

Знакомство с функциональной парадигмой языка программирования Kotlin.

Романов Владимир Юрьевич
МГУ им. М.В.Ломоносова, ф-т ВМК
romanov.rvy@yandex.ru

2.1. ОБЗОР ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ЯЗЫКА KOTLIN

Функции в языке Kotlin:

Особенности функций в языке Kotlin:

- могут храниться в переменных и структурах данных
- передаваться как параметры других функций
- возвращаться как результат работы других функций

Для этих целей в языке используются:

- ***функциональные типы***
- конструкции языка называемые ***лямбда-выражениями***

Анонимная функция в Kotlin

Переменной **sum** присваивается *анонимная функция*:

```
val sum  
    = fun(a: Int, b: Int): Int = a + b
```

```
fun main() {  
    println("sum: " + sum(1,2))  
}
```

Функциональный тип

Явное задание функционального типа **(Int,Int) -> Int**
для переменной **sum**:

```
val sum: (Int,Int) -> Int  
    = fun(a: Int, b: Int): Int = a + b
```

```
fun main() {  
    println("sum: " + sum(1,2))  
}
```

Псевдоним функционального типа

Задание псевдонима **BinFun** для функционального типа **(Int,Int) -> Int**:

```
typealias BinFun = (Int, Int) -> Int
```

```
val sum: BinFun  
    = fun(a: Int, b: Int): Int = a + b
```

```
fun main() {  
    println("sum: " + sum(1,2))  
}
```

Лямбда - выражение

Замена анонимной функции

```
fun(a: Int, b: Int): Int = a + b
```

на лямбда-выражение

```
{ a: Int, b: Int -> a + b }
```

```
typealias BinFun = (Int, Int) -> Int
```

```
val sum: BinFun = { a: Int, b: Int -> a + b }
```

```
fun main() {  
    println("sum: " + sum(1,2))  
}
```

Выведение типа переменной

Выведение типа **Int** для результата вызова функции:

```
val sum = { a: Int, b: Int -> a + b }
```

```
fun main() {  
    println("sum: " + sum(1,2))  
}
```

Лямбда-выражение для вызова функции

Явное использование лямбда-выражения для вызова функции с помощью операции ():

```
fun main() {  
    println("sum: "  
        + { a: Int, b: Int -> a + b }(1,2) )  
}
```

Явное использование лямбда-выражения для вызова функции с помощью функции **invoke**

```
fun main() {  
    println("sum: "  
        + { a: Int, b: Int -> a + b }.invoke(1,2) )  
}
```

Пример функции с лямбдой-параметром

Функцию ***forEach*** можно вызвать для любой коллекции.

Она принимает один аргумент ***action***:

функцию, определяющую какие действия надо выполнить для каждого элемента.

```
public inline  
fun <T> Iterable<T>.forEach  
    (action: (T) -> Unit): Unit {  
    for (element in this)  
        action(element)  
}
```

Лямбда-выражение как последний параметр функции

Особенности синтаксиса языка Kotlin при использовании лямбда выражений

```
data class Person(val name: String, var age: Int)
fun Person.print()
    = println("$name age is $age years")
```

```
val people: List<Person>
    = listOf( Person("Аня", 5), Person("Оля", 7) )
```

```
people.forEach({ p: Person -> p.print() })
people.forEach() { p: Person -> p.print() }
people.forEach { p: Person -> p.print() }
people.forEach { p -> p.print() }
people.forEach { it.print() }
```

Лямбда-выражение как ссылка на функцию

Лямбда-выражения - параметр хранимый в переменной которые можно передавать в качестве параметров функций

```
data class Person(val name: String, var age: Int)
```

```
fun Person.print()  
    = println("$name is $age years")
```

```
people.forEach(Person::print)
```

Пример лямбда-выражения как последнего параметра

```
fun <T> Iterable<T>.joinToString  
    (separator: CharSequence = ", ",  
     prefix: CharSequence = "",  
     postfix: CharSequence = "",  
     limit: Int = -1,  
     truncated: CharSequence = "...",  
     transform: ((T) -> CharSequence)? = null)  
: String
```

```
people.joinToString(" ") { p: Person -> p.name }
```

```
people.joinToString(prefix="(", postfix=")")  
    { p: Person -> "${p.name}[${p.age}]" }
```