Introdução à Linguagem Scala

Paradigmas de Linguagens de Programação

Marcos Bruno P. Campos
Ausberto S. Castro Vera

June 12, 2024

Disciplina: *Paradigmas de Linguagens de Programação 2023*
Linguagem: *Scala*
Aluno: *Marcos Bruno Pereira Campos*

Ficha de avaliação:

Aspectos de avaliação (requisitos mínimos)	**Pontos**
Introdução (Máximo: 01 pontos) • Aspectos históricos • Áreas de Aplicação da linguagem	
Elementos básicos da linguagem (Máximo: 01 pontos) • Sintaxe (variáveis, constantes, comandos, operações, etc.) • Cada elemento com exemplos (código e execução)	
Aspectos Avançados da linguagem (Máximo: 2,0 pontos) • Sintaxe (variáveis, constantes, comandos, operações, etc.) • Cada elemento com exemplos (código e execução) • Exemplos com fonte diferenciada (listing)	
Mínimo 5 Aplicações completas - Aplicações (Máximo : 2,0 pontos) • Uso de rotinas-funções-procedimentos, E/S formatadas • Uma Calculadora • Gráficos • Algoritmo QuickSort • Outra aplicação • Outras aplicações ...	
Ferramentas (compiladores, interpretadores, etc.) (Máximo : 1,0 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 com o uso das ferramentas • Descrição das ferramentas (autor, versão, homepage, tipo, etc.)	
Organização do trabalho (Máximo: 01 ponto) • Conteúdo, Historia, Seções, gráficos, exemplos, conclusões, bibliografia • Cada elemento com exemplos (código e execução, ferramenta, nome do aluno)	
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

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, Abril 2023

Contents

1. Introdução

Scala é uma linguagem de programação moderna e **multi-paradigma** desenvolvida para expressar padrões de programação comuns em uma forma concisa, elegante e com tipagem segura. Integra facilmente características de linguagens orientadas a objetos e funcional.

1.1 Aspectos históricos da linguagem Scala

Scala é uma linguagem de programação de uso geral. Foi criada e desenvolvida por Martin Odersky. Martin começou a trabalhar no Scala em 2001 na École Polytechnique Federale de Lausanne (EPFL). Foi lançado oficialmente em 20 de janeiro de 2004.

Scala não é uma extensão do Java, mas é totalmente interoperável com ele. Durante a compilação, o arquivo Scala é traduzido para bytecode Java e executado em JVM (Java Virtual machine).

> "Eu ainda queria combinar programação funcional e orientada a objetos, mas sem as restrições impostas pelo Java. Eu havia descoberto sobre o cálculo de junção e acreditava que essa seria uma excelente base para basear tal unificação. O resultado foi Funnel, uma linguagem de programação para redes funcionais. Este era um design lindamente simples, com poucos recursos de linguagem primitiva.(...)
>
> Contudo, descobriu-se que a linguagem não era muito agradável de usar na prática. O minimalismo é ótimo para designers de linguagem, mas não para usuários. Os usuários não especialistas não sabem como fazer as codificações necessárias, e os usuários experientes ficam entediados de ter que fazê-las repetidas vezes. Além disso, tornou-se rapidamente evidente que qualquer nova linguagem só terá chance de ser aceita se vier com um grande conjunto de bibliotecas padrão." [BV09]

Scala foi projetado para ser **orientado a objetos e funcional**. É uma linguagem pura orientada a objetos no sentido de que todo valor é um objeto e uma linguagem funcional no sentido de que toda função é um valor. O nome scala é derivado da palavra escalável, o que significa que pode crescer com a demanda dos usuários

1.2 Áreas de Aplicação da Linguagem

Neste Capítulo serão apresentadas as princípais áreas de aplicação da linguagem Scala.

1.2.1 Desenvolvimento Web

Figure 1.1: Web Dev, Autor: PNGWING

O desenvolvimento web com Scala envolve a criação de aplicativos web dinâmicos e escaláveis. Frameworks como Play Framework e Lift são amplamente utilizados para construir aplicativos web modernos. Scala oferece uma sintaxe concisa e expressiva, facilitando o desenvolvimento rápido e seguro de aplicativos web complexos.

1.2.2 Processamento de Big Data

Figure 1.2: Big Data, Autor: PNGWING

Big data refere-se ao processo de análise, processamento e interpretação de grandes conjuntos de dados. Scala é uma escolha popular para o processamento de big data, especialmente em projetos que envolvem tecnologias como Apache Spark, Apache Flink e Akka. Sua sintaxe concisa e expressiva é adequada para lidar com pipelines de processamento de dados complexos e distribuídos.

1.2.3 Machine Learning e Inteligência Artificial

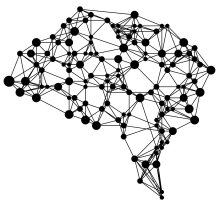


Figure 1.3: Rede Neural, Autor: PNGWING

Machine learning e inteligência artificial envolvem o desenvolvimento de algoritmos e modelos que podem aprender e tomar decisões com base em dados. Scala é frequentemente utilizada em projetos de machine learning e inteligência artificial devido à sua capacidade de lidar com algoritmos complexos e manipulação eficiente de grandes conjuntos de dados. Bibliotecas como Apache Mahout e Breeze oferecem suporte para desenvolvimento de modelos de machine learning em Scala.

1.2.4 Desenvolvimento de Aplicações Desktop e Mobile

Scala também pode ser utilizada no desenvolvimento de aplicativos desktop e mobile. Frameworks como JavaFX e ScalaFX permitem aos desenvolvedores criar interfaces gráficas de usuário utilizando a linguagem Scala. Sua integração com o ambiente de desenvolvimento Java oferece acesso a uma ampla gama de bibliotecas e ferramentas para criar aplicativos interativos e visualmente atraentes.

1.2.5 Finanças e Serviços Financeiros

Figure 1.4: Finance, Autor: PNGWING

Na área financeira, Scala é frequentemente utilizada para análise de dados, modelagem financeira e desenvolvimento de sistemas de negociação. Sua capacidade de lidar com cálculos complexos e manipulação eficiente de dados financeiros torna-a uma escolha popular entre empresas que trabalham com análise de dados financeiros e gerenciamento de riscos.

1.2.6 Educação e Pesquisa

Scala é uma escolha popular em ambientes acadêmicos e de pesquisa, onde é utilizada para ensinar conceitos avançados de programação e realizar pesquisas em áreas como linguagens de programação, algoritmos e sistemas distribuídos. Sua combinação única de programação orientada a objetos e funcional torna-a uma ferramenta poderosa para explorar conceitos de computação moderna e desenvolver soluções inovadoras para problemas computacionais complexos.

2. Conceitos básicos da Linguagem Scala

Os livros básicos para o estudo da Linguagem Scala são: [Wha20], [Wam21], [Hun18] e [Upa19]

Neste capítulo é apresentado .... Segundo [Hun18], a linguagem Scala, . . .

De acordo com [Seb18] e [RH04], a linguagem Scala . . . [Seb18] afirma que a linguagem Python . . .

Considerando que a linguagem Scala ([Wha20], [Upa19]) é considerada como ....

2.1 Variáveis e constantes

Scala é uma linguaguem com tipagem estatica, diferente de Python por exemplo em que sua tipagem é dinamica, isso significa que uma **variavel em Scala não pode ter seu tipo mudado**, geralmente em linguagens dinamicas o valor de uma variavel é processado durante o tempo de execução, já nas linguagens dinamicas o valor é processado no tempo de parse Em questão de performance, otimizar um codigo em linguaguem dinamica é muito mais desafiador do que em linguaguem estatica. [Wam21]

Existem duas declarações da variaveis em Scala Val que é constante e var que pode ser mudada: "val" Quando a variavel é imutavel(Read-Only)

```
Scala> val Numero: int = 1
```

"var" Quando a variavel é mutavel(Read and Write)

```
Scala> var Numero: int = 1
resultado: Numero = 1
Scala> Numero = 2
resultado: Numero = 2
```

Scala recomenda que você use as varaiveis do tipo **val sempre que possivel**, pois isso promove um melhor design orientado a objeto e é consistente com o conceito de programação funcional "pura". [Wam21]

2.2 Tipos de Dados Básicos

2.2.1 String

Código fonte para a linguagem Scala:

```
    object HelloWorld {
  def main(args: Array[String]): Unit = {
    println("Hello, world!")
  }
}
```

```
  class Rational(n: Int, d: Int) {

    require(d != 0)

    private val g = gcd(n.abs, d.abs)
    val numer = n / g
    val denom = d / g

    def this(n: Int) = this(n, 1)

    def + (that: Rational): Rational =
      new Rational(
        numer * that.denom + that.numer * denom,
        denom * that.denom
      )

    def + (i: Int): Rational =
      new Rational(numer + i * denom, denom)

    def - (that: Rational): Rational =
      new Rational(
        numer * that.denom - that.numer * denom,
        denom * that.denom
      )
```

2.2.2 Int

O tipo Int em Scala representa números inteiros de 32 bits. Ele é usado para armazenar valores inteiros, que podem variar de -2 elevado 31 a 2 elevado 31-1.

Em Scala, Int é um objeto, o que significa que pode ser usado com métodos e operações como qualquer outro objeto 1.

Exemplo:

```
val num: Int = 10
```

2.2.3 Float

O tipo Float é usado para representar números de ponto flutuante de precisão simples de 32 bits, conforme definido pelo padrão IEEE 754. Isso permite armazenar valores de ponto flutuante com uma precisão limitada. Em Scala, Float é um objeto, permitindo operações aritméticas e acesso a métodos.

```
val numFloat: Float = 3.14
```

2.2.4 Boolean

O tipo Boolean em Scala representa valores verdadeiro (true) ou falso (false). É usado para operações lógicas e condicionais. Assim como os outros tipos, Boolean é um objeto em Scala, permitindo operações lógicas e acesso a métodos

```
val isTrue: Boolean = true
```

2.2.5 Char

O tipo Char é usado para representar um único caractere Unicode de 16 bits. Isso permite armazenar qualquer caractere Unicode dentro de um único valor. Em Scala, Char é um objeto, permitindo operações com caracteres e acesso a métodos 1.

Exemplo:

```
val letra: Char = 'a'
```

2.2.6 Byte

O tipo Byte é usado para representar valores inteiros de 8 bits. Ele pode armazenar valores de -128 a 127. Em Scala, Byte é um objeto, permitindo operações aritméticas e acesso a métodos 1.

Exemplo:

```
val numByte: Byte = 127
```

2.2.7 Short

O tipo Short é usado para representar valores inteiros de 16 bits. Ele pode armazenar valores de -32768 a 32767. Em Scala, Short é um objeto, permitindo operações aritméticas e acesso a métodos 1.

Exemplo:

```
val numShort: Short = 32767
```

2.2.8 Long

O tipo Long é usado para representar valores inteiros de 64 bits. Ele pode armazenar valores de -$2^63a2^63-1.EmScala, Longumobjeto, permitindooperaesaritmticaseacessoamtodos1$.

Exemplo:

```
val numLong: Long = 9223372036854775807L
```

2.2.9 Double

O tipo Double é usado para representar números de ponto flutuante de precisão dupla de 64 bits, conforme definido pelo padrão IEEE 754. Isso permite armazenar valores de ponto flutuante com uma precisão maior do que o Float. Em Scala, Double é um objeto, permitindo operações aritméticas e acesso a métodos 1.

Exemplo:

```
val numDouble: Double = 3.141592653589793
```

2.2.10 Unit

O tipo Unit em Scala é usado para representar um valor que não carrega informação significativa. É o tipo de retorno padrão para funções que não retornam um valor. Em Scala, Unit é um objeto, permitindo operações e acesso a métodos 1.

Exemplo:

```
    def printMessage(): Unit = println("Hello, World!")
```

2.3 Operadores e Expressões em Scala

2.3.1 Operadores Aritmeticos

Os operadores aritméticos são utilizados para realizar operações matemáticas básicas.

- Adição (+): Soma dois valores.
- Subtração (-): Subtrai um valor de outro.
- Multiplicação (*): Multiplica dois valores.
- Divisão (/): Divide um valor por outro.
- Módulo (%): Retorna o resto da divisão entre dois valores.

```
val soma = 10 + 5 // soma     15
val subtracao = 20 - 7 // subtracao     13
val multiplicacao = 8 * 4 // multiplicacao     32
val divisao = 100 / 5 // divisao     20
val modulo = 15 % 4 // modulo     3
```

2.3.2 Operadores de Comparação

Os operadores de comparação são utilizados para comparar valores e produzir resultados booleanos (verdadeiro ou falso).

- Igualdade (==): Verifica se dois valores são iguais.
- Diferença (!=): Verifica se dois valores são diferentes.
- Maior que (>): Verifica se um valor é maior que outro.
- Menor que (<): Verifica se um valor é menor que outro.
- Maior ou igual (>=): Verifica se um valor é maior ou igual a outro.
- Menor ou igual (<=): Verifica se um valor é menor ou igual a outro.

```
val igualdade = 10 == 10 // igualdade     true
val diferenca = 20 != 15 // diferenca     true
val maiorQue = 30 > 25 // maiorQue     true
val menorQue = 40 < 35 // menorQue     false
val maiorOuIgual = 50 >= 50 // maiorOuIgual     true
val menorOuIgual = 60 <= 55 // menorOuIgual     false
```

2.3.3 Operadores Logicos

Os operadores lógicos são utilizados para combinar expressões booleanas.

- E lógico (&&): Retorna verdadeiro se ambas as expressões forem verdadeiras.
- Ou lógico (||): Retorna verdadeiro se pelo menos uma das expressões for verdadeira.
- Não lógico (!): Inverte o valor de uma expressão.

```
val expressao1 = true
val expressao2 = false
val eLogico = expressao1 && expressao2 // eLogico     false
```

```
val ouLogico = expressao1 || expressao2 // ouLogico     true
val naoLogico = !expressao1 // naoLogico     false
```

3. Programação em Scala

3.1 Entradas e saídas

As entradas e saídas de dados em Scala são fundamentais para a interação entre programas Scala e seus usuários ou sistemas externos. A principal e mais básica forma de entrar com dados e ter suas respostas é o console.

3.1.1 Entrada

```
    val input = scala.io.StdIn.readLine("Digite alguma coisa:")
```

assim que o programa for executado e chegar nessa linha, ele ira aguardar o usuario inserir a informação através do console, após isso o usuairo aperta enter e confirma o que foi digitado, que é enviado para um espaço de memoria.

3.1.2 Saida

```
    println("Hello, World!")
```

através dessa linha o que está entre colchetes é imprimido(mostrado) na janela do console.

3.1.3 E/S Exemplo

```
        println("Digite o primeiro n mero:")
    val numero1 = readInt()

    // Solicitar o segundo n mero ao usu rio
    println("Digite o segundo n mero:")
    val numero2 = readInt()

    // Calcular a soma dos dois n meros
    val soma = numero1 + numero2
```

```
    // Solicitar ao usu rio a resposta esperada
    println(s"A soma de {numero1} e {numero2}    {soma}.
    Digite a resposta correta:")

    // Ler a resposta do usu rio
    val respostaUsuario = readInt()
```

Nesse trecho de codigo podemos ver os dois comandos serem usados de forma simples, o computador imprimi as mensagens para o usuario digitar dois numeros, o computador lê e armazena esses numeros e em seguida o usuario digita a soma deles, apesar de que o computador processou a soma desses numeros na linha "val soma = numero1 + numero2" ele ainda não mostrou ao usuario se a soma que ele colocou está correta.

3.1.4 Entrada e Saída formatada

A Formatação de strings é essencial para apresentar informações de maneira clara e organizada, em Scala existem dois principais metodos format() e formatted()

O método format() permite formatar strings, aceitando parâmetros que substituem placeholders na string. Os placeholders são indicados por % seguido por um especificador de formato, como %d para inteiros e %f para pontos flutuantes ou doubles. exemplo:

```
        // Criando uma string formatada
    val x = "There are %d books and cost of each book is %f"

    // Atribuindo valores
    val y = 15
    val z = 345.25

    // Aplicando o m todo format
    val r = x.format(y, z)

    // Exibindo a string formatada
    println(r)
```

3.1.5 entrada e saida de dados em arquivos

Para ler um arquivo de texto, você pode usar o método source da classe Source do pacote scala.io.Source:

```
        import scala.io.Source
val fileContent = Source.fromFile("/path/to/file.txt").mkString
println(fileContent)
```

atráves desse codigo todo conteudo dentro do arquivo "file.txt" é mostrado no console de forma que é possivel ler seu conteudo.

Para escrever em um arquivo, você pode usar o método write da classe PrintWriter:

```
    import java.io.PrintWriter

val writer = new PrintWriter(new File("/path/to/file.txt"))
writer.write("Hello, Scala!")
writer.close()
```

Nesse codigo o arquivo "file.txt" é aberto pelo programa, e nele é escrito "Hello, Scala!" depois o arquivo é fechado.

3.2 Seleção e condicional

Em Scala, você pode usar a estrutura condicional if para executar diferentes blocos de código com base em uma condição booleana. Além disso, Scala oferece uma variedade de formas de expressar condições complexas e múltiplas usando if, else if e else, bem como o match, que é semelhante ao switch em outras linguagens de programação.

if, else if, else A estrutura básica do if em Scala é a seguinte:

```
    if (condicao) {
  // bloco de c digo executado se a condi   o for verdadeira
} else if (outraCondicao) {
  // bloco de c digo executado se a outra
  //condi   o for verdadeira
} else {
  // bloco de c digo executado se
  //nenhuma das condi   es anteriores for verdadeira
}
```

exemplo de condição:

```
    val x = 10

if (x > 0) {
  println("x    positivo")
} else if (x < 0) {
  println("x    negativo")
} else {
  println("x    zero")
}
```

Em Scala, você pode usar a estrutura condicional if para executar diferentes blocos de código com base em uma condição booleana. Além disso, Scala oferece uma variedade de formas de expressar condições complexas e múltiplas usando if, else if e else, bem como o match, que é semelhante ao switch em outras linguagens de programação.

if, else if, else A estrutura básica do if em Scala é a seguinte:

if (condicao) // bloco de código executado se a condição for verdadeira else if (outraCondicao) // bloco de código executado se a outra condição for verdadeira else // bloco de código executado se nenhuma das condições anteriores for verdadeira Exemplo:

val x = 10

if (x > 0) println("x é positivo") else if (x < 0) println("x é negativo") else println("x é zero") match O match em Scala é semelhante ao switch em outras linguagens de programação, mas é muito mais poderoso e flexível. Ele pode ser usado para comparar um valor com uma série de padrões e executar o bloco de código correspondente ao padrão que corresponde ao valor.

```
    val diaDaSemana = "segunda-feira"

val mensagem = diaDaSemana match {
  case "segunda-feira" | "ter a-feira"
  | "quarta-feira" | "quinta-feira" | "sexta-feira" =>
    "Dia de trabalho"
```

```
  case "s bado" | "domingo" =>
    "Fim de semana"
  case _ =>
    "Dia inv lido"
}

println(mensagem)
```

No exemplo acima, o match compara diaDaSemana com diferentes padrões e executa o bloco de código correspondente ao padrão que coincide. O _ é um padrão curinga que corresponde a qualquer valor.

3.3 Repetição

3.3.1 while/do-while

O while é uma estrutura de repetição que executa um bloco de código enquanto uma condição específica for verdadeira.

```
    var i = 0
while (i < 5) {
  println(s"Valor de i: $i")
  i += 1
}
```

O do-while é semelhante ao while, mas garante que o bloco de código seja executado pelo menos uma vez, mesmo que a condição seja falsa na primeira vez. var j = 0 do println(s"Valor de j: $j") j += 1 while (j < 5)

3.3.2 for/foreach

O for em Scala pode ser usado para percorrer uma sequência de valores, como uma faixa numérica ou uma coleção.

```
    for (i <- 0 until 5) {
  println(s"Valor de i: $i")
}
```

Este loop também imprimirá os valores de i de 0 a 4. A expressão 0 until 5 cria uma faixa de números de 0 a 4, e o loop itera sobre esses valores.

O foreach é usado para percorrer elementos em uma coleção, como listas, arrays ou sequências.

```
val lista = List("a", "b", "c", "d", "e")
lista.foreach { elemento =>
  println(s"Elemento: $elemento")
}
```

Este loop imprimirá cada elemento da lista, de "a" a "e". O método foreach é chamado em uma coleção e uma função é passada como argumento. Essa função é então aplicada a cada elemento da coleção.

3.4 Funções

Funções são estruturas de codigo que tem o objetivo de tornar o codigo mais encapsulado, elas servem, elas são amplamente utilizadas em trechos que podem se repetir muitas vezes em um codigo, ou em partes diferentes do programa. exemplo

```
    def somar(a: Int, b: Int): Int = {
  a + b
}
```

Neste exemplo, a função somar recebe dois parâmetros inteiros a e b e retorna a soma deles como um inteiro.

Chamando Funções Depois de definir uma função, você pode chamá-la em qualquer lugar do seu código, fornecendo os valores necessários para os parâmetros.

scala Copiar código

```
val resultado = somar(3, 5) // resultado    8
```

Neste exemplo, estamos chamando a função somar com os valores 3 e 5 e atribuindo o resultado a uma variável chamada resultado.

3.5 Módulos e Subprogramas

```
class Rational(n: Int, d: Int) {

    require(d != 0)

    val numer: Int = n
    val denom: Int = d

    def this(n: Int) = this(n, 1) // auxiliary constructor

    override def toString = numer +"/"+ denom

    def add(that: Rational): Rational =
      new Rational(
        numer * that.denom + that.numer * denom,
        denom * that.denom
      )
  }
```

4. Aplicações da Linguagem Scala

Devem ser mostradas pelo menos CINCO aplicações completas da linguagem, e em cada caso deve ser apresentado:

- Uma breve descrição da aplicação
- O código completo da aplicação,
- Imagens do código fonte no compilador,
- Imagens dos resultados após a compilação-interpretação do código fonte
- Links e referencias bibliográficas de onde foi obtido a aplicação

4.1 Operações básicas

Implementar um Programa INTERATIVO para calcular o VOLUME de um cilindro (menu interativo)

- **Descrição da aplicação**:
- **Código Scala completo da aplicação**:
- **Capturas de tela da aplicação rodando no compilador**:
- **Capturas de telas dos RESULTADOS da aplicação**:
- **Referências**: bibliografia, links da Internet, etc.

4.2 O algoritmo Quicksort en Scala

Este algoritmo esta disponível na internet: só copiar, adaptar, comentar o código e compilar

- **Descrição da aplicação**:
- **Código Scala completo da aplicação**:
- **Capturas de tela da aplicação rodando no compilador**:
- **Capturas de telas dos RESULTADOS da aplicação**:
- **Referências**: bibliografia, links da Internet, etc.

4.3 Programa de Cálculo Numérico

Pode ser a solução de um sistema de equações, o cálculo das raízes de uma função, interpolação, etc.

- **Descrição da aplicação**:
- **Código Scala completo da aplicação**:
- **Capturas de tela da aplicação rodando no compilador**:
- **Capturas de telas dos RESULTADOS da aplicação**:
- **Referências**: bibliografia, links da Internet, etc.

4.4 Aplicação usando Matrizes

- **Descrição da aplicação**:
- **Código Scala completo da aplicação**:
- **Capturas de tela da aplicação rodando no compilador**:
- **Capturas de telas dos RESULTADOS da aplicação**:
- **Referências**: bibliografia, links da Internet, etc.

4.5 Aplicações Profissionais

Aqui pode ser qualquer aplicação de outra área de conhecimento, por exemplo: Física, Mecãnica de Fluídos, Biologia, Astronomia, Jogos, Química, etc. Pesquisar na Internet, para aplicações prontas e pequenas.

- **Descrição da aplicação**:
- **Código Scala completo da aplicação**:
- **Capturas de tela da aplicação rodando no compilador**:
- **Capturas de telas dos RESULTADOS da aplicação**:
- **Referências**: bibliografia, links da Internet, etc.

5. Ferramentas existentes e utilizadas

Neste capítulo devem ser apresentadas pelo menos DUAS (e no máximo 5) ferramentas consultadas e utilizadas para realizar o trabalho, e usar nas aplicações. Considere em cada caso:

- Nome da ferramenta (compilador-interpretador)
- Endereço na Internet
- Versão atual e utilizada
- Descrição simples (máx 2 parágrafos)
- Telas capturadas da ferramenta
- Outras informações

5.1 Editores para Fortran

5.2 Compiladores

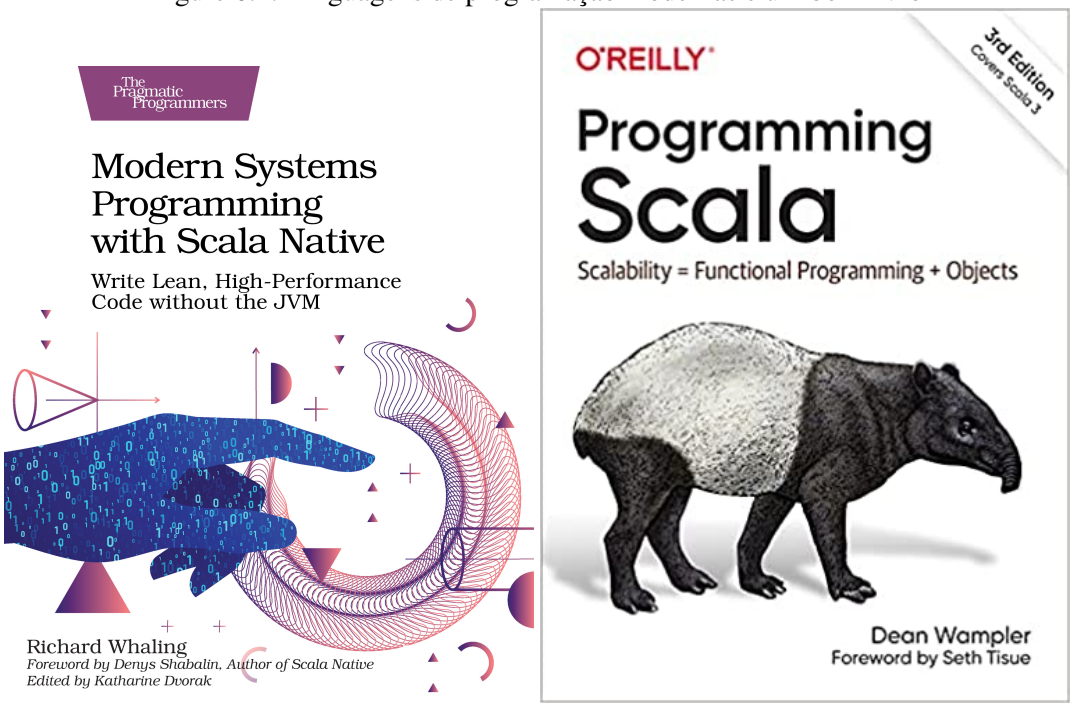
- Site principal : https://www.scala-lang.org/
- Scala 3 : https://www.scala-lang.org/download/
-
-
-

5.3 Ambientes de Programação IDE para Scala

6. Considerações Finais

Os problemas enfrentados neste trabalho ...
O trabalho que foi desenvolvido em forma resumida ...
Aspectos não considerados que poderiam ser estudados ou úteis para ...

Figure 6.1: Linguagens de programação modernas e um bom livro



Fonte: O autor

Bibliography

[BV09] Frank Sommers Bill Venners. The origins of scala a conversation with martin odersky. *artima*, 2009. Citado na página 7.

[Hun18] John Hunt. *A Beginner's Guide to Scala, Object Orientation and Functional Programming*. Springer-Verlag International, Cham, Switzerland, 2 edition, March 2018. Citado na página 11.

[RH04] Peter Van Roy and Seif Haridi. *Concepts, Techniques and Models of Computer Programming*. The MIT Press, Cambridge, 2004. Citado na página 11.

[Seb18] Robert W. Sebesta. *Conceitos de Linguagens de Programação*. Bookman, Porto Alegre, RS, 11 edition, 2018. Citado na página 11.

[Upa19] Bhim P. Upadhyaya. *Data Structures and Algorithms with Scala*. Springer-Verlag International, Cham, Switzerland, 1 edition, March 2019. Citado na página 11.

[Wam21] Dean Wampler. *Programming Scala*. O'Reilly Media Inc., Sebastopol, CA, 3 edition, July 2021. Citado na página 11.

[Wha20] Richard Whaling. *Modern Systems Programming with Scala Native*. O'Reilly Media Inc., Raleigh, NC, 1 edition, February 2020. Citado na página 11.