**Функции (kotlinlang.ru)**

**fun** double(x: **Int**): **Int** { ... }

**val** result = double(2)

Для вызова вложенной функции используется знак точки:

Sample().foo() *//создаёт экземпляр класса Sample и вызывает foo*

*// вызываем функцию, используя инфиксную запись*

1 shl 2 */\* то же самое, что \*/* 1.shl(2)

**Аргументы по умолчанию**

**fun** read(b: **Array<Byte>, off: Int** = 0, len: **Int** = b.size()) { ... }

**fun** reformat(str: **String, normalizeCase: Boolean** = true, upperCaseFirstLetter: **Boolean** = true, divideByCamelHumps: **Boolean** = false, wordSeparator: **Char** = ' ')

reformat(str)

reformat(str, wordSeparator = '\_')

**Функции с одним выражением:**

Когда функция возвращает одно-единственное выражение, фигурные скобки { } могут быть опущены, и тело функции может быть описано после знака =

**fun** double(x: **Int**): **Int** = x \* 2

Нефиксированное число аргументов **(Varargs):**  может быть помечен модификатором vararg:

**fun** **<T>** asList(vararg ts: **T**): List**<T>** {

**val** list = asList(1, 2, 3)

**val** a = arrayOf(1, 2, 3)

**val** list = asList(-1, 0, \*a, 4)

(оператор spread (необходимо пометить массив знаком \*))

## **Область действия функций**

Функции в языке **Kotlin** могут быть объявлены:

в самом начале файла:(не обязательно создавать объект класса (как в **Java**, **C#** или **Scala**))  
**локально,** (Например, функции, вложенные в другие функции)  
как **функции-члены** (ориг. *"member functions"*) (объявленные внутри классов или объектов)  
**Функции-элементы** - это функции, объявленные внутри классов или объектов  
**Функции-обобщения** (*Generic Functions*) треугольными скобками перед именем функции  
 **fun** **<T>** singletonList(item: **T**): List**<T>** {  
**Встроенные функции** (*Inline Functions*)

**Функции-расширения** (*Extension Functions*)

Функции с **хвостовой рекурсией** (tailrec)  циклические алгоритмы вместо рекурсивных

tailrec **fun** findFixPoint(x: **Double** = 1.0): **Double** = **if** (x == Math.cos(x)) x **else** findFixPoint(Math.cos(x))

функция должна вызывать сама себя в качестве последней операции, которую она предпринимает.

Функции так же могут быть вызваны при помощи **инфиксной записи (**infix) если:

* Они являются членом другой функции или [расширения](http://kotlinlang.ru/docs/reference/extensions.html)  
   f**un** MutableList**<Int>**.swap(index1: **Int, index2: Int**)
* В них используется один параметр
* Когда они помечены ключевым словом infix

*// Определяем выражение как Int*

infix **fun** **Int**.shl(x: **Int**): **Int** {...}

*// вызываем функцию, используя инфиксную запись*

1 shl 2 */\* то же самое, что \*/* 1.shl(2)

**Высокоуровневые функции и лямбды**

# Высокоуровневые функции и лямбды

**Функция высшего порядка** - это функция, которая принимает функции как параметры, или возвращает функцию в качестве результата.

Хорошим примером такой функции является lock(), которая берёт блокирующий объект и функцию, получает блокировку, выполняет функцию и отпускает блокировку:

**fun** **<T>** lock(lock: **Lock, body: (**) -> T): T{

lock.lock()

**try**{

**return** body()

}

**finally** {

lock.unlock()

}

}

Проанализируем этот код: body имеет функциональный тип: () -> T, то есть параметр должен быть функцией без параметров, возвращающей значение типа T. Она вызывается внутри блока try, под защитой объекта lock, получившего блокировку вызовом функции lock().

Если мы хотим вызвать метод lock(), можно передать другую функцию в качестве входящего аргумента:

**fun** toBeSynchronized() = sharedResource.operation()

**val** result = lock (lock, ::toBeSynchronized)

Обычно удобней передавать [лямбда-выражения](https://kotlinlang.ru/docs/reference/lambdas.html#lambda-expressions-and-anonymous-functions):

**val** result = lock(lock, { sharedResource.operation() })

Лямбда-выражения подробно описаны [здесь](https://kotlinlang.ru/docs/reference/lambdas.html#lambda-expressions-and-anonymous-functions), сделаем краткий обзор:

* Лямбда-выражение всегда заключено в фигурные скобки;
* Его параметры (если они есть) объявлены до знака -> (допустимо не указывать);
* Тело выражения следует после знака ->.

В Kotlin есть конвенция, согласно которой, если последний параметр функции является функцией, которая передается в виде лямбда-выражения, можно вынести его за скобки:

lock (lock) {

sharedResource.operation()

}

## **Ключевое слово it: неявное имя единственного параметра**

## **Символ подчеркивания для неиспользуемых переменных**

## **Деструктуризация в лямбдах-** присвоить объект сразу нескольким переменным

## **Инлайн - Встроенные (inline) функции -** встроена в место вызова

## **Лямбда-выражения и/** **или анонимные функции -** "функциональный литерал"

## **Функциональные типы -** принять функцию в качестве параметра - тип функции

## **Синтаксис лямбда-выражений val** sum = { x: **Int**, y: **Int** -> x + y }

**Анонимные функции -** fun(x: **Int, y: Int**): **Int** = x + y

## **Замыкания -** то есть к переменным, объявленным вне этого выражения имеет доступ

## **Литералы функций с объектом-приёмником**

литерал функции: sum : **Int**.(other: **Int**) -> **Int**

может быть вызван так, будто он является методом объекта-приёмника: 1.sum(2)

Лямбда-выражения могут быть использованы как литералы функций с приёмником,  
 когда тип приёмника может быть выведен из контекста