Знакомство со Scala

Второе занятие

Анойкин Д., Галако О.
ФИТ НГУ 2011-2012

Установка

- Open Source, http://scala-lang.org
- Платформа: JDK 1.6
- **scala** – REPL, script runner
- **scalac** – компилятор
- Результат компиляции – .class-файлы JVM

Документация

- **Programming in Scala**, 2nd ed., Odersky и др.
- **Scala для нетерпеливых,** Хорстман К.
- Twitter's "**Effective Scala**" (рус.)
- Twitter's "**Scala Школа**" (рус.)
- **Google Groups:** scala-user, scala-lang

Scala REPL

- REPL = Read-Evaluate-Print Loop
- Удобный способ проверки выражений
- Пригождается даже для Java
- Авто-дополнение, проверка типов и др.

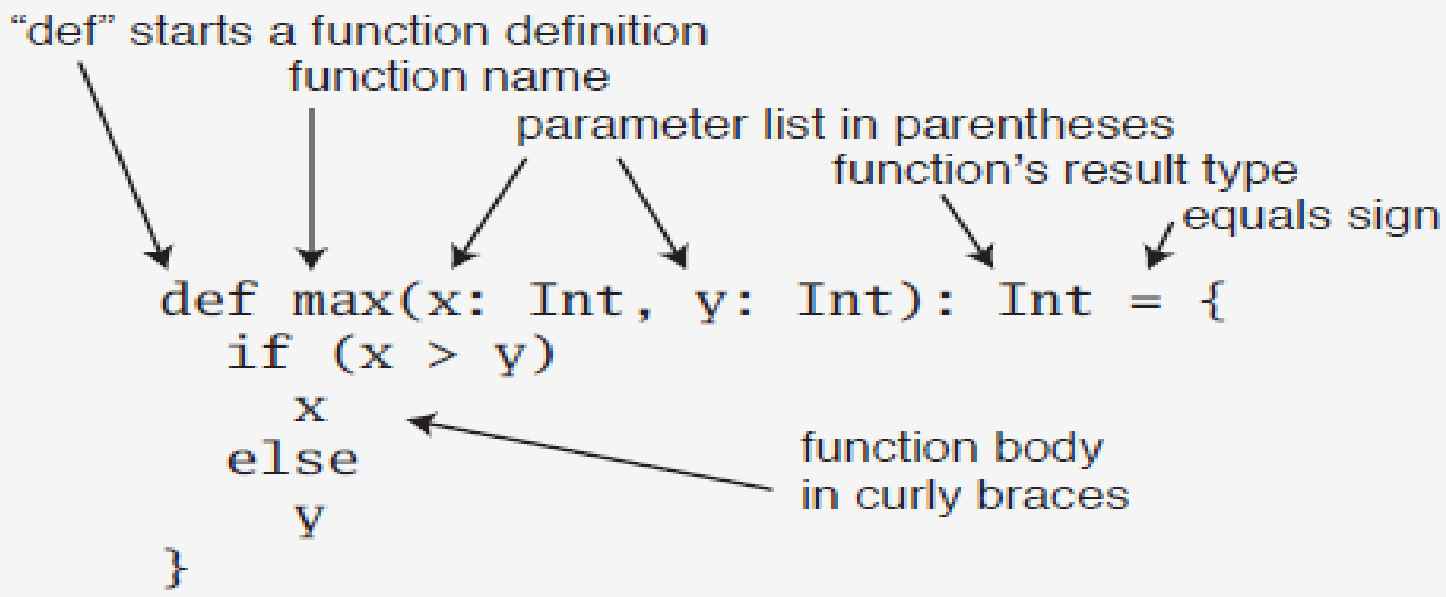
Основы синтаксиса

- Дружественный C, Java-разработчикам
- Тип после имени: **a: String** вместо **String a**
- Альтернативные формы как в функц. яз.
- Послабления, вывод типов
- Переменные и поля: **val**, **var**
- Функции и методы: **def**

Основы синтаксиса

```
class Person(name: String, var age: Int) {
  val i: Int = 0; // #1
  val j = 5 // #2
  val s = “abc” // #3
  def this(name:String) = this(name, 18) // #4
  def getName():String = { name } // #5
  def getNameShort = name // #6
  def setAge(age: Int):Unit { this.age = age } // #7
  println(“Hello from constructor, j = ” + j) // #8
}
```

Функции




```
def sum1(x:Int, y:Int):Int = { x+y }                    // #1
def sum2(x:Int, y:Int):Int = x+y                        // #2
def sum3(x:Int, y:Int) = x+y                            // #3
def sum4(x:Int, y:Int) = println("sum"); x+y;           // #4, compilation error
def sum5(x:Int, y:Int) = {println("sum"); x+y}          // #5
def sum6(x:Int, y:Int) {x+y}                            // #6, возвращает Unit
```

Вызов функции

```
class Food {
  def transforms(quickly:Boolean) = if (quickly) 10 else 1000
  def @?#! = 0
}
class Human {
  def quickly() = true
  def licks(food:Food) = food.transforms(!quickly())
  def eats(food:Food) = food transforms quickly
  def ####(food:Food) = food @?#!
  def lives {this eats new Food }    // eats new Food – compilation error
  def life {this.lives}
}
```

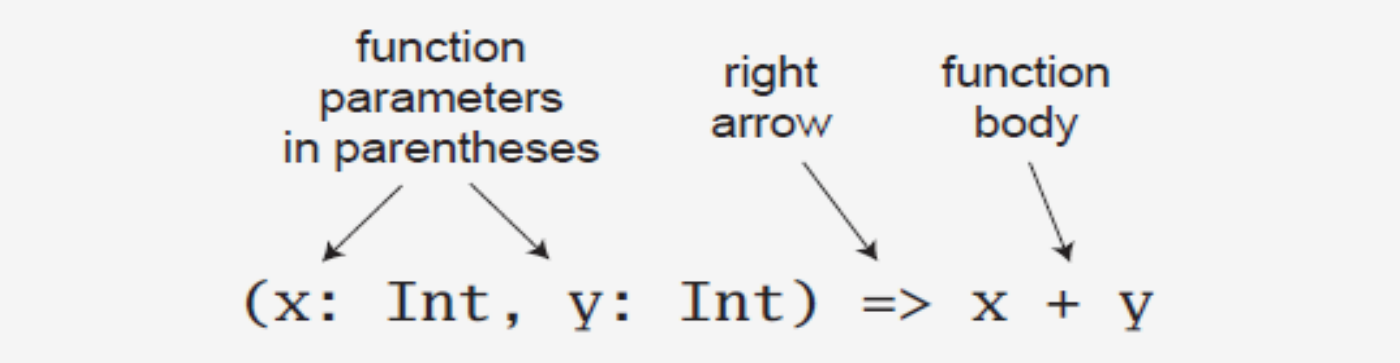
Функция как тип

(Arg1_Type, …, ArgN_Type) => Result_Type

```
def sum(x:Int, y:Int) = x+y                                          // #1
val sumAsValue:(Int,Int)=>Int = sum                                  // #2
val sumResult = sumFunc(1, 2)                                        // #3

def sumMeta(x:Int, y:Int, sumF:((Int,Int)=>Int)) = sumF(x,y)  // #4
val sumResult2 = sumMeta(1, 2, sum)                                  // #5
val sumResult3 = sumMeta(1, 2, sumAsValue)                           // #6
```

Анонимные функции




```
val sumFunc = (x:Int, y:Int) => x+y                                  // #1

def sumMeta(x:Int, y:Int, sumF:((Int,Int)=>Int)) = sumF(x,y)

sumMeta(1, 2, sumFunc)                                               // #2
sumMeta(1, 2, (x:Int, y:Int) => x+y)                                 // #3
sumMeta(1, 2, (x,y) => x+y)                                          // #4
sumMeta(1, 2, _+_)                                                   // #5
```

Функция как объект

```
def func(x:Int) = x                  // это будет транслировано как метод в Java
val funcAsValue:Int=>Int = func    // это будет поле типа Function[Int, Int]
val funcAsValue2 = func _      // без _ непонятно – присвоение или вызов?
val funcAsObject:Function[Int, Int]= func

func(10)
funcAsObject(10)
funcAsObject.apply(10)
funcAsValue.apply(10)       // func.apply(10) – compilation error

class MyFunc extends Function[Int, Int] {
    override def toString() ="My func!"
    def apply(x:Int) = x*x
}
val myfunc:Int=>Int=new MyFunc
println(funcAsValue+" result: "+funcAsValue(10)) // <function1> result: 10
println(myfunc+" result: "+myfunc(10))  // My func! result: 100
```

Передача параметров

```
def info():Int = {println(“param"); 0}
def callByValue(param: Int) = {          // передача по значению
        println("by value"); param
}
def callByName(param: => Int) = {        // передача по имени
        println("by name"); param
}
callByValue(info)                // param \n by value
callByName(info)                 // by name \n param
```

Передача параметра по имени позволяет избежать ненужных вычислений:

```
def search (pattern:String, cache:Array[String], con: => Connection ) {
        if (findInCache(pattern, cache)) true
        else if (findInDB(pattern, con)) true
        else false
}
```

Соединение с базой устанавливается только тогда, когда в кэше не нашли.

Область видимости

Вспомним рекурсию:

```
def factorial(x: Int): Int = if (x == 0) 1 else x * factorial(x-1)
```

Посчитаем число сочетаний из n элементов по m элементов:

```
def combinations (m:Int, n:Int) = {
  def factorial(x: Int): Int = if (x == 0) 1 else x * factorial(x-1)

  factorial (n) / (factorial(m) * factorial(n-m))
}

combinations(2, 3)
factorial(10)           // compile error
```

Hello World!

```
object Program {
    def main(args:Array[String]) {
        println("Hello World!")
    }
}
```

```
object Program {
    def main(args:Array[String]) {
        println("Hello "+(if (args.length>0) args(0) else
"World!"));
    }
}
```

Задание №1-1

Написать программу, которая распечатывает свои аргументы с помощью рекурсивной функции

Документация доступна по адресу:
http://ccfit.nsu.ru/~den/Scala/api/

Нюансы по слайду «Функция как объект»
http://jim-mcbeath.blogspot.com/2009/05/scala-functions-vs-methods.html

Задание №1-2

Используя рекурсивную функцию, написать программу, распечатывающую треугольник Паскаля:

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
….

Кол-во строк в треугольнике передается в виде первого аргумента.
Используйте метод toInt() для преобразования строки в число:
val n = “3”.toInt

ВАЖНО: постарайтесь написать чистый функциональный код, не использующий циклов, mutable переменных и return

Итерируем императивно

Цикл foreach:

for (arg <- args) println(arg)

Внутри **for** разворачивается в декларативную конструкцию

Цикл **for** с явным указанием границ итерации:

for (i <- 0 to args.length-1) println(args(i))

Цикл **while** и **do … while** (нет встроенных ++ и --):

var i = 0;

while (i<args.length) { println(args(i)); i = i + 1;}

Итерируем декларативно

Хвостовая рекурсия :

```
@tailrec def printArr(arr:Array[String], pos:Int) {
        if (pos<args.length) {
                println(arr(pos));  printArr(arr, pos+1)
        }
}; printArr(args, 0)
```

Вызываем функцию для всех элементов:

```
args.foreach(x=>println(x))
args.foreach(println _)
args.foreach(println)
```

Вызываем функцию для всех индексов:

```
(0 to args.length-1).foreach(i=>println(args(i)))
```

Задание №1-3

Написать программу, которая на вход принимает 4 числа, а на выходе указывает можно ли найти такие арифметические действия (умножение, деление, разность или сумма), чтобы приравнять результат первых двух чисел к результату вторых двух чисел. Если можно, то программа выводит Y, если нельзя, то N.

Например, на вход подается:
1 5 8 2
Программа отвечает Y, потому что 1+5 = 8-2

Для преобразования строки в число используйте функцию toInt, применяемую к объекту типа String, например: args(1).toInt
Синтаксис создания массива:
val arr:Array(Int) = Array(1,2,3) или val arr=Array(1,2,3)

Если использовать только 4-х символьные идентификаторы, то попытайтесь написать программу в одну строчку и меньше чем в 200 символов (не считая пробелы и объявление функции main)

Задание №1-4

Напишите рекурсивную функцию, которая проверяет строку на то, что для каждой открывающей скобки есть соответствующая закрывающая. Сигнатура функции:
def balance(chars: List[Char]): Boolean

Примеры корректных строк:
(if (zero? x) max (/ 1 x))
I told him (that it’s not (yet) done). (But he wasn’t listening)

Примеры некорректных строк:
:-)
())(

Строка конвертируется в список символов List[Char] с помощью функции toList:
"(just an) example".toList

Можно использовать следующие методы List:
chars.isEmpty: Boolean - returns whether a list is empty
chars.head: Char - returns the first element of the list
chars.tail: List[Char] - returns the list without the first element