FUNDAMENTOS DE PROGRAMAÇÃONA CIÊNCIA DE DADOS

AUTOR
João Vicente Ferreira Lima

FUNDAMENTOS DE PROGRAMAÇÃO NA CIÊNCIA DE DADOS

AUTOR
João Vicente Ferreira Lima

1ª Edição
UAB/CTE/UFSM

UNIVERSIDADE FEDERAL DE SANTA MARIA
Santa Maria | RS
2023

PRESIDENTE DA REPÚBLICA FEDERATIVA DO BRASIL
Luiz Inácio Lula da Silva

MINISTRO DA EDUCAÇÃO
Camilo Sobreira de Santana

PRESIDENTE DA CAPES
Mercedes Maria da Cunha Bustamante

UNIVERSIDADE FEDERAL DE SANTA MARIA

REITOR
Luciano Schuch

VICE-REITORA
Martha Bohrer Adaime

PRÓ-REITOR DE PLANEJAMENTO
Rafael Lazzari

PRÓ-REITOR DE GRADUAÇÃO
Jerônimo Siqueira Tybusch

PRÓ-REITORA DE PÓS-GRADUAÇÃO E PESQUISA
Cristina Wayne Nogueira

COORDENAÇÃO DO CURSO DE ESPECIALIZAÇÃO EM CIÊNCIA DE DADOS NA ADMINISTRAÇÃO PÚBLICA – UAB/UFSM
Vânia Medianeira Flores Costa

COORDENADORIA DE TECNOLOGIA EDUCACIONAL

COORDENADORA DA CTE
Liziany Müller

CHEFE DA SUBDIVISÃO DE TECNOLOGIAS EDUCACIONAIS EM REDE (STER)
Victor Matheus Portela Ribeiro

CHEFE DA SUBDIVISÃO DE EDUCAÇÃO A DISTÂNCIA (SEAD)
Elizeu da Silva Costa Junior

COORDENADORIA DE TECNOLOGIA EDUCACIONAL

COORDENADORA DA CTE
Liziany Müller

ELABORAÇÃO DO CONTEÚDO
João Vicente Ferreira Lima

REVISÃO PEDAGÓGICA
Raiane da Rosa Dutra
Karine Josieli König Meyer

REVISÃO LINGUÍSTICA
Maria Carolina de Christo Lima

APOIO PEDAGÓGICO
Patrícia Nunes Pezzini
Raiane da Rosa Dutra
Karine Josieli König Meyer

EQUIPE DE DESIGN
Ana Caroline Alves Crema
Ana Luiza Mozzaquatro de Mattos
Marcel Santos Jacques

EQUIPE DE DIAGRAMAÇÃO
Ana Carolina Scherer Cipriani
Ana Caroline Alves Crema

PROJETO GRÁFICO
Ana Letícia Oliveira do Amaral

L732f Lima, João Vicente Ferreira
Fundamentos de programação na ciência de dados [recurso eletrônico] / João Vicente Ferreira Lima. – 1. ed. – Santa Maria, RS : UFSM, CTE, 2023.
1 e-book. : il.

"Este caderno foi elaborado pela Coordenadoria de Tecnologia Educacional da Universidade Federal de Santa Maria"
ISBN 978-85-64049-76-5

1. Ciência de dados 2. Algoritmo 3. Programação 4. Análise de dados 5. Linguagem de programação I. Universidade Federal de Santa Maria. Coordenadoria de Tecnologia Educacional II. Título.

CDU 004.43

Ficha catalográfica elaborada por Lizandra Veleda Arabidian - CRB-10/1492
Biblioteca Central - UFSM

APRESENTAÇÃO

Este livro foi produzido para servir como material didático básico para a disciplina Fundamentos de Programação na Ciência de Dados do curso de Especialização em Ciências de Dados na Gestão Pública.

O objetivo da disciplina é apresentar aos discentes os fundamentos da construção de algoritmos e da programação de soluções para problemas típicos de ciência de dados, em linguagens Python e R. Ao final da disciplina, os estudantes deverão ser capazes de manejar e aplicar algoritmos para manipulação e análise de dados tipicamente encontrados em arquivos de bases públicas.

O livro está organizado em 4 unidades assim divididas:

Unidade 1: Conceitos de Algoritmos e Programação: apresenta os principais conceitos sobre algoritmos, programas e estruturas de fluxo de controle;

Unidade 2: A Linguagem de Programação Python: apresenta as principais características da linguagem de programação Python tais como operadores, ambientes de programação, variáveis, estruturas de dados, estruturas de fluxo e depuração;

Unidade 3: A Linguagem de Programação R - apresenta as principais características da linguagem de programação R tais como operadores, ambientes de programação, variáveis, estruturas de dados, estruturas de fluxo e depuração;

Unidade 4: Situações-Problema de Programação em Ciência de Dados: apresenta uma introdução sobre análise de dados nas linguagens de programação Python e R, seguido de um exemplo de análise com os dados públicos disponíveis pelo Instituto Nacional do Seguro Social (INSS).

Neste livro, assim como na disciplina, usaremos o conceito de Literate Programming ou programação letrada introduzido por Donald E. Knuth em 1992. O conceito consiste em combinar código-fonte com documentação podendo ser texto, gráficos, tabelas, etc. O principal objetivo é escrever e documentar programas como se fossem textos literários de modo que possam ser lidos, entendidos e apreciados por seres humanos.

ENTENDA OS ÍCONES

ATENÇÃO: faz uma chamada ao leitor sobre um assunto, abordado no texto, que merece destaque pela relevância.

INTERATIVIDADE: aponta recursos disponíveis na internet (sites, vídeos, jogos, artigos, objetos de aprendizagem) que auxiliam na compreensão do conteúdo da disciplina.

SAIBA MAIS: traz sugestões de conhecimentos relacionados ao tema abordado, facilitando a aprendizagem do aluno.

TERMO DO GLOSSÁRIO: indica definição mais detalhada de um termo, palavra ou expressão utilizada no texto.

SUMÁRIO

1

CONCEITOS DE ALGORITMOS E PROGRAMAÇÃO

INTRODUÇÃO

O estudo de algoritmos data desde matemáticos persas e gregos quando era utilizado para cálculos matemáticos. O algoritmo consiste em uma sequência de passos finitos para resolver um problema. A escrita de algoritmos pode ser feita em linguagem natural e segue algumas regras para evitar ambiguidades.

Uma questão pertinente é: por que não escrevemos programas de computador em português? A linguagem natural é usada para comunicação entre humanos e de fácil compreensão. O problema está na ambiguidade onde uma sentença pode ter significados diferentes a depender do contexto e ter diferentes interpretações. O computador não é capaz de interpretar sentenças e executa comandos exatos sem questionar. Também não podemos escrever programas em binário porque apenas computadores entendem esse formato. A compreensão de um programa binário por humanos seria praticamente impossível.

O desenvolvimento de algoritmos para computador depende das linguagens de programação. Elas definem uma gramática com sintaxe e semântica, a fim de aproximar o algoritmo da máquina. As instruções são claras, precisas, sem ambiguidade, de forma que humanos conseguem ler e compreender o código-fonte. Não obstante, um programa intermediário recebe esse código-fonte e traduz para binário de modo que o computador entenda as instruções escritas.

Este capítulo introduz os principais conceitos sobre algoritmo, programação e pseudocódigo. Veremos os principais operadores matemáticos e lógicos além das principais estruturas de decisão e repetição.

1.1 ALGORITMO E PROGRAMAÇÃO

Um algoritmo é uma sequência de passos finitos que recebe um conjunto de valores como entrada e transforma esses dados em saída (CORMEN; LEISERSON; RIVEST, 2009). Esses passos são escritos em linguagem natural de forma que outros humanos entendam o algoritmo e o problema que ele está resolvendo. Uma analogia ao algoritmo é a receita de um bolo, que descreve ingredientes e modo de preparo, na qual ordem dos passos segue uma sequência a ser respeitada estritamente, obedecendo a uma sequência lógica. Cada etapa não pode ser dividida em outras e não faz sentido isoladamente, como por exemplo: "bater ovos". Também devemos evitar ambiguidades da linguagem natural na escrita de algoritmos.

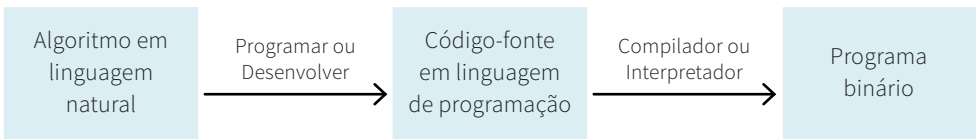
O programa é também uma sequência de instruções em formato binário para ser executado por um computador. O *hardware* do computador executa as instruções de forma rígida sem interpretações. O formato binário é a forma que o computador entende instruções que são incompreensíveis para seres humanos.

Uma linguagem de programação é o meio termo entre um algoritmo em linguagem natural e o programa em formato binário. Ela é definida por uma gramática onde descreve um conjunto de regras de sintaxe e semântica. A linguagem de programação tem uma definição mais restrita e cada instrução corresponde a uma ou várias operações básicas no computador. Algumas das linguagens de programação mais populares são Python, Java, C++, C, Javascript, R, etc (PIVA JR, 2019).

As linguagens de programação podem ser compiladas ou interpretadas. O código-fonte das linguagens compiladas deve ser transformado em binário através do **compilador** que é um programa especial que verifica a gramática do código e gera o programa binário para execução. Uma linguagem interpretada traduz o código-fonte em tempo de execução através do **interpretador**. As linguagens Python e R são linguagens interpretadas, ou seja, a tradução de código para binário é feita durante a execução.

A Figura 1 mostra os passos do desenvolvimento de um programa através de algoritmo, código-fonte e programa binário.

Figura 1 - Etapas no processo entre algoritmo e um programa



Fonte: Autor.

1.2 PSEUDOCÓDIGO

Neste capítulo, os exemplos de algoritmos são descritos em pseudocódigo que é uma descrição de algoritmo em linguagem natural menos rigorosa que uma linguagem de programação (MENÉNDEZ, 2023). O pseudocódigo deve ser fácil de entender e independente de linguagem de programação. Podemos listar algumas regras de escrita tais como:

- Um verbo por frase;
- Não escrever instruções que apenas programadores entendem;
- Frases curtas e simples;
- Ser objetivos;
- Usar palavras sem duplo sentido.

Um pseudocódigo tem o formato abaixo:

```
ALGORITMO NOME
INÍCIO
          TIPO: VARIÁVEIS
          BLOCO DE COMANDOS
FIM
```

O algoritmo começa com a declaração das variáveis necessárias seguido dos comandos executados.

1.2.1 Variáveis

Variáveis são abstrações entre um nome e um valor em memória. Uma variável representa um lugar ou endereço na memória do computador contendo dados de um determinado tipo. As variáveis devem ser declaradas no início do algoritmo para determinar o nome e tipo de dado associado. O nome de uma variável é o identificador e não pode ser alterado durante a execução. O tipo de dado delimita o intervalo de valores que uma variável pode receber. Os tipos básicos podem ser:

- **Inteiros:** qualquer número inteiro como idade;
- **Real:** número com casas decimais ou ponto flutuante tais como salário;
- **Cadeia de caracteres:** letras, números e caracteres especiais como nome, endereço residencial, etc;
- **Lógico:** valores verdadeiro ou falso.

Esse valor da variável pode variar durante a execução do algoritmo através de uma operação de atribuição a depender da sintaxe da linguagem de programação. São exemplos de alteração de valor as linhas abaixo:

```
MEDIA = (N1+N2)/2
LER MEDIA
```

Podemos dizer que *MEDIA* recebe o valor da expressão matemática "*(N1+N2)/2*"

1.2.2 Entrada e saída

A entrada de dados é uma forma de obter os dados necessários para chegar ao resultado desejado. Essa entrada pode ser tanto por entrada do usuário como teclado quanto arquivos como textos ou planilhas. Em pseudocódigo podemos escrever:

```
LER ALTURA
```

Onde o comando *LER* pausa a execução do programa neste ponto e espera a entrada do usuário para armazenar o valor na variável *ALTURA*. Os valores aceitos dependem do tipo da variável declarado no começo do algoritmo.

A saída de dados é a forma de informar os resultados do processamento desejado ao usuário. Essa saída pode ser tanto por impressão quanto escrita em arquivos. Em pseudocódigo, podemos representar o comando de saída como:

```
ESCREVER 'A altura é', ALTURA
```

Onde *ESCREVER* é o comando para mostrar alguma informação. A saída terá uma cadeia de caracteres delimitas por aspas simples 'A altura é' seguida de uma variável *ALTURA*. A separação entre elementos de saída no comando *ESCREVER* é feita por vírgula.

1.3 OPERADORES ARITMÉTICOS, RELACIONAIS E LÓGICOS

Os operadores representam a manipulação, ou processamento, de informações para resolução de um problema. Os operadores podem ser aritméticos, relacionais e lógicos.

1.3.1 Operadores aritméticos

Os operadores aritméticos atuam sobre operandos que podem ser constantes numéricas, variáveis ou funções matemáticas. O Quadro 1 mostra os operadores matemáticos mais comuns nas linguagens de programação.

Quadro 1: Operadores aritméticos

SÍMBOLO	OPERAÇÃO
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
//	Divisão com quociente inteiro
%	Resto da divisão de dois números inteiros
**	Exponenciação

Fonte: Autor.

A precedência de um operador pode ser determinada por parêntesis dentro de uma expressão matemática. Toda expressão é avaliada da esquerda para direita conforme a lista abaixo em ordem de prioridade:

1. Parêntesis mais interno;
2. Exponenciação ou raiz quadrada (se existir);
3. Multiplicação e divisão;
4. Adição e subtração.

Podemos usar como exemplo a expressão matemática abaixo:

```
X = 3 + 24/2**2 - 3
```

O exemplo não tem parêntesis. Então, a primeira operação será a exponenciação "2**2" seguido da divisão "24/4" que resulta em 6. A expressão final será "3 + 6 – 3" que resulta em 6.

1.3.2 Operadores relacionais

Operadores relacionais comparam valores que resultam em um valor lógico verdadeiro ou falso. Eles são fundamentais à tomada de decisões em algoritmos e outras áreas tais como matemática e filosofia. O Quadro 2 lista os operadores relacionais mais comuns em algoritmos.

Quadro 2: Operadores relacionais

SÍMBOLO	OPERAÇÃO
A == B	Igual
A != B	Diferente
A < B	Menor que
A > B	Maior que
A <= B	Menor ou igual
A >= B	Maior ou igual

Fonte: Autor.

O exemplo do Quadro 3 mostra os resultados lógicos dos operadores relacionais assumindo como valores A=3, B=7 e C=7.

Quadro 3: Exemplo de operações relacionais sobre A, B e C

EXPRESSÃO	RESULTADO
(A+C) > B	Verdadeiro
B >= (A+2)	Falso
C == (B-A)	Falso
(B+A) <= C	Falso
(C+A) > B	Verdadeiro

Fonte: Autor.

1.3.2 Operadores lógicos

Os operadores lógicos combinam o resultado de expressões lógicas retornando se é verdadeiro ou falso. Essas expressões devem ter por resultado um valor lógico de verdadeiro ou falso. Os operandos de um operador lógico são valores lógicos e o resultado é lógico. O Quadro 4 mostra os três principais operadores lógicos.

Quadro 4: Operadores lógicos

SÍMBOLO	OPERAÇÃO
AND	Verdadeiro se todos são verdadeiros
OR	Verdadeiro se pelos menos um é verdadeiro
NOT	Inverte o valor da expressão

Fonte: Autor.

Os exemplos do Quadro 5 mostram a aplicação de operações lógicas tendo como valores A=5, B=8 e C=1.

Quadro 5: Exemplos de operadores lógicos

EXPRESSÃO	PARCIAL	RESULTADO
(A==B) AND (B>C)	(V AND F)	Falso
(A != B) or (B < C)	(V AND F)	Verdadeiro
NOT (A > B)	(NOT F)	Verdadeiro
(A < B) AND (B > C)	(V AND V)	Verdadeiro
(A >= B) OR (B == C)	(F OR F)	Falso
NOT (A <= B)	NOT V	Falso

Fonte: Autor.

1.4 ESTRUTURAS DE SELEÇÃO

Um algoritmo toma decisões no decorrer da execução baseado nos dados de entrada para processamento dos dados de saída. As decisões usam estruturas de seleção para determinar desvios no algoritmo, ou seja, se blocos de comando serão executados ou não. Esses desvios são baseados em expressões lógicas por meio de operadores relacionais e lógicos.

A estrutura de decisão *SE* determina se um bloco de instruções deve ser executado apenas quando a condição for verdadeira. Por exemplo, um teste para saber se um número informado pelo usuário é positivo em pseudocódigo:

```
ALGORITMO 'NÚMERO POSITIVO'
INÍCIO
          NUMÉRICO: N
          LER N
          SE N > 0 ENTÃO
                ESCREVER 'O número é positivo'
FIM
```

A delimitação entre o início e o fim do bloco de instruções dentro da estrutura SE deve ser feito por espaçamentos no começo da linha tais como espaços ou recuo.

A estrutura *SE* pode vir acompanhada do *SENÃO*, que executa um bloco de instruções caso a condição for falsa. Complementando nosso exemplo anterior para informar quando um número é negativo:

```
ALGORITMO 'NÚMERO POSITIVO'
INÍCIO
          NUMÉRICO: N
          LER N
          SE N > 0 ENTÃO
                ESCREVER 'O número é positivo'
          SENÃO
                ESCREVER 'O número é negativo'
FIM
```

Há casos específicos onde devemos encadear vários testes condicionais. O exemplo abaixo mostra o exemplo de um algoritmo que determina se uma pessoa deve votar a partir de sua idade. O voto é obrigatório entre 18 e 70 anos, mas facultativo entre 16 e 17, e acima de 70. O algoritmo usa testes condicionais en-

cadeados para testar diferentes faixas de idade. Por fim, o último *SENÃO* é executado apenas quando nenhuma das condições anteriores foram verdadeiras, ou seja, quando a idade informada é maior ou igual a 70 anos.

```
ALGORITMO 'IDADE PARA VOTO'
INÍCIO
          NUMÉRICO: IDADE
          LER IDADE
          SE IDADE < 16 ENTÃO
                ESCREVER 'Não pode votar'
          SENÃO SE IDADE >= 16 AND IDADE < 18 ENTÃO
                ESCREVER 'O voto é facultativo'
          SENÃO SE IDADE >= 18 AND IDADE < 70 ENTÃO
                ESCREVER 'O voto é obrigatório'
          SENÃO
                ESCREVER 'O voto é facultativo'
FIM
```

1.5 ESTRUTURAS DE REPETIÇÃO

As estruturas de repetição ou laços são mecanismos para repetição de um bloco de instruções por um número definido ou indefinido de vezes, ou até uma determinada condição.

A estrutura de repetição *PARA/FAÇA* permite a escrita de laços com repetição de número finito de passos. Para tanto, é necessário o uso de uma variável chamada *CONTADOR* que permite determinar quantas vezes repetimos o bloco de instruções. O exemplo abaixo permite imprimir 10x a frase 'Olá mundo!':

```
ALGORITMO 'REPETE 10x'
INÍCIO
          NUMÉRICO: CONTADOR
          PARA CONTADOR DE 1 ATÉ 10 FAÇA
                ESCREVER 'Olá mundo!'
FIM
```

A variável *CONTADOR* terá os valores 1, 2, até 10 quando termina o número de repetições necessárias. Note que nesse tipo de laço precisamos definir o número de repetições desejadas.

A estrutura *ENQUANTO/FAÇA* permite a repetição até que uma determinada condição seja alcançada. Essa condição deve ser uma expressão lógica que continua a repetição enquanto for verdadeiro. A repetição termina quando a condição alcançada resultar em falso. Caso contrário, se a condição nunca for alcançada, podemos ter o problema de laço infinito. O exemplo abaixo ilustra um algoritmo simples de verificação de senha:

```
ALGORITMO 'SENHA'
INÍCIO
          CADEIA DE CARACTERES: SENHA
          LER SENHA
          ENQUANTO SENHA != 'elefante' FAÇA
                ESCREVER 'Senha errada! Digite novamente:'
                LER SENHA
FIM
```

O algoritmo solicita ao usuário uma senha e o laço testa se o valor informado é diferente de 'elefante'. Quando a condição é verdadeira, significa que a senha está incorreta e o bloco de instruções é executado, ou seja, a senha é solicitada

novamente. Note que o número de repetições não é determinado e depende das tentativas de acerto.

Laços *ENQUANTO/FAÇA* podem ter seu número de repetições determinados por meio da utilização de uma variável contador. Esse contador é responsável por determinar o número de repetições que está sendo executado. O algoritmo abaixo repete 'Olá mundo!' 10x nesse formato:

```
ALGORITMO 'REPETE 10x'
INÍCIO
          NUMÉRICO: CONTADOR
          CONTADOR = 1
          ENQUANTO CONTADOR <= 10 FAÇA
                ESCREVER 'Olá mundo!'
                CONTADOR = CONTADOR + 1
FIM
```

A utilização do contador tem requisitos importantes para funcionamento do algoritmo. Primeiro, o contador deve ter um valor inicial como *CONTADOR = 1*. Em seguida, a condição determina qual será o valor de parada do laço. Aqui o laço repete enquanto o valor de *CONTADOR* é menor ou igual a 10, ou seja, quando o valor de *CONTADOR* atingir 11 a condição será falsa e o laço para de repetir. Por fim, *CONTADOR* precisa ser atualizado a cada repetição do laço com valores 1, 2, 3, ... etc como na linha *CONTADOR = CONTADOR + 1*.

ATENÇÃO: O nome da variável utilizada como contador pode ser qualquer nome válido. Exemplos de nomes para contadores são: CONTADOR, I, J, etc.

ATIVIDADES | UNIDADE 1

1. Escreva um algoritmo que lê a, b e c de uma equação de 2º grau e diga se a equação possui: duas raízes reais; uma raiz única ou nenhuma raiz.

2. Escreva um algoritmo que leia como valor um ano qualquer e determine se esse ano é bissexto ou não. Um ano é bissexto se: é múltiplo de 400, ou é múltiplo de 4 e não é múltiplo de 100.

3. Escreva um algoritmo que efetue a leitura de três valores para os lados de um triângulo, considerando lados como: A, B e C. O algoritmo deverá verificar se os lados fornecidos formam realmente um triângulo (cada lado é menor que a soma dos outros dois lados). Se for esta condição verdadeira, deverá ser indicado qual tipo de triângulo foi formado: isósceles (dois lados iguais e um diferente), escaleno (todos os lados diferentes) ou equilátero (todos os lados são iguais).

4. Escreva um algoritmo que mostre a tabuada do 9.

2

A LINGUAGEM DE PROGRAMAÇÃO PYTHON

INTRODUÇÃO

Neste capítulo, veremos as principais características da linguagem de programação Python. Python é uma das mais utilizadas atualmente para ciência de dados e inteligência artificial e aparece no topo das principais listas de linguagens de programação. Criada no fim dos anos 80, ganhou popularidade a partir dos anos 2000. Seu nome teve origem no grupo de comédia britânica Monty Python como uma homenagem ao programa favorito de seu criador Guido van Rossum (LUTZ, 2013). Python é uma linguagem interpretada que pode ser compilada para melhor desempenho de execução.

Veremos os princípios básicos de programação, tais como operadores aritméticos, relacionais e lógicos. Os ambientes disponíveis para desenvolvimento Web tais como *Google Colab* e *Jupyter* são apresentados como opções para desenvolvimento sem necessidade de recursos computacionais e único requisito conexão de internet. Variáveis e tipos de dados essenciais são apresentados assim como estruturas de dados nativas.

As estruturas de fluxo de execução são parte fundamental de algoritmos e de linguagens de programação. Estruturas de decisão permitem desvios condicionais baseados em valores de entrada ou condições resultantes do processamento dos dados. Estruturas de repetições permitem executar um bloco de instruções diversas vezes podendo ter repetições indefinidas de acordo com condições lógicas ou definias. As principais estruturas de decisão e repetição são apresentadas.

O papel das funções é organizar o código e evitar repetição de código com reaproveitamento de tarefas repetitivas. A depuração em Python possibilita detecção de erros em execução e a execução de blocos de instruções para tratar estes erros. Ambos os assuntos finalizam este capítulo.

2.1 PRINCIPAIS CARACTERÍSTICAS

A linguagem de programação Python foi criada no fim dos anos 80 por Guido van Rossum como um sucessor da linguagem de programação ABC para o sistema operacional Amoeba. A primeira versão 0.9.0 foi lançada em 1991, tendo crescido sua base de usuários desde então. A linguagem aparece com umas das mais populares nos principais rankings especializados. A Figura 2 mostra como imprimir a frase 'Olá mundo!'.

Figura 2: Exemplo do 'Olá mundo!' em Python

```
1 print( "Ola mundo!" )
```

Fonte: Autor

INTERATIVIDADE: Saiba mais acessando:
https://www.tiobe.com/tiobe-index/
https://spectrum.ieee.org/top-programming-languages-2022

Uma das suas principais características é a codificação de programas que não precisam de acesso direto ao *hardware* do computador. Ou seja, o código-fonte precisa ser interpretado ou traduzido para binário em tempo de execução. O interpretador Python é o responsável pela tradução entre código-fonte e binário que permite sua portabilidade entre diferentes sistemas operacionais e plataformas de *hardware*. Apesar de ser uma das vantagens da linguagem, a interpretação em tempo de execução penaliza o desempenho dos programas principalmente operações aritméticas (ZELLE, 2016).

2.1.1 Atribuição

A operação de atribuição é feita com um simples "=" significando que o valor a esquerda recebe o valor resultante da expressão a direita. Python também suporta a atribuição simultânea, ou seja, várias atribuições na mesma linha separadas por vírgula. A Figura 3 mostra duas atribuições: "soma" recebe "a+b" enquanto que "diferenca" recebe "a-b".

Figura 3: Exemplo de atribuição simultânea em Python

```
soma, diferenca = a+b, a-b
```

Fonte: Autor.

2.1.2 Entrada de dados

O comando para entrada de dados do usuário é a função "*input()*" que recebe como argumento uma mensagem a ser escrita com a solicitação de entrada. Quando um programa Python encontra esta função, ele pausa sua execução e espera que o usuário digite algo e pressione *ENTER*. A Figura 4 mostra o exemplo de um programa que pede nome e idade. O nome já foi digitado e lido na linha 1 e está aguardando a idade na linha 2. Note que a linha 3 faz uma conversão de tipos, pois a função "*input()*" lê apenas cadeias de caracteres, ou seja, quando digitamos "8" ele acha que é a letra "8" mas precisamos converter essa letra "8" no número 3.

Figura 4: Entrada de dados em Python

```
1 nome = input( 'Digite seu nome: ' )
2 idade = input( 'Digite sua idade: ' )
3 idade = int(idade)
```

```
Digite seu nome:  Joao
Digite sua idade:
```

Fonte: Autor.

2.1.3 Operadores aritméticos

Os principais operadores aritméticos estão listados no Quadro 6 incluindo operadores de números inteiros.

Quadro 6: Operadores aritméticos do Python

SÍMBOLO	OPERAÇÃO
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
//	Divisão com quociente inteiro
%	Resto da divisão de dois números inteiros
**	Exponenciação

Fonte: Autor.

2.1.4 Operadores relacionais e lógicos

Os operadores relacionais são aqueles que comparam dois valores e retornam verdadeiro (*True*) ou falso (*False*). O Quadro 7 ilustra os operadores relacionais de Python.

Quadro 7: Operadores relacionais em Python

SÍMBOLO	OPERAÇÃO
A == B	A igual a B
A != B	A diferente de B
A < B	A menor que B
A > B	A maior que B
A <= B	A menor ou igual a B
A >= B	A maior ou igual a B

Fonte: Autor.

Os operadores lógicos têm como operandos valores lógicos e retornam um valor lógico. O Quadro 8 mostra os operadores lógicos "`and`", "`or`" e "`not`" sendo "A" e "B" expressões de valor lógico.

Quadro 8: Operadores lógicos em Python

SÍMBOLO	OPERAÇÃO
A and B	*True* se A e B são *True*
A or B	*True* se A ou B são *True*
not A	Inverte o resultado

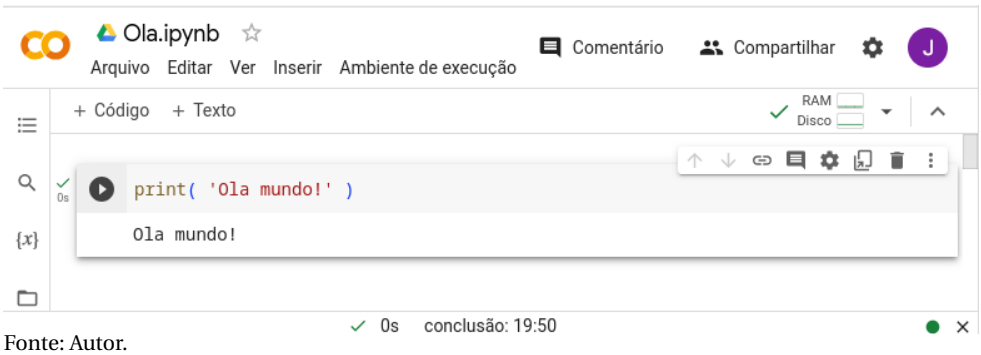
Fonte: Autor.

2.2 AMBIENTES DE PROGRAMAÇÃO

Python possui o interpretador de referência para programação e disponível de forma livre para *download* em seu site oficial (https://www.python.org/). O interpretador sozinho não possui as funcionalidades necessárias para análise de dados e produção de gráficos. Dessa forma, podemos optar por duas alternativas Web gratuitas de programação: *Google Colab* e *Jupyter*. Ambos seguem o paradigma de *Literate Programming* por meio de um *notebook* que é um documento contendo texto e blocos de código.

O Google Colab (https://colab.research.google.com/) é a plataforma do Google que oferece funcionalidades para cientistas de dados e pesquisadores em inteligência artificial. Ele oferece acesso à nuvem do Google que conta com *hardware* especializado além dos principais pacotes de *software* para Python. A plataforma é integrada com outros serviços tais como Google Drive e possibilita a contratação de mais recursos através de pagamento. O Colab é restrito à linguagem Python e tem limitações de tempo de uso. A Figura 5 ilustra um *notebook* vazio com o exemplo 'Olá mundo' em Python.

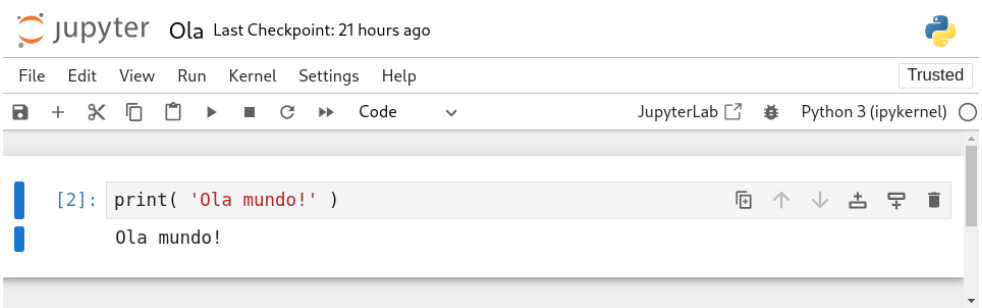
Figura 5: Plataforma Google Colab para Python



Fonte: Autor.

O Jupyter (https://jupyter.org/) é um projeto de código fonte livre criado em 2014 do antigo projeto IPython. O projeto suporta diversas linguagens de programação tendo foco em Python, R e Julia. O *software* pode ser instalado localmente ou usado online no site oficial do Jupyter. A Figura 6 mostra a interface Web do Jupyter com um programa 'Olá mundo!'.

Figura 6: Jupyter *notebook* para programação Python



Fonte: Autor.

2.3 VARIÁVEIS E TIPOS PRIMITIVOS

Variáveis são essencialmente nomes para valores na memória do computador como vimos anteriormente. Os nomes de variáveis, ou identificadores, podem conter os caracteres:

- Letras maiúsculas ou minúsculas entre "A" e "Z"
- O sublinhado "_"
- Dígitos de "0" a "9" exceto no começo do nome.

Assim sendo, nomes como "x", "celsius", "valor_total" e "_interno" são válidos enquanto que "1nome" é inválido. Espaços como no nome "meu teste" não são permitidos. Nomes com letras maiúsculas ou minúsculas definem nomes diferentes como em "celsius" e "Celsius".

Alguns identificadores fazem parte da linguagem Python, chamados **palavras reservadas**, e não podem ser usadas em outros nomes. O Quadro 9 mostra a lista completa de nomes reservados.

Quadro 9: Palavras-chave da linguagem Python

False	*Await*	*else*	*import*	*pass*
None	*Break*	*except*	*in*	*raise*
True	*Class*	*finally*	*is*	*return*
and	*Continue*	*for*	*lambda*	*try*
as	*Def*	*from*	*nonlocal*	*while*
assert	*Del*	*global*	*not*	*with*
async	*Elif*	*if*	*or*	*yield*

Fonte: *The Python LanguageReference.*

O tipo de um dado determina que valores pode ter e operações sobre esse dado. Os tipos primitivos em Python são:

- Número inteiro: *int*;
- Número com ponto flutuante: *float*;
- Número complexo: *complex*;
- Lógico: *bool* que tem apenas dois valores *True* (verdadeiro) e *False* (falso).

Python é definido como uma linguagem com tipagem dinâmica, ou seja, o tipo de uma variável é definido no momento da atribuição podendo ser alterado no decorrer da execução (SEVERANCE, 2016). A função especial *type( )* permite mostrar qual é o tipo associado a um dado ou variável. A Figura 7 mostra alguns exemplos de tipos primitivos. O primeiro trecho de código tem um exemplo da soma de dois números inteiros em "*x*", o segundo mostra um exemplo de "*y*" com número de ponto flutuante e o último atribui a "*z*" um número complexo.

ATENÇÃO: Python simplifica a representação de números complexos por meio da letra "j" após um valor numérico para escrever a parte imaginária do número.

Figura 7: Códigos Python com tipos primitivos

```
1 x = 2 + 3
2 print( type(x) )
```

```
<class 'int'>
```

```
1 y = 2.0
2 print( type (y) )
```

```
<class 'float'>
```

```
1 z = 2 + 5j
2 print( type )
```

```
<class 'type'>
```

Fonte: Autor.

2.4 ESTRUTURAS DE DADOS

As estruturas de dados nativas do Python oferecem formas de organizar os dados em alto nível sem penalidades de desempenho à execução de programas. As principais estruturas são listas e dicionários.

2.4.1 Listas

Listas são sequências de dados mais genéricas do que *string* providas pela linguagem. Os dados podem ser de qualquer tipo sem tamanho fixo de elementos. Listas também são mutáveis, ou seja, permitem alterações dos elementos por indexação. Uma lista pode ser criada por meio de colchetes "[]" e seus elementos dentro separados por vírgula. Uma lista de números "[*1, 2, 3, 4*]" tem quatro elementos e uma lista de tipos diferentes pode ser criada com "[*1, 'dois', 3, 'quatro'*]".

A Figura 8 ilustra alguns exemplos de manipulação de listas. A lista da linha 1 tem elementos numéricos e *string* totalizando três elementos como mostrado pela função "*len()*" da linha 4. O acesso a elementos por indexação usa "[]" com o número do elvemento como mostrado na linha 5. A linha 7 mostra a alteração de valor "0" para "9" do elemento na posição zero. A adição de elementos pode ser feita por adição (linha 10) ou através do método "*append()*" (linha 14).

Figura 8: Exemplos de métodos listas em Python

```
1  L = [1, 'ovo', 2]
2
3  print( L )
4  print( len(L) )
5  print( L[0] )
6
7  L[0] = 9
8  print( L[0] )
9
10 L = L + [3, 4]
11 print( L )
12
13 L.append('novo')
14 print( L )
```

```
[1, 'ovo', 2]
3
1


9


[9, 'ovo', 2, 3, 4]


[9, 'ovo', 2, 3, 4, 'novo']
```

Fonte: Autor.

2.4.2 Dicionários

Dicionários são mapas de dados em Python onde mapeamos dados através de chaves sem tamanho fixo de elementos. Essas chaves podem ser de tipos primitivos e indexam os dados em uma coleção. Um dicionário Python é definido por meio de chaves e seus elementos são separados por vírgula como em "*D= {'ovo': 4}*" onde a chave é "*ovo*" e o valor associado é "*4*". A indexação desse dicionário é feita por "*D['ovo']*". Os dicionários são mutáveis, ou seja, podemos alterar seus valores após a criação.

A Figura 9 abaixo ilustra exemplos básicos de dicionários. A linha 1 mostra a criação de um dicionário com diferentes tipos de elementos. Na linha 5, mostra a criação de uma nova associação da chave "*novo*" com o valor "9". A linha 9 mostra o exemplo de uma associação entre a chave "*items*" e a lista "*['sal', 'pimenta']*".

Figura 9: Exemplos de manipulação de dicionários em Python

```
1 D = { 'ovo' : 2, 'gema': 'sim' }
2 print( D )
3 print( D['gema'] )
4
5 D['novo'] = 9
6 print( D )
7
8 D['items'] = ['sal', 'pimenta']
9 print( D )
```

```
{'ovo': 2, 'gema': 'sim'}
sim


{'ovo': 2, 'gema': 'sim', 'novo': 9}


{'ovo': 2, 'gema': 'sim', 'novo': 9,
'items': ['sal', 'pimenta']}
```

Fonte: Autor.

2.5 ESTRUTURAS DE DECISÃO

Uma estrutura de decisão executa um conjunto de instruções caso uma determinada condição lógica seja avaliada como verdadeira. Nesta seção veremos as variações da estrutura de decisão SE do Python "if/elif" e SENÃO "*else*".

2.5.1 SE / IF

O teste condicional mais simples é o "if" que avalia uma condição lógica e executa as instruções abaixo quando esta condição é verdadeira ("*TRUE*"). A Figura 10 mostra o exemplo de comparação de dois números na linha 3. Note que o teste condicional da linha 3 termina com ":" para delimitar o fim das condições de teste e início das instruções. Logo abaixo, as instruções a serem executadas são delimitadas por um recuo tabular ou espaçamento no início de cada linha. Esse recuo tem por objetivo delimitar o início e o fim do bloco de código do "*if*". Note que a linha 5 não tem recuo, portanto, não está dentro da estrutura de decisão.

Figura 10– Exemplo da estrutura de decisão "*if*" em Python

```
1 x = 5
2 y = 4
3 if x > y:
4     print( 'O maior:', x )
5 print( 'Fim' )
6
```

```
O maior: 5
Fim
```

Fonte: Autor.

2.5.2 SENÃO / ELIF e ELSE

Teremos problemas específicos onde queremos avaliar outros testes condicionais ou mesmo executar instruções quando todas as condições lógicas resultarem em falso ("*False*"). O teste condicional "*elif*"deve vir após um teste "*if*"e será avaliado quando o teste condicional resultar em falso. Caso todos os testes anteriores forem falsos, podemos usar o "*else*" que deve vir seguido de um "*if*" ou "*elif*".

A Figura 11 ilustra um exemplo de testes condicionais com "*if* / *elif* / *else*". A estrutura do "*elif* "da linha 6 é semelhante ao "*if* " com uma expressão lógica e ":" para delimitar as instruções. O "*else*" da linha 8 executa seu bloco de instruções quando os testes do "*if* "e "*elif*" resultam em falso. Ele também tem um delimitador de instruções ":" sem qualquer expressão lógica.

Figura 11: Exemplo de estrutura de decisão *if* / *elif* / *else* em Python

```
x <- 5
y <- 4
if( x > y ){
    print( paste('O maior e:', x) )
}
print( 'Fim' )
```

```
[1] "O maior e: 5"
[1] "Fim"
```

Fonte: Autor.

2.6 ESTRUTURAS DE REPETIÇÃO

Uma estrutura de repetição ou laço repete um conjunto de instruções por um número determinado ou não de vezes até que uma certa condição seja alcançada. A linguagem Python conta com duas estruturas de repetição: *while* e *for*.

2.6.1 *while*

Uma estrutura de laço "*while*" repete um conjunto de instruções enquanto a condições lógica for verdadeira. A Figura 12 mostra um exemplo simples de laço onde repetimos 5x um comando para imprimir a frase "Olá mundo!". A linha 5 inicia uma variável "i" com zero para ser um contador, ou seja, um valor que determina o número de repetições executadas. A laço repete suas instruções enquanto "i" é menor que cinco na linha 2. A linha 4 incrementa o contador "i" que será testado novamente na linha 2. Note que o intervalo de valores para "i" está entre [0, 5) pois quando o valor de "i" é igual a cinco a condição será falsa e o laço será interrompido.

Figura 12 - Laço de repetição *while* com número determinado de repetições

```
1 i = 0
2 while i < 5:
3     print( "Ola mundo!" )
4     i= i + 1
```

```
Ola mundo!
Ola mundo!
Ola mundo!
Ola mundo!
Ola mundo!
```

Fonte: Autor.

Outra forma de usar laços "*while*" é quando não temos um número determinado de repetições até que a condição testada seja atendida. O exemplo da Figura 13 mostra um programa que entra em um laço enquanto a senha estiver incorreta. O programa pede a senha na linha 1 e testa se ela está incorreta solicitando novamente até o usuário digitar "elefante". Note que o número de repetições ficou indefinido sendo até que a condição "senha correta" seja atingida.

Figura 13: Laço de repetição *while* com número indeterminado de repetições

```
1 senha= input( 'Digite sua senha: ')
2 while senha != 'elefante':
3     senha = input( 'Senha errada! Digite novamente:' )
4 print( 'Senha correta! Bem vindo.' )
```

Fonte: Autor.

Outra forma de criar laços sem repetições determinadas é ao utilizar a instrução "*break*" que faz com que o laço seja interrompido. A Figura 14 mostra o programa anterior com a instrução "*break*" como critério de parada. Note que o teste condicional do laço "*while*" tem apenas o valor "*TRUE*" o que faz dele um laço sem condição de parada. A linha 3 é responsável pela condição de parada e testa se a senha está correta. Em caso afirmativo a instrução "*break*" interrompe o laço.

Figura 14 - Laço de repetição *while* com critério de parada utilizando *break*

```
1 while True:
2     senha = input( 'Digite a senha:' )
3     if senha == 'elefante':
4         break
5 print( 'Senha correta! Bem vindo.' )
```

Fonte: Autor.

2.6.1 *for*

Uma estrutura de laço "*for*" repete um conjunto de instruções com um número de repetições determinado. A Figura 15 mostra um exemplo simples de laço onde repetimos 5x um comando para imprimir a frase "Olá mundo!". A variável "i" logo após a palavra-chave "*for*" é chamada de índice do laço que assume cada um dos valores da sequência "range(5)" e executa o bloco de instruções para cada valor. A função "range()" recebe um inteiro como argumento e gera uma sequência de números.

Figura 15: Laço de repetição *for* com número determinado de repetições

```
1 for i in range(5):
2     print( 'Ola mundo!' )
```

```
Ola mundo!
Ola mundo!
Ola mundo!
Ola mundo!
Ola mundo!
```

Fonte: Autor.

O laço da Figura 16 é semelhante ao laço anterior, mas usa-se explicitamente uma lista de elementos. Nota-se que os valores impressos de "i" são cada um dos elementos da lista de dados.

Figura 16 - Laço de repetição *for* com lista de elementos

```
1 for i in [0, 1, 2, 3, 4]:
2     print( i )
3
4
```

```
0
1
2
3
4
```

Fonte: Autor.

2.7 FUNÇÕES

Uma função é um conjunto de instruções que podem ser executados diversas vezes. A função pode retornar um valor e receber valores de entrada como parâmetros. O código da Figura 17 mostra dois exemplos de funções simples. A primeira "*ola()*" não tem parâmetros nem retorno e apenas imprime a mensagem "Ola mundo!". A segunda função "*soma()*" tem dois parâmetros e retorna a soma de ambos. Importante destacar que a segunda função espera dois valores numéricos para efetuar a soma.

Figura 17: Uso de funções em Python

```
1  def ola():
2      print('Ola mundo!')
3
4  def soma(a, b):
5      return a + b
6
7  ola()
8  c = soma(2, 3)
9  print( c )
10
```

```
Ola mundo!
5
```

Fonte: Autor.

2.8 DEPURAÇÃO

A linguagem Python não tem programas depuradores por ser uma linguagem interpretada ao contrário de outras linguagens compiladas. A forma mais simples de entender o funcionamento do programa é por meio do estado de execução com impressão dos valores de variáveis durante a execução. Todavia, esse método não detecta erros em tempo de execução.

TERMO DO GLOSSÁRIO: **Depurador** ou *debugger* é um programa que testa outros programas e controlar suas ações. Em geral os depuradores oferecem execução passo a passo, suspensão do programa em um ponto (*breakpoint*), inspeção de contexto como variáveis, salvamento de todo o contexto de execução para análise futura (*post mortem*), etc.

A linguagem permite o uso de tratamento de execuções a fim de detectar erros e executar instruções em caso de falha. A estrutura é "*try*" seguido de um bloco de comandos a ser monitorado, "*except*" para executar um bloco de instruções em caso de erro, "*else*" se nenhum erro ocorrer e "finally" que sempre executa ao final. A Figura 18 ilustra um exemplo com erro de divisão por zero. Note que imprime a mensagem "Erro!" do bloco "*except*" e logo depois "Termina" do bloco "finally". Pode-se tratar erros específicos com o tipo de erro logo após a palavra-chave "*except*" como por exemplo a divisão por zero com "*ZeroDivisionError*" ou quando a memória do computador está cheia com "*MemoryError*".

Figura 18: Tratamento de erros com exceções em Python

```
1  a = 10
2  b = 0
3  try:
4      c = a / b
5  except:
6      print( 'Erro!' )
7  else:
8      print( 'Nenhum erro' )
9  finally:
10     print( 'Termina' )
11
```

```
Erro!
Termina
```

Fonte: Autor.

ATIVIDADES | UNIDADE 2

1. Escreva um programa em linguagem Python que leia como valor um ano qualquer informado pelo usuário e determine se esse ano é bissexto ou não. Um ano é bissexto se: é múltiplo de 400, ou é múltiplo de 4 e não é múltiplo de 100.

2. Desenvolva um programa que efetue a leitura de três valores para os lados de um triângulo, considerando lados como: A, B e C. O algoritmo deverá verificar se os lados fornecidos formam realmente um triângulo (cada lado é menor que a soma dos outros dois lados). Se for esta condição verdadeira, deverá ser indicado qual tipo de triângulo foi formado: isósceles (dois lados iguais e um diferente), escaleno (todos os lados diferentes) ou equilátero (todos os lados são iguais).

3. Utilize um laço for para calcular a soma dos 100 primeiros termos da série harmônica: 1/1 + 1/2 + 1/3 + 1/4 + 1/5 + .

4. Faça um programa que mostre a tabuada do 9, de 1 a 10.

3

A LINGUAGEM DE PROGRAMAÇÃO R

INTRODUÇÃO

Neste capítulo, veremos as principais características da linguagem de programação R. O R é uma das mais utilizadas atualmente para ciência de dados e aparece nas principais listas de linguagens de programação populares. R é uma linguagem interpretada que pode ser compilada para melhor desempenho de execução.

Veremos os princípios básicos de programação tais como operadores aritméticos, relacionais e lógicos. Os ambientes disponíveis para desenvolvimento tais como RStudio e Jupyter são apresentados como opções para desenvolvimento que podem ser instalados localmente ou Web. Variáveis e tipos de dados essenciais são apresentados assim como estruturas de dados nativas.

As estruturas de fluxo de execução são parte fundamental de algoritmos e de linguagens de programação. Estruturas de decisão permitem desvios condicionais baseados em valores de entrada ou condições resultantes do processamento dos dados. Estruturas de repetições permitem executar um bloco de instruções diversas vezes, podendo ter repetições indefinidas de acordo com condições lógicas ou definias. As principais estruturas de decisão e repetição são apresentadas.

O papel de funções é organizar o código e evitar repetição de código com reaproveitamento de tarefas repetitivas. A depuração em R possibilita verificar todos os passos de uma função e inserir pontos de depuração em qualquer função. Ambos os assuntos finalizam este capítulo.

3.1 PRINCIPAIS CARACTERÍSTICAS

A linguagem de programação R teve sua primeira versão publicada em 1993 inspirada na linguagem S. R tem ganhado popularidade nos últimos anos por sua utilização em estatística e ciência de dados tendo crescido nos principais rankings especializados. A Figura 19 ilustra um exemplo para mostrar "Ola mundo!".

Figura 19: Exemplo de "Olá mundo!" em R

```
1 print( "Ola mundo!" )
```

Fonte: Autor.

INTERATIVIDADE: Saiba mais em:

https://www.tiobe.com/tiobe-index/

https://spectrum.ieee.org/top-programming-languages-2022

O código-fonte precisa ser interpretado ou traduzido para binário em tempo de execução. O interpretador R é o responsável pela tradução entre código-fonte e binário que permite sua portabilidade entre diferentes sistemas operacionais e plataformas de *hardware*. Apesar de ser uma das vantagens da linguagem, a interpretação em tempo de execução penaliza o desempenho dos programas principalmente operações aritméticas (VERNABLES, SMITH; TEAM, 2023).

A operação de atribuição é feita com um simples "<-" significando que o valor a esquerda recebe o valor resultante da expressão a direita. R também suporta a atribuição à direta com "->". A Figura 20 mostra duas atribuições: "soma" recebe "a+b" enquanto que "diferenca" recebe "a-b" por atribuição à direita.

Figura 20: Operações de atribuição a variáveis em R

```
1 soma <- a+b
2 a-b -> diferenca
```

Fonte: Autor.

O comando para entrada de dados do usuário é a função "*readline()*" que irá ler a linha inteira digitada pelo usuário e recebe como argumento uma mensagem a ser escrita com a solicitação de entrada. Quando um programa R encontra esta função, ele pausa sua execução e espera que o usuário digite algo e pressione ENTER. A Figura 21 mostra o exemplo de um programa que pede nome e idade. O nome já foi digitado e lido na linha 1 e está aguardando a idade na linha 2. Note que a linha 3 faz uma conversão de tipos pois a função "*readline()*" lê apenas cadeia de caracteres, ou seja, quando digitamos "8" ele acha que é a letra "8" mas precisamos converter essa letra "8" no número 3.

Figura 21: Exemplo de entrada de dado em R

```r
nome <- readline('Digite seu nome: ')
idade <- readline('Digite sua idade: ')
idade <- as.integer(idade)
```

```
Digite seu nome:  Joao
Digite sua idade:
```

Fonte: Autor.

3.1.1 Operadores aritméticos

Os principais operadores aritméticos estão listados no Quadro 10, sendo que alguns diferem de outras linguagens como operadores de números inteiros e exponenciação.

Quadro 10: Operadores aritméticos em R

SÍMBOLO	OPERAÇÃO
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%/%	Divisão com quociente inteiro
%%	Resto da divisão de dois números inteiros
^	Exponenciação

Fonte: Autor.

3.1.2 Operadores relacionais e lógicos

Os operadores relacionais são aqueles que comparam dois valores e retornam verdadeiro (*TRUE*) ou falso (*FALSE*). O Quadro 11 ilustra os operadores relacionais de R.

Quadro 11: Operadores relacionais em R

SÍMBOLO	OPERAÇÃO
A == B	A igual a B
A != B	A diferente de B
A < B	A menor que B
A > B	A maior que B
A <= B	A menor ou igual a B
A >= B	A maior ou igual a B

Fonte: Autor.

Os operadores lógicos têm como operandos valores lógicos e resultam em um valor lógico. O Quadro 12 mostra os operadores lógicos "&", "|" e "!" sendo "A" e "B" expressões de valor lógico.

Quadro 12: Operadores lógicos em R

SÍMBOLO	OPERAÇÃO
A & B	TRUE se A e B são TRUE
A | B	TRUE se A ou B são TRUE
! A	Inverte o resultado

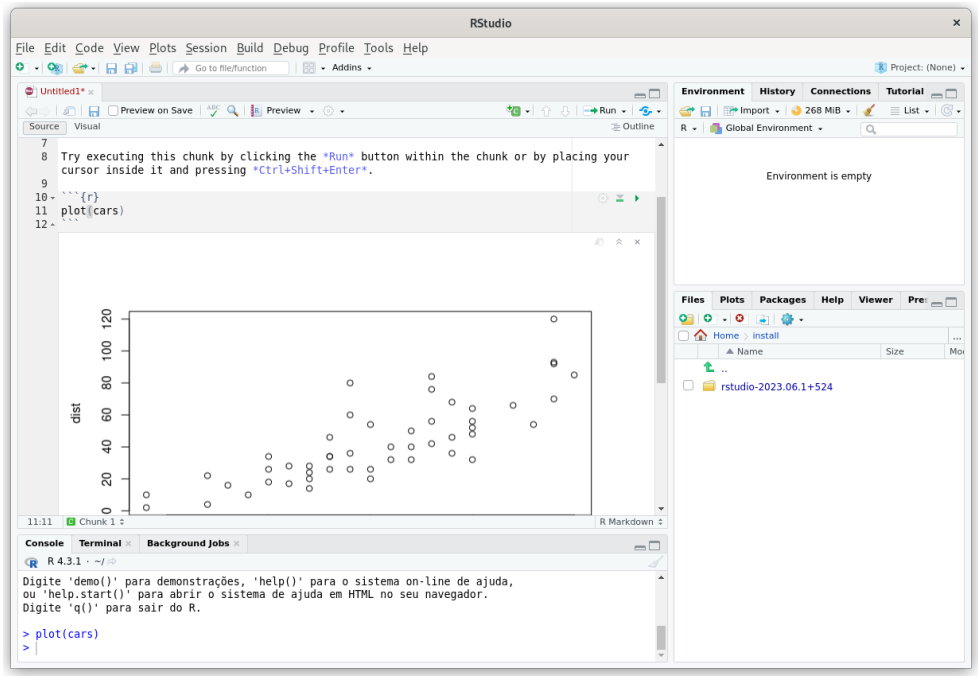
Fonte: Autor.

3.2 AMBIENTES DE PROGRAMAÇÃO

R possui o interpretador de referência para programação e disponível de forma livre para *download* em seu site oficial (https://www.r-project.org/). O interpretador sozinho possui algumas funcionalidades para análise estatística e geração de gráficos. Todavia, a relevância do *software* R está no amplo conjunto de pacotes adicionais disponíveis. Dessa forma, podemos optar por duas alternativas gratuitas de programação: RStudio e Jupyter. Ambos seguem o paradigma de *Literate Programming* por meio de um *notebook* que é um documento contendo texto e blocos de código.

O RStudio (https://www.rstudio.com/) é um ambiente integrado para R e Python que permite escrita de *notebook*, geração de gráficos, inspeção de variáveis, etc. Suas versões livres têm código-fonte aberto e disponíveis para diversas plataformas. A Figura 22 mostra o ambiente de execução.

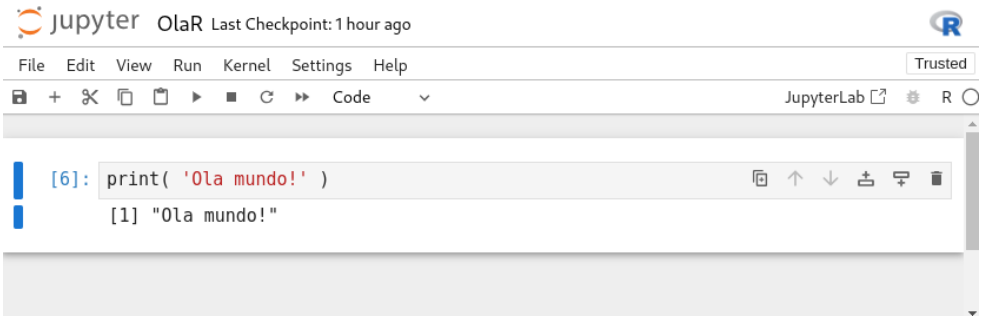
Figura 22: Ambiente integrado RStudio para R



Fonte: Autor.

O Jupyter (https://jupyter.org/) é um projeto de código fonte livre criado em 2014 do antigo projeto IPython. O projeto suporta diversas linguagens de programação tendo foco em Python, R e Julia. O *software* pode ser instalado localmente ou usados online no site oficial do Jupyter. A Figura 23 mostra a interface Web do Jypyter com um bloco de código R.

Figura 23: Jupyter *notebook* para programação R



Fonte: Autor.

3.3 VARIÁVEIS E TIPOS PRIMITIVOS

Variáveis são essencialmente nomes para valores na memória do computador como vimos anteriormente. Os nomes de variáveis em R, ou identificadores, podem conter os caracteres:

- Letras maiúsculas ou minúsculas entre "A" e "Z";
- O sublinhado '_' e ponto '.';
- Dígitos de "0" a "9" exceto no começo do nome;
- Se começa com ".", o segundo caractere do nome não pode ser dígito.

Assim sendo, nomes como "x", "celsius", "valor.total" e "_interno" são válidos enquanto que "1nome" e ".1valor" são inválidos. Espaços são permitidos, mas precisam estar entre crases como em `meu valor`. Nomes com letras maiúsculas ou minúsculas definem nomes diferentes como em "celsius" e "Celsius".

Alguns identificadores fazem parte da linguagem R, chamados **palavras reservadas**, e não podem ser usadas em outros nomes. O Quadro 13 mostra a lista completa de nomes reservados.

Quadro 13: Palavras-chave reservadas da linguagem R

`if`	`else`	`repeat`	`while`	`function`
`for`	`in`	`next`	`break`	`TRUE`
`FALSE`	`NULL`	`Inf`	`NaN`	`NA`
`NA_integer_`	`NA_real_`	`NA_complex_`	`NA_character_`	`...`

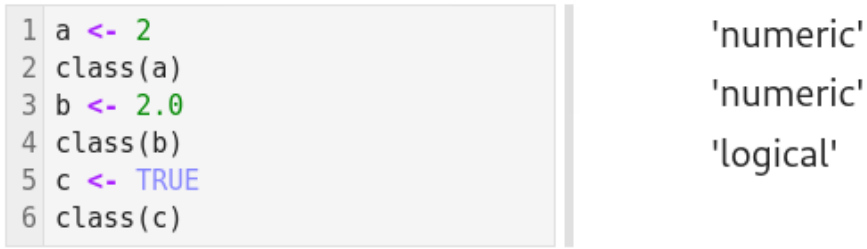
Fonte: R Documentation.

Os tipos de dados primitivos em R são:

- Número inteiro: *integer*;
- Número com ponto flutuante: *numeric*;
- Cadeia de caracteres: *character*;
- Lógico: logical que tem apenas dois valores *TRUE* (verdadeiro) e *FALSE* (falso).

R é definido como uma linguagem com tipagem dinâmica, ou seja, o tipo de uma variável é definido no momento da atribuição podendo ser alterado no decorrer da execução. A função especial class() permite mostrar qual é o tipo associado a um dado ou variável. A Figura 24 mostra alguns exemplos de tipos primitivos.

Figura 24: Código R com tipos primitivos



Fonte: Autor.

3.4 ESTRUTURAS DE DADOS

As estruturas de dados nativas do R oferecem formas de organizar os dados em alto nível sem penalidades de desempenho à execução de programas. As principais estruturas são vetores e *dataframe*.

3.4.1 Vetores

Vetores são sequências de dados de uma dimensão que podem armazenar caracteres, números e valores lógicos sem tamanho fixo de elementos. Os dados não podem ser de tipos diferentes, senão o R converte os outros elementos para forçar o mesmo tipo de dados entre todos. Vetores também são mutáveis, ou seja, permitem alterações dos elementos por indexação. Um vetor pode ser criado por meio da função "*c()*" e seus elementos dentro separados por vírgula. Uma lista de números "*c(1, 2, 3, 4)*" tem quatro elementos.

A Figura 25 ilustra alguns exemplos de manipulação de vetores. O vetor da linha 1 contém apenas letras. O acesso a elementos por indexação usa "[]" com o número do elemento como mostrado na linha 2. Note que o primeiro elemento do vetor está na posição 1. A linha 3 mostra a seleção de elementos pelos índices, e a linha 4 mostra que podemos extrair um intervalo de elementos com o formato "início:fim". Podemos também realizar operações vetoriais como a divisão da linha 7.

Figura 25: Exemplos de métodos sobre vetores em R

```
1 v <- c('a', 'b', 'c', 'd')
2 print( v[1] )
3 print( v[c(1,3)] )
4 print( v[2:4] )
5
6 u <- c(3.1, 9.2, 5.7, 10.0)
7 w <- u / 3
8 print( w )
```

```
[1] "a"
[1] "a" "c"
[1] "b" "c" "d"
[1] 1.033333 3.066667 1.900000 3.333333
```

Fonte: Autor.

3.4.2 *Dataframe*

Um dataframe é uma tabela que suporta colunas de tipos de dados diferentes além de diversas funcionalidades para manipulação de dados. O R inclui dataframes disponíveis com a função "*data()*". Um *dataframe* pode ser criado com a função "*data.frame()*" tendo como argumento cada uma das colunas desejadas na tabela. A Figura 26 mostra um exemplo de dataframe com três colunas. Cada coluna de dados de um *dataframe* é um vetor e podemos acessar a coluna com um "$" seguido do nome da coluna como na linha 6 ou colocar crases em torno

do nome como na linha 7. As crases podem ser necessárias quando o nome da coluna possui acentos ou espaços.

Figura 26: Criação de um dataframe em R

```r
dias <- c('seg', 'ter', 'qua', 'qui', 'sex')
temp <- c(12.2, 24, 23, 24.8, 20)
chuva <- c(TRUE, TRUE, FALSE, FALSE, TRUE)

df <- data.frame(dias, temp, chuva)
df$dias
df$`temp`
df
```

'seg' · 'ter' · 'qua' · 'qui' · 'sex'

12.2 · 24 · 23 · 24.8 · 20

A data.frame: 5 × 3

dias	temp	chuva
<chr>	<dbl>	<lgl>
seg	12.2	TRUE
ter	24.0	TRUE
qua	23.0	FALSE
qui	24.8	FALSE
sex	20.0	TRUE

Fonte: Autor.

3.5 ESTRUTURAS DE DECISÃO

Uma estrutura de decisão executa um conjunto de instruções caso uma determinada condição lógica seja avaliada como verdadeira. Nesta seção, veremos as variações da estrutura de decisão SE do R "if" e SENÃO "*else*".

3.5.1 SE / IF

O teste condicional mais simples é o "*if*" que avalia uma condição lógica e executa as instruções abaixo quando esta condição é verdadeira ("*TRUE*"). A Figura 27 mostra o exemplo de comparação de dois números na linha 3. Note que o teste condicional da linha 3 é delimitado entre parêntesis. Logo abaixo, as instruções a serem executadas são delimitadas por abre e fecha chaves. Note que a linha 5 está fora do fecha chaves, portanto não está dentro da estrutura de decisão.

Figura 27: Exemplo da estrutura de decisão "*if*" em R

```
1 x <- 5
2 y <- 4
3 if( x > y ){
4     print( paste('O maior e:', x) )
5 }
6 print( 'Fim' )
```

```
[1] "O maior e: 5"
[1] "Fim"
```

Fonte: Autor.

3.5.2 SENÃO / ELSE

Haverá problemas específicos onde queremos avaliar outros testes condicionais ou mesmo executar instruções quando todas as condições lógicas resultarem em falso ("*FALSE*"). O teste condicional "*if*" pode vir após um "*else*" e será avaliado quando o teste condicional anterior resultar em falso. Caso todos os testes anteriores forem falsos, podemos usar o "*else*" que deve vir seguido de um "*if*".

A Figura 28 ilustra um exemplo de testes condicionais com "*if*/*else*". O "*else*" da linha 8 executa seu bloco de instruções quando os testes dos dois "*if*" anteriores resultam em falso.

Figura 28: Exemplo de estrutura de decisão *if/ else* em R

```
x = 5
y = 6
z = 7
if x > y:
    print( 'O maior:', x )
elif y > z:
    print( 'O maior:', y )
else:
    print( 'O maior:', z )
print('Fim')
```

```
O maior: 7
Fim
```

Fonte: Autor.

3.6 ESTRUTURAS DE REPETIÇÃO

Uma estrutura de repetição ou laço repete um conjunto de instruções por um número determinado ou não de vezes até que uma certa condição seja alcançada. A linguagem R conta com duas estruturas de repetição: *while* e for.

3.6.1 *while*

Uma estrutura de laço "*while*" repete um conjunto de instruções enquanto a condições lógica for verdadeira. A Figura 29 mostra um exemplo simples de laço onde repetimos 5x um comando para imprimir a frase "Olá mundo!". A linha 1 inicia uma variável "i" com zero para ser um contador, ou seja, um valor que determina o número de repetições executadas. A laço repete suas instruções enquanto "i" é menor que cinco na linha 2. A linha 4 incrementa o contador "i" que será testado novamente na linha 2. Note que o intervalo de valores para "i" está entre [0, 5) pois quando o valor de "i" é igual a cinco a condição será falsa e o laço será interrompido.

Figura 29: Laço de repetição *while* com repetições determinadas em R

```
1 i = 0
2 while i < 5:
3     print( "Ola mundo!" )
4     i= i + 1
```

```
Ola mundo!
Ola mundo!
Ola mundo!
Ola mundo!
Ola mundo!
```

Fonte: Autor.

Outra forma de usar laços "*while*" é quando não temos um número determinado de repetições até que a condição testada seja atendida. O exemplo da Figura 30 mostra um programa que entra em um laço enquanto a senha estiver incorreta. O programa pede a senha na linha 1 e testa se ela está incorreta solicitando novamente até o usuário digitar "elefante". Note que o número de repetições ficou indefinido até que a condição "senha correta" seja atingida.

Figura 30: Laço de repetição *while* com repetições indeterminadas em R

```
1 senha = readline( 'Digite sua senha:' )
2 while( senha != 'elefante' ){
3     senha = readline( 'Senha errada! Digite novamente:')
4 }
5 print( 'Senha correta! Bem vindo.' )
```

Fonte: Autor.

Outra forma de criar laços sem repetições determinadas é ao utilizar a instrução "*break*" que faz com que o laço seja interrompido. A Figura 31 mostra o programa anterior com critério de parada com a instrução "*break*". Note que o teste condicional do laço "*while*" tem apenas o valor "*TRUE*" o que faz dele um laço sem condição de parada. A linha 3 é responsável pela condição de parada e testa se a senha está correta. Em caso afirmativo, instrução "*break*" interrompe o laço.

Figura 31: Laço de repetição *while* com critério de parada usando *break* em R

```
1 while( TRUE ){
2     senha = readline( 'Digite sua senha:' )
3     if( senha == 'elefante' ){
4         break
5     }
6 }
7 print( 'Senha correta! Bem vindo.' )
```

Fonte: Autor.

3.6.2 *for*

Uma estrutura de laço "*for*" repete um conjunto de instruções com um número de repetições determinado. A Figura 32 mostra um exemplo simples de laço onde repetimos 5x um comando para imprimir a frase "Olá mundo!". A variável "i" logo após a palavra-chave "*for*" é chamada de índice do laço que assume cada um dos valores da sequência "*c(1:5)*" e executa o bloco de instruções para cada valor. A função "*c()*" recebe como argumento o intervalo de valores gerados.

Figura 32: Laço de repetição for com número determinado de repetições em R

```
1 for( i in c(1, 2, 3, 4, 5) ){
2     print( i )
3 }
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Fonte: Autor.

O laço da Figura 33 é semelhante ao laço anterior, mas usa-se explicitamente uma lista de elementos. Nota-se que os valores impressos de "i" são cada um dos elementos da lista de dados.

Figura 33: Laço de repetição for com vetor de elementos

```
1 for( i in c(1:5) ){
2     print( i )
3 }
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Fonte: Autor.

3.7 FUNÇÕES

Uma função é um conjunto de instruções que podem ser executados diversas vezes. A função pode retornar um valor e receber valores de entrada como parâmetros. O código da Figura 34 mostra dois exemplos de funções simples. A primeira "*ola()*" não tem parâmetros nem retorno e apenas imprime a mensagem "Ola mundo!". A segunda função "*soma()*" tem dois parâmetros e retorna a soma de ambos. Importante destacar que a segunda função recebe dois valores numéricos para efetuar a soma.

Figura 34: Uso de funções em R

```
1  def ola():
2      print('Ola mundo!')
3
4  def soma(a, b):
5      return a + b
6
7  ola()
8  c = soma(2, 3)
9  print( c )
10
```

```
Ola mundo!
5
```

Fonte: Autor

3.8 DEPURAÇÃO

R disponibiliza funções para depuração do programa em execução onde explicaremos quatro delas:

- *debug()* - permite que o usuário acompanhe a execução uma função linha por linha;
- *traceback()* - mostra todas as chamadas de funções até o ponto de erro;
- *browser()* - pausa a execução do programa e permite ao usuário inspecionar o estado de execução;
- *trace()* - permite inserir pontos de depuração em qualquer função.

A Figura 35 mostra um exemplo do método "*debug()*" para a função "*soma*" na linha 7. Note que a saída mostra informações sobre cada linha executada pelo programa dentro da função "*soma*".

Figura 35: Depuração de programas R com o método debug

```
1 soma <- function(a, b){
2     print( 'Ola mundo!' )
3     c = a + b
4     return( c )
5 }
6
7 debug(soma)
8 c <- soma(2, 3)
```

```
debugging in: soma(2, 3)
debug em <text>#1: {
    print("Ola mundo!")
    c = a + b
    return(c)
}
debug em <text>#2: print("Ola mundo!")
[1] "Ola mundo!"
debug em <text>#3: c = a + b
debug em <text>#4: return(c)
exiting from: soma(2, 3)
```

Fonte: Autor.

ATIVIDADES | UNIDADE 3

1. Escreva um programa em R que leia como valor um ano qualquer, informado pelo usuário, e determine se esse ano é bissexto ou não. Um ano é bissexto se: é múltiplo de 400, ou é múltiplo de 4 e não é múltiplo de 100.

2. Desenvolva um programa R que inicia com um *dataframe* contendo dados sobre alunos e faça o cálculo da média final de todos. O *dataframe* deve conter as colunas 'matricula', 'nome', 'nota1' e 'nota2'.

3. Utilize um laço for para calcular a soma dos 100 primeiros termos da série harmônica: 1/1 + 1/2 + 1/3 + 1/4 + 1/5 + .

4. Faça um programa R que mostre a tabuada do 9, de 1 a 10.

4

SITUAÇÕES-PROBLEMA DE PROGRAMAÇÃO EM CIÊNCIA DE DADOS

INTRODUÇÃO

A ciência de dados tem sido cada vez mais relevante em diversas áreas especialmente na administração pública. A importância de coletar, interpretar e divulgar diferentes tipos de informação demanda habilidades cada vez mais especializadas em ferramentas para dados. Os dados podem qualificar decisões e corroborar políticas públicas em diferentes órgãos públicos e setores. A ciência de dados pode ser um instrumento para a concepção, desenvolvimento, melhoria e avaliação de serviços e políticas públicas, seja na esfera municipal, estadual ou federal.

Não obstante, diversas ferramentas e linguagens para dados tem ganhado popularidade nos últimos anos. Python cresceu em popularidade devido ao seu uso em ciência de dados e inteligência artificial, principalmente por ser uma linguagem multi-paradigma completa com uma base de usuários cada vez maior. A linguagem R tem crescido devido ao seu foco em análise estatística, suporte a gráficos e grande variedade de pacotes disponíveis.

Este capítulo introduz a análise de dados com as linguagens Python e R assim como exemplos introdutórios de análise. Exemplos básicos de leitura de arquivos de entrada assim como extração de informações estatísticas são demonstrados. Nosso estudo de caso é sobre os dados públicos do Instituto Nacional do Seguro Social (INSS) obtidos do Portal de Dados Abertos do governo federal. O objetivo é aplicar os conceitos demostrados nos capítulos anteriores de programação e demonstrar algumas noções sobre como analisar dados em ambas as linguagens.

4.1 INTRODUÇÃO A ANÁLISE DE DADOS

O objetivo desta seção é mostrar de forma resumida as principais funcionalidades dos pacotes de análise de dados tanto de Python quanto de R. Ambos os exemplos usam um conjunto de dados de exemplo baseado em informações de passageiros do Titanic.

4.1.1 Introdução a análise em Python

Usaremos como exemplo de análise de dados o pacote **Pandas** que suporta estruturas de dados flexíveis para manipulação e análise (MUELLER; MASSARON, 2020). Os principais usos de Pandas são: dados em tabelas de bancos de dados ou planilhas; séries temporais ordenadas ou não; matrizes homogêneas ou não; e conjuntos de dados estatísticos.

O Pandas suporta basicamente dois tipos de estruturas de dados:

- **Series**: dados de uma dimensão (1D)
- **DataFrame**: tabela de duas dimensões (2D)

INTERATIVIDADE: Acesso em: https://pandas.pydata.org/

A Figura 36 mostra o exemplo de uso do Pandas para criação de uma Series com números, e a Figura 37 mostra a criação de um *DataFrame* com nome, idade e sexo. O comando para importar pacotes em Python é o "*import*" como mostrado na linha 1 da Figura 36.

Figura 36: Exemplo de criação de dados 1D com Pandas

```
1 import pandas as pd
2
3 s = pd.Series([1, 3, 5, 6, 8])
4 s
```

```
[1]: 0    1
     1    3
     2    5
     3    6
     4    8
     dtype: int64
```

Fonte: Autor.

Figura 37: Exemplo da criação de um *dataframe* com Pandas

```
1 df = pd.DataFrame({
2     'Nome': [ 'Joao Pereira Cavalcanti',
3               'Ana Pinto Rocha',
4               'Maria Silva Campos',
5               'Ana Novaes Alves'],
6     'Idade': [22, 35, 58, 38],
7     'Sexo': ['M', 'F', 'F', 'F']}
8   )
9 df
10
```

[11]:

	Nome	Idade	Sexo
0	Joao Pereira Cavalcanti	22	M
1	Ana Pinto Rocha	35	F
2	Maria Silva Campos	58	F
3	Ana Novaes Alves	38	F

Fonte: Autor.

Pandas possui funcionalidades de leitura de dados tanto localmente quanto remoto em diversos formatos. A Figura 38 mostra a leitura de um arquivo CSV remoto em um *DataFrame* e posterior resumo dos dados com a função "*describe()*". É possível observar informações sobre as colunas lidas e dado estatísticos sobre os valores tais como média, desvio padrão, mínimo e máximo, etc.

Figura 38: Leitura de um arquivo CSV com Pandas

```
import pandas as pd

titanic = pd.read_csv("https://raw.githubusercontent.com/pandas-dev/pandas/master/doc/data/titanic.csv")
titanic.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Fonte: Autor.

Podemos selecionar apenas um subconjunto de colunas da tabela maior como no exemplo da Figura 39 linha 1 onde seleciona as colunas "*Age*" e "*Sex*" e mostra as primeiras cinco linhas com a função "*head()*".

Figura 39: Subconjunto de dados de um *dataframe* Pandas

```
idade_sexo = titanic[['Age', 'Sex']]
idade_sexo.head()
```

[3]:

	Age	Sex
0	22.0	male
1	38.0	female
2	26.0	female
3	35.0	female
4	35.0	male

Fonte: Autor.

A consulta sobre dados estatísticos pode ser feita por coluna como na Figura 40 onde as informações de valores mínimo, máximo e média são mostrados para a coluna "*Age*".

Figura 40: Informações estatísticas sobre a coluna "*Age*" com Pandas

```
print( titanic['Age'].min()  )
print( titanic['Age'].max()  )
print( titanic['Age'].mean() )
```

```
0.42
80.0
29.69911764705882
```

Fonte: Autor.

Também podemos aplicar testes relacionais e lógicos para selecionar determinadas linhas da tabela como no exemplo da Figura 41. O primeiro exemplo cria a tabela "acima_35" com todos os passageiros que tinham idade acima de 35 anos resultando em 217 linhas. Depois, a outra seleção cria a tabela "acima_M_35" com todos os passageiros acima de 35 anos e do sexo masculino resultando em 144 linhas. Note que em ambos os casos usamos operadores relacionados para efeitos de comparação e um operador lógico no segundo exemplo.

Figura 41: Teste relacional aplicado a um *dataframe* do Pandas

```
acima_35 = titanic[ titanic['Age'] > 35 ]
len( acima_35 )
```

```
217
```

```
acima_M_35 = titanic[ (titanic['Age'] > 35) & (titanic['Sex'] == 'male') ]
len( acima_M_35 )
```

```
144
```

Fonte: Autor.

Pandas possui operadores lógicos próprios quando aplicados a estruturas de dados do próprio pacote como ilustrado no Quadro 14.

Quadro 14: Operadores lógicos do Pandas

SÍMBOLO	OPERAÇÃO
A & B	Operador lógico AND
A | B	Operador lógico OR
~ A	Operador lógico NOT

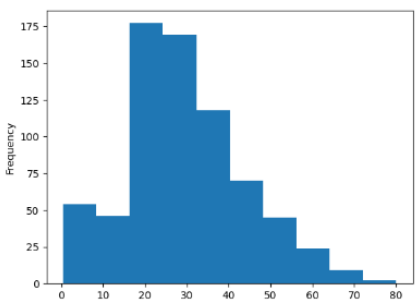
Fonte: Autor.

A geração de gráficos é integrada e usa o pacote "*matplotlib*" como no exemplo da Figura 42. O gráfico exemplo é um histograma baseado na idade dos passageiros.

Figura 42: Gráfico de histograma com Pandas

```
1 titanic['Age'].plot.hist()
```

```
[10]: <Axes: ylabel='Frequency'>
```



Fonte: Autor.

4.1.2 Introdução a análise em R

O *software* R possui funcionalidades de análise de dados tal como tabela de dados e métodos de leitura de arquivos locais e remotos (LACERDA *et al*, 2021). A Figura 43 mostra o exemplo da leitura de dados remoto e um resumo dos dados com a função "*summary()*".

Figura 43: Leitura de dados e resumo em R

```
1 titanic <- read.csv('https://raw.githubusercontent.com/pandas-dev/pandas/master/doc/data/titanic.csv')
2 summary(titanic)
```

```
  PassengerId       Survived          Pclass          Name
 Min.   :  1.0   Min.   :0.0000   Min.   :1.000   Length:891
 1st Qu.:223.5   1st Qu.:0.0000   1st Qu.:2.000   Class :character
 Median :446.0   Median :0.0000   Median :3.000   Mode  :character
 Mean   :446.0   Mean   :0.3838   Mean   :2.309
 3rd Qu.:668.5   3rd Qu.:1.0000   3rd Qu.:3.000
 Max.   :891.0   Max.   :1.0000   Max.   :3.000

     Sex                 Age            SibSp           Parch
 Length:891         Min.   : 0.42   Min.   :0.000   Min.   :0.0000
 Class :character   1st Qu.:20.12   1st Qu.:0.000   1st Qu.:0.0000
 Mode  :character   Median :28.00   Median :0.000   Median :0.0000
                    Mean   :29.70   Mean   :0.523   Mean   :0.3816
                    3rd Qu.:38.00   3rd Qu.:1.000   3rd Qu.:0.0000
                    Max.   :80.00   Max.   :8.000   Max.   :6.0000
                    NA's   :177
    Ticket               Fare           Cabin              Embarked
 Length:891         Min.   :  0.00   Length:891         Length:891
 Class :character   1st Qu.:  7.91   Class :character   Class :character
 Mode  :character   Median : 14.45   Mode  :character   Mode  :character
                    Mean   : 32.20
                    3rd Qu.: 31.00
                    Max.   :512.33
```

Fonte: Autor.

Podemos selecionar apenas um subconjunto de colunas da tabela maior como no exemplo da Figura 44 linha 1 onde escrevemos uma vírgula "," para selecionar todas as linhas e uma lista de colunas após a vírgula "*Age*" e "*Sex*". As primeiras linhas do *dataframe* "idade_sexo" são impressas com a função "*head()*".

Figura 44: Título da figura

```
1 idade_sexo = titanic[,c('Age', 'Sex')]
2 head( idade_sexo )
```

A data.frame: 6 × 2

	Age	Sex
	<dbl>	<chr>
1	22	male
2	38	female
3	26	female
4	35	female
5	35	male
6	NA	male

Fonte: Autor.

A consulta sobre dados estatísticos pode ser feita por coluna como na Figura 45 onde as informações de valores mínimo, máximo e média são mostrados para a coluna "*Age*". Cada função tem a opção "*na.rm*" para remover as linhas com dados inexistentes.

Figura 45: Título da figura

```r
min( titanic$Age, na.rm = TRUE )
max( titanic$Age, na.rm = TRUE )
mean( titanic$Age, na.rm = TRUE )
```

```
0.42
80
29.6991176470588
```

Fonte: Autor.

Também podemos aplicar testes relacionais e lógicos para selecionar determinadas linhas da tabela como no exemplo da Figura 46. O primeiro exemplo cria a tabela "acima_35" com todos os passageiros que tinham idade acima de 35 anos. Depois, a outra seleção cria a tabela "acima_M_35" com todos os passageiros acima de 35 anos e do sexo masculino. Note que em ambos os casos usamos operadores relacionados para efeitos de comparação e um operador lógico no segundo exemplo.

Figura 46: Título da figura

```r
acima_35 <- titanic[ titanic$Age > 35, ]
nrow( na.omit(acima_35) )

```

```
217
```

```r
acima_M_35 <- titanic[ (titanic$Age > 35) & (titanic$Sex == 'male'),]
nrow( na.omit(acima_M_35) )
```

```
144
```

Fonte: Autor.

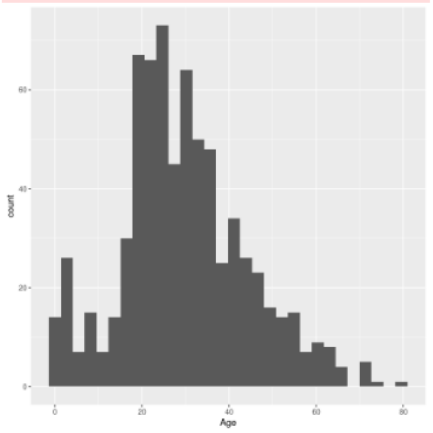
A geração de gráficos depende do pacote "*ggplot2*" como no exemplo da Figura 47. Primeiro, importamos o pacote com a função "require()" que recebe como argumento o nome do pacote. O gráfico exemplo é um histograma baseado na idade dos passageiros. Usamos a opção "na.rm" para "*TRUE*" para remover as linhas com dados inexistentes.

Figura 47: Exemplo de gráfico de histograma em R

```r
require(ggplot2)

ggplot(titanic, aes(Age)) +
    geom_histogram(na.rm = TRUE)
```

```
`stat_bin()` using `bins = 30`. Pick
better value with `binwidth`.
```



Fonte: Autor.

4.2 ESTUDO DE CASO

O estudo de caso mostra uma análise simples de dados abertos do Instituto Nacional do Seguro Social (INSS) relativos aos benefícios concedidos em maio de 2023. Os dados estão disponíveis publicamente no Portal de Dados Abertos (https://dados.gov.br/) em três formatos: CSV, XML e JSON.

4.2.1 Estudo de caso em Python

A Figura 48 mostra o exemplo de carregamento dos dados com Pandas do arquivo de entrada. Usamos opções adicionais devido ao formato da planilha: "*sep*" para usar o separador ';' das colunas, "*encoding*" para definir a codificação de caracteres como acentos. A saída do código mostra as colunas lidas de um total de 508 mil linhas.

Figura 48: Leitura da base dados em R

```
inss_maio_2023 = pd.read_csv('D.SDA.PDA.001.CON.202305.csv',
                     sep=';', encoding='latin_1')

inss_maio_2023.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 508583 entries, 0 to 508582
Data columns (total 13 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   Competência concessão  508583 non-null  int64
 1   Espécie                508583 non-null  object
 2   CID                    508583 non-null  object
 3   Despacho               508583 non-null  object
 4   Dt Nascimento          508583 non-null  object
 5   Sexo.                  508583 non-null  object
 6   Clientela              508583 non-null  object
 7   Tipo de Cálculo        508583 non-null  object
 8   Mun Resid              508583 non-null  object
 9   Vínculo dependentes    508583 non-null  object
 10  Forma Filiação         508583 non-null  object
 11  UF                     508583 non-null  object
 12  Qt SM RMI              508583 non-null  object
dtypes: int64(1), object(12)
memory usage: 50.4+ MB
```

Fonte: Autor.

Nosso primeiro exemplo é a análise de pedidos de auxílio-doença entre as áreas urbanas e rural na Figura 49. A informação sobre áreas está na coluna "Clientela". Primeiro, selecionamos todas as linhas que possuem "Doenca" na coluna "Espécie" na linha 1 e o resultado é salvo no *DataFrame* "inss_doenca".

Na linha 2 agrupamos os dados pela coluna "Clientela" com a função "*groupby*" e contamos as ocorrências de cada valor com "*count()*".

Figura 49: Análise de pedidos de auxílio doença com Pandas

```
1 inss_doenca = inss_maio_2023[inss_maio_2023['Espécie'].str.contains('Doenca')]
2 inss_doenca.groupby(['Clientela'])['Clientela'].count()
```

```
Clientela
Rural        14767
Urbano      183614
Name: Clientela, dtype: int64
```

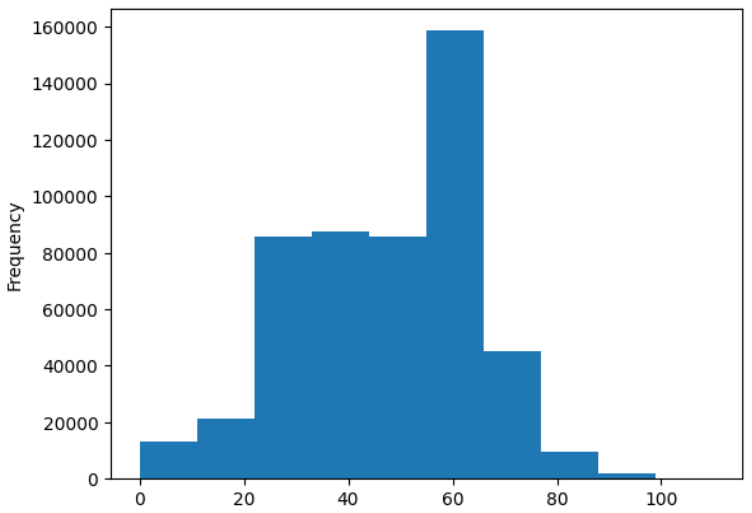
Fonte: Autor.

A Figura 50 mostra a segunda análise sobre a idade dos beneficiários baseada na data de nascimento. Para tanto, precisamos usar a coluna "Dt Nascimento" para converter de *string* para data e calcular a idade com relação a data atual. A linha 8 usa a função "*apply()*" que irá aplicar a função "*calcula_idade*" em todas as linhas da coluna 'Dt Nascimento' e armazena os resultados em uma nova coluna 'Idade'.

Figura 50: Análise de idade em beneficiários do INSS com Python

```
1 from datetime import datetime, date
2
3 def calcula_idade( nascimento ):
4     nascimento = datetime.strptime(nascimento, '%d/%m/%Y').date()
5     hoje = date.today()
6     return hoje.year - nascimento.year - (hoje.month < nascimento.month)
7
8 inss_maio_2023['Idade'] = inss_maio_2023['Dt Nascimento'].apply( calcula_idade )
9 inss_maio_2023['Idade'].plot.hist()
```

```
<Axes: ylabel='Frequency'>
```



Fonte: Autor.

A função "*calcula_idade*" da linha 3 converte a data de nascimento para data na linha 4, determina a data atual na linha 5, e calcula a diferença na linha 6. O

teste da linha 6 "*hoje.month<nascimento.month*" determina se a pessoa já faz aniversário no ano atual para subtrair um ano caso contrário.

Por fim, os dados geram o gráfico de histograma na linha 9 onde podemos observar que a grande maioria dos pedidos de benefícios são de pessoas de 60 anos.

4.2.2 Estudo de caso em R

A Figura 51 repete a análise mostrada anteriormente agora feita na linguagem R. Usamos opções adicionais devido ao formato da planilha: "*sep*" para usar o separador ';' das colunas, "*fileEncoding*" para definir a codificação de caracteres como acentos. A saída do código mostra as colunas lidas de um total de 508 mil linhas.

Figura 51: Leitura da base dados em R

```
1 inss_maio_2023 <- read.csv('D.SDA.PDA.001.CON.202305.csv', sep=';', fileEncoding="latin1")
2 summary(inss_maio_2023)
```

```
 Competência.concessão   Espécie              CID              Despacho
 Min.   :202305        Length:508583      Length:508583      Length:508583
 1st Qu.:202305        Class :character   Class :character   Class :character
 Median :202305        Mode  :character   Mode  :character   Mode  :character
 Mean   :202305
 3rd Qu.:202305
 Max.   :202305
 Dt.Nascimento         Sexo.             Clientela         Tipo.de.Cálculo
 Length:508583      Length:508583      Length:508583      Length:508583
 Class :character   Class :character   Class :character   Class :character
 Mode  :character   Mode  :character   Mode  :character   Mode  :character


  Mun.Resid         Vínculo.dependentes Forma.Filiação          UF
 Length:508583      Length:508583       Length:508583      Length:508583
 Class :character   Class :character    Class :character   Class :character
 Mode  :character   Mode  :character    Mode  :character   Mode  :character


  Qt.SM.RMI
 Length:508583
 Class :character
 Mode  :character
```

Fonte: Autor.

A análise de pedidos de auxílio-doença entre as áreas urbanas e rural é mostrada na Figura 52. A informação sobre áreas está na coluna "Clientela". Esta análise faz uso de um pacote adicional chamado "*dplyr*" que importa uma gramática de análise e manipulação de dados em R. Primeiro, selecionamos todas as linhas que possuem "Doenca" na coluna "Espécie" na linha 2 e o resultado é salvo no DataFrame "inss_doenca". Note que acessamos a coluna com crase em "`Espécie`" porque nomes de colunas com espaços ou acentos precisam ser escritos nesse formato. Na linha 3, usamos a gramática do pacote "*dplyr*" para agrupar os dados pela coluna "Clientela" com a função "*group_by*" e contar as ocorrências de cada valor com "*n()*" dentro da função "*summarise()*".

Figura 52: Análise de pedidos de auxílio-doença em R

```r
require(dplyr)
inss_doenca <- inss_maio_2023[grep('Doenca', inss_maio_2023$`Espécie`),]
inss_doenca %>%
    group_by(Clientela) %>%
    summarise(
        n = n()
    )
```

A tibble: 2 × 2

Clientela	n
<chr>	<int>
Rural	14767
Urbano	183614

Fonte: Autor.

A Figura 53 mostra a segunda análise sobre a idade dos beneficiários baseada na data de nascimento. Para tanto, precisamos usar a coluna "Dt.Nascimento" para converter de *string* para data e calcular a idade com relação a data atual. Neste exemplo importamos mais dois pacotes R "*lubridate*" para funções com datas e "*ggplot2*" para gráficos. A linha 4 usa a conversão "*as.Date()*" para converter a coluna "Dt.Nascimento" em data. A linha 5 calcula a diferença entre os anos de cada data usando funções do pacote "*lubridate*". A linha 7 gera o gráfico de histograma a partir da coluna "Idade". O gráfico gerado é semelhante ao anterior gerado com Pandas com esperado.

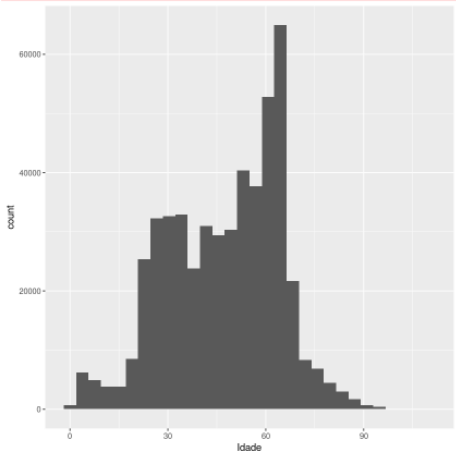
Figura 53: Análise de idade em beneficiários do INSS com R

```r
require(lubridate)
require(ggplot2)

inss_maio_2023$`Nascimento` <- as.Date(inss_maio_2023$`Dt.Nascimento`, format='%d/%m/%Y')
inss_maio_2023$Idade <- year(Sys.Date()) - year(inss_maio_2023$`Nascimento`)

ggplot(inss_maio_2023, aes(Idade)) +
    geom_histogram(na.rm = TRUE)
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Fonte: Autor.

ATIVIDADES | UNIDADE 4

1. Baseado no exemplo de análise de beneficiários do INSS da Seção 4.2, compare o número de benefícios rural e urbano de maio de 2023 com outros meses do ano de 2023.

2. Ainda sobre a análise do INSS, faça uma análise de quantos benefícios foram concedidos por estado usando a coluna "UF".

3. Use a base de dados abaixo para fazer um gráfico da qualidade do ar de três estações de medição.

   https://raw.githubusercontent.com/pandas-dev/pandas/master/doc/data/air_quality_no2.csv

CONSIDERAÇÕES FINAIS

Este livro apresentou os conceitos básicos de algoritmos e programação, assim com as linguagens de programação Python e R. As principais características das linguagens de programação foram apresentadas assim como uma breve apresentação da análise de dados por meio de suas bibliotecas. O estudo de caso para análise de dados usou uma base de dados pública para leitura, análise e apresentação de resultados por meio de gráficos.

O estudo de algoritmos e programação para ciência de dados é fundamental para analisar dados de forma consistente. Somente através de técnicas de programação podemos tratar e analisar bases de dados com centenas de milhares de linhas/colunas e criar relações essas informações a fim de extrair conhecimento. A tomada de decisões para cargos de gestão dependerá essencialmente da coleta e compreensão de dados em um futuro próximo.

Este material propôs exercícios básicos a fim de introduzir conceitos de programação e análise de dados. Você deve praticar com outras bases de dados e explorar as infinitas possibilidades que a ciência de dados pode nos oferecer.

REFERÊNCIAS

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. **Introduction to Algorithms**. 3. ed. USA: The MIT Press, 2009.

LACERDA, P. S. P. de; PEREIRA, M. A.; LENZ, M. L.; SILVA, M. S.; MARIANO, D. C. B.; ALVES, N. S. R.; NETO, R. M. **Programação em Big Data com R**. Editora Grupo A, 2021. ISBN 9786556901091.

LUTZ, M. **Learning Python**. 5. ed. EUA: O'Reilly, 2013.

MENÉNDEZ, A. **Simplificando Algoritmos**. 1. ed. Editora Grupo GEN, 2023. E-book. ISBN 9788521638339.

MUELLER, J P.; MASSARON, L. **Python Para Data Science Para Leigos**. Editora Alta Books, 2020. ISBN 9786555201512.

PIVA J.R., D. **Algoritmos e Programação de Computadores**. Editora Grupo GEN, 2019. E-book. ISBN 9788595150508.

SEVERANCE, C. R. **Python Para Todos: Explorando Dados com Python 3**. 2016. Disponível em: https://www.py4e.com/book Acesso em: out. 2023.

VERNABLES, W. N.; SMITH, D. M.; TEAM, R. C. **Thismanual isforR, version 4.3.2**. 2023. Disponível em: <https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf> Acesso em nov. 2023.

ZELLE, J. M. **Python programming: an introduction to computer science**. 3. ed. EUA: Franklin, Beedle & Associates Inc., 2016.

SOBRE O AUTOR

JOÃO VICENTE FERREIRA LIMA

O professor João Vicente Ferreira Lima possui doutorado em co-tutela entre a Université de Grenoble e Universidade Federal do Rio Grande do Sul (2014) na área de Processamento de Alto Desempenho, mestrado em Computação pela Universidade Federal do Rio Grande do Sul (2009) e graduação em Ciência da Computação pela Universidade Federal de Santa Maria (2006). Atualmente é docente do Departamento de Linguagens e Sistemas de Computação (DLSC) da Universidade Federal de Santa Maria (UFSM). Tem experiência na área de Ciência da Computação, com ênfase em Processamento paralelo de alto desempenho, atuando principalmente nos seguintes temas: processamento de alto desempenho, programação paralela, linguagens de programação, sistemas distribuídos e computação em nuvem.