Introdução à Linguagem Scala

Paradigmas de Linguagens de Programação

Daniel Terra Gomes
Ausberto S. Castro Vera

26 de novembro de 2021

UENF - UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE DARCY RIBEIRO

CCT - CENTRO DE CIÊNCIA E TECNOLOGIA
LCMAT - LABORATÓRIO DE MATEMÁTICAS
CC - CURSO DE CIÊNCIA DA COMPUTAÇÃO

Primeira edição, Setembro 2021

Sumário

1. Introdução

Scala é a abreviação das palavras SCalable LAnguage. Scala vem com o objetivo de suprir as necessidades dos desenvolvedores de Software modernos. Scala foi iniciada por Martin Odersky em 2001. A sua primeira publicação foi em 20 de janeiro de 2004. Martin, por sua vez, é um professor na School of Computer and Communication Sciences em Ecole Polytechnique Fédérale de Lausanne (EPFL). É uma linguagem com tipagem estática e dinâmica, orientada a objeto, funcional e de alta performance. Esses fundamentos possibilitam trabalhar em cima de problemas de maneira eficiente e com boa abstração. Scala deriva da linguagem de programação Java. Assim sendo, Scala é compilado em "Java-specific formar" chamado de Java Bytecode. Java Bytecode por sua vez é uma linguagem abstraída de máquina ou um conjunto de instruções executadas pelo Java Virtual Machine (JVM). Se pode pensar no JVM como um programa que executa outros programas como uma ambiente de máquina virtual e roda em uma Java Virtual Machine (JVM). Apesar das similaridades com Java há muitas diferenças entre as duas linguagens. A linguagem Scala vem com a intenção de superar o Java em suas limitações.

No geral, linguagens de programações são categorizadas nos segundos tipos:

- High level or Low Level: Sendo relacionado ao seu nível abstração do mundo real. Essa abstração diz sobre o quão próximo à linguagem de programação trabalha próximo ao que os computadores interpretam.
- Orientação a objeto: Se baseia no desenvolvimento de programas pensando em seus blocos; objetos. Esses objetos são programados pensando em suas intenções uns com os outros.
- Estático ou Dinâmico: Estético consiste em que os tipos de dados são checados antes do programa ser executado. Dinâmico, por outro lado, os tipos só são conhecidos após o programa estar em execução.

Ademais, podemos ver que Scala permite Tipagem estática dando a possibilidade de criar aplicações robustas, consertando muitas das falhas presente na linguagem Java.

A Orientação a Objeto (POO) é suportada em Scala disponibilizando uma maneira completa de trabalhar de diversas formas com os Objetos criados. Tudo em Scala é um Objeto. Adicionalmente a isso, a programação funcional permite que a linguagem trabalhe com Big Data. Pois há a possibilidade de valores imutáveis, funções sem efeitos colaterais. Dessa forma, o com o crescimento da linguagem Scala desde seu lançamento em 2004 com a sua presença crescente nas áreas mais atuais, como Big Data, mostra que é uma linguagem para a atualidade. [Ela19], [Ray13], [Wam21], [Xia20], [Hav14], [Dro12], [Wha20].

1.1 Aspectos históricos da linguagem Scala

O Design inicial da Linguagem de programação Scala foi iniciado no ano de 2001, e teve a sua primeira publicação em Janeiro de 2004. Desde seu projeto foi pensada para ser uma linguagem Funcional e Orientada a Objeto. Um dos princípios levados em conta nos primeiros passos foi de introduzir uma linguagem derivada do Java, sendo assim algo melhor que o Java. Mas conectando com as suas infraestruturas, logo o JVM e suas bibliotecas. Inicialmente, os desenvolvedores da linguagem Scala trabalhavam em uma linguagem chamada de Funnel, que buscava ser uma linguagem de classes puras, e de forma padronizada. Apesar de ser uma boa abordagem vista do âmbito acadêmico, mas não foi para os demais. Dessa forma, o time de desenvolvedores decidiram recomeçar no desenvolvimento de uma linguagem derivada do Java, que se encontrava no meio de uma linguagem para a comunidade acadêmica como a Funnel, mas também sendo uma linguagem prática. Sem demora, surgiu a linguagem Scala para cumprir esses requisitos.

1.2 Áreas de Aplicação da Linguagem

As aplicações da linguagem Scala varia entre o espaço acadêmico, e prático. Sendo assim, se aplica na área de Inteligência Artificial, primordialmente na parte de Processamento de Dados, Big Data, Machine Learning. Indo até ao Desenvolvimento Web trabalhando com Scala Js, e Scala Native. Fazendo uso extensivo do JVM e de bibliotecas Java.

1.2.1 Big Data

A linguagem Scala é usada no desenvolvimento de software na indústria para Big Data algumas das ferramentas usadas a partir da linguagem Scala e o Apache Spark e Apache Kafka que oferecem diversas APIs. Scala vem com o seu teor estatístico para a análise de dados nesses âmbitos. Entretanto, especialistas da área de Dados no geral por exemplo: Deep Learning(DL), Reinforcement learning (RL), and artificial intelligence (AI) que sao as areas maior destaque dentro do escopo de Machine Learning (ML). É possível ver que os desenvolvedores desses campos fazem uso majoritariamente da linguagem de programação Python, C + +. Devido a isso o uso dessas linguagens cresceram e o uso da Scala vem caindo comparado com as mesmas. A quantidade de dados produzidos vem crescendo a cada momento, e o processamento de dados para pequenos e grandes volumes de dados estão cada vez mais comuns. Assim para extrair informações valiosas no meio de milhares ou milhões de dados, ferramentas e Frameworks de processamento de dados. Scala se torna um dessas ferramentas para trabalhar com Big Data. Scala, diferentemente de umas das linguagens mais usadas para Big Data; Python, se diferencia na necessidade de especificar o seu tipo de

dado, sendo que Python não requisita isso. Ademais, o objetivo central da linguagem ao trabalhar com BIg Data é alcançar uma solução fácil de processamento, sendo assim com opções de processamento paralelo com pouca mudança no código inicial. [Wam21], [Xia20], [Hav14].

1.2.2 Programação Web

Scala também é usado para programação Web através do uso conjunto com JavaScript por meio do Scala.js. O Scala ainda está disponível no Scala Native. Atrelado com a sua possibilidade de programação funcional e Programação Orientada a Objeto (POO) proporciona uma maneira elegante e concisa de desenvolvimento. Como Scala é integrável com Java rodando através da JVM, os programadores usando Scala podem usar bibliotecas de Java de maneira direta, podendo até mesmo chamar códigos Scala a partir do Java. Por Scala ser uma linguagem de programação Open Source podendo ser usada por qualquer desenvolvedor, diminuindo os custos de desenvolvimento de páginas Web.

As vantagens do desenvolvimento Web com a linguagem Scala:

- Alta versatilidade;
- Alta performance e código mais enxuto;
- Comunidade Open Source;
- Vasto conjunto de ferramentas, bibliotecas.

Dessa forma, Scala vem para trazer essas vantagens com um ecossistema bem equipado de Ferramentas para a várias situações permitindo suporte modular para as necessidades de cada implementação.

Ferramentas para o Scala:

- Scalatra: Scalatra é um Framework que é uma forma de se conectar com Ruby Sinatra. Dando aos desenvolvedores uma forma poderosa de conectar o JVM com o Scala ajudando a criar páginas Web e APIs.
- BlueEyes: BlueEyes Framework focado em performance e composição.
- Akka HTTP: O Akka HTTP é uma implementação para o Scala pensando para o seu desenvolvimento como se fosse uma idealização de um formato. Traz em seus princípios de não ser uma Framework. Mas mais uma opção de ferramental universal para serviços HTTP. Desse modo, apesar de não ter sido a sua intenção inicial. É visto por muitos como um excelente Framework para o Scala.

A lista de opções de Framework para a linguagem Scala é imensa, caberá a cada desenvolvedor entender as suas necessidades e buscar entender mais das opções de ferramentas para trabalhar junto ao Scala.

[Wha20], [Dro12].

1.2.3 outras

- Engenharia de Dados;
- Infraestrutura de Sistemas;
- Computação distribuída.

2. Conceitos básicos da Linguagem Scala

Iniciando na linguagem Scala. Como já introduzido no capítulo anterior, Scala é bem similar a linguagem Java, sabendo que as duas fazem uso do Java Virtual Machine(JVM) para fazer a execução dos códigos desenvolvidos. Portanto, para os leitores desse material que já são familiarizados com a linguagem Java não terão dificuldades para entender como se desenvolve em Scala. [Wam21]

2.1 Variáveis e constantes

Variáveis reservam lugar em memória para serem usados durante a execução do programa. A linguagem Scala possui dos tipos de variáveis; "Imutáveis"e “mutáveis”. Sendo, imutáveis variáveis que não podem ter o seu valor alterado durante a execução do programa, esse tipo de variável é declarada com a palavra chave (val). Por outro lado, mutáveis podem ser alteradas durante a execução, declaradas com (var). [Wam21]

Exemplos de variáveis imutáveis e mutáveis:

```
    object Main {
  def main(args: Array[String]) {
      val b: Int = 230 // Imutaveis
      var c: Long = 9223372036854775807 // mutaveis
  }
}
```

2.2 Palavras reservadas

São um conjunto de palavras ou caracteres que são de uso exclusivo da linguagem. Essas palavras chaves não podem ser usadas como nome de variáveis. [Wam21]

Palavras chaves em Scala:

Palavras chaves		
abstract	throw	try
class	catch	super
override	private	extends
for	import	null
this	case	trait
protected	return	sealed
true	type	val
while	with	yield
def	do	else
false	final	finally
forSome	if	implicit
lazy	match	new
object	var	package
-	:	=
<-	</	<:
>:	@	=>

2.3 Tipos de Dados Básicos

Os tipos de dados básicos da linguagem Scala são objetos e são categorizados da seguinte forma: Int, Byte, Short, Long, Float, Double, Char, String, Boolean, Unit, Null, Nothing, AnyRef, Any. Sabendo que, os tipos de dados em Scala não tem tipo primitivo como no Java. Portanto, é possível chamar métodos em Long, Int, etc. [Wam21]

Tipo de dados em Scala:

Tipo de Dado	Descrição
Int	32 bit signed value. Tamanho -2147483648 até 2147483647
Boolean	true ou false
Byte	8 bit signed value. Tamanho -128 até 127
Short	16 bit signed value. Tamanho -32768 até 32767
Int	32 bit signed value. Tamanho -2147483648 até 2147483647
Long	64 bit signed value. -9223372036854775808 ate 9223372036854775807
Float	32 bit IEEE 754 single-precision float
Double	64 bit IEEE 754 double-precision float
Char	16 bit unsigned Unicode character. Tamanho de U+0000 to U+FFFF
String	Sequencia de caracteres
Unit	Sem valores correspondente
Null	null ou referência vazia
Nothing	Sub-tipo de qualquer outro valor; incluindo sem valores
Any	Sub-tipo de qualquer outro valor; Qualquer objeto é do tipo An

Exemplos de tipo de dados em Scala:

```
  object Main {
def main(args: Array[String]) {
    val a: Byte = 4
    val b: Int = 230
```

```
      val c: Long = 9223372036854775807
      val d: Short = -32768
      val e: Double = 742.0
  }
}
```

Importante mencionar que se o tipo de dado não for mencionado o compilador ira detectá-lo automaticamente.

2.4 Operadores e Expressões em Scala

Tudo em Scala pode se dizer uma expressão apenas (val) e (class) que não. Podemos dizer que uma expressão retornará um valor. Normalmente, é evitado fazer uso de Loops na linguagem Scala. Pois pode trazer problemas ao programa se mal implementado. [Wam21]

Exemplos de expressões em Scala:

- Expressão If:

```
val someCondition = true
val conditionedValue = if(someCondition) 10 else 5
```

- Expressão if/else:
  Uma outra forma de se usar condicional (if):

```
  if x < 10 then
    println("negative")
  else if x == 10 then
   println("10")
  else
    println("Nao eh igual a 10")
```

- Expressão Loop While:

```
var i = 0
  while(i < 11) {
    println(i)
    i += 1
  }
```

- Expressão Loop for:

```
var ints = List(1, 2, 3)
  for i <- ints do println(i)
```

- Expressão Match:

```
val a = Person("Daniel")

a match
  case Person(name) if name == "Daniel" =>
    println(s"$name says, Name found")

  case Person(name) if name == "Daniel T." =>
    println(s"$name says, Name found!")
```

- Expressão try/catch/finally:

```
try
  arquivoEscrita(text)
catch
  case nho: IOException => println("Palavras com
  final nho.")
  case lho: NumberFormatException => println("Pala
  vras com final lho.")
finally
  println("Fim").
```

Operadores condicionais:

Operador	Operação	descrição
ee	e	Os dois valores da operação são verdadeiros.
||	ou	Pelo menos um dos valores é verdadeiro.
>	maior que	O valor da esquerda é maior que o da direita.
>=	maior ou igual	O valor da esquerda é maior ou igual ao da direita.
<	menor que	O valor da esquerda é menor que o da direita.
<=	menor ou igual	O valor da esquerda é menor ou igual ao da direita.
==	igual	Os dois valores são iguais.
!=	diferetens	Os valores são diferentes.

3. Programação em Scala

3.1 Entradas e saídas

Como já mencionado Scala é muito semelhante à linguagem Java e ambos usam Java Virtual Machine (JVM) para executar o código. Portanto, Scala está próximo da linguagem Java. [Wam21]

Exemplo de saida de um Println:

```
object Hallo{
  def main(args: Array[String]) {
    println("Hallo Welt!")
  }
}
```

Exemplo de saida e entrada Scala:

```
object Entrada extends App {

    print("Primeiro nome: ")
    val pNome = readLine()

    print("Segundo nome: ")
    val sNome = readLine()

    println(s"Nome completo $pNome $sNome")

}
```

3.2 Seleção

Seleção é usado para tomar uma decisão com base nas condições. Um trecho de código é executado ou não é executado com base em uma condição especificada. Isso controla

o fluxo de execução do programa permitindo decidir os passs seguintes. [Wam21]
Exemplos de Seleção em Scala:

- if
- if-else
- Nested if-else
- if-else if ladder

Exemplo de Instruções:

- if/else:

```
    if x < 10 then
      println("negative")
    else if x == 10 then
     println("10")
    else
      println("Nao eh igual a 10")
```

- Nested if-else:

```
object teste {

def main(args: Array[String]) {

    var a: Int = 20
    var b: Int = 100
    var c: Int = 200

    if (a > b) // condition1
    {
        if(a > c) // condition2
        {
            println("o maior (a)");
        }

        else
        {
            println("o maior (c)")
        }

    }

    else
    {

        if(b > c)  // condition3
        {
            println("o maior (b)")
        }

        else
```

```
            {
                println("o maior (c)")
            }
        }
    }
```

3.3 Repetição

Repetições são uma parte muito importante de qualquer linguagem de programação. Isso nos permite executar um conjunto de código várias vezes. [Wam21]

Exemplos de repetições em Scala:

- While
- Do while
- For

Exemplo de Instruções:

- for:

```
var ints = List(1, 2, 3)
  for i <- ints do println(i)
```

- while:

```
var i = 0
  while(i < 11) {
    println(i)
    i += 1
  }
```

3.4 Classes e Objetos

As classes são projetos para a criação de objetos. Quando definimos uma classe, podemos então criar novos objetos (instâncias) da classe. [Wam21]Portanto,

- Classe é um projeto de um objeto.
- Objeto nada mais é do que uma instância de uma classe.

Exemplo de Classes e seus objetos em Scala:

```
class Empregado(idc:Int,namec:String) {
    var ID:Int=idc
    var nome:String=namec

    def getNome(): String =
    {
       nome
    }

}

object Main {
  def main(args: Array[String]): Unit = {
```

```
    val empr1:Empregado =new Empregado(23,"Daniel")
    val empr2:Empregado =new Empregado(23,"Terra")
    print(emp1.getNome())

  }
}
```

3.5 Funções

Scala também oferece suporte a uma abordagem de programação funcional. Portanto, possui funções integradas e também permite criar funções personalizadas. Funções são valores principais em Scala. Dessa forma, é possível armazenar o valor de uma função, passar uma função como um argumento e retornar uma função como um valor de outra função. Tornando possível o uso de POO - (Programação Orientada a Objeto). [Wam21]

Exemplo de Syntax de função em Scala:

```
def funcNome(parameters : type) : return type = {
 // corpo da funcao
}
```

Exemplo de função em Scala:

```
object Main {
  def printFunc(n:Int): Unit ={
    if(n<0)
    return ;
    println(n)
    printFun(n-1)
    }

    def main(args: Array[String]): Unit = {
      printFun(3);
      }
      }
```

3.6 Herança

Herança é um pilar importante da OOP (Programação Orientada a Objetos). É o mecanismo em Scala pelo qual uma classe pode herdar as funções de outras classes. [Wam21]

Informações Importantes:

- Superclasse: a classe cujas características são herdadas é conhecida como superclasse ou uma classe base ou uma classe pai.
- Subclasse: a classe que a outra classe herda é chamada de subclasse ou uma classe derivada, classe estendida ou classe filha. A subclasse pode adicionar seus próprios campos e métodos, além dos campos e métodos da superclasse.
- Reutilização: a herança apóia o conceito de polimofismo; se quisermos criar uma nova classe e já houver uma classe que contenha parte do código que desejamos, podemos derivar nossa nova classe da classe existente. Desta forma, reutilizamos os campos e métodos da classe existente.

Exemplo da Syntax de Herança em Scala:

```
class parent_class_name extends child_class_name{
// metodos etc
}
```

Exemplo de Herança em Scala:

```

class Aluno{
    var Name: String = "Daniel"
}

class Aluno1 extends Aluno
{
    var num: Int = 20

    def details()// Metodo
    {
    println("Nome aluno: " +Nome);
    println("Total de materias: " +num);
    }
}

object Main
{

    def main(args: Array[String])
    {

        val ts = new Aluno1();
        ts.details();
    }
}
```

4. Aplicações da Linguagem Scala

A seguir serão apresentados um será de aplicações com a Linguagem Scala. Dessa forma, será possível ver na prática o que é possível ser feito na linguagem. [Ale13] Entre os exemplos estão os seguintes:

- Operações básicas aritmética;
- O algoritmo Quicksort;
- Programa de Cálculo Numérico, equação linear;
- Aplicação usando Matrizes
- Aplicações Profissionais da linguagem de Linear Regression usando Spark.

4.1 Operações básicas

Exemplo algoritmo operador aritmético: [Ale13]

```
object Atitmetico {
// criando o objeto
  def main(args:Array[String]) {
    var x = 50;
    var y = 10;
      // soma de x + y
      println("Add of x + y = " + (x + y));
      // Subitracao entre x e y
      println("Sub of x - y = " + (x - y));
      // Multiplicacao dos valores apresentados
      println("Multi of x * y = " + (x * y));
      // Divisao x e y
      println("Divisao of x / y = " + (x / y));
      // Modulo dos valores
      println("Mod of x % y = " + (x % y));
```

```
        }
      }
```

4.2 O algoritmo Quicksort em Scala

Exemplo algoritmo Quicksort em Scala: [Ale13]

```
// Implementa um exemplo para quicksort.
// O algoritmo funciona da seguinte maneira:
// Mova da esquerda para a direita todos os
// valores
// que sao menores que
// o valor pivo para o
// a esquerda de uma divisao flutuante, em que
// a divisao
// sempre marca
// o ponto para
// a esquerda disso, todos os valores sao menores
// que o pivo.
// Mova-se no
// final
// o valor pivo para a posicao de divisao.
// Em seguida,
// recursao para
// os subarrays
// em ambos os lados do ponto de pivo.



object Quicksort {
  def main(args: Array[String]) {
    var mess = Array(3, 9, 8, 13, 2, 5, 4);

    quicksort(mess, 0, mess.length-1);
    mess.foreach( println )
  }

  // Troca dois valores em uma matriz
  def swap(a: Array[Int], pos1: Int, pos2: Int):
   Unit = {
    val stash = a(pos1)
    a(pos1) = a(pos2)
    a(pos2) = stash
  }

  // Executa a classificacao rapida recursiva
  // em uma matriz
  // Existe uma opcao para limitar a complexidade
  // do espaco
```

```
    // para
    // O (ln (n)) aqui, quando da ultima vez recursiva
    // na
    // particao maior
    // (Recursao da cauda).

    def quicksort(a: Array[Int], low: Int, hi: Int):
     Unit = {
      if (low < hi) {
        val p = partition(a, low, hi)
        quicksort(a, low, p-1)
        quicksort(a, p+1, hi)
      }
    }

    // Particionar uma matriz em torno de um pivo
    // Seleciona um pivo, desloca todos os valores
    // menores que
    // o pivo para a esquerda e desloca todos
    // os valores
    // maior do que o ponto de pivo a direita dele.
    //
    // Para otimizacao: escolha um pivo de acordo
    // com uma
    // mediana
    // Valor de uma amostra. Isso ajuda que
    // O particionamento leva a submatrizes de
    // tamanhos
    // semelhantes,
    // e, portanto, o desempenho ideal de
    // O (n // ln (n)).
    //
    // Indice de retorno (localizacao) do pivo
    // na matriz
    // particionada

    def partition(subArray: Array[Int], low: Int,
     hi: Int):
      Int = {
      val pivot = hi;
      var i = low;
      for (
        j <- low to hi
        if subArray(j) < subArray(pivot)
      ) {swap(subArray, i, j); i+=1}

      // Finalmente, mova o valor pivo para a linha
      // divisoria
      // em i
```

```
        swap(subArray, i, pivot);
        return i
      }
    }
```

4.3 Programa de Cálculo Numérico

Os seguintes exemplos fazem uso da Biblioteca Apache Commons Math é um projeto Apache que visa resolver os problemas matemáticos e estatísticos mais comuns que não estão disponíveis na linguagem Java padrão. Ele suporta classes de matriz densa e esparsa que são equipadas com operações básicas, bem como algoritmos de decomposição de matriz. [Ale13]

Exemplo algoritmo calculando uma equação linear:

```
// Import biblioteca
import org.apache.commons.math3.linear._

// Crie a matriz
val A = new Array2DRowRealMatrix(3, 6)

// Preencha a matriz com numeros aleatorios
val r = new scala.util.Random(0)

for(i <- 0 until A.getRowDimension())
    for(j <- 0 until A.getColumnDimension())
        A.setEntry(i, j, r.nextDouble())

// Defina o primeiro valor como o ultimo valor
A.setEntry(0, 0,
    A.getEntry(A.getRowDimension() - 1, A.
      getColumnDimension() - 1))

// Obtenha uma submatriz, 1 linha a 3 linha,
// 1 coluna a 3 coluna
val B = A.getSubMatrix(0, 2, 0, 2)

// Resolva a equacao linear
val solver = new LUDecomposition(B).getSolver()
val a = A.getColumnVector(0)
val x = solver.solve(a)
```

4.4 Aplicação usando Matrizes

Exemplo algoritmo calculo de produto de matrizes: [Ale13]

```
// Obtenha uma submatriz, 1 linha a 3
// linha, 1 coluna a
// 3 coluna
val B = A.getSubMatrix(0, 2, 0, 2)
```

```
    // Defina uma sub-matriz de A a B, da
    // 1 linha a 3 linha,
    // 2 coluna a 4 coluna
    A.setSubMatrix(B.getData(), 0, 1)

    // Produto de matriz C = A'B
    val C = A.transpose().multiply(B)
```

4.5 Aplicações Profissionais

Aplicações de Linear Regression usando Spark desenvolvido em Scala.

Nessa aplicação iremos fazer um modelo para predizer a nota de alunos com base na sua nota anterios e seu tempo de estudo. [Nic17]

```
    //Instale todas as dependencias no ambiente
    //Apache Spark
    //2.4.4 with hadoop 2.7, Java 8 e Findspark
    //para localizar o
    //Spark no sistema
    !apt-get install openjdk-8-jdk-headless -qq > /
    dev/null
    !wget -q http://apache.osuosl.org/spark/
    spark-2.4.4/
    spark-2.4.4-bin-hadoop2.7.tgz
    !tar xf spark-2.4.4-bin-hadoop2.7.tgz
    !pip install -q findspark

    // Variaveis de ambiente
    import os
    os.environ["JAVA_HOME"] =
    "/usr/lib/jvm/java-8-openjdk-amd64"
    os.environ["SPARK_HOME"] =
    "/content/spark-2.4.4-bin-hadoop2.7"

    //Iniciar sessao do Spark
    import findspark
    findspark.init()
    from pyspark.sql import SparkSession
    spark =
    SparkSession.builder.master("local[*]").getOrCreate
    ()
    //Carregar o arquivo Student_Grades_Data.csv do
    //sistema local para o local remoto da colab
    from google.colab import files
    files.upload()

    //Carregando o arquivo Student_Grades_Data.csv,
    // carregado na etapa anterior
    data = spark.read.csv('Student_Grades_Data.csv',
     header=True, inferSchema=True)
```

```
//Dando uma olhada no tipo de dados de cada
//coluna para ver quais tipos de dados o parametro
//inferSchema = TRUE definiu para cada coluna
data.printSchema()

//root
// |-- Time_to_Study: integer (nullable = true)
// |-- Grades: double (nullable = true)

//Exibir as primeiras linhas de dados
data.show()
//+--------------+------+
//|Time_to_Study|Grades|
//+--------------+------+
//|             1|   1.5|
//|             5|   2.7|
//|             7|   3.1|
//|             3|   2.1|
//|             2|   1.8|
//|             9|   3.9|
//|             6|   2.9|
//|            12|   4.5|
//|            11|   4.3|
//|             2|   1.8|
//|             4|   2.4|
//|             8|   3.5|
//|            13|   4.8|
//|             9|   3.9|
//|            14|   5.0|
//|            10|   4.1|
//|             6|   2.9|
//|            12|   4.5|
//|             1|   1.5|
//|             4|   2.4|
//+--------------+------+
//only showing top 20 rows

//Crie uma matriz de recurso omitindo
// a ultima coluna
feature_cols = data.columns[:-1]
from pyspark.ml.feature import VectorAssembler
vect_assembler = VectorAssembler
(inputCols=feature_cols,outputCol="features")

//Utilize o Assembler criado acima
//para adicionar a coluna de feicao
data_w_features = vect_assembler.transform(data)
```

```
// Exibe os dados com colunas adicionais
//denominadas recursos.
//Se fosse um problema de regressao linear
//multipla, voce poderia ver todos os
// valores de variaveis independentes
// combinados em uma lista
data_w_features.show()
//+-------------+------+--------+
//|Time_to_Study|Grades|features|
//+-------------+------+--------+
//|            1|   1.5|   [1.0]|
//|            5|   2.7|   [5.0]|
//|            7|   3.1|   [7.0]|
//|            3|   2.1|   [3.0]|
//|            2|   1.8|   [2.0]|
//|            9|   3.9|   [9.0]|
//|            6|   2.9|   [6.0]|
//|           12|   4.5|  [12.0]|
//|           11|   4.3|  [11.0]|
//|            2|   1.8|   [2.0]|
//|            4|   2.4|   [4.0]|
//|            8|   3.5|   [8.0]|
//|           13|   4.8|  [13.0]|
//|            9|   3.9|   [9.0]|
//|           14|   5.0|  [14.0]|
//|           10|   4.1|  [10.0]|
//|            6|   2.9|   [6.0]|
//|           12|   4.5|  [12.0]|
//|            1|   1.5|   [1.0]|
//|            4|   2.4|   [4.0]|
//+-------------+------+-------
//only showing top 20 rows

//Selecione apenas recursos e rotulo do conjunto
//de dados anterior, pois precisamos dessas duas
//entidades para construir o modelo de aprendizado
//de maquina
finalized_data = data_w_features.select
("features","Grades")

finalized_data.show()

//+--------+------+
//|features|Grades|
//+--------+------+
//|   [1.0]|   1.5|
//|   [5.0]|   2.7|
//|   [7.0]|   3.1|
//|   [3.0]|   2.1|
```

```
//|    [2.0]|    1.8|
//|    [9.0]|    3.9|
//|    [6.0]|    2.9|
//|   [12.0]|    4.5|
//|   [11.0]|    4.3|
//|    [2.0]|    1.8|
//|    [4.0]|    2.4|
//|    [8.0]|    3.5|
//|   [13.0]|    4.8|
//|    [9.0]|    3.9|
//|   [14.0]|    5.0|
//|   [10.0]|    4.1|
//|    [6.0]|    2.9|
//|   [12.0]|    4.5|
//|    [1.0]|    1.5|
//|    [4.0]|    2.4|
//+--------+------+
//only showing top 20 rows

//Divida os dados em treinamento e modelo
//de teste com 70% obs. indo em
//treinamento e 30% em testes
train_dataset, test_dataset =
finalized_data.randomSplit([0.7, 0.3])

//Peek into training data
train_dataset.describe().show()

//+-------+------------------+
//|summary|            Grades|
//+-------+------------------+
//|  count|                31|
//|   mean|3.3451612903225802|
//| stddev|1.1609877144562206|
//|    min|               1.5|
//|    max|               5.0|
//+-------+------------------+

//Classe de regressao linear de
//importacao chamada LinearRegression
from pyspark.ml.regression import LinearRegression

//Crie o objeto de regressao linear
//denominado tendo coluna de feicao
//como feicoes e coluna de rotulo
//como Time_to_Study
LinReg = LinearRegression
(featuresCol="features", labelCol="Grades")
```

```
//Treine o modelo no treinamento
// usando o metodo fit ().
model = LinReg.fit(train_dataset)

//Preveja as notas usando o metodo evulate
pred = model.evaluate(test_dataset)

//Mostrar os valores de Grau previstos
//ao lado dos valores de Grau reais
pred.predictions.show()

//+--------+------+------------------+
//|features|Grades|        prediction|
//+--------+------+------------------+
//|    [1.0]|   1.5|1.5663703357117775|
//|    [1.0]|   1.5|1.5663703357117775|
//|    [2.0]|   1.8|1.8366768043045956|
//|    [3.0]|   2.1|2.1069832728974136|
//|    [3.0]|   2.1|2.1069832728974136|
//|    [4.0]|   2.4| 2.377289741490232|
//|    [4.0]|   2.4| 2.377289741490232|
//|    [4.0]|   2.4| 2.377289741490232|
//|    [7.0]|   3.1| 3.188209147268686|
//|    [7.0]|   3.1| 3.188209147268686|
//|    [7.0]|   3.1| 3.188209147268686|
//|    [8.0]|   3.5|3.4585156158615042|
//|    [8.0]|   3.5|3.4585156158615042|
//|    [8.0]|   3.5|3.4585156158615042|
//|    [9.0]|   3.9|3.7288220844543223|
//|   [10.0]|   4.1|3.9991285530471403|
//|   [10.0]|   4.1|3.9991285530471403|
//|   [12.0]|   4.5| 4.539741490232776|
//|   [13.0]|   4.8| 4.810047958825595|
//+--------+------+------------------+

//Descubra o valor do coeficiente
coefficient = model.coefficients
print ("The coefficient of the model
is : %a" %coefficient)
//The coefficient of the model is
//: DenseVector([0.2703])

//Descubra o valor de interceptacao
intercept = model.intercept
print ("The Intercept of the model is
 : %f" %intercept)
//The Intercept of the model is : 1.296064

//Avalie o modelo usando metricas como
```

```
    // erro medio absoluto (MAE), erro
    //quadratico medio (RMSE) e R-quadrado
    from pyspark.ml.evaluation import
    RegressionEvaluator
    evaluation = RegressionEvaluator
    (labelCol="Grades", predictionCol="prediction")

    // Erro de raiz quadrada media
    rmse = evaluation.evaluate
    (pred.predictions, {evaluation.metricName: "rmse"})
    print("RMSE: %.3f" % rmse)

    // Erro Quadrado Medio
    mse = evaluation.evaluate
    (pred.predictions, {evaluation.metricName: "mse"})
    print("MSE: %.3f" % mse)

    // Erro Medio Absoluto
    mae = evaluation.evaluate
    (pred.predictions, {evaluation.metricName: "mae"})
    print("MAE: %.3f" % mae)

    // r2 - coeficiente de determinacao
    r2 = evaluation.evaluate
    (pred.predictions, {evaluation.metricName: "r2"})
    print("r2: %.3f" %r2)
    //RMSE: 0.069
    //MSE: 0.005
    //MAE: 0.056
    //r2: 0.995

    //Crie um conjunto de dados nao rotulado
    //para conter apenas a coluna de recurso
    unlabeled_dataset = test_dataset.select
    ('features')

    Display the content of unlabeled_dataset
    unlabeled_dataset.show()
   // +--------+
   // |features|
   // +--------+
   // |   [1.0]|
   // |   [1.0]|
   // |   [2.0]|
   // |   [3.0]|
   // |   [3.0]|
   // |   [4.0]|
   // |   [4.0]|
   // |   [4.0]|
```

```
// |    [7.0]|
// |    [7.0]|
// |    [7.0]|
// |    [8.0]|
// |    [8.0]|
// |    [8.0]|
// |    [9.0]|
// |   [10.0]|
// |   [10.0]|
// |   [12.0]|
// |   [13.0]|
// +--------+

//Prever a saida do modelo para dados de
//teste novos e nao vistos usando o metodo
//transform ()
new_predictions =
model.transform(unlabeled_dataset)

//Display the new prediction values
new_predictions.show()
//+--------+------------------+
//|features|        prediction|
//+--------+------------------+
//|    [1.0]|1.5663703357117775|
//|    [1.0]|1.5663703357117775|
//|    [2.0]|1.8366768043045956|
//|    [3.0]|2.1069832728974136|
//|    [3.0]|2.1069832728974136|
//|    [4.0]| 2.377289741490232|
//|    [4.0]| 2.377289741490232|
//|    [4.0]| 2.377289741490232|
//|    [7.0]| 3.188209147268686|
//|    [7.0]| 3.188209147268686|
//|    [7.0]| 3.188209147268686|
//|    [8.0]|3.4585156158615042|
//|    [8.0]|3.4585156158615042|
//|    [8.0]|3.4585156158615042|
//|    [9.0]|3.7288220844543223|
//|   [10.0]|3.9991285530471403|
//|   [10.0]|3.9991285530471403|
//|   [12.0]| 4.539741490232776|
//|   [13.0]| 4.810047958825595|
//+--------+------------------+
```

5. Ferramentas existentes e utilizadas

Neste capítulo encontrará o processo de criação desse livro, as ferramentas utilizadas para processar e visualizar os códigos que foram apresentados. Portanto, nome de ferramentas, IDE, compiladores e suas versões serão apresentados a seguir. Em cada caso será levado em consideração os seguintes critérios:

- Nome da ferramenta (compilador-interpretador);
- Versão atual e utilizada;
- Captura de tela da ferramenta;
- Informações adicionais.

5.1 Editores para Scala

No processo de desenvolvimento das aplicações e programação apresentados neste livro foi utilizado a ferramenta de edição Visual Studio Code na sua última versão "October 2021 (version 1.62)". Visual Studio Code é um editor de código-fonte da Microsoft para Windows, Linux e macOS. Os recursos incluem suporte para depuração, realce de sintaxe, autocompletar de código inteligente, snippets, refatoração de código e Git incorporado. Link para o site oficial: Visual Studio Code - Code Editing.

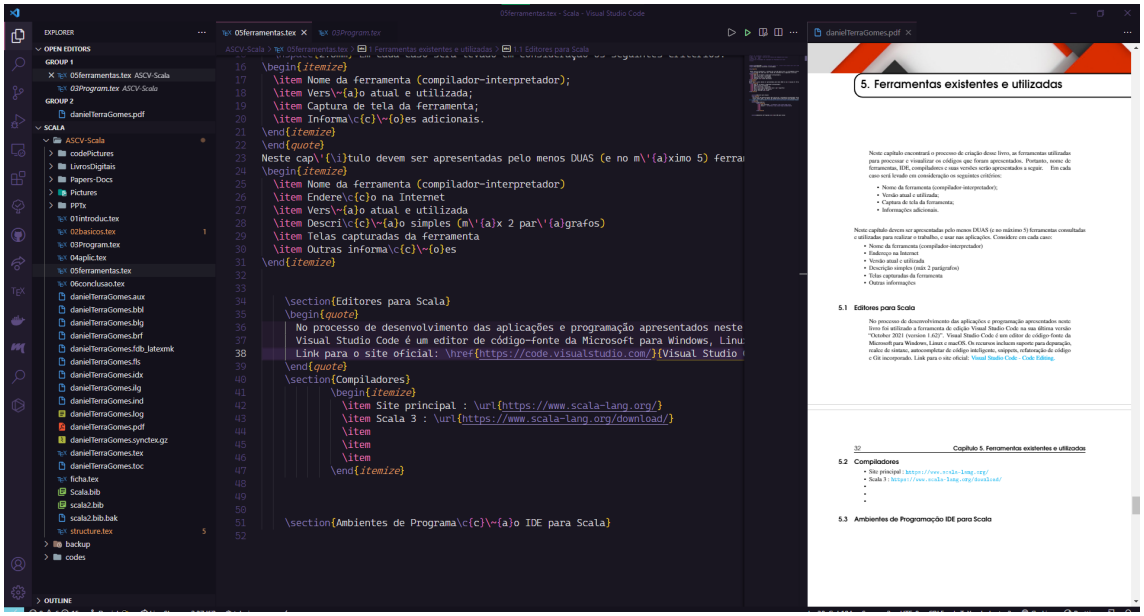

Figura 5.1: Visual Studio Code - Code Editing.

Fonte: O autor

5.2 Compiladores

O códigos desenvolvidos foram compilados usando a seguinte versão do compilador Scala: “Scala compiler version 2.13.6 – Copyright 2002-2021, LAMP/EPFL and Lightbend, Inc”. Link para o site oficial: The Scala Programming Language Ressalvo que a última versão do compilador Scala e a seguinte: “Scala compiler version 3.1.0 – Copyright 2002-2021, LAMP/EPFL and Lightbend, Inc.”

```
PS C:\Users\danie\Documents\Faculdade\21-2_Paradigmas\Scala> scalac -version
Scala compiler version 2.13.6 -- Copyright 2002-2021, LAMP/EPFL and Lightbend, Inc.
PS C:\Users\danie\Documents\Faculdade\21-2_Paradigmas\Scala>
```

Figura 5.2: Scala compiler version 2.13.6

Fonte: O autor

5.3 Ambientes de Programação IDE para Scala

Uma IDE, do inglês “Integrated Development Environment”, permite que os programadores consolidem os vários aspectos da escrita de um programa de computador. IDEs aumentam a produtividade do programador combinando atividades comuns de escrita de software em um único aplicativo: edição de código-fonte, criação de executáveis e depuração.

NetBeans:

O NetBeans IDE permite que os desenvolvedores desenvolvam de forma rápida e fácil aplicativos de desktop, móveis e web. Graças às suas muitas funções de edição, análise e conversão, o NetBeans IDE torna o trabalho mais fácil para os desenvolvedores. A ferramenta de gerenciamento de projetos por si só vale a pena dar uma olhada.

O plug-in Scala para NetBeans oferece um editor Scala completo com sintaxe e coloração semântica, um navegador de estrutura de tópicos, autocompletar código e muito mais. Há também um depurador, console interativo e integração com Junit e Maven.

IDE Scala para Eclipse:

Há, também, uma extensão Scala para Eclipse. Esta IDE Scala oferece suporte dedicado para o desenvolvimento de Scala puro e aplicativos mistos. Scala IDE oferece aos desenvolvedores uma variedade de ferramentas e recursos, bem como algumas notaveis correcoes de bugs. Ferramentas de edição avançadas incluem autocompletar código, realce implícito e semântico e um guia de indentação completamente novo. Há um depurador Scala brilhante para facilitar sua vida, junto com um localizador de teste Junit confiável e depurador assíncrono.

6. Conclusões

Diante de tudo que aprendemos durante essa jornada. Nos encontramos aqui, mais fortes e mais sábios. Mesmo com os problemas enfrentados na hora de executar muitos dos códigos apresentados, versão do compilador desatualizada, dificuldades com o entendimento da linguagem Scala. Sabendo que, este livro foi produzido pensando em ser uma forma rápida de recapitular conceitos essenciais na hora de programar em uma das suas linguagens favoritas; Scala. Dessa forma, foi alcançado o objetivo desejado de entregar algo que possibilita um aprendizado rápido de umas das linguagens de programação mais usadas do mercado. Abrindo portas para programação Web, Computação distribuída, Data Science, programação funcional (cujo não cobrimos durante este livro) etc. Sendo assim, é altamente recomendado que no futuro venha aprender os conceitos da programação funcional na linguagem de programação Scala. Esse novo paradigma de programação proporcionará novas formas de enxergar os problemas que encontrará durante a sua carreira na academia e no mercado de trabalho.

Nossa jornada não termina aqui. Deixo uma recomendação de um livro que trabalha a fundamentar os conceitos de programação funcional e de programação orientada a objeto. Segue abaixo o material recomendado para continuar os estudos em Scala.

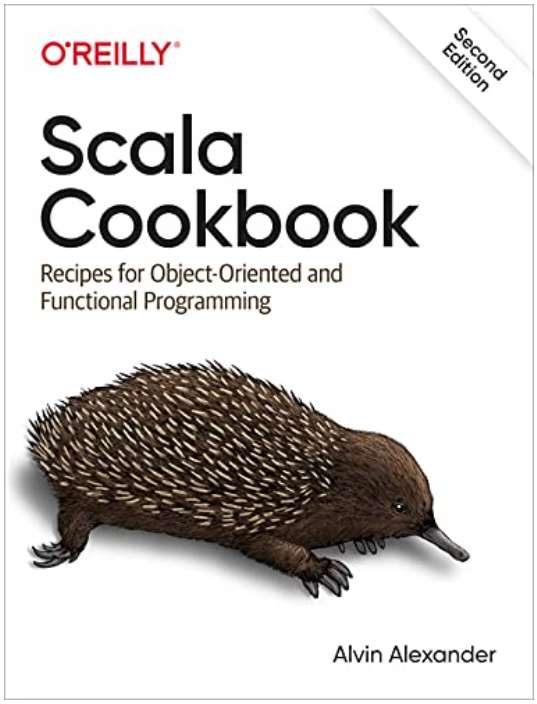

Figura 6.1: Scala Cookbook: Recipes for Object-Oriented and Functional Programming

Fonte: Amazon Serviços de Varejo do Brasil Ltda.

Referências Bibliográficas

[Ale13] Alvin Alexander. *Scala cookbook*. O'Reilly, Sebastopol, CA, 2013. Citado 3 vezes nas páginas 19, 20 e 22.

[Dro12] Sadek Drobi. Play2: A new era of web application development. *IEEE Internet Computing*, 16(4), 2012. Citado 2 vezes nas páginas 6 e 7.

[Ela19] Irfan Elahi. *Scala Programming for Big Data Analytics*. APRESS L.P., Notting Hill, VIC, Australia, 1 edition, July 2019. Citado na página 6.

[Hav14] Klaus Havelund. Data automata in scala. In *2014 Theoretical Aspects of Software Engineering Conference*, 2014. Citado 2 vezes nas páginas 6 e 7.

[Nic17] Patrick R. Nicolas. *Scala for Machine Learning, Second Edition*. Packt Publishing, September 2017. Citado na página 23.

[Ray13] Nilanjan Raychaudhuri. *Scala in Action: Covers Scala 2.10*. MANNING PUBN, Shelter Island, NY, 2 edition, April 2013. Citado na página 6.

[Wam21] Dean Wampler. *Programming Scala*. O'Reilly Media, Inc., Sebastopol, CA, 3 edition, July 2021. Citado 9 vezes nas páginas 6, 7, 9, 10, 11, 13, 14, 15 e 16.

[Wha20] Richard Whaling. *Modern Systems Programming with Scala Native: Write Lean, High-Performance Code Without the Jvm*. PRAGMATIC BOOKSHELF, 1 edition, February 2020. Citado 2 vezes nas páginas 6 e 7.

[Xia20] Guangming Xian. Parallel machine learning algorithm using fine-grained-mode spark on a mesos big data cloud computing software framework for mobile robotic intelligent fault recognition. *IEEE Access*, 8, 2020. Citado 2 vezes nas páginas 6 e 7.

Disciplina: *Paradigmas de Linguagens de Programação 2021*
Linguagem: *Linguagem Scala*
Aluno: *Daniel Terra Gomes*

Ficha de avaliação:

Aspectos de avaliação (requisitos mínimos)	**Pontos**
Elementos básicos da linguagem (Máximo: 01 pontos) • Sintaxe (variáveis, constantes, comandos, operações, etc.) • Usos e áreas de Aplicação da Linguagem	
Cada elemento da linguagem (definição) com exemplos (Máximo: 02 pontos) • Exemplos com fonte diferenciada (Courier , 10 pts, azul)	
Mínimo 5 exemplos completos - Aplicações (Máximo : 2 pontos) • Uso de rotinas-funções-procedimentos, E/S formatadas • Menu de operações, programas gráficos, matrizes, aplicações	
Ferramentas (compiladores, interpretadores, etc.) (Máximo : 2 pontos) • Ferramentas utilizadas nos exemplos: pelo menos DUAS • Descrição de Ferramentas existentes: máximo 5 • Mostrar as telas dos exemplos junto ao compilador-interpretador • Mostrar as telas dos resultados obtidos nas ferramentas • Descrição das ferramentas (autor, versão, homepage, tipo, etc.)	
Organização do trabalho (Máximo: 01 ponto) • Conteúdo, Historia, Seções, gráficos, exemplos, conclusões, bibliografia	
Uso de Bibliografia (Máximo: 01 ponto) • Livros: pelo menos 3 • Artigos científicos: pelo menos 3 (IEEE Xplore, ACM Library) • Todas as Referências dentro do texto, tipo [ABC 04] • Evite Referências da Internet	
Conceito do Professor (Opcional: 01 ponto)	
Nota Final do trabalho:	

Observação: Requisitos mínimos significa a *metade* dos pontos