

# Функциональное программирование

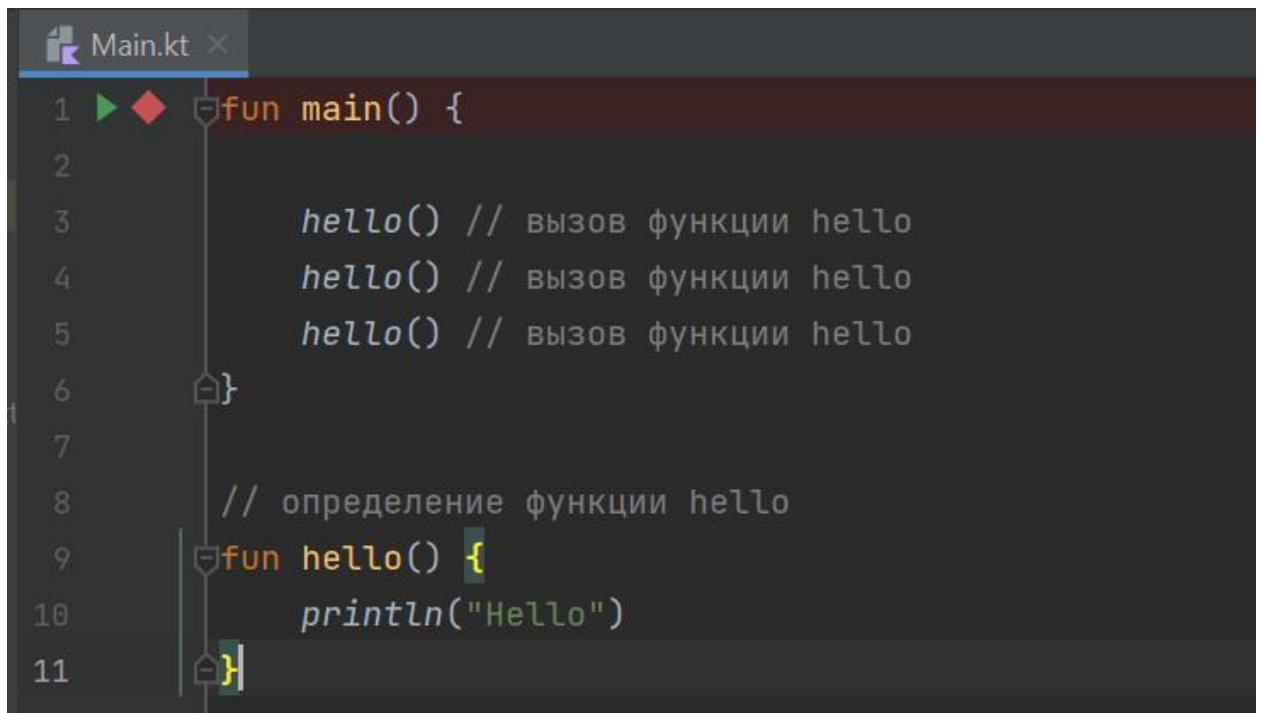
## Функции и их параметры

Одним из строительных блоков программы являются функции. Функция определяет некоторое действие. В Kotlin функция объявляется с помощью ключевого слова **fun**, после которого идет название функции. Затем после названия в скобках указывается список параметров. Если функция возвращает какое-либо значение, то после списка параметров через запятую можно указать тип возвращаемого значения. И далее в фигурных скобках идет тело функции.

```
111
112 fun имя_функции (параметры) : возвращаемый_тип{
113     выполняемые инструкции
114 }
```

Параметры необязательны.

Например, определим и вызовем функцию, которая просто выводит некоторую строку на консоль:



```
Main.kt x
1  fun main() {
2
3      hello() // вызов функции hello
4      hello() // вызов функции hello
5      hello() // вызов функции hello
6  }
7
8      // определение функции hello
9  fun hello() {
10     println("Hello")
11 }
```

Функции можно определять в файле вне других функций или классов, сами по себе, как например, определяется функция `main`. Такие функции еще называют функциями верхнего уровня (top-level functions).

Здесь кроме главной функции `main` также определена функция `hello`, которая не принимает никаких параметров и ничего не возвращает. Она просто выводит строку на консоль.

Функция `hello` (и любая другая определенная функция, кроме `main`) сама по себе не выполняется. Чтобы ее выполнить, ее надо вызвать. Для вызова функции указывается ее имя (в данном случае `"hello"`), после которого идут пустые скобки.

Таким образом, если необходимо в разных частях программы выполнить одни и те же действия, то можно эти действия вынести в функцию, и затем вызывать эту функцию.

## Передача параметров

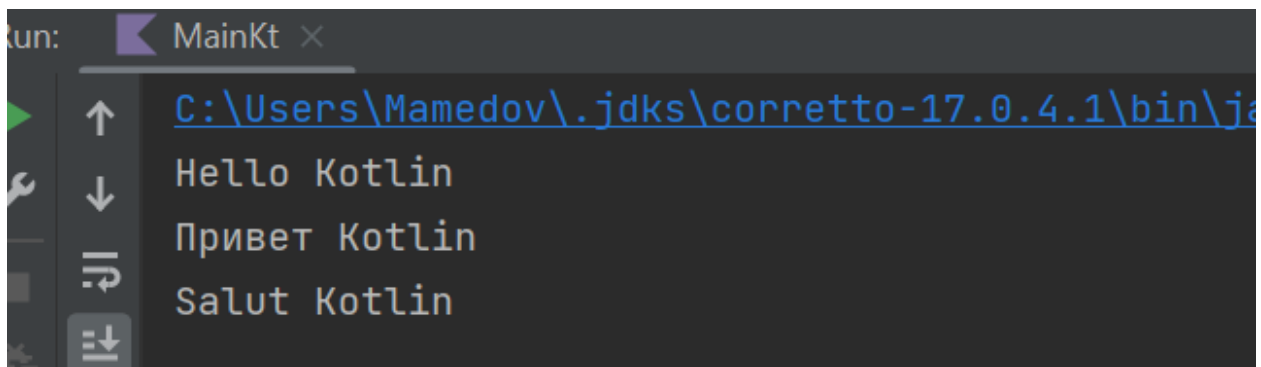
Через параметры функция может получать некоторые значения извне. Параметры указываются после имени функции в скобках через запятую в формате имя\_параметра : тип\_параметра. Например, определим функцию, которая просто выводит сообщение на консоль:



```
1 fun main() {  
2  
3     showMessage("Hello Kotlin")  
4     showMessage("Привет Kotlin")  
5     showMessage("Salut Kotlin")  
6 }  
7  
8 fun showMessage(message: String) {  
9     println(message)  
10 }  
11
```

Функция `showMessage()` принимает один параметр типа `String`. Поэтому при вызове функции в скобках необходимо передать значение для этого параметра: `showMessage("Hello Kotlin")`. Причем это значение должно представлять тип `String`, то есть строку. Значения, которые передаются параметрам функции, еще называют аргументами.

Консольный вывод программы:



```
run: MainKt x  
C:\Users\Mamedov\.jdk\corretto-17.0.4.1\bin\ja  
Hello Kotlin  
Привет Kotlin  
Salut Kotlin
```

Другой пример - функция, которая выводит данные о пользователе на консоль:

```
Main.kt x
1  fun main() {
2
3      displayUser(name: "Tom", age: 23)
4      displayUser(name: "Alice", age: 19)
5      displayUser(name: "Kate", age: 25)
6  }
7
8  fun displayUser(name: String, age: Int) {
9      println("Name: $name   Age: $age")
10 }
```

Функция `displayUser()` принимает два параметра - `name` и `age`. При вызове функции в скобках ей передаются значения для этих параметров. При этом значения передаются параметрам по позиции и должны соответствовать параметрам по типу. Так как вначале идет параметр типа `String`, а потом параметр типа `Int`, то при вызове функции в скобках вначале передается строка, а потом число.

## Аргументы по умолчанию

В примере выше при вызове функций `showMessage` и `displayUser` мы обязательно должны предоставить для каждого их параметра какое-то определенное значение, которое соответствует типу параметра. Мы не можем, к примеру, вызвать функцию `displayUser`, не передав ей аргументы для параметров, это будет ошибка:

```
7      displayUser()
```

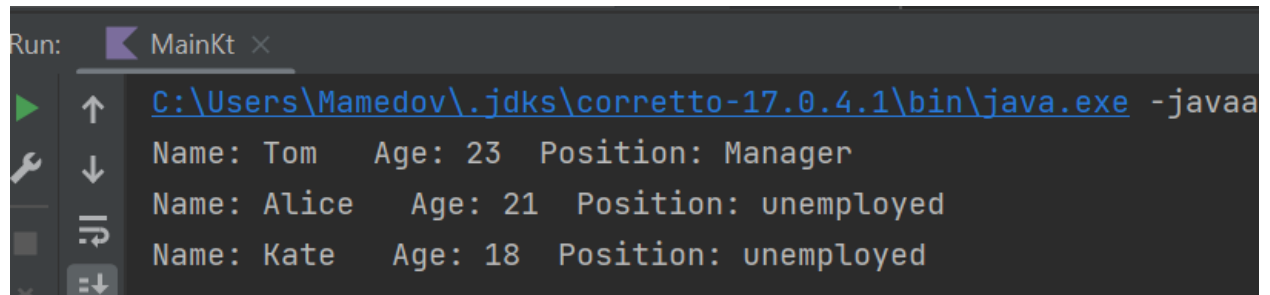
Однако мы можем определить какие-то параметры функции как необязательные и установить для них значения по умолчанию:

```
Main.kt x
1  fun displayUser(name: String, age: Int = 18, position: String = "unemployed") {
2      println("Name: $name   Age: $age   Position: $position")
3  }
4
5  fun main() {
6
7      displayUser(name: "Tom", age: 23, position: "Manager")
8      displayUser(name: "Alice", age: 21)
9      displayUser(name: "Kate")
10 }
```

В данном случае функция `displayUser` имеет три параметра для передачи имени, возраста и должности. Для первого параметра `name` значение по умолчанию не установлено, поэтому для него значение по-прежнему обязательно передавать значение. Два последующих - `age` и `position` являются необязательными, и для них установлено значение по умолчанию. Если для этих

параметров не передаются значения, тогда параметры используют значения по умолчанию. Поэтому для этих параметров в принципе нам необязательно передавать аргументы. Но если для какого-то параметра определено значение по умолчанию, то для всех последующих параметров тоже должно быть установлено значение по умолчанию.

Консольный вывод программы



```
Run: MainKt
C:\Users\Mamedov\.jdk\corretto-17.0.4.1\bin\java.exe -javaa
Name: Tom   Age: 23   Position: Manager
Name: Alice  Age: 21   Position: unemployed
Name: Kate   Age: 18   Position: unemployed
```

## Именованные аргументы

По умолчанию значения передаются параметрам по позиции: первое значение - первому параметру, второе значение - второму параметру и так далее. Однако, используя именованные аргументы, мы можем переопределить порядок их передачи параметрам:



```
Main.kt
1 fun main() {
2
3     displayUser( name: "Tom", position = "Manager", age = 28)
4     displayUser(age = 21, name = "Alice")
5     displayUser( name: "Kate", position = "Middle Developer")
6 }
7
8 fun displayUser(name: String, age: Int = 18, position: String="unemployed"){
9     println("Name: $name   Age: $age   Position: $position")
10 }
11
```

При вызове функции в скобках мы можем указать название параметра и с помощью знака равно передать ему нужное значение.

При этом, как видно из последнего случая, необязательно все аргументы передавать по имени. Часть аргументов могут передаваться параметрам по позиции. Но если какой-то аргумент передан по имени, то остальные аргументы после него также должны передаваться по имени соответствующих параметров.

Также если до обязательного параметра функции идут необязательные параметры, то для обязательного параметра значение передается по имени:



```
Main.kt
1 fun displayUser(age: Int = 18, name: String) {
2     println("Name: $name   Age: $age")
3 }
4
5 fun main() {
6
7     displayUser(name = "Tom", age = 28)
8     displayUser(name = "Kate")
9 }
```

## Изменение параметров

По умолчанию все параметры функции равносильны val-переменным, поэтому их значение нельзя изменить. Например, в случае следующей функции при компиляции мы получим ошибку:

```
1 fun double(n: Int) {  
2     n = n * 2    // !ошибка - значение параметра нельзя изменить  
3     println("Значение в функции double: $n")  
4 }  
5
```

Однако если параметр представляет какой-то сложный объект, то можно изменять отдельные значения в этом объекте. Например, возьмем функцию, которая в качестве параметра принимает массив:

```
Main.kt x  
1 fun double(numbers: IntArray) {  
2     numbers[0] = numbers[0] * 2  
3     println("Значение в функции double: ${numbers[0]}")  
4 }  
5  
6 fun main() {  
7  
8     var nums = intArrayOf(4, 5, 6)  
9     double(nums)  
10    println("Значение в функции main: ${nums[0]}")  
11 }
```

Здесь функция double принимает числовой массив и увеличивает значение его первого элемента в два раза. Причем изменение элемента массива внутри функции приведет к тому, что также будет изменено значение элемента в том массиве, который передается в качестве аргумента в функцию, так как этот один и тот же массив. Консольный вывод:

```
MainKt x  
C:\Users\Mamedov\.jdk\corretto-17.0.4.1\bin\java.exe -javaagent  
Значение в функции double: 8  
Значение в функции main: 8
```

## Переменное количество параметров. Vararg

Функция может принимать переменное количество параметров одного типа. Для определения таких параметров применяется ключевое слово `vararg`. Например, нам необходимо передать в функцию несколько строк, но сколько именно строк, мы точно не знаем. Их может быть пять, шесть, семь и т.д.:

```
1 fun main() {  
2  
3     printStrings(...strings: "Tom", "Bob", "Sam")  
4     printStrings(...strings: "Kotlin", "JavaScript", "Java", "C#", "C++")  
5 }  
6  
7 fun printStrings(vararg strings: String) {  
8     for (str in strings)  
9         println(str)  
10 }  
11
```

Функция `printStrings` принимает неопределенное количество строк. В самой функции мы можем работать с параметром как с последовательностью строк, например, перебирать элементы последовательности в цикле и производить с ними некоторые действия.

При вызове функции мы можем ей передать любое количество строк.

Другой пример - подсчет суммы неопределенного количества чисел:

```
1 fun main() {  
2  
3     sum(...numbers: 1, 2, 3, 4, 5)  
4     sum(...numbers: 1, 2, 3, 4, 5, 6, 7, 8, 9)  
5 }  
6  
7 fun sum(vararg numbers: Int) {  
8     var result = 0  
9     for (n in numbers)  
10         result += n  
11     println("Сумма чисел равна $result")  
12 }
```

Если функция принимает несколько параметров, то обычно `vararg`-параметр является последним.

```

3  ▶ fun main() {
4
5      printUserGroup(count: 3, ...users: "Tom", "Bob", "Alice")
6  }
7
8  fun printUserGroup(count: Int, vararg users: String) {
9      println("Count: $count")
10     for (user in users)
11         println(user)
12 }

```

Однако это необязательно, но если после vararg-параметра идут еще какие-нибудь параметры, то при вызове функции значения этим параметрам передаются через именованные аргументы:

```

1
2  ▶ fun main() {
3
4      printUserGroup(group: "KT-091", ...users: "Tom", "Bob", "Alice", count=3)
5  }
6
7  fun printUserGroup(group: String, vararg users: String, count: Int) {
8      println("Group: $group")
9      println("Count: $count")
10     for (user in users)
11         println(user)
12 }

```

Здесь функция printUserGroup принимает три параметра. Значения параметрам до vararg-параметра передаются по позициям. То есть в данном случае "KT-091" будет представлять значение для параметра group. Последующие значения интерпретируются как значения для vararg-параметра вплоть до именованных аргументов.

## Оператор \*

Оператор \* (spread operator) (не стоит путать со знаком умножения) позволяет передать параметру в качестве значения элементы из массива:

```

2  ▶ fun main() {
3      val nums = intArrayOf(1, 2, 3, 4)
4      changeNumbers(*nums, koef=2)
5  }
6
7  fun changeNumbers(vararg numbers: Int, koef: Int) {
8      for (number in numbers)
9          println(number * koef)
10 }

```

Обратите внимание на звездочку перед `nums` при вызове функции: `changeNumbers(*nums, coef=2)`. Без применения данного оператора мы столкнулись бы с ошибкой, поскольку параметры функции представляют не массив, а неопределенное количество значений типа `Int`.

## Возвращение результата. Оператор `return`

Функция может возвращать некоторый результат. В этом случае после списка параметров через двоеточие указывается возвращаемый тип. А в теле функции применяется оператор `return`, после которого указывается возвращаемое значение.

Например, определим функцию, которая возвращает сумму двух чисел:

```
2  fun main() {
3
4      val a = sum(x: 4, y: 3)
5      val b = sum(x: 5, y: 6)
6      val c = sum(x: 6, y: 9)
7      println("a=$a b=$b c=$c")
8  }
9
10 fun sum(x: Int, y: Int): Int {
11
12     return x + y
13 }
```

В объявлении функции `sum` после списка параметров через двоеточие указывается тип `Int`, который будет представлять тип возвращаемого значения:

```
fun sum(x: Int, y: Int): Int {
```

В самой функции с помощью оператора `return` возвращаем полученное значение - результат операции сложения:

```
return x + y
```

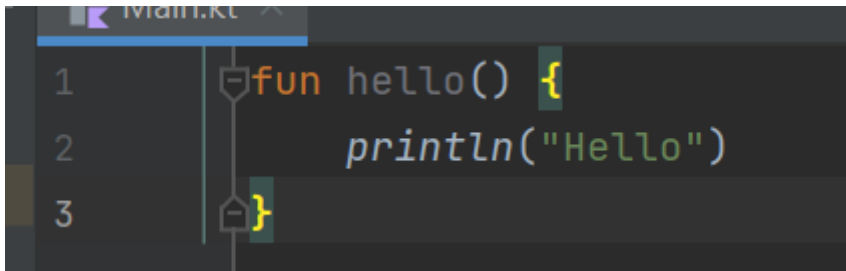
Так как функция возвращает значение, то при ее вызове это значение можно присвоить переменной:

```
val a = sum(x: 4, y: 3)
```

## Тип `Unit`

Если функция не возвращает какого-либо результата, то фактически неявно она возвращает значение типа `Unit`. Этот тип аналогичен типу `void` в ряде языков программирования, которое указывает, что функция ничего не возвращает. Например, следующая функция



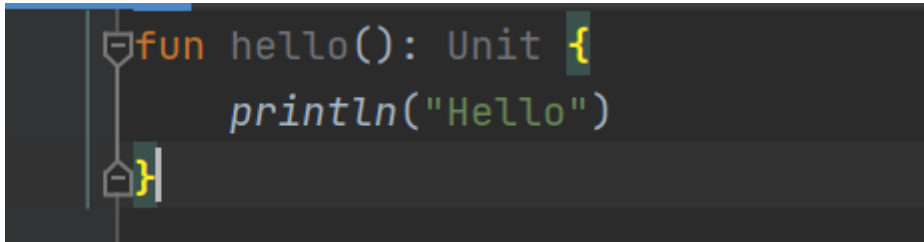


```

1 fun hello() {
2     println("Hello")
3 }

```

Будет аналогична следующей:

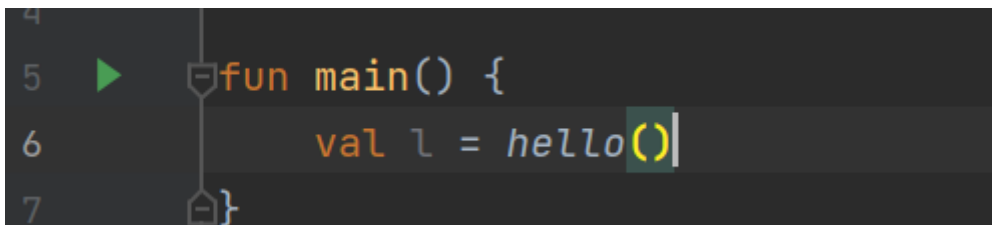


```

fun hello(): Unit {
    println("Hello")
}

```

Формально мы даже можем присвоить результат такой функции переменной:



```

5 fun main() {
6     val l = hello()
7 }

```

Однако практического смысла это не имеет, так как возвращаемое значение представляет объект Unit, который больше никак не применяется.

Если функция возвращает значение Unit, мы также можем использовать оператор return для возврата из функции:



```

1 fun main() {
2     checkAge( age: -10)
3     checkAge( age: 10)
4 }
5
6 fun checkAge(age: Int) {
7     if (age < 0 || age > 110) {
8         println("Invalid age")
9         return
10    }
11    println("Age is valid")
12 }
13

```

В данном случае если значение параметра `age` выходит за пределы диапазона от 0 до 110, то с помощью оператора `return` осуществляется выход из функции, и последующие инструкции не выполняются. При этом если функция возвращает значение `Unit`, то после оператора `return` можно не указывать никакого значения.