Условные выражения в Kotlin. Операторы if и when

Ветвление if-else

```
fun main() {
    val a = "String"

    if (a[3] == 'r') {
        println("Yes")
    }
    else {
        println(a[3]) // i
    }
}
```

Операторы сравнения в Kotlin: `>`, `<`, `>=`, `<=`, `==` (значения равны), `!=` (значения не равны), `===` (обе переменные ссылаются на один объект), `!==` (обе переменные ссылаются на разные объекты).

Логические операторы: `&&` – И, `||` – ИЛИ, `!` – НЕ. Также есть словесные `and`, `or`, `xor`, при использовании которых операнды (простые логические выражения) надо брать в скобки.

Особенностью условного оператора в Kotlin является то, что он является выражением. Это значит, возвращает результат. Другими словами, можно сделать так:

```
fun main() {
    val a = "String"

    val b = if (a[3] == 'r') {
        println("Yes")
    }
    else {
```

```
        println(a[3]) // i
    }

    println(b) // kotlin.Unit
}
```

Хотя правильно будет сделать так:

```
fun main() {
    val a = "String"

    val b = if (a[3] == 'r') {
        "Yes"
    }
    else {
        a[3].toString()
    }

    println(b) // i
}
```

Здесь переменной *b* присваивается строка из ветки `if` или ветки `else`. В ветке else мы приводим символьный тип данных к строковому. Иначе тип переменной *b* будет `Any`, код все равно скомпилируется, но с предупреждениями.

Если тело ветки состоит только из одного выражения, фигурные скобки можно опустить:

```
fun main() {
    val a = "String"

    val b = if (a[3] == 'r')
        "Yes"
    else
        a[3].toString()

    println(b) // i
}
```

Если тело ветки состоит из нескольких выражений, в качестве возвращаемого значения выступает последнее выражение тела. Так в примере ниже переменная *c* принимает значение, которое возвращает либо выражение `a + 1`, либо `b + 1`.

```
fun main() {
    val a = Math.random()
    val b = Math.random()

    val c = if (a > b) {
        println("a > b")
        a + 1
    }
    else {
        println("b > a")
        b + 1
    }

    println(c)
}
```

Пример выполнения кода:

```
b > a
1.8443752126032709
```

В случае использования оператора if-else в качестве возвращающего результат выражения, наличие ветки **`else`** является обязательным. Оно и понятно, переменной в любом случае надо что-то присвоить, даже если ветка **`if`** не сработала.

Условное выражение when

Оператор **`when`** в Kotlin – это аналог **`switch`**, который имеется в ряде языков программирования.

```
fun main() {
    val a = "String"

    when (a[3]) {
        'r' -> println("Yes")
        'i' -> println("Not 'r'")
    }
}
```

Или как выражение:

```
fun main() {
    val a = "String"

    val b = when (a[3]) {
        'r' -> "Yes"
        else -> "Not 'r'"
    }

    println(b)
}
```

В случае использования `when` в качестве выражения ветка `else` почти всегда является обязательной, за исключением тех случаев, когда для компилятора очевидно, что все значения учтены.

```
fun main() {
    val a = (Math.random() * 3).toInt()

    println(a)

    val b = when(a) {
        0 -> "Zero"
        1 -> "One"
        else -> "Two"
    }
```

```
    println(b)
}
```

В этом примере нам очевидно, что *a* не может быть больше 2-х, компилятору – нет. Поэтому приходится использовать ветку `else` (а не число 2), что делает код менее наглядным. С другой стороны,

```
enum class Numbers {
    ZERO, ONE, TWO
}

fun main() {
    val c = Numbers.ONE

    val d = when(c) {
        Numbers.ZERO -> 0
        Numbers.ONE -> 1
        Numbers.TWO -> 2
    }

    println(d)
}
```

в этом примере компилятор знает, что в `when` были перечислены все возможные значения переменной *c*.

Если тело ветки состоит из нескольких выражений, оно заключается в фигурные скобки.

```
fun main() {
    when (readln()) {
        "Hello" -> {
            println("Hello")
            println("World")
            println("!!!")
        }
        else -> {
            println("Hi")
```

```
            println("?")
        }
    }
}
```

До стрелки можно указывать не только отдельные значения, также несколько, комбинируя их через запятую. Могут использоваться различные выражения.

```
fun main() {
    when ((Math.random() * 100).toInt()) {
        in 0..10, in 90..99 -> println("Bad")
        in 11..50, in 60..89 -> println("Good")
        in 51..59 -> println("Excellent")
    }
}
```

Множественное ветвление

В Kotlin также как в других языках можно вкладывать if-else во внешнюю для них ветку `if` или, как это бывает обычно, `else`, реализуя тем самым множественное ветвление. Однако для этих целей в большинстве случаев лучше подходит оператор `when`. При этом он используется без аргумента в заголовке.

```
fun main() {
    val a = (Math.random() * 100).toInt()

    when {
        a < 10 || a > 90 -> println("Bad")
        a in 51..59 -> println("Excellent")
        else -> println("Good")
    }
}
```

PDF-версия курса с дополнительными уроками