

Условные выражения

Условные выражения представляют некоторое условие, которое возвращает значение типа Boolean: либо true (если условие истинно), либо false (если условие ложно).

Операции отношения

- (больше чем): возвращает true, если первый операнд больше второго. Иначе возвращает false

```
1 fun main() {
2     val a = 11
3     val b = 12
4     val c: Boolean = a > b
5     println(c)    // false - a меньше чем b
6
7     val d = 35 > 12
8     println(d)    // true - 35 больше чем 12
9 }
```

- < (меньше чем): возвращает true, если первый операнд меньше второго. Иначе возвращает false

```
1 fun main() {
2     val a = 11
3     val b = 12
4     val c = a < b    // true
5
6     val d = 35 < 12  // false
7 }
```

Formatted 1 line

- >= (больше чем или равно): возвращает true, если первый операнд больше или равен второму

```
1 fun main() {
2     val a = 11
3     val b = 12
4     val c = a >= b    // false
5     val d = 11 >= a    // true
6 }
```

- <= (меньше чем или равно): возвращает true, если первый операнд меньше или равен второму.

```
1 fun main() {
2     val a = 11
3     val b = 12
4     val c = a <= b    // true
5     val d = 11 <= a    // false
6 }
```

- `==` (равно): возвращает `true`, если оба операнда равны. Иначе возвращает `false`

```
1  ▶ fun main() {  
2      val a = 11  
3      val b = 12  
4      val c = a == b      // false  
5      val d = b == 12     // true  
6  }
```

- `!=` (не равно): возвращает `true`, если оба операнда НЕ равны

```
1  ▶ fun main() {  
2      val a = 11  
3      val b = 12  
4      val c = a != b      // true  
5      val d = b != 12     // false  
6  }
```

Логические операции

Операндами в логических операциях являются два значения типа `Boolean`. Нередко логические операции объединяют несколько операций отношения:

- `and`: возвращает `true`, если оба операнда равны `true`.

```
1  ▶ fun main() {  
2      val a = true  
3      val b = false  
4      val c = a and b      // false  
5      val d = (11 >= 5) and (9 < 10) // true  
6      println(c)  
7      println(d)  
8  }
```

- `or`: возвращает `true`, если хотя бы один из операндов равен `true`.

```
1  ▶ fun main() {  
2      val a = true  
3      val b = false  
4      val c = a or b      // true  
5      val d = (11 < 5) or (9 > 10) // false  
6  }
```

- xor: возвращает true, если только один из операндов равен true. Если операнды равны, возвращается false

```
1  ▶ fun main() {  
2      val a = true  
3      val b = false  
4      val c = a xor b           // true  
5      val d = a xor (90 > 10)  // false  
6  }  
7
```

- !: возвращает true, если операнд равен false. И, наоборот, если операнд равен true, возвращается false.

```
1  ▶ fun main() {  
2      val a = true  
3      val b = !a // false  
4      val c = !b // true  
5  }
```

- В качестве альтернативы оператору ! можно использовать метод not():

```
1  ▶ fun main() {  
2      val a = true  
3      val b = a.not() // false  
4      val c = b.not() // true  
5  }
```

- in: возвращает true, если операнд имеется в некоторой последовательности.

```
1  ▶ fun main() {  
2      val a = 5  
3      val b = a in 1 ≤ .. ≤ 6    // true - число 5 входит в последовательность от 1 до 6  
4  
5      val c = 4  
6      val d = c in 11 ≤ .. ≤ 15  // false - число 4 НЕ входит в последовательность от 11 до 15  
7  }
```

Выражение 1..6 создает последовательность чисел от 1 до 6. И в данном случае оператор in проверяет, есть ли значение переменной a в этой последовательности. Поскольку значение переменной a имеется в данной последовательности, то возвращается true.

А выражение 11..15 создает последовательность чисел от 11 до 15. И поскольку значение переменной c в эту последовательность не входит, поэтому возвращается false.

Если нам, наоборот, хочется возвращать true, если числа нет в указанной последовательности, то можно применить комбинацию операторов !in:

```
1 fun main() {  
2     val a = 8  
3     val b = a !in 1 ≤ .. ≤ 6 // true - число 8 не входит в последовательность от 1 до 6  
4 }
```

Условные конструкции

Условные конструкции позволяют направить выполнение программы по одному из путей в зависимости от условия.

if...else

Конструкция if принимает условие, и если это условие истинно, то выполняется последующий блок инструкций.

```
1 fun main() {  
2     val a = 10  
3     if(a == 10) {  
4  
5         println("a равно 10")  
6     }  
7 }
```

В данном случае в конструкции if проверяется истинность выражения `a == 10`, если оно истинно, то выполняется последующий блок кода в фигурных скобках, и на консоль выводится сообщение "a равно 10". Если же выражение ложно, тогда блок кода не выполняется.

Если необходимо задать альтернативный вариант, то можно добавить блок else:

```
1 fun main() {  
2     val a = 10  
3     if (a == 10) {  
4         println("a равно 10")  
5     } else {  
6         println("a НЕ равно 10")  
7     }  
8 }  
9
```

Таким образом, если условное выражение после оператора if истинно, то выполняется блок после if, если ложно - выполняется блок после else.

Если блок кода состоит из одного выражения, то в принципе фигурные скобки можно опустить:

```

1  fun main() {
2      val a = 10
3      if (a == 10)
4          println("a равно 10")
5      else
6          println("a НЕ равно 10")
7  }
8

```

Если необходимо проверить несколько альтернативных вариантов, то можно добавить выражения **else if**:

```

1  fun main() {
2      val a = 10
3      if (a == 10) {
4          println("a равно 10")
5      } else if (a == 9) {
6          println("a равно 9")
7      } else if (a == 8) {
8          println("a равно 8")
9      } else {
10         println("a имеет неопределенное значение")
11     }
12 }

```

Возвращение значения из if

Стоит отметить, что конструкция **if** может возвращать значение. Например, найдем максимальное из двух чисел:

```

1  fun main() {
2      val a = 10
3      val b = 20
4      val c = if (a > b) a else b
5
6      println(c) // 20
7  }

```

Более подробно

```

1  fun main() {
2      val a = 10
3      val b = 20
4      val c = if (a > b) {
5          a
6      } else {
7          b
8      }
9      println(c) // 20
10 }

```

Если при определении возвращаемого значения надо выполнить еще какие-нибудь действия, то можно заключить эти действия в блоки кода:

```
1 fun main() {  
2     val a = 10  
3     val b = 20  
4     val c = if (a > b) {  
5         println("a = $a")  
6         a  
7     } else {  
8         println("b = $b")  
9         b  
10    }  
11 }
```

В конце каждого блока указывается возвращаемое значение.

Конструкция when

Конструкция when проверяет значение некоторого объекта и в зависимости от его значения выполняет тот или иной код. Конструкция when аналогична конструкции switch в других языках. Формальное определение:

```
16 when(объект){  
17  
18     значение1 -> действия1  
19     значение2 -> действия2  
20  
21     значениеN -> действияN  
22 }
```

Если значение объекта равно одному из значений в блоке кода when, то выполняются соответствующие действия, которые идут после оператора -> после соответствующего значения.

Например:

```
1 fun main() {  
2     val isEnabled = true  
3     when (isEnabled) {  
4         false -> println("isEnabled off")  
5         true -> println("isEnabled on")  
6     }  
7 }  
8
```

Здесь в качестве объекта в конструкцию `when` передается переменная `isEnabled`. Далее ее значение по порядку сравнивается со значениями в `false` и `true`. В данном случае переменная `isEnabled` равна `true`, поэтому будет выполняться код

```
println("isEnabled on")
```

Выражение `else`

В примере выше переменная `isEnabled` имела только два возможных варианта: `true` и `false`. Однако чаще бывают случаи, когда значения в блоке `when` не покрывают все возможные значения объекта. Дополнительное выражение `else` позволяет задать действия, которые выполняются, если объект не соответствует ни одному из значений. Например:

```
1 fun main() {  
2     val a = 30  
3     when (a) {  
4         10 -> println("a = 10")  
5         20 -> println("a = 20")  
6         else -> println("неопределенное значение")  
7     }  
8 }  
9
```

То есть в данном случае если переменная `a` равна 30, поэтому она не соответствует ни одному из значений в блоке `when`. И соответственно будут выполняться инструкции из выражения `else`.

Если надо, чтобы при совпадении значений выполнялось несколько инструкций, то для каждого значения можно определить блок кода:

```
1 fun main() {  
2     var a = 10  
3     when (a) {  
4         10 -> {  
5             println("a = 10")  
6             a *= 2  
7         }  
8  
9         20 -> {  
10            println("a = 20")  
11            a *= 5  
12        }  
13  
14        else -> {  
15            println("неопределенное значение")  
16        }  
17    }  
18    println(a)  
19 }
```

Сравнение с набором значений

Можно определить одни и те же действия сразу для нескольких значений. В этом случае значения перечисляются через запятую:

```
1  ▶ fun main() {  
2      val a = 10  
3      when (a) {  
4          10, 20 -> println("a = 10 или a = 20")  
5          else -> println("неопределенное значение")  
6      }  
7  }
```

Также можно сравнивать с целым диапазоном значений с помощью оператора in:

```
1  ▶ fun main() {  
2      val a = 10  
3      when (a) {  
4          in 10 ≤ .. ≤ 19 -> println("a в диапазоне от 10 до 19")  
5          in 20 ≤ .. ≤ 29 -> println("a в диапазоне от 20 до 29")  
6          !in 10 ≤ .. ≤ 20 -> println("a вне диапазона от 10 до 20")  
7          else -> println("неопределенное значение")  
8      }  
9  }
```

Если оператор in позволяет узнать, есть ли значение в определенном диапазоне, то связка операторов !in позволяет проверить отсутствие значения в определенной последовательности.

when и динамически вычисляемые значения

Выражение в when также может сравниваться с динамически вычисляемыми значениями:

```
1  ▶ fun main() {  
2      val a = 10  
3      val b = 5  
4      val c = 3  
5      when (a) {  
6          b - c -> println("a = b - c")  
7          b + 5 -> println("a = b + 5")  
8          else -> println("неопределенное значение")  
9      }  
10 }
```

Так, в данном случае значение переменной a сравнивается с результатом операций b - c и b + 5.

Кроме того, when также может принимать динамически вычисляемый объект:


```

1  ► fun main() {
2      val a = 10
3      val b = 20
4      when (a + b) {
5          10 -> println("a + b = 10")
6          20 -> println("a + b = 20")
7          30 -> println("a + b = 30")
8          else -> println("Undefined")
9      }
10 }
11

```

Можно даже определять переменные, которые будут доступны внутри блока when:

```

1  ► fun main() {
2      val a = 10
3      val b = 26
4      when (val c = a + b) {
5          10 -> println("a + b = 10")
6          20 -> println("a + b = 20")
7          else -> println("c = $c")
8      }
9  }

```

when как альтернатива для if..else

Причем в принципе нам необязательно вообще сравнивать значение какого-либо объекта. Конструкция when аналогично конструкции if..else просто может проверять набор условий и если одно из условий возвращает true, то выполнять соответствующий набор действий:

```

1  ► fun main() {
2      val a = 15
3      val b = 6
4      when {
5          (b > 10) -> println("b больше 10")
6          (a > 10) -> println("a больше 10")
7          else -> println("и a, и b меньше или равны 10")
8      }
9  }
10

```

Возвращение значения

Как и if конструкция when может возвращать значение. Возвращаемое значение указывается после оператора ->:

```
1 fun main() {
2     val sum = 1000
3     val rate = when (sum) {
4         in 100 ≤ .. ≤ 999 -> 10
5         in 1000 ≤ .. ≤ 9999 -> 15
6         else -> 20
7     }
8     println(rate) // 15
9 }
```

Таким образом, если значение переменной sum располагается в определенном диапазоне, то возвращается то значение, которое идет после стрелки.

Циклы

Циклы представляют вид управляющих конструкций, которые позволяют в зависимости от определенных условий выполнять некоторое действие множество раз.

For

Цикл for пробегается по всем элементам коллекции. В этом плане цикл for в Kotlin эквивалентен циклу for-each в ряде других языков программирования. Его формальная форма выглядит следующим образом:

```
13
14     for(переменная in последовательность){
15         выполняемые инструкции
16     }
```

Например, выведем все квадраты чисел от 1 до 9, используя цикл for:

```
1 fun main() {
2     for (n in 1 ≤ .. ≤ 9) {
3         print("${n * n} \t")
4     }
5 }
```

В данном случае перебирается последовательность чисел от 1 до 9. При каждом проходе цикла (итерации цикла) из этой последовательности будет извлекаться элемент и помещаться в

переменную `n`. И через переменную `n` можно манипулировать значением элемента. То есть в данном случае мы получим следующий консольный вывод:

```
MathKt x
C:\Users\Mamedov\.jdk\corretto-17.0.4.1\bin\java.exe ...
1  4  9  16  25  36  49  64  81
Process finished with exit code 0
```

Циклы могут быть вложенными. Например, выведем таблицу умножения:

```
1 fun main() {
2     for (i in 1..9) {
3         for (j in 1..9) {
4             print("${i * j} \t")
5         }
6         println()
7     }
8 }
9
```

```
Run: MathKt x
1  2  3  4  5  6  7  8  9
2  4  6  8 10 12 14 16 18
3  6  9 12 15 18 21 24 27
4  8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
Process finished with exit code 0
```

Цикл `while`

Цикл `while` повторяет определенные действия пока истинно некоторое условие:

```
1 fun main() {
2     var i = 10
3     while (i > 0) {
4         println(i * i)
5         i--
6     }
7 }
```

Здесь пока переменная `i` больше 0, будет выполняться цикл, в котором на консоль будет выводиться квадрат значения `i`.

В данном случае вначале проверяется условие (`i > 0`) и если оно истинно (то есть возвращает `true`), то выполняется цикл. И вполне может быть ситуация, когда к началу выполнения цикла условие не будет выполняться. Например, переменная `i` изначально меньше 0, тогда цикл вообще не будет выполняться.

Но есть и другая форма цикла `while` - `do..while`:

A screenshot of an IDE showing a Kotlin program. The code is as follows:

```
1 fun main() {  
2     var i = -1  
3     do {  
4         println(i * i)  
5         i--;  
6     } while (i > 0)  
7 }
```

The code is highlighted with a dark background and light text. The `while (i > 0)` part is highlighted with a yellow background.

В данном случае вначале выполняется блок кода после ключевого слова `do`, а потом оценивается условие после `while`. Если условие истинно, то повторяется выполнение блока после `do`. То есть несмотря на то, что в данном случае переменная `i` меньше 0 и она не соответствует условию, тем не менее блок `do` выполнится хотя бы один раз.

A screenshot of an IDE showing the output of the program. The output is as follows:

```
Run: MathKt x  
C:\Users\Mamedov\.jdk\corretto-17.0.4.1\bin\java.exe ...  
1  
Process finished with exit code 0
```

The output is displayed in a dark-themed window with a light background for the text.

Операторы `continue` и `break`

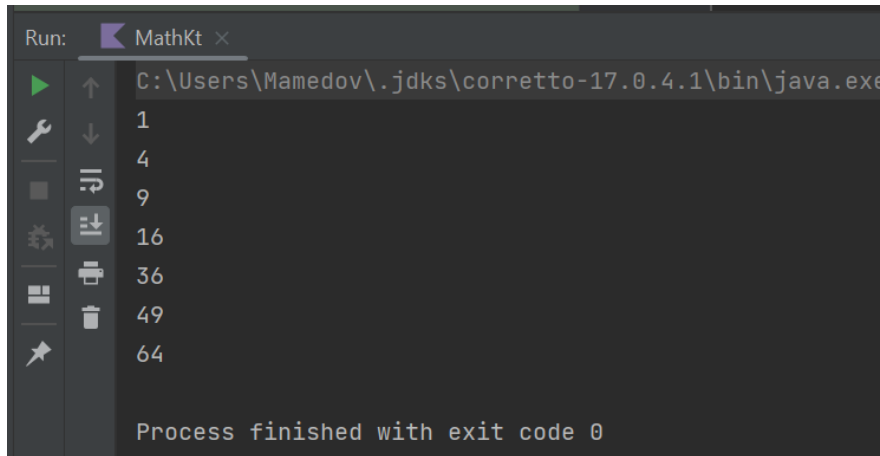
Иногда при использовании цикла возникает необходимость при некоторых условиях не дожидаться выполнения всех инструкций в цикле, перейти к новой итерации. Для этого можно использовать оператор `continue`:

A screenshot of an IDE showing a Kotlin program. The code is as follows:

```
1 fun main() {  
2     for (n in 1..8) {  
3         if (n == 5) {  
4             continue  
5         }  
6         println(n * n)  
7     }  
8 }  
9
```

The code is highlighted with a dark background and light text. The `continue` statement is highlighted with a yellow background.

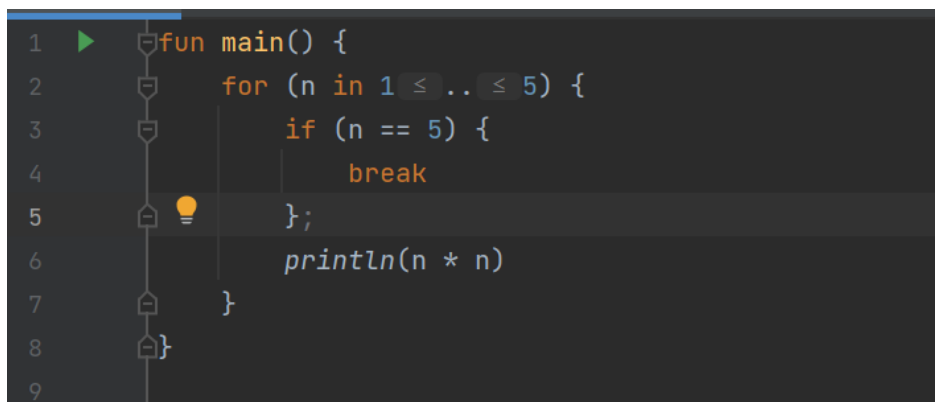
Вывод:



```
Run: MathKt x
C:\Users\Mamedov\.jdk\corretto-17.0.4.1\bin\java.exe
1
4
9
16
36
49
64
Process finished with exit code 0
```

В данном случае, когда n будет равно 5, сработает оператор `continue`. И последующая инструкция, которая выводит на консоль квадрат числа, не будет выполняться. Цикл перейдет к обработке следующего элемента в массиве

Бывает, что при некоторых условиях нам вовсе надо выйти из цикла, прекратить его выполнение. В этом случае применяется оператор `break`:



```
1 fun main() {
2     for (n in 1..5) {
3         if (n == 5) {
4             break
5         };
6         println(n * n)
7     }
8 }
9
```

Вывод:



```
Run: MathKt x
C:\Users\Mamedov\.jdk\corretto-17.0.4.1\bin\java.exe
1
4
9
16
Process finished with exit code 0
```

В данном случае, когда n окажется равен 5, то с помощью оператора `break` будет выполнен выход из цикла. Цикл полностью завершится.