## Практическая работа 6. Работа с функциями в Котлин

## Теоретический материал

[Изучение Kotlin | #4 - Использование функций - поиск Яндекса по видео (yandex.ru)](https://yandex.ru/video/preview/14157724651578982928)

Объявление функции начинается с ключевого слова **fun**, затем идёт имя функции, в круглых скобках указываются параметры. Тип возвращаемого значения указывается после списка параметров и отделяется от него двоеточием. Функция всегда возвращает значение. Если вы сами не указали возвращаемое значение, то функция вернёт **Unit**, который схож с **void**, но является объектом.

Параметры в функциях объявляется немного иначе.

fun add(x: Int, y: Int): Int {

return x + y

}

С функциями можно работать как с значениями - можно сохранить в переменной, передать в качестве параметра, возвратить из другой функции.

Стандартный вывод "Hello Kitty" для Kotlin-программы (Desktop, не Android):

fun main() {

println("Hello Kitty!")

}

Данная функция ничего не возвращает. Напишем другую функцию, возвращающую результат.

fun max(a: Int, b: Int): Int

{ return if (a > b) a else

b

}

Вызываем функцию.

println(max(7, 2))

// выводит 7

Если в теле функции нужно выполнить несколько выражений (операторов), то используется обычный синтаксис объявления функции.

**Пример 1.** Функция, которая вычисляет модуль комплексного числа. Функция получает параметрами вещественную (re) и мнимую (im) части комплексного числа.

// Функция, которая вычисляет модуль комплексного числа

**fun** AbsComplex(re:Double, im:Double) =

Math.sqrt(re\*re+im\*im)

**fun** main(args:Array<String>)

{

// Демонстрация использования функции AbsComplex()

**var** module : Double

**var** re : Double

**var** im : Double

print(**"re = "**)

re = readLine().toString().toDouble()

print(**"im = "**)

im = readLine().toString().toDouble()

module = AbsComplex(re,im)

print(**"module = "** + module)

}

## Параметры по умолчанию

Очень удобная функциональность - создание параметров по умолчанию. Если вы предполагаете, что какой-то параметр будет часто использовать какое-то конкретное значение, то мы можем сразу его указать. При вызове функции мы можем опустить этот параметр, он применится автоматически. Если нам нужно указать другое значение, то параметр добавим.

Добавим в класс активности новую функцию для вывода всплывающего сообщения.

fun Context.toast(message: CharSequence, duration: Int = Toast.LENGTH\_SHORT) {

Toast.makeText(this, message, duration).show()

}

Второй параметр использует значение по умолчанию и мы можем его не указывать при вызове. Вызываем функцию.

toast("Meow") // просто и аккуратно

toast("Meow-w-w", Toast.LENGTH\_LONG) // используем второй параметр

С параметрами по умолчанию нужно быть внимательными, возможна ситуация, когда Kotlin не поймёт, что вы от него хотите. Создадим функцию из трёх параметров, один из них будет иметь значение по умолчанию.

fun sayHello(firstWord: String, secondWord: String = "Kitty", thirdWord: String) {

println("$firstWord $secondWord $thirdWord")

}

Вызываем функцию с двумя параметрами, надеясь, что третий подставится самостоятельно. Но Kotlin не может решить, какой параметр пропущен.

sayHello("Hello", "Kitty") // не компилируется

В этом случае на помощь приходят именованные параметры.

sayHello("Hello", thirdWord = "Kitty")

Третий параметр теперь нам известен, опущенный параметр относится ко второму, оставшийся относится к первому.

##### Тип Unit. Функции, возвращающие тип Unit

Если функция не возвращает значения, то в Kotlin эта функция определяется такой, что она возвращает тип Unit. Если в теле функции не используется ключевое слово return, то считается что функция возвращает значение типа Unit.

В языке Kotlin тип Unit указывает функцию, которая ничего не возвращает, но может применяться к обобщенным функциям. В обобщенных функциях возвращение хотя бы какого-то типа является обязательным.

В наиболее общем случае, функция возвращающая тип Unit, имеет вид

**fun** FuncName(parameters) : Unit {

// ...

}

здесь

* FuncName – имя функции;
* parameters – параметры функции.

### Пример 2

**Задача.** Разработать функцию, которая получает целое число и выводит на экран двоичное представление этого числа.

**Решение.**

// Функция, возвращающая тип Unit.

// Функция, которая выводит двоичный эквивалент целого числа.

**fun** Print\_10\_to\_2(number : Int) : Unit {

**var** t = number

**var** s = **""**

**while** (t>0) {

s = (t%2).toString() + s

t = t/2

}

println(number.toString() + **" => "** + s)

}

**fun** main(args:Array<String>)

{

// Демонстрация использования функции Print\_10\_to\_2()

// 1. Ввод числа

print(**"number = "**)

**val** number = readLine().toString().toInt()

// 2. Вызов функции

Print\_10\_to\_2(number)

}

**Задание**

1. Разработать функцию, которая по заданному радиусу вычисляет объем шара.
2. Разработать функцию, которая выводит на экран целое число, представленное в обратном порядке.
3. Задано положительное число от 0 до 999999. Разработать функцию с единственным выражением, которая вычисляет количество цифр в числе.
4. Разработать функцию, которая по номеру месяца (1..12) и номеру года определяет количество дней в этом месяце.