WIN
0 - 100	You do not qualify
101 – 400	You Win a Smartphone
401 – 1000	You win a Laptop
1000 and above	You win a Trip to Canada

The points input cannot be negative.

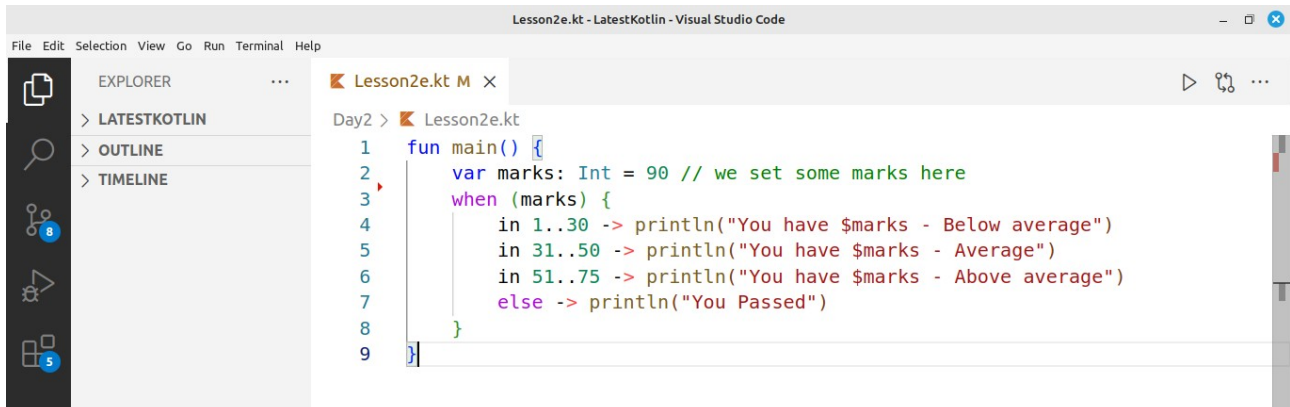
## when Statement

The when statement in Kotlin is like a switch case in other languages. It checks a value against multiple conditions and executes the matching block.

## Syntax

```
when (expression) {  
    case1 -> // Code to execute if case1 matches  
    case2 -> // Code to execute if case2 matches  
    else -> // Code to execute if no cases match  
}
```

## Example 5



```
1 fun main() {
2     var marks: Int = 90 // we set some marks here
3     when (marks) {
4         in 1..30 -> println("You have $marks - Below average")
5         in 31..50 -> println("You have $marks - Average")
6         in 51..75 -> println("You have $marks - Above average")
7         else -> println("You Passed")
8     }
9 }
```

## Explanation

- **Declares a variable** marks with a value of 90.
- **when statement** checks the range in which marks fall.
- **in 1..30** → Prints "Below average".
- **in 31..50** → Prints "Average".
- **in 51..75** → Prints "Above average".
- **else** → If marks do not match any range, prints "You Passed".

## Practice Assignment

1. Using if statements or when statement, create a program to find if given year is leap or not
2. Using if statements check if a given number is ODD or Even

## Summary

- **if Statement:** Runs a block of code only when the condition is true.
- **if-else Statement:** Executes one block if the condition is true, otherwise runs the else block.
- **if-else-if Statement:** Checks multiple conditions in order and executes the first one that is true.
- **when Statement:** Works like a switch case, matching a value against multiple possible cases.

## Chapter 3: Looping/Iterative Statements and Arrays

Looping statements allow us to repeat a block of code multiple times. Kotlin provides different types of loops to handle various scenarios efficiently.

Like other programming languages, Kotlin provides two types of Loops namely;

1. For Loop
2. While Loop

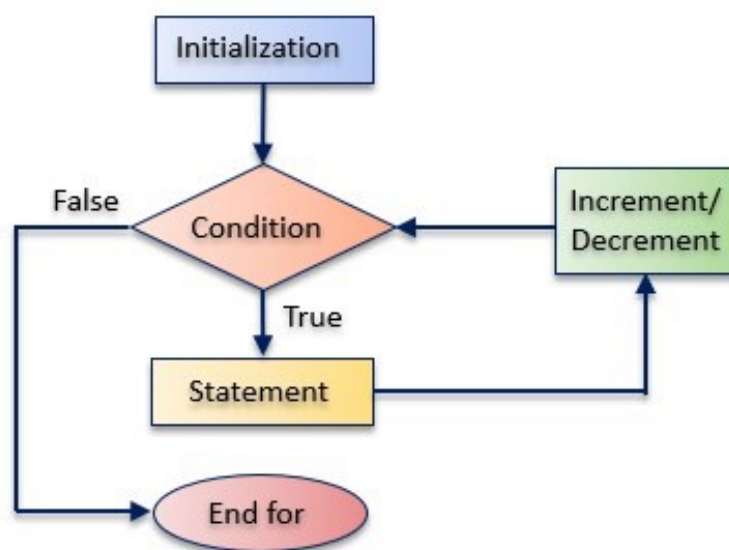
### 1. for Loop

Used to repeat a Block of code several times. For loop runs a fixed number of times

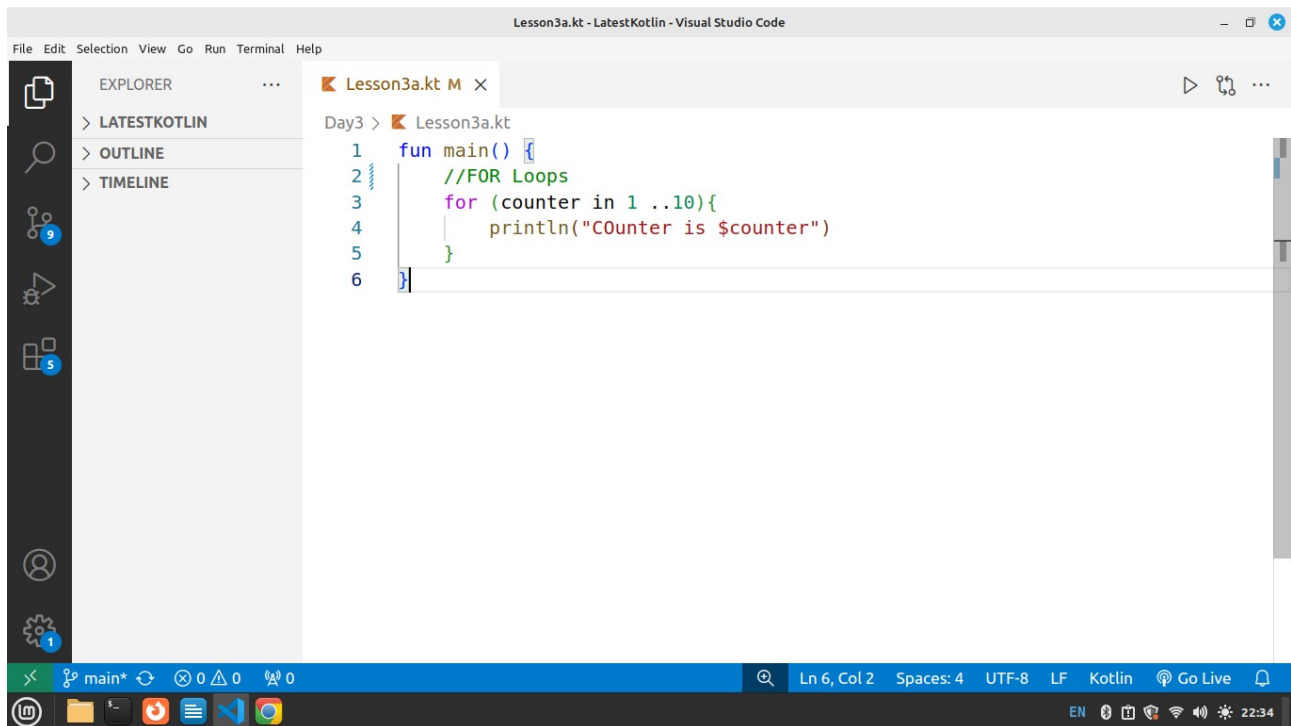
#### Syntax

```
for (variable in range) {  
    // Code to execute in each iteration  
}
```

#### For Loop – Flow Chart.



Create a File named **Lesson3a.kt** and write below for loop code.



The screenshot shows the Visual Studio Code editor with a file named `Lesson3a.kt` open. The code is as follows:

```
1 fun main() {  
2     //FOR Loops  
3     for (counter in 1..10){  
4         println("COUNTER is $counter")  
5     }  
6 }
```

The interface includes the Explorer sidebar on the left, the main editor area, and the status bar at the bottom showing 'Ln 6, Col 2', 'Spaces: 4', 'UTF-8', 'LF', 'Kotlin', and 'Go Live'.

## Explanation

- **Uses a for loop** to repeat code multiple times.
- **counter in 1..10** means the loop runs from 1 to 10.
- **Each iteration**, counter takes the next value in the range.
- **Prints** "Counter is" followed by the current value of counter.
- **Stops after 10**, as the loop completes its range.

## Assignment

1. Create a For loop to print year from 2000 to 2025
2. Create a For loop to print from 100 to 1

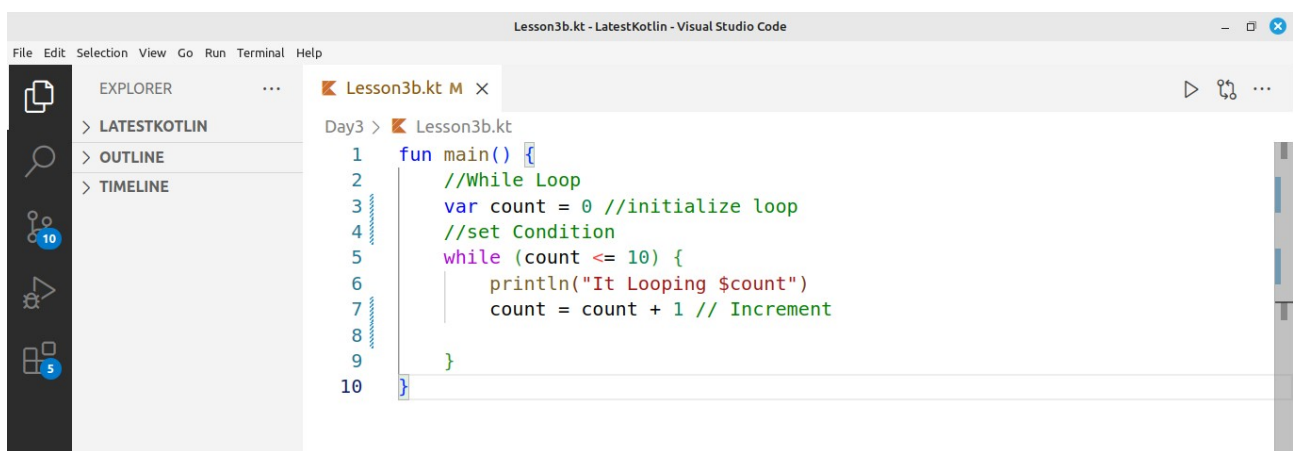
## 2. while Loop

Executes and repeats a block of code **as long as** the condition is true.

### Syntax

```
while (condition) {  
  
    // Code to execute while the condition is true  
  
}
```

Create file named **Lesson3b.kt** and write below code.



```
Lesson3b.kt - LatestKotlin - Visual Studio Code  
File Edit Selection View Go Run Terminal Help  
EXPLORER  
> LATESTKOTLIN  
> OUTLINE  
> TIMELINE  
Lesson3b.kt M X  
Day3 > Lesson3b.kt  
1 fun main() {  
2     //While Loop  
3     var count = 0 //initialize loop  
4     //set Condition  
5     while (count <= 10) {  
6         println("It Looping $count")  
7         count = count + 1 // Increment  
8     }  
9 }  
10 }
```

### Explanation

- **Initializes** count to **0** before the loop starts.
- **while (count <= 10)** checks if count is 10 or less before looping.
- **Prints** "It Looping" followed by the current value of count.
- **Increments count by 1** in each iteration to prevent an infinite loop.
- **Stops when count reaches 11**, as the condition becomes false.

### Assignment

1. Create a while loop to print from -20 to +20
2. Create a while loop to print all EVEN years from 2000 to 2025.

## Summary

- Loops are used to run a block of code several times
- There are two types of loops for and while loops
- **For Loop**: Runs a block of code a set number of times.
- while **Loop**: Runs **as long as** the condition remains true.

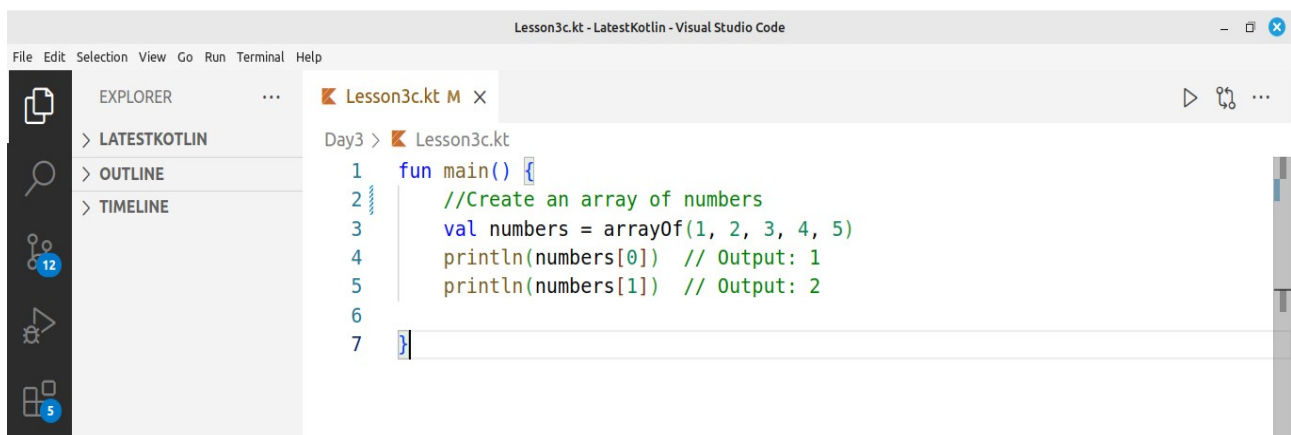
## Introduction to Arrays in Kotlin

An **array** is a collection of elements of the same data type, stored in a single variable.

Arrays allow us to store and manage multiple values efficiently.

### Example 1: Creating and Accessing an Array

Create a File named **Lesson3c.kt** and write below code.

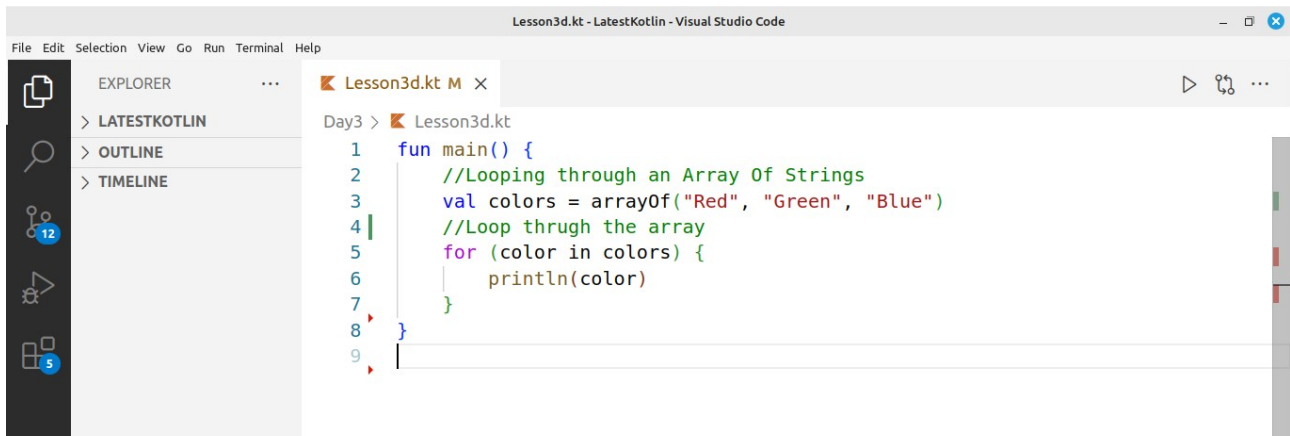


```
Lesson3c.kt - LatestKotlin - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
> LATESTKOTLIN
> OUTLINE
> TIMELINE
Lesson3c.kt M x
Day3 > Lesson3c.kt
1 fun main() {
2     //Create an array of numbers
3     val numbers = arrayOf(1, 2, 3, 4, 5)
4     println(numbers[0]) // Output: 1
5     println(numbers[1]) // Output: 2
6
7 }
```

### Explanation:

- The arrayOf() function creates an array with elements 1, 2, 3, 4, 5.
- Elements are accessed using an **index**, starting from 0.
- numbers[0] prints the first element (1), and numbers[1] prints the second element (2).

## Example 2: Looping/Iterate Through an Array Using a for Loop



```
Lesson3d.kt - LatestKotlin - Visual Studio Code
File Edit Selection View Go Run Terminal Help

EXPLORER
> LATESTKOTLIN
> OUTLINE
> TIMELINE

Lesson3d.kt M x
Day3 > Lesson3d.kt
1 fun main() {
2     //Looping through an Array Of Strings
3     val colors = arrayOf("Red", "Green", "Blue")
4     //Loop through the array
5     for (color in colors) {
6         println(color)
7     }
8 }
9 |
```

### Explanation:

- Defines an array colors with values "Red", "Green", "Blue".
- Uses a **for loop** to iterate through the array and print each color.

### Assignment:

1. Create an **array of planets** (e.g., Mercury, Venus, Earth, etc.).
2. Use a **loop** to print each planet.

### Summary

- **Array:** A collection of elements of the same data type stored in a single variable.
- **Accessing Elements:** Elements are accessed using an **index**, starting from 0.
- **Creating an Array:** Use the arrayOf() function to create an array.
- **Looping through Arrays:** You can use a for loop to iterate through all elements in the array.



- **Benefits:** Arrays help store multiple values of the same type, making data management easier.
- **Example Usage:** Storing lists like numbers, colors, or names in an organized way.

## Chapter 4: Functions in Kotlin

A **function** in Kotlin is a group of statements that are executed together to perform a specific task. Functions help to organize the code, making it reusable, modular, and easy to understand.

### Function Definition:

A function is defined using the **fun** keyword, followed by its name and the block of code (statements) that it performs when called

### Syntax

```
fun functionName() {  
  
    // Function body  
  
}
```

### Function Basic Usage.

```
fun sayHello() {  
    println("Hello, world!") // Statement 1  
    println("Welcome to Kotlin!") // Statement 2  
}
```

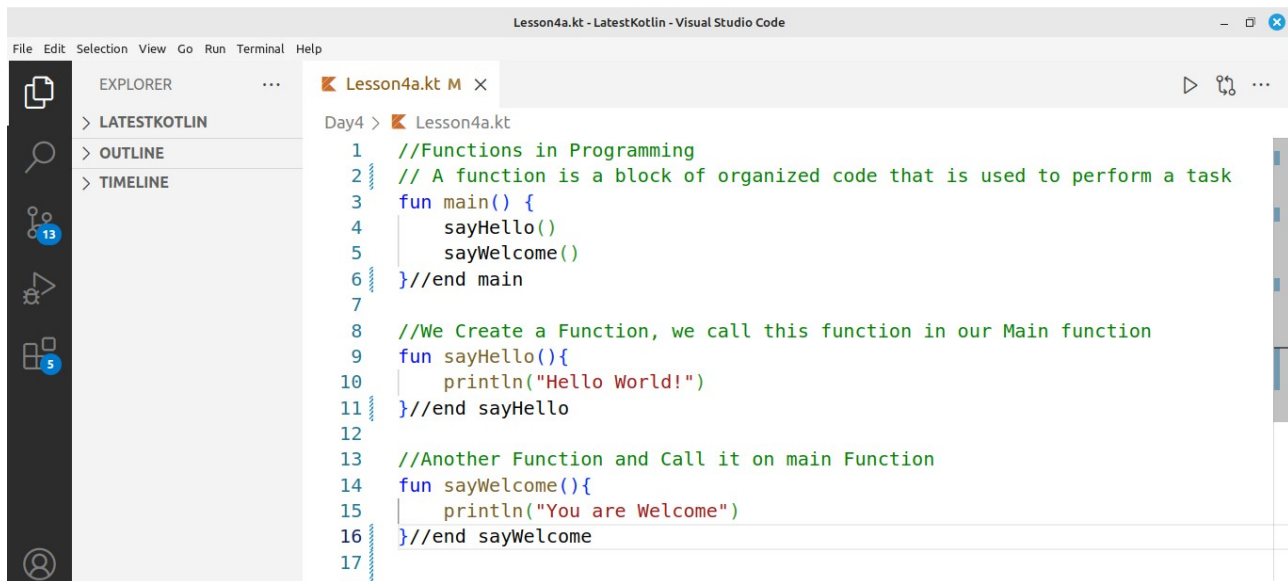
### Calling a Function

A function in Kotlin is Accessed/Called/Invoked using its name. Like below.

```
sayHello() // Calls the function and executes the statements inside it
```

## Example 1

Create a File named Lesson4a.kt and write below function.



```
Lesson4a.kt - LatestKotlin - Visual Studio Code
File Edit Selection View Go Run Terminal Help

EXPLORER
> LATESTKOTLIN
> OUTLINE
> TIMELINE

Lesson4a.kt M x
Day4 > Lesson4a.kt
1 //Functions in Programming
2 // A function is a block of organized code that is used to perform a task
3 fun main() {
4     sayHello()
5     sayWelcome()
6 }//end main
7
8 //We Create a Function, we call this function in our Main function
9 fun sayHello(){
10     println("Hello World!")
11 }//end sayHello
12
13 //Another Function and Call it on main Function
14 fun sayWelcome(){
15     println("You are Welcome")
16 }//end sayWelcome
17
```

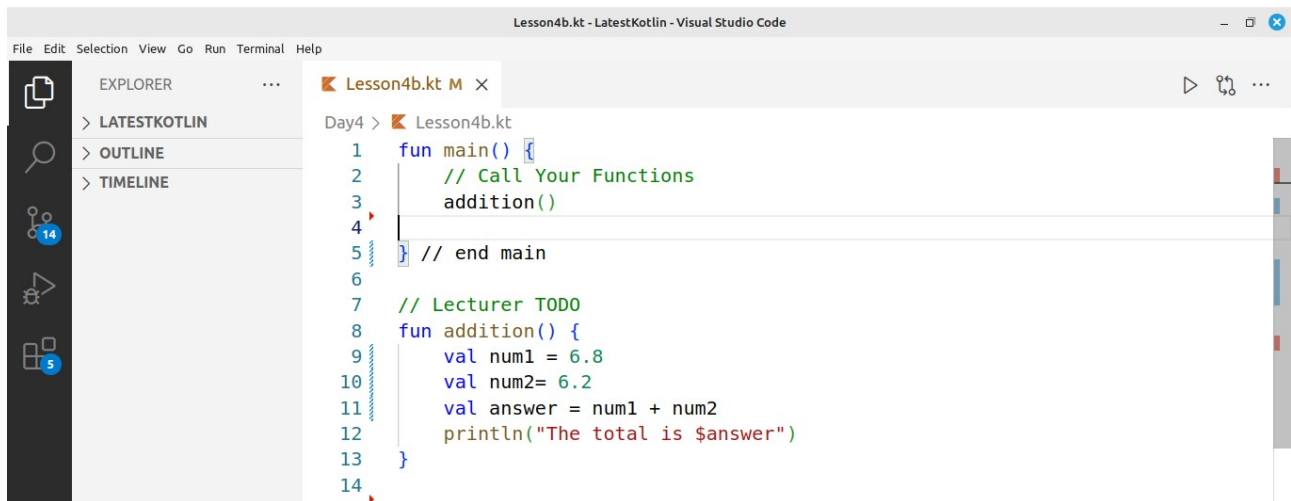
## Explanation

- **Main Function:** The main() function is where the program starts executing.
- **sayHello() Function:** Prints "Hello World!" when called.
- **sayWelcome() Function:** Prints "You are Welcome" when called.
- **Calling Functions:** Inside main(), we call two functions: sayHello() and sayWelcome().
- **Reusability:** Functions allow us to reuse code, making the program more organized and readable.

Next we create a function to add two numbers and call it in the main function.

## Example 2

Create a file named **Lesson4b.kt** and write below code.



```
1 fun main() {
2     // Call Your Functions
3     addition()
4 }
5 // end main
6
7 // Lecturer TODO
8 fun addition() {
9     val num1 = 6.8
10    val num2 = 6.2
11    val answer = num1 + num2
12    println("The total is $answer")
13 }
14
```

## Explanation

- **main Function:** The main() function calls the addition() function.
- **addition() Function:** Adds two numbers (6.8 and 6.2) and prints the result.
- num1 and num2 store the numbers to be added.
- answer stores the result of the addition.
- **Output:** The println() function prints the total (13.0).
- **Reusability:** The addition() function can be called anytime to perform the addition task.

## Assignment: Multiplication/Subtraction Function

- a) Create a function in Kotlin that multiplies two numbers and prints the answer.
- b) Create a function in Kotlin that subtracts two numbers and prints the answer.

## Summary

- A function is a block of code that performs a specific task.

- It is defined using the `fun` keyword followed by the function name.
- Functions are called by their name followed by parentheses.

## Functions with Parameters

Functions can have parameters, which allow them to accept input values and perform tasks based on those inputs. Parameters make functions more flexible and reusable because you can use the same function to work with different data.

### Syntax

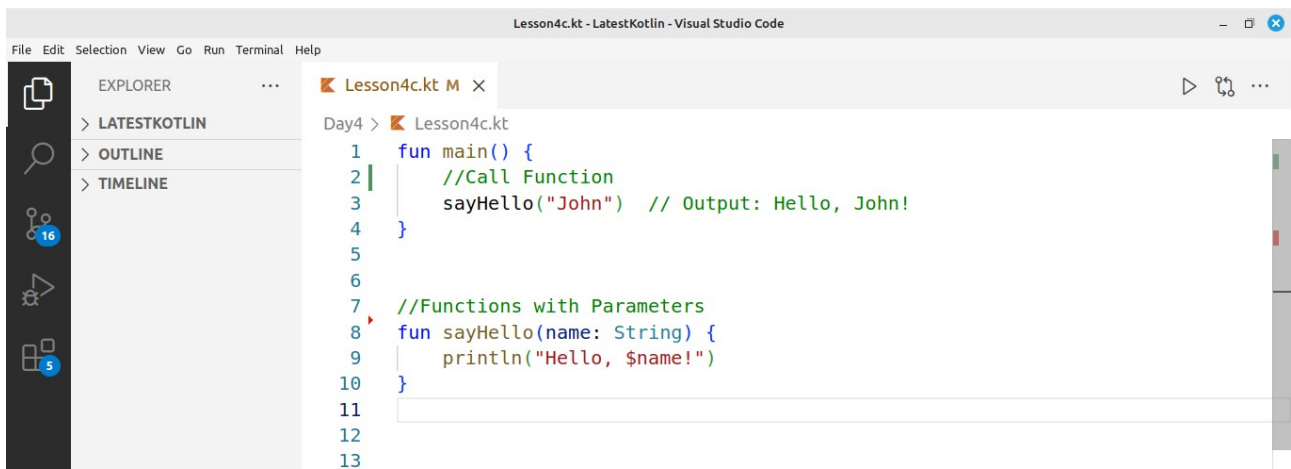
```
fun functionName(parameterName: ParameterType) {  
    // Function body: code to perform the task  
}
```

### Explanation

- **fun**: Keyword to define a function.
- **functionName**: The name of the function you create.
- **parameterName**: The name of the parameter inside the function.
- **ParameterType**: The type of the parameter (e.g., `String`, `Int`, `Double`).
- **Function body**: Code that defines what the function does using the provided parameter.

### Example 3

Create a File named **Lesson4c.kt** and write below code



```
1 fun main() {
2     //Call Function
3     sayHello("John") // Output: Hello, John!
4 }
5
6
7 //Functions with Parameters
8 fun sayHello(name: String) {
9     println("Hello, $name!")
10 }
11
12
13
```

### Explanation

#### sayHello() Function:

- Defined with a parameter name of type String.
- Prints a greeting message with the value passed to the name parameter.

#### main Function:

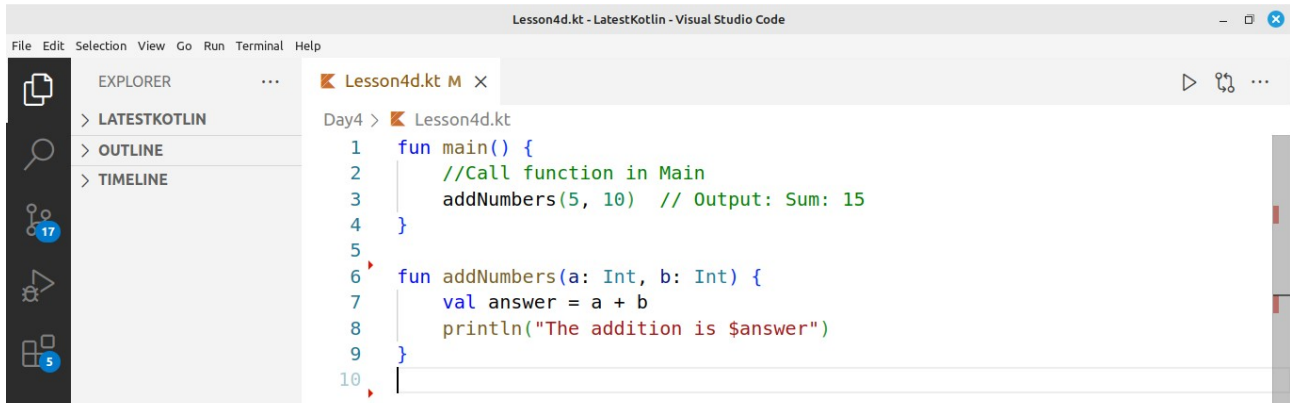
- Calls the sayHello() function, passing the argument "John".
- The output is: "Hello, John!".

#### Purpose:

- This function demonstrates how parameters make a function reusable by allowing it to work with different input values. In this case , we can call the function with different names and it works with new name/data provided.
- The function prints "Hello, John!" when called with "John" as the argument.

## Example 4

Create a File named **Lesson4c.kt** and write below code



```
Lesson4d.kt - LatestKotlin - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
> LATESTKOTLIN
> OUTLINE
> TIMELINE
Lesson4d.kt M x
Day4 > Lesson4d.kt
1 fun main() {
2     //Call function in Main
3     addNumbers(5, 10) // Output: Sum: 15
4 }
5
6 fun addNumbers(a: Int, b: Int) {
7     val answer = a + b
8     println("The addition is $answer")
9 }
10
```

## Explanation

- Defined with two parameters: a and b of type Int.
- Adds the two numbers and stores the result in the variable answer.
- Calls the addNumbers() function, passing 5 and 10 as arguments.
- Prints the result with a message: "The addition is \$answer".
- The output will be: "The addition is 15".

## Assignment

- a) Students to do multiplication of 2 numbers.
- b) Students to do subtraction of 2 numbers.

## Summary

- A function is a block of code that performs a specific task.
- Use **fun** keyword to create a Function.

- Use **function name** to call the function
- Functions can accept parameters to work with different data.
- You call a function by using its name followed by parentheses.
- Functions help in reusing code, making the program more organized.
- They improve code readability and maintainability by breaking down tasks.

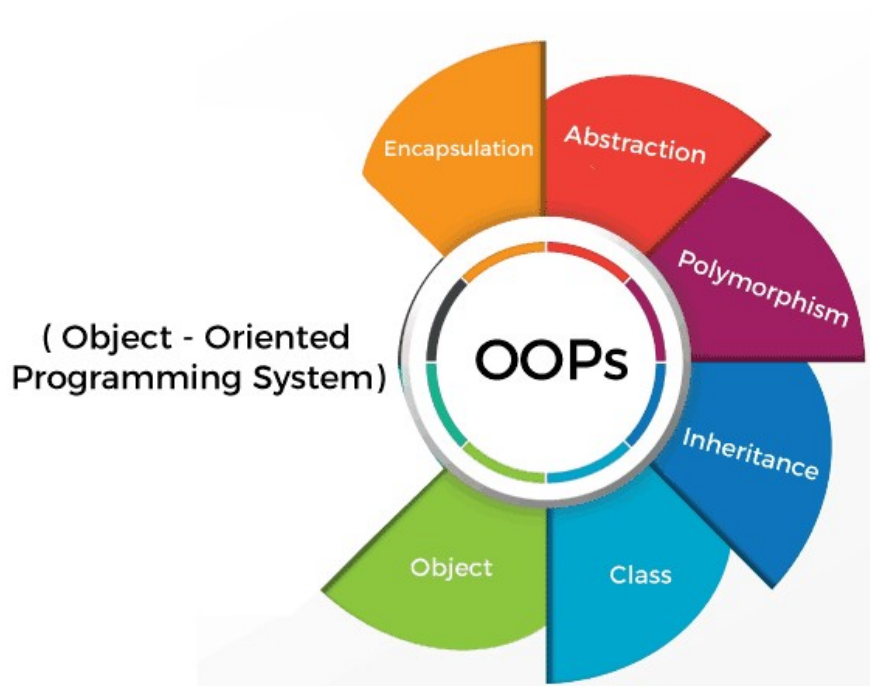


## Chapter 5: Introduction to Object-Oriented Programming (OOP)

### What is OOP?

- Object-Oriented Programming (OOP) is a way to organize code using **objects**.

Object oriented has different categories as shown in below image.



### Advantages of Object-Oriented Programming (OOP)

#### 1. Modularity:

- OOP organizes code into classes and objects, making it easier to manage and update parts of the program without affecting others.

#### 2. Reusability:

- Once a class is written, it can be reused in other programs. This reduces redundancy and saves time.

#### 3. Scalability:

- OOP makes it easier to add new features to a program by creating new classes or modifying existing ones without disrupting the entire system.

#### 4. Maintainability:

- Since code is organized into small, self-contained units (objects), it's easier to maintain, debug, and improve over time.

### Objects in OOP

- Think of an **object** as a real-world thing, like a dog. **A Dog, Car, Person etc are Objects**
- **Objects** have two key parts, Using example of a Person Object
  - **State:** The characteristics or attributes of the object (e.g., color, name, age).
  - **Behavior:** The actions the object can perform or the things it can do (e.g., speak(), walk(), run()).

### Relate states and behaviors to Programming;

- **State** refers to **properties/attributes** (or attributes) of an object. These are the characteristics that define the object, like age, name, color, height etc
- **Behavior** refers to **functions** (or methods) of an object. These are the actions that the object can perform, like running(), walking(), or eating().

### Classes in OOP

In Kotlin **Object-Oriented Programming (OOP)**, a class is a blueprint/design for creating objects. It defines properties (variables) and functions that the objects will have.

#### Syntax

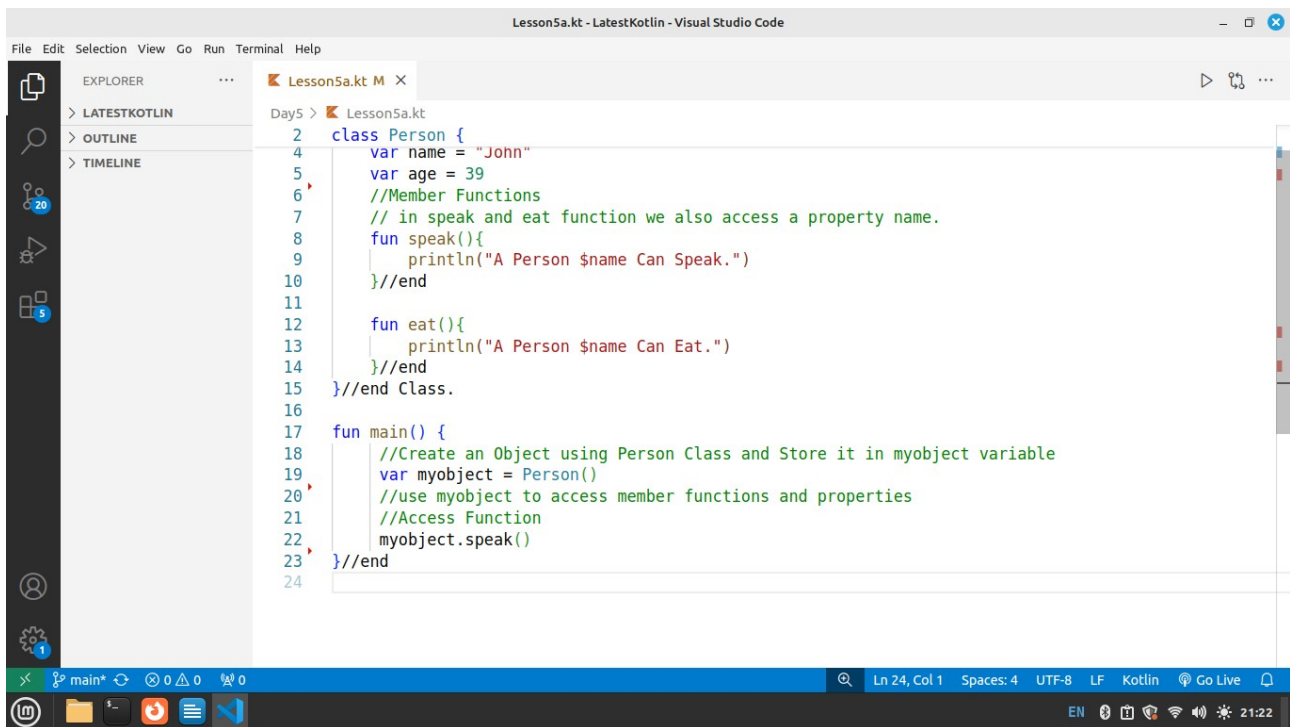
```
class Person {
    // Attributes (properties) go here
    // var name: String = ""
    // var age: Int = 0

    // Functions (methods) go here
    // fun speak() {
    //     println("Hello, I am a person!")
    // }
}
```

The **class keyword** tells the compiler that we want to create a class. From above, we see that a class contains **states/attributes/properties** and member **functions/methods**

### Example 1

Create a File named **Lesson5a.kt** and write below Code.



```
Lesson5a.kt - LatestKotlin - Visual Studio Code
File Edit Selection View Go Run Terminal Help

EXPLORER
> LATESTKOTLIN
> OUTLINE
> TIMELINE

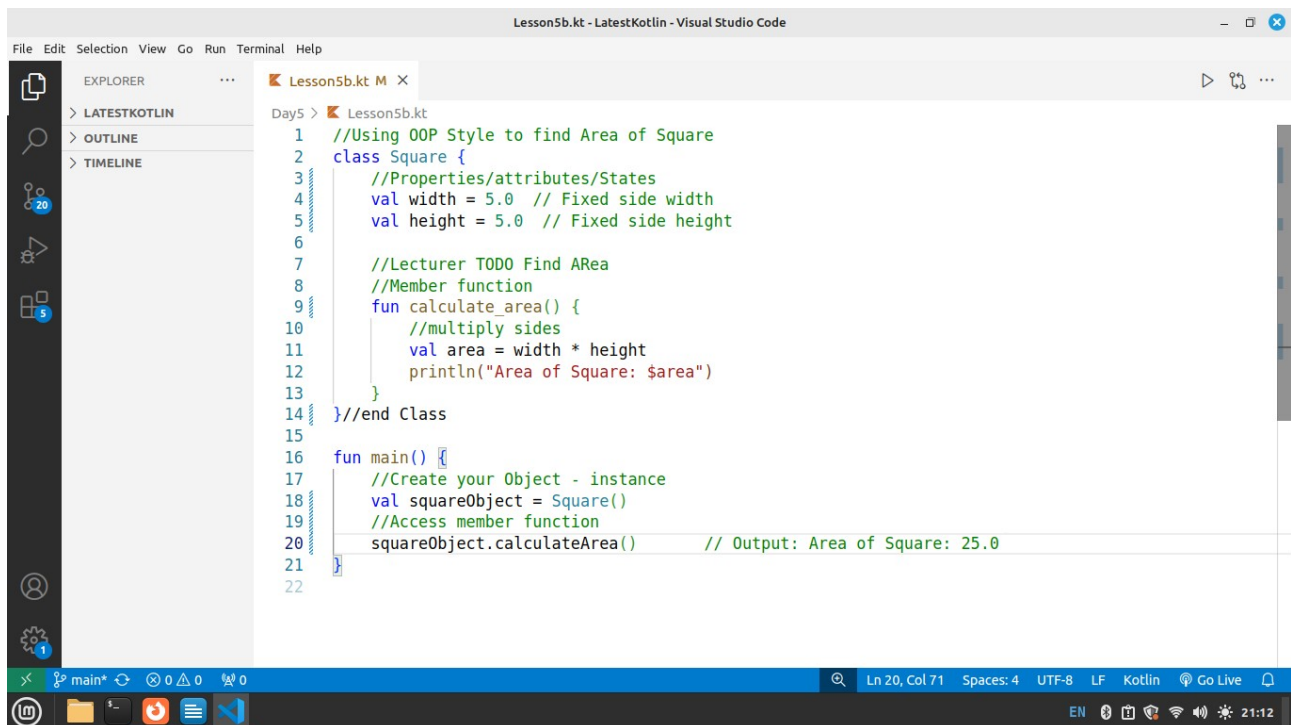
Day5 > Lesson5a.kt
2 class Person {
4     var name = "John"
5     var age = 39
6     //Member Functions
7     // in speak and eat function we also access a property name.
8     fun speak(){
9         println("A Person $name Can Speak.")
10    }//end
11
12    fun eat(){
13        println("A Person $name Can Eat.")
14    }//end
15 }//end Class.
16
17 fun main() {
18     //Create an Object using Person Class and Store it in myobject variable
19     var myobject = Person()
20     //use myobject to access member functions and properties
21     //Access Function
22     myobject.speak()
23 }//end
24
```

### Explanation

- A **Person class** is created with **attributes/properties** and **methods/functions**
- **name**: A string property initialized with **"John"**.
- **age**: An integer property initialized with **39**.
- **speak()**: Prints a message that the person can **speak**.
- **eat()**: Prints a message that the person can **eat**.
- The **main()** function creates an object of the Person class (**myobject**).
- Calls the **speak()** function using **myobject**.

## Example 2

In the next example we use OOP design to find the area of Square. Create a File named **Lesson5b.kt** and write below code.



```
1 //Using OOP Style to find Area of Square
2 class Square {
3     //Properties/attributes/States
4     val width = 5.0 // Fixed side width
5     val height = 5.0 // Fixed side height
6
7     //Lecturer TODO Find ARea
8     //Member function
9     fun calculate_area() {
10         //multiply sides
11         val area = width * height
12         println("Area of Square: $area")
13     }
14 } //end Class
15
16 fun main() {
17     //Create your Object - instance
18     val squareObject = Square()
19     //Access member function
20     squareObject.calculateArea() // Output: Area of Square: 25.0
21 }
22
```

## Explanation

- A **Square class** is created with properties and a function/method.
- **width**: A fixed property with a value of 5.0.
- **height**: A fixed property with a value of 5.0.
- **calculate\_area()**: A function/method that calculates the area by multiplying width and height, then prints the result.
- The **main() function** creates an object of the Square class (**squareObject**).
- Calls the **calculate\_area()** function using **squareObject**.

### Student Assignment:

Define **calculate\_perimeter()** inside the Square class to correctly calculate and print the perimeter of the square.

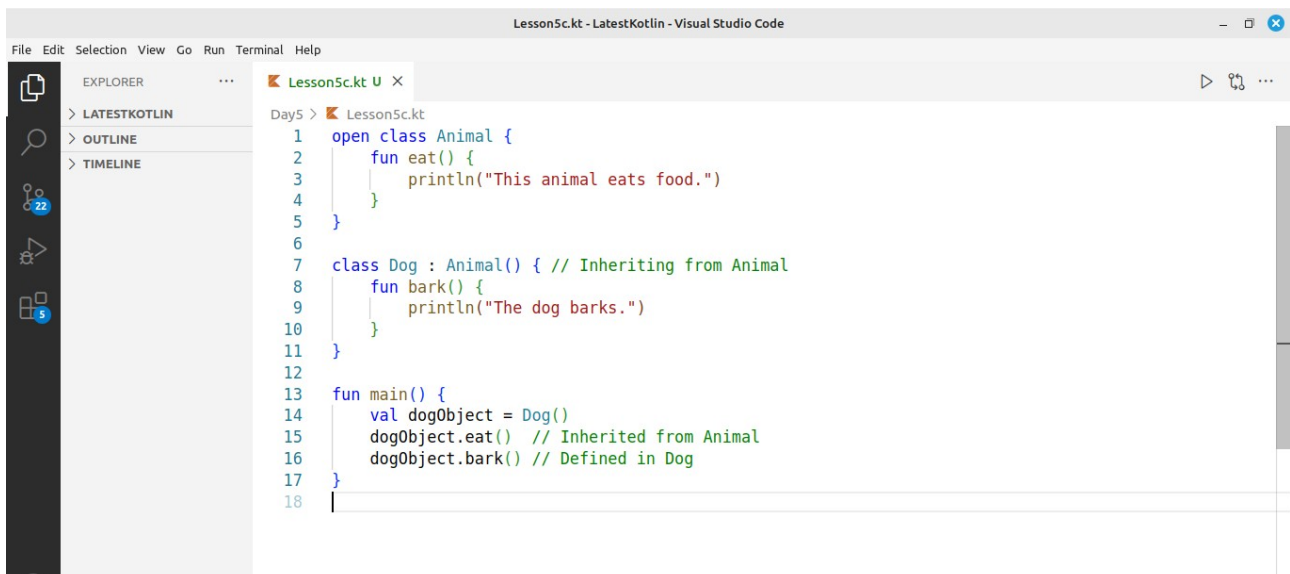
Todo: Add a Function/method to **calculate\_perimeter()** of a Square, Call this function **calculate\_perimeter()** in your main class using the **squareObject**.

### Inheritance in OOP

Inheritance is when one class (child class) gets the properties and functions of another class (parent class).

### Example 3

Create a File named **Lesson5c.kt** and write below code.



```
Lesson5c.kt - LatestKotlin - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
> LATESTKOTLIN
> OUTLINE
> TIMELINE
Lesson5c.kt
Day5 > Lesson5c.kt
1 open class Animal {
2     fun eat() {
3         println("This animal eats food.")
4     }
5 }
6
7 class Dog : Animal() { // Inheriting from Animal
8     fun bark() {
9         println("The dog barks.")
10    }
11 }
12
13 fun main() {
14     val dogObject = Dog()
15     dogObject.eat() // Inherited from Animal
16     dogObject.bark() // Defined in Dog
17 }
18
```

This is **inheritance**—the **Dog** gets everything from **Animal**.

## Explanation

- **Animal** class is created with a function **eat()**.
- **Dog** class **inherits** from **Animal** using **: Animal()**.
- **Dog** gets access to **eat()** from **Animal**.
- **Dog** has its own function **bark()**.
- In **main()**, an object of **Dog** (**dogObject**) is created.
- **dogObject.eat()** works because **Dog** inherits it from **Animal**.
- **dogObject.bark()** works because it is defined in **Dog**.

## Chapter 6: Introduction to try, catch, and Exceptions:

In programming, **exceptions** are errors that occur during the execution of a program.

When an error happens, it causes the program to stop, unless we handle it properly.

In Kotlin, we can handle exceptions using the try and catch blocks. The try block contains the code that might cause an error, and the catch block catches and handles that error.

### Key Points:

- **Exception:** An error that occurs during program execution (e.g., dividing by zero, accessing a null value).
- **try block:** Contains code that might cause an error.
- **catch block:** Catches and handles the exception if an error occurs in the try block.

### Syntax

**try** {

*// Code that might throw an exception*

} **catch** (error: ExceptionType) {

*// Code to handle the exception*

}

### Example 1



**Explanation:**

1. **try block:** The division  $10 / 0$  is inside the try block. This will throw an Exception.
2. **catch block:** The catch block catches the exception and prints a message instead of the program crashing.
3. **Error Handling:** Without try and catch, the program would stop executing when an error occurs. But with exception handling, we can control what happens when an error happens.



## Appendix A

### Reserved Keywords

These are reserved keywords in Kotlin, and they can't be used as identifiers (variable names, class names, etc.). They serve specific purposes in the language syntax.

Keyword	Description
abstract	Defines an abstract class or function
assert	Used for debugging to assert that a condition is true
break	Exits a loop or a when expression
class	Defines a class
continue	Skips the current iteration of a loop
do	Starts a do-while loop
else	Defines the else part of an if expression
enum	Defines an enum class
false	Represents a boolean false value
for	Defines a for loop
fun	Defines a function
if	Starts an if expression
in	Used for ranges or checking if an item is in a collection
interface	Defines an interface
is	Used to check the type of a variable

null	Represents a null value
object	Defines a singleton or an object declaration
package	Defines the package for a class or function
return	Exits a function and optionally returns a value
super	Refers to the superclass of a class
this	Refers to the current instance of a class
throw	Used to throw exceptions
true	Represents a boolean true value
try	Starts a try-catch block for exception handling
val	Defines a read-only (immutable) variable
var	Defines a mutable variable
when	Defines a when expression (similar to switch)
while	Starts a while loop

## Revision Questions

- 1) What does the main function do in kotlin
- 2) What is the difference between val and var in Kotlin?
- 3) How do you declare a variable in Kotlin
- 4) List any 5 Data types in Kotlin
- 5) Name 5 Arithmetic Operators
- 6) Name any 5 Relational/Comparison Operators
- 7) What is an array in Kotlin.
- 8) How do you declare a function in Kotlin? Provide an example.
- 9) What is the purpose of the fun keyword in Kotlin?
- 10) What is a when expression in Kotlin, and how is it different from a switch statement in other languages?
- 11) What is the purpose of an if statement in Kotlin?
- 12) How do you write an if statement to check whether a number is greater than 10 in Kotlin?
- 13) What is the basic syntax of a for loop in Kotlin?
- 14) How does a while loop work in Kotlin?
- 15) What do you understand by the term Object Oriented
- 16) What is the purpose of a class in Kotlin?
- 17) What is a member function in a Class
- 18) What is inheritance in OOP

- 19) What are exceptions in Kotlin
- 20) Why do we need try and catch in Kotlin
- 21) Declare a variable of type String called greeting and assign it the value "Hello, Kotlin!". Then print the variable.
- 22) Write a Kotlin program that checks if a given number is even or odd using an if-else statement.
- 23) Write a Kotlin function multiply() that takes two integers as parameters and returns their product.
- 24) Write a Kotlin program that tries to access an element outside the bounds of an array and handles the exception using try-catch
- 25) Write a Kotlin program that finds the largest number in an array of integers.
- 26) Write a Kotlin program that prints the numbers from 1 to 10 using a for loop.
- 27) Write a Kotlin program that prints numbers from 10 to 1 in reverse order using a while loop.

## References

<https://kotlinlang.org/docs/getting-started.html>

<https://www.w3schools.com/KOTLIN/index.php>

<https://developer.android.com/courses/pathways/android-development-with-kotlin-1>

<https://www.tutorialspoint.com/kotlin/index.htm>