МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ОТЧЁТ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ № 13  
по дисциплине "Функциональное и логическое программирование"**

**Вариант 12**

Работу выполнил\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.В.Чухиль

Направление подготовки 02.03.02 Фундаментальная информатика и

информационные технологии

Направленность (профиль) Математическое и программное обеспечение  
 компьютерных технологий

Преподаватель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.А.Климец

Краснодар

2020

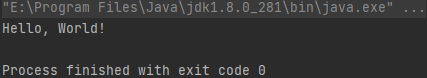
**Задание 1.**

Установить Intellij Idea, JDK и плагин для Kotlin. Реализовать Hello World.

**Код:**

fun main(args: Array<String>)  
{  
 *println*("Hello, World!")  
}

**Скрины:**



**Задание 2.**

Собрать файл \*.jar из предыдущей программы, запустить его из командной строки.

**Код:**

fun main(args: Array<String>)  
{  
 *println*("Hello, World!")  
}

**Скрины:**



**Задание 3.**

Принять имя пользователя как аргумент программы. Поздороваться с пользователем. Применить форматирование строки.

**Код:**

fun main(args: Array<String>)  
{  
 *println*("Hello, ${args[0]}!")  
}

**Скрины:**



**Задание 4.**

Рассмотреть методы класса String, включить в отчёт половину методов.

**Таблица 1. Методы и их описание**

|  |  |
| --- | --- |
| Метод | Описание |
| fun compareTo(**other**: String): Int | Сравнивает текущий объект с указанным. Возвращает 0, если текущий равен указанному объекту, отрицательное число, если текущий меньше указанного, или положительное число, если текущий больше указанного. |
| fun equals(**other**: Any?): Boolean | Указывает, равен ли другой объект текущему. Реализация должна соответствовать следующим требованиям:   1. Reflexive: для любого ненулевого x, x.equals(x) должен возвращать true 2. Symmetric: для любых ненулевых x и y, x.equals(y) должен возвращать true тогда, когда y.equals(x) возвращает true 3. Transitive: для любых ненулевых x, y и z, если x.equals(y) возвращает true и y.equals(z) возвращает true, то x.equals(z) должен возвращать true 4. Consistent: для любых ненулевых x и y, множественные вызовы x.equals(y) последовательно возвращают true или последовательно возвращают false, при условии, что никакая информация, используемая в равных сравнениях для объектов, не изменена 5. Never equal to null: для любого ненулевого x, x.equals(null) должен возвращать false |
| fun get(**index**: Int): Char | Возвращает символ этой строки по указанному индексу.  Если индекс выходит за пределы этой строки, выдаёт исключение IndexOutOfBoundsException. |
| fun hashCode(): Int | Возвращает значение хэш-кода для объекта. Общий контракт hashCode:   1. Каждый раз, когда он вызывается для одного и того же объекта более одного раза, метод hashCode должен последовательно возвращать одно и то же целое число, при условии, что никакая информация, используемая в равных сравнениях для объекта, не изменяется. 2. Если 2 объекта равны согласно методу equals(), то вызов метода hashCode для каждого из двух объектов должен привести к одинаковому целочисленному результату. |
| operator fun plus(**other**: Any?): String | Возвращает строку, полученную путем объединения этой строки со строковым представлением данного другого объекта. |
| fun toString(): String | Возвращает строковое представление объекта. |
| fun subSequence(**startIndex**: Int, **endIndex**: Int): [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html) | Возвращает новую последовательность символов, которая является подпоследовательностью этой последовательности символов, начиная с указанного startIndex и заканчивая прямо перед указанным endIndex. |
| inline fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).all(     **predicate**: (Char) -> Boolean ): Boolean | Возвращает истину, если все символы соответствуют заданному предикату. |
| fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).any(): Boolean | Возвращает истину, если последовательность символов содержит хотя бы один символ. |
| inline fun <K, V> [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).associate(     **transform**: (Char) -> [Pair](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-pair/index.html)<K, V> ): [Map](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-map/index.html)<K, V> | Возвращает карту, содержащую пары ключ-значение, предоставленные функцией преобразования, примененной к символам данной последовательности символов.  Если у любой из двух пар будет один и тот же ключ, на карту добавляется последняя.  Возвращенная карта сохраняет порядок итерации входа исходной последовательности символов. |
| fun String.byteInputStream(     **charset**: [Charset](https://docs.oracle.com/javase/8/docs/api/java/nio/charset/Charset.html) = Charsets.UTF\_8 ): [ByteArrayInputStream](https://docs.oracle.com/javase/8/docs/api/java/io/ByteArrayInputStream.html) | Создает новый байтовый поток ввода для строки. |
| fun String.codePointAt(**index**: Int): Int | Возвращает символ (кодовую точку Unicode) по указанному индексу. |
| fun String.codePointCount(     **beginIndex**: Int,     **endIndex**: Int ): Int | Возвращает количество кодовых точек Unicode в указанном текстовом диапазоне этой String. |
| fun String.concat(**str**: String): String | Возвращает новую строку, состоящую из добавления указанной строки в конец текущей. |
| operator fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).contains(     **other**: [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html),     **ignoreCase**: Boolean = false ): Boolean | Возвращает истину, если эта последовательность символов содержит указанную другую последовательность символов в качестве подстроки.  Параметр ignoreCase – true игнорирует регистр символов при сравнении строк. По умолчанию false. |
| fun String.contentEquals(**charSequence**: [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html)): Boolean | Возвращает true, если эта строка равна содержимому указанной CharSequence, в противном случае - false. |
| fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).count(): Int | Возвращает длину этой последовательности символов. |
| inline fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).count(     **predicate**: (Char) -> Boolean ): Int | Возвращает количество символов, соответствующих заданному предикату. |
| fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).drop(**n**: Int): [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html) | Возвращает подпоследовательность этой последовательности символов с удаленными первыми n символами. |
| inline fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).dropLastWhile(     **predicate**: (Char) -> Boolean ): [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html)  Example:  val string = "<<<First Grade>>>"  println(string.dropLastWhile { !it.isLetter() })  // <<<First Grade | Возвращает подпоследовательность этой последовательности символов, содержащую все символы, кроме последних символов, которые удовлетворяют заданному предикату. |
| inline fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).elementAtOrElse(     **index**: Int,     **defaultValue**: (Int) -> Char ): Char  Example:  val list = listOf(1, 2, 3)  println(list.elementAtOrElse(2) { 42 }) // 3  println(list.elementAtOrElse(3) { 42 }) // 42  val emptyList = emptyList<Int>()  println(emptyList.elementAtOrElse(0) { "no int" })  // no int | Возвращает символ по заданному индексу или результат вызова функции defaultValue, если индекс находится за пределами этой последовательности символов. |
| fun String.endsWith(     **suffix**: String,     **ignoreCase**: Boolean ): Boolean | Возвращает истину, если эта строка заканчивается указанным суффиксом. |
| inline fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).filter(     **predicate**: (Char) -> Boolean ): [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html)  Example:  val text = "a1b2c3d4e5"  val textWithOnlyDigits = text.filter { it.isDigit() }  println(textWithOnlyDigits) // 12345 | Возвращает последовательность символов, содержащую только те символы из исходной последовательности символов, которые соответствуют заданному предикату. |
| inline fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).find(     **predicate**: (Char) -> Boolean ): Char?  Example:  val numbers = listOf(1, 2, 3, 4, 5, 6, 7)  val firstOdd = numbers.find { it % 2 != 0 }  val lastEven = numbers.findLast { it % 2 == 0 }  println(firstOdd) // 1  println(lastEven) // 6 | Возвращает первый символ, соответствующий данному предикату, или null, если такой символ не был найден. |
| inline fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).first(): Char | Возвращает первый символ.  NoSuchElementException - если последовательность символов пуста. |
| inline fun <R> [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).flatMap(     **transform**: (Char) -> [Iterable](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-iterable/index.html)<R> ): [List](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-list/index.html)<R>  Example:  val list = listOf("123", "45")  println(list.flatMap { it.toList() })  // [1, 2, 3, 4, 5] | Возвращает единый список всех элементов, полученных в результате вызова функции преобразования для каждого символа исходной последовательности символов. |
| inline fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).forEach(**action**: (Char) -> Unit) | Выполняет заданное действие над каждым персонажем. |
| fun String.format(vararg **args**: Any?): String  Example by Java:  String str = String.format("Привет - %s! Как дела %s?", "Саша", "на работе"); | Использует эту строку как строку формата и возвращает строку, полученную путем подстановки указанных аргументов с использованием языкового стандарта по умолчанию. |
| inline fun <C, R> C.ifBlank(     **defaultValue**: () -> R ): R where C : [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html), C : R  Example:  val blank = " "  val blankOrNull: String? = blank.ifBlank { null }  println(blankOrNull) // null  val blankOrDefault = blank.ifBlank { "default" }  println(blankOrDefault) // default  val nonBlank = "abc"  val sameString = nonBlank.ifBlank { "def" }  println("nonBlank === sameString is ${nonBlank === sameString}") // true | Возвращает эту последовательность символов, если она не пуста и не состоит только из пробельных символов, или результат вызова функции defaultValue в противном случае. |
| fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).indexOf(     **char**: Char,     **startIndex**: Int = 0,     **ignoreCase**: Boolean = false ): Int | Возвращает индекс в этой строке первого вхождения указанного символа, начиная с указанного startIndex, или -1, если ничего не найдено. |
| fun String.intern(): String | Возвращает каноническое представление для этого строкового объекта. |
| operator fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).iterator(): [CharIterator](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-char-iterator/index.html) | Итератор для символов данной последовательности символов. |
| fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).lines(): [List](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-list/index.html)<String> | Разбивает эту последовательность символов на список строк, разделенных любой из следующих последовательностей символов: CRLF (\r\n), LF (перевод строки \n) или CR (возврат каретки \r).  Возвращаемые строки не содержат завершающих разделителей строк. |
| @ExperimentalStdlibApi fun Char.lowercase(): String | Преобразует этот символ в нижний регистр, используя правила отображения Unicode для инвариантной локали. |
| inline fun <R> [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).map(     **transform**: (Char) -> R ): [List](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-list/index.html)<R>  Example:  val string = "kotlin"  println(string.map { it.toUpperCase() }) // [K, O, T, L, I, N] | Возвращает список, содержащий результаты применения данной функции преобразования к каждому символу в исходной последовательности символов. |
| fun String.match(**regex**: String): Array<String>? | Метод match производит поиск по заданной строке с использованием регулярного выражения и возвращает массив, предоставленные результаты этого поиска. |
| infix fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).matches(**regex**: [Regex](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.text/-regex/index.html)): Boolean | Возвращает истину, если эта последовательность символов соответствует заданному регулярному выражению. |
| fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).maxOrNull(): Char? | Возвращает самый большой символ или null, если символов нет. |
| inline fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).maxOf(     **selector**: (Char) -> Double ): Double  inline fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).maxOf(     **selector**: (Char) -> Float ): Float | Возвращает наибольшее значение среди всех значений, созданных функцией селектора, примененной к каждому символу в последовательности символов.  Если какое-либо из значений, созданных функцией селектора, - NaN, возвращаемый результат - NaN.  Исключения  NoSuchElementException - если последовательность символов пуста. |
| fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).none(): Boolean | Возвращает истину, если последовательность символов не имеет символов. |
| inline fun <S : [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html)> S.onEach(     **action**: (Char) -> Unit ): S | Выполняет заданное действие с каждым символом и впоследствии возвращает саму последовательность символов. |
| fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).random(): Char | Возвращает случайный символ из этой последовательности символов.  Исключения  NoSuchElementException - если эта последовательность символов пуста. |
| fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).removePrefix(     **prefix**: [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html) ): [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html) | Если эта последовательность символов начинается с данного префикса, возвращает новую последовательность символов с удаленным префиксом. В противном случае возвращает новую последовательность символов с теми же символами. |
| fun String.replaceAfter(     **delimiter**: Char,     **replacement**: String,     **missingDelimiterValue**: String = this ): String  fun String.replaceAfter(     **delimiter**: String,     **replacement**: String,     **missingDelimiterValue**: String = this ): String | Заменить часть строки после первого появления данного разделителя на заменяющую строку. Если строка не содержит разделителя, возвращается значение missingDelimiterValue, которое по умолчанию соответствует исходной строке. |
| fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).replaceRange(     **startIndex**: Int,     **endIndex**: Int,     **replacement**: [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html) ): [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html) | Возвращает последовательность символов с содержимым этой последовательности символов, где ее часть в заданном диапазоне заменяется заменяющей последовательностью символов. |
| fun String.reversed(): String | Возвращает строку с символами в обратном порядке. |
| inline fun <R> [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).runningFold(     **initial**: R,     **operation**: (**acc**: R, Char) -> R ): [List](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-list/index.html)<R>  Example:  val strings = listOf("a", "b", "c", "d")  println(strings.runningFold("s") { acc, string -> acc + string }) // [s, sa, sab, sabc, sabcd]  println(strings.runningFoldIndexed("s") { index, acc, string -> acc + string + index })  // [s, sa0, sa0b1, sa0b1c2, sa0b1c2d3]  println(emptyList<String>().runningFold("s") { \_, \_ -> "X" }) // [s] | Возвращает список, содержащий последовательные значения накопления, сгенерированные путем применения операции слева направо к каждому символу и текущему значению аккумулятора, которое начинается с начального значения.  Обратите внимание, что значение acc, передаваемое в функцию операции, не должно изменяться; в противном случае это повлияет на предыдущее значение в результирующем списке.  operation - функция, которая берет текущее значение аккумулятора и символ и вычисляет следующее значение аккумулятора. |
| fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).slice(**indices**: [IntRange](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.ranges/-int-range/index.html)): [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html)  # IntRange example  for (i in 0..10) (от 0 до 10)  for (i in 10 downto 0) (от 10 до 0)  for (i in 1..4 step 2) (от 1 до 4 с шагом 2)  for (i in 1 until 10) (от 1 до 10 невключительно) | Возвращает последовательность символов, содержащую символы исходной последовательности символов в указанном диапазоне индексов. |
| fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).split(     vararg **delimiters**: String,     **ignoreCase**: Boolean = false,     **limit**: Int = 0 ): [List](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-list/index.html)<String> | Разбивает эту последовательность символов на список строк вокруг вхождений указанных разделителей.  delimiters - одна или несколько строк, которые будут использоваться в качестве разделителей.  ignoreCase - true игнорировать регистр символов при сопоставлении разделителя. По умолчанию false.  limit - максимальное количество возвращаемых подстрок. Ноль по умолчанию означает, что ограничение не установлено.  Чтобы избежать неоднозначных результатов, когда строки в разделителях имеют общие символы, этот метод переходит от начала до конца этой строки и сопоставляет в каждой позиции первый элемент в разделителях, который в данном случае равен разделителю в этой позиции. |
| fun String.substring(**startIndex**: Int, **endIndex**: Int): String | Возвращает подстроку этой строки, начиная с startIndex и заканчивая прямо перед endIndex. |
| fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).take(**n**: Int): [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html)  Example:  val string = "<<<First Grade>>>"  println(string.take(8)) // <<<First  println(string.takeLast(8)) // Grade>>>  println(string.takeWhile { !it.isLetter() }) // <<<  println(string.takeLastWhile { !it.isLetter() }) // >>> | Возвращает подпоследовательность этой последовательности символов, содержащую первые n символов из этой последовательности символов, или всю последовательность символов, если эта последовательность символов короче. |
| fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).toList(): [List](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-list/index.html)<Char> | Возвращает список, содержащий все символы. |
| fun [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).toHashSet(): [HashSet](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-hash-set/index.html)<Char> | Возвращает новый HashSet всех символов. |
| fun <C : [MutableCollection](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-mutable-collection/index.html)<in Char>> [CharSequence](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-char-sequence/index.html).toCollection(     **destination**: C ): C | Добавляет всех символов в указанную целевую коллекцию. |

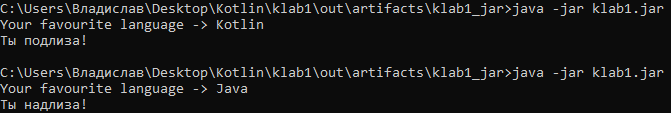
**Задание 5.**

Продолжение задания 3. Спросить у пользователя, какой язык у него любимый, если это Kotlin или Prolog, ответить пользователю, что он подлиза, для других языков придумать комментарий, воспользоваться для решения задачи условным оператором и оператором when.

**Код:**

fun main(args: Array<String>)  
{  
 *print*("Your favourite language -> ")  
 val language = *readLine*()  
 /\*  
 // first type (if)  
 if (language.equals("Kotlin") ||  
 language.equals("Prolog"))  
 println("Ты подлиза!")  
 else  
 println("Ты надлиза!")  
 \*/  
 /\*  
 // second type (when)  
 when (language)  
 {  
 "Kotlin", "Prolog" -> println("Ты подлиза!")  
 else -> println("Ты надлиза!")  
 }  
 \*/  
 // third type (when)  
 val output =  
 when (language) // или when(readLine())  
 {  
 "Kotlin", "Prolog" -> "Ты подлиза!"  
 else -> "Ты надлиза!"  
 }  
 *println*(output)  
}

**Скрины:**



**Задание 6.**

Новая программа “Работа с числами”. Найти сумму цифр числа. Рассмотреть класс целое число. Разобрать и включить в отчёт половину методов с использованием официальной документации языка.

**Код:**

// Задание 6. Работа с числами. Операторы ?. и ?:  
fun sumOfNumber(number: Int, sum: Int = 0) : Int  
= if (number == 0)  
 sum  
 else  
 {  
 val digit = number % 10  
 *sumOfNumber*(number / 10, sum + digit)  
 }  
  
fun main(args: Array<String>)  
{  
 /\*  
 Использование ?. вызывает метод только в том случае, если значение не равно null,  
 в противном случае он просто передает null.  
 Использование ?: означает, что значение слева возвращается, если оно не является null,  
 в противном случае возвращается значение справа  
 \*/  
 *print*("Number -> ")  
 try {  
 val number = *readLine*()?.*toInt*() ?: 0  
 *println*("Сумма цифр числа $number равна ${*sumOfNumber*(number)}")  
 }  
 catch (ex: NumberFormatException)  
 {  
 *println*("Некорректный ввод")  
 }  
}

**Скрины:**



**Таблица 1. Методы и их описание**

|  |  |
| --- | --- |
| Метод | Описание |
| infix fun and(**other**: Int): Int | Выполняет побитовую операцию И между двумя значениями. |
| operator fun compareTo(**other**: Byte): Int  operator fun compareTo(**other**: Short): Int  operator fun compareTo(**other**: Int): Int  operator fun compareTo(**other**: Long): Int  operator fun compareTo(**other**: Float): Int  operator fun compareTo(**other**: Double): Int | Сравнивает это значение с указанным значением для заказа. Возвращает ноль, если это значение равно указанному другому значению, отрицательное число, если оно меньше другого, или положительное число, если оно больше другого. |
| operator fun dec(): Int | Уменьшает это значение на 1. |
| operator fun div(**other**: Byte): Int  operator fun div(**other**: Short): Int  operator fun div(**other**: Int): Int  operator fun div(**other**: Long): Long  operator fun div(**other**: Float): Float  operator fun div(**other**: Double): Double | Делит это значение на другое значение. |
| fun equals(**other**: Int): Boolean | Указывает, равны ли 2 объекта класса Int. Вместо Int может быть Any?, объект любого класса. |
| fun hashCode(): Int | Возвращает значение хэш-кода для объекта. |
| operator fun inc(): Int | Увеличивает это значение на 1. |
| fun inv(): Int | Инвертирует биты в этом значении. |
| operator fun minus(**other**: Byte): Int  operator fun minus(**other**: Short): Int  operator fun minus(**other**: Int): Int  operator fun minus(**other**: Long): Long  operator fun minus(**other**: Float): Float  operator fun minus(**other**: Double): Double | Вычитает другое значение из этого значения. |
| infix fun or(**other**: Int): Int | Выполняет побитовую операцию ИЛИ между двумя значениями. |
| operator fun rangeTo(**other**: Byte): [IntRange](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.ranges/-int-range/index.html)  operator fun rangeTo(**other**: Short): [IntRange](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.ranges/-int-range/index.html)  operator fun rangeTo(**other**: Int): [IntRange](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.ranges/-int-range/index.html)  operator fun rangeTo(**other**: Long): [LongRange](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.ranges/-long-range/index.html) | Создает диапазон (массив) от этого значения до указанного другого значения. |
| operator fun rem(**other**: Byte): Int  operator fun rem(**other**: Short): Int  operator fun rem(**other**: Int): Int  operator fun rem(**other**: Long): Long  operator fun rem(**other**: Float): Float  operator fun rem(**other**: Double): Double | Вычисляет остаток от деления этого значения на другое значение. |
| infix fun shl(**bitCount**: Int): Int | Сдвигает это значение влево на количество битов bitCount. Функция shr сдвигает вправо. |
| operator fun times(**other**: Byte): Int  operator fun times(**other**: Short): Int  operator fun times(**other**: Int): Int  operator fun times(**other**: Long): Long  operator fun times(**other**: Float): Float  operator fun times(**other**: Double): Double | Умножает это значение на другое значение. |
| fun toString(): String | Возвращает строковое представление объекта. |
| infix fun ushr(**bitCount**: Int): Int | Сдвигает это значение вправо на количество битов bitCount, заполняя крайние левые биты нулями. |
| fun <T : [Comparable](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-comparable/index.html)<T>> T.coerceIn(     **minimumValue**: T?,     **maximumValue**: T? ): T  Example:  val workingDays = DayOfWeek.MONDAY..DayOfWeek.FRIDAY  println(DayOfWeek.WEDNESDAY.coerceIn(workingDays)) // WEDNESDAY  println(DayOfWeek.SATURDAY.coerceIn(workingDays)) // FRIDAY  println(DayOfWeek.FRIDAY.coerceIn(DayOfWeek.SATURDAY, DayOfWeek.SUNDAY)) // SATURDAY | Гарантирует, что это значение находится в указанном диапазоне minimumValue..maximumValue.  Вернёт это значение, если оно находится в диапазоне, или minimumValue, если это значение меньше minimumValue, или maximumValue, если это значение больше maximumValue. |

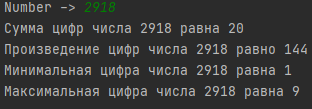
**Задание 7.**

“Работа с числами”. Вынести нахождение суммы цифр числа в отдельный метод. Реализовать методы поиска максимальной и минимальной цифры числа, произведения цифр числа. Не удалять файл, в следующих лабораторных работах будет внесена модификация.

**Код:**

// Задание 7. Работа с числами.  
// сумма цифр числа  
fun sumOfNumber(number: Int, sum: Int = 0) : Int  
= if (number == 0)  
 sum  
else  
 *sumOfNumber*(number / 10, sum + (number % 10))  
  
// произведение цифр числа  
fun prodOfNumber(number: Int, prod: Int = 1) : Int  
= if (number == 0)  
 prod  
else  
 *prodOfNumber*(number / 10, prod \* (number % 10))  
  
// минимальная цифра числа  
fun minDigitInNumber(number: Int, min: Int = 9) : Int  
= if (number == 0)  
 min  
else  
 *minDigitInNumber*(number / 10, (if (number % 10 < min) number % 10 else min))  
  
// максимальная цифра числа  
fun maxDigitInNumber(number: Int, max: Int = 0) : Int  
= if (number == 0)  
 max  
else  
 *maxDigitInNumber*(number / 10, (if (number % 10 > max) number % 10 else max))  
  
fun main(args: Array<String>)  
{  
 *print*("Number -> ")  
 try {  
 val number = *readLine*()?.*toInt*() ?: 0  
 *println*("Сумма цифр числа $number равна ${*sumOfNumber*(number)}")  
 *println*("Произведение цифр числа $number равно ${  
 if (number != 0) *prodOfNumber*(number) else 0}")  
 *println*("Минимальная цифра числа $number равна ${  
 if (number != 0) *minDigitInNumber*(number) else 0}")  
 *println*("Максимальная цифра числа $number равна ${*maxDigitInNumber*(number)}")  
 }  
 catch (ex: NumberFormatException)  
 {  
 *println*("Некорректный ввод")  
 }  
}

**Скрины:**



**Задание 8.**

“Работа с числами”. Составить 3 метода для работы с цифрами или делителями числа на основании варианта. Каждый метод отдельным коммитом.

Метод 1. Найти сумму непростых делителей числа.

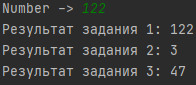
Метод 2. Найти количество цифр числа, меньших 3.

Метод 3. Найти количество чисел, не являющихся делителями исходного числа, не взаимно простых с ним и взаимно простых с суммой простых цифр этого числа.

**Код:**

// Задание 8. Работа с числами.  
// проверка числа на простоту  
fun simpleNumber(number: Int, del: Int = number - 1) : Boolean =  
 when  
 {  
 number == 1 -> true  
 del == 1 -> true  
 number % del == 0 -> false  
 else -> *simpleNumber*(number, del - 1)  
 }  
  
// Подзадание 1. Сумма непростых делителей числа.  
fun sumOfNoSimpleDelOfNumber(number: Int, del: Int = number, sum: Int = 0) : Int =  
 when  
 {  
 (number == 0) -> 0  
 (del == 0) -> sum  
 (number % del == 0 && !*simpleNumber*(del)) -> *sumOfNoSimpleDelOfNumber*(number, del - 1, sum + del)  
 else -> *sumOfNoSimpleDelOfNumber*(number, del - 1, sum)  
 }  
  
// Подзадание 2. Количество цифр числа, меньших 3.  
fun countDigitsLess3(number: Int, count: Int = 0) : Int =  
 when  
 {  
 (number == 0) -> count  
 (number % 10 < 3) -> *countDigitsLess3*(number / 10, count + 1)  
 else -> *countDigitsLess3*(number / 10, count)  
 }  
  
// Подзадание 3. Количество чисел, которые  
// не являются делителями исходного числа  
// не взаимно простые с ним  
// взаимно простые с суммой простых цифр этого числа  
  
// НОД двух чисел  
fun nodTwoNumbers(number1: Int, number2: Int, nod: Int = (if (number1 > number2) number2 else number1)) : Int =  
 when  
 {  
 (number1 == 0 || number2 == 0) -> -1  
 (number1 % nod == 0 && number2 % nod == 0) -> nod  
 else -> *nodTwoNumbers*(number1, number2, nod - 1)  
 }  
  
// Сумма простых цифр числа  
fun sumOfSimpleDigitsOfNumber(number: Int, sum: Int = 0) : Int =  
 when  
 {  
 (number == 0) -> sum  
 *simpleNumber*(number % 10) -> *sumOfSimpleDigitsOfNumber*(number / 10, sum + (number % 10))  
 else -> *sumOfSimpleDigitsOfNumber*(number / 10, sum)  
 }  
  
// Основное подзадание 3  
fun task8method3(number: Int, del: Int = number - 1, count: Int = 0) : Int =  
 when  
 {  
 (del == -1 || del == 0) -> 0  
 (del == 1) -> count  
 (  
 number % del != 0 &&  
 *nodTwoNumbers*(number, del) != 1 &&  
 *nodTwoNumbers*(*sumOfSimpleDigitsOfNumber*(number), del) == 1  
 ) -> *task8method3*(number, del - 1, count + 1)  
 else -> *task8method3*(number, del - 1, count)  
 }  
  
fun main(args: Array<String>)  
{  
 *print*("Number -> ")  
 try {  
 val number = *readLine*()?.*toInt*() ?: 0  
 *println*("Результат задания 1: ${*sumOfNoSimpleDelOfNumber*(number)}")  
 *println*("Результат задания 2: ${*countDigitsLess3*(number)}")  
 *println*("Результат задания 3: ${*task8method3*(number)}")  
 }  
 catch (ex: NumberFormatException)  
 {  
 *println*("Некорректный ввод")  
 }  
}

**Скрины:**



**Задание 9.**

“Работа с числами”. Реализовать возможность пользователю выбирать, какие из методов для введённого числа он хочет исполнить, и продолжать работу программы, пока пользователь не укажет обратное, то есть пользователь может вводить числа и методы для них, пока не введёт соответствующую команду.

**Код:**

// Задание 8-9. Работа с числами.  
// проверка числа на простоту  
fun simpleNumber(number: Int, del: Int = number - 1) : Boolean =  
 when  
 {  
 number == 1 -> true  
 del == 1 -> true  
 number % del == 0 -> false  
 else -> *simpleNumber*(number, del - 1)  
 }  
  
// Подзадание 1. Сумма непростых делителей числа.  
fun sumOfNoSimpleDelOfNumber(number: Int, del: Int = number, sum: Int = 0) : Int =  
 when  
 {  
 (number == 0) -> 0  
 (del == 0) -> sum  
 (number % del == 0 && !*simpleNumber*(del)) -> *sumOfNoSimpleDelOfNumber*(number, del - 1, sum + del)  
 else -> *sumOfNoSimpleDelOfNumber*(number, del - 1, sum)  
 }  
  
// Подзадание 2. Количество цифр числа, меньших 3.  
fun countDigitsLess3(number: Int, count: Int = 0) : Int =  
 when  
 {  
 (number == 0) -> count  
 (number % 10 < 3) -> *countDigitsLess3*(number / 10, count + 1)  
 else -> *countDigitsLess3*(number / 10, count)  
 }  
  
// Подзадание 3. Количество чисел, которые  
// не являются делителями исходного числа  
// не взаимно простые с ним  
// взаимно простые с суммой простых цифр этого числа  
  
// НОД двух чисел  
fun nodTwoNumbers(number1: Int, number2: Int, nod: Int = (if (number1 > number2) number2 else number1)) : Int =  
 when  
 {  
 (number1 == 0 || number2 == 0) -> -1  
 (number1 % nod == 0 && number2 % nod == 0) -> nod  
 else -> *nodTwoNumbers*(number1, number2, nod - 1)  
 }  
  
// Сумма простых цифр числа  
fun sumOfSimpleDigitsOfNumber(number: Int, sum: Int = 0) : Int =  
 when  
 {  
 (number == 0) -> sum  
 *simpleNumber*(number % 10) -> *sumOfSimpleDigitsOfNumber*(number / 10, sum + (number % 10))  
 else -> *sumOfSimpleDigitsOfNumber*(number / 10, sum)  
 }  
  
// Основное подзадание 3  
fun task8method3(number: Int, del: Int = number - 1, count: Int = 0) : Int =  
 when  
 {  
 (del == -1 || del == 0) -> 0  
 (del == 1) -> count  
 (  
 number % del != 0 &&  
 *nodTwoNumbers*(number, del) != 1 &&  
 *nodTwoNumbers*(*sumOfSimpleDigitsOfNumber*(number), del) == 1  
 ) -> *task8method3*(number, del - 1, count + 1)  
 else -> *task8method3*(number, del - 1, count)  
 }  
  
fun continueMode() : String  
 {  
 *print*("Continue? y/n -> ")  
 return when (*readLine*()) {  
 "y" -> "mainRelease"  
 "n" -> "return"  
 else -> *continueMode*()  
 }  
 }  
  
fun mainRelease()  
{  
 *print*("Number -> ")  
 try {  
 val number = *readLine*()?.*toInt*() ?: 0  
 *print*("Method 1/2/3 -> ")  
 when(*readLine*()?.*toInt*() ?: 1)  
 {  
 1 -> *println*("Сумма непростых делителей числа $number: ${*sumOfNoSimpleDelOfNumber*(number)}")  
 2 -> *println*("Количество цифр числа $number, меньших 3: ${*countDigitsLess3*(number)}")  
 3 -> *println*("Количество чисел по заданию 3: ${*task8method3*(number)}")  
 else -> *println*("Такого метода нет в списке")  
 }  
 }  
 catch (ex: NumberFormatException)  
 {  
 *println*("Некорректный ввод")  
 }  
  
 when(*continueMode*())  
 {  
 "mainRelease" -> *mainRelease*()  
 "return" -> return  
 }  
}  
  
fun main(args: Array<String>)  
{  
 *mainRelease*()  
}

**Скрины:**

