**Week 1: Kotlin Basic**

There are a lot of useful on-line resources where you can learn Kotlin quickly. Here, we recommend two approaches. You can visit <https://www.w3schools.com/kotlin/index.php> to learn the Kotlin basic or visit <https://play.kotlinlang.org/byExample/overview> to learn Kotlin by examples.

Note that for either approach, the efficient study is to learn the basic knowledge of Kotlin only such as how to define variables, how to create a function, how to create a class and how to use a function in a class and so on. We can further practice Kotlin programming skills from the later Android app programming. We’ll use the problem-solving driven approach rather than the full studying of Kotlin with details.

For running the Kotlin programs, we recommend that you try the on-line play from <https://kotlinlang.org/> and click “Play” from the right-top corner. This one is much quicker than the compiler in w3schools. However, the on-line compiler does not support user inputs, and therefore, you must use an IDE or VSCode to accept user inputs.

Here, we give few examples. If you can understand them, you are ready to finish most Android apps required by this module.

1. Data types in Kotlin

Please use the following example to refresh the data types in Kotlin.

fun main() {

    // Integer data types

    val byteValue: Byte = 42

    val shortValue: Short = 100

    val intValue: Int = 1000

    val longValue: Long = 1000000000L // Note the 'L' suffix for Long literals

    // Floating-point data types

    val floatValue: Float = 3.14f // Note the 'f' suffix for Float literals

    val doubleValue: Double = 2.71828

    // Boolean data type

    val isKotlinAwesome: Boolean = true

    // Character data type

    val grade: Char = 'A'

    // String data type

    val greeting: String = "Hello, Kotlin!"

    // Arrays

    val numbers: IntArray = intArrayOf(1, 2, 3, 4, 5)

    val fruits: Array<String> = arrayOf("Apple", "Banana", "Cherry")

    // Nullable data types

    val nullableValue: String? = null

    // Printing values

    println("Byte: $byteValue")

    println("Short: $shortValue")

    println("Int: $intValue")

    println("Long: $longValue")

    println("Float: $floatValue")

    println("Double: $doubleValue")

    println("Boolean: $isKotlinAwesome")

    println("Char: $grade")

    println("String: $greeting")

    println("IntArray: ${numbers.joinToString()}")

    println("Array<String>: ${fruits.joinToString()}")

    println("Nullable Value: $nullableValue")

}

Note that the data types can be implicit for most cases and Kotlin is smart enough to match the data type with the provided value.

1. A simple Calendar

We develop a simple program to translate days of the month such as from 1st to 31st into the days of the week date such as Monday to Sunday. You need to define the first day of the month as which day of the week. You can create a list of days of the month as the testing data.

You can finish the simple task yourself, or you can consider the following solution.

fun main() {

    val firstday = 1 // 1 is Monday, 2 is Tuesday and so on

    val dates = listOf(2, 5, 32, 30) // the testing data

    for (date in dates){

        if(date<0 || date>31){

            println("$date is an invalid date!")

        }else{

            var day = date%7 + firstday-1 // the week day

            when (day) {

                1 -> println("$date is Monday!")

                2 -> println("$date is Tuesday!")

                3 -> println("$date is Wednesday!")

                4 -> println("$date is Tursday!")

                5 -> println("$date is Friday!")

                6 -> println("$date is Saturday!")

                7 -> println("$date is Sunday!")

            }

        }

    }

}

The major practice of the example is about the control flow such as if else, for loop, while and when. Note that “listOf” is immutable. If you want to change, add or remove the elements of a list, “mutableListOf” should be used. Please check the difference from the following example.

fun main() {

    // Create a mutable list of integers

    val mutableList = mutableListOf(1, 2, 3, 4, 5)

    // Add an element to the mutable list

    mutableList.add(6)

    // Modify an element in the list

    mutableList[0] = 10

    // Remove an element from the list

    mutableList.removeAt(3)

    // Print the modified list

    println("Modified List: $mutableList")

}

1. The bookstore example

This exercise is about the class and its instantiations.

data class Book(val title: String, val author: String, val price: Double)

class Bookstore {

    private val books = listOf(

        Book("The Great Gatsby", "F. Scott Fitzgerald", 9.99),

        Book("To Kill a Mockingbird", "Harper Lee", 8.99),

        Book("1984", "George Orwell", 7.99),

        Book("Pride and Prejudice", "Jane Austen", 10.99)

    )

    fun listBooks() {

        println("Welcome to the Online Bookstore!")

        println("Available Books:")

        books.forEachIndexed { index, book ->

            println("${index + 1}. Title: ${book.title}")

            println("   Author: ${book.author}")

            println("   Price: $${book.price}")

            println()

        }

    }

}

fun main() {

    val bookstore = Bookstore()

    bookstore.listBooks()

}

Here, **data class** in Kotlin is a powerful and concise way to define classes that primarily exist to hold and manage data. It simplifies your code, improves readability, and reduces the potential for errors when working with data objects. You can refer to <https://kotlinlang.org/docs/data-classes.html> for more information.

To create a list in Kotlin, you should use listOf with the elements. So, **books** is a private variable of the class and the function **listBooks** simply outputs each book from **books**. Here, **forEachIndexed** is an in-line function to get the index and the value sequentially. The arrow symbol ‘->’ defines a lambda expression.

For instance, the following example demonstrates how to create a lambda expression.

fun main() {

    // Define a lambda expression to calculate the sum of two numbers

    val sum = { x: Int, y: Int -> x + y }

    // Example usage of the sum lambda

    val result1 = sum(5, 3)

    val result2 = sum(10, 20)

    // Display the results

    println("Result 1: $result1") // Output: Result 1: 8

    println("Result 2: $result2") // Output: Result 2: 30

}

In this Kotlin program, We define a lambda expression using the syntax **val sum = { x: Int, y: Int -> x + y }**. This lambda takes two integer parameters x and y and calculates their sum x + y. We then use the sum lambda to calculate the sum of two sets of numbers and store the results in result1 and result2. Finally, we print the results, demonstrating how the lambda expression is used to perform the addition operation.