Scala *Overview*

Fernando P. G. de Sá

ERAD 2019

05/09/2019

Paradigmas

Paradigma: Na ciência, um *paradigma* descreve conceitos distintos ou padrões de ideias em alguma disciplina científica.
Podemos enunciar os principais paradigmas de programação:

- programação imperativa
- programação funcional
- programação lógica

De forma ortogonal:

- programação orientada a objetos

Paradigma da Programação Imperativa

A programação imperativa consiste de

- modificação de variáveis mutáveis
- utilização de atribuições
- e controle de estruturas, (if-then-else), break, continue, return

Paradigma da Programação Funcional

- Em um senso restrito, programação funcional significa programação sem *loops*, variáveis mutáveis, atribuições e outras estruturas de controle imperativas
- Em um senso amplo, programação funcional significa focar em **funções**
- Em particular, funções podem ser valores que são produzidos, consumidos, e compostos
- Tudo isso torna-se mais fácil em uma linguagem funcional

Linguagens de Programação Funcional

- Em um senso restrito, uma linguagem de programação funcional não possui variáveis mutáveis, atribuições ou estruturas de controle imperativas
- Em um amplo senso, uma linguagem de programação funcional permite a construção de programas elegantes focados em funções
- Em particular, na programação funcional as funções são entidades privilegiadas:
  - elas podem ser deifinidas em qualquer lugar
  - como qualquer outro valor, elas podem ser passadas como parâmetro para funções e retornar como resultados
  - como para outros valores, existe um conjunto de operadores utilizados na composição de funções

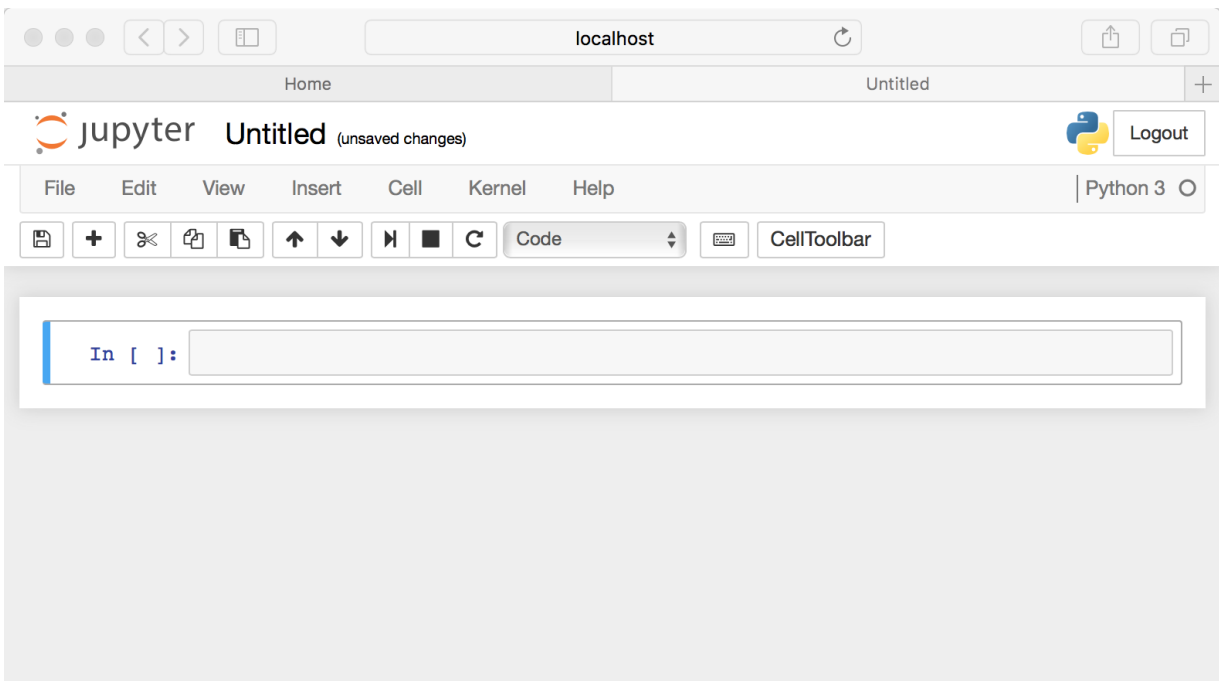
Exercícios Utilizando Scala
Vamos à prática!

Instruções:

- Abra o Jupyter Notebook, cujo ícone localiza-se na área de trabalho
- Aguarde.
- Duas janelas serão abertas: Navegador Web e o Terminal Linux
- Navegue pelas pastas e localize o notebook `/Códigos/Exercícios de Scala.ipynb`

Alguns atalhos úteis:

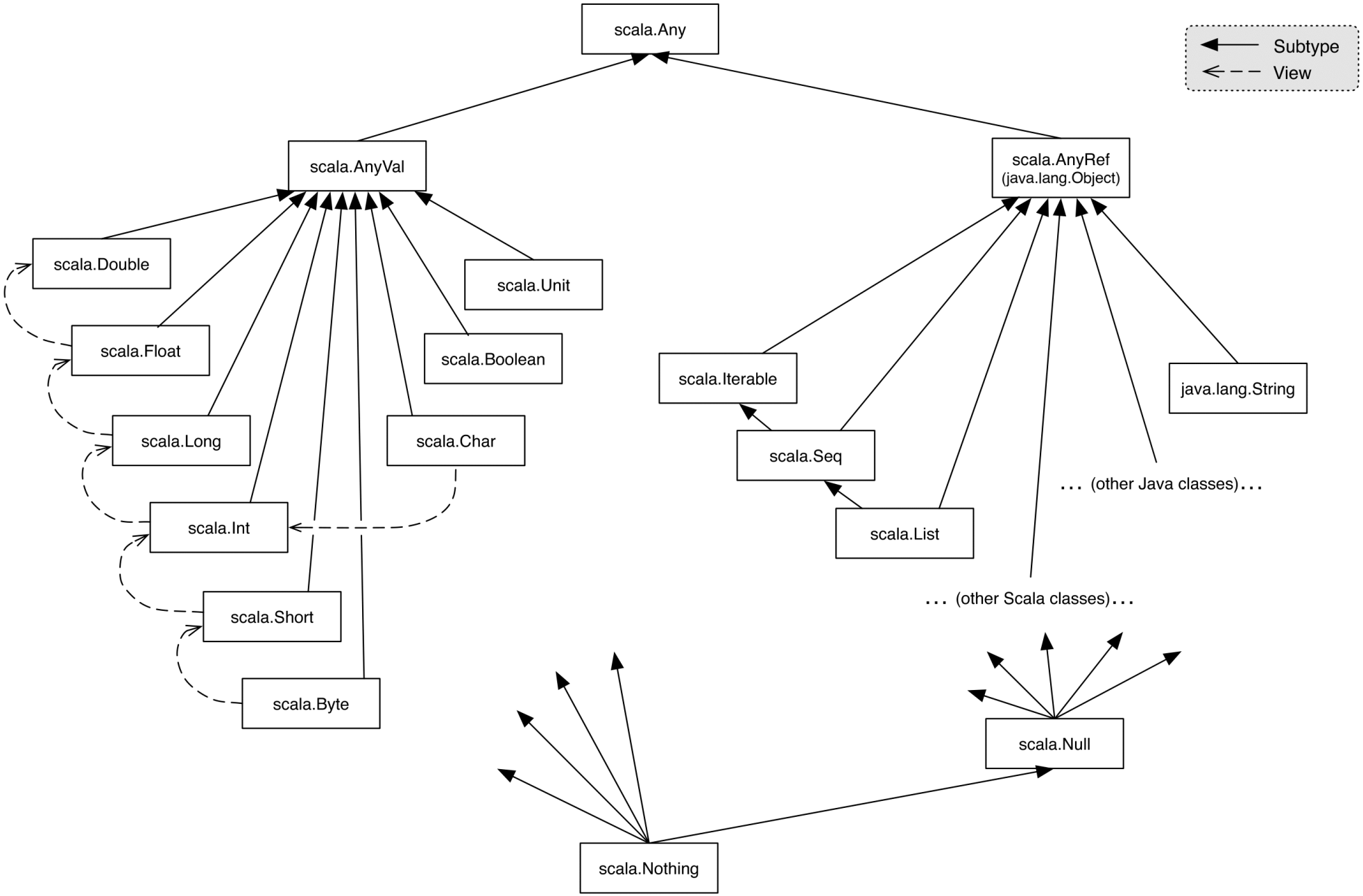
- **a**: insere uma nova célula anterior à atual
- **b**: insere uma nova célula posterior à atual
- **d + d**: remove a célula atual
- **Ctrl + Enter**: executa a célula atual
- **Ctrl + Shift + -**: divide a célula atual a partir da posição do cursor
- **Shift + M**: Aglutina as células selecionadas
- Maiores informações sobre o Jupyter: `https://jupyter.org/`



Operações algébricas:

- São exibidas as operações algébricas
- +(soma), −(diferença), ∗(multiplicação), /(divisão) e %(resto da divisão)

Name	Description	Size	Min	Max
Byte	Signed integer	1 byte	−127	128
Short	Signed integer	2 bytes	−32768	32767
Int	Signed integer	4 bytes	-2^{31}	$2^{31}-1$
Long	Signed integer	8 bytes	-2^{63}	$2^{63}-1$
Float	Signed floating point	4 bytes	n/a	n/a
Double	Signed floating point	8 bytes	n/a	n/a


scala.Any
Subtype
View
scala.AnyVal
scala.AnyRef
(java.lang.Object)
scala.Double
scala.Unit
scala.Float
scala.Boolean
scala.Long
scala.Char
scala.Int
scala.Short
scala.Byte
scala.Iterable
java.lang.String
scala.Seq
... (other Java classes)...
scala.List
... (other Scala classes)...
scala.Null
scala.Nothing

Declaração de Variáveis

Em Scala temos duas formas de declarar variáveis

- `var`: pode ser reatribuída
- `val`: não pode ser reatribuída

Tipos:

- `var` erad: Int = 2019
- `var` erad: String = "2019"
- `val` erad: Int = 2019
- `val` erad: String = "2019"

Funções em Scala

A função é identificada pela palavra-chave `def`, acompanhada pelo nome da função, que pode possuir em seu escopo de declaração os parâmetros de entrada. O tipo é declarado postriormente `:T`. Podemos declarar a função em uma única linha ou utilizando chaves `def(x:T): T = { }`, que nos permite utilizar múltiplas linhas se necessário.

```
def soma(x:Int,y:Int):Int = {
    x + y
}
```

Estruturas de Controle

Usamos exemplos: `if-else`, `for` e `while`

```
var nota: Int = 100 // digite aqui a idade

if(nota > 85 )
    print("Conceito␣A")
else if(nota > 75 && nota <= 85)
    print("Conceito␣B")
else if(nota >= 60 && nota <= 75)
    print("Conceito␣C")
else if(nota < 60)
    print("conceito␣D")
```

Estruturas de Controle

Usamos exemplos: `if-else`, `for` e `while`

```
var nota = 100

val resultado = nota match{
    case valor if 86 until 101 contains valor => "Conceito A"
    case valor if 76 until 86  contains valor => "Conceito B"
    case valor if 60 until 76 contains valor => "Conceito C"
    case _ => "Conceito D"
}
```

Estruturas de Controle

Usamos exemplos: `if-else`, `for` e `while`

```
var contador = 0

while (contador < 25){
    println("Erad␣2019")
    contador += 1
}
```

Estruturas de Controle

Usamos exemplos: `if-else`, `for` e `while`

```
import util.control.Breaks._

var contador = 0

breakable{
while (contador < 25){
    if(contador%3 > 1){
        break
    }
    println("Erad␣2019")
    contador += 1
    }
}
```

Estruturas de Controle

Usamos exemplos: `if-else`, `for` e `while`

```
import util.control.Breaks._

var contador = 0

breakable{
while (contador < 25){
    if(contador%3 > 1){
        break
    }
    println("Erad␣2019")
    contador += 1
    }
}
```

Estruturas de Controle

Usamos exemplos: `if-else`, `for` e `while`

```
val arrayA: Array[Int] = Array(1,2,3,4,5,6)

a.foreach(a => printf("O␣valor␣e␣%d␣\n",a))
```

Estruturas de Controle

Usamos exemplos: `if-else`, `for` e `while`

```
val arrayB: Array[Int] = Array(1,2,3,4,5,6)

for(i <- arrayB)
    printf("O␣valor␣e␣%d␣\n",i)
```

Estruturas de Controle

Usamos exemplos: `if-else`, `for` e `while`

```
for(i <- 1 until 10)
    println(i)
```

Estrutura de Dados

`Array` é uma estrutura de dados mutáveis, não redimensionável. Entretanto, podemos concatenar `Arrays` para formar um novo contendo todos os elementos desejados.

```
val valoresArray = Array("erad","erad")

val notasAlunosA = Array(86,87,90,98)

val notasAlunosB = Array(77,79,81)

// concatenamos
val notasConcat = notasAlunosA ++ notasAlunosB
```

Estrutura de Dados

`ArrayBuffer` é uma implementação da estrutura `Array` mutável e redimensionável

```
import scala.collection.mutable.ArrayBuffer

val notasAlunosC = ArrayBuffer(67,68,71)

notasAlunosC += 73

notasAlunosC ++= ArrayBuffer(74,75)

notasAlunosC ++= notasAlunosA

// elementos duplicados
notasAlunosC ++= notasAlunosA
```

Estrutura de Dados

List é uma estrutura de dados imutáveis, não-redimensionável. Entretanto, há implementações mutáveis, se desejado.

```
val listaNotaA   = List(86,87,90,98)

val listaNotaB   = List(77,79,81)

val listaNotaALL = listaNotaA ::: listaNotaB

val listaNotaExemplo2 = 100 :: listaNotaA

val geraLista   = 1 :: 2 :: 3 :: Nil
```

Estrutura de Dados

Pairs forma tuplas de dados.

```
val pair = ("erad",2019)

// acessando cada elemento do par
println(pair._1)

println(pair._2)
```

Estrutura de Dados

Set é uma estrutura de dados imutável redimensionável.

```
var jetSet = Set("Boeing","Airbus")

jetSet += "Lear"
```

Estrutura de Dados

Map é uma estrutura de dados que contêm uma chave que referencia um valor, que pode ser outra estrutura de dados qualquer.

```
// importamos uma versao mutavel
import scala.collection.mutable.Map

val mapValues = Map[Int, String]()

mapValues += (1 -> "erad")

mapValues += (2 -> "erad")

mapValues += (3 -> "erad")

// mesma tarefa realizada acima
val mapValues2 : Map[Int, String] =
Map(1 -> "erad",2 -> "erad",3 -> "erad")
```

Classes e Objetos

`class` é a palavra-chave que declara uma classe, uma abstração que reúne um conjunto de métodos, parâmetros, que descreve um objeto.

```
class ClasseErad {

    // declaracoes

}

var obj  = new ClasseErad
```

Classes e Objetos

`class` é a palavra-chave que declara uma classe, uma abstração que reúne um conjunto de métodos, parâmetros, que descreve um objeto.

```
class ClasseCefet {

    var valor = 10

}

var objCefet1 = new ClasseCefet

objCefet1.valor

objCefet1.valor = 20
```

Classes e Objetos

`class` é a palavra-chave que declara uma classe, uma abstração que reúne um conjunto de métodos, parâmetros, que descreve um objeto.

```
class ClasseCefet2 {

    private var valor = 10 // sem acesso externo

}
```

Classes e Objetos

`class` é a palavra-chave que declara uma classe, uma abstração que reúne um conjunto de métodos, parâmetros, que descreve um objeto.

```
class ClasseCefet3 {

    private var valor = 10 // sem acesso externo

    def add(a:Int):Int = {
        a + valor
    }
}

val objCefet3 = new ClasseCefet3
objCefet3.add(10)
```

Classes e Objetos

object é a palavra-chave que declara um objeto.

```
object ObjectCefet3 {

    private val valor = 10 // sem acesso externo

    def add(a:Int):Int = {
        a + valor
    }
}

objCefet3.add(10)
```

Classes e Objetos

`case class` é uma classe imutável.

```
case class ClasseCefet3(valor: Int  = 10)

var objCaseClass3 = ClasseCefet3(20)
```