МЕТОДЫ

(продолжение)

ПОВТОРЯЕМЫЕ ПАРАМЕТРЫ

```
def sumAllTimes(u: Int, nums: Int*): Int = u * nums.sum
```

- Последний параметр в списке параметров может быть "повторяемым", тогда в теле метода его можно использовать как последовательность
- О последовательностях и других коллекциях мы узнаем в следующих уроках

```
sumAllTimes(3, 1, 2, 3) // 3 * (1 + 2 + 3) = 18
```

ЗНАЧЕНИЯ ПО УМОЛЧАНИЮ

```
def plus3(x: Int, y: Int = 0, z: Int = 0): Int =
  100 * x + 10 * y + z
```

- Тип результата иногда можно опускать
- Scala попробует вывести тип сама

```
plus3(1) // 100
plus3(1, 2) // 120
plus3(1, 2, 3) // 123
```

ИМЕНОВАННЫЕ АРГУМЕНТЫ

```
def plus3(x: Int, y: Int = 0, z: Int = 0): Int =
  100 * x + 10 * y + z
```

- Если параметров много, или несколько параметров имеют значение по умолчанию, при вызове аргументы можно упоминать по имени

```
plus3(x = 1) // 100
plus3(1, z = 2) // 102
plus3(x = 1, z = 3, y = 2) // 123
```

ПЕРЕДАЧА ПО ИМЕНИ

```
def replaceNegative(x: Int, z: => Int): Int =
  if (x >= 0) x else z
```

- Если вы хотите, чтобы значение выражения вычислялось не всегда, можно использовать передачу "по имени"
- Значение параметров, переданных по имени, вычисляется только в случае необходимости

```
replaceNegative(1, 3 * 3 * 3) // 1
replaceNegative(-1, 3 * 3 * 3) // 27
```

ПЕРЕДАЧА БЛОКА

```
def replaceNegative(x: Int, z: => Int): Int =
  if (x >= 0) x else z
```

- Если список параметров содержит ровно один параметр, его значение можно передать блоком в фигурных скобках

```
replaceNegative(1){
  println("calculated")
  3 * 3 * 3
} // 1
```

```
replaceNegative(-1){
  println("calculated")
  3 * 3 * 3
} // prints "calculated"; returns 27
```

РЕКУРСИЯ

```
def sumRange(from: Int, to: Int): Int =
  if(to < from) 0
  else from + sumRange(from + 1, to)
```

- Функция в своём теле может снова вызывать себя
- Это называется рекурсия
- Для рекурсивных функций обязательно указывать тип

```
sumRange(1, 10) // 55
```

ХВОСТОВАЯ РЕКУРСИЯ

```
def sumRange(from: Int, to: Int, acc: Int = 0): Int =
  if (to < from) acc
  else sumRange(from + 1, to, acc + from)
```

- Если вызовы самой себя происходят только в "хвостовых точках" вычислений, функция может быть оптимизирована в "хвостовую рекурсию", которая внутри себя будет представлять из себя цикл

```
sumRange(1, 10) // 55
```

Мы изучили методы

В следующем разделе практика