



## Лекция #2. Основные операции в Kotlin

### Комментарии в коде на Kotlin

Компилятор **Kotlin** генерирует байт-код или исполняемый код из вашего исходного кода. Для этого он использует подробный набор правил. Иногда эти детали не показывают общую картину того, почему код написан определенным образом или даже какую проблему вы решаете. Чтобы этого не произошло, желательно задокументировать то, что вы написали, чтобы другой человек смог разобраться в вашей работе. В конце концов, этим человеком можете стать вы в будущем.

**Kotlin**, как и большинство других языков программирования, позволяет документировать код с помощью так называемых комментариев. Это позволяет вам писать любой текст рядом с вашим кодом, и данный текст игнорируется компилятором.

Первый способ написать комментарий на **Kotlin** выглядит следующим образом:

```
1 // Это комментарий. Он не выполняется.
```

Это комментарий в одну строку.

Таким образом, можно написать несколько комментариев с новой строки:

```
1 // Это тоже комментарий.  
2 // На несколько строчек.
```

Однако есть более удобный способ написания многострочных комментариев в **Kotlin**, к примеру:

```
1  /* Это также комментарий.
2     На много...
3     много...
4     много строчек. */
```

Это многострочный комментарий. Он начинается с **/\*** и заканчивается **\*/**.

В **Kotlin** также можно создавать вложенные комментарии:

```
1  /* Это комментарий.
2
3     /* И внутри него
4     находится
5     другой комментарий.
6     */
7
8     Вернемся к первому.
9     */
```

Это может показаться не особенно интересным, но, если вы работали с другими языками программирования, вы должны понимать важность правильного использования комментариев.

Многие языки не позволяют писать вложенные комментарии, так как когда язык видит первый **\*/**, он думает, что вы закрываете первый комментарий. Вы должны использовать комментарии там, где это необходимо — чтобы задокументировать код, объяснить свои рассуждения или просто оставить шутки для коллег.

## Арифметические операции в Kotlin

Проще всего разобраться с операциями можно с помощью арифметики. Операция сложения берет два числа и преобразует их в сумму. Операция вычитания берет два числа и преобразует их в разность. Во всех приложениях вы найдете простую арифметику — от подсчета количества лайков до расчета правильного размера и положения кнопки или окна.

## Простые математические операции в Kotlin

Все операции в **Kotlin** используют символ, известный как оператор, для указания типа осуществляемой операции. Рассмотрим четыре арифметические операции, с которыми вы знакомы: сложение, вычитание, умножение и деление.

Для данных операций в **Kotlin** используются следующие операторы:

- Сложение: **+**
- Вычитание: **-**
- Умножение: **\***

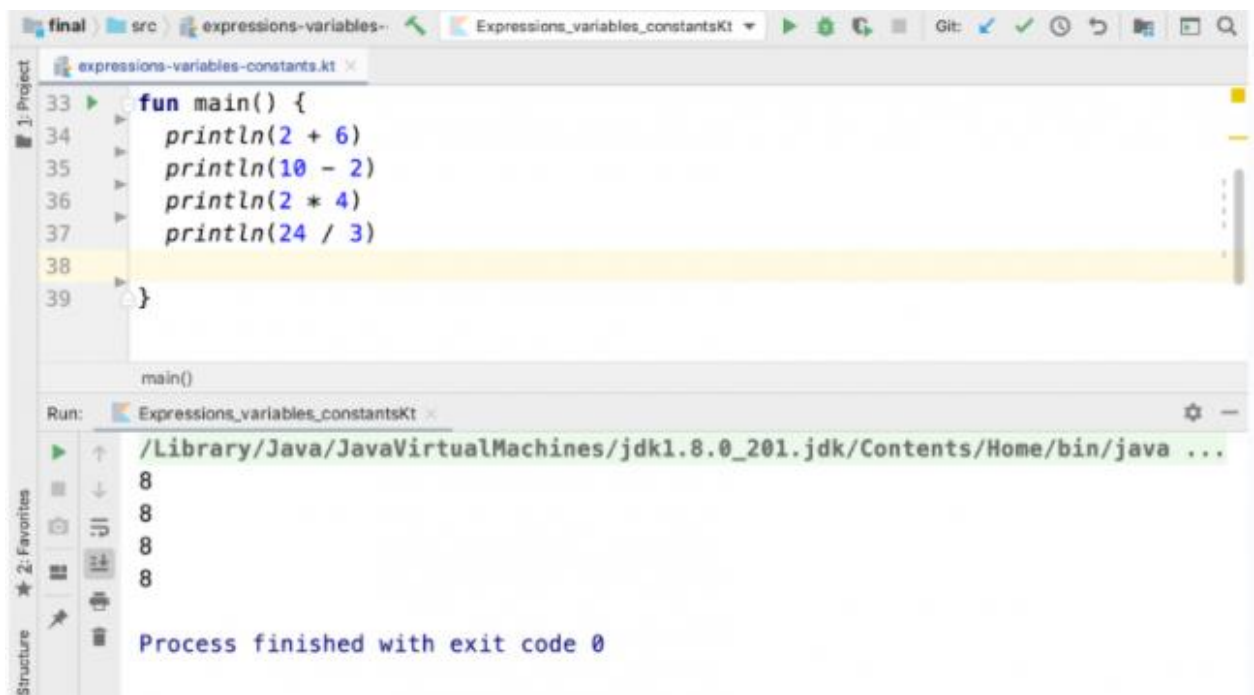
- Деление: /

Данные операторы используются следующим образом:

```
1 2 + 6
2 10 - 2
3 2 * 4
4 24 / 3
```

Каждая из данных строчек является выражением. У выражения есть значение. В данных случаях у всех четырех выражений одинаковые значения: 8. Вы набираете код для выполнения операций по аналогии с написанием примеров на бумаге.

В **IDE** вы увидите значения выражений в выводе на консоли с помощью команды **println()**:



The screenshot shows an IDE window with a file named `expressions-variables-constants.kt`. The code defines a `main()` function that prints the results of four arithmetic operations: `2 + 6`, `10 - 2`, `2 * 4`, and `24 / 3`. Below the code editor, the Run console shows the output of the `main()` function, which is four lines of the number `8`, corresponding to the results of the operations. The console also shows the Java command used to run the program and a message indicating that the process finished with exit code 0.

```
fun main() {
    println(2 + 6)
    println(10 - 2)
    println(2 * 4)
    println(24 / 3)
}
```

```
Run: Expressions_variables_constantsKt
/Library/Java/JavaVirtualMachines/jdk1.8.0_201.jdk/Contents/Home/bin/java ...
8
8
8
8
Process finished with exit code 0
```

Если хотите, можете убрать пробелы возле оператора:

```
1 2+6
```

Разницы нет. Вы можете даже использовать варианты с пробелами и без вперемежку. К примеру:

```
1 2+6 // OK
2 2 + 6 // OK
3 2 +6 // OK
4 2+ 6 // OK
```

Однако зачастую удобнее читать выражения с пробелами возле оператора.

## Десятичные числа в Kotlin

Во всех операциях выше использовались целые числа, которые относятся к типу **integer**. Однако не все числа целые.

Рассмотрим следующий пример:

```
1 | 22 / 7
```

Результатом данной операции будет число 3. Если вы используете в выражении только целые числа, **Kotlin** сделает результат также целым числом. В данном случае результат округляется до ближайшего целого числа.

Вы можете указать **Kotlin**, что нужно использовать десятичные числа, записав их в следующей форме:

```
1 | 22.0 / 7.0
```

На этот раз результатом будет число 3.142857142857143.

## Операция для получения остатка % в Kotlin

Четыре операции, с которыми мы работали до сих пор, легко понять, потому что вы с ними хорошо знакомы в жизни. В **Kotlin** также есть более сложные операции, которые вы можете использовать. Это стандартные математические операции, только менее распространенные. Рассмотрим их.

Сначала разберем операцию остатка, или операцию по модулю. При делении числителя на знаменатель — результатом будет целое число, а также остаток. Этот остаток и является результатом операции по модулю. Например, 10 по модулю 3 равно 1, потому что 3 трижды переходит в 10 с остатком 1.

В **Kotlin** оператором остатка является символ **%**, который используется следующим образом:

```
1 | 28 % 10
```

В данном случае результатом будет число 8, которое является остатком при делении 28 на 10. Если вам нужно посчитать то же самое, используя десятичные числа, это можно сделать следующим образом:

```
1 | 28.0 % 10.0
```

Результат идентичен **%** без десятичных чисел, что можно увидеть на выводе при использовании указателя формата:

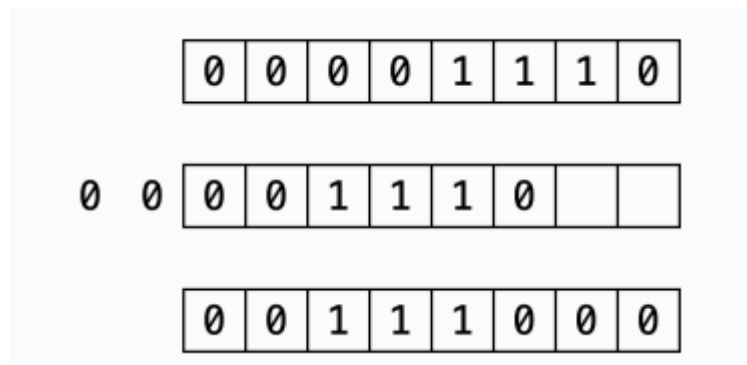
```
1 | println("%.0f".format(28.0 % 10.0))
```

## Операции смещения в Kotlin

Операции правого и левого смещения (Shift) принимают двоичную форму десятичного числа и сдвигают цифры налево или направо соответственно. Затем они возвращают десятичную форму нового двоичного числа.

К примеру, десятичное число 14 в бинарном виде будет состоять из восьми цифр — 00001110. Сдвиг на два пункта влево приведет к числу 00111000, которое равно 56 в десятичной системе.

Далее дана схема того, что происходит во время операции смещения:



Цифры, которые приходят на замену пустым местам, становятся 0. Отсеченные числа будут потеряны. Сдвиг вправо аналогично, но цифры сдвигаются в правую часть. Функции **Kotlin** для данных двух операций являются следующими:

- Сдвиг влево: **shl**
- Сдвиг вправо: **shr**

Это **инфиксные** функции, которые помещаются между операндами, чтобы вызов функции выглядел как операция:

```
1 | 1 shl 3
2 |
3 | 32 shr 2
```

Оба значения равны 8.

Одна из причин использования сдвигов — облегчить умножение или деление на числа во второй степени. Обратите внимание, что сдвиг влево на единицу — это то же самое, что умножение на два, сдвиг влево на два — это то

же самое, что умножение на четыре и так далее. Со смещением вправо ситуация аналогична.

Раньше код часто использовал этот трюк, потому что сдвиг битов для CPU намного проще, чем сложная арифметика умножения и деления. Следовательно, код был быстрее при использовании смещения. Однако в наши дни процессоры намного быстрее, и компиляторы могут даже преобразовывать умножение и деление на числа во второй степени в сдвиги.

## Порядок выполнения операций в Kotlin

Высока вероятность, что при вычислении значений вам нужно будет использовать много различных операций. Далее дан пример того, как это можно сделать в Kotlin:

```
1 | ((8000 / (5 * 10)) - 32) shr (29 % 5)
```

Обратите внимание на использование скобок, у которых в **Kotlin** две цели: сделать код понятным для чтения и устранить неоднозначность.

Рассмотрим пример:

```
1 | 350 / 5 + 2
```

*Результат будет равен 72 (350 делится на 5, а потом прибавляется 2) или 50 (350 делится на 7)?*

**Kotlin** использует так называемый **приоритет операторов**. Оператор деления (/) имеет более высокий приоритет, чем оператор сложения (+), поэтому в этом примере код сначала выполняет операцию деления.

Если вы хотите, чтобы **Kotlin** сначала выполнял сложение, то есть возвращал 50, вы можете использовать круглые скобки, например:

```
1 | 350 / (5 + 2)
```

Правила приоритета аналогичны тому, что вы изучали в курсе математики. У умножения и деления более высокий, чем у сложения и вычитания.