1. A program to calculate the factorial of a given number.

```
fun main() {
    val number = 5
    var factorial = 1

    for (i in 1..number) {
        factorial *= i
    }

    println("Factorial of $number is: $factorial")
}
```

```
Factorial of 5 is: 120
```

2. A program that uses map and filter functions on a list of strings.

```
fun main() {
    val stringList = listOf("apple", "banana", "cherry", "date", "elderberry")

    // Example 1: Using map to convert each string to uppercase
    val uppercaseList = stringList.map { it.toUpperCase() }
    println("Uppercase List: $uppercaseList")

    // Example 2: Using filter to find strings starting with 'b'
    val filteredList = stringList.filter { it.startsWith('b') }
    println("Filtered List: $filteredList")
}
```

```
Uppercase List: [APPLE, BANANA, CHERRY, DATE, ELDERBERRY]
Filtered List: [banana]
```

3. A program that checks if a user-inputted string is a palindrome.

```
fun main() {
    print("Enter a string : ")
    val str = readLine()!!

    if (str == str.reversed()) {
        print("$str is palindrome.")
    } else {
        print("$str is not palindrome.")
    }
}
```

```
TestKt (1)
"C:\Program Files\Android\Android Studio\jre\bin\java.exe"
Enter a string : viv
viv is palindrome.
Process finished with exit code 0
```

4. An extension function for the List<Int> class that calculates and returns the average of the list elements.

Here's an extension function for the `List<Int>` class in Kotlin that calculates and returns the average of the list elements:

```
fun List<Int>.average(): Double {
    if (isEmpty()) return 0.0 // Return 0 if the list is empty
    val sum = sum()
    return sum.toDouble() / size
}
```

Here's an example usage of the `average()` extension function:

```
fun main() {
    val numbers = listOf(1, 2, 3, 4, 5)
    val average = numbers.average()
    println("Average: $average")
}
```

Sum Comman Example:

```
fun main() {
    val numbers = listOf(6, 42, 10, 4)

    println("Count: ${numbers.count()}")
    println("Max: ${numbers.maxOrNull()}")
    println("Min: ${numbers.minOrNull()}")
    println("Average: ${numbers.average()}")
    println("Sum: ${numbers.sum()}")
}
```

```
Count: 4
Max: 42
Min: 4
Average: 15.5
Sum: 62
```

5. A program that demonstrates the use of let, with, run, apply, and also functions in Kotlin.

```
data class Person(var name: String, var age: Int)

fun main() {
    // let function
    val person1 = Person("John", 30)
    person1.let {
        it.age += 1
        println("Person 1: $it")
    }

    // with function
    val person2 = Person("Alice", 25)
    with(person2) {
        age += 2
        println("Person 2: $this")
    }

    // run function
    val person3 = Person("Mike", 35)
    val updatedPerson3 = person3.run {
        age += 3
        this
    }
    println("Person 3: $updatedPerson3")

    // apply function
    val person4 = Person("Emily", 28).apply {
        age += 4
    }
    println("Person 4: $person4")

    // also function
    val person5 = Person("David", 32)
    val updatedPerson5 = person5.also {
        it.age += 5
    }
    println("Person 5: $updatedPerson5")
}
```

```
Person 1: Person(name=John, age=31)
Person 2: Person(name=Alice, age=27)
Person 3: Person(name=Mike, age=38)
Person 4: Person(name=Emily, age=32)
Person 5: Person(name=David, age=37)
```

6. A program that implements a Student class with properties: name, age, and grades.

```
class Student(val name: String, val age: Int, val grades: List<Double>) {
    fun averageGrade(): Double {
        if (grades.isEmpty()) return 0.0
        val sum = grades.sum()
        return sum / grades.size
    }
}

fun main() {
    val student = Student("John Doe", 18, listOf(85.5, 90.0, 76.5, 92.0, 88.5))
    println("Name: ${student.name}")
    println("Age: ${student.age}")
    println("Grades: ${student.grades}")
    println("Average Grade: ${student.averageGrade()}")
}
```

```
Name: John Doe
Age: 18
Grades: [85.5, 90.0, 76.5, 92.0, 88.5]
Average Grade: 86.5
```

7. A program that converts the Student class into a data class.

In Kotlin, the `data class` keyword is used to create a class that is primarily used to hold data. It automatically generates useful methods such as `toString()`, `equals()`, `hashCode()`, and `copy()`. To convert the `Student` class into a `data class`, you can follow these steps:

data class Student(val name: String, val age: Int, val grade: String)

By adding the `data` modifier in front of the class declaration, you convert the class into a data class. In this example, the `Student` class has three properties: `name`, `age`, and `grade`. The `data class` will generate the default implementations for the methods mentioned above based on the properties.

Here's an example of how you can use the `Student` data class:

```
data class Student(val name: String, val age: Int, val grade: String)

fun main() {
    val student1 = Student("John Doe", 18, "A")
    val student2 = Student("Jane Smith", 17, "B")

    println(student1) // Output: Student(name=John Doe, age=18, grade=A)
    println(student2) // Output: Student(name=Jane Smith, age=17, grade=B)

    val copiedStudent = student1.copy(age = 19)
    println(copiedStudent) // Output: Student(name=John Doe, age=19, grade=A)

    println(student1 == student2) // Output: false
    println(student1 == copiedStudent) // Output: true
}
```

```
Student(name=John Doe, age=18, grade=A)
Student(name=Jane Smith, age=17, grade=B)
Student(name=John Doe, age=19, grade=A)
false
false
```

In this example, we create two instances of the `Student` data class and print them using the `toString()` method generated by the data class. We also demonstrate the `copy()` method, which allows us to create a new instance of the data class with modified properties. Finally, we compare instances using the `equals()` method generated by the data class.

8. A program that uses lambda expressions and higher-order functions to find the maximum value in a list of integers.

```
fun main() {
    val numbers = listOf(10, 20, 5, 15, 30, 25)
    val maxNumber = numbers.maxByOrNull { it }
    println("Maximum number: $maxNumber")
}
```

In this program, we have a list of integers called `numbers`. The `maxByOrNull` function is a higher-order function that takes a lambda expression as an argument. The lambda expression `{ it }` is a shorthand notation for a lambda that takes a single parameter and returns its value. In this case, `it` represents each element in the list, and the lambda expression simply returns the element itself.

The `maxByOrNull` function iterates over each element in the list and finds the maximum element based on the values returned by the lambda expression. Finally, we print the maximum number using the `println` function.

```
Maximum number: 30
```

This indicates that the maximum value in the list is 30.

Note that `maxByOrNull` returns `null` if the list is empty. If you want to handle that case, you can use the `?.` operator to safely access the result:

```
val maxNumber = numbers.maxByOrNull { it }
println("Maximum number: ${maxNumber?.toString() ?: "No maximum value found"}")
```

In this case, if `maxNumber` is `null`, the program will print "No maximum value found" instead of trying to call `toString()` on a `null` value.

9. A program that implements a Shape interface with two classes, Circle and Rectangle, that implement the Shape interface.

```
// Shape interface
interface Shape {
    fun area(): Double
}

// Circle class implementing the Shape interface
```

```
class Circle(private val radius: Double) : Shape {
    override fun area(): Double {
        return Math.PI * radius * radius
    }
}

// Rectangle class implementing the Shape interface
class Rectangle(private val width: Double, private val height: Double) : Shape {
    override fun area(): Double {
        return width * height
    }
}

// Main function to test the classes
fun main() {
    val circle = Circle(5.0)
    println("Circle Area: ${circle.area()}")

    val rectangle = Rectangle(4.0, 6.0)
    println("Rectangle Area: ${rectangle.area()}")
}
```

```
Circle Area: 78.53981633974483
Rectangle Area: 24.0
```

10. A program to read a text file, count the frequency of each word, and save the results in a new text file.

```
import java.io.File

fun main() {
    // Path to input text file
    val inputFile =
"C:\\Users\\Admin\\AndroidStudioProjects\\MyApplication2\\app\\src\\main\\java\\com\\example\
\myapplication\\input.txt"

    // Path to output text file
    val outputFile =
"C:\\Users\\Admin\\AndroidStudioProjects\\MyApplication2\\app\\src\\main\\java\\com\\example\
\myapplication\\output.txt"

    // Read input file
    val text = File(inputFile).readText()

    // Split text into words
    val words = text.split("\\s+".toRegex())

    // Count the frequency of each word
    val wordCountMap = mutableMapOf<String, Int>()
    for (word in words) {
        val count = wordCountMap.getOrDefault(word, 0)
        wordCountMap[word] = count + 1
```

```
    }

    // Create output file and write word frequencies
    File(outputFile).printWriter().use { writer ->
        for ((word, count) in wordCountMap) {
            writer.println("$word: $count")
        }
    }

    println("Word frequencies saved to $outputFile.")
}
```

Input.txt

```
Hello world! This is a sample text file.
Hello Kotlin. Kotlin is awesome.
```

Output.txt

test.kt × output.txt × input.txt ×

```
Hello: 2
world!: 1
This: 1
is: 2
a: 1
sample: 1
text: 1
file.: 1
Kotlin.: 1
Kotlin: 1
awesome.: 1
: 1
```

11. A program that demonstrates the use of coroutines to fetch data from two different web APIs concurrently.

```
import kotlinx.coroutines.async
import kotlinx.coroutines.delay
import kotlinx.coroutines.runBlocking

suspend fun fetchDataFromApi1(): String {
    delay(2000) // Simulating API latency
    return "Data from API 1"
}
```

```
suspend fun fetchDataFromApi2(): String {
    delay(3000) // Simulating API latency
    return "Data from API 2"
}

fun main() = runBlocking {
    println("Fetching data from APIs...")

    val dataFromApi1 = async { fetchDataFromApi1() }
    val dataFromApi2 = async { fetchDataFromApi2() }

    println("Data received:")
    println("API 1: ${dataFromApi1.await()}")
    println("API 2: ${dataFromApi2.await()}")
}
```

```
Fetching data from APIs...
Data received:
API 1: Data from API 1
API 2: Data from API 2
```

13. A program to find the GCD of two given numbers using the Euclidean algorithm.

Here's a program in Kotlin that uses the Euclidean algorithm to find the greatest common divisor (GCD) of two given numbers

```
fun main() {
    val num1 = 36
    val num2 = 48

    val gcd = findGCD(num1, num2)
    println("The GCD of $num1 and $num2 is: $gcd")
}

fun findGCD(a: Int, b: Int): Int {
    var num1 = a
    var num2 = b

    while (num2 != 0) {
        val temp = num2
        num2 = num1 % num2
        num1 = temp
    }

    return num1
}
```

Here's a simplified version of the program using a recursive function to find the GCD:

```
fun main() {
    val num1 = 36
```

```
    val num2 = 48

    val gcd = findGCD(num1, num2)
    println("The GCD of $num1 and $num2 is: $gcd")
}

fun findGCD(a: Int, b: Int): Int {
    if (b == 0) {
        return a
    }

    return findGCD(b, a % b)
}
```

```
The GCD of 36 and 48 is: 12
```

14. An extension function for the String class that returns the number of vowels in the string.

```
fun String.countVowels(): Int {
    val vowels = setOf('a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U')
    var count = 0

    for (char in this) {
        if (char in vowels) {
            count++
        }
    }

    return count
}
fun main() {
    val sentence = "Hello, Worldi!"
    val vowelCount = sentence.countVowels()
    println("Number of vowels in the sentence: $vowelCount")
}
```

```
Number of vowels in the sentence: 4
```

15. A program to generate a list of the first n Fibonacci numbers, where n is a user input.

```
import java.util.*

fun main() {

    val read = Scanner(System.`in`)
    println("Enter number of terms:")
    val number = read.nextInt()

    var t1 = 0
    var t2 = 1
```

```
    println("\nFibonacci series of first $number terms: ")
    for (index in 1..number) {
        print("$t1  ")

        val sum = t1 + t2
        t1 = t2
        t2 = sum
    }

}
```

```
TestKt (1)
"C:\Program Files\Android\Android Studio\jre\bin\java.exe" ...
Enter number of terms:
30

Fibonacci series of first 30 terms:
0  1  1  2  3  5  8  13  21  34  55  89  144  233  377  610  987  1597  2584  4181  6765  10946  17711  28657  46368  75025  121393  196418  317811  514229
```

16. A function that takes two integer arrays as arguments and returns a new array that contains the intersection of the two input arrays.

```
fun findIntersection(array1: IntArray, array2: IntArray): IntArray {
    val set1 = array1.toSet()
    val set2 = array2.toSet()
    val intersection = set1.intersect(set2)
    return intersection.toIntArray()
}

fun main() {
    val array1 = intArrayOf(1, 2, 3, 4, 5)
    val array2 = intArrayOf(4, 5, 6, 7, 8)
    val intersection = findIntersection(array1, array2)
    println(intersection.contentToString()) // Output: [4, 5]
}
```

```
[4, 5]
```

17. A program that creates a Person class with properties: firstName, lastName, and age.

```
class Person(val firstName: String, val lastName: String, val age: Int)

fun main() {
    val person = Person("John", "Doe", 25)

    println("First Name: ${person.firstName}")
    println("Last Name: ${person.lastName}")
    println("Age: ${person.age}")
}
```

```
First Name: John
Last Name: Doe
Age: 25
```

18. A program that creates a sealed class named Result and two subclasses Success and Failure.

```
sealed class Result {
    data class Success(val data: String) : Result()
    data class Failure(val error: String) : Result()
}

fun performOperation(): Result {
    // perform some operation
    val successResult: Result = Result.Success("Operation successful")
    val failureResult: Result = Result.Failure("Operation failed")

    return successResult
}

fun main() {
    val result = performOperation()
    when (result) {
        is Result.Success -> println("Success: ${result.data}")
        is Result.Failure -> println("Failure: ${result.error}")
    }
}
```

```
Success: Operation successful
```

19. A program that uses anonymous functions and higher-order functions to calculate the sum of all even numbers in a list of integers.

```
fun main() {
    val numbers = listOf(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

    val sumOfEvens = numbers.filter { it % 2 == 0 }.sum()

    println("Sum of even numbers: $sumOfEvens")
}
```

```
Sum of even numbers: 30
```

20. A program that creates an abstract class Vehicle with two subclasses: Car and Motorcycle.

```
abstract class Vehicle {
    abstract fun start()
    abstract fun stop()
}
```

```
class Car : Vehicle() {
    override fun start() {
        println("Car started")
    }

    override fun stop() {
        println("Car stopped")
    }
}

class Motorcycle : Vehicle() {
    override fun start() {
        println("Motorcycle started")
    }

    override fun stop() {
        println("Motorcycle stopped")
    }
}

fun main() {
    val car = Car()
    car.start()
    car.stop()

    val motorcycle = Motorcycle()
    motorcycle.start()
    motorcycle.stop()
}
```

```
Car started
Car stopped
Motorcycle started
Motorcycle stopped
```

21. A program to read a CSV file, filter the records based on a given condition, and save the filtered records in a new CSV file.

```
import java.io.File

fun main() {
    val inputFile = File("input.csv")
    val outputFile = File("output.csv")
    val condition = "some condition" // Replace with your specific condition

    val filteredRecords = mutableListOf<String>()

    inputFile.bufferedReader().use { reader ->
        var line = reader.readLine()
        while (line != null) {
            if (meetsCondition(line, condition)) {
```

```
                filteredRecords.add(line)
            }
            line = reader.readLine()
        }
    }

    outputFile.bufferedWriter().use { writer ->
        filteredRecords.forEach { record ->
            writer.write(record)
            writer.newLine()
        }
    }

    println("Filtered records saved in ${outputFile.absolutePath}")
}

fun meetsCondition(record: String, condition: String): Boolean {
    // Replace this with your own logic for filtering based on the condition
    return record.contains(condition)
}
```

22. A program that demonstrates the use of Kotlin Flow to emit a sequence of integers and perform a transformation on each emitted value.

```
import kotlinx.coroutines.*
import kotlinx.coroutines.flow.*

fun main() = runBlocking<Unit> {
    // Create a Flow that emits a sequence of integers from 1 to 10
    val flow = flow {
        for (i in 1..10) {
            emit(i)
            delay(100) // Simulate some asynchronous work
        }
    }

    // Apply a transformation to each emitted value
    flow.map { value ->
        "Transformed: $value"
    }.collect { transformedValue ->
        println(transformedValue)
    }
}
```

```
Transformed: 1
Transformed: 2
Transformed: 3
Transformed: 4
Transformed: 5
Transformed: 6
Transformed: 7
Transformed: 8
Transformed: 9
Transformed: 10
```