**Практическая работа по Kotlin 1**

**Ввод, вывод данных через консоль. Работа со структурой ветвления.**

Теоретический материал

**Вывод на консоль**

Для вывода информации на консоль в Kotlin есть две встроенные функции:

fun main() {

     print("Hello ")

    print("Kotlin ")

    print("on Metanit.com")

    println()

    println(a)

}

**Ввод с консоли**

Для ввода с консоли применяется встроенная функция **readLine()**. Она возвращает введенную строку. Стоит отметить, что результат этой функции всегда представляет объект типа **String**. Соответственно введённую строку мы можем передать в переменную типа String:

|  |  |
| --- | --- |
|  | fun main() {        print("Введите имя: ")      val name = readLine()        println("Ваше имя: $name")  }  **Для ввода в клавиатуры числа, используем конвертацию символа в число**  **val** z= *readLine*().*toString*().*toInt*()  **Переменные**  Неизменяемые (только для чтения) локальные переменные определяются с помощью ключевого слова val. Присвоить им значение можно только один раз.  **val** a: Int = 1 *// Инициализация при объявлении*  **val** b = 1 *// Тип `Int` определен автоматически*  **val** c: Int *// Указывать тип обязательно, если переменная не инициализирована сразу*  c = 1 *// Последующее присвоение*  Изменяемые переменные объявляются с помощью ключевого слова var.  **var** x = 5 *// Тип `Int` определен автоматически*  x += 1  Вы можете объявлять глобальные переменные.  **val** PI = 3.14  **var** x = 0  **fun** incrementX() {  x += 1  } |

**Типы данных**

Все переменные, инициализированные целыми значениями, не превышающими максимальное значение Int, имеют предполагаемый тип Int. Если начальное значение превышает это значение, то тип Long. Чтобы явно указать тип Long, добавьте после значения L.

**val** one = 1 *// Int*

**val** threeBillion = 3000000000 *// Long*

**val** oneLong = 1L *// Long*

**val** oneByte: Byte = 1

Вы можете инициализировать переменные Double и Float числами, имеющими дробную часть. Она должна быть отделена от целой части точкой (.). Для переменных, инициализированных дробными числами, компилятор автоматически определяет тип Double.

**val** pi = 3.14 *// Double*

*// val one: Double = 1 // Ошибка: несоответствие типов*

**val** oneDouble = 1.0 *// Double*

Тип Boolean представляет логический тип данных и принимает два значения: true и false.

При необходимости использования nullable-ссылок логические переменные оборачиваются Boolean?.

Встроенные действия над логическими переменными включают:

* || – ленивое логическое *ИЛИ*
* && – ленивое логическое *И*
* ! – отрицание

**val** myTrue: Boolean = **true**

**val** myFalse: Boolean = **false**

**val** boolNull: Boolean? = **null**

println(myTrue || myFalse)

println(myTrue && myFalse) println(!myTrue)

**Функции**

Функция принимает два аргумента Int и возвращает Int.

**fun** sum(a: **Int**, b: **Int**): Int {

**return** a + b

}

В качестве тела функции может выступать выражение. Тогда тип возвращаемого значения определяется автоматически.

**fun** sum(a: **Int**, b: **Int**) = a + b

Функция, не возвращающая никакого значения (void в Java).

**fun** printSum(a: **Int**, b: **Int**): Unit {

println(**"сумма $a и $b равна** ${a + b}**"**)

}

Тип возвращаемого значения Unit может быть опущен.

**fun** printSum(a: **Int**, b: **Int**) {

println(**"сумма $a и $b равна** ${a + b}**"**)

}

<https://youtu.be/TrT02ZcU754>

**Ветвление**

fun max(m: Int, n: Int) = if (m > n) m else n

**if..else** — *оператор* или *конструкция* ветвления, переводится с английского как **если..иначе**. После ключевого слова **if** в скобках следует *условие* ветвления m > n. Если условие *истинно*, в качестве результата используется выражение сразу за условием ветвления, в данном случае m. Если же условие *ложно*, используется выражение за ключевым словом **else**, в данном случае n. Конструкцию можно прочитать по-русски как «Если m > n, (то) m, иначе n». Функция max, аналогичная данной, имеется в стандартной библиотеке Kotlin.

**Операции сравнения**

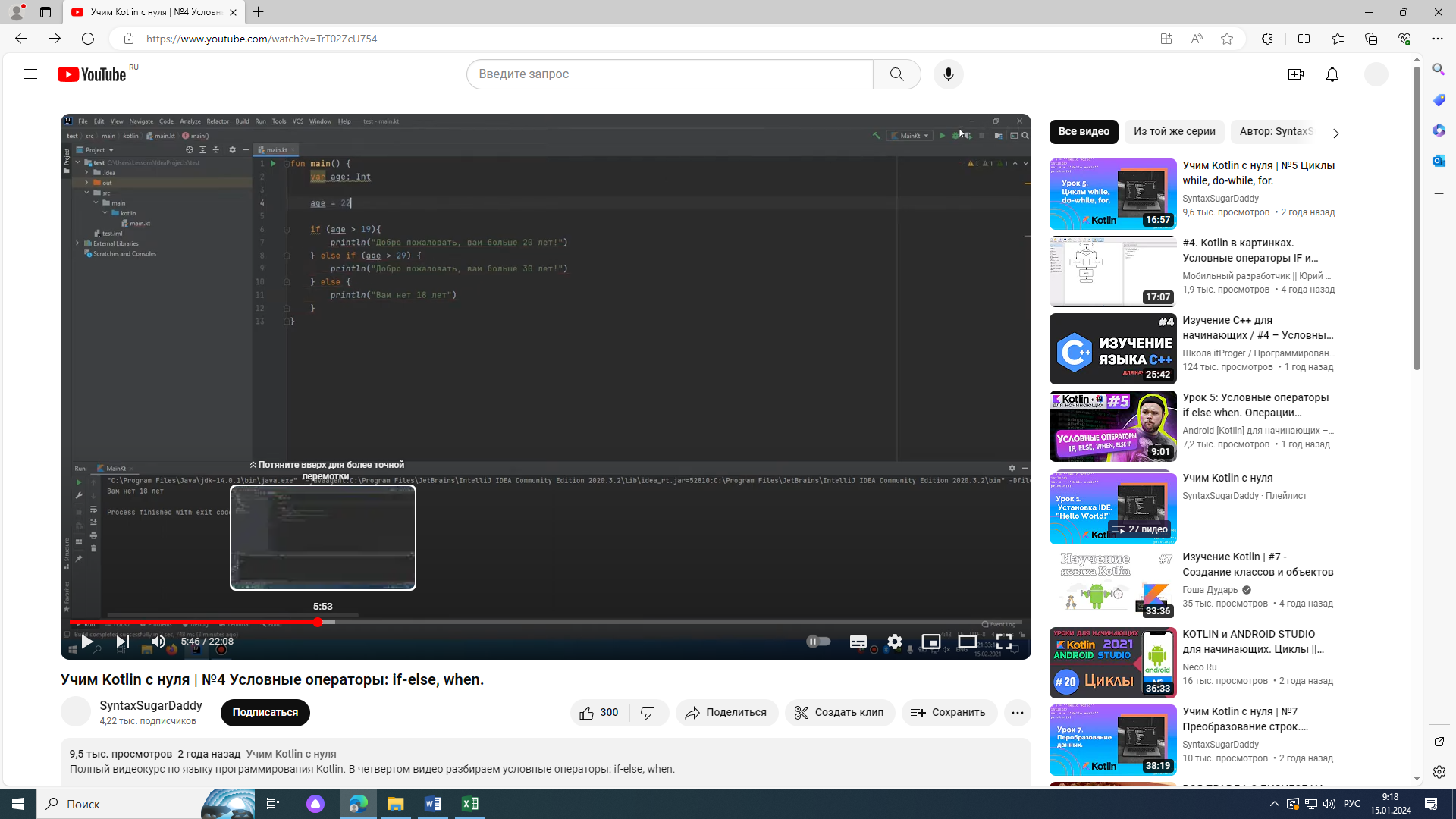
В обоих примерах в условиях мы использовали *операции сравнения*. Таких операций имеется восемь:

* > строго больше;
* >= больше или равно, аналог математического ≥
* < строго меньше;
* <= меньше или равно, аналог математического ≤
* x in a..b — x принадлежит интервалу от a до b, аналог математического a ≤ x ≤ b;
* x !in a..b — x НЕ принадлежит интервалу от a до b;
* != не равно, аналог математического ≠
* == равно (используется два знака равенства, чтобы не путать данную операцию с инициализацией / вычислением результата =).

Операции == и != в Котлине применимы для сравнения аргументов произвольных типов. В частности, разрешается сравнивать на равенство строки — они равны, если имеют равную длину, и соответствующие их символы совпадают: «abc» != «cba».

Остальные операции применимы только в том случае, если для аргументов определена специальная функция сравнения compareTo — об этом мы будем говорить позже. Первое время, нам придётся применять их только для числовых типов.

Математически, результат всех операций сравнения имеет тип Boolean с ровно двумя возможными значениями: **true**, **false**.



**Табличная форма ветвлений (when)**

Каскадную запись **if..else if..else** часто можно представить более изящно в табличной форме, используя конструкцию when (когда). Для примера нужно определить, равно ли число нулю или нет. это делается так:

fun main() {

    val number = readLine()

    when (number) {

0 -> println("Ноль")

1 -> println("Ноль")

…….

else -> println("Не-ноль")

    }

}

**Математические функции**

В библиотеке kotlin.math определено большое количество математических *функций*, предназначенных для выполнения более сложных операций.

Поскольку готовых функций существует очень много, они разбиты на так называемые *пакеты* и *классы* внутри пакетов например kotlin.math.sqrt (f).

Примеры функций из kotlin.math:

* abs(x: Int) или abs(x: Double) — модуль;
* sqrt(x: Double) — квадратный корень;
* Double.pow(x: Double) — возведение в степень xy, используется как x.pow(y);
* sin(x: Double) — синус, cos(x: Double) — косинус, tan(x: Double) — тангенс, все три функции считают, что x задан в радианах;
* exp(x: Double) — экспонента ex;
* log(x: Double), log10(x: Double) — соответственно натуральный и десятичный логарифм;
* min(x: Int, y: Int) или min(x: Double, y: Double) — минимум из двух чисел;
* max(x: Int, y: Int) или max(x: Double, y: Double) — максимум из двух чисел.

В том же пакете kotlin.math определены константы PI = 3.14…​ и E = 2.718…​

Разрешение использовать короткие имена для ВСЕХ функций из пакета kotlin.math

import kotlin.math.\*

**Задания:**

1. Напишите программу, которая запросит и затем выведет на экран ваше Имя , Фамилию, отчество, группу в которой вы учитесь, специальность и ФИО куратора
2. Найти минимум из двух введенных с клавиатуры чисел
3. Найти среднее значение трех введенных с клавиатуры чисел
4. Вывести расшифровку названия специальности по ее коду

09.02.07 – Информационные системы и программирование

09.02.01 – Системное администрирование

09.02.03 Программное обеспечение вычислительной техники и автоматизированных систем

09.02.04 – Информационные системы

1. Найти число корней квадратного уравнения ax^2 + bx + c = 0
2. \*\* Найти корни биквадратного уравнения ax^4 + bx^2 + c = 0

алгоритм решения задачи в виде последовательности действий:

1. Если a равно 0, уравнение вырождается в bx2 + c = 0. Вырожденное уравнение:

* при b равном 0 не имеет решений (или имеет бесконечно много)
* при c / b > 0 также не имеет решений
* в противном случае минимальный корень — это x = -sqrt(-c / b)

1. Рассчитаем дискриминант d = b2 - 4ac.
2. Если d меньше 0, у квадратного уравнения нет решений, как и у биквадратного.
3. В противном случае найдём корни квадратного уравнения y1 = (-b + sqrt(d))/(2a) и y2 = (-b - sqrt(d))/(2a).
4. Вычислим y3 = Max(y1, y2).
5. Если y3 < 0, у уравнения x2 = y3 нет решений.
6. В противном случае, минимальный корень биквадратного уравнения — это x = -sqrt(y3).

Запишем теперь то же самое на Котлине. Для обозначения ситуации, когда решений нет, будем использовать специальную константу Double.NaN, так называемое не-число. На практике она может получиться как результат некоторых некорректных действий с вещественными числами, например, после вычисления квадратного корня из -1.