ФУНКЦИИ

ФУНКЦИИ

```
val x: Int => Int = ...
val y: (Int, Int) => Int = ...
```

- **Функция** — это значение, которое может быть использовано как метод
- Максимальное количество параметров у функции — **22**

ЛЯМБДА-АБСТРАКЦИЯ

```
val addOne: Int => Int = x => x + 1
val plus: (Int, Int) => Int = (x, y) => x + y
```

- Значения для функций можно задавать через лямбда-синтаксис

ЛЯМБДА-АБСТРАКЦИЯ

```
val addOne = (x: Int) => x + 1
val plus = (x: Int, y: Int) => x + y
```

- Можно указывать тип параметров прямо в лямда-абстракции, тогда scala попробует вывести тип всего выражения сама

КОРОТКАЯ ЗАПИСЬ

```
val addOne: Int => Int = _ + 1
val plus = (_: Int) + (_ : Int)
```

- Вместо именованых параметров, если они встречаются один раз в выражении, можно использовать запись с подчёркиваниями

ЭТА-КОНВЕРСИЯ

```
def addOne(x: Int) = x + 1
val add1 = addOne _

def plus(x: Int, y: Int) = x + y
val pl: (Int, Int) => Int = plus
```

- Также в функции можно превращать методы. Сделать это можно, поставив подчёркивание после метода
- Если компилятор понимает, что значением должна быть функция, подчеркивание можно не ставить

ФУНКЦИЯ - ЗНАЧЕНИЕ

```
def greaterOn(f: Int => Int): (Int, Int) => Boolean =
  (x, y) => f(x) > f(y)
```

- Функции можно передавать в качестве параметра и возвращать из методов и функций в качестве значений

```
val greaterOnOnes = greaterOn(x => x % 10)
greaterOnOnes(23, 45) // false
greaterOnOnes(27, 45) // true
```

КАРРИРОВАНИЕ

```
def plus: Int => Int => Int = x => y => x + y
```

- Вы можете представить функцию многих параметров, как последовательность функций от одного параметра, возвращающих функцию

```
plus(1)(2) // 3
```

КАРРИРОВАНИЕ

```
val plus3 = (x: Int, y: Int, z: Int) => x + y + z
val plus3c: Int => Int => Int => Int = plus3.curried
```

- Вы можете превратить функцию многих параметров в каррированный вариант с помощью метода **curried**

```
plus3c(1)(2)(3)
```

КОМПОЗИЦИЯ

```
val plus1 = (_: Int) + 1
val mul3 = (_: Int) * 3

val plusThenMul = plus1 andThen mul3
val plusBeforeMul = plus1 compose mul3
```

- Функции, состоящие из последовательного вызова других функций, можно задавать композицией с помощью методов **andThen** и **compose**

```
plusThenMul(5) // mul3(plus1(5)) = 18
plusBeforeMul(5) // plus1(mul3(5)) = 16
```

Мы изучили функции

В следующем разделе практика