Практическая №8

1.

```kotlin
fun sumList(list: List<Int>): Int {
    return list.sum()
}

// Альтернативный вариант с использованием reduce:
fun sumListReduce(list: List<Int>): Int {
    return list.reduce { acc, num -> acc + num }
}

// Вариант с явным циклом:
fun sumListLoop(list: List<Int>): Int {
    var sum = 0
    for (num in list) {
        sum += num
    }
    return sum
}

// Вариант с sumOf:
fun sumListSumOf(list: List<Int>): Int {
    return list.sumOf { it }
}


fun main() {
    val numbers = listOf(1, 2, 3, 4, 5)

    println(sumList(numbers))       // Output: 15
    println(sumListReduce(numbers)) // Output: 15
    println(sumListLoop(numbers))   // Output: 15
    println(sumListSumOf(numbers)) // Output: 15
}
```

2.

```kotlin
import kotlin.math.max
import kotlin.math.min

fun differenceMinMax(list: List<Int>): Int? {
    if (list.isEmpty()) return null // Возвращаем null, если список пуст
    var min = list[0]
    var max = list[0]

    for (i in 1 until list.size) {
        min = min(min, list[i])
        max = max(max, list[i])
    }
    return max - min
}


// Более идиоматичный вариант с использованием minOrNull() и maxOrNull():

fun differenceMinMaxMinMaxOrNull(list: List<Int>): Int? {
    val min = list.minOrNull()
    val max = list.maxOrNull()

    if (min == null || max == null) return null // Обработка пустого списка
    return max - min
}
```

```
fun main() {
    val numbers1 = listOf(1, 5, 2, 8, 3)
    println(differenceMinMax(numbers1)) // Output: 7

    val numbers2 = emptyList<Int>()
    println(differenceMinMax(numbers2)) // Output: null

    val numbers3 = listOf(1, 5, 2, 8, 3)
    println(differenceMinMaxMinMaxOrNull(numbers3)) // Output: 7

    val numbers4 = emptyList<Int>()
    println(differenceMinMaxMinMaxOrNull(numbers4)) // Output: null

}
```

3.

```
fun concatenateLists(list1: List<Int>, list2: List<Int>): List<Int> {
    return list1 + list2
}


// Альтернативный вариант с использованием plus():
fun concatenateListsPlus(list1: List<Int>, list2: List<Int>): List<Int> {
    return list1.plus(list2)
}

// Еще один вариант с использованием addAll() (изменяет первый список):
fun concatenateListsAddAll(list1: MutableList<Int>, list2: List<Int>):
MutableList<Int> {
    list1.addAll(list2)
    return list1
}



fun main() {
    val list1 = listOf(1, 2, 3)
    val list2 = listOf(4, 5, 6)

    val combinedList = concatenateLists(list1, list2)
    println(combinedList) // Output: [1, 2, 3, 4, 5, 6]

    val combinedListPlus = concatenateListsPlus(list1, list2)
    println(combinedListPlus) // Output: [1, 2, 3, 4, 5, 6]

    val mutableList1 = mutableListOf(1, 2, 3)
    val combinedListAddAll = concatenateListsAddAll(mutableList1, list2)
    println(combinedListAddAll) // Output: [1, 2, 3, 4, 5, 6]
    println(mutableList1) // Output: [1, 2, 3, 4, 5, 6] - mutableList1 был
изменен

}
```

4.

```
fun profitableGamble(prob: Double, prize: Double, pay: Double): Boolean {
    return prob * prize > pay
}
```

```
fun main() {
    println(profitableGamble(0.2, 50.0, 9.0)) // Output: true
    println(profitableGamble(0.9, 1.0, 2.0))    // Output: false
    println(profitableGamble(0.33, 1000.0, 300.0)) // Output: true
}
```

5.

```
fun profitableGamble(prob: Double, prize: Int,
                     pay: Int): Boolean {
    return prob * prize > pay
}

fun main() {
    println(profitableGamble(0.2,50,9))
    println(profitableGamble(0.9, 1, 2))
    println(profitableGamble(0.1, 1000, 70))
}
```

6.

```
fun lessThan100(num1: Int, num2: Int): Boolean {
    return num1 + num2 < 100
}

fun main() {
    println(lessThan100(20, 30))
    println(lessThan100(50, 60))
    println(lessThan100(0, 100))
    println(lessThan100(0, 99))

    val a = 50
    val b = 60

    val result = when {
        a + b < 100 -> true
        else -> false
    }
    println(result)
}
```

7.

```
fun isDivisibleBy100(num: Int): Boolean {
    return num % 100 == 0
}
fun main () {
    println(isDivisibleBy100(100))
    println(isDivisibleBy100(200))
    println(isDivisibleBy100(30))
    println(isDivisibleBy100(1000))
    println(isDivisibleBy100(1001))

}
```

8.

```
fun calculateFrames(minutes: Int, fps: Int):
        Long {
    return (minutes.toLong() * 60 * fps)
}
fun main () {
    println(calculateFrames(1, 60 ))
    println(calculateFrames(2,30))
```

```
    println(calculateFrames(3, 24))
    println(calculateFrames(10, 120))
}
```

9.

```
import kotlin.math.pow
fun checkPower(n: Int, k: Int):Boolean {
    return n == k.toDouble().pow(k).toInt()
}
fun main() {
    println(checkPower(4, 2))
    println(checkPower(387420489, 9))
    println(checkPower(30, 3))
    println(checkPower(27, 3))
    println(checkPower(1, 1))
}
```

10.

```
fun repeatString(txt: String, n: Int): String {
    return if (n <= 0) {
        ""
    } else {
        txt + repeatString(txt, n - 1)
    }
}
fun main() {
    println(repeatString("Hello", 3))
    println(repeatString("World", 1))
    println(repeatString("Kotlin", 0))
    println(repeatString("Test", -1))
}
```

11.

12.

```
fun google(number: Int): String {
    if (number < 0) {
        return "Количество 'o' не может быть отрицательным"
    }
    val base = "GL"
    val oChars = "o".repeat(number)
    val end = "gle"
    return base + oChars + end
}
fun main() {
    println(google(2))
    println(google(5))
    println(google(0))
    println(google(-2))
}
```

13.

```
fun greetTheWorld() {
    println("Привет, мир!")
}
fun main() {
    greetTheWorld()
}
```

14.

```
fun sum(a: Int, b: Int): Int {
    return a + b
}
fun sumConcise(a: Int, b: Int) = a + b
fun main() {
    val num1 = 10
    val num2 = 5

    val result = sum(num1, num2)
    println("Сумма $num1 и $num2 равна: $result")
    val resultConcise = sumConcise(num1, num2)
    println("Сумма (краткая версия): $resultConcise")

    println(sum(2,3))
    println(sumConcise(2,3))
}
```

15.

```
fun findMax(num1: Int, num2: Int): Int {
    return if (num1 > num2) {
        num1
    } else {
        num2
    }
}

fun findMaxExpressionBody(num1: Int, num2: Int): Int = if (num1 > num2) num1
else num2


fun main() {
    val a = 10
    val b = 5
    val max = findMax(a, b)
    println("Большее число: $max") // Выведет: Большее число: 10

    val max2 = findMaxExpressionBody(a,b)
    println("Большее число (с использованием expression body): $max2")

    val c = 3
    val d = 7
    val max22 = findMax(c, d)
    println("Большее число: $max22") // Выведет: Большее число: 7

}
```

16.

```
fun isEven(num: Int): Boolean {
    return num % 2 == 0
}

// Альтернативный вариант с expression body:
fun isEvenExpressionBody(num: Int): Boolean = num % 2 == 0


fun main() {
    val number1 = 10
    val isEven1 = isEven(number1)
    println("$number1 четное? $isEven1") // Выведет: 10 четное? true
```

```
    val number2 = 7
    val isEven2 = isEven(number2)
    println("$number2 четное? $isEven2") // Выведет: 7 четное? false

    val number3 = 7
    val isEven3 = isEvenExpressionBody(number3)
    println("$number3 четное? (с использованием expression body) $isEven3")
// Выведет: 7 четное? false

}
```

17.

```
fun factorial(n: Int): Long {
    if (n < 0) {
        throw IllegalArgumentException("Факториал не определен для
отрицательных чисел")
    }
    return if (n == 0) 1 else (1..n).map { it.toLong() }.reduce { acc, i ->
acc * i }
}

// Альтернативная реализация с циклом:
fun factorialLoop(n: Int): Long {
    if (n < 0) {
        throw IllegalArgumentException("Факториал не определен для
отрицательных чисел")
    }
    var result = 1L
    for (i in 1..n) {
        result *= i
    }
    return result
}


fun main() {
    val num = 5
    try {
        val result = factorial(num)
        println("Факториал $num = $result") // Выведет: Факториал 5 = 120

        val resultLoop = factorialLoop(num)
        println("Факториал (с циклом) $num = $resultLoop") // Выведет:
Факториал 5 = 120

    } catch (e: IllegalArgumentException) {
        println(e.message)
    }


    val num2 = -1
    try {
        val result2 = factorial(num2)
        println("Факториал $num2 = $result2")
    } catch (e: IllegalArgumentException) {
        println(e.message) // Выведет сообщение об ошибке
    }
}
```

18.

```
fun isPrime(num: Int): Boolean {
    if (num <= 1) return false // 1 и числа меньше 1 не являются простыми
    if (num <= 3) return true // 2 и 3 - простые числа
    if (num % 2 == 0 || num % 3 == 0) return false // Проверка делимости на 2
и 3

    var i = 5
    while (i * i <= num) {
        if (num % i == 0 || num % (i + 2) == 0) return false
        i += 6
    }

    return true
}

fun main() {
    val num1 = 17
    println("$num1 простое? ${isPrime(num1)}") // Output: 17 простое? true

    val num2 = 20
    println("$num2 простое? ${isPrime(num2)}") // Output: 20 простое? false

    val num3 = 1
    println("$num3 простое? ${isPrime(num3)}") // Output: 1 простое? false

    val num4 = 2
    println("$num4 простое? ${isPrime(num4)}") // Output: 2 простое? true

    val num5 = 97
    println("$num5 простое? ${isPrime(num5)}") // Output: 97 простое? true
}
```

19.

```
fun sumOfArray(arr: IntArray): Int {
    var sum = 0
    for (num in arr) {
        sum += num
    }
    return sum
}

// Альтернативный вариант с использованием функции sum():
fun sumOfArray2(arr: IntArray): Int {
    return arr.sum()
}

// Еще один вариант с использованием reduce:
fun sumOfArrayReduce(arr: IntArray): Int {
    return arr.reduce { acc, num -> acc + num }
}



fun main() {
    val numbers = intArrayOf(1, 2, 3, 4, 5)
    val sum = sumOfArray(numbers)
    println("Сумма элементов массива: $sum") // Выведет: 15


    val numbers2 = intArrayOf(10, 20, 30)
    val sum2 = sumOfArray2(numbers2)
```

```
    println("Сумма элементов массива (с использованием sum()): $sum2") //
Output: 60


    val numbers3 = intArrayOf(10, 20, 30)
    val sum3 = sumOfArrayReduce(numbers3)
    println("Сумма элементов массива (с использованием reduce): $sum3") //
Output: 60
}
```

20.

```
import kotlin.math.max

fun findMaxInArray(arr: IntArray): Int? {
    if (arr.isEmpty()) {
        return null // Возвращаем null для пустого массива
    }
    var max = arr[0]
    for (i in 1 until arr.size) {
        max = max(max, arr[i]) // Используем max() из kotlin.math
    }
    return max
}

// Альтернативный вариант с использованием maxOrNull():
fun findMaxInArrayMaxOrNull(arr: IntArray): Int? {
    return arr.maxOrNull()
}

// Еще один вариант с использованием reduce:
fun findMaxInArrayReduce(arr: IntArray): Int? {
    return arr.reduceOrNull { acc, num -> max(acc, num)}
}




fun main() {
    val numbers = intArrayOf(1, 5, 2, 8, 3)
    val max = findMaxInArray(numbers)
    println("Максимальное число: $max") // Output: 8

    val emptyArray = intArrayOf()
    val maxEmpty = findMaxInArray(emptyArray)
    println("Максимальное число в пустом массиве: $maxEmpty") // Output: null

    val numbers2 = intArrayOf(1, 5, 2, 8, 3)
    val max2 = findMaxInArrayMaxOrNull(numbers2)
    println("Максимальное число (с использованием maxOrNull()): $max2") //
Output: 8

    val numbers3 = intArrayOf(1, 5, 2, 8, 3)
    val max3 = findMaxInArrayReduce(numbers3)
    println("Максимальное число (с использованием reduce): $max3") // Output:
8

}
```

21.

```
fun sortArrayAscending(arr: IntArray): IntArray {
    // Создаем копию массива, чтобы не изменять исходный
    val sortedArr = arr.copyOf()
    sortedArr.sort() // Используем встроенную функцию sort()
    return sortedArr
}

// Альтернативный вариант с sortedArray():
fun sortArrayAscendingSortedArray(arr: IntArray): IntArray {
    return arr.sortedArray()
}

fun main() {
    val numbers = intArrayOf(5, 2, 8, 1, 9, 4)

    val sortedNumbers = sortArrayAscending(numbers)
    println("Отсортированный массив: ${sortedNumbers.contentToString()}") //
Output: [1, 2, 4, 5, 8, 9]

    // Проверяем, что исходный массив не изменился
    println("Исходный массив: ${numbers.contentToString()}") // Output: [5,
2, 8, 1, 9, 4]


    val sortedNumbers2 = sortArrayAscendingSortedArray(numbers)
    println("Отсортированный массив (с sortedArray()):
${sortedNumbers2.contentToString()}") // Output: [1, 2, 4, 5, 8, 9]

}
```

22.

```
fun isPalindrome(text: String): Boolean {
    val cleanedText = text.toLowerCase().replace(Regex("[^a-zA-Z0-9]"), "")
// Очистка строки
    return cleanedText == cleanedText.reversed()
}


// Альтернативная реализация с циклом (более эффективная для очень длинных
строк)
fun isPalindromeLoop(text: String): Boolean {
    val cleanedText = text.toLowerCase().replace(Regex("[^a-zA-Z0-9]"), "")
    val n = cleanedText.length
    for (i in 0 until n / 2) {
        if (cleanedText[i] != cleanedText[n - 1 - i]) {
            return false
        }
    }
    return true
}




fun main() {
    val text1 = "A роза упала на лапу Азора"
    println("'$text1' палиндром? ${isPalindrome(text1)}") // Output: true

    val text2 = "hello"
    println("'$text2' палиндром? ${isPalindrome(text2)}") // Output: false
```

```
    val text3 = "A man, a plan, a canal: Panama"
    println("'$text3' палиндром? ${isPalindrome(text3)}") // Output: true

    val text4 = " racecar " // с пробелами
    println("'$text4' палиндром? ${isPalindrome(text4)}") // Output: true



    val text5 = "А роза упала на лапу Азора"
    println("'$text5' палиндром? (с циклом) ${isPalindromeLoop(text5)}") //
Output: true

    val text6 = "hello"
    println("'$text6' палиндром? (с циклом) ${isPalindromeLoop(text6)}") //
Output: false

}
```

23.

```
fun countCharacters(text: String): Int {
    return text.length
}


// Альтернативный вариант (менее предпочтительный):
fun countCharactersAlternative(text: String): Int {
    var count = 0
    for (char in text) {
        count++
    }
    return count
}



fun main() {
    val text = "Hello, world!"
    println(countCharacters(text)) // Output: 13

    val text2 = "Kotlin"
    println(countCharactersAlternative(text2)) // Output: 6

}
```

24.

```
fun toUpperCase(text: String): String {
    return text.uppercase()
}

fun main() {
    val text = "Hello, world!"
    val upperCaseText = toUpperCase(text)
    println(upperCaseText) // Output: HELLO, WORLD!
}
```

25.

```
fun concatenateStrings(str1: String, str2: String): String {
    return str1 + str2
}

// Альтернативный вариант с использованием String.plus():

fun concatenateStringsPlus(str1: String, str2: String): String {
```

```
    return str1.plus(str2)
}


// Еще один вариант с использованием StringBuilder (более эффективно для
многократного объединения строк):
fun concatenateStringsStringBuilder(str1: String, str2: String): String {
    return StringBuilder().append(str1).append(str2).toString()
}


fun main() {
    val str1 = "Hello"
    val str2 = ", world!"

    val combinedString = concatenateStrings(str1, str2)
    println(combinedString) // Output: Hello, world!

    val combinedStringPlus = concatenateStringsPlus(str1, str2)
    println(combinedStringPlus) // Output: Hello, world!

    val combinedStringSB = concatenateStringsStringBuilder(str1, str2)
    println(combinedStringSB) // Output: Hello, world!


}
```

26.

27.

28.

```
fun createArrayOfNumbers(n: Int): IntArray {
    return (1..n).toList().toIntArray()
}

// Альтернативный вариант с использованием IntArray and mapIndexed
fun createArrayOfNumbersMapIndexed(n: Int): IntArray {
    return IntArray(n) { index -> index + 1 }
}




fun main() {
    val n = 5
    val array = createArrayOfNumbers(n)
    println(array.contentToString()) // Output: [1, 2, 3, 4, 5]

    val array2 = createArrayOfNumbersMapIndexed(n)
    println(array2.contentToString()) // Output: [1, 2, 3, 4, 5]

}
```

29.

```
import kotlin.math.max
import kotlin.math.min

data class MinMax(val min: Int, val max: Int)

fun findMinMax(arr: IntArray): MinMax? {
    if (arr.isEmpty()) return null // Обработка пустого массива
```

```
    var min = arr[0]
    var max = arr[0]

    for (i in 1 until arr.size) {
        min = min(min, arr[i])
        max = max(max, arr[i])
    }

    return MinMax(min, max)
}



// Более функциональный стиль, но может быть менее производителен на больших массивах

fun findMinMaxFunctional(arr: IntArray): MinMax? =
    if (arr.isEmpty()) null
    else MinMax(arr.minOrNull()!!, arr.maxOrNull()!!)



fun main() {
    val array = intArrayOf(3, 1, 4, 1, 5, 9, 2, 6)
    val minMax = findMinMax(array)
    println(minMax) // Output: MinMax(min=1, max=9)


    val emptyArray = intArrayOf()
    val minMaxEmpty = findMinMax(emptyArray)
    println(minMaxEmpty) // Output: null



    val array2 = intArrayOf(3, 1, 4, 1, 5, 9, 2, 6)
    val minMax2 = findMinMaxFunctional(array2)
    println(minMax2) // Output: MinMax(min=1, max=9)

}
```

30.

```
fun sumOfNumbers(n: Int): Int {
    return n * (n + 1) / 2
}


// Альтернативный вариант с использованием цикла (менее эффективный):
fun sumOfNumbersLoop(n: Int): Int {
    var sum = 0
    for (i in 1..n) {
        sum += i
    }
    return sum
}

// Альтернативный вариант с использованием reduce (менее эффективный, но функциональный стиль)
fun sumOfNumbersReduce(n: Int): Int = (1..n).reduce { acc, i -> acc + i }



fun main() {
    val n = 5
    println(sumOfNumbers(n))        // Output: 15
```

```
    println(sumOfNumbersLoop(n)) // Output: 15
    println(sumOfNumbersReduce(n)) // Output: 15
}
```

31.

32.

```
fun reverseString(inputString: String): String {
    return inputString.reversed()
}

fun main() {
    val originalString = "Hello, World!"
    val reversedString = reverseString(originalString)
    println("Исходная строка: $originalString")
    println("Перевернутая строка: $reversedString")
}
```

33.

```
fun findElementAtIndex(array: IntArray, index: Int): Int? {
    if (index >= 0 && index < array.size) {
        return array[index]
    } else {
        return null // Или выбросить исключение, в зависимости от требований
    }
}

fun main() {
    val myArray = intArrayOf(10, 20, 30, 40, 50)

    // Пример 1: Поиск элемента по валидному индексу
    val index1 = 2
    val element1 = findElementAtIndex(myArray, index1)
    if (element1 != null) {
        println("Элемент по индексу $index1: $element1") // Вывод: Элемент по
индексу 2: 30
    } else {
        println("Индекс $index1 выходит за границы массива.")
    }

    // Пример 2: Поиск элемента по невалидному индексу (отрицательному)
    val index2 = -1
    val element2 = findElementAtIndex(myArray, index2)
    if (element2 != null) {
        println("Элемент по индексу $index2: $element2")
    } else {
        println("Индекс $index2 выходит за границы массива.") // Вывод:
Индекс -1 выходит за границы массива.
    }

    // Пример 3: Поиск элемента по невалидному индексу (больше размера
массива)
    val index3 = 5
    val element3 = findElementAtIndex(myArray, index3)
    if (element3 != null) {
        println("Элемент по индексу $index3: $element3")
    } else {
        println("Индекс $index3 выходит за границы массива.") // Вывод:
Индекс 5 выходит за границы массива.
    }
}
```

34.

```
fun removeSpaces(inputString: String): String {
    return inputString.replace(" ", "")
}

fun main() {
    val stringWithSpaces = "  Hello   World!  "
    val stringWithoutSpaces = removeSpaces(stringWithSpaces)
    println("Исходная строка: '$stringWithSpaces'")
    println("Строка без пробелов: '$stringWithoutSpaces'")
}
```

35.

```
fun sumOfFirstNNaturalNumbers(n: Int): Int {
    if (n <= 0) {
        return 0 // Сумма первых 0 или отрицательного числа чисел равна 0
    }
    return n * (n + 1) / 2 // Формула для суммы арифметической прогрессии
}

fun main() {
    val n1 = 5
    val sum1 = sumOfFirstNNaturalNumbers(n1)
    println("Сумма первых $n1 натуральных чисел: $sum1") // Вывод: Сумма
первых 5 натуральных чисел: 15

    val n2 = 10
    val sum2 = sumOfFirstNNaturalNumbers(n2)
    println("Сумма первых $n2 натуральных чисел: $sum2") // Вывод: Сумма
первых 10 натуральных чисел: 55

    val n3 = 0
    val sum3 = sumOfFirstNNaturalNumbers(n3)
    println("Сумма первых $n3 натуральных чисел: $sum3") // Вывод: Сумма
первых 0 натуральных чисел: 0

    val n4 = -3
    val sum4 = sumOfFirstNNaturalNumbers(n4)
    println("Сумма первых $n4 натуральных чисел: $sum4") // Вывод: Сумма
первых -3 натуральных чисел: 0
}
```

36.

```
fun containsSubstring(text: String, substring: String): Boolean {
    return text.contains(substring)
}

fun main() {
    val mainString = "This is a sample string for testing."
    val substring1 = "sample"
    val substring2 = "not found"

    val contains1 = containsSubstring(mainString, substring1)
    println("Строка '$mainString' содержит подстроку '$substring1':
$contains1") // Вывод: Строка 'This is a sample string for testing.' содержит
подстроку 'sample': true

    val contains2 = containsSubstring(mainString, substring2)
    println("Строка '$mainString' содержит подстроку '$substring2':
$contains2") // Вывод: Строка 'This is a sample string for testing.' содержит
```

```
подстроку 'not found': false
}
```

37.

```
fun printMultiplicationTable(number: Int) {
    if (number <= 0) {
        println("Пожалуйста, введите положительное целое число.")
        return
    }

    println("Таблица умножения для $number:")
    for (i in 1..10) {
        val result = number * i
        println("$number x $i = $result")
    }
}

fun main() {
    val num = 7
    printMultiplicationTable(num)

    println("\nТаблица умножения для 5:")
    printMultiplicationTable(5)

    println("\nПроверка с отрицательным числом:")
    printMultiplicationTable(-2) // Вывод: Пожалуйста, введите положительное
целое число.
}
```

38.

```
fun stringLength(inputString: String): Int {
    return inputString.length
}

fun main() {
    val myString = "Hello, Kotlin!"
    val length = stringLength(myString)
    println("Длина строки '$myString' равна $length")

    val emptyString = ""
    val emptyLength = stringLength(emptyString)
    println("Длина пустой строки равна $emptyLength")
}
```

39.

```
fun reverseArray(array: IntArray): IntArray {
    val reversedArray = IntArray(array.size)
    for (i in array.indices) {
        reversedArray[i] = array[array.size - 1 - i]
    }
    return reversedArray
}

fun main() {
    val originalArray = intArrayOf(1, 2, 3, 4, 5)
    val reversedArray = reverseArray(originalArray)
    println("Исходный массив: ${originalArray.contentToString()}")
    println("Перевернутый массив: ${reversedArray.contentToString()}")
}
```

40.

```
fun copyArray(array: IntArray): IntArray {
    val newArray = IntArray(array.size)
    for (i in array.indices) {
        newArray[i] = array[i]
    }
    return newArray
}

fun main() {
    val originalArray = intArrayOf(1, 2, 3, 4, 5)
    val copiedArray = copyArray(originalArray)

    println("Исходный массив: ${originalArray.contentToString()}")
    println("Скопированный массив: ${copiedArray.contentToString()}")

    // Проверка, что это разные массивы (изменение copiedArray не влияет на
originalArray)
    copiedArray[0] = 100
    println("Исходный массив после изменения copiedArray:
${originalArray.contentToString()}")
    println("Скопированный массив после изменения:
${copiedArray.contentToString()}")
}
```

41.

```
fun countVowels(text: String): Int {
    val vowels = "aeiouAEIOU" // Или можно использовать setOf('a', 'e', 'i',
'o', 'u', 'A', 'E', 'I', 'O', 'U')
    var count = 0
    for (char in text) {
        if (vowels.contains(char)) {
            count++
        }
    }
    return count
}

fun main() {
    val myString = "Hello, World!"
    val vowelCount = countVowels(myString)
    println("Строка '$myString' содержит $vowelCount гласных.")

    val anotherString = "AEIOUaeiou"
    val anotherVowelCount = countVowels(anotherString)
    println("Строка '$anotherString' содержит $anotherVowelCount гласных.")

    val emptyString = ""
    val emptyVowelCount = countVowels(emptyString)
    println("Пустая строка содержит $emptyVowelCount гласных.")
}
```

42.

```
fun indexOf(array: IntArray, element: Int): Int {
    for (i in array.indices) {
        if (array[i] == element) {
            return i // Возвращаем индекс, как только элемент найден
        }
    }
    return -1 // Возвращаем -1, если элемент не найден в массиве
```

```
}

fun main() {
    val myArray = intArrayOf(10, 20, 30, 20, 40, 50)

    val element1 = 20
    val index1 = indexOf(myArray, element1)
    println("Индекс первого вхождения элемента $element1: $index1") // Вывод:
Индекс первого вхождения элемента 20: 1

    val element2 = 35
    val index2 = indexOf(myArray, element2)
    println("Индекс первого вхождения элемента $element2: $index2") // Вывод:
Индекс первого вхождения элемента 35: -1
}
```