**WHEN STATEMENT:**

val day = 4

val result = when (day) {

1 -> "Monday"

2 -> "Tuesday"

3 -> "Wednesday"

4 -> "Thursday"

5 -> "Friday"

6 -> "Saturday"

7 -> "Sunday"

else -> "Invalid day."

}

println(result)

**ARRAYS**

val cars = arrayOf("Volvo", "BMW", "Ford", "Mazda")

println(cars.size)

**Unlike Java and other programming languages, there is no traditional for loop in Kotlin.**

for (chars in 'a'..'x') {

println(chars)

}

for (nums in 5..15) {

println(nums)

}

**FUNCTIONS**

fun myFunction(x: Int): Int {

return (x + 5)

}

fun main() {

var result = myFunction(3)

println(result)

}

fun myFunction(x: Int, y: Int) = x + y

**Class**

class Car {

var brand = ""

var model = ""

var year = 0

}

// Create a c1 object of the Car class

val c1 = Car()

// Access the properties and add some values to it

c1.brand = "Ford"

c1.model = "Mustang"

c1.year = 1969

println(c1.brand) // Outputs Ford

println(c1.model) // Outputs Mustang

println(c1.year) // Outputs 1969

**class Car** {

var brand = ""

var model = ""

var year = 0

}

fun main() {

val c1 = Car()

c1.brand = "Ford"

c1.model = "Mustang"

c1.year = 1969

}

class Car(var brand: String, var model: String, var year: Int)

fun main() {

val c1 = Car("Ford", "Mustang", 1969)

val c2 = Car("BMW", "X5", 1999)

val c3 = Car("Tesla", "Model S", 2020)

}

class Car(var brand: String, var model: String, var year: Int) {

// Class function

fun drive() {

println("Wrooom!")

}

}

fun main() {

val c1 = Car("Ford", "Mustang", 1969)

// Call the function

c1.drive()

}

**SINGLE INHERITANCE**

open class Parent {

fun parentFunction() {

println("Parent function")

}

}

class Child : Parent() {

fun childFunction() {

println("Child function")

}

}

**MULTIPLE INHERITANCE**

interface InterfaceA {

fun functionA() {

println("Function A")

}

}

interface InterfaceB {

fun functionB() {

println("Function B")

}

}

class CombinedClass : InterfaceA, InterfaceB {

// Can use both functions from InterfaceA and InterfaceB

}

fun main() {

val combined = CombinedClass()

combined.functionA()

combined.functionB()

**}**

**Handling Conflicts in Multiple Inheritance (Interfaces):**

interface InterfaceA {

fun show() {

println("InterfaceA show")

}

}

interface InterfaceB {

fun show() {

println("InterfaceB show")

}

}

class CombinedClass : InterfaceA, InterfaceB {

// Must override and choose which implementation to use

override fun show() {

super<InterfaceA>.show() // or super<InterfaceB>.show()

}

}

fun main() {

val combined = CombinedClass()

combined.show() // Output: InterfaceA show

}