



Introdução à R

Paradigmas de Linguagens de Programação

**Frederico Rangel Sader
Ausberto S. Castro Vera**

16 de maio de 2023



Disciplina: *Paradigmas de Linguagens de Programação 2023*

Linguagem: *R*

Aluno: *Frederico Rangel Sader*

Ficha de avaliação:

Aspectos de avaliação (requisitos mínimos)	Pontos
Introdução (Máximo: 01 pontos) <ul style="list-style-type: none"> • Aspectos históricos • Áreas de Aplicação da linguagem 	
Elementos básicos da linguagem (Máximo: 01 pontos) <ul style="list-style-type: none"> • Sintaxe (variáveis, constantes, comandos, operações, etc.) • Cada elemento com exemplos (código e execução) 	
Aspectos Avançados da linguagem (Máximo: 2,0 pontos) <ul style="list-style-type: none"> • 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) <ul style="list-style-type: none"> • 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) <ul style="list-style-type: none"> • 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) <ul style="list-style-type: none"> • Conteúdo, Historia, Seções, gráficos, exemplos, conclusões, bibliografia • Cada elemento com exemplos (código e execução, ferramenta, nome do aluno) 	
Uso de Bibliografia (Máximo: 01 ponto) <ul style="list-style-type: none"> • 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

Copyright © 2023 Frederico Rangel Sader e Ausberto S. Castro Vera

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, Maio 2019



Sumário

1	Introdução	7
1.1	Aspectos históricos da linguagem R	7
1.2	Áreas de Aplicação da Linguagem	8
1.2.1	Data Science	8
1.2.2	Orientação a objetos	8
1.2.3	outras	9
2	Conceitos básicos da Linguagem R	11
2.1	Variáveis e constantes	11
2.2	Tipos de Dados Básicos	12
2.2.1	Dados numéricos	12
2.2.2	Dado de caracteres	12
2.2.3	Dados lógicos	13
2.3	Tipos de Dados de Coleção	13
2.3.1	Vetores	13
2.3.2	Listas	14
2.3.3	Matrizes	14
2.3.4	DataFrame	15
2.4	Operações Lógicas	16
2.5	Estrutura de Controle e Funções	16
2.5.1	Iterativas	16
2.5.2	Condicionais	17
2.6	Módulos	18
2.7	Orientação a Objetos	19
2.7.1	S3	19

2.7.2	S4	20
2.7.3	Reference Class	21
3	Aplicações da Linguagem R	23
3.1	Operações básicas	23
3.1.1	Função menu	24
3.1.2	Função calcular_volume	24
3.1.3	Função main	25
3.2	Programas gráficos	26
3.2.1	Dados para os gráficos	27
3.2.2	Criando os gráficos	27
3.3	Programas com Objetos	28
3.4	O algoritmo Quicksort - Implementação	29
3.4.1	Chamada da função	30
3.4.2	Quicksort	30
3.5	Aplicações com Banco de Dados	31
3.5.1	Conectando e criando o banco de dados	32
3.5.2	Inserindo dados e os exibindo	32
	Bibliografia	33



1. Introdução

Se tornando uma das principais referencias quando se trata de diversas necessidades, a linguagem R é uma das mais populares atualmente.Na introdução deste livro, apresentaremos primeiramente um pouco da jornada dessa linguagem, com seus aspectos históricos. Além de suas principais áreas de aplicação, uma breve explicação sobre tais áreas e alguns exemplos desses usos.

1.1 Aspectos históricos da linguagem R

Segundo [Lan17], a história da Linguagem R remonta aos anos 70, com a criação da linguagem S, que serviu de base para a implementações de novos recursos. A linguagem S, desenvolvida na AT&T Bell Labs, primariamente pelo John Chambers, inventada a partir da necessidade de um sistema interativo de Analise Estatística, serve de fundamento para o surgimento da R, que por sua vez, foi produzida por conta de uma motivação parecida com de sua "antecessora", a busca por um melhor ambiente de software para Analise Estatística.A linguagem R então, surge nos anos 90, no Departamento de Estatística da Universidade de Auckland pelas mãos dos Estatísticos Ross Ihaka e Robert Gentleman.

Acostumados com o S, percebendo a necessidade de um melhor ambiente de Software e observando que não haviam softwares compatíveis no mercado, decidiram criar sua própria linguagem, nascendo assim a R.

De acordo com [Cot13], o fato da jornada dessa linguagem ter seu relativo começo desde os anos 70 se torna importante pois ela foi se desenvolvendo por décadas, não começou a partir do nada. Ela veio de algo e se tornou o que é hoje. Uma de suas principais características sendo a forma como se desenvolveu, tendo uma natureza de maior liberdade para contribuições, com seus usuários sendo capazes de, caso necessário, criarem seus próprios pacotes e compartilharem, R continua crescendo a partir de quem a usa.[Cha19]

Figura 1.1: Logos da Linguagem R



Fonte: O autor deste trabalho

Seu nome, como mostrado na logo acima, provém das iniciais dos seus criadores (Ross Ihaka e Robert Gentleman) e de um jogo figurado com a Linguagem S.

1.2 Áreas de Aplicação da Linguagem

A linguagem R, como dito, surge com a filosofia de ser uma ferramenta para a Analise de Dados em geral. Sendo útil e aplicada em diversas áreas relacionadas com essa ciência [WG16]. Tendo por exemplo: a Data Science, Big Data, Analise de Dados, Machine Learning, Estatística Computacional, entre outros. Além desses, a Linguagem R, sendo uma linguagem multi-paradigma, também suporta e é utilizada na Programação Orientada a Objetos. Veremos a seguir, alguns desses exemplos.

1.2.1 Data Science

Segundo [Spi21] a Data Science, sendo uma das áreas da programação que mais cresce atualmente, abarca diversas necessidades e possibilidades. Tendo como objetivo criar, manipular e extrair o máximo possível de conhecimento a partir de dados para então gerar valor e melhorar processos, produtos e serviços, requer de seu profissional habilidades em diferentes campos, incluindo estatística, matemática, programação, visualização de dados entre outros.

De acordo com [Kab15], cresce cada vez mais a quantidade de dados e informações com as quais empresas devem lidar para terem seu desempenho e custo otimizados, e se torna necessária a utilização de outras tecnologias mais eficientes para tratar dessa área. R surge então como uma das principais opções para esses desafios, tendo sido desenvolvida justamente para análise de dados e visualização desses resultados.

Por ser um software open-source, conta com uma massiva comunidade de usuários e desenvolvedores, o que garante uma evolução constante da linguagem e de suas utilidades. Essa sendo uma de suas maiores vantagens, possui milhares de pacotes à disposição de qualquer um para qualquer finalidade. Alguns desses se tornando referências para objetivos específicos, nos baseando no [WG16] achamos, por exemplo: ggplot2 para a visualização de dados, data.table para a manipulação de dados, entre outros.

Embora um cientista de dados necessite de especialização em diversas áreas de conhecimento, desde matemática até a programação em si, é bem amparado pela linguagem R e sua comunidade que constantemente cresce e facilita seus processos.

1.2.2 Orientação a objetos

Um dos paradigmas da linguagem de programação mais atuais é a Programação Orientada a Objetos (POO), se baseando em objetos, que podem ser entendidos como instâncias de classes, contendo atributos e métodos relacionados. Esse paradigma busca encapsular objetos, permitindo

maior independência das suas operações. Com outros conceitos fundamentais, a POO busca tornar o código mais flexível e adaptável. [DD18]

Embora muitos algoritmos de análise de dados e estatística não sejam implementados utilizando a Programação Orientada a Objetos, o uso desse paradigma pode ser benéfico quanto à organização do código em módulos reutilizáveis, facilitando a manipulação dos dados. Sendo comum usar objetos para representar conjuntos de dados, e métodos para ser possível acessar e manipular esses dados, como adicionar ou remover colunas ou calcular estatísticas referentes a esses conjuntos.

Podemos encontrar então a utilização do POO em R no [Mai17]. A linguagem sendo uma das principais da área, além de multi-paradigma, também utiliza esse paradigma, da forma citada acima, utilizando objetos para encapsular dados, ou também na construção de modelos estatísticos, por exemplo.

R trabalha com três diferentes sistemas de Orientação a Objetos: S3, S4 e RC. S3 sendo o mais simples sistema de classes, trabalha com poucas restrições e não tendo uma estrutura definida formalmente, tendo como ideia que existam funções genéricas para serem utilizadas como os métodos, de acordo com a classe do objeto. S4 é uma evolução do anterior, tendo uma estrutura mais formal e definida, permitindo uma maior especificação dos seus métodos e atributos. Já o RC, ou Reference Class, já é uma abordagem avançada de Orientação a Objetos, permitindo a criação de classes utilizando todos seus fundamentos, como a herança, polimorfismo e encapsulamento.

1.2.3 outras

Apesar de amplamente utilizada na Data Science e na POO, a linguagem R embarca diversas outras áreas, como por exemplo a área de Estatística Computacional, possuindo um vasto número de pacotes referentes a esse assunto, é uma das principais opções para estatísticos. Com grande capacidade para a análise de dados e modelagem estatística, se torna um destaque pra quem busca novas ferramentas.

Outra área em que a linguagem R tem sido utilizada é a Machine Learning, R possui um ampla gama de pacotes voltados para a área do aprendizado de máquina, como regressão linear e logística, clusterização e outros também para análise preditiva. Com o foco na automação através de algoritmos, é um assunto bem amparado pelos pacotes desenvolvidos em R.



2. Conceitos básicos da Linguagem R

[Cot13] afirma que a linguagem R é, em seu cerne, imperativa, mas que também suporta programação orientada a objetos e programação funcional, ou seja, é uma linguagem multi-paradigmas. Sendo assim, essa "mistura" faz com que R tenha muitas similaridades com outras linguagens de programação. Podemos escrever códigos imperativos parecidos com os em C. Ou caso utilizemos as "Reference Class" vistas no capítulo passado, também somos capazes de construir algoritmos que se pareçam com os em Java. Buscando ser flexível nos métodos que utiliza para trabalhar com as principais áreas em que é inserida (Data science, Estatística, Big Data,etc.), R é utilizada por diversos tipos de desenvolvedores. Nesse capítulo aprenderemos a base dessa linguagem, o "esqueleto" que possibilita toda a movimentação diversificada do R. Aprenderemos como funcionam as suas variáveis, seus tipos de dados, suas operações e estruturas para começarmos a entender como se tornou uma linguagem tão abrangente.

2.1 Variáveis e constantes

Basicamente, uma variável é um espaço de armazenamento na memória do computador que pode conter um valor ou uma referência a um valor. Uma variável é identificada por um nome que é usado para acessar o seu valor ou referência. O valor contido nessa variável pode ser manipulado através de cálculos ou entradas do usuário.

Já uma constante, é um valor que não pode ser alterado durante a execução do programa. Um valor fixo atribuído a uma variável no momento da sua declaração.

Vamos aprender como as variáveis e constantes são trabalhadas em R. Em R, as variáveis podem ser declaradas utilizando dois tipos de sintaxe de atribuição, sendo esses: -> ou =. Segue exemplo:

Figura 2.1:

```
> x <- 5
> y = 4
> x
[1] 5
> y
[1] 4
> |
```

Fonte: O autor deste trabalho

Como dito acima, os valores atribuídos à variáveis podem ser manipulados através de cálculos ou entradas de usuários, segue exemplo abaixo de uma alteração do valor da mesma variável 'x' através de uma conta:

Figura 2.2:

```
> x = 5 + 5
> x
[1] 10
> |
```

Fonte: O autor deste trabalho

Demonstramos, de uma forma básica, como declarar variáveis em R, atribuindo valores à elas. Utilizamos, nesse caso, o tipo de dado numeral, mas veremos no próximo tópico que em R existem diversos tipos de dados disponíveis.

2.2 Tipos de Dados Básicos

A linguagem R tem uma variedade de tipos de dados, vamos lista-los e fazer uma breve descrição de cada um. São esses:

2.2.1 Dados numéricos

Demonstrados nos exemplos acima, os dados numéricos em R são utilizados para armazenar valores como inteiros, reais e complexos. Existem dois tipos principais de números: inteiros e de ponto flutuante. Os números inteiros são números sem frações, enquanto os números de ponto flutuante têm uma parte fracionária. O R também suporta números que possuam uma parte real e imaginária.

Porém, esse tipo de dado, possui algumas limitações, como por exemplo, como visto em [Cot13], limitação na precisão, esse tipo de dado pode sofrer com cálculos que possuem números grandes demais ou pequenos demais. Outro exemplo de limitação seria o arredondamento, algumas operações matemáticas podem gerar resultados arredondados, o que pode levar a erros em cálculos que dependem de alta precisão.

2.2.2 Dado de caracteres

Outro tipo de dado existente em R é o de texto, ou caracteres, é utilizado para armazenar texto, palavras e frases, além de códigos alfanuméricos. Uma string (basicamente, uma sequência de caracteres que podem ser letras, números, símbolos ou até espaços em branco) é criada ao colocar o texto entre aspas simples(“) ou duplas (”).

As strings em R são tratadas como vetores de caracteres, permitindo operações como a seleção de um ou mais caracteres específicos, a seleção de subconjuntos de caracteres e a combinação de diferentes strings.

Entretanto, esse tipo de dado também possui algumas limitações, como por exemplo, o tamanho máximo de uma string, apesar de poder variar com a versão do R e do sistema operacional utilizado, ainda pode ser um problema em casos de manipulação de textos muito longos, como em análises de textos de grande extensão. Além dessa, pode ser citado o tratamento de caracteres especiais, algumas linguagens, incluindo o R, podem ter dificuldades em manipular caracteres como acentos e outros símbolos, podendo afetar a utilização e processamento de textos em línguas que utilizam esses caracteres.

2.2.3 Dados lógicos

Os dados lógicos, também conhecidos como booleanos, possibilitam a representação de valores verdadeiros ou falsos. Em R, o valor verdadeiro é representado pela palavra chave 'TRUE' e o valor falso, pela palavra chave 'FALSE'.

Esses valores lógicos são frequentemente utilizados em operações de condição e controle de fluxo, como nos comandos 'if', 'else', 'while' e 'for'. Essas operações podem funcionar se baseando no valor lógico dessas variáveis.

Apesar de simples, os valores lógicos são fundamentais na programação para controlar o fluxo da execução do programa.

Todavia, também possui limitações, sendo elas o fato de os valores lógicos serem limitados a dois estados (verdadeiro ou falso), o que significa que eles não podem representar nuances ou escalas de valores. Além disso, operações lógicas podem ser um pouco mais complexas e difíceis de entender, especialmente quando há várias condições envolvidas.

2.3 Tipos de Dados de Coleção

De acordo com [Lau08], os tipos de dados podem ser divididos dois grupos: atômicos e complexos, atômicos sendo o que nós estudamos acima, aqueles cujos elementos do conjunto de valores são indivisíveis, como são os inteiros, caracteres e lógicos. Agora estudaremos os complexos (ou compostos, ou de coleção) que, por sua vez, são aqueles cujos elementos do conjunto de valores podem ser decompostos em partes mais simples. Entenderemos melhor a partir de uma explicação mais detalhada de cada um existente em R.

2.3.1 Vetores

Os vetores são as coleções mais simples do R, e podem ser criados com diferentes classes de dados, como numéricos, inteiros, caracteres, fatores e lógicos. Eles podem ser considerados como uma sequência unidimensional de elementos de um mesmo tipo, e podem ser acessados e manipulados por meio de índices. São úteis em casos como se você deseja armazenar os dados totais de 50 clientes, em vez de criar 50 variáveis diferentes para cada um, basta criar um vetor de comprimento 50, que vai armazenar assim todos os dados do clientes. Vetores são estruturas de dados que podem ser declaradas com a função 'c()'. Como no exemplo abaixo:

Figura 2.3:

```
> a <- c(1,3,6,7,8)
> a
[1] 1 3 6 7 8
> |
```

Fonte: O autor deste trabalho

Como limitação, existe o fato de que os vetores não funcionam com tipos diferentes de dados. Um vetor não pode ter números e caracteres dentro dele, por exemplo.

2.3.2 Listas

As listas, por sua vez, são coleções que podem armazenar elementos de diferentes tipos e dimensões. Cada elemento da lista pode ser um vetor, uma matriz, uma outra lista, ou até mesmo um objeto R qualquer. As listas podem ser manipuladas de diversas formas, como adicionar ou remover elementos, modificar elementos existentes, ou até mesmo acessar elementos específicos utilizando índices.

As listas podem ser declaradas com a função 'list()'. Abaixo um exemplo de uma lista sendo criada com três tipos diferentes de dados (uma lista, um vetor e um numeral.):

Figura 2.4:

```
> lista1 <- list(1,2,3)
> lista1
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

> a <- c(4,5,6)
> a
[1] 4 5 6
> lista2 <- list(lista1, a, 7)
> lista2
[[1]]
[[1]][[1]]
[1] 1

[[1]][[2]]
[1] 2

[[1]][[3]]
[1] 3

[[2]]
[1] 4 5 6

[[3]]
[1] 7

>
```

Fonte: O autor deste trabalho

Embora as listas sejam flexíveis e úteis para muitas aplicações, é importante levar em consideração algumas limitações, como por exemplo, a manipulação de uma lista pode ter seu índice um pouco confuso, já que seus elementos podem ser de diferentes tipos e tamanhos.

2.3.3 Matrizes

Assim como os vetores, as matrizes são coleções de elementos do mesmo tipo, entretanto, são bidimensionais e organizadas em linhas e colunas. Podem ser criadas a partir de vetores, ou diretamente utilizando a função 'matrix()'. A sintaxe da função é, basicamente, 'matrix(dados,

`nrow = r, ncol = c, byrow = FALSE'` dentro dela, podemos definir o número de linhas (`nrow`), colunas (`ncol`) e se os dados serão inseridos por colunas (`byrow = FALSE`) ou linhas (`byrow = TRUE`). Segue exemplo:

Figura 2.5:

```
> mat_2E <- matrix(data = 1:18, nrow = 6, ncol = 3, byrow=FALSE)
>
> mat_2E
 [,1] [,2] [,3]
[1,]    1    7   13
[2,]    2    8   14
[3,]    3    9   15
[4,]    4   10   16
[5,]    5   11   17
[6,]    6   12   18
> |
```

Fonte: O autor deste trabalho

Compartilhando da limitação dos vetores, as matrizes não podem receber mais de um tipo de dados em sua estrutura, além de serem fixas em tamanho, ou seja, o número de linhas e colunas deve ser especificado no momento da criação da matriz. Se você precisar adicionar ou remover linhas ou colunas, pode ser necessário criar uma nova matriz ou transformá-la em outro tipo de estrutura de dados.

2.3.4 DataFrame

Por fim, temos os dataframes, sendo uma forma mais generalizada das matrizes, contém dados de forma tabular e cada coluna de sua estrutura pode conter tipos diferentes de dados. Em resumo, enquanto as matrizes em R são estruturas de dados bidimensionais com elementos do mesmo tipo, os data frames são mais flexíveis e permitem diferentes tipos de dados em cada coluna. Por isso, os data frames são frequentemente usados em análise de dados e modelagem estatística em R. Além da fácil visualização, manipulação e análise por conta de seu modelo tabular. Exemplo:

Figura 2.6:

```
> dataframe1 <- data.frame(
+   Pessoa = c("Aldo", "Aline", "Alan"),
+   Idade = c(26, 26, 27),
+   Peso = c(81, 85, 90),
+   Altura = c(6.5, 8.6, 2),
+   Salario = c(50000, 80000, 100000)
+ )
> dataframe1
Pessoa Idade Peso Altura Salario
1 Aldo     26   81     6.0  5e+04
2 Aline    26   85     5.8  8e+04
3 Alan     27   90     6.2  1e+05
>
```

Fonte: O autor deste trabalho

Apesar de amplamente utilizado por conta de sua flexibilidade e utilidade, o dataframe possui limitações, como o fato de não serem adequados para representar dados com mais de duas dimensões ou por ter a necessidade de que todos os elementos de um dataframe tenham o mesmo comprimento.

2.4 Operações Lógicas

Operações lógicas são operações matemáticas que retornam valores lógicos, isto é, verdadeiro (TRUE) ou falso (FALSE). Em R, essas operações podem ser listadas em:

- Operador de igualdade: '=='
- Operador de desigualdade: '!='
- Operador de maior que: '>'
- Operador de menor que: '<'
- Operador de maior ou igual que: '>='
- Operador de menor ou igual que: '<='
- Operador lógico AND: '&' ou '&&'
- Operador lógico OR: 'l' ou '||'
- Operador lógico NOT: '!'

Esses operadores retornam valores lógicos (TRUE ou FALSE) com base nas condições que são avaliadas. Por exemplo, se você quiser verificar se um número x é maior que 5, você pode usar o operador de maior que (>):

Figura 2.7:

```
> x <- 10
> x > 5
[1] TRUE
>
```

Fonte: O autor deste trabalho

Operações lógicas podem também possuir mais de um operador, por exemplo, é possível usar operadores lógicos para combinar múltiplas condições. Por exemplo, se você quiser verificar se um número x está entre 5 e 10, você pode usar o operador lógico AND (&):

Figura 2.8:

```
> x <- 7
> x > 5 & x < 10
[1] TRUE
> |
```

Fonte: O autor deste trabalho

2.5 Estrutura de Controle e Funções

De acordo com [Kab15], as declarações de um programa em R são executadas sequencialmente do topo do código até o fim. Porém, em alguns casos, é necessário que alguma dessas declarações seja executada repetidas vezes, enquanto em outros casos, que só seja executada caso cumpra algumas condições. e é onde entram as estruturas de controle.

Essas estruturas podem ser divididas em dois tipos: iterativas e condicionais. Veremos das duas:

2.5.1 Iterativas

As estruturas iterativas, como o nome sugere, são usadas para iterar sobre um conjunto de dados. As principais estruturas iteradoras em R são: "for" e "while".

For

A estrutura 'for' executa uma declaração repetidamente até que o valor de uma variável de controle alcance um valor já estabelecido nos parâmetros da função. Sua sintaxe é 'for (var in seq)' var sendo a variável de controle e seq o valor final que ela alcança. Nesse exemplo:

```
Figura 2.9:
```

```
> for (i in 1:10) print("olá")
[1] "olá"
>
```

Fonte: O autor deste trabalho

'Olá' foi imprimido 10 vezes.

While

A estrutura 'while' executa uma declaração repetidamente enquanto a condição dentro da estrutura for verdadeira. A sintaxe é 'while (cond) declaração'. Nesse exemplo:

```
Figura 2.10:
```

```
> i <- 10
> while (i > 0) {print("olá"); i <- i - 1}
[1] "olá"
>
```

Fonte: O autor deste trabalho

'Olá' foi novamente imprimido 10 vezes. Nessa estrutura é importante observar que é necessário uma modificação da variável de controle, dentro do bloco da declaração, que faça a condição ser alcançada, caso contrário, o while entrará em um loop infinito.

2.5.2 Condicionais

Nas estruturas condicionais, uma declaração só sera executada se uma condição especificada for alcançada. Essas são a 'if-else' e 'switch'.

If-else

A estrutura if-else executa uma declaração se uma condição específica for verdadeira. Opcionalmente, uma outra declaração pode ser executada, caso a condição seja falsa. A sintaxe é 'if (cond) declaração' ou 'if (cond) declaração1 else declaração2'. Nesse exemplo:

Figura 2.11:

```
> x <- 10
> if (x < 20) print("menor que 20") else print("maior ou igual a 20")
[1] "menor que 20"
>
```

Fonte: O autor deste trabalho

A primeira declaração foi executada, visto que a condição retornou verdadeira (x era menor que 20).

Switch

A estrutura switch em R é usada para realizar a seleção entre várias alternativas com base em uma expressão de teste. O switch é frequentemente usado em R para substituir uma longa sequência de if-else quando se trata de testar uma única variável em várias condições. A sintaxe é 'switch(expressão, caso1, caso2, ..., casoN)'. No exemplo abaixo:

Figura 2.12:

```
> x <- 2
> mensagem <- switch(x,
+                       "olá, mundo!",
+                       "Bem-vindo ao R!",
+                       "Adeus, R!")
> print(mensagem)
[1] "Bem-vindo ao R!"
>
>
```

Fonte: O autor deste trabalho

a variável 'mensagem' recebeu e imprimiu a segunda frase, pois x = 2.

Por fim, além das estruturas de controle, R possui como grande vantagem o fato do usuário ser capaz de adicionar funções. As funções em R são blocos de código que executam uma tarefa específica e retornam um valor. Elas permitem que você reutilize o mesmo código várias vezes em seu programa, tornando o seu código mais modular e mais fácil de manter e modificar. A estrutura de uma função é essa 'minhaFuncao <- function(arg1, arg2, ...)declaracoes return(object)' seguido da declaração 'function' estão os argumentos da função entre os parênteses e o corpo da função entre as chaves. Um exemplo simples:

Figura 2.13:

```
> soma <- function(a, b) {
+   resultado <- a + b
+   return(resultado)
+ }
>
> soma (4,5)
[1] 9
> |
```

Fonte: O autor deste trabalho

Nesse exemplo, os números passados como argumentos são o 4 e o 5, foram somados dentro da função e o resultado foi retornado. Apesar da função do exemplo ter sido simples, as funções podem ter todos os tipos de estruturas estudados (inclusive outras funções ou até mesmo a própria função executada), é um bloco de código que pode ser chamado a qualquer momento com diversos tipos de argumentos.

2.6 Módulos

Como dito acima, uma função pode ter em si todos os tipos de estruturas, o que facilita a modularização e a manutenção do programa, dessa forma, possibilitando a criação de módulos na linguagem

R. Módulos podem ser definidos como conjuntos de funções e objetos relacionados. Um módulo é um conjunto de funcionalidades que podem ser agrupadas e organizadas em um pacote para facilitar a reutilização e a manutenção do código. Assim como as funções facilitam a manipulação do código em um ponto de vista mais fechado, os módulos fazem de uma forma mais ampla com o programa, o modularizando e permitindo a reutilização, caso necessário. Além disso, cada módulo pode ter sua própria documentação, permitindo que os usuários entendam facilmente como as funções e objetos do módulo são usados e quais são suas finalidades específicas.

Como visto no primeiro capítulo, um dos destaques da linguagem R é a sua ampla oferta de pacotes que fornecem módulos para tarefas específicas. Pode se dizer então que os pacotes estão para os módulos, como os módulos estão para as funções, do ponto de vista da modularização do programa. Em resumo, as funções são a menor unidade de funcionalidade, os módulos são um conjunto de funções e objetos relacionados e os pacotes são um conjunto de módulos, dados e documentação relacionados. A modularização do código em R é uma prática recomendada para garantir a reutilização e a manutenção do código.

Para finalizar, um exemplo de pacote poderia ser o "ggplot2" que, por sua vez, contém vários módulos, como o "scale", "theme" e "geom", cada um com funções e objetos específicos para diferentes aspectos da criação de gráficos.

2.7 Orientação a Objetos

A orientação a objetos é uma abordagem de programação que enfatiza a criação de objetos que têm atributos (variáveis) e métodos (funções) que podem ser acessados e manipulados por outros objetos ou funções. Como visto no começo desse capítulo, R não é uma linguagem puramente orientada a objetos, mas tem características desse paradigma, em [Cot13], aprendemos que, em R, todas as funções, em si, são objetos de primeira classe, e de fato, de acordo com o próprio criador da linguagem R, John Chambers, "Tudo o que existe em R é um objeto". Em algumas circunstâncias, é útil aproveitar do estilo de POO presente em R, e pra isso é necessário entender como funcionam os três diferentes sistemas de Orientação a Objetos existentes em R.

2.7.1 S3

Em R, a classe S3 não possui uma definição pré-definida e não segue uma estrutura rígida de programação orientada a objetos como em outras linguagens como Java, C++ e C#. Em vez disso, a implementação de S3 é baseada na passagem de mensagens genéricas para métodos específicos, o que torna a implementação mais flexível e fácil de usar. Dessa forma, a função genérica age como um intermediário que despacha a chamada para o método específico correspondente à classe do objeto em questão.

Para criar um objeto S3, basta atribuir a ele uma classe definida pelo usuário usando a função `class()`. Por exemplo:

Figura 2.14:

```

> meuobjeto <- list(valor1 = 1, valor2 = 2)
> class(meuobjeto) <- "meuobjeto"
>
> meuobjeto
$valor1
[1] 1

$valor2
[1] 2

attr(,"class")
[1] "meuobjeto"
> |

```

Fonte: O autor deste trabalho

2.7.2 S4

O S4 é um sistema de orientação a objetos mais formalizado em R do que o S3, com definições mais precisas e estritas para classes e métodos. Diferente do S3, as classes S4 são definidas explicitamente, com o uso da função `setClass()`, que define os slots (atributos) e métodos associados a uma determinada classe. Além disso, para adicionar um método a uma classe S4 em R, é preciso usar a função `setGeneric()` para criar uma função genérica (como uma interface) que irá definir o nome e a assinatura da função. Em seguida, é necessário usar a função `setMethod()` para criar a implementação dessa função genérica para a classe específica.

Um exemplo de um objeto em S4 em R seria a definição de uma classe "Pessoa", com atributos como "nome" e "idade", e métodos como "aniversario" para incrementar a idade da pessoa em 1 ano:

Figura 2.15:

```

1 # definindo a função genérica "aniversario"
2 setGeneric("aniversario", function(objeto) {
3   standardGeneric("aniversario")
4 })
5 setClass("Pessoa",
6   slots = c(nome = "character", idade = "numeric"),
7   prototype = list(nome = NA_character_, idade = NA_real_))
8
9 setMethod("aniversario", "Pessoa",
10   function(objeto) {
11     objeto@idade <- objeto@idade + 1
12     objeto
13   })
14 # criando um objeto da classe "Pessoa"
15 joao <- new("Pessoa", nome = "João", idade = 25)
16
17 # chamando o método "aniversario" para aumentar a idade do objeto em um ano
18 joao <- aniversario(joao)
19
20 # imprimindo a nova idade do objeto
21 cat("Nova idade:", joao@idade)
22 |

```

Fonte: O autor deste trabalho

O resultado da última linha será "Nova idade: 26".

2.7.3 Reference Class

Reference Class já é um sistema de orientação a objetos mais próximos dos robustos das linguagens realmente voltadas a esse paradigma. No RC, os objetos são criados como instâncias de uma classe específica, e essa classe pode ter propriedades (ou atributos) e métodos, que podem ser acessados e modificados de forma semelhante à essas linguagens. Basicamente, tem todas as propriedades do S4, mas com mais estruturas e restrições. Exemplo:

Figura 2.16:

```
1 Pessoa <- setRefClass("Pessoa",
2   fields = list(
3     nome_pessoa = "character",
4     idade_pessoa = "numeric"
5   ),
6   methods=list(
7     show = function() {
8       (idade_pessoa <-> idade_pessoa + 1)
9       cat("Feliz aniversário, ", nome_pessoa, "! Agora você tem ", idade_pessoa, " anos.\n")
10    }
11  ))
12
13 pessoal <- Pessoa(nome_pessoa ="João",idade_pessoa = 31)
14 show(pessoal)
15
16
```

Fonte: O autor deste trabalho

O resultado da última linha sendo: "Feliz aniversário, João ! Agora você tem 32 anos."



3. Aplicações da Linguagem R

Neste capítulo vamos, a partir do que aprendemos nos capítulos anteriores, colocar à teste a Linguagem R quanto as suas aplicações.

Veremos, na prática, como os conceitos básicos conversam com as áreas de aplicação, desde operações básicas até aplicações com banco de dados. Praticaremos com exemplos, códigos completos com resultados e principalmente, a descrição desses programas, como funcionam e suas características. Vamos à prática.

3.1 Operações básicas

Em [Lan17] vemos que na programação é melhor diminuir a redundância sempre que possível, e as funções são uma ótima maneira de fazer isso.

Nessa seção vamos analisar um código básico para calcular o volume de um cilindro de uma forma totalmente interativa. Utilizaremos as ferramentas detalhadas no capítulo anterior.

Figura 3.1:

```

1+ menu <- function() {      # Função para o menu;
2   cat("===== Calculadora de volume de cilindro =====\n") # Imprimir o menu na tela;
3   cat("Escolha uma opção:\n")
4   cat("1. Calcular volume\n")
5   cat("2. Sair\n")
6 }
7
8+ calcular_volume <- function() {          # Função para o cálculo do volume;
9   cat("===== Cálculo do volume =====\n")
10
11  cat("Digite o raio do cilindro: ")
12  raio <- as.numeric(readline())           # Recebe o raio como string e converte para numérico;
13
14  cat("Digite a altura do cilindro: ")
15  altura <- as.numeric(readline())
16
17  volume <- pi * raio^2 * altura
18
19  cat("O volume do cilindro é: ", volume, "\n")
20 }
21
22+ main <- function(){        # Implementação da função principal (main);
23  escolha <- 0
24
25+ while (escolha != 2) {     # Loop para o menu;
26  menu()                   # Chamada da função menu;
27  escolha <- as.numeric(readline())
28
29+ if (escolha == 1) {       # Condicional interativa
30  calcular_volume()
31+ } else if (escolha == 2) {
32  cat("Encerrando o programa...\n")
33+ } else {
34  cat("Opção inválida. Tente novamente.\n")
35+ }
36
37  cat("\n")
38+ }
39+ }
40
41 # Iniciar o programa
42 main()

```

Fonte: O autor deste trabalho

Aprenderemos na prática a utilização das funções. Assim como na aplicação, a descrição será dividida pelas funções.

3.1.1 Função menu

No começo do código, declaramos e implementamos a função para o menu interativo, sendo o procedimento mais simples do programa, consiste apenas na impressão das quatro linhas de menu na tela, dando ênfase às duas opções dadas ao usuário.

Essa impressão funciona com a função nativa 'cat()' que serve para imprimir mensagens concatenadas. Também poderia ter sido usado a função 'print()', nesse caso.

3.1.2 Função calcular_volume

Após o título do cálculo de volume, impresso por 'cat()', é pedido ao usuário o raio do cilindro. O R possui algumas formas de entrada de dados, como 'scan()' que armazena os valores em vetores de acordo com o tipo de dado detectado. Nesse caso utilizaremos a função 'readline()' que recebe qualquer tipo de valor de entrada como uma string. Pode-se notar que a variável 'raio' não recebe diretamente o dado da entrada recebido por 'readline()', como se fosse "raio <- readline()". Esse fato se dá por conta do que foi dito acima, essa função de entrada recebe qualquer coisa como string, sendo assim, necessitaremos de uma outra função pra nos auxiliar nessa conversão de string para numérico. Intuitivamente, utilizaremos a função 'as.numeric()' passando como argumento o que for recebido na função 'readline()'. Sendo assim, a variável 'raio' receberá o que for retornado pela função 'as.numeric()', que por sua vez, recebe o que for retornado pela função 'readline()'.

É feita a exata mesma operação para pedir e receber o valor da altura do cilindro.

Depois de recebidas todas as informações necessárias, é declarada a variável 'volume' que recebe o resultado da conta "pi*raio²*altura".

Por fim, a função imprime o valor desse volume. Uma outra opção seria retornar o valor para fora da função e então imprimi-la, mas foi escolhido a impressão dentro da função.

3.1.3 Função main

Em R, diferentemente de outras linguagens, não há a necessidade de uma função main para iniciar um programa. As funções funcionam independentemente. Por conta de organização e estruturação, chamaremos de main a função onde acontecerá de fato o programa como uma unidade.

Para começar, declaramos a variável 'escolha' como zero, essa sendo a definição do usuário no menu. Em sequência, entraremos no laço while que tem como condição a variável 'escolha' ser diferente de 2, veremos como funciona esse laço. Assim que entramos na repetição, é chamada a função 'menu()' que imprime o menu indicando o que o usuário deve fazer, entretanto, a resposta não é recebida dentro dessa função, ela unicamente imprime na tela, sendo assim, temos, em seguida, o recebimento da opção do usuário, que é guardada na variável 'escolha'. Esse recebimento funciona com base na função 'as.numeric()' vista anteriormente.

Após isso, entramos em uma condicional if-else (ainda dentro do while) que tem como condição o valor da variável 'escolha' ser 1, 2 ou outra, cada uma dessas opções tem linhas a se seguir.

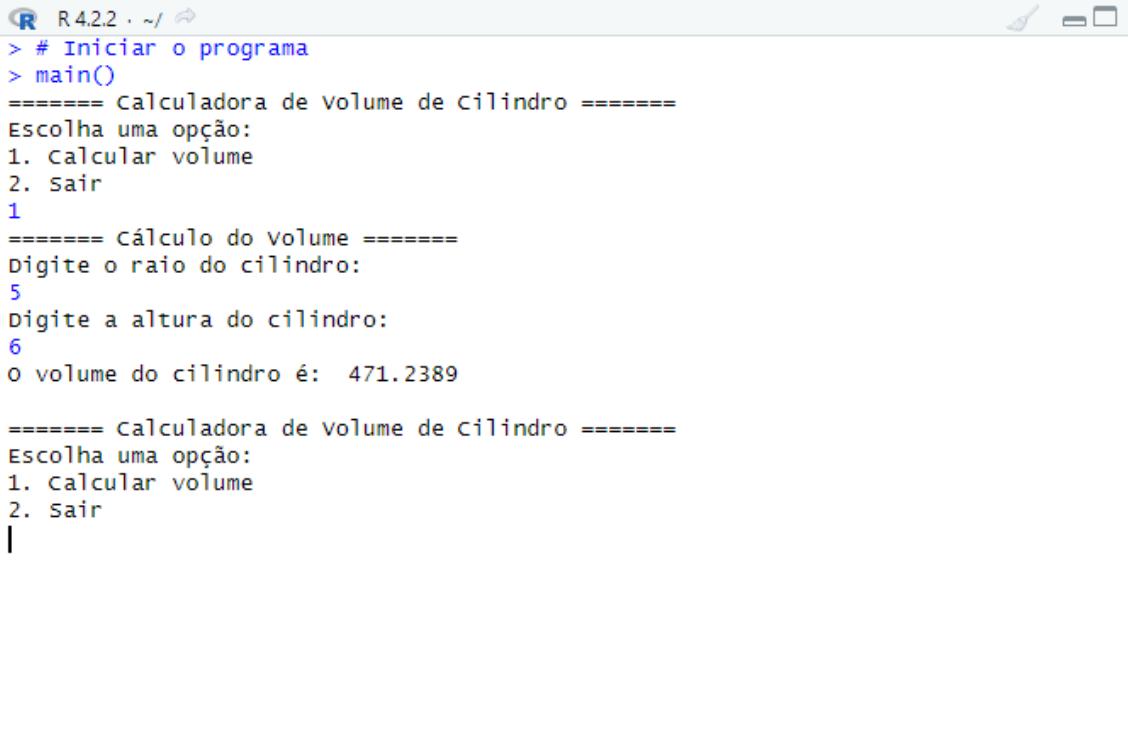
A primeira, significando que o usuário escolheu calcular o volume do cilindro, chamaria a função 'calcula_volume' que, como dito, imprime o resultado dentro dela própria, sem a necessidade de retornar o valor calculado em outra variável para ser impressa.

A segunda opção, significando que o usuário escolheu sair do programa, mostra na tela uma mensagem encerrando o programa.

Por fim, a terceira opção, caso o usuário digite algum valor que não sejam os dois válidos, é impressa uma mensagem avisando que a opção é inválida.

Como dito, em R as funções funcionam independentemente, então por mais que em outras linguagens, a implementação da função main já seja a execução dela, nesse caso, o main foi apenas o nome dado, não existe de fato uma função principal. Sendo assim, é necessário que no fim do programa, essa função "principal" seja chamada e passe a executar. Veremos abaixo a execução e o resultado desse programa.

Figura 3.2:



```
R 4.2.2 : ~/ 
> # Iniciar o programa
> main()
===== calculadora de volume de cilindro =====
Escolha uma opção:
1. Calcular volume
2. Sair
1
===== Cálculo do Volume =====
Digite o raio do cilindro:
5
Digite a altura do cilindro:
6
o volume do cilindro é: 471.2389

===== calculadora de volume de cilindro =====
Escolha uma opção:
1. Calcular volume
2. Sair
|
```

Fonte: O autor deste trabalho

Essa foi uma aplicação básica de um programa para calcular o volume de um cilindro de forma interativa. Vimos sobre funções em geral e laços básicos. Continuaremos vendo outras aplicações dessa linguagem.

3.2 Programas gráficos

Como dito anteriormente, a linguagem R é bastante utilizada para representar problemas e programas gráficos, e para fazer isso, recebe o auxílio de diversos pacotes que disponibilizam variadas funções com esse objetivo. Nessa seção utilizaremos o pacote, já citado, 'ggplot2'. Tudo referido à esse pacote teve como base o [WG16] e o index oficial do pacote <https://ggplot2.tidyverse.org/reference/index.html>.

Analisaremos um programa simples que recebe três equações e gera um gráfico pra cada uma delas.

Figura 3.3:

```

1 library(tidyverse)
2 # Dados para os gráficos
3 k <- seq(-2 * pi, 2 * pi, by = 0.1)
4 y1 <- 2 * x^2 + 5 * x - 3
5 y2 <- x^3 + 3 * x + 1
6 y3 <- (x - 2) * (x + 2) * (x - 5)
7
8 # Criar os gráficos usando ggplot2
9 p1 <- ggplot() +
10   geom_line(aes(x = x, y = y1), color = "red") +
11   labs(title = "y = 2*x^2 + 5*x - 3")
12
13 p2 <- ggplot() +
14   geom_line(aes(x = x, y = y2)) +
15   labs(title = "y = x^3 + 3*x + 1")
16
17 p3 <- ggplot() +
18   geom_line(aes(x = x, y = y3), color = "green") +
19   labs(title = "y = (x - 2) * (x + 2) * (x - 5)")
20 p1
21 p2
22 p3
23

```

Fonte: O autor deste trabalho

3.2.1 Dados para os gráficos

Primeiramente, precisamos carregar a biblioteca que usaremos. Nesse caso, como visto no [WG16], utilizaremos a 'tidyverse' que nada mais é do que uma coleção de pacotes em R. Nesse programa, será usado o 'ggplot2', que faz parte dessa coleção.

As próximas quatro linhas são apenas os valores das variáveis que serão utilizadas nos gráficos. Cada Y recebendo uma equação diferente em função de X.

3.2.2 Criando os gráficos

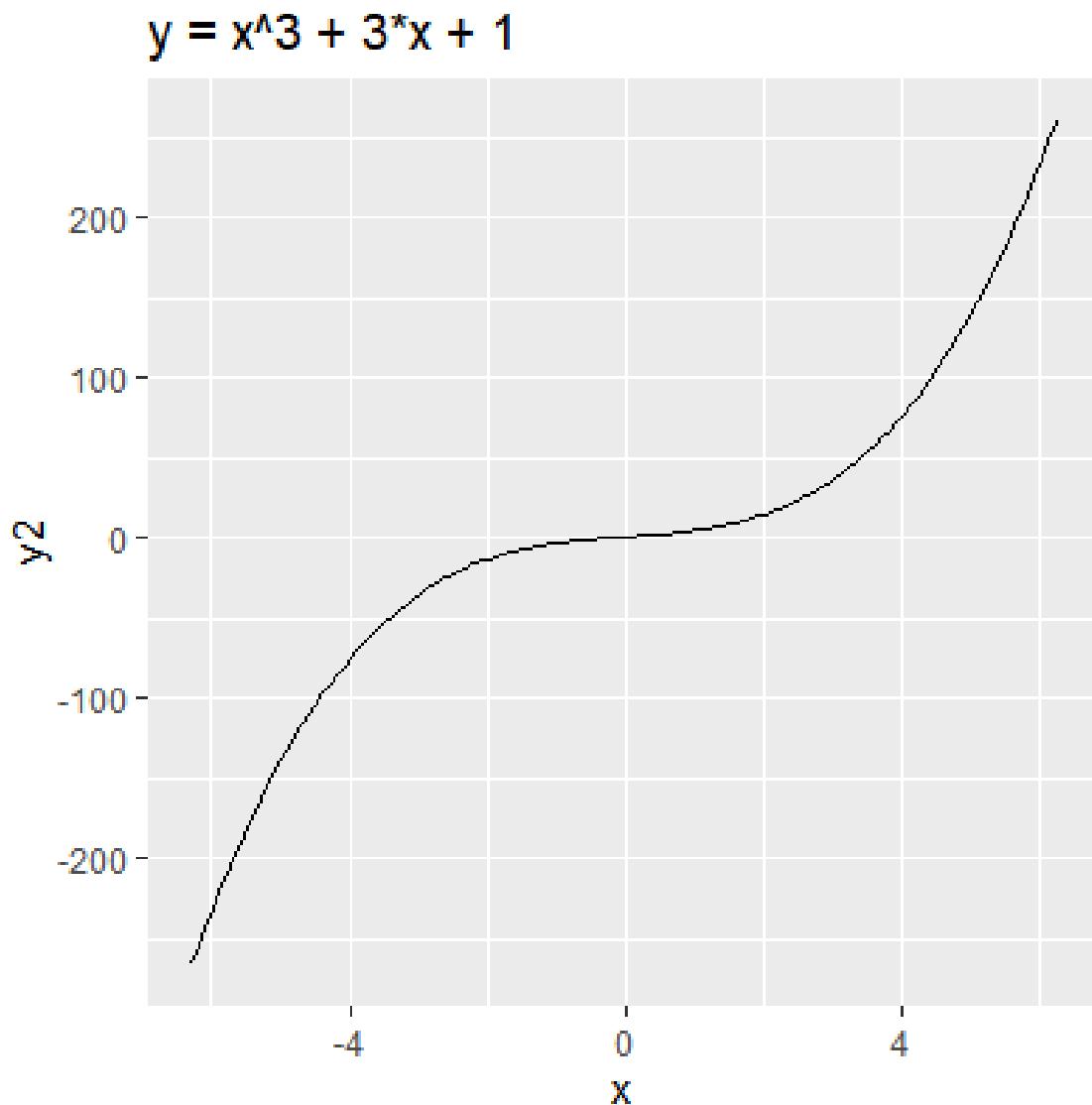
A variável 'p1' recebe o gráfico de y1 através da função "ggplot()". Essa função cria um sistema de coordenadas em que é possível adicionar camadas. Pode-se rodar o programa a partir dessa linha, visto que o gráfico foi criado, mas apenas mostraria um gráfico vazio, ou seja, precisamos adicionar camadas à ele.

Para completar o gráfico vamos usar a função 'geom_line', que adiciona uma camada de linha para o nosso gráfico (o pacote disponibiliza diversas formas de gráficos, mas utilizaremos de linha). Cada função no ggplot2 possui um mapeamento de argumentos que definem como as variáveis serão mapeadas em um gráfico. Podemos ver esse mapeamento com o 'aes()' que tem como argumentos x e y, que especificam quais variáveis se devem mapear nos eixos x e y. Nesse caso, a variável X foi a mesma para os três gráficos, e o Y foi variando entre as equações definidas. Por fim, fora do mapeamento, é definida a cor da linha manualmente com o "color = red" no 'p1' e "color = green" no 'p3'. O 'p2' não foi definida a cor e, por isso, foi estabelecido como preto.

Depois de adicionadas as camadas do gráfico, foi utilizada a função 'labs()' para adicionar o

título. Essa função é utilizada para adicionar rótulos, em geral, ao gráfico. Nesse caso, adicionamos apenas o título, que é a equação que o gráfico representa.

Figura 3.4: Gráfico p2:



Fonte: O autor deste trabalho

Na Figura acima, é mostrado o gráfico da equação 'p2'. Como dito acima, por não ter recebido mapeamento de cor, a linha foi padronizada preta.

3.3 Programas com Objetos

Aprendemos agora um pouco mais sobre a já comentada programação orientada a objetos. Como vimos, R possui três sistemas diferentes de orientação a objetos (s3, s4 e RC), e cada uma tem suas próprias características. Vamos então analisar uma aplicação partindo do princípio da Reference Class, que é o sistema mais completo dos três.

Figura 3.5: Reference Class

```

1 # Definir a classe "Retangulo"
2 Retangulo <- setRefClass(
3   "Retangulo",
4   fields = list(
5     comprimento = "numeric",
6     largura = "numeric"
7   ),
8   methods = list(
9     area = function() {
10       comprimento * largura
11     },
12     perimetro = function() {
13       2 * (comprimento + largura)
14     }
15   )
16 )
17
18 # Criar um objeto retângulo
19 meuRetangulo <- Retangulo(comprimento = 5, largura = 3)
20
21 # Calcular e exibir a área e o perímetro do retângulo
22 cat("Área:", meuRetangulo$area(), "\n")
23 cat("Perímetro:", meuRetangulo$perimetro(), "\n")
24
25

```

Fonte: O autor deste trabalho

Logo no começo, definimos uma nova classe 'Retangulo' com a função "setRefclass" que é exclusiva para criação de classes no Reference Class, assim iniciando nosso projeto nesse sistema.

Em seguida, estabelecemos seus atributos utilizando o 'fields', que é a forma utilizada pelo RC, e criamos essa lista de atributos que consiste no comprimento e na largura, ambos numéricos. Por fim, foram definidos os métodos da classe, esses sendo 'area', que é uma função que multiplica o comprimento pela largura, e 'perimetro' que é uma função que multiplica por 2 a soma dos dois atributos.

Segundo então, criamos um novo objeto "meuRetangulo" pertencente à essa classe atribuindo a ele a passagem da classe com os atributos comprimento e largura como 5 e 3, respectivamente.

Por fim, imprimimos a área do objeto "meuRetangulo" ao chamar o método área com o "meuRetangulo\$area". A mesma coisa é feita com o perímetro.

3.4 O algoritmo Quicksort - Implementação

Caminharemos sobre um algoritmo importante de ordenação chamado de Quicksort. Trabalhando com base em "dividir e conquistar", esse algoritmo define um pivô onde todos os números à sua esquerda são menores do que ele e todos à sua direita são maiores que ele, então recursivamente faz o mesmo procedimento com suas partes divididas. Entenderemos melhor na prática.

Figura 3.6: Quicksort

```

1 # Função de ordenação quicksort
2 quicksort <- function(vet) {
3   if (length(vet) <= 1) {
4     return(vet)
5   } else {
6     pivot <- vet[0]
7     smaller <- vet[vet < pivot]
8     equal <- vet[vet == pivot]
9     greater <- vet[vet > pivot]
10
11    return(c(quicksort(smaller), equal, quicksort(greater)))
12  }
13}
14
15 # Exemplo de uso
16 vetor <- c(9, 4, 7, 2, 1, 5, 3, 6, 8)
17 ordenado <- quicksort(vetor)
18 cat("vetor ordenado:", ordenado, "\n")
19
20 |

```

Fonte: O autor deste trabalho

3.4.1 Chamada da função

Por organização, vamos, primeiramente, analisar a parte de baixo do código. Primeiramente, um vetor desordenado é declarado na variável 'vetor'. Então é declarada a variável 'ordenado', que recebe o que é retornado da função "quicksort(vetor)"(analisaremos ela em seguida). Por fim, é impresso na tela essa variável 'ordenado'. Já veremos o que ela recebe.

3.4.2 Quicksort

Agora vamos discorrer sobre a função de ordenação. Primeiro, percebemos que ela recebe como argumento o vetor desordenado que foi declarado embaixo, então entra num 'If-else' que tem como condição o 'length(vet) <= 1', ou seja, se o tamanho do vetor for menor ou igual a 1, ele entra na condição de retornar o vetor como está (essa condição que dará fim à nossa recursão, veremos à frente), caso não satisfaça a condição e o vetor seja maior que 1, a função passa a declarar quatro variáveis: um pivô que recebe o valor que está na posição 0 do vetor, um smaller que recebe um vetor com todos os valores menores que o valor de pivô, um equal que recebe um vetor com todos os números iguais ao valor de pivô, e um greater que recebe um vetor com todos os números maiores. Ou seja, agora já temos a primeira divisão: um vetor à esquerda com todos os valores menores que o pivô, o meio e um vetor à direita com todos os maiores.

Agora é que a magia acontece. Depois de criar esses três dispositivos, a função retorna três coisas: o valor recebido de uma nova função quicksort, mas dessa vez passando o vetor recebido por smaller como argumento, o equal e o valor recebido de mais uma nova função quicksort, mas dessa vez passando o vetor recebido por greater como argumento. Assim acontecem mais duas ordenações paralelas, a do vetor smaller e do vetor greater, e cada uma dessas gera mais duas, e mais duas... Assim vai até cada uma se resolver totalmente e os vetores passados sejam apenas um valor, fazendo o que foi dito acima, saírmos do if, retornando a concatenação de cada uma dessas ordenações com a função 'c()' para a variável 'ordenado', que é impresso logo após.

Figura 3.7: Quicksort

```
> vetor
[1] 9 4 7 2 1 5 3 6 8
> ordenado
[1] 1 2 3 4 5 6 7 8 9
>
```

Fonte: O autor deste trabalho

Por mais que um pouco complexa, essa forma de ordenação é bastante eficiente.

3.5 Aplicações com Banco de Dados

Por fim, veremos a aplicação da R com os bancos de dados. Analisaremos um código onde utilizaremos o banco de dados disponibilizado no 'SQLite', criaremos uma tabela básica de dados e a imprimiremos, recebendo essas informações. Vamos analisar.

Figura 3.8: Quicksort

```
1 # Instale o pacote 'RSQLite' se ainda não estiver instalado
2 install.packages("RSQLite")
3
4 # Carregue o pacote 'RSQLite'
5 library(RSQLite)
6
7 # Crie uma conexão com o banco de dados
8 con <- dbConnect(RSQLite::SQLite(), "exemplo.db")
9
10 # Crie uma tabela chamada 'clientes'
11 dbExecute(con, "CREATE TABLE clientes (id INTEGER, nome TEXT, idade INTEGER)")
12
13 # Insira alguns dados na tabela
14 dbExecute(con, "INSERT INTO clientes (id, nome, idade) VALUES (1, 'João', 25)")
15 dbExecute(con, "INSERT INTO clientes (id, nome, idade) VALUES (2, 'Maria', 30)")
16 dbExecute(con, "INSERT INTO clientes (id, nome, idade) VALUES (3, 'Pedro', 35)")
17
18 # Execute uma consulta SQL para recuperar todos os registros
19 dados <- dbGetQuery(con, "SELECT * FROM clientes")
20
21 # Exiba os dados recuperados
22 print(dados)
23
24 # Execute uma consulta SQL para recuperar apenas clientes com idade superior a 30
25 dados_mais_velhos <- dbGetQuery(con, "SELECT * FROM clientes WHERE idade > 30")
26
27 # Exiba os dados recuperados
28 print(dados_mais_velhos)
29
30 # Feche a conexão com o banco de dados
31 dbDisconnect(con)
```

Fonte: O autor deste trabalho

3.5.1 Conectando e criando o banco de dados

Primeiramente, precisaremos instalar o pacote 'RSQLite', que é um pacote que permite a conexão e interação com bancos de dados SQLite usando o R. Em seguida, carregamos esse pacote no nosso programa. Então estabelecemos uma conexão com o banco de dados. A função 'dbConnect()' é usada para criar essa conexão, e o argumento RSQLite::SQLite() especifica o driver SQLite para a conexão. O segundo argumento, "exemplo.db", é o nome do arquivo do banco de dados SQLite que será criado. Nesse caso, o arquivo será chamado "exemplo.db".

Por fim, depois de termos instalado o pacote e conectado ao banco de dados, usaremos a função 'dbExecute()' para criar uma nova tabela no banco de dados. A função recebe dois argumentos: 'con' representa a conexão com o banco de dados e a criação da tabela clientes com o comando 'CREATE TABLE <nome da tabela>', recebendo então três elementos, 'id' e 'idade' sendo inteiros e 'nome' sendo texto.

3.5.2 Inserindo dados e os exibindo

Após criada a tabela, utilizaremos novamente a função 'dbExecute()', dessa vez recebendo, além do argumento con, argumento com o comando 'INSERT INTO <tabela>' para inserir dados na tabela com o comando 'VALUES <valores inseridos>', inserindo então o id, o nome e a idade de cada elemento da tabela.

Após inseridas as informações, a variável 'dados' recebe o retornado pela função 'dbGetQuery()'. Essa função é usada para executar uma consulta no banco de dados e recuperar os resultados. A consulta SQL especificada é "SELECT * FROM clientes", que retorna todos os registros da tabela "clientes". Então imprime na tela essa variável 'dados'.

Cria uma outra variável 'dados_mais_velhos' que recebe o retornado por outra função 'dbGetQuery()'. Dessa vez é executada para recuperar apenas os clientes com idade superior a 30. A consulta é "SELECT * FROM clientes WHERE idade > 30". Imprime esses dados mais velhos e para terminar, desconecta do banco de dados.



Referências Bibliográficas

- [Cha19] Winston Chang. *R Graphics Cookbook*. O'Reilly UK Ltd., Sebastopol, CA, 2 edition, February 2019. Citado na página 7.
- [Cot13] Richard Cotton. *Learning R*. O'Reilly, Beijing Sebastopol, CA, 2013. Citado 4 vezes nas páginas 7, 11, 12 e 19.
- [DD18] Paul Deitel and Harvey Deitel. *Java: How to Program*. How to Program. Pearson, Boston, MA, 11th edition, 2018. Citado na página 9.
- [Kab15] Robert Kabacoff. *R in action : data analysis and graphics with R*. Manning, Shelter Island, 2015. Citado 2 vezes nas páginas 8 e 16.
- [Lan17] Jared Lander. *R for Everyone*. Addison-Wesley Data and Analytics Series. Pearson Education (US), New York, NY, 2 edition, June 2017. Citado 2 vezes nas páginas 7 e 23.
- [Lau08] Marcos Laureano. *Estrutura de Dados com Algoritmos e C*. Brasport Livros e Multimídia Ltda, 1 edition, 2008. Citado na página 13.
- [Mai17] Thomas Mailund. *Advanced Object-Oriented Programming in R: Statistical Programming for Data Science, Analysis and Finance*. Apress, Berkeley, CA, 2017. Citado na página 9.
- [Spi21] David Spiegelhalter. *The Art of Statistics: How to Learn from Data*. Pelican Books, London, 1st edition, 2021. Citado na página 8.
- [WG16] Hadley Wickham and Garrett Grolemund. *R for data science*. O'Reilly Media, Sebastopol, CA, 1 edition, 2016. Citado 3 vezes nas páginas 8, 26 e 27.