

Переменные и базовые классы данных в Kotlin

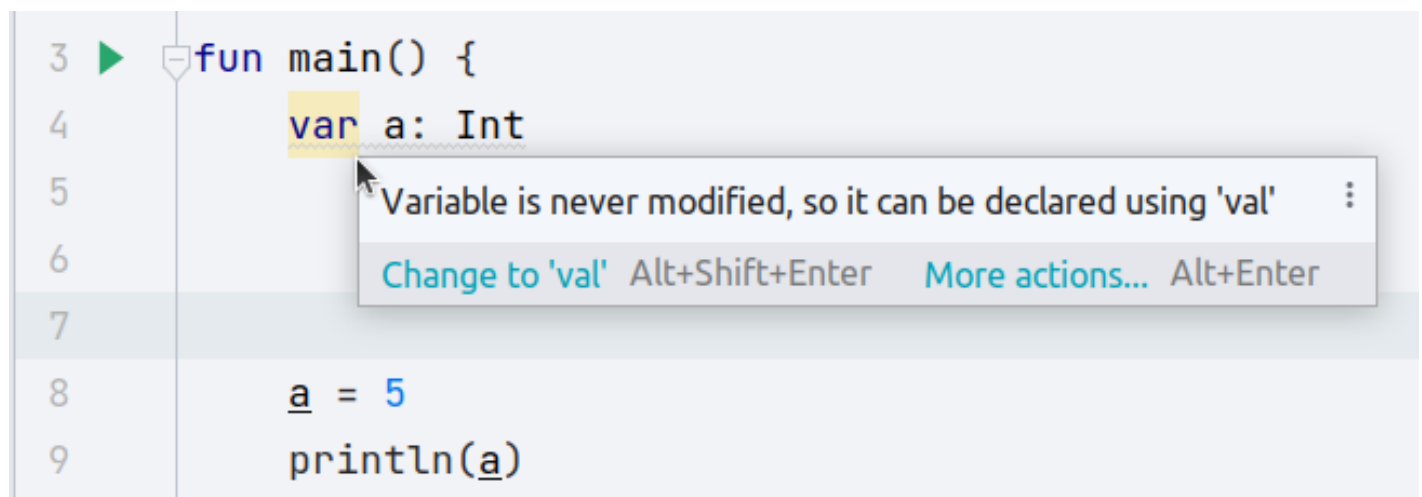
Переменные в Kotlin объявляются с помощью ключевых слов **var** или **val**, за которыми идет имя переменной, через двоеточие указывается ее тип – класс данных.

val
var **ИМЯ : ТИП**

Переменной, объявленной через **val**, значение в программе можно присвоить только один раз. В дальнейшем значение можно только читать, val-переменные являются неизменяемыми.

```
fun main() {  
    var a: Int  
  
    a = 5  
    println(a)  
}
```

Приведенный выше код правильный, но IntelliJ будет подсказывать, что его можно улучшить. Поскольку в программе мы не меняем значение переменной, ее следовало бы объявить через **val**.



```
10 }

Run: Exm1Kt x
/usr/lib/jvm/java-19-openjdk-amd64/bin/java -javaagent
5
Process finished with exit code 0
```

Однако поступим иначе. В процессе выполнения программы будем менять значение переменной:

```
fun main() {
    var a: Int

    a = 5
    println(a)

    a = a + 10
    println(a)
}
```

Тут тоже не все гладко. IntelliJ советует совместить объявление переменной с присвоением ей значения.

```
fun main() {
    var a: Int
    a = 5
    println(a)

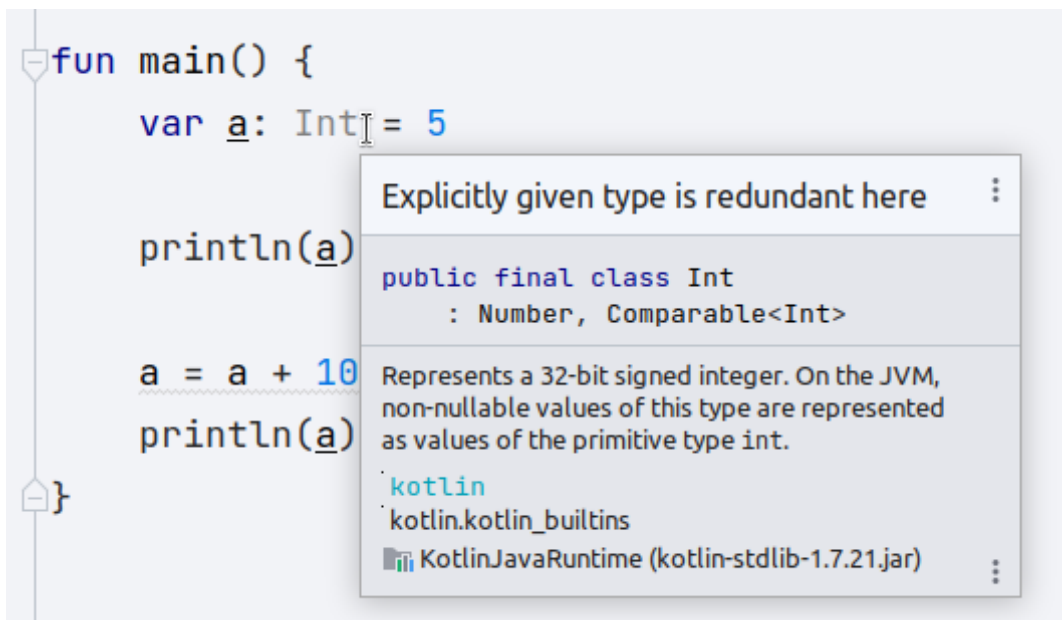
    a = a + 10
    println(a)
}
```

Can be joined with assignment
Join declaration and assignment Alt+Shift+Enter More actions... Alt+Enter
var a: Int
KotlinBaseCode

Действительно, это уместно в данном случае. Ведь мы, например, не запрашиваем значение с ввода после выполнения каких-либо предварительных действий.

```
fun main() {  
    var a: Int = 5  
  
    println(a)  
  
    a = a + 10  
    println(a)  
}
```

Теперь IntelliJ советует избавиться от **Int**, так как по значению и так понятен тип переменной.



То есть Kotlin способен вывести тип переменной из присвоенных ей данных. Это называется автоматическим определением типов. Поэтому приводим код к такому виду:

```
fun main() {  
    var a = 5  
  
    println(a)  
  
    a = a + 10  
}
```

```
println(a)
}
```

Это не значит, что переменная стала без типа. Kotlin – статически типизированный язык. Просто тип переменной был выведен, исходя из ее значения. Чтобы увидеть тип переменной в IntelliJ, надо установить на нее курсор и нажать **Ctrl + Shift + P**.

Осталось последнее замечание. IntelliJ считает, что добавление значения к переменной лучше записывать в сокращенной форме:

```
fun main() {
    var a = 5

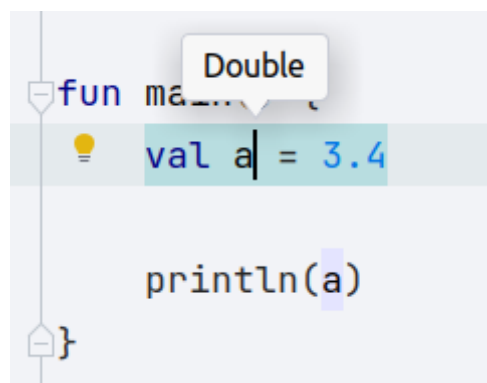
    println(a)

    a += 10
    println(a)
}
```

В качестве вещественного типа по умолчанию используется класс **Double**:

```
fun main() {
    val a = 3.4

    println(a)
}
```



В данном случае тип переменной автоматически определится как Double. В Kotlin есть и другие числовые типы помимо `Double` и `Int` – это `Float`, `Long`, `Short`, `Byte`.

Булевый тип:

```
fun main() {
    val a = true
    val b = false
    var c: Boolean = a > b

    println(c) // true

    c = a <= b
    println(c) // false
}
```

В Kotlin есть как строковый, так и символьный типы:

```
fun main() {
    val s = "Hello"
    val s2: String
    val c = 'W'
    val c2: Char = s[0]

    s2 = s + c

    println(s2) // HelloW
    println(c2) // H
}
```

В Kotlin есть классы коллекций – списки (List), словари (Map), множества (Set).

Константы

Переменные, объявленные через `val`, – не константы. Ведь им не обязательно сразу присваивать значение. Это происходит в процессе выполнения программы. Например:

```
fun main() {  
    val a: String = readln()  
  
    println(a)  
}
```

Здесь более очевидно, что до компиляции значение переменной `a` неизвестно. Присвоение значения происходит в процессе выполнения программы, и значением может быть любая строка.

С другой стороны, в Kotlin есть, так сказать, настоящие константы – константы времени компиляции. Их значение известно заранее, "вшивается" в программу в момент ее компиляции.

Объявление и определение констант происходит за пределами функции. Потому что код внутри функций выполняется при исполнении программы, а сигнатуры функций и полей должны быть известны на момент компиляции.

```
const val N: Int = 10  
  
fun main() {  
    println(N)  
}
```

Модификатор **const** не получится использовать внутри функции. Это не значит, что все переменные за пределами функции должны быть константами. Там могут быть и обычные переменные. Очевидно, сочетания `const var` быть не может.

Константы могут принадлежать только числовым типам, строковому, символьному, а также **Boolean**.

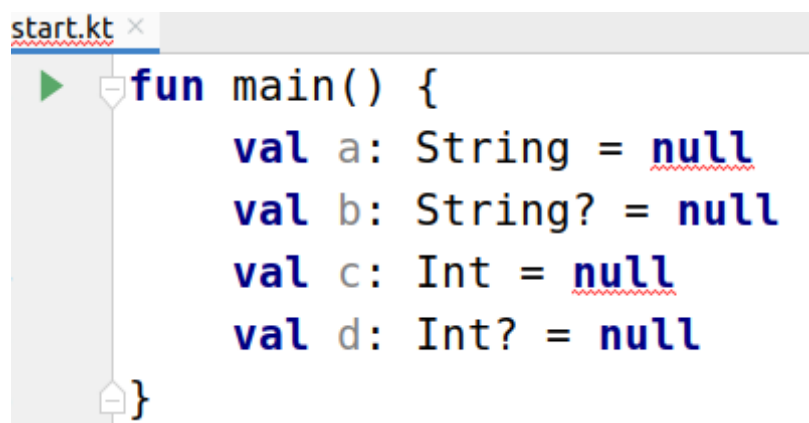
Nullable-типы

В Kotlin обычным типам нельзя присвоить литерал **null** – значение, говорящее, что переменная не связана с обычным объектом. Подобное ограничение в Kotlin призвано обеспечить nullable-безопасность, то есть таким образом избегается ряд ошибок при выполнении программы. Например, определенная функция не предполагает, что ей передадут **null**, и не умеет обрабатывать это значение, но

получает `null`. В этом случае в процессе выполнения программы будет выброшено исключение.

Во избежании подобного, обработка потенциально возможного `null` перекладывается на плечи программиста в принудительном порядке. Либо он напишет функцию так, чтобы она могла принимать и обрабатывать `null`, либо проверит значение на неравенство `null` до его передачи в качестве аргумента в функцию.

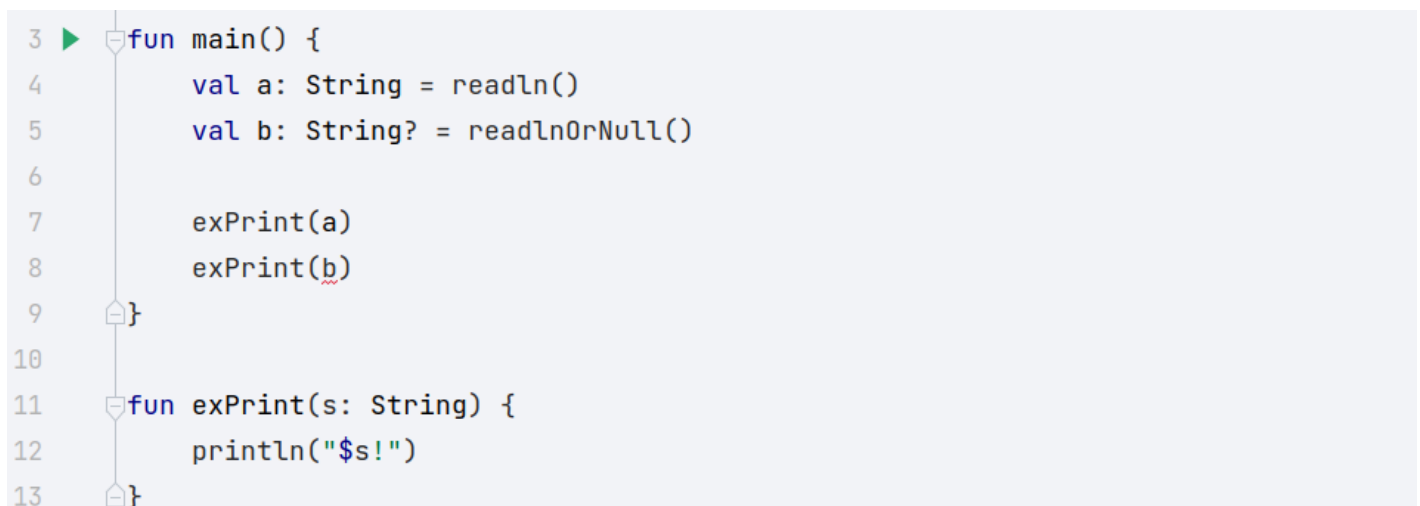
С другой стороны, в Kotlin у каждого типа без поддержки `null` имеется аналогичный ему с поддержкой null-значения. При объявлении типа, второй отличается от первого знаком вопроса в конце. Например, есть два строковых типа: `String` – без поддержки `null` и `String?` с поддержкой `null`.



```
start.kt x
fun main() {
    val a: String = null
    val b: String? = null
    val c: Int = null
    val d: Int? = null
}
```

На скрине показано, что попытка присвоить `null` переменным `a` и `c` приводит к ошибке. В то же время, для nullable-переменных это позволительно, хотя не обязательно. Их значениями также могут быть числа и строки, как у обычных переменных.

При этом в функцию, параметр которой не поддерживает `null`, передать nullable-аргумент нельзя. Компилятор Котлина не даст скомпилировать такую программу.



```
3 fun main() {
4     val a: String = readln()
5     val b: String? = readlnOrNull()
6
7     exPrint(a)
8     exPrint(b)
9 }
10
11 fun exPrint(s: String) {
12     println("$s!")
13 }
```

14

15

```
/home/pl/IdeaProjects/KotlinBaseCode/src/main/kotlin/_1_data_type/nullable_2.kt:8:13
```

```
Kotlin: Type mismatch: inferred type is String? but String was expected
```

Встроенные функции `readln()` и `readlnOrNull()` возвращают строки, прочитанные с ввода. Функция `readlnOrNull()` потенциально может вернуть не только строку, но и значение `null`. Поэтому возвращаемый из нее тип `String?`. Переменную типа `String?` мы не можем передать в нашу функцию `exPrint()`, даже если значение этой переменной не будет `null`. Котлин – строго типизированный язык, поэтому `String?` и `String` для него – разные типы.

Для обработки nullable-переменных в Kotlin предусмотрено несколько особых операторов, которые будут рассмотрены в следующем уроке.

[PDF-версия курса с дополнительными уроками](#)