

Лекция 10. Определение и вызов функций

Лекция 10. Определение и вызов функций

1. Упрощение вызова функций
2. Добавление методов в сторонние классы: функции-расширения и свойства-расширения
3. Работа с коллекциями: переменное число аргументов, инфиксная форма записи вызова и поддержка в библиотеке
4. Работа со строками и регулярными выражениями

```
>>> val list = listOf(1, 2, 3)
```

```
>>> println(list)
```

```
[1, 2, 3]
```

← **Вызов метода toString()**

```
fun <T> joinToString(  
    collection: Collection<T>,  
    separator: String,  
    prefix: String,  
    postfix: String  
): String {  
  
    val result = StringBuilder(prefix)  
  
    for ((index, element) in collection.withIndex()) {  
        if (index > 0) result.append(separator)  
        result.append(element)  
    }  
  
    result.append(postfix)  
    return result.toString()  
}
```

← Не нужно вставлять разделитель
перед первым элементом

```
>>> val list = listOf(1, 2, 3)
>>> println(joinToString(list, "; ", "(" , ")"))
(1; 2; 3)
```

```
joinToString(collection, separator = " ", prefix = " ", postfix = ".")
```

```
fun <T> joinToString(  
    collection: Collection<T>,  
    separator: String = ", ",  
    prefix: String = "",  
    postfix: String = ""  
): String
```

Параметры со значениями
по умолчанию

```
>>> joinToString(list, ", ", "", "")  
1, 2, 3  
>>> joinToString(list)  
1, 2, 3  
>>> joinToString(list, "; ")  
1; 2; 3
```



```
>>> joinToString(list, suffix = ";", prefix = "# ")  
# 1, 2, 3;
```

```
package strings  
fun joinToString(...): String { ... }
```

```
var opCount = 0
```

◀ Объявление свойства
верхнего уровня

```
fun performOperation() {  
    opCount++  
    // ...  
}
```

◀ Изменение значения
свойства

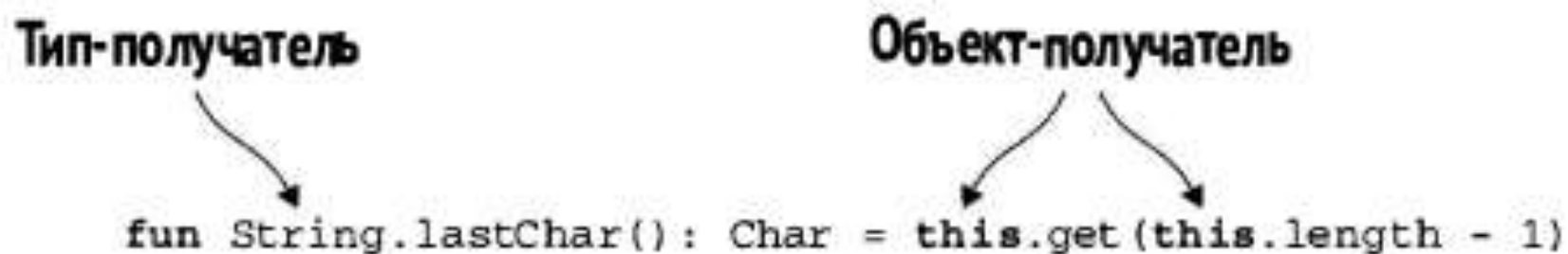
По сути, *функция-расширение* – очень простая штука: это функция, которая может вызываться как член класса, но определена за его пределами. Для демонстрации добавим метод получения последнего символа в строке:

```
package strings
```

```
fun String.lastChar(): Char = this.get(this.length - 1)
```

Тип-получатель

Объект-получатель



```
fun String.lastChar(): Char = this.get(this.length - 1)
```

Рис. 3.1. Тип-получатель – это тип, для которого определяется расширение, а объект-получатель – это экземпляр данного типа

```
import strings.lastChar
```

```
val c = "Kotlin".lastChar()
```

```
import strings.*
```

```
val c = "Kotlin".lastChar()
```

```
import strings.lastChar as last
```

```
val c = "Kotlin".last()
```

```
fun <T> Collection<T>.joinToString(
    separator: String = ", ",
    prefix: String = "",
    postfix: String = ""
): String {
    val result = StringBuilder(prefix)

    for ((index, element) in this.withIndex())
        if (index > 0) result.append(separator)
        result.append(element)

    result.append(postfix)
    return result.toString()
}
```

← Объявление функции-расширения
для типа `Collection<T>`

Значения по умолчанию
для параметров

← «this» ссылается на объект-приемник:
коллекцию элементов типа `T`

```
>>> val list = listOf(1, 2, 3)
>>> println(list.joinToString(separator = "; ",
... prefix = "(", postfix = ")"))
(1; 2; 3)
```

```
fun Collection<String>.join(  
    separator: String = ", ",  
    prefix: String = "",  
    postfix: String = ""  
) = joinToString(separator, prefix, postfix)
```

```
>>> println(listOf("one", "two", "eight").join(" "))  
one two eight
```

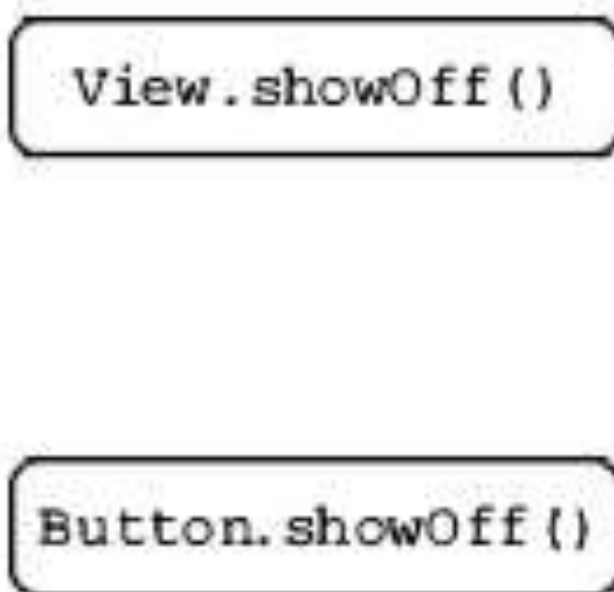
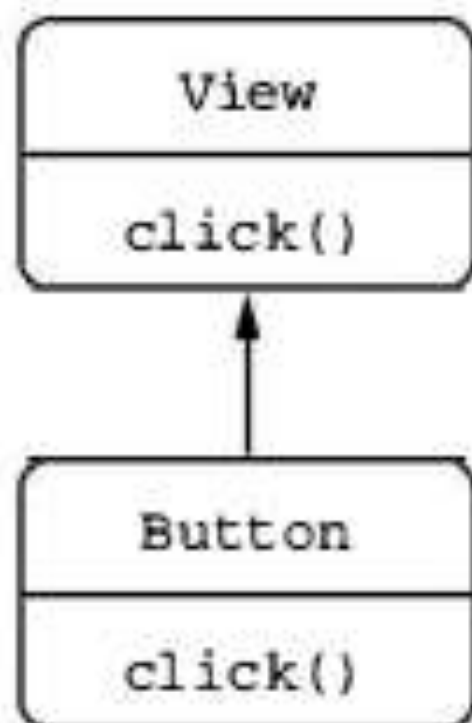


```
open class View {  
    open fun click() = println("View clicked")  
}  
  
class Button: View() {  
    override fun click() = println("Button clicked")  
}
```

← Класс Button
наследует View

```
>>> val view: View = Button()  
>>> view.click()  
Button clicked
```

← Вызываемый метод определяется
фактическим значением переменной «view»



```
fun View.showOff() = println("I'm a view!")  
fun Button.showOff() = println("I'm a button!")
```

```
>>> val view: View = Button()  
>>> view.showOff()  
I'm a view!
```

← Вызываемая функция-расширение
определяется статически

```
val String.lastChar: Char  
    get() = get(length - 1)
```

```
var StringBuilder.lastChar: Char
    get() = get(length - 1)
    set(value: Char) {
        this.setCharAt(length - 1, value)
    }
```

← **Метод чтения для свойства**

← **Метод записи для свойства**

- ключевое слово `vararg` позволяет объявить функцию, принимающую произвольное количество аргументов;
- *инфиксная* нотация поможет упростить вызовы функций с одним аргументом;
- *мультидекларации* (destructuring declarations) позволяют распаковать одно составное значение в несколько переменных.

Вызывая функцию создания списка, вы можете передать ей любое количество аргументов:

```
val list = listOf(2, 3, 5, 7, 11)
```

В месте объявления этой библиотечной функции вы найдете следующее:

```
fun listOf<T>(vararg values: T): List<T> { ... }
```


называется вызовом с *оператором распаковки* (spread operator), а на практике это просто символ `*` перед соответствующим аргументом:

```
fun main(args: Array<String>) {  
    val list = listOf("args: ", *args)  
    println(list)  
}
```

← Оператор «звездочка» распаковывает
содержимое массива

```
val map = mapOf(1 to "one", 7 to "seven", 53 to "fifty-three")
```

```
1.to("one")
```

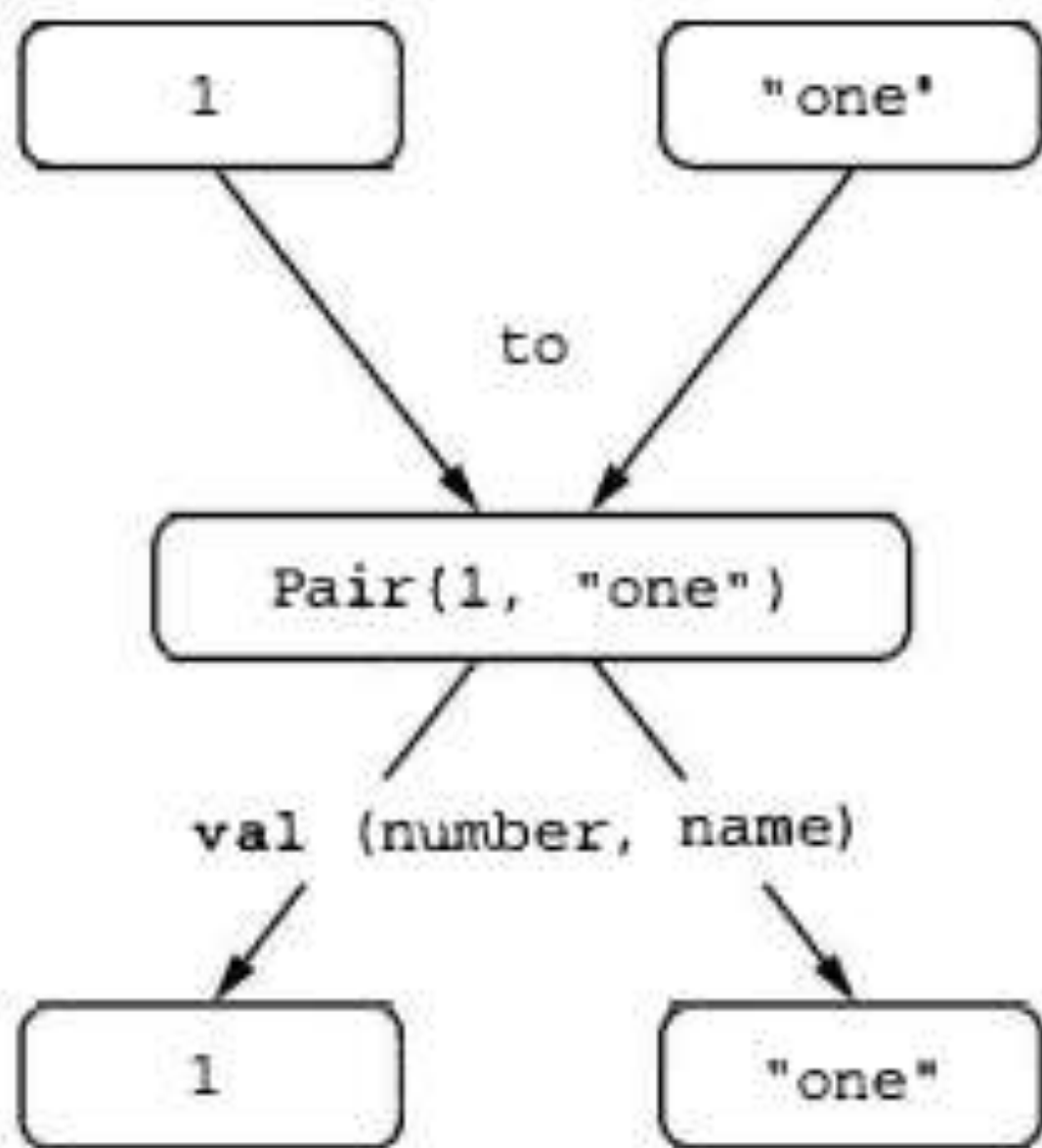
← Вызов функции to обычным способом

```
1 to "one"
```

← Вызов функции to с использованием
инфиксной нотации

```
infix fun Any.to(other: Any) = Pair(this, other)
```

```
val (number, name) = 1 to "one"
```



```
for ((index, element) in collection.withIndex()) {  
    println("$index: $element")  
}
```

```
fun <K, V> mapOf(vararg values: Pair<K, V>): Map<K, V>
```

<https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-string/>

- KotlinAsFirst
- `[A-Z0-9._%-]@[A-Z0-9.-]+\.[A-Z]{2,}`
- `^4[0-9]{12}(:[0-9]{3})?$`
- `[-]?[0-9]*\.[0-9]`
- `<()([<^])?(:>(<\/\1>|\s+\/>)`

- KotlinAsFirst
- Трансмогрификация
- Мама мыла раму
- 42

- [0123456789] — любая цифра
- [aeiouy] — любая буква из перечисленных
- [~!@#\$%^&*+ -] — любой символ из перечисленных

- `[^0123456789]` — всё, что угодно, кроме цифры
- `[^a-z]` — всё, что угодно, кроме строчной латинской буквы
- `[^ -az]` — всё, что угодно, кроме -, a, z

- `^fun` — `fun` в начале строки
- `\.$` — точка в конце строки
- `^Kotlin is great as the first language!$` — ВСЯ строка с заданной фразой (и более ничем)

Здесь ^ используется для обозначения начала строки, а \$ для обозначения конца. Следует иметь в виду, что якоря никак не учитывают переводы строк — имеется в виду начало или конец всего текста, а не одной строки в тексте.

\. использует экранирование для обозначения символа ., поскольку в регулярных выражениях точка является специальным символом (и обозначает любой символ). Таким образом, \ в регулярных выражениях экранирует последующий символ, делая его из специального символа обыкновенным. Для обозначения символа \ применяется пара \\. Аналогично, ^ обозначает символ-шапку, \$ — символ доллара, [— открывающую квадратную скобку,] — закрывающую квадратную скобку.

- \t — табуляция, \n — новая строка, \r — возврат каретки (два последних символа унаследованы компьютерами от эпохи пишущих машинок, когда для начала печати с новой строки необходимо было выполнить два действия — *возврат каретки* в начало строки и перевод каретки на *новую строку*)
- \s — произвольный вид пробела (пробел, табуляция, новая строка, возврат каретки)
- \d — произвольная цифра, аналог [0-9]
- \w — произвольная «символ в слове», обычно аналог [a-zA-z0-9], то есть, латинская буква или цифра
- \S — НЕ пробел, \D — НЕ цифра, \W — НЕ «символ в слове»

- `Марат|Михаил` — Марат или Михаил
- `^\[|\]$` — открывающая квадратная скобка в начале строки или закрывающая в конце
- `for.(val|var).` — цикл `for` с последующим `val` или `var`

- $.^*$ — любое количество (в том числе ноль) любых символов
- $(\text{Марат})^+$ — строка Марат один или более раз (но не ноль)
- $(\text{Михаил})?$ — строка Михаил ноль или один раз
- $([0-9]\{4\})$ — последовательность из ровно четырёх любых цифр
- $\backslash w\{8, 16\}$ — последовательность из 8-16 «символов в слове»

Круглые скобки () задают так называемые *группы поиска*, объединяя несколько символов вместе.

- `(Kotlin)+AsFirst` — `KotlinAsFirst`, `KotlinKotlinAsFirst`, `KotlinKotlinKotlinAsFirst`, ...
- `(?:\$\$)+` — `,` `,` `,` ...
- `(\w+)\s\1` — слово, за которым следует пробел и то же самое слово.
- `fun\s+(/w+)\s*\{.\1.\}` — `fun` с последующими пробелами, произвольным словом в круглых скобках, пробелами и тем же словом в фигурных скобках

Здесь `\1` (`\2`, `\3`, ...) ищет **уже описанную** группу поиска по её номеру внутри регулярного выражения (в данном случае — первую группу). Комбинация `(?:...)` задаёт группу поиска **без номера**. В целом, `(?...)` задаёт *группы особого поиска*:

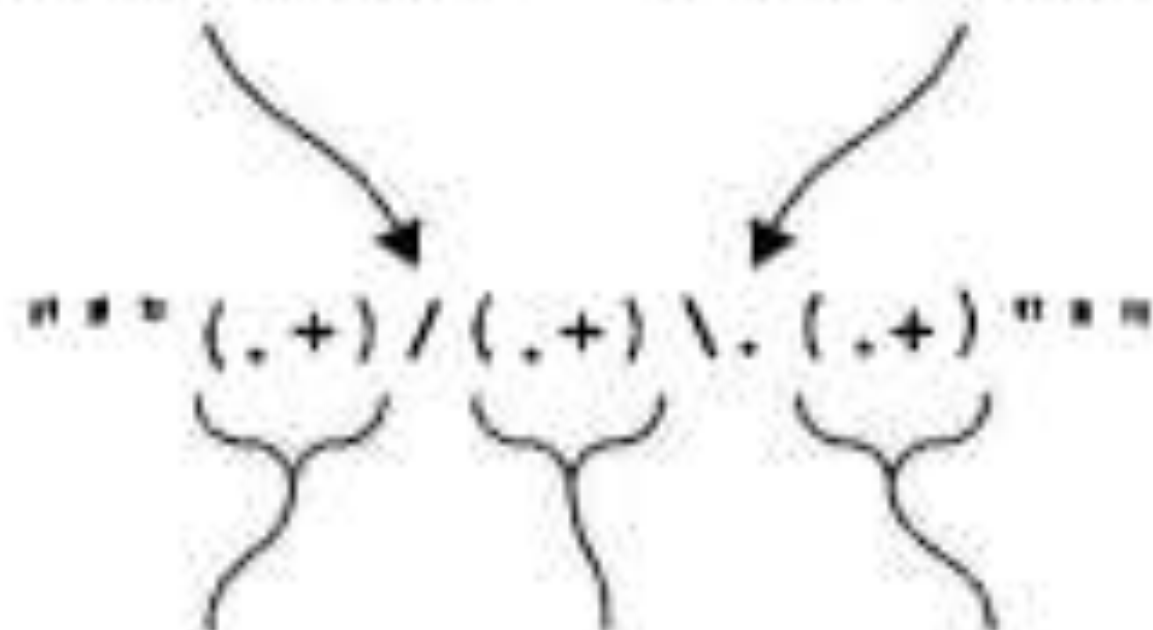
- `Марат (?=\sАхин)` — Марат, за которым следует пробел и Ахин
- `(?<Михаил\s)Глухих` — Глухих, перед которым стоит Михаил с пробелом
- `\d+(?![\$\d])` — число, после которого НЕ стоит знак доллара
- `(?<!\root\s)beer` — beer, перед которым НЕ стоит root с пробелом


```
fun parsePath(path: String) {  
    val directory = path.substringBeforeLast("/")  
    val fullName = path.substringAfterLast("/")  
  
    val fileName = fullName.substringBeforeLast(".")  
    val extension = fullName.substringAfterLast(".")  
  
    println("Dir: $directory, name: $fileName, ext: $extension")  
}  
>>> parsePath("/Users/yole/kotlin-book/chapter.adoc")  
Dir: /Users/yole/kotlin-book, name: chapter, ext: adoc
```

```
fun parsePath(path: String) {  
    val regex = """.+/.+\.(.+)""".toRegex()  
    val matchResult = regex.matchEntire(path)  
    if (matchResult != null) {  
        val (directory, filename, extension) = matchResult.destructured  
        println("Dir: $directory, name: $filename, ext: $extension")  
    }  
}
```

Последний слеш

Последняя точка



Каталог

Имя файла

Расширение

