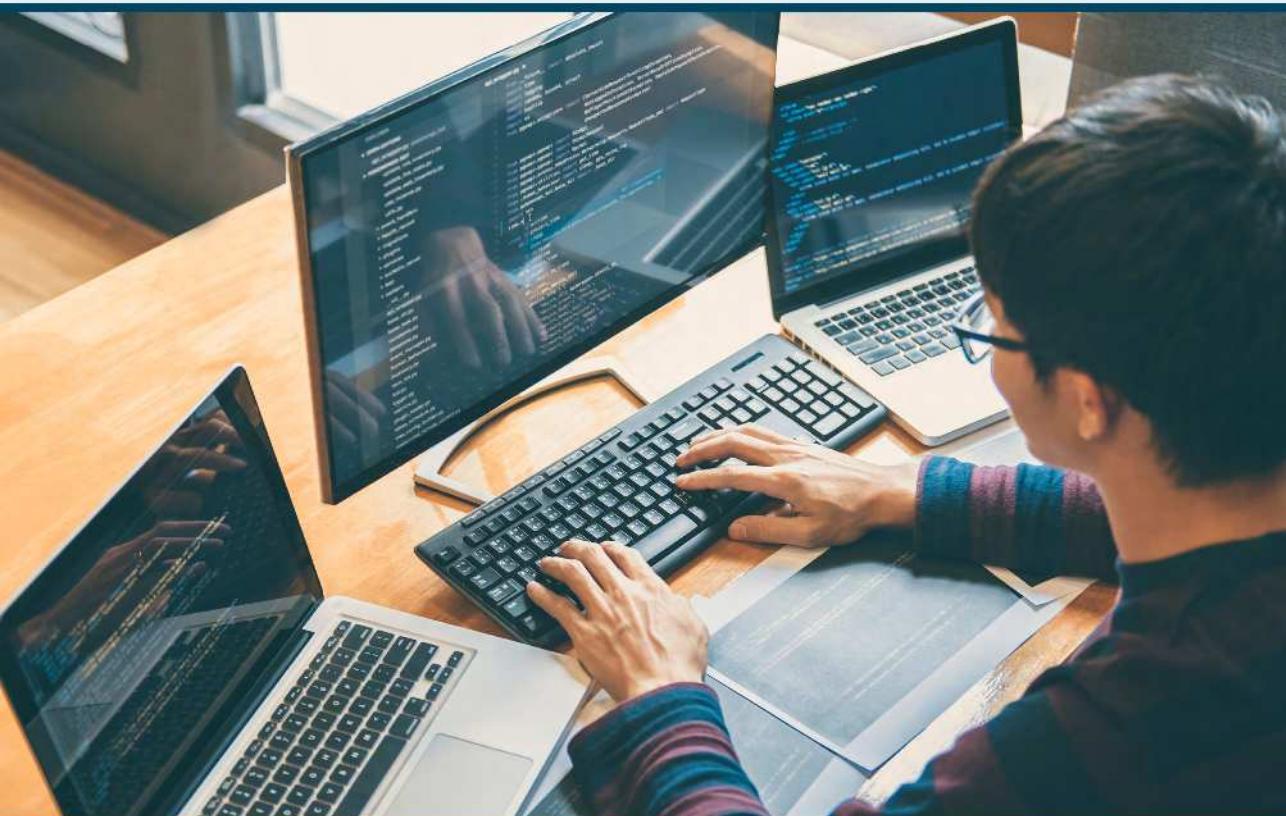


LÓGICA DE PROGRAMAÇÃO E ALGORITMOS



PROFESSOR

Me. Pietro Martins de Oliveira



ACESSO AQUI
O SEU LIVRO
NA VERSÃO
DIGITAL!

DIREÇÃO UNICESUMAR

Reitor Wilson de Matos Silva **Vice-Reitor** Wilson de Matos Silva Filho **Pró-Reitor de Administração** Wilson de Matos Silva Filho **Pró-Reitor Executivo de EAD** William Victor Kendrick de Matos Silva **Pró-Reitor de Ensino de EAD** Janes Fidélis Tomelin **Presidente da Mantenedora** Cláudio Ferdinandi

NEAD - NÚCLEO DE EDUCAÇÃO A DISTÂNCIA

Diretoria Executiva Chrystiano Mincoff, James Prestes, Tiago Stachon **Diretoria de Design Educacional** Débora Leite **Diretoria de Graduação e Pós-graduação** Kátia Coelho **Diretoria de Cursos Híbridos** Fabricio Ricardo Lazilha **Diretoria de Permanência** Leonardo Spaine **Head de Curadoria e Inovação** Tania Cristiane Yoshiie Fukushima **Head de Produção de Conteúdo** Franklin Portela Correia **Gerência de Contratos e Operações** Jislaine Cristina da Silva **Gerência de Produção de Conteúdo** Diogo Ribeiro Garcia **Gerência de Projetos Especiais** Daniel Fuverki Hey **Supervisora de Projetos Especiais** Yasminn Talyta Tavares Zagonel **Supervisora de Produção de Conteúdo** Daniele C. Correia

FICHA CATALOGRÁFICA

Coordenador(a) de Conteúdo

Flavia Lumi Matuzawa

Projeto Gráfico e Capa

Arthur Cantareli, Jhonny Coelho
e Thayla Guimarães

Editoração

Lavínia da Silva Santos
Matheus Silva de Souza

Design Educacional

Ana Elisa Fultz Davanço Portela

Revisão Textual

Ariane Andrade Fabreti

Ilustração

Bruno C. Pardinho Figueiredo

Fotos

Shutterstock

C397 **CENTRO UNIVERSITÁRIO DE MARINGÁ.**

Núcleo de Educação a Distância. **OLIVEIRA**, Pietro Martins.

Lógica de Programação e Algoritmos.

Pietro Martins de Oliveira.

Maringá - PR.: UniCesumar, 2020.

184 p.

"Graduação - EaD".

1. Programação 2. Algoritmos 3. Python. EaD. I. Título.

CDD - 22 ed. 005.1

CIP - NBR 12899 - AACR/2

ISBN 978-65-5615-318-6

Bibliotecário: João Vivaldo de Souza CRB- 9-1679



NEAD - Núcleo de Educação a Distância

Av. Guedner, 1610, Bloco 4 - Jd. Aclimação - Cep 87050-900 | Maringá - Paraná

www.unicesumar.edu.br | 0800 600 6360

Neste mundo globalizado e dinâmico, nós trabalhamos com princípios éticos e profissionalismo, não somente para oferecer educação de qualidade, como, acima de tudo, gerar a conversão integral das pessoas ao conhecimento. Baseamo-nos em 4 pilares: intelectual, profissional, emocional e espiritual.

Assim, iniciamos a Unicesumar em 1990, com dois cursos de graduação e 180 alunos. Hoje, temos mais de 100 mil estudantes espalhados em todo o Brasil, nos quatro campi presenciais (Maringá, Londrina, Curitiba e Ponta Grossa) e em mais de 500 polos de educação a distância espalhados por todos os estados do Brasil e, também, no exterior, com dezenas de cursos de graduação e pós-graduação. Por ano, produzimos e revisamos 500 livros e distribuímos mais de 500 mil exemplares. Somos reconhecidos pelo MEC como uma instituição de excelência, com IGC 4 por sete anos consecutivos e estamos entre os 10 maiores grupos educacionais do Brasil.

A rapidez do mundo moderno exige dos educadores soluções inteligentes para as necessidades de todos. Para continuar relevante, a instituição de educação precisa ter, pelo menos, três virtudes: inovação, coragem e compromisso com a qualidade. Por isso, desenvolvemos, para os cursos de Engenharia, metodologias ativas, as quais visam reunir o melhor do ensino presencial e a distância.

Reitor

Wilson de Matos Silva



Tudo isso para honrarmos a nossa missão, que é promover a educação de qualidade nas diferentes áreas do conhecimento, formando profissionais cidadãos que contribuam para o desenvolvimento de uma sociedade justa e solidária.



Me. Pietro Martins de Oliveira

Possui graduação em Engenharia de Computação pela Universidade Estadual de Ponta Grossa (2011). É mestre em Ciência da Computação na área de Visão Computacional pela Universidade Estadual de Maringá (2015). Atuou como analista de sistemas e programador nas empresas Siemens Enterprise Communications (Centro Internacional de Tecnologia de Software — CITS, 2011-2012) e Benner Saúde Maringá (2015). Experiência como docente e coordenador dos cursos de Bacharelado em Engenharia de Software, Sistemas de Informação e Engenharia de Produção.

<http://lattes.cnpq.br/1793084774574585>

LÓGICA DE PROGRAMAÇÃO E ALGORÍTMOS

Olá, prezado(a) estudante! Seja bem-vindo(a) à disciplina de Lógica de Programação e Algoritmos. Sou o professor Pietro Martins de Oliveira e, nesta disciplina, aprendere-mos, juntos, a utilizar a linguagem Python na construção de nossos programas. Para tanto, con-heceremos os conceitos fundamentais para que você seja capaz de interpretar problemas e desenvolver soluções computacionais algorítmicas. Nesta disciplina, você aprenderá os conceitos básicos da linguagem de programação Python, a qual vem se popularizando por ser uma linguagem de propósito geral e não vinculada a um hardware específico ou a qualquer outro sistema.

Apresentamos a você o livro que norteará seus estudos nesta disciplina, auxiliando no aprendizado do Python. Em cada unidade, serão apresentados exemplos de códigos-fonte. É importante que você tente executar cada um desses programas, verificando o funcionamento de cada um deles e a lógica por trás dos resultados. Apenas a leitura dos exemplos não é suficiente — o aprendizado de algoritmos requer muita prática.

Na Unidade 1, veremos um breve histórico do Python, suas características e os conceitos iniciais sobre programação, destacando as etapas para a criação de um programa bem como sua estrutura. Estudaremos os tipos de dados mais comuns em Python, bem como nomear identificadores, declarar variáveis, realizar operações de atribuição, entrada e saída de dados, tudo isso dentro de uma lógica estritamente sequencial. Conheceremos, também, as principais palavras reservadas dessa linguagem, os operadores e algumas funções pré programadas, muito úteis durante a solução de nossos problemas algorítmicos. A partir destes conteúdos, iniciaremos a construção de nossos primeiros programas em Python.

Na Unidade 2, aprenderemos a construir programas com controle de fluxo de execução, isto é, seremos capazes de impor condições para a execução de determinada instrução ou um conjunto de instruções. Trataremos de como construir programas em Python, utilizando estruturas condicionais simples e compostas. Na Unidade 3, abordaremos a construção de programas com repetição de determinado trecho de código sem a necessidade de reescrevê-lo várias vezes. Estudaremos as estruturas de repetição disponíveis na linguagem Python: estrutura `while` e estrutura `for`. Discutiremos as diferenças de cada uma delas e como utilizá-las.

APRESENTAÇÃO DA DISCIPLINA

Na Unidade 4, apresentaremos os conceitos de estruturas de dados nas quais é possível armazenar mais de um dado simultaneamente, sejam estruturas de uma única dimensão, ou com múltiplas dimensões. Aprenderemos como declará-las e manipulá-las nas operações de entrada, atribuição e saída. Trataremos de exemplos práticos que serão de grande valia no seu dia a dia enquanto programador(a).

Por fim, na Unidade 5, trabalharemos com a modularização de nossos programas, utilizando funções. Abordaremos os conceitos relacionados ao escopo de variáveis e recursividade. Estudaremos, ainda, os recursos disponíveis para que seja possível manipular arquivos em linguagem Python. Arquivos são, particularmente, úteis quando se deseja gravar dados de forma persistente em um computador, tornando tais dados disponíveis em futuras execuções de nossos programas.

Em cada unidade deste livro, você encontrará indicações de leitura complementar, as quais enriquecerão o seu conhecimento e apresentarão mais exemplos de programas em Python. Além disso, serão apresentadas atividades de auto estudo, as quais permitem que você coloque, em prática, os conhecimentos e exercícios de fixação, que apresentam exercícios resolvidos para auxiliar no aprendizado e sanar eventuais dúvidas. Desejamos a você um bom estudo!

ÍCONES



pensando juntos

Ao longo do livro, você será convidado(a) a refletir, questionar e transformar. Aproveite este momento!



explorando ideias

Neste elemento, você fará uma pausa para conhecer um pouco mais sobre o assunto em estudo e aprenderá novos conceitos.



quadro-resumo

No fim da unidade, o tema em estudo aparecerá de forma resumida para ajudar você a fixar e a memorizar melhor os conceitos aprendidos.



conceituando

Sabe aquele termo ou aquela palavra que você não conhece? Este elemento ajudará você a conceituá-lo(a) melhor da maneira mais simples.



conecte-se

Enquanto estuda, você encontrará conteúdos relevantes on-line e aprenderá de maneira interativa usando a tecnologia a seu favor.



Quando identificar o ícone de QR-CODE, utilize o aplicativo Unicesumar Experience para ter acesso aos conteúdos on-line. O download do aplicativo está disponível nas plataformas:



Google Play



App Store

CONTEÚDO

PROGRAMÁTICO

UNIDADE 01

10

ESTRUTURA
SEQUENCIAL

UNIDADE 02

43

ESTRUTURAS
CONDICIONAIS

UNIDADE 03

71

ESTRUTURAS DE
REPETIÇÃO

UNIDADE 04

92

ESTRUTURAS DE
DADOS BÁSICAS

UNIDADE 05

122

SUB-ROTIÑAS

FECHAMENTO

141

CONCLUSÃO GERAL



ESTRUTURA SEQUENCIAL

PROFESSOR

Me. Pietro Martins de Oliveira

PLANO DE ESTUDO

A seguir, apresentam-se as aulas que você estudará nesta unidade:

- A linguagem Python e os conceitos iniciais de programação
- Os identificadores e os tipos de dados
- As palavras reservadas
- As variáveis, a atribuição, os operadores e as expressões
- As funções embutidas e a entrada e saída de dados

OBJETIVOS DE APRENDIZAGEM

Compreender os conceitos iniciais de programação.

- Estudar a estrutura de um programa em Python.
- Compreender o que são e como utilizar os identificadores, os tipos de dados e as palavras reservadas.
- Estudar as variáveis, a atribuição, os operadores e as expressões.
- Compreender as funções intrinsecas e a entrada e saída de dados, por meio da construção de programas.

INTRODUÇÃO



Nesta primeira unidade, você é convidado(a) a iniciar os estudos sobre a linguagem de programação Python – uma linguagem poderosa e que tem sido utilizada em diversas aplicações, principalmente no contexto de ciência de dados.

Como toda linguagem, a Python conta com características próprias, um histórico, além de destaques, como os pontos fortes e fracos e o interpretador de código-fonte. Você verá que o interpretador de código-fonte busca converter um código em um programa executável bem como verá conceitos básicos, a estrutura básica e os elementos que compõem essa linguagem.

Como conceitos básicos, comprehenda a necessidade de saber como organizar e guardar informações e usar variáveis, quais os tipos de dados que temos à disposição nessa linguagem bem como as expressões e os operadores, as funções intrínsecas e como utilizá-los em nossos programas.

Por meio de funções relacionadas à entrada de dados, você aprenderá a obter dados dos usuários e mostrar mensagens e resultados de processamento. Além disso, você verá que o comando de atribuição possibilita modificar o conteúdo de variáveis em memória e em funções de saída de dados, que permitem o envio de mensagens e a exibição dos resultados do processamento na saída padrão do dispositivo computacional (geralmente, o monitor, o display, ou seja, a tela do computador).

O assunto é vasto e, ao final desta unidade, você terá adquirido conhecimentos importantes acerca da linguagem Python. Aqui, você conhecerá a estrutura de um programa e de variáveis, as palavras reservadas do Python, as expressões e os operadores, as funções intrínsecas, o comando de atribuição, as funções de entrada e saída de dados. Acredito que, com estes conceitos, você poderá desenvolver os seus primeiros programas em Python.

Antes de finalizar, vale ressaltar que, como todo assunto de estudo, quanto mais você explorar, buscar por conteúdos complementares e atualizados, mais desenvolverá os seus conhecimento e domínio dessa linguagem.

Vamos iniciar os estudos!



1 A LINGUAGEM PYTHON E OS CONCEITOS INICIAIS DE PROGRAMAÇÃO

Olá, prezado(a) acadêmico(a), imagino que você esteja empolgado(a) para começarmos a desenvolver nossos primeiros programas. Porém, antes disso, é interessante contextualizar alguns detalhes históricos sobre a linguagem Python bem como entender seus fundamentos básicos.

A linguagem Python é amplamente utilizada como uma linguagem de propósito geral e de alto nível. Isso quer dizer que, em tese, pode-se fazer de tudo um pouco com o Python, e que essa linguagem abstrai vários detalhes de complexidade dos computadores na hora de desenvolver programas. Inicialmente, ela foi proposta em 1991, por Guido Van Rossum, e, posteriormente, desenvolvida e mantida pela *Python Software Foundation*. Focada na legibilidade de seus códigos-fonte, essa linguagem permite que programadores expressem sua lógica de programação em pouquíssimas linhas de código.

É importante dizer que a Python é uma linguagem multiplataforma, uma vez que é possível instalar seu interpretador em diversos ambientes, como Windows, Linux e, também, nos sistemas operacionais da Apple. Uma de suas principais vantagens é que ela é uma linguagem de sintaxe bastante simples, sendo bastante indicada para iniciantes em programação. Isto não quer dizer que ela não seja utilizada por programadores experientes.

Apesar de suas vantagens, pode-se dizer que há discussões importantes sobre o desenvolvimento de aplicações de alto desempenho em Python, quando comparado com o desenvolvimento em linguagens como o C e o C++.



Toda linguagem escrita deve seguir regras, certo? Assim sendo, em programação, o termo sintaxe pode ser interpretado como “ortografia” da linguagem: quais são os operandos, onde é preciso inserir parênteses, aspas, pontuações etc.

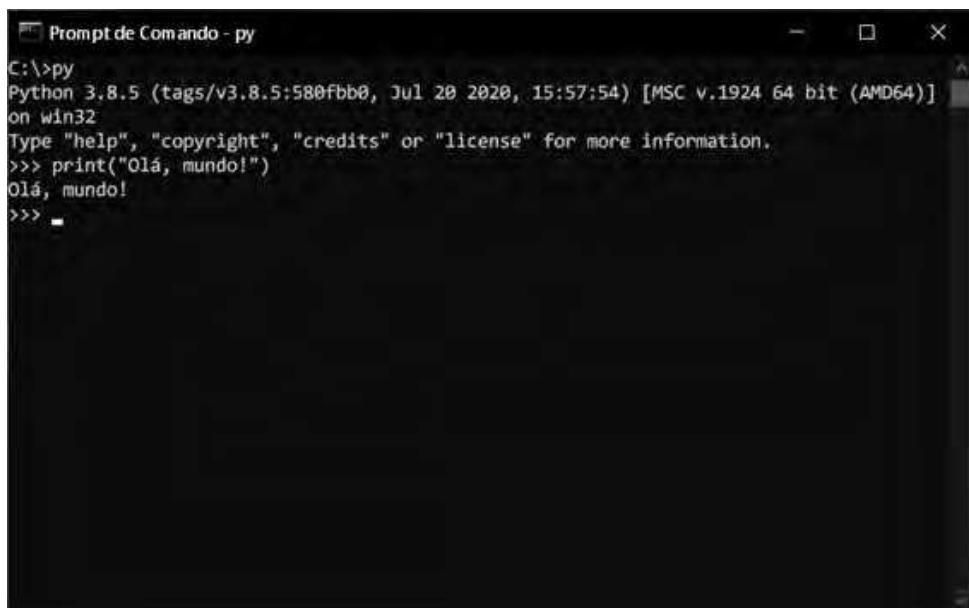
Ainda, sempre que utilizamos uma linguagem para nos comunicar é porque queremos ser entendidos em nossa mensagem. Assim, a semântica é a “lógica”, é o sentido que queremos dar a uma sentença: qual o efeito prático que um trecho de código sintaticamente correto produzirá durante a execução do respectivo código.

Falando em interpretador, precisamos entender como ele funciona na prática. O interpretador, em si, é como uma espécie de máquina virtual que interpreta o código em Python para que o programa possa ser executado pelo processador. Existem, basicamente, as linguagens compiladas e as linguagens interpretadas. Os códigos-fonte de linguagens compiladas têm o passo a passo um pouco mais complexo para a execução de programas, quando comparado às linguagens interpretadas.

Em suma, as **linguagens interpretadas** dependem de um software intermediário entre o código-fonte escrito pelo(a) desenvolvedor(a) e o sistema operacional do computador em que se deseja executar os programas, enquanto que uma **linguagem compilada** deve fazer com que o código passe por um processo de transformação para, assim, ser executado, diretamente, no hardware da máquina.

O interpretador Python possui diversas funcionalidades prontas para que nós, programadores, possamos utilizá-las durante o desenvolvimento de nossas aplicações. Sem o interpretador, o código-fonte se torna apenas um texto pleno, sem capacidade de ser executado. Uma desvantagem do uso das linguagens interpretadas é que a possibilidade de utilizar funções nativas do sistema torna-se limitada, assim como acessar recursos de hardware da mesma forma que as linguagens compiladas.

Sempre que instalamos o Python em uma máquina, temos acesso ao Ambiente Integrado de Desenvolvimento e Aprendizagem (do inglês *Integrated Development and Learning Environment* – IDLE). Assim sendo, uma vez que instalamos uma versão qualquer da linguagem em nosso computador, podemos ter acesso ao IDLE para testar comandos e scripts escritos em Python. Na figura 1, a seguir, podemos ver como o programador, neste caso, iniciou a execução do IDLE via o prompt de comando do Windows, inicialmente, invocando a sua execução por meio do comando `py`, depois, testando um comando de saída de dados na tela.



```
C:\>py
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC V.1924 64 bit (AMD64)]
on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Olá, mundo!")
Olá, mundo!
>>>
```

Figura 1 - Execução do IDLE e teste da função print(). / Fonte: o autor.

Descrição da Imagem: Na imagem, tem-se um recorte do terminal de comandos de texto do Microsoft Windows, em que aparece a execução de um programa cujo resultado é a impressão da mensagem "Olá, mundo!" na tela.

Apesar de o IDLE ser bastante útil para iniciantes que querem testar alguns comandos, não é muito prático utilizá-lo quando os programas têm muitas linhas de código ou vários arquivos fonte. Dessa forma, utilizaremos o ambiente de desenvolvimento (em inglês, *Integrated Development Environment – IDE*) *open-source* da Microsoft, o *Visual Studio Code* (VS Code). Para isso, basta fazer seu download no site oficial da Microsoft bem como instalar a extensão que permite que o Python seja integrado ao VS Code. Lembre-se: antes de mais nada, é necessário ter instalado o interpretador Python, por meio da página oficial – python.org.

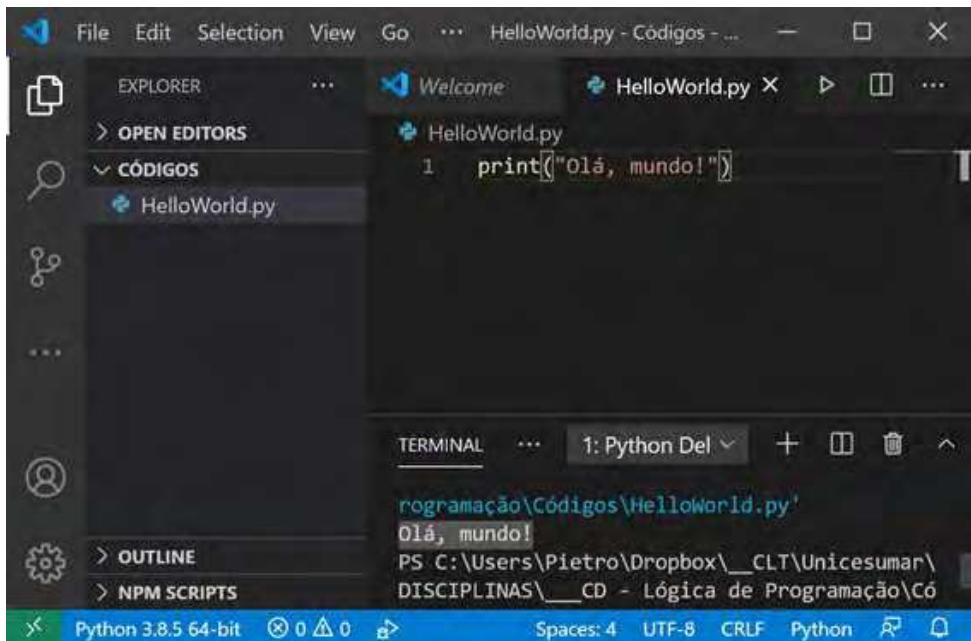


Figura 2 - *Visual Studio Code*: resultado da execução de um programa em Python
Fonte: o autor.

Descrição da Imagem: Na imagem, tem-se um recorte do ambiente de programação Visual Studio Code, em que aparece, na parte superior da figura, o código-fonte de um programa cujo propósito é a impressão da mensagem "Olá, mundo!". Já na parte inferior, mostra-se o resultado da execução do programa e a respectiva mensagem sendo impressa no terminal embutido do referido ambiente de programação.

A Figura 2 mostra como é o VS Code, visualmente. Na porção esquerda dessa IDE, temos o arquivo fonte denominado *HelloWorld.py*. Repare, portanto, que a extensão de um arquivo de código-fonte escrito em Python leva a extensão ".py", obrigatoriamente. Na porção direita do VS Code, temos o código-fonte na parte superior e o respectivo resultado da execução no terminal embutido (prompt de comando), que fica na parte de baixo. Repare como o código `print("Olá, mundo!")` da porção direita superior foi executado pelo VS Code, resultando na impressão da mensagem "Olá, mundo" na porção direita inferior, dentro do terminal embutido.

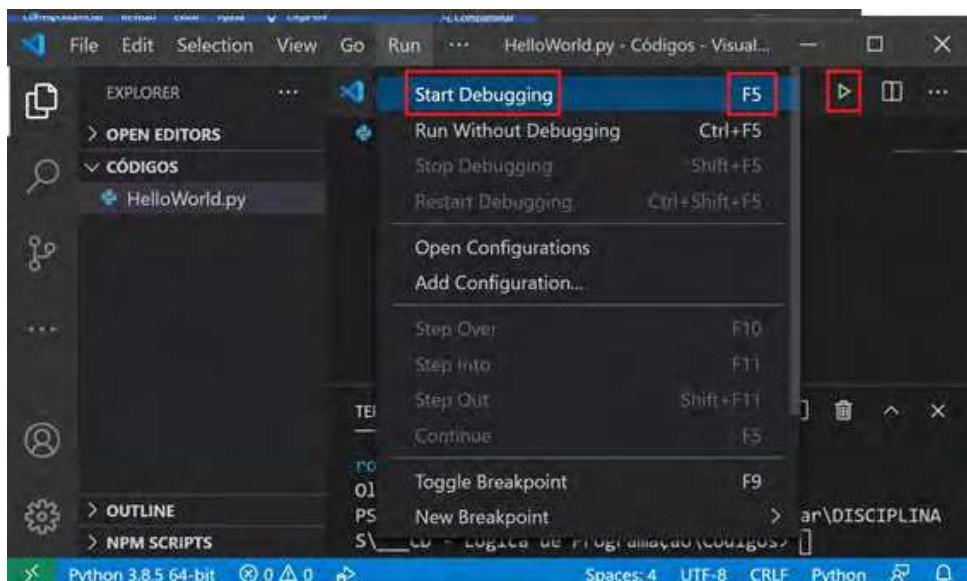


Figura 3 - Executando um programa em Python dentro do VS Code / Fonte: o autor.

Descrição da Imagem: Na imagem, são mostradas as opções existentes no Visual Studio Code para executar um código-fonte escrito na linguagem Python, a saber: apertar a tecla F5 do teclado é a alternativa mais rápida.

Uma vez que temos algum trecho de código-fonte escrito, podemos tentar executá-lo pressionando a tecla F5 do teclado, ou indo até o menu *Run > Start Debugging* do VS Code. Além disso, no canto superior direito da IDE, podemos visualizar que existe um ícone verde (similar ao ícone que conhecemos como *play* em dispositivos de som), bastando, como alternativa, clicar nele, para, assim, poder executar o código, como podemos ver na Figura 3.

É importante lembrar que a IDLE nativa do Python é limitada, por isso, utilizar o VS Code como IDE se torna mais recomendado, uma vez que podemos editar e salvar os arquivos fonte.



OS IDENTIFICADORES E OS TIPOS DE DADOS

Sempre que precisamos manipular dados, é necessário que eles fiquem registrados em algum lugar, em nosso computador. No caso de programas em Python, durante o tempo de execução, tais dados ficam registrados na memória RAM do dispositivo computacional. Para a nossa alegria, por se tratar de uma linguagem de alto nível, os programadores não precisam se preocupar com detalhes de manipulação de memória e processamento em hardware. Na realidade, nos apoiamos no conceito de variáveis.

Assim sendo, podemos dar identificadores (nomes) a essas regiões de memória em que os dados ficam registrados. Neste caso, cada identificador é conhecido, comumente, na ciência da computação, como uma **variável**.

Uma variável pode armazenar um dado específico que possui um tipo específico e, ao longo do programa, o desenvolvedor pode elaborar um conjunto de instruções lógicas que manipulam esse valor, alterando o conteúdo da respectiva variável durante a execução do programa.

É preciso entender muito bem as regras para criar um novo identificador, como um nome de variável. Cria-se uma sequência de uma ou mais letras, dígitos ou sublinhas (caractere *underline* ou *underscore* “_”). Essa sequência de caracteres alfanuméricos deve, sempre, ser iniciada com uma letra, obrigatoriamente. Além disso, deve-se saber que a linguagem Python é *case sensitive*, ou seja, o interpretador considera que letras minúsculas e maiúsculas são caracteres distintos, diferenciando caixa alta de caixa baixa. Um identificador não pode ser separado por espaços em branco.

Por fim, como veremos mais adiante, nenhum identificador criado pelo(a) programador(a) deve ser igual a alguma palavra reservada. No Quadro 1, podemos visualizar, para a criação de identificadores, exemplos corretos bem como exemplos inválidos.

Identificadores válidos	Identificadores inválidos
Aa	2 ^a
media	b@
altura2	media idade
media_idade	x*y
x36	#media
_teste	idade!

Quadro 1 - Exemplos de identificadores / Fonte: o autor.

Em Python, o tipo de uma variável é estabelecido em tempo de execução, de forma automática, pois essa é uma linguagem de tipagem dinâmica. Todavia, uma vez que uma variável é instanciada em memória, seu tipo não pode ser modificado. No Quadro 2, podemos conhecer os principais tipos de dados nativos da linguagem Python, a saber: dados numéricos sem casas decimais, dados numéricos com casas decimais, dados de texto e dados lógicos (booleanos).

Tipo de dados	Descrição
int	Valores numéricos sem casas decimais.
float	Valores numéricos com casas decimais.
str	Dados alfanuméricos em formato texto.
bool	Dados lógicos ou booleanos (True ou False).

Quadro 2 - Principais tipos primitivos de dados em Python / Fonte: o autor.



AS PALAVRAS RESERVADAS

KEYWORD

Existem alguns identificadores que são reservados à sintaxe da linguagem Python. Assim como em qualquer linguagem de programação, em Python, algumas palavras possuem significado semântico próprio, podendo ser compreendidos comandos específicos, com funcionalidades específicas.

and	except	lambda	with
as	finally	nonlocal	while
assert	false	None	yield
break	for	not	
class	from	or	
continue	global	pass	
def	if	raise	
del	import	return	
elif	in	True	
else	is	try	

Quadro 3 - Palavras reservadas da linguagem Python / Fonte: o autor.

O Quadro 3 lista as palavras reservadas do Python. Por enquanto, você não precisa se preocupar com o que cada palavra desta faz, por ora, basta saber que não devemos criar nomes de variáveis que sejam idênticos a tais palavras.

Ainda assim, vale a pena mencionar as funcionalidades de algumas das palavras recém-listadas. As palavras `True` e `False` devem ser interpretadas, literalmente, como dados lógicos do tipo `bool`, assim como um número inteiros, sem casas decimais, são interpretados como dados do tipo `int`. As palavras `if`, `else` e `elif` são especialmente importantes para quando for necessário controlar o fluxo de execução do programa, por meio de condições lógico-relacionais. Por fim, as palavras `while` e `for` são bastante úteis quando se deseja criar uma lógica de repetição em seu programa.



AS VARIÁVEIS, A ATRIBUIÇÃO, OS OPERADORES E AS EXPRESSÕES

Agora que pudemos ter contato com o conceito de identificadores e, também, com as palavras-chave da linguagem Python, podemos elaborar melhor o conceito de variáveis. Como dito anteriormente, uma variável é um espaço reservado na memória principal do computador, disponível para armazenarmos os dados de nossos programas. Vale reforçar que toda variável possui, obrigatoriamente, um nome e um tipo.

Imagine uma situação em que queremos criar um programa que controla a idade de uma ou mais pessoas. Poderíamos ter uma variável de nome e idade que, conforme um novo ano se passa, o valor da variável idade é acrescido de uma unidade. Perceba, portanto, que a variável é um espaço, em memória, cujo conteúdo, o dado, pode variar ao longo do tempo.

Em Python, sempre que quisermos criar, em memória, uma nova variável, devemos dar um nome e um conteúdo a essa variável. Esse conteúdo pode ser composto por dados numéricos, lógicos de texto, e assim por diante. Todavia existe uma forma específica de inicializar ou modificar o conteúdo a uma variável. O operador de atribuição (`=`) é o responsável por esta tarefa em nossos códigos. Observe, a seguir, a sintaxe utilizada para realizarmos atribuição:

```
<identificador_da_variável> = <conteúdo>
```

A sintaxe anterior fará com que a variável identificada, genericamente, pelo campo <identificador_da_variável> seja preenchida com os dados relacionados ao campo <conteúdo>. Ou seja, em nosso código-fonte, basta substituir o campo <identificador_da_variável> pelo nome da variável que se deseja criar/modificar, e o campo <conteúdo> pelo dado que se deseja armazenar em memória, durante a execução do programa. Veja, a seguir, no Quadro 4, alguns exemplos de atribuição de dados a variáveis quaisquer.

Exemplo

<code>x = 3</code>	O dado numérico inteiro 3 é atribuído à variável <code>x</code> .
<code>x = x + 5</code>	É atribuído à variável <code>x</code> o valor da própria variável <code>x</code> , acrescido em 5.
<code>y = 2.5</code>	O valor com casa decimal 2.5 é atribuído à variável <code>y</code> .
<code>sexo = "M"</code>	O valor "M" é atribuído à variável denominada <code>sexo</code> .
<code>nome = "Maria"</code>	O valor "Maria" é atribuído à cadeia de caracteres (str) chamada <code>nome</code> .

Quadro 4 - Exemplos de atribuição de dados a variáveis / Fonte: o autor.

Perceba que é possível utilizar uma mesma variável como parte de uma expressão matemática, como pudemos ver no exemplo `x = x + 5`. Além disso, é interessante perceber que dados do tipo str (texto alfanumérico) levam aspas duplas.

Os operadores e as expressões

No fundo, a programação de computadores em qualquer linguagem se baseia em um conjunto de comandos, que, na verdade, são definidos matematicamente, ou seja, na realidade, tudo não passa, pura e simplesmente, de matemática e lógica. Por isso, poderíamos relacionar o conceito de variáveis em memória com o conceito de variáveis do cálculo, por exemplo. Com isso, temos o poder de criar expressões e cálculos dentro de nossos programas.

A primeira categoria que veremos, aqui, são as expressões aritméticas, cujo resultado consiste em um dado numérico, seja do tipo inteiro, seja do tipo real. Dessa forma, apenas variáveis do tipo `int` ou `float` deveriam ser utilizadas nessa expressões.

O Quadro 5 apresenta os operadores aritméticos do Python, destacando suas representações e formas de uso.

Operador	Operação	Uso	Descrição
<code>+</code>	Adição	<code>x + y</code>	Soma o conteúdo de <code>x</code> com <code>y</code> .
<code>-</code>	Subtração	<code>x - y</code>	Subtrai <code>y</code> do conteúdo de <code>x</code> .
<code>*</code>	Multiplicação	<code>x * y</code>	Multiplia <code>x</code> por <code>y</code> .
<code>/</code>	Divisão	<code>x / y</code>	Divide <code>x</code> por <code>y</code> .
<code>%</code>	Resto de divisão	<code>x % y</code>	Retorna o resto da divisão inteira de <code>x</code> por <code>y</code> .
<code>**</code>	Exponenciação	<code>x ** y</code>	Eleva <code>x</code> à potência <code>y</code> .
<code>//</code>	Divisão arredondada para baixo	<code>x // y</code>	Divide <code>x</code> por <code>y</code> e arredonda o resultado para baixo.

Quadro 5 - Exemplos de operadores aritméticos em Python / Fonte: o autor.

Lembra-se de quando falamos que é possível fazer com que uma variável seja utilizada em ambos os lados de uma operação de atribuição? Pois bem, pensando nisso, existe uma forma de combinar o operador de atribuição com operadores aritméticos, para situações em que se deseja atualizar o conteúdo da variável, com base no valor que essa mesma variável carrega antes da atualização. De maneira sintética, pode-se dizer que a máquina faz uma operação aritmética e, em seguida, uma operação de atribuição.

Operador	Exemplo	Descrição
<code>+=</code>	<code>x += y</code>	Equivale a <code>x = x + y</code> .
<code>-=</code>	<code>x -= y</code>	Equivale a <code>x = x - y</code> .
<code>*=</code>	<code>x *= y</code>	Equivale a <code>x = x * y</code> .
<code>/=</code>	<code>x /= y</code>	Equivale a <code>x = x / y</code> .
<code>%=</code>	<code>x %= y</code>	Equivale a <code>x = x % y</code> .
<code>**=</code>	<code>x **= y</code>	Equivale a <code>x = x ** y</code> .
<code>//=</code>	<code>x //= y</code>	Equivale a <code>x = x // y</code> .

Quadro 6 - Operadores de atribuição combinados com operações aritméticas

Fonte: o autor.

Repare o operador `+=`, do Quadro 6, por exemplo. Imagine que, antes de executar a expressão `x += 1`, a variável `x` valia 10. Após a execução da expressão `x += 1`, a variável `x` passaria a valer 11. Ou seja, somou-se 1 ao conteúdo anterior de `x`, que era igual a 10. Esta mesma lógica se aplica a todos os outros operadores do Quadro 6, é claro, respeitando a respectiva operação aritmética associada à atribuição.

Existem, ainda, as expressões relacionais, cujos operadores referem-se à comparação entre dois valores numéricos quaisquer. Tal comparação relacional, em Python, pode resultar ou verdadeiro (`True`) ou falso (`False`). Os operadores relacionais estão disponíveis no Quadro 7, em que é possível visualizar o operador, o símbolo associado e a forma de uso.

Operador	Símbolo	Exemplo
Igual	$==$	$x == 1$
Diferente	$!=$	$x != y$
Maior	$>$	$x > 5$
Menor que	$<$	$y < 12$
Maior ou igual a	\geq	$x \geq 6$
Menor ou igual a	\leq	$y \leq 7$

Quadro 7 - Operadores relacionais / Fonte: o autor.

Já as expressões lógicas são aquelas cujo resultado consistem em um valor lógico verdadeiro ou falso. Neste tipo de expressão, podem ser usados os operadores relacionais, os operadores lógicos ou as expressões matemáticas.

No Quadro 8, a seguir, é descrito cada um dos operadores lógicos de conjunção, disjunção e negação.

Operador	Comando	Exemplo
Disjunção	<code>or</code>	A disjunção entre duas variáveis resulta em um valor verdadeiro quando, pelo menos, uma das variáveis é verdadeira.
Conjunção	<code>and</code>	A conjunção entre duas variáveis resulta em um valor verdadeiro somente quando as duas variáveis são verdadeiras.
Negação	<code>not</code>	A negação inverte o valor de uma variável. Se a variável x é verdadeira, então, a negação de x torna o valor da variável falso.

Quadro 8 - Operadores relacionais / Fonte: adaptado de Leal e Oliveira (2020).

Vale destacar que, em uma mesma expressão, podemos ter mais de um operador, seja ele aritmético, seja ele lógico ou relacional. Em casos nos quais há apenas um único operador, a avaliação da expressão é realizada de forma direta. Todavia, quando há mais de um, é necessária uma avaliação da expressão tomando por base as prioridades entre os operadores, analisando-os um por vez, passo a passo.

A precedência entre operadores se dá da seguinte forma:

1. Toda e qualquer operação destacada entre parênteses tem mais prioridade, lembrando-se que, quando há parênteses aninhados (uns dentro dos outros), deve-se resolver, primeiramente, os parênteses mais internos e, só depois, os mais externos.
2. Operações aritméticas são as primeiras a serem avaliadas pelo interpretador Python: primeiramente, tenta-se resolver operações de expo-
nenciação, em um segundo momento, as operações de multiplicação ou divisão são avaliadas e, por fim, as operações de adição e subtração são resolvidas.
3. Expressões relacionais são as próximas na cadeia de precedência. Após ter avaliado as expressões aritméticas, pode-se comparar os resultados numéricos dos operandos envolvidos na operação relacional. Como resultado, pode-se obter um valor lógico True ou False, dependendo se a comparação relacional é verdadeira ou falsa, em tempo de execução, respectivamente.
4. Por fim, a máquina tentará resolver as negações lógicas, depois, as con-
junções lógicas, finalmente, as disjunções lógicas, nesta ordem.



Caso você esteja em dúvida sobre como criar algum comando específico em Python ou sobre como funcionam certos conceitos, acesse a documentação oficial.



The diagram features a central vertical line with three horizontal branches extending from it. The top branch contains the word 'AS FUNÇÕES' in large white capital letters. The bottom branch contains the words 'EMBUTIDAS E' in large white capital letters. The leftmost branch contains the word 'Input' in blue capital letters. The rightmost branch contains the word 'Output' in blue capital letters. Below the central line, there is a large, stylized graphic of a person's head and shoulders, facing right. The person is wearing a light blue shirt and dark trousers. A large blue arrow points to the right, starting from the person's head and ending at the 'Output' text.

AS FUNÇÕES EMBUTIDAS E A ENTRADA E SAÍDA DE DADOS

Existem diversas funcionalidades que já foram pré-programadas e estão prontas para o uso do(a) programador(a), tais funcionalidades se manifestam em forma de funções em Python. O Quadro 9, a seguir, apresenta algumas das principais funções dessa linguagem e os seus respectivos usos.

Função	Descrição
<code>int(dado)</code>	Tenta retornar o dado convertido para números inteiros.
<code>str(dado)</code>	Tenta retornar o dado em seu formato texto.
<code>float(dado)</code>	Tenta retornar o dado convertido para números reais.
<code>bool(dado)</code>	Tenta retornar o dado convertido para dados booleanos.
<code>len(string)</code>	Retorna a quantidade de caracteres de uma <code>string</code> ou a quantidade de elementos em uma lista qualquer.
<code>abs(número)</code>	Retorna o valor absoluto do número.
<code>pow(x, y)</code>	Retorna o cálculo de <code>x</code> elevado à potência <code>y</code> . Similar ao operador <code>**</code> .
<code>round(número)</code>	Arredonda um número com casas decimais para seu inteiro respectivo.
<code>min(arg1, arg2, ..., argN)</code>	Retorna o menor valor dentre uma sequência de valores (<code>arg1, arg2, ..., argN</code>).
<code>max(arg1, arg2, ..., argN)</code>	Retorna o maior valor dentre uma sequência de valores (<code>arg1, arg2, ..., argN</code>).
<code>print(dado)</code>	Mostra uma mensagem de texto na saída padrão da máquina, geralmente, o terminal (console – prompt) do display – monitor, da tela do computador.
<code>input(msg)</code>	Imprime o texto contido dentro da <code>string msg</code> na tela e aguarda o usuário inserir alguma coisa via teclado e, depois, pressionar <code>enter</code> .
<code>type(dado)</code>	Retorna o tipo do dado.
<code>isinstance(objeto, tipoclasse)</code>	Retorna se o <code>objeto</code> (variável) é uma instância do tipo (classe) <code>tipoclasse</code> .
<code>id(objeto)</code>	Retorna o identificador único do <code>objeto</code> (variável), mais especificamente, o endereço do <code>objeto</code> em memória.

Quadro 9 - Operadores relacionais / Fonte: o autor.

Existem diversas outras funções embutidas em Python, todavia, neste momento, não é interessante mencioná-las, para não tornar sua vida de aspirante a profissional da programação ainda mais complexa. Para além das funções embutidas, também há a possibilidade de incluirmos módulos à parte, em Python, para adicionar ainda mais funcionalidades aos nossos programas. Módulos bastante comuns são: `random`, `time`, `collections`, `csv`, `array` etc.

Para além dos módulos nativos do Python, existe, ainda, a possibilidade de adicionarmos bibliotecas (*packages*) de terceiros, as quais são munidas de vários módulos importantes para quem quer avançar, ainda mais, na programação em Python. Como exemplo de *packages* famosas, temos: `matplotlib`, `pandas`, `numpy`, `opencv`, `django`, `flask`, `pygame`, `wxpython`, `scipy`, `tensorflow`, entre outras.

A entrada e saída de dados

Existem duas funções embutidas do Python que utilizaremos com muita frequência, na maioria de nossos programas. São as funções de entrada e saída de dados, `input()` e `print()`. Afinal de contas, para que serve um programa se ele não for útil às pessoas? Portanto, é importante que os usuários sejam capazes de interagir com nossos programas, de alguma forma.

A maneira mais simples de fazer com que um usuário entenda o que está se passando na máquina é imprimindo alguma mensagem de texto na tela. Por isso, inicialmente, elaboraremos sobre a função `print()`. Observe a Figura 4, a seguir. Nela, o código-fonte é tão simples quanto uma única linha de código. Este nosso programa de uma única linha contém nada menos que a invocação da função `print()`, cujo argumento é o texto “Olá, mundo!”. Desse modo, após executar o programa, podemos ver que o resultado de sua respectiva execução é a impressão da mensagem “Olá, mundo!” no terminal embutido do VS Code.

The screenshot shows the Visual Studio Code interface. In the top bar, the file 'HelloWorld.py' is open under the 'Códigos' section. The code editor displays the single line: `print("Olá, mundo!")`. Below the editor is the 'TERMINAL' tab, which shows the output of the program's execution: `Olá, mundo!`. The bottom status bar indicates the environment is 'Python 3.8.5 64-bit'.

Figura 4 - Imprimindo uma mensagem na tela para o usuário (saída de dados) / Fonte: o autor.

Descrição da Imagem: Na imagem, tem-se um recorte do ambiente de programação Visual Studio Code, no qual aparece, na parte superior da figura, o código-fonte de um programa cujo propósito é a impressão da mensagem "Olá, mundo!". Já na parte inferior, mostra-se o resultado da execução do programa e a respectiva mensagem sendo impressa no terminal embutido do referido ambiente de programação.

Essencialmente, tudo o que o programa da Figura 4 fez foi realizar, somente, uma saída de dados. Todavia, vamos além: e se quiséssemos trabalhar com variáveis e realizar a impressão de seus conteúdos na tela? Observe a Figura 5, a seguir.

```
File Edit Selection View Go ... TestandoPrint.py - Códigos - Vis...
EXPLORER ...
OPEN EDITORS ...
CÓDIGOS ...
TestandoPrint.py ...
1 idade = 18
2 print("O estudante tem", idade, "anos")
...
1: Python Debug Consc ...
t.py'
O estudante tem 18 anos
PS C:\Users\Pietro\Dropbox\__CLT\Unicesumar\DISCIP
LINAS\__CD - Lógica de Programação\Códigos>
Python 3.8.5 64-bit ⚡ 0 △ 0 ⏴ Ln 2, Col 31 Spaces: 4 UTF-8 CRLF Python ⌂ ⌂
```

Figura 5 - Imprimindo uma mensagem na tela para o usuário (saída de dados) / Fonte: o autor.

Descrição da Imagem: No código-fonte apresentado nesta imagem, cria-se uma variável de nome `idade`, cujo conteúdo recebe o valor numérico 18. Em seguida, o código apresenta uma instrução que imprime, na tela, uma mensagem contendo o conteúdo da variável `idade`. Na porção inferior da imagem, mostra-se a execução do respectivo código-fonte, resultando na impressão da mensagem “O estudante tem 18 anos”.

Perceba, na Figura 5, que, por meio da variável `idade`, pudemos armazenar o valor numérico 18 (linha de código 1). Isso se deve ao fato de que, no momento em que a máquina executa a expressão de atribuição `idade = 18`, cria-se um espaço em memória que armazenará o dado inteiro 18. Em seguida, na linha de código 2, temos a invocação da função `print()`, à qual passamos os seguintes argumentos, em sequência, separados por vírgula: “`O estudante tem`”; `idade` e “`anos`”. Os três argumentos passados à função são concatenados (emendados) e impressos na tela, da seguinte forma: “O estudante tem 18 anos”.

É importante observar que os textos que aparecem, entre aspas duplas, na linha 2 do código-fonte, são impressos, literalmente, na tela, durante a execução. Já a palavra `idade`, que não está entre aspas duplas, é interpretada como uma variável. Por isso, ao invés de ser impressa a mensagem “`idade`”, literalmente, na tela, é impresso o respectivo conteúdo da variável `idade`, que é igual a 18 (dezoito).

Por fim, é interessante falarmos da função que permite não só que o usuário veja informações impressas na tela bem como permite que o usuário, ele próprio,

insira os dados que desejar, dentro de nossos programas. Portanto, para realizar a entrada de dados, precisaremos utilizar comandos que envolvem `input()`. Essa função permite que o usuário digite dados, inseridos por meio da entrada padrão do dispositivo computacional (geralmente, o teclado). Tudo o que o usuário digitar, antes de pressionar a tecla *enter*, será capturado pelo `input()`, e poderá ser utilizado dentro de nossos programas, em conjunto com variáveis, expressões e outras funções.

The screenshot shows a Python code editor interface. In the center, there is a code editor window titled "TestandoInput.py - Códigos - Vis...". The code contains two lines of Python:idade = input("Qual sua idade? ")
print("O estudante tem", idade, "anos")A terminal window below the code editor shows the execution of the script. It prompts the user with "Qual sua idade? 26" and then prints "O estudante tem 26 anos". The terminal also displays the file path "PS C:\Users\Pietro\Dropbox__CLT\Unicesumar\DISCIP LINAS__CD - Lógica de Programação\Códigos>". At the bottom of the screen, the Python version "Python 3.8.5 64-bit" and other status information like line count and encoding are visible.

Figura 6 - Realizando entrada de dados por meio da função `input()`. / Fonte: o autor.

Descrição da Imagem: No código-fonte apresentado, nesta imagem, cria-se uma variável de nome `idade`, cujo conteúdo recebe um valor a ser informado pelo usuário, via teclado, em tempo de execução. Em seguida, o código apresenta uma instrução que imprime, na tela, uma mensagem contendo o conteúdo da variável `idade`. Na porção inferior da imagem, mostra-se a execução do respectivo código-fonte, considerando que o usuário inseriu o valor 26 para a variável `idade`, resultando na impressão da mensagem "O estudante tem 26 anos".

A Figura 6 traz um exemplo de como devemos utilizar a função `input()`. Perceba, na linha 1, que a variável `idade` está recebendo o resultado da execução da função `input()`. Ou seja, sempre que realizamos uma entrada de dados, precisaremos reservar algum espaço em memória para manter aquele dado armazenado durante a execução de nosso programa. Ainda na linha 1, percebe-se que o argumento passado entre parênteses para a função `input()` é, exatamente, o texto que queremos que o usuário leia, antes de realizar a ação que desejamos, neste caso, informar a sua idade. Por fim, nosso código-fonte imprime o conteúdo da variável `idade`, na tela, mostrando ao usuário qual foi o valor que ele inseriu.

Ainda na Figura 6, podemos ver que o resultado da execução mostra que o usuário, ao testar o programa, no momento da execução da linha de código 1, fez a máquina imprimir a mensagem “Qual sua idade?” e aguardou pela ação de inserção do usuário. Em seguida, a pessoa que estava testando o programa inseriu um valor igual a 26 e, depois, pressionou *enter*. Assim, finalmente, durante a execução da linha de código 2, a mensagem “O estudante tem 26 anos” foi impressa na tela.

The screenshot shows the Visual Studio Code interface. In the Explorer sidebar, there is a file named 'TestandoInput.py'. The code editor contains the following Python script:

```
idade = int(input("Qual sua idade? "))
print("No ano que vem, terá", idade+1, "anos")
```

In the terminal window, the output is:

```
Qual sua idade? 30
No ano que vem, terá 31 anos
```

The terminal also shows the current working directory: C:\Users\Pietro\Dropbox\CLT\Unicesumar\DISCIPLINAS\CD - Lógica de Programação\Códigos> |

Figura 7 - Convertendo o resultado da função `input()` de `str` para `int` / Fonte: o autor.

Descrição da Imagem: No código-fonte apresentado nesta imagem, cria-se uma variável de nome `idade`, cujo conteúdo recebe um valor a ser informado pelo usuário, via teclado, em tempo de execução. Em seguida, o código apresenta uma instrução que imprime, na tela, uma mensagem contendo o conteúdo da variável `idade` acrescida em uma unidade. Na porção inferior da imagem, mostra-se a execução do respectivo código-fonte, considerando que o usuário inseriu o valor 30 para a variável `idade`, resultando na impressão da mensagem “No ano que vem, terá 31 anos”.

É importante ressaltar que a função `input()` sempre retorna os dados do usuário no formato texto, de tipo `str`. Assim sendo, sempre que precisarmos trabalhar os dados como sendo dados numéricos, ou de outros tipos, devemos realizar a respectiva conversão de dados, utilizando, para isso, a função adequada. Observe, na Figura 7, um caso no qual convertemos, automaticamente, o dado inserido pelo usuário de `str` (retornado pela função `input()`) para `int` (retornado pela função `int()`).

Construindo um programa

Agora que já conhecemos os principais conceitos da lógica de programação em Python, podemos elaborar nosso primeiro código-fonte. Assim sendo, construiremos um programa que leia nome, idade e altura de uma pessoa e exiba nome, idade, altura e ano de nascimento dessa pessoa. Para facilitar o entendimento do problema, o estruturaremos em três partes distintas: entrada, processamento e saída.

No que diz respeito a operações e inserção de dados, é preciso que o usuário insira os dados de nome, idade e altura. É preciso que exista uma variável em memória para cada uma das informações recém citadas. Em seguida, utiliza-se a função `input()` para preencher o conteúdo de tais variáveis. Na parte do processamento, deve-se realizar o cálculo do ano de nascimento do indivíduo, subtraindo o ano atual pelo ano de nascimento. A saída de dados será a impressão, na tela, de nome, idade, altura e ano de nascimento, com base no que foi informado pelo usuário, via teclado. As operações de saída de dados serão feitas pela função `print()`.

Uma vez que entendemos o problema, podemos partir para a construção do código-fonte do programa. Os valores obtidos na entrada de dados precisam ser armazenados em variáveis. Assim sendo, quais seriam os tipos de cada variável? O nome deve ser armazenado em uma variável do tipo texto, assim, não precisaremos converter os dados lidos pela função `input()`, uma vez que essa função já retorna dados do tipo `str`. Os dados de altura, geralmente, admitem casas decimais, por isso, neste caso, precisaremos converter os dados retornados pela função `input()`, por meio de uma segunda função, a `float()`. Além disso, a variável `idade`, geralmente, é um número inteiro, assim, neste ponto, utilizaremos a função `int()` para converter os dados lidos com o `input()`.

Quadro 10 - Código-fonte em Python / Fonte: o autor.

O Quadro 10 apresenta o código-fonte para o programa recém descrito. Note que, na linha 5 desse código, existe um comentário (texto precedido do caractere #). Comentários são utilizados como anotações “inertes” em nossos códigos, nem são interpretados, nem executados pela máquina.

Figura 8 - Resultado da execução do código-fonte do Quadro 10 / Fonte: o autor.

Descrição da Imagem: Nesta imagem temos o resultado da execução do programa escrito no Quadro 10. Dessa forma, considerando que o usuário inseriu, nesta ordem, os dados "Zé Dev do Python", "1.75" e "28", o programa, ao final, imprime as mensagens: "O nome é Zé Dev do Python", "A altura é: 1.75", "A idade é: 28" e "O ano de nascimento é: 1993".

Os resultados da execução do código-fonte do Quadro 10 podem ser visualizados na Figura 8. Observe que, inicialmente, os dados foram obtidos do usuário (entradas) bem como um breve cálculo sobre a idade na linha de código 06 (processamento) e, em seguida, apresentados (saída).

CONSIDERAÇÕES FINAIS

Nesta unidade você deu os primeiros passos no aprendizado da linguagem de programação Python, conhecendo as características e como o código que escrevemos se torna um programa executável.

Conhecemos a estrutura básica de um programa em Python, que pode ser composta de operações de entrada, processamento e saída de dados. Nossos códigos-fonte seguem um raciocínio lógico que, pode, neste caso, ser sequencial, tendo começo, meio e fim. Quando estamos tentando entender o enunciado de um problema, é interessante tentar identificar o que pode ser entrada de dados, o que deve ser processado e, ao final, o que será exibido ao usuário por meio de uma ou mais saídas de dados.

Aprendemos como realizar anotações em nossos códigos inserindo comentários, os quais são mensagens escritas, diretamente, no código-fonte, porém precedidas do caractere cerquilha (#). Entendemos quais são as regras que devem ser seguidas para se nomear uma variável e estudamos os tipos de dados básicos, a saber: `int`, `float`, `str` e `bool`.

Vimos quais são os operadores aritméticos, relacionais e lógicos, além das funções embutidas da linguagem Python. Entendemos, também, como realizar atribuição e modificação do conteúdo das variáveis, por meio do operador de atribuição (=). Percebemos que é possível combinar o operador de atribuição com diversos operadores aritméticos distintos, no intuito de atualizar o conteúdo de uma variável.

Nas funções de entrada e saída de dados, conhecemos o `print()` e o `input()`, que são úteis para quando o(a) programador(a) tem o intuito de instruir o programa a se comunicar com o usuário, de alguma forma. Por fim, foram propostas diversas atividades cuja resolução encontra-se na seção Gabarito.

Lembre-se: programação se aprende fazendo programas, somente realizar leituras não adiantará muito.



1. Assinale V nos identificadores corretos e F nos identificadores inválidos, de acordo com as regras de nomeação de variáveis da linguagem Python.

- () idade
- () nome*r
- () media_peso%
- () aluno_nota
- () media_idade
- () x2
- () endereço+cep
- () A
- () 2nome
- () 012
- () /fone
- () 1234P

A sequência correta para a resposta da questão é:

- a) a)V, F, F, V, F, V, F, V, F, F, F.
- b) b)V, F, V, V, F, V, F, V, V, V, F, F.
- c) c)F, F, F, V, V, V, F, V, V, V, F, F, F.
- d) d)V, F, F, V, F, V, F, V, F, F, V, F.
- e) e)F, F, V, V, F, V, F, V, F, V, F, F.



2. Identifique os erros sintáticos e semânticos no programa, a seguir, cujo objetivo é ler dois números e apresentar a soma entre ambos. Ao final, reescreva o código para que ele funcione como o proposto.

```
01 #Entrada de dados
02 #Solicitando o primeiro número
03 num1 = int(input(Digite o primeiro número:\n))
04 #Solicitando o segundo número
05 num2 = int(input(Digite o segundo número:\n))
06
07 #Processamento
08 #Cálculo do ano de nascimento
09 num1 + num2 = total
10
11 #Saída de dados
12 print("A soma dos números é:", total)
```

Fonte: o autor.

3. Escreva um programa que leia um número inteiro e apresente seu antecessor e seu sucessor.
4. Elabore um programa que receba quatro notas e calcule a média aritmética entre elas.
5. Faça um programa que receba o valor de um depósito e o valor da taxa de juros, calcule e apresente o valor do rendimento e o valor total (valor do depósito + valor do rendimento).
6. Escreva um programa que receba dois números, calcule e apresente o resultado do primeiro número elevado ao segundo.



7. Elabore um programa que calcula a área de um trapézio.
8. Escreva um programa que leia o nome de uma pessoa e imprima a seguinte mensagem, na tela: "Bem-vindo(a) à disciplina de Lógica de Programação e Algoritmos, <nome>", onde o campo <nome> deve ser substituído pelo nome informado pelo usuário.
9. Escreva um programa que leia um número positivo inteiro e apresente o quadrado e a raiz quadrada desse número.
10. Escreva um programa que receba três números e seus respectivos pesos, calcule e apresente a média ponderada entre eles. Observação: a média pode ser um dado numérico com casas decimais, portanto, prepare seu programa para que ele seja capaz de apresentar o resultado apropriado, quando necessário.
11. Escreva um programa que, dado o raio de um círculo, calcule a sua área e o perímetro. A área é a superfície do objeto, dada por 3,14 multiplicados pelo raio elevado ao quadrado. O perímetro é a medida do contorno do objeto dado por 2 multiplicados por 3,14, multiplicados, novamente, pelo raio.



TODA VARIÁVEL EM PYTHON É, NA VERDADE, UM OBJETO!

Python é uma linguagem que dá suporte à orientação a objetos. Na realidade, quase tudo, em Python, é um objeto.

Assim sendo, um objeto em linguagem de programação abstrata representa a posição onde será armazenada a respectiva informação. Os objetos em Python apresentam os seguintes atributos:

- **Tipo:** o tipo de um objeto determina os valores que o objeto pode receber e as operações que podem ser executadas nesse objeto.
- **Valor:** o valor de um objeto é o índice de memória ocupada por essa variável. Como os índices das posições da memória são interpretados, isto é determinado pelo tipo da variável.
- **Tempo de vida:** a vida de um objeto é o intervalo de tempo em que ele se mantém em memória principal, durante a execução de um programa em Python. É durante esse tempo que o objeto existe.

Python define uma extensa hierarquia de tipos. Essa hierarquia inclui os tipos numéricos (tais como `int`, `float` e `complex`), sequências (tais como `a tuple` e `a list`), funções (tipo `function`), classes e métodos (tipos `classobj` e `instancemethod`) e as instâncias da classe (tipo `instance`).

A fim de utilizar um objeto em um programa em Python, ele deve ter um nome. O nome de um objeto é uma variável usada para identificar esse objeto em um programa. Em Python, um objeto não pode ter zero, um ou mais nome.

Veja o seguinte trecho de código:

```
i = 57
```

Esta indicação cria um objeto com nome `i` e cria vários atributos com esse objeto. O tipo do objeto é `int`, e seu valor é 57. Alguns atributos de um objeto, tais como seu tipo, são limitados quando o objeto é criado e não podem ser mudados. Isso é chamado de tipagem forte. As instanciações para outros atributos de um objeto, tais como seu valor, podem permitir mudanças durante a execução do programa, onde o objeto está.

Veja o seguinte trecho de código:

```
i = int(57)
```



Se seguirmos a instrução anterior com uma instrução seguinte, de atribuição, como:

```
j = i
```

então, os nomes `i` e `j` se referem ao mesmo objeto em memória!

A comparação entre os conteúdos de tais variáveis (objetos) ficaria:

```
if i == j:  
    print("valores iguais")
```

O código apresentado é o teste que verifica se o valor (conteúdo) do objeto `i` é o mesmo valor do objeto `j`. Entretanto nada impede que objetos distintos tenham o mesmo valor. Para testar se dois identificadores distintos são, estritamente, um mesmo objeto em memória (para além do conteúdo, apenas), é necessário utilizar o comando `is`:

```
if i is j:  
    print("mesmo objeto")
```

Para saber se os tipos de dados de dois objetos são iguais, é necessário fazer algo como:

```
i = 57  
j = 47  
if type(i) == type(j):  
    print("mesmo tipo")
```

Ainda, em Python, um nome (identificador) sempre será tratado como um objeto (para além de uma mera variável simples). Entretanto, às vezes, é conveniente usar uma palavra reservada chamada `None`, quando se quer tratar um objeto como "nulo". Python fornece um tipo especial de dados para esta finalidade, chamado `NoneType`. Sempre há somente um objeto do tipo `NoneType`, e o nome desse objeto é `None`.

Nós podemos, explicitamente, atribuir o `None` a uma variável qualquer:

```
f = None
```

Também, podemos testar, explicitamente, se um objeto é `None`, como:

```
if f is None:  
    print("é nulo")
```

Fonte: adaptado de Python Brasil (2008, on-line).



eu recomendo!



filme

O Jogo da Imitação

Ano: 2015

Sinopse: em 1939, a recém-criada agência de inteligência britânica MI6 recruta Alan Turing, um aluno da Universidade de Cambridge, para entender códigos nazistas, incluindo o Enigma, que os criptógrafos acreditavam ser inquebrável. A equipe de Turing analisa as mensagens de Enigma, enquanto Turing constrói uma máquina para decifrá-las. Após desvendar as codificações, ele se torna herói, porém, em 1952, autoridades revelam sua homossexualidade, e a vida de Turing vira um pesadelo.



Comentário: se a sua dificuldade é apenas com relação aos conceitos e à sua forma de estudar, inspire-se neste filme, que mostra como Alan Turing tinha, para além das dificuldades conceituais e das limitações tecnológicas da época, problemas graves com uma sociedade cujos valores conservadores se mostraram prejudiciais à vida de quem tanto contribuiu e contribui para a evolução tecnológica de toda a humanidade.



ESTRUTURAS CONDICIONAIS

PROFESSOR

Me. Pietro Martins de Oliveira

PLANO DE ESTUDO

A seguir, apresentam-se as aulas que você estudará nesta unidade:

- Estrutura condicional
- Estrutura condicional simples
- Estrutura condicional composta
- Estruturas condicionais combinadas e aninhadas.

OBJETIVOS DE APRENDIZAGEM

- Compreender a estrutura condicional.
- Conhecer a estrutura condicional simples.
- Conhecer a estrutura condicional composta.
- Elaborar algoritmos utilizando estruturas condicionais simples e compostas.

INTRODUÇÃO



Nesta unidade, você estudará as estruturas de decisão, também conhecidas como estruturas de seleção. Estas podem ser categorizadas em estrutura condicional simples e estrutura condicional composta. Com os conhecimentos adquiridos na Unidade 1, conseguimos construir programas sequenciais, isto é, em que, a partir da entrada, os dados são processados, sequencialmente, sem desvios de fluxo de execução. Ao fim do algoritmo, apresentamos algumas informações na saída.

Todavia, com estruturas condicionais, podemos impor condições para a execução de uma instrução ou um conjunto de instruções, ou seja, podemos criar condições que permitem desviar o fluxo de execução de um programa. Para construir essas condições, utilizaremos os conceitos de variáveis, de expressões lógicas e expressões relacionais, vistos na Unidade 1.

Veremos como a estrutura condicional simples nos auxilia, enquanto programadores, a originar programas que possibilitem criar tomadas de decisão em nossos algoritmos. Este tipo de instrução permite que um bloco de comandos seja executado, ou não, de acordo com a avaliação de uma expressão lógico-relacional. Ainda, veremos a estrutura de condição composta, na qual o código-fonte é criado para que, obrigatoriamente, um, dentre dois blocos de comandos, seja executado. Se a condição for verdadeira, o primeiro bloco é executado, caso contrário, o segundo bloco é executado.

Além disso, poderemos combinar estruturas condicionais, colocando-as em sequência, uma após a outra, ou mesmo, inserindo uma condição dentro da outra. Ao estudar cada estrutura condicional, veremos que, em alguns exemplos, teremos condições de construir algoritmos para visualizar a aplicação dos conceitos estudados.

No fim desta unidade, estaremos aptos a construir programas com desvios de fluxo, aumentando, assim, o leque de problemas que poderemos resolver.



ESTRUTURA CONDICIONAL

Para que os programas sejam capazes não só de seguir um fluxo contínuo, mas também de executar tomadas de decisão, é imprescindível que as linguagens de programação sejam capazes de oferecer estruturas de desvio de fluxo. Assim sendo, as estruturas condicionais, também conhecidas como estruturas de controle ou estruturas de seleção, são as responsáveis por dar este tipo de poder aos programadores.

A partir de uma expressão lógico-relacional, é possível testar se uma condição é verdadeira, caso ela seja, então, o bloco de comandos condicionado a ela será executado, se não, esse bloco de comandos será ignorado, sem ser executado pelo computador.

Nas aulas seguintes, estudaremos as estruturas de condição da linguagem Python.

2 ESTRUTURA CONDICIONAL SIMPLES

Na estrutura condicional simples, o bloco de comandos é executado se, e somente se, a expressão da condição for avaliada como sendo verdade. Os valores lógicos que uma expressão lógico-relacional pode assumir são dois, a saber: verdadeiro ou falso (True ou False, respectivamente).

A sintaxe do comando é:

`if <condição>:`

`<Bloco de comandos p/ condição verdadeira>`

A estrutura condicional simples tem, por finalidade, possibilitar ao programa que tome uma decisão de forma automática, em tempo de execução. De modo que, se a condição indicada no campo `<condição>` for verdadeira, são executadas todas as instruções que dependem daquela condição. Note, pela sintaxe anterior, que o campo `<Bloco de comandos p/ condição verdadeira>` está levemente “recuado”, em relação ao comando `if`, isso se chama endentação, e, a partir deste recuo, saberemos se um bloco de comandos está ou não condicionado à expressão lógico-relacional que compõe o teste lógico do comando `if`.

Ainda considerando a sintaxe do comando `if`, recém apresentada, se em tempo de execução, a expressão lógico-relacional que deve substituir o campo `<condição>` for falsa, então, o algoritmo ignorará todo o bloco de comandos indentado (com recuo) que se encontra, imediatamente, após o `if`. Assim, o programa executará as instruções seguintes, não indentadas.

Em Python, é obrigatória a indentação de, ao menos, um comando após um comando `if`. Esse recuo torna a legibilidade do código muito melhor e, além de ser parte da própria sintaxe de estruturas condicionais, trata-se, ainda, de uma boa prática de programação para manutenção de códigos.

Agora que conhecemos a sintaxe da estrutura condicional simples, construiremos nosso primeiro programa em Python com desvio de fluxo de execução. O problema consiste em obter um número inteiro, informado pelo usuário, e, se for par, deve-se imprimir a raiz quadrada do respectivo número, na tela.

O Quadro 1 apresenta o programa em Python para o problema anteriormente descrito. Lembre-se de que os textos precedidos de `#` correspondem a comentários e não são interpretados, pela máquina, como comandos executáveis.

Note que, nesse programa, devemos utilizar a função embutida `pow()`, de forma a podermos calcular o resultado da raiz quadrada. Perceba que todo número elevado a uma fração representa, na realidade, o cálculo de uma raiz, tomando por base os ensinamentos de matemática básica. Caso fosse necessário, a linguagem Python teria, ainda, várias outras funções embutidas para realizarmos operações matemáticas, por exemplo cálculos trigonométricos.

Em relação às variáveis, foram declaradas utilizadas duas: `num` e `raiz`. O número inteiro obtido do usuário é armazenado na variável `num`, como podemos ver na linha de código 02, do Quadro 1. Já o resultado do cálculo da raiz quadrada deste número será armazenado na variável `raiz`, na linha de código 08. O teste lógico-relacional, realizado na estrutura condicional simples (`if` da linha 05) consiste em verificar se o valor do resto da divisão de `num` por 2 é igual a zero. Se esta condição for verdadeira, é executado o conjunto de instruções delimitado pela indentação (linhas de código de 06 a 10). Ou seja, temos a execução da instrução de atribuição do valor da raiz quadrada de `num` à variável `raiz` (linha 08) seguida da execução da impressão da mensagem da linha 10. Se o valor do teste lógico da linha de código 05 for falso, então os comandos das linhas 08 e 10 serão ignorados e o programa se encerrará sem imprimir nada a mais, na tela (LEAL; OLIVEIRA, 2020).

```
01 #Entrada de dados
02 num = int(input("Digite um número inteiro:\n"))
03
04 #Teste condicional
05 if num % 2 == 0:
06     #Se a condição anterior for verdadeira:
07     #Calcula-se a raiz quadrada de num
08     raiz = pow(num, 1/2)
09     #Saída de dados
10     print("A raiz quadrada é:", raiz)
```

Quadro 1 - Exemplo de uso do comando if em Python / Fonte: o autor.



Sabia que quando o Python encontra um `\n` em uma *string*, ele pula de linha?



Analisaremos o resultado da simulação do programa para os valores 18 (Figura 1) e 15 (Figura 2), respectivamente. Com o valor de `num` sendo igual a 18, temos um número par, sendo que o resultado da divisão inteira de 18 por 2 é exatamente 9, restando 0. Ou seja, nesse caso, o teste lógico da linha 05 (`num % 2 == 0`) é verdadeiro. Com isso, temos a impressão do valor da raiz quadrada, no caso, aproximadamente 4.243 (execução das linhas de código 08 e 10) (LEAL; OLIVEIRA, 2020).

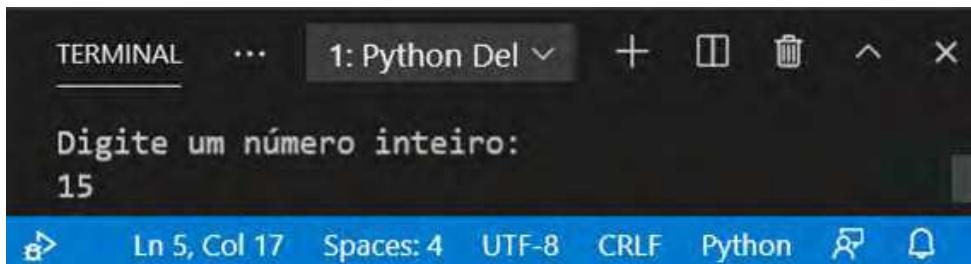


```
Digite um número inteiro:  
18  
A raiz quadrada é: 4.242640687119285
```

Figura 1 - Resultado da execução do programa do Quadro 1 para num igual a 18
Fonte: o autor.

Descrição da Imagem: na imagem, exibe-se o terminal embutido do Visual Studio Code, no qual o programa imprime a mensagem “Digite um número inteiro:” e, em seguida, o usuário insere o valor “18”. Por fim, o programa mostra o resultado com a seguinte mensagem: “A raiz quadrada é: 4.242640687119285”.

Observe a Figura 2, no caso da execução na qual o usuário atribui a num o valor 15, o resultado do teste lógico é falso, pois 15 divididos por 2 resultam em 7 como quociente, restando 1. Desse modo, as instruções compreendidas pelo bloco de comandos indentado, das linhas 06 a 10, não serão executadas (LEAL; OLIVEIRA, 2020).



```
Digite um número inteiro:  
15
```

Figura 2 - Resultado da execução do programa do Quadro 1 para num igual a 15
Fonte: o autor.

Descrição da Imagem: na imagem, o resultado do mesmo programa anterior é exibido. Todavia, agora, quando o programa imprime a mensagem “Digite um número inteiro:” o usuário insere o valor “15”. Isso faz com que mais nada seja impresso na tela e o programa se encerre.

Assim, o programa se encerra sem apresentar o resultado do cálculo da raiz quadrada, como manda o enunciado do problema.



3 ESTRUTURA CONDICIONAL COMPOSTA

Na estrutura condicional composta, é realizada a avaliação de uma única expressão lógico-relacional. Se o resultado desta avaliação for verdadeiro, é executada a instrução ou o conjunto de instruções indentado, relacionado ao respectivo comando `if`. Caso contrário, se o resultado da avaliação for falso, é executado um outro conjunto de instruções indentadas após o comando `else` (LEAL; OLIVEIRA, 2020). Nesse caso, temos certeza de que ao menos um dos blocos de comando será executado, ou o do `if`, ou o do `else`, todavia, nunca os dois simultaneamente.

A sintaxe da estrutura condicional composta é:

`if <condição>:`

`<Bloco de comandos p/ condição verdadeira>`

`else:`

`<Bloco de comandos p/ condição falsa>`

Para facilitar o entendimento quanto ao funcionamento da estrutura condicional composta, construiremos um programa para identificar se um número é par ou ímpar. Se o número for par, devemos apresentar sua raiz quadrada e, se for ímpar, devemos apresentar o número elevado ao quadrado (LEAL; OLIVEIRA, 2020).

No Quadro 2, a seguir, é apresentado um programa em Python que verifica se o número é par ou ímpar. Em relação às variáveis, nesse caso, foram declaradas três: num, quadrado e raiz. O número inteiro obtido pelo usuário é armazenado em num (LEAL; OLIVEIRA, 2020).

No teste lógico realizado na estrutura condicional da linha de código 03, verifica-se se o valor do resto da divisão do número lido, por dois, é igual a zero. Se esta condição for verdadeira, é executado o conjunto de instruções do if, delimitado pelas chaves que vão da linha 13 à linha 17, isso é, temos a execução da instrução de atribuição do valor da raiz quadrada de num à variável raiz (linha 04), seguido da impressão de uma mensagem ao usuário, informando o resultado (linha 05). Nessa situação, o bloco de comandos das linhas 06 a 08 é sumariamente ignorado. Todavia, caso contrário, se o valor do teste lógico da linha de código 03 for falso, então é o bloco de comandos das linhas 04 e 05 que é ignorado e, por sua vez, as linhas de código 07 e 08 serão executadas, e o cálculo do quadrado de num será exibido na tela para o usuário. Observe o código-fonte do Quadro 2 a seguir (LEAL; OLIVEIRA, 2020).

```
01 num = int(input("Digite um número inteiro:\n"))
02
03 if num % 2 == 0:
04     raiz = pow(num, 1/2)
05     print("A raiz quadrada é:", raiz)
06 else:
07     quadrado = pow(num, 2)
08     print("O número ao quadrado é:", quadrado)
```

Quadro 2 - Exemplo de uso dos comandos if e else em Python / Fonte: o autor.

Tomemos como exemplos de execução os dois casos de teste distintos, nos quais o usuário insere 15 (Figura 3) e 16 (Figura 4), respectivamente. Primeiramente, vamos ao resultado da execução do programa para num igual a 15. Nesse caso, a avaliação do teste lógico-relacional da linha 03, temos que o resto da divisão de 15 por dois não é igual a zero e, por isso, o bloco os comandos das linhas de código 04 e 05 são ignorados, fazendo com que o bloco de comandos do else seja executado por completo. Como podemos ver na Figura 3, nesse caso, o resultado do cálculo quadrado do número inserido pelo usuário é exibido na tela.

```
TERMINAL ... 1: Python Del
Digite um número inteiro:
15
O número ao quadrado é: 225
```

Ln 6, Col 6 Spaces: 4 UTF-8 CRLF Python ⚙️ 📡

Figura 3 - Resultado da execução do programa do Quadro 2 para num igual a 15
Fonte: o autor.

Descrição da Imagem: na figura, mostra-se o resultado da execução do programa do quadro 2. Aqui, o programa imprime a mensagem "Digite um número inteiro:", em seguida o usuário insere o dado "15". Por fim, o programa encerra com a mensagem "O número ao quadrado é: 225".

```
TERMINAL ... 2: Python
Digite um número inteiro:
16
A raiz quadrada é: 4.0
```

Spaces: 4 UTF-8 CRLF Python ⚙️ 📡

Figura 4 - Resultado da execução do programa do quadro 2 para num igual a 16
Fonte: o autor.

Descrição da Imagem: na figura, mostra-se outro resultado da execução do programa do quadro 2. Nesse caso, o programa imprime a mensagem "Digite um número inteiro:", em seguida o usuário insere o dado "16". Agora, o programa se encerra com a mensagem "Q raiz quadrada é: 4.0".

Ao analisar a execução na qual o usuário insere um num igual a 16, temos que o resultado do teste lógico da linha de código 03 é verdadeiro. Assim sendo, como podemos ver na Figura 4, o bloco de comandos do `if` é que é executado, e o bloco de comandos do `else` é ignorado, o que resulta na exibição do cálculo da raiz quadrada de 16 (LEAL; OLIVEIRA, 2020).



É possível utilizar um comando `else` sem que exista um comando `if` imediatamente anterior a ele? Tente criar um programa de teste qualquer, que utiliza apenas o `else`, sem `if`, e veja se o interpretador Python aceita esse tipo de sintaxe.



ESTRUTURAS CONDICIONAIS COMBINADAS E ANINHADAS

Existe ainda uma terceira forma de se utilizar estruturas condicionais, combinando-as em estruturas diversas. Para isso, podemos incluir o comando `elif` que é uma espécie de `else` emendado com um `if`. Basicamente, quando há mais de uma condição a ser avaliada, com comandos distintos associados a cada condição respectiva, podemos criar cadeias de decisões, combinando comandos `if`, `elif` e `else`. Isso dá a possibilidade da criação de múltiplas condições.

A estrutura `elif` é comumente utilizada, mas não restrita, a situações nas quais existem condições mutuamente exclusivas, em que, se um bloco de instruções for executado, os demais não serão. Para este tipo de situação poderíamos nos apoiar em uma sintaxe similar à seguinte:

```
if <condição 1>:  
    <Bloco de comandos 1>  
elif <condição 2>:  
    <Bloco de comandos 2>  
elif <condição N>:  
    <Bloco de comandos N>  
else:  
    <Bloco de comandos padrão>
```

Nesse tipo de cadeia de condições, avalia-se uma condição após a outra, em sequência. Se nenhuma das condições dos if e elif forem satisfeitas, então, inevitavelmente, o bloco de comandos do else será avaliado, e se executa o <Bloco de comandos padrão>. Esse tipo de encadeamento de decisões é bastante útil quando desejamos criar uma espécie de menu.

Para ilustrar o funcionamento desses conceitos, construiremos um programa que permita ao usuário escolher qual operação deseja realizar com dois números distintos. Assim, caso o usuário escolha a opção 1, deve-se realizar a soma; caso seja escolhida a opção 2, então o programa deve calcular a subtração entre os dois números; caso a opção seja a 3, então calcula-se a multiplicação; se o usuário escolher a opção 4, aí teremos o cálculo da divisão entre os dois números; e, por fim, caso nenhuma das opções 1, 2, 3 ou 4 sejam escolhidas, então o programa mostrará a mensagem “Opção inválida.” (LEAL; OLIVEIRA, 2020).

Em nossa solução, a entrada de dados consiste na leitura dos dois números além da operação desejada. Como processamento, temos que identificar a opção selecionada pelo usuário, utilizando estruturas combinadas if, elif e else. A saída de dados é a impressão do resultado da respectiva operação escolhida pelo usuário em tempo de execução. O programa em Python que implementa essa lógica pode ser visualizado no Quadro 3 a seguir (LEAL; OLIVEIRA, 2020).

```
01 num1 = float(input("Digite o primeiro número:\n"))
02 num2 = float(input("Digite o segundo número:\n"))
03
04 print("Escolha uma operação:")
05 print("Digite 1 para soma;")
06 print("Digite 2 para subtração;")
07 print("Digite 3 para Multiplicação;")
08 print("Digite 4 para Divisão.")
09 op = int(input())
10
11 if op == 1:
12     resultado = num1 + num2
13     print("A soma é:", resultado)
14 elif op == 2:
15     resultado = num1 - num2
16     print("A subtração é:", resultado)
17 elif op == 3:
18     resultado = num1 * num2
19     print("A multiplicação é:", resultado)
20 elif op == 4:
21     resultado = num1 / num2
22     print("A divisão é:", resultado)
23 else:
24     print("Opção inválida.")
```

Quadro 3 - Exemplo de uso dos comandos `if`, `elif` e `else` combinados / Fonte: o autor.

Os resultados obtidos pelas execuções do programa recém apresentado, para cada um dos casos de teste e cada uma das operações aritméticas implementadas podem ser visualizados nas Figuras 5, 6, 7, 8 e 9.

The screenshot shows a terminal window titled "1: Python Dek". The window contains the following text:

```
Digite o primeiro número:  
10  
Digite o segundo número:  
15  
Escolha uma operação:  
Digite 1 para soma;  
Digite 2 para subtração;  
Digite 3 para Multiplicação;  
Digite 4 para Divisão.  
1  
A soma é: 25.0
```

Below the terminal window, the status bar displays: "Ln 16, Col 21" and "Spaces: 4". The bottom right corner of the terminal window has icons for file operations.

Figura 5 - Resultado da execução do programa do Quadro 3 para a operação de soma (op = 1)
Fonte: o autor.

Descrição da Imagem: nesta imagem exibe-se o resultado da execução do programa do Quadro 3. A sequência de mensagens exibida pelo programa, e a respectiva resposta do usuário ocorreram na seguinte ordem: primeiramente o programa imprime a mensagem "Digite o primeiro número:" e então o usuário responde com o dado 10. Na sequência, o programa mostra a mensagem "Digite o segundo número:" e aí o usuário insere o dado 15. Em seguida, o programa mostra mensagens para que o usuário digite o número 1, caso queira realizar uma soma; número 2, caso queira subtração; número 3 em caso de multiplicação; ou que digite 4 para realizar uma divisão. Após isso, o usuário escolhe a opção 1 e então o programa finaliza com a mensagem "A soma é: 25.0".

A seguir, um outro caso de teste para o mesmo programa do Quadro 3.

Figura 6 - Resultado da execução do programa do Quadro 3 para a operação de soma (op = 2)
Fonte: o autor.

Descrição da Imagem: nesta imagem exibe-se o resultado da execução do programa do Quadro 3. A sequência de mensagens exibida pelo programa, e a respectiva resposta do usuário ocorreram na seguinte ordem. Primeiramente o programa imprime a mensagem "Digite o primeiro número:" e então o usuário responde com o dado 15. Na sequência o programa mostra a mensagem "Digite o segundo número:" e aí o usuário insere o dado 10. Em seguida, o programa mostra mensagens para que o usuário digite o número 1, caso queira realizar uma soma; número 2, caso queira subtração; número 3 em caso de multiplicação; ou que digite 4 para realizar uma divisão. Após isso, o usuário escolhe a opção 2 e então o programa finaliza com a mensagem "A subtração é: 5.0".

Para compreender melhor o programa do Quadro 3, segue mais um caso de teste.

Figura 7 - Resultado da execução do programa do Quadro 3 para a operação de multiplicação (op = 3) / Fonte: o autor.

Descrição da Imagem: nesta imagem exibe-se o resultado da execução do programa do Quadro 3. A sequência de mensagens exibida pelo programa, e a respectiva resposta do usuário ocorreram na seguinte ordem. Primeiramente o programa imprime a mensagem "Digite o primeiro número:" e então o usuário responde com o dado 10. Na sequência o programa mostra a mensagem "Digite o segundo número:" e aí o usuário insere o dado 15. Em seguida, o programa mostra mensagens para que o usuário digite o número 1, caso queira realizar uma soma; número 2, caso queira subtração; número 3 em caso de multiplicação; ou que digite 4 para realizar uma divisão. Após isso, o usuário escolhe a opção 3 e então o programa finaliza com a mensagem "A multiplicação é: 150.0".

O caso de teste a seguir, mostra como o programa do Quadro 3 se comporta quando o usuário deseja realizar uma divisão.

The screenshot shows a terminal window titled "1: Python". The window contains the following text:

```
Digite o primeiro número:  
15  
Digite o segundo número:  
10  
Escolha uma operação:  
Digite 1 para soma;  
Digite 2 para subtração;  
Digite 3 para Multiplicação;  
Digite 4 para Divisão.  
4  
A divisão é: 1.5
```

The terminal interface includes a toolbar at the top with icons for new tab, close, etc., and a status bar at the bottom showing "Spaces: 4", "UTF-8", "CRLF", "Python", and some icons.

Figura 8 - Resultado da execução do programa do Quadro 3 para a operação de divisão ($op = 4$) / Fonte: o autor.

Descrição da Imagem: nesta imagem exibe-se o resultado da execução do programa do Quadro 3. A sequência de mensagens exibida pelo programa, e a respectiva resposta do usuário ocorreram na seguinte ordem. Primeiramente o programa imprime a mensagem "Digite o primeiro número:" e então o usuário responde com o dado 15. Na sequência o programa mostra a mensagem "Digite o segundo número:" e aí o usuário insere o dado 10. Em seguida, o programa mostra mensagens para que o usuário digite o número 1, caso queira realizar uma soma; número 2, caso queira subtração; número 3 em caso de multiplicação; ou que digite 4 para realizar uma divisão. Após isso, o usuário escolhe a opção 4 e então o programa finaliza com a mensagem "A divisão é: 1.5".

Para completar os casos de teste do programa do Quadro 3, veja uma situação na qual o usuário tenta realizar uma operação não listada pelo programa.

The screenshot shows a terminal window titled "1: Python Det". The user has typed the following sequence:

```
Digite o primeiro número:  
15  
Digite o segundo número:  
10  
Escolha uma operação:  
Digite 1 para soma;  
Digite 2 para subtração;  
Digite 3 para Multiplicação;  
Digite 4 para Divisão.  
0  
Opção inválida.
```

At the bottom of the terminal, the status bar displays: "Ln 24, Col 29" and "Spaces: 4". To the right of the status bar are icons for "UTF-8", "CRLF", "Python", and a refresh symbol.

Figura 9 - Resultado da execução do programa do Quadro 3 para opção inválida (opção diferente de 1, 2, 3 e 4, simultaneamente) / Fonte: o autor.

Descrição da Imagem: nesta imagem exibe-se o resultado da execução do programa do Quadro 3. A sequência de mensagens exibida pelo programa, e a respectiva resposta do usuário ocorreram na seguinte ordem: primeiramente o programa imprime a mensagem “Digite o primeiro número.” e então o usuário responde com o dado 15. Na sequência, o programa mostra a mensagem “Digite o segundo número.” e aí o usuário insere o dado 10. Em seguida, o programa mostra mensagens para que o usuário digite o número 1, caso queira realizar uma soma; número 2, caso queira subtração; número 3 em caso de multiplicação; ou que digite 4 para realizar uma divisão. Após isso, o usuário escolhe a opção 0 e então o programa finaliza com a mensagem “Opção inválida.”.

Você provavelmente está se perguntando se é possível que, quando o usuário seleciona a opção 4, ocorra uma divisão por zero. Da forma como o programa foi escrito no Quadro 3, não estamos tratando desse possível erro. Por isso, o programa do Quadro 4, a seguir, mostra uma solução que impede a ocorrência de um erro de execução pela ocorrência de uma divisão por zero.

```
01 num1 = float(input("Digite o primeiro número:\n"))
02 num2 = float(input("Digite o segundo número:\n"))
03
04 print("Escolha uma operação:")
05 print("Digite 1 para soma;")
06 print("Digite 2 para subtração;")
07 print("Digite 3 para Multiplicação;")
08 print("Digite 4 para Divisão.")
09 op = int(input())
10
11 if op == 1:
12     resultado = num1 + num2
13     print("A soma é:", resultado)
14 elif op == 2:
15     resultado = num1 - num2
16     print("A subtração é:", resultado)
17 elif op == 3:
18     resultado = num1 * num2
19     print("A multiplicação é:", resultado)
20 elif op == 4:
21     if num2 != 0:
22         resultado = num1 / num2
23         print("A divisão é:", resultado)
24     else:
25         print("Você tentou realizar uma divisão por 0.")
26 else:
27     print("Opção inválida.")
```

Quadro 4 - Estruturas de decisão aninhadas para resolver o problema da divisão por 0
Fonte: o autor.

Repare, no Quadro 4, que foram adicionados comandos `if` e `else` entre as linhas de código 21 e 25. Repare que os comandos das linhas 21 e 22 estão indentados em relação ao comando `elif` da linha 20. Assim sendo, todo o bloco de comandos que vai das linhas 21 a 25 está condicionado à instrução da linha 20. Além disso, os comandos das linhas 22 e 23 estão indentados em relação à linha 21, o que faz com que as linhas 22 e 23 estejam, por sua vez, condicionadas à linha 21, além da linha 20 já mencionada anteriormente. O mesmo raciocínio se aplica ao comando da linha 25 que está subordinado ao `else` da linha 24. Para verificar

a funcionalidade desta nova versão, presente no Quadro 4, é recomendável que você experimente executar o programa, inserindo dados que levem à divisão por zero, para testar o código. Você não irá aprender, se não testar na prática!

CONSIDERAÇÕES FINAIS

Estruturas de decisão permitem que os desenvolvedores criem algoritmos nos quais um conjunto de comandos depende da avaliação de uma expressão condicional.

Quando se trata de estruturas de decisão simples, o bloco de comandos será executado somente se a expressão foi avaliada como verdadeira. Se, eventualmente, em outra execução do programa, a mesma expressão for avaliada como falsa, então o respectivo bloco de comandos é ignorado.

Ainda vimos a estrutura de seleção composta que, da mesma forma, depende de uma condição para escolher entre a execução de um entre dois blocos de comandos. Nesse caso, existem, obrigatoriamente, dois caminhos a serem seguidos pelo programa. Quando a expressão é avaliada como verdadeira, o bloco de comandos associado ao `if` é executado, sendo que o bloco de comandos do `else` é ignorado. Em outras ocasiões, quando a condição é avaliada como falsa, o bloco de instruções do `if` é ignorado e executa-se o bloco do `else`.

Além disso, foi possível ter contato com o caso em que é possível combinar o `if` com o `else`, obtendo o comando `elif`. Vimos que esse comando é comumente utilizado em casos em que existem condições que se excluem mutuamente.

Tivemos condições comparar cada uma das estruturas de condição e como elas podem estar associadas à rapidez de um programa. Construímos programas utilizando essas estruturas e tivemos contato com exemplos de expressões lógico-relacionais que compõem as condições das estruturas de decisão.

Assim sendo, nesta unidade pudemos colocar em prática esse novo conhecimento, associando-o a operações de entrada e saída de dados, variáveis, atribuição e outros conceitos vistos anteriormente. Por isso, nota-se a importância de se praticar a construção de programas, percebendo que quando se trata de lógica de programação, os conteúdos são cumulativos.



1. Faça um programa que leia um número e informe se ele é divisível por cinco.
2. Elabore um programa que receba o nome e a idade de uma pessoa e informe o nome, a idade e o valor da mensalidade do plano de saúde. A tabela a seguir apresenta os valores de mensalidade com base na faixa etária.

Até 18 anos	R\$ 50,00
De 19 a 29 anos	R\$ 70,00
De 30 a 45 anos	R\$ 90,00
De 46 a 65 anos	R\$ 130,00
Acima de 65 anos	R\$ 170,00

Fonte: o autor.

3. Construa um programa que receba a idade de uma pessoa e identifique sua classe eleitoral: não eleitor (menor que 16 anos de idade), eleitor obrigatório (entre 18 e 65 anos) e eleitor facultativo (entre 16 e 18 e maior que 65).



4. De acordo com uma tabela médica, o peso ideal está relacionado com a altura e o sexo de uma pessoa. Elabore um algoritmo que receba altura e sexo de uma pessoa e calcule e imprima seu peso ideal, sabendo que:
 - a) Para homens o peso ideal é igual a 72,7 multiplicados pela altura, subtraídos por 58.
 - b) Para mulheres o peso ideal é igual a 62,1 multiplicados pela altura, subtraídos por 44,7.
5. Faça um programa que informe a quantidade total de calorias a partir da escolha do usuário, que deve informar o prato típico e a bebida. A tabela de calorias encontra-se a seguir.

Prato	Bebida
Italiano	750 cal Chá 30 cal
Japonês	324 cal Suco de laranja 80 cal
Salvadorenho	545 cal Refrigerante 90 cal

Fonte: o autor.

6. É comum, em uma aplicação, ter de determinar quais números são o maior ou o menor, dentre todos os valores de um conjunto de dados informados pelo usuário. Assim sendo, escreva um programa que receba cinco números inteiros e apresente o maior e o menor entre todos eles.



7. Em algumas situações, em uma aplicação, é preciso determinar quais são os números múltiplos de um ou mais valores, dentre todos os valores de um conjunto de dados. Dessa forma, faça um programa que receba, do usuário, um número e informe se ele é divisível por três ou por sete.
8. Considere uma aplicação que necessita de um calendário embutido. Assim sendo, construa um programa que, dado um número inteiro, informe o mês correspondente, por extenso.
9. Elabore um programa que receba o salário de um funcionário e o código do cargo e apresente o cargo, o valor do aumento e o novo salário. A tabela a seguir apresenta os respectivos código, cargo e percentual do aumento para cada tipo de colaborador.

Código	Cargo	Percentual do aumento
1	Servente	40%
2	Pedreiro	35%
3	Mestre de obras	20%
4	Técnico de segurança	10%

Fonte: o autor.



10. Faça um programa que receba o código do estado de origem da carga de um caminhão, o peso da carga em toneladas e o código dela.

Código do estado	Taxa de imposto
1	20%
2	15%
3	10%
4	5%

Fonte: o autor.

Código da carga	Preço por quilo
10 a 20	180
21 a 30	120
31 a 40	230

Fonte: o autor.

Tomando por base as tabelas acima, calcule e apresente: o peso da carga em quilos, o preço da carga, o valor do imposto e o valor total da carga (preço da carga mais imposto).



As estruturas de seleção são a base para a criação de sistemas que podem auxiliar os seres humanos no processo de tomada de decisão. Existe uma classe especial de sistemas que têm esta característica: são os ditos **Sistemas Especialistas**.

No artigo a seguir, você conhecerá um pouco mais deste tipo de sistema. A expressão inteligência artificial está associada, geralmente, ao desenvolvimento de sistemas especialistas. Estes sistemas, baseados em conhecimento, construídos, principalmente, com regras que reproduzem o conhecimento do perito, são utilizados para solucionar determinados problemas em domínios específicos. A área médica, desde o início das pesquisas, tem sido uma das áreas mais beneficiadas pelos sistemas especialistas, por ser considerada detentora de problemas clássicos com todas as peculiaridades necessárias para serem instrumentalizados por tais sistemas.

Nem todos os problemas devem ser resolvidos por meio de sistemas especialistas. Existem características que indicam se determinado problema deve ou não ser instrumentalizado por esta tecnologia. A análise do problema, então, constitui-se no primeiro estágio do ciclo de desenvolvimento dos sistemas especialistas, contribuindo fortemente para o sucesso da implementação do sistema. Buscando facilitar o processo de análise do problema, distinguimos, dentre outras, algumas condições, que, se observadas, poderão contribuir para a identificação do nível de adequação do uso da tecnologia de sistemas especialistas para a sua resolução:

- Existência de peritos que dominem o segmento do conhecimento que encerra o problema, pois é exatamente esse conhecimento que será o responsável direto pela resolução do problema;
- Existência de tarefas que, para serem realizadas, necessitam da participação de vários especialistas que, isolados, não possuem conhecimentos suficientes para realizá-la, ou seja, o conhecimento necessário para a análise e resolução do problema é multidisciplinar;
- Existência de tarefas que requeiram conhecimento de detalhes que, se esquecidos, provocam a degradação do desempenho;



- Existência de tarefas que demonstram grandes diferenças entre o desempenho dos melhores e dos piores peritos;
- Escassez de mão de obra especializada no conhecimento requerido para a solução do problema.

Com a emergência desta técnica, evidenciaram-se alguns importantes aspectos, até então inexplorados, por exemplo, o aumento significativo da produtividade de um especialista, na execução de tarefas especializadas, quando assistido por um sistema inteligente.

Outro aspecto relevante é a portabilidade destes sistemas especialistas, por serem passíveis de desenvolvimento e utilização em microcomputadores. Isto os torna bastante populares e acessíveis. Em geral, os sistemas com raciocínio automatizado podem ser utilizados incorporando bancos de dados já existentes na organização ou sendo incorporados ao conjunto de ferramentas disponíveis nos bancos de dados.

Sob nosso ponto de vista, a ciência da informação e muitas outras áreas podem encontrar, nos sistemas especialistas, eficientes ferramentas para o gerenciamento da informação. Disponibilizar ferramentas para suporte à tomada de decisão, neste caso, vai mais além do que fornecer gráficos e tabelas ao usuário: significa prestar-lhe orientação, na identificação de suas necessidades, simulando cenários e possibilitando maior exatidão e confiabilidade nos seus resultados.

Fonte: adaptado de Mendes (1997, on-line).



eu recomendo!



conecte-se

É imprescindível que você tenha contato com o maior número de exemplos possível. A internet é um lugar vasto, e existem diversos materiais gratuitos disponíveis para seus estudos. No site a seguir, você encontra mais exemplos de programas com comandos if, else e elif





anotações



ESTRUTURAS DE REPETIÇÃO

PROFESSOR

Me. Pietro Martins de Oliveira

PLANO DE ESTUDO

A seguir, apresentam-se as aulas que você estudará nesta unidade:

- Estrutura de repetição
- Estrutura while
- Estrutura for
- Exemplos de programas com estruturas de repetição.

OBJETIVOS DE APRENDIZAGEM

- Entender as estruturas de repetição.
- Estudar a estrutura `while`.
- Compreender a estrutura `for`.
- Aplicar os comandos `while` e `for` na solução de problemas.

INTRODUÇÃO



Nesta unidade, veremos como construir programas que façam a repetição de trechos de código, de forma automática. Quando usamos estruturas de repetição, o objetivo é que sejamos capazes de fazer com que o programa execute um bloco de comandos repetidas vezes, sem que sejamos obrigados a replicar o código-fonte.

É uma boa prática de programação evitar códigos repetitivos, incentivando o reuso, bem como a legibilidade e a manutenibilidade do código-fonte de um programa, independentemente da linguagem de programação. Sendo assim, a partir daqui, desenvolveremos um raciocínio voltado a identificar blocos de comandos que são repetitivos e cuja repetição pode ser automatizada por meio de estruturas de repetição.

Veremos, basicamente, dois grandes tipos de laços de repetição. Nos laços condicionais, geralmente, não se sabe, de antemão, o número de execuções a serem iteradas, por isso, condicionam-se as repetições a uma expressão lógico-relacional. Para isso, utilizamos o comando `while`, que repetirá o seu bloco de comandos, enquanto a expressão condicional for verdadeira.

Estruturas de repetição por contagem são utilizadas quando se sabe, de antemão, quantas vezes o bloco de comandos é iterado. Na linguagem Python, a estrutura de repetição com laço contado é criada a partir do comando `for`. Sua sintaxe permite que uma variável de controle seja inicializada, que seja estabelecido um critério de parada e, também, que seja possível estabelecer como essa variável de controle será atualizada, seja com o incremento, seja com o decremento de seu valor. Além disso, também é possível utilizar o `for` para navegar por elementos de estruturas de dados multidimensionais, como veremos, futuramente.

Para facilitar o seu aprendizado, construiremos programas utilizando cada um desses comandos. Ao final desta unidade, você poderá construir códigos-fonte com estruturas de repetição, podendo automatizar diversos processos repetitivos.



1 ESTRUTURA DE REPETIÇÃO

Há situações, em nossos programas, em que precisamos repetir determinado trecho de código ou todo o código do programa, por várias vezes, em sequência. Nestes casos, utilizaremos uma estrutura de repetição que nos permite criar um laço, um *loop* para efetuar o processamento de um trecho de código, quantas vezes forem necessárias. Na literatura, essas estruturas de repetição (*loops*) são, também, denominadas laços de repetição e malhas de repetição (LEAL; OLIVEIRA, 2020).

A vantagem da estrutura de repetição é que não precisamos reescrever trechos de código idênticos, reduzindo, assim, o tamanho do algoritmo. Além disso, podemos determinar repetições com um número de vezes variável (LEAL; OLIVEIRA, 2020).

Nas estruturas de repetição, o número de repetições pode ser fixo ou estar relacionado a uma condição, isto é, os laços de repetição podem ser classificados em laços contados e condicionais. O laço contado é indicado para quando conhecemos, previamente, o número de iterações que precisa ser realizado (LEAL; OLIVEIRA, 2020). Em contrapartida, em um laço condicional, esse número de iterações é desconhecido e, para controlar o *loop*, utilizamos uma condição lógico-relacional.

Em Python, utilizamos os comandos `while` e `for` para criar estruturas que possibilitam a repetição de blocos de comandos. Assim sendo, estudaremos cada uma dessas estruturas de repetição do Python, destacando as suas sintaxe e aplicação.



2 ESTRUTURA WHILE

A estrutura while é um laço que depende de um critério de parada formado por uma expressão lógico-relacional. Essa estrutura é bastante intuitiva e, geralmente, é utilizada quando temos um número indefinido de repetições e, também, se caracteriza por realizar, logo no início do bloco de comandos, um teste condicional. Por causa disso, podem ocorrer casos em que as instruções da estrutura de repetição nunca sejam executadas. Isso acontece quando o teste condicional da estrutura resulta, logo na primeira comparação, em falso (LEAL; OLIVEIRA, 2020).

A sintaxe do while é:

```
while <condição>:  
    <instruções>
```

Os comandos delimitados pelo campo <instruções> devem ser indentados em relação à linha e à coluna nas quais se encontra o comando while, isso fará com que os respectivos comandos estejam subordinados à expressão lógico-relacional contida no campo <condição>. O bloco de <instruções> será executado enquanto a <condição> se mantiver verdadeira, e analisaremos, por meio de exemplos de código-fonte, o funcionamento dessa estrutura.

Para facilitar o entendimento, construiremos um programa que efetua a leitura do nome de uma pessoa e, em seguida, o imprime, por dez vezes, na tela. Para isso, teremos uma variável de controle que será iniciada com o valor 0, uma condição de parada que fará com que as repetições se interrompam quando a variável atingir o valor 10, e, por fim, incrementaremos a variável de controle de uma em uma unidade, a cada nova repetição.

O Quadro 1 mostra como esse programa pode ser implementado em linguagem Python.

```
01 nome = input("Informe o nome: ")
02
03 print("Imprimindo 10x:")
04 i = 0
05 while i < 10:
06     print(nome)
07     i += 1
```

Quadro 1 - Exemplo do uso da estrutura `while` em Python / Fonte: o autor.

A Figura 1, a seguir, ilustra o resultado da execução do programa, em que o laço é repetido até que a expressão lógico-relacional `i < 10` se torne falsa.

```
Informe o nome: Jeremias
Imprimindo 10x:
Jeremias
```

Figura 1 - Resultado da execução do programa do Quadro 1 / Fonte: o autor.

Descrição da Imagem: Na ilustração, o programa do Quadro 1 é executado, e o terminal embutido do Visual Studio Code mostra a impressão das seguintes mensagens, em ordem: "Informe o nome" (então, o usuário insere o dado "Jeremias"); "Imprimindo 10x:", "Jeremias", "Jeremias", "Jeremias", "Jeremias", "Jeremias", "Jeremias", "Jeremias", "Jeremias", "Jeremias". Feito isso, o programa se encerra.

Neste ponto, você pode estar se perguntando o porquê de a variável `i` não ser inicializada com 1. Consegue imaginar? Ocorre que, quando o valor de `i` é igual a 10, as instruções contidas no interior do laço não são executadas, pois a expressão relacional resulta em falso. Caso contrário, teríamos a impressão do nome apenas 9 vezes (de 1 até 9, inclusive). Se quiséssemos inicializar a variável `i` com 1, teríamos que alterar a expressão da linha de código 05, do Quadro 1, para algo como `i <= 10` (LEAL; OLIVEIRA, 2020).

Outro detalhe importante de notarmos: é imprescindível que, dentro do bloco de comandos da estrutura `while`, tenhamos alguma expressão de atualização do conteúdo da variável de controle. Repare que, na linha de código 07, temos a expressão `i += 1`, o que fará com que a variável `i` seja incrementada em uma unidade, a cada nova repetição. Essa expressão, em conjunto com a inicialização da linha 04, e a condição de parada da linha 05, fará com que o nosso laço não se repita, indefinidamente, evitando uma situação-problema que chamamos de **loop infinito**.



ESTRUTURA FOR



O `for` é um tipo de laço que pode ser considerado uma estrutura de repetição por contagem. Intuitivamente, é mais comum utilizar essa estrutura quando se sabe, de antemão, o número de vezes que o trecho de código precisa ser repetido.

Em Python, para realizar a iteração de blocos de comandos com o laço contendo `for`, precisaremos utilizar a função embutida `range()`. Esta função gera uma série numérica com `inicio`, `fim` e um `passo`. Assim, a sintaxe da função `range` é a seguinte:

```
range(<início>, <fim>, <passo>)
```

O campo `<início>` determina qual é o número inicial da série, incluindo esse número na própria série. Já o campo `<fim>`, delimita qual será o último termo da série, neste caso, excluindo o número representado no campo `<fim>`. Finalmente, o campo `<passo>` deixa explícito qual será a razão na qual a série numérica evolui do `<início>` (*inclusive*) até o `<fim>` (*exclusive*).

Por exemplo, caso seja necessário gerar uma série numérica de números inteiros que vá de 0 até 9, incrementando cada termo, em uma unidade em relação ao anterior, devemos invocar a função da seguinte forma: `range(0, 10, 1)`. Ou seja, neste caso, o `<início>` é igual a 0, o `<fim>` é igual a 10, e, de um termo para o outro, temos uma unidade de diferença: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Caso não esteja claro, ainda, fique em paz, pois veremos como essa função atua, em conjunto, com o comando `for`.

A sintaxe desta estrutura é:

```
for <variável> in <sequência>:  
    <instruções>
```

O campo `<variável>` deve ser substituído pelo identificador da variável de controle, que, por sua vez, terá o seu conteúdo atualizado a cada nova iteração do bloco de comandos definido pelo campo `<instruções>`. Já o campo `<sequência>` pode ser substituído pela função `range()`, que ditará quais serão os valores inicial, intermediário e final que a variável de controle assumirá a cada nova repetição. Repare que, mais uma vez, o campo `<instruções>` está indentado em relação à linha e à coluna do `for`, fazendo com que todos os comandos contidos nesse campo sejam subordinados e, consequentemente, iterados na estrutura de repetição.



Ambos, `for` e `while`, são estruturas de repetição. Você acha que, em Python, tudo o que fazemos com `while`, conseguimos fazer com `for`, e vice-versa?

Para verificarmos um exemplo de como a estrutura `for` se comporta em nossos programas, podemos usar o mesmo exemplo mostrado anteriormente, que pede o nome da pessoa e o imprime, por dez vezes, na tela. Observe como podemos criar uma nova versão do código-fonte do Quadro 1, porém, agora, usando o comando `for` ao invés do `while`. Segue a nova versão do programa no Quadro 2, a seguir.

```
01     nome = input("Informe o nome: ")  
02  
03     print("Imprimindo 10x:")  
04     for i in range(0, 10, 1):  
05         print(nome)
```

Quadro 2 - Exemplo da utilização do comando `for` em Python / Fonte: o autor.

O resultado da execução do programa do Quadro 2 pode ser visualizado na Figura 2, a seguir.

Figura 2 - Resultado da execução do programa do Quadro 2 / Fonte: o autor.

O Quadro 2 mostra como, na linha 04, temos, ao mesmo tempo, a definição da variável `i` para controlar as repetições; o início da contagem em 0; o término da contagem em 10 (*exclusive*); e o incremento de `i` em uma unidade por repetição. Assim sendo, temos condições de perceber que, de fato, o laço `for`, em conjunto com a função `range()`, permite que realizemos repetições contadas com base em uma série numérica, como dito anteriormente.



EXEMPLOS DE PROGRAMAS COM ESTRUTURAS DE REPETIÇÃO

Para que você tenha um pouco mais de confiança para aplicar os comandos `while` e `for`, veremos algumas situações-problema e as suas respectivas soluções. Leia o enunciado com calma, analise o código-fonte para ver como a solução foi criada, e o mais importante: teste o código dos programas, fazendo alterações e tentando compreender a sua lógica.

PROBLEMA 1

O problema consiste em ler um conjunto de números inteiros e contar a quantidade de números ímpares e pares. A leitura deve ser realizada até que seja lido o valor zero (LEAL; OLIVEIRA, 2020).

A entrada de dados consiste na leitura de número inteiros repetidas vezes, até que o valor 0 seja digitado. O processamento é contar a quantidade de números pares e ímpares, a saída é informar quantos números lidos durante a entrada são pares e quantos são ímpares. No Quadro 3, a seguir, há um código-fonte que resolve esse problema, utilizando a estrutura de repetição condicional `while` (LEAL; OLIVEIRA, 2020).

```
01     par = 0
02     impar = 0
03
04     num = int(input("Informe um número:\n"))
05
06     while num != 0:
07         if num % 2 == 0:
08             par = par + 1
09         else:
10             impar = impar + 1
11     num = int(input("Informe um número:\n"))
12
13     print("A quantidade de pares é:", par)
14     print("A quantidade de ímpares é:", impar)
```

Quadro 3 - Primeira solução, em Python, para o Problema 1/ Fonte: o autor.

Repare que, como a estrutura `while` obriga que o teste condicional seja feito, imediatamente, antes de tentar executar qualquer instrução do bloco de comandos, que vai das linhas 07 a 11, é preciso pedir ao usuário que insira o valor da variável `num`, na linha 04. Além disso, para os próximos números a serem lidos, somos obrigados a replicar uma linha de código idêntica (linha 11).

O que você acha que aconteceria se não tivéssemos a leitura do número dentro do laço de repetição? Neste caso, teríamos um laço infinito (*loop* infinito), pois a condição `num != 0` sempre resultaria em verdadeiro. Isso ocorreria porque a variável `num` nunca mais teria o seu conteúdo alterado, eliminando a chance de ela, em algum momento, valer 0 para encerrar as repetições (LEAL; OLIVEIRA, 2020).

Pensando nisso, para que o código fique um pouco mais coerente, criaremos uma nova versão de código para esse problema, que permite que a leitura de `num` seja feita apenas dentro do laço.

Observe, no Quadro 4, a seguir, que a única operação de entrada de dados está na linha 05, dentro do bloco de comandos, condicionado ao `while` da linha 04. Repare, ainda, que a condição da linha de código 04 é, na verdade, o valor lógico `True`, o que significa que, em tese, esse laço se repetirá infinitamente, todavia o critério de parada está nas linhas 06 e 07. Observe que, caso o usuário insira um valor de `num` igual a 0, na linha 05, ao executar a linha 06, teremos que a condição `num == 0` é verdadeira e, assim, o comando de interrupção de laço, o `break`, será acionado, encerrando as iterações do bloco de comandos, que vai das linhas 04 a 11.

```
01     par = 0
02     impar = 0
03
04     while True:
05         num = int(input("Informe um número:\n"))
06         if num == 0:
07             break
08         elif num % 2 == 0:
09             par = par + 1
10         else:
11             impar = impar + 1
12
13     print("A quantidade de pares é:", par)
14     print("A quantidade de ímpares é:", impar)
```

Quadro 4 - Segunda solução, em Python, para o Problema 1 / Fonte: o autor.

Vale destacar que ambas as soluções, tanto a do Quadro 3 quanto a do Quadro 4, resolvem o mesmo problema, de formas distintas, ou seja, podem existir códigos-fonte distintos para resolver uma mesma situação. Vale observar que a diferença mais expressiva, aqui, está no uso do comando `break`, na segunda solução, o que pode ser muito útil em situações como essa.

PROBLEMA 2

O problema consiste em auxiliar um professor no fechamento das notas de uma turma. Para tanto, deve ser construído um programa que leia as quatro notas da disciplina para uma turma com um total de dez alunos. Para cada aluno, deve-se informar ao programa o RA e as quatro notas. Calcule a média final de cada aluno e informe o número de alunos aprovados e reprovados. Para ser aprovado, o estudante precisa obter média maior ou igual a seis (LEAL; OLIVEIRA, 2020).

Assim sendo, podemos analisar como a solução será construída. Como entrada de dados, para cada aluno, temos de efetuar a leitura do seu RA e das respectivas notas. Enquanto são lidos o RA e as quatro notas de cada aluno, deve-se elaborar uma lógica de processamento que calcule a média de cada aluno e contabilize se este estudante foi aprovado, ou não, caso a respectiva média seja maior ou igual a seis, ou não. Ao final, a saída de dados deve mostrar quantos alunos foram aprovados e quantos foram reprovados.

O Quadro 5, a seguir, mostra uma possível solução para esse problema, por meio da estrutura de repetição `for`.

```
01 aprovados = 0
02 reprovados = 0
03
04 for i in range(0,10,1):
05     print("Informe o RA do aluno", i+1, ":")
06     RA = int(input())
07     soma = 0
08     for j in range(0,4,1):
09         print("Informe a nota", j+1, "do aluno de")
10         RA, RA, ":")

11         nota = float(input())
12         soma = soma + nota
13     media = soma/4
14     if media >= 6:
15         aprovados = aprovados + 1
16     else:
17         reprovados = reprovados + 1
18
19 print("O número de aprovados é", aprovados, ".")
20 print("O número de reprovados é", reprovados, ".")
```

Quadro 5 - Solução, em Python, para o Problema 2 / Fonte: o autor.

Esse exemplo é, particularmente, importante, pois utiliza duas estruturas de repetição aninhadas, ou seja, temos “um `for` dentro do outro”. Assim, para cada iteração do laço definido, a partir da linha 04, o laço que se inicia na linha 08 será repetido quatro vezes. Assim, teremos, na verdade, $10 \times 4 = 40$ repetições do bloco de comandos, que vai das linhas 08 a 12. Isso ocorre pois, para cada aluno, precisamos ler quatro notas, se temos dez alunos, então, existirão 40 notas.

Note, ainda, que a indentação é de extrema importância, em Python, para identificarmos quais comandos estão condicionados a quais respectivas estruturas. Experimente executar todos os códigos desta unidade, para entender melhor como as lógicas de cada programa se comportam, em tempo de execução.



conecte-se

Caso tenha interesse em ter contato com explicações diferentes sobre estruturas de repetição em Python, assista ao vídeo indicado.



CONSIDERAÇÕES FINAIS

Nesta unidade, você aprendeu a construir programas utilizando estruturas de repetição, que, por sua vez, permitem a execução de um trecho de código, repetidas vezes. Essas estruturas também são conhecidas como *loops* ou laços de repetição.

Estudamos os laços de repetição que levam, como base, uma condição, ou que se apoiam em uma série numérica para controlar as repetições. Nos laços condicionais, utilizamos o comando `while`. O laço de repetição condicional, `while`, geralmente, é empregado quando o programador não pode prever, de antemão, a quantidade de iterações que serão executadas pela máquina. Assim, impõe-se uma condição que é testada logo no início da estrutura. Vimos, também, um exemplo de situação na qual a parada das repetições do laço é realizada com o comando `break`.

Em se tratando de repetições baseadas em sequências numéricas contadas, temos o comando `for`, em conjunto com a função `range()`. Essa estrutura de repetição é mais indicada quando sabemos, de antemão, quantas vezes o trecho de código deve se repetir. A contagem em si fica a cargo da função `range()`, que delimita um valor de início e um valor final e, também, qual é a razão na qual a sequência numérica se desenvolve. Basicamente, utilizar o `for`, em conjunto com o `range()`, é, simplesmente, criar uma série numérica. O comando `for` será particularmente útil em situações nas quais precisamos percorrer estruturas de dados com uma ou mais dimensões.

Ao longo desta unidade, construímos algoritmos utilizando todos os conceitos apresentados e, também, discutimos particularidades de cada estrutura de repetição, enfatizando as situações nas quais cada uma delas é mais comumente empregada. Em se tratando de estruturas de repetição, é ainda mais imprescindível que você treine bastante, execute os programas. Tente fazer os exercícios, procure materiais auxiliares na biblioteca virtual e, também, na internet. Não assuma que aprenderá de forma passiva, pois, quando falamos em programação, isso não existe.



1. Faça um programa que leia um número inteiro e, depois, calcule e mostre o seu respectivo fatorial.
2. Elabore um programa que apresente todos os números divisíveis por três que sejam menores que 100.
3. Construa um programa que receba um número inteiro maior do que um e verifique se ele é primo.
4. A prefeitura de uma cidade inteligente está coletando informações sobre o salário e o número de filhos dos habitantes. A leitura dos dados é realizada até que seja informado o valor -1 por parte do usuário, para o salário. Apresente a média de salário da população informada, a média de filhos e o maior salário.
5. Escreva um programa que receba a idade e a altura de várias pessoas, em sequência. Calcule e apresente a média de altura e idade das pessoas. A entrada de dados deve ser encerrada se, e somente se, for digitado 0 para o valor de idade.



6. Na avaliação de um produto, o cliente responde a sua opinião (1- satisfatório; 2-in-diferente; 3-insatisfatório). Faça um programa que leia a idade e a opinião e, em seguida, apresente: o número de clientes que responderam satisfatório; a média de idade dos clientes que opinaram como satisfatório; o número de clientes que responderam insatisfatório. O programa se encerra quando for digitado o valor 0 para idade.
7. Faça um programa que leia números inteiros até que seja informado o valor 0. Apresente a média dos valores, o maior e o menor valor e a quantidade de números pares e ímpares.
8. Para os seres humanos, trabalhar com séries numéricas, sem o auxílio de ferramentas computacionais, pode ser um grande transtorno. Assim sendo, construa um programa que leia o número de termos da série e imprima o valor de S, sendo que:

$$S = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \dots + \frac{1}{N}$$



9. Imagine que você esteja ensinando a tabuada para uma criança e precisa mostrar a ela todas as possíveis multiplicações entre os números que vão de 1 até 10. Assim, elabore um programa que imprima a tabuada de 1 a 10.
10. É muito comum que programas tenham de implementar funcionalidades para identificar características específicas e realizar operações sobre um conjunto de dados. Dessa forma, faça um programa que apresente a soma de todos os números inteiros ímpares entre 200 e 500.
11. Para classificar um conjunto de dados, é comum que programadores utilizem estruturas de decisão associadas a estruturas de repetição. Assim sendo, construa um programa que apresente todos os números divisíveis por 3 e por 7 que sejam menores que 30.
12. Nada melhor do que uma máquina para realizar tarefas repetitivas, evitando esforço humano desnecessário. Assim, elabore um programa que leia uma frase e o número de vezes que deseja imprimi-la. Ao final, imprima a frase por tantas vezes que o usuário desejar.



13. Faça um programa que leia um conjunto de pedidos e calcule o total da compra. O pedido possui os seguintes campos: número, data (dia, mês e ano), preço unitário e quantidade. A entrada de pedidos é encerrada quando o usuário informa 0 como número do pedido.
14. Elabore um programa que receba a idade, o peso, o sexo e o estado civil de várias pessoas e imprima a quantidade daquelas que são casadas, solteiras, separadas e viúvas. Apresente a média de idade e de peso. O algoritmo finaliza quando for informado o valor 0 para idade.
15. Construa um programa que dê a possibilidade de o usuário calcular a área total de uma residência (sala, cozinha, banheiro, quartos etc.). O programa deve solicitar a entrada do nome, a largura e o comprimento de determinado cômodo até que o nome do cômodo informado seja igual a FIM. O programa deve apresentar o valor total acumulado da área residencial.



Você sabia que estruturas de repetição podem ser omitidas em alguns algoritmos repetitivos, sem que seja necessário replicar código-fonte, desnecessariamente? É possível, de certa forma, “imitar” as iterações (repetições) dessas estruturas de repetição, utilizando o conceito de recursividade. Para entender um pouco mais sobre a relação entre iterações e recursão, acompanhe o trecho do artigo que segue.

“Um objeto é denominado recursivo quando sua definição é parcialmente feita em termos dele mesmo. A **recursividade** (ou recursão) é encontrada principalmente na matemática, mas está presente em algumas situações do cotidiano. Por exemplo, quando um objeto é colocado entre dois espelhos planos paralelos e frente a frente surge uma imagem recursiva, porque a imagem do objeto refletida num espelho passa a ser o objeto a ser refletido no outro espelho e, assim, sucessivamente.

Em programação, a recursividade é um mecanismo útil e poderoso que permite a uma função chamar a si mesma direta ou indiretamente, ou seja, uma função é dita recursiva se ela contém pelo menos uma chamada explícita ou implícita a si própria.

A ideia básica de um algoritmo recursivo consiste em diminuir sucessivamente o problema em um problema menor ou mais simples, até que o tamanho ou a simplicidade do problema reduzido permita resolvê-lo de forma direta, sem recorrer a si mesmo. Quando isso ocorre, diz-se que o algoritmo atingiu uma **condição de parada**, a qual deve estar presente em pelo menos um local dentro do algoritmo. Sem esta condição, o algoritmo não para de chamar a si mesmo, até estourar a capacidade da pilha, o que geralmente causa efeitos colaterais e até mesmo o término indesejável do programa.



Para todo algoritmo recursivo existe um outro correspondente iterativo (não recursivo), que executa a mesma tarefa. Implementar um algoritmo recursivo, partindo de uma definição recursiva do problema, em uma linguagem de programação de alto nível, como o Python, é simples e quase imediato, pois o seu código é praticamente transscrito para a sintaxe da linguagem. Por essa razão, em geral, os algoritmos recursivos possuem código mais claro (legível) e mais compacto do que os correspondentes iterativos. Além disso, muitas vezes, é evidente a natureza recursiva do problema a ser resolvido, como é o caso de problemas envolvendo grafos e árvores – estruturas de dados naturalmente recursivas. Entretanto também há desvantagens:

- I - algoritmos recursivos quase sempre consomem mais recursos (especialmente memória, devido uso intensivo da pilha) do computador, logo tendem a apresentar um desempenho inferior aos iterativos;
- II - algoritmos recursivos são mais difíceis de serem depurados, especialmente quando for alta a profundidade de recursão (número máximo de chamadas simultâneas)".

Fonte: adaptado de Santos (2013, p. 1).



eu recomendo!



livro

Python Fluente: Programação Clara, Concisa e Eficaz

Autor: Luciano Ramalho

Editora: Novatec

Sinopse: a simplicidade de Python permite que você se torne produtivo, rapidamente, porém, isso, muitas vezes, significa que você não estará usando tudo o que ela tem a oferecer. Com este guia prático, você aprenderá a escrever um código Python eficiente e idiomático, aproveitando os seus melhores recursos – alguns deles, pouco conhecidos. O autor Luciano Ramalho apresenta os recursos essenciais da linguagem e das bibliotecas de Python, mostrando como você pode tornar o seu código mais conciso, mais rápido e mais legível, ao mesmo tempo.





ESTRUTURAS DE DADOS BÁSICAS

PROFESSOR
Me. Pietro Martins de Oliveira

PLANO DE ESTUDO

A seguir, apresentam-se as aulas que você estudará nesta unidade: • Listas • Strings • Matrizes • Dicionários.

OBJETIVOS DE APRENDIZAGEM

Estudar o conceito de listas • Entender o conceito de strings e como manipulá-las • Dominar o conceito de matrizes e suas possíveis aplicações • Aplicar o uso de dicionários para resolver problemas e combinar estruturas de dados básicas em soluções.

INTRODUÇÃO



Nesta unidade, você estudará estruturas de dados um pouco mais poderosas do que as variáveis simples que utilizamos até o momento. As variáveis simples permitem que armazenemos, apenas, um dado por vez, para um mesmo nome de variável. A partir de agora, seremos capazes de representar diversas informações simultaneamente, usando, listas e dicionários.

Com as listas, seremos capazes de armazenar diversas informações em uma estrutura de dados dita unidimensional. Isso quer dizer que, a partir de um índice, podemos acessar distintos elementos dentro de uma região de memória identificada por um mesmo nome. Podemos considerar que as listas são similares aos vetores da matemática.

Ainda, através de uma composição de listas, é possível criar estruturas de dados multidimensionais, as quais conhecemos por matrizes na matemática. Dessa forma, através de dois ou mais índices, é possível navegar por essa estrutura, manipulando seus respectivos conteúdos de acordo com a lógica necessária para resolver nossos problemas algoritmos. Também, falaremos um pouco sobre as strings, ou o tipo de dados str que, na realidade, trata-se de dados tipo texto. Assim, quando for necessário armazenar caracteres alfanuméricos em seus programas, você poderá se apropriar desse tipo de estrutura de dados.

Por fim, veremos como utilizar os dicionários, que são estruturas de dados bastante úteis quando é necessário armazenar dados de tipos distintos. Nos dicionários, o índice a ser acessado, para manipular os dados dessa estrutura, pode ser algo como identificadores. Por hora, basta saber que essa estrutura de dados é bastante útil para se armazenar dados variados em uma região de memória identificada por um mesmo nome. Ao fim da unidade, você terá contato com as estruturas de dados básicas mais utilizadas na programação em Python. Portanto, vamos aos estudos!



1 LISTAS

Na linguagem Python, uma lista é uma espécie de dados organizados em sequência na memória do computador. Cada elemento de uma lista pode ser acessado por um único identificador. Ou seja, por mais que exista todo um conjunto de dados, todos eles ficam disponíveis através do nome da variável lista. É como se a lista fosse um amontoado de elementos dispostos de forma linear, os quais podem ser modificados ou recuperados por meio de um índice que varia de acordo com a posição na qual o dado foi inserido.

Como veremos na sintaxe a seguir, podemos identificar uma lista em Python pelo uso dos colchetes que delimitam os elementos ali armazenados. Todos os dados que estiverem dentro dos colchetes farão parte da lista em memória.

```
<nome_da_lista> = [<dados_separados_por_vírgula>]
```

Geralmente, esse tipo de estrutura é utilizado para se armazenar dados de um mesmo tipo. Por exemplo, é comum que uma lista possua, apenas, dados do tipo `int`. Todavia, devido à sua flexibilidade, ela, também, admite que sejam armazenados dados de tipos distintos em uma mesma lista. Por exemplo, poderíamos ter uma lista com dados do tipo `int`, `float` e `str` simultaneamente.

Sempre, antes de utilizarmos uma lista, é preciso instanciá-la, como se fosse uma espécie de declaração ao interpretador: “Por favor, interpretador, crie uma lista para mim”. A seguir, vemos a instanciação de uma lista vazia, ou seja, inicialmente, sem elementos:

```
lista1 = []
```

Ainda, podemos declarar a instanciação de uma lista homogênea, contendo, apenas, dados do tipo inteiro:

```
lista2 = [10, 20, 30]
```

No caso acima, na primeira posição da lista, tem-se o dado 10, na segunda posição, o dado 20 e, na última posição, o dado 30. Perceba que, como dito anteriormente, é uma lista homogênea, pois está armazenando, apenas, dados do mesmo tipo. Para fechar nosso exemplo de declaração de listas, temos, também, a possibilidade de armazenar dados de tipos distintos, como podemos ver a seguir:

```
lista3 = [3.14, 999, "teste"]
```

Repare, portanto, que a `lista3` armazena um dado do tipo `float` em sua primeira posição, um dado do tipo `int` na segunda posição e um dado do tipo `str` na última posição. É preciso tomar cuidado com a utilização de listas heterogêneas como essa, pois seu programa pode apresentar erros de incompatibilidade de tipos, dependendo da lógica necessária para se resolver o problema algorítmico.

Uma lista não possui, necessariamente, um tamanho máximo. Ou seja, quando o(a) programador(a) instancia uma lista, pode fazê-lo de forma a deixá-la vazia (com zero elementos), com 1 ou mais elementos. E, durante o tempo de execução, o(a) programador(a) pode adicionar ou remover elementos, conforme necessário.

Basicamente, existem dois métodos próprios das listas, que permitem a inserção e a remoção de elementos, respectivamente. Para se adicionar elementos, utilizamos o método `append()` que, por sua vez, irá “pendurar” o dado ao fim da mesma lista. Já o método `pop()` fará a remoção do elemento que se encontra na última posição da lista. Observe o Quadro 1 a seguir, para ver, na prática, como é possível adicionar e remover elementos no fim de uma lista.

```

01 lista = [10, 20, 30]
02 print("Lista original:", lista)
03
04 lista.append(99)
05 print("Adicionado o 99:", lista)
06
07 aux = lista.pop()
08 print("Elemento removido:", aux)
09
10 print("Resultado final:", lista)

```

Quadro 1 - Exemplo do uso dos métodos `append()` e `pop()` / Fonte: o autor.

A Figura 1, a seguir, mostra o resultado da execução do programa apresentado no Quadro 1.

The screenshot shows a terminal window with the following content:

```

OUTPUT TERMINAL ... 1: Python Debug Consc +
+ □ 🗑 ⌂ ⌂ ✎
Lista original: [10, 20, 30]
Adicionado o 99: [10, 20, 30, 99]
Elemento removido: 99
Resultado final: [10, 20, 30]
Ln 10, Col 33 Spaces: 4 UTF-8 CRLF Python ⌂ 🔍

```

Figura 1 - Resultado da execução do programa do Quadro 1 / Fonte: o autor.

Descrição da Imagem: na imagem, a seguinte sequência de mensagens é impressa no terminal embutido do Visual Studio Code, após a execução do programa contido no Quadro 1: "Lista original: [10, 20, 30]", "Adicionado o 99: [10, 20, 30, 99]", "Elemento removido: 99", "Resultado final: [10, 20, 30]".

Como poderíamos proceder, porém, em situações nas quais queremos operar elementos intermediários da lista? Digamos que uma lista tenha três elementos, e eu queria alterar o dado que está no meio, ou seja, na segunda posição. Como seria? Pois, nesse caso, devemos utilizar uma outra sintaxe que permita especificarmos, exatamente, qual é o índice da posição na qual o respectivo dado se encontra. Observe a sintaxe a seguir:

`<nome da lista>[<índice>] = <novo dado>`

É importante saber sobre o valor do índice de início e do índice final de uma lista. Em uma lista com N elementos, o primeiro deles sempre estará na posição de índice zero, e o último, na posição $N-1$. Digamos que nossa lista tenha $N = 5$ elementos: nesse caso, o primeiro elemento, como dito, estará no índice 0, já o último elemento, na posição de índice 4. Observe o exemplo do Quadro 2, no qual mostramos como acessar e alterar um elemento qualquer.

```
01 lista = [10, 20, 30]
02
03 print(lista)
04
05 lista[1] = 99
06
07 print(lista)
```

Quadro 2 - Exemplo de como se acessar elementos específicos em uma lista / Fonte: o autor.

A Figura 2, a seguir, mostra o resultado da execução do programa apresentado no Quadro 2.

Figura 2 - Resultado da execução do programa do Quadro 2 / Fonte: o autor.

Descrição da Imagem: na imagem, a seguinte sequência de mensagens é impressa no terminal embutido do Visual Studio Code, após a execução do programa contido no Quadro 2: "10, 20, 30!", "10, 99, 30!".

Algo que todo(a) programador(a) Python precisa saber, com relação aos índices das posições de uma lista, é que não é possível fazer acesso a um elemento cuja posição, ainda, não foi preenchida. Por exemplo, no programa do Quadro 2, da forma como o programa foi escrito, não seria possível, na linha de código 5, tentar acessar, por exemplo, `lista[10]`, já que, em todo o código que antecede a linha 5, a respectiva estrutura de dados não possui mais que 3 elementos. Ou seja, só seria possível, a priori, acessar os elementos que vão da posição 0 a 2.

No programa do Quadro 3, a seguir, mostra a utilização de estruturas de repetição para solicitar ao usuário que preencha uma lista de 10 elementos. Ao final, o programa imprime o índice da posição de cada elemento e o respectivo conteúdo armazenado na posição.

```
01 lista = []
02
03 for i in range(10):
04     print("Digite o elemento da posição", i)
05     dado = int(input())
06     lista.append(dado)
07
08 print("Lista preenchida:")
09
10 for i in range(10):
11     print("O elemento da posição", i, "é",
12         lista[i])
```

Quadro 3 - Pedindo para o usuário preencher os dados de uma lista / Fonte: o autor.

Um recurso bastante interessante das listas em Python é o uso de índices negativos. Ocorre que, da forma que esta linguagem foi criada, quando utilizamos um índice igual a -1, por exemplo, na realidade, estaremos acessando a última posição da lista. Se utilizarmos o índice -2, o penúltimo elemento, e assim por diante. Ou seja, tomando por base que o primeiro elemento sempre tem índice igual a 0, então, podemos considerar que a lista tem um comportamento circular, no qual a referência é o primeiro elemento. Observe o Quadro 4, a seguir, no qual temos um programa que mostra exemplos de acesso a índices negativos, bem como a uma posição inválida.

```
01 lista = []
02
03 for i in range(5):
04     print("Digite o elemento da posição", i)
05     dado = int(input())
06     lista.append(dado)
07
08 print("Lista preenchida:", lista)
09
10 print("Acessando o último elemento (-1):",
11      lista[-1])
12 print("Acessando o penúltimo elemento (-2):",
13      lista[-2])
14
15 #Erro: acessando índice fora do limite da lista
16
17 print("Tentando acessar um elemento fora da faixa (5):",
18      lista[5])
```

Quadro 4 - Trabalhando com índices negativos em listas / Fonte: o autor.



Será que é possível, em Python, combinar estruturas de dados distintas, com tipos de dados distintos, em uma mesma solução?

Para exercitar nossos conhecimentos, criaremos um programa um pouco mais complexo. O programa deverá pedir ao usuário que insira 10 números inteiros. Após isso, o programa faz a ordenação dos dados armazenados e, em seguida, solicita ao usuário que insira a chave de busca desejada. Com base nessa chave, o programa varre a lista, posição a posição, tentando encontrar o primeiro índice no qual o elemento chave estaria armazenado. Caso o programa encontre uma posição na qual o elemento é igual à chave de busca, encerra-se a execução, mostrando a respectiva posição. Todavia, caso o vetor seja verificado por completo, sem que um elemento seja igual à chave, aí o programa se encerra com a informação de que não foi possível encontrar o elemento chave.

```
01 lista = []
02
03 for i in range(10):
04     print("Digite o elemento da posição", i)
05     dado = int(input())
06     lista.append(dado)
07
08 chave = int(input("Informe o elemento que deseja buscar:
09 \n"))
10
```

```
11 i = 0
12 achou = False
13
14 while achou == False and i < 10:
15     if lista[i] == chave:
16         achou = True
17     else:
18         i += 1
19
20 if achou:
21     print("O elemento", chave, "foi encontrado",
22           na posição",
23           i, ".")
24 else:
25     print("O elemento", chave, "não foi en-contrado.")
```

Quadro 5 - Script Python para simular a busca de dados / Fonte: o autor.

Para que você fixe bem os conhecimentos adquiridos até o momento, é imprescindível que execute os programas aqui demonstrados. Assim sendo, antes de prosseguir, abra sua IDE, copie, cole e ajuste, se necessário, os códigos para que você os veja em ação. Como sugestão, comece pelo código do Quadro 3, depois, execute o programa do Quadro 4 e, por fim, entenda bem o que ocorre com o Quadro 5.



2 STRINGS

Em qualquer linguagem de programação, o conceito de *strings* remete a dados do tipo texto, ou seja, informações que não estão centradas em números matemáticos. Em suma, quando se utiliza de uma *string*, um(a) programador(a), na realidade, quer manipular caracteres alfanuméricos. Em Python, o tipo de dados relacionado a texto é o *str*.

A função *input()* que utilizamos neste livro em vários momentos, de forma nativa, reconhece os dados inseridos pelo usuário, via teclado, como sendo do tipo *str*. Por isso, em ocasiões anteriores, fomos obrigados a converter o resultado da execução do *input()* utilizando as funções *int()* ou *float()*, quando necessário. Observe um caso de utilização desse tipo de dados, no Quadro 6, a seguir.

```
01 nome = input("Digite seu nome completo:\n")
02
03 tamanho = len(nome)
04
05 print("O comprimento do nome é:", tamanho)
```

Quadro 6 - Utilizando variáveis do tipo *str* / Fonte: o autor.

No exemplo anterior, mais especificamente, na linha de código 1, temos a solicitação para que o usuário informe seu nome completo, via teclado. O nome do usuário é texto puro e está sendo armazenado na variável `nome`. Em seguida, na linha 03, a função `len()` é invocada, recebendo, como parâmetro, o conteúdo da variável que armazena o nome informado pelo usuário. A função `len()`, por sua vez, retorna o comprimento do texto inserido via teclado e armazenado esse número dentro da variável `tamanho`. Por fim, o programa do Quadro 6 se encerra mostrando ao usuário quantos caracteres ele digitou.

A função `len()` é bastante importante quando se trata de operações com *strings*. Uma outra operação muito comum com dados de texto é a concatenação, ou seja, a junção de textos, pela emenda do final de uma *string* junto ao início de outra. Para concatenar *strings*, basta utilizarmos o operador `+`, como podemos ver na linha de código 5, do programa do Quadro 7, na qual se concatenam o conteúdo da variável `nome`, um espaço em branco (delimitado por aspas duplas) e o conteúdo da variável `sobreNome`.

```
01 nome = input("Digite seu primeiro nome:\n")  
02  
03 sobrenome = input("Digite seu sobrenome:\n")  
04  
05 nome = nome + " " + sobrenome  
06  
07 print("Após concatenar as strings temos:",  
      nome)  
08  
09  
10 if sobrenome in nome:  
11     print("O sobrenome", sobrenome, "está con-  
      tido no nome",  
           nome, ".")  
12  
13  
14 print("O nome completo em minúsculo é:", nome.  
      lower())  
15  
16  
17 print("O nome completo em maiúsculo é:", nome.  
      upper())
```

Quadro 7 - Operações sobre strings / Fonte: o autor.

Ainda no Quadro 7, temos, entre as linhas 9 e 11, uma operação de busca utilizando os comandos `if` e `in`. Ou seja, caso a condição da linha 9 seja verdadeira, significa que o conteúdo da variável `sobrenome` está contido no conteúdo da variável `nome`. Ainda, nas linhas 13 e 15, tem-se o uso dos métodos `lower()` e `upper()`, os quais são responsáveis por transformar a respectiva *string* em caixa baixa ou caixa alta, respectivamente. Execute esse programa, fazendo as adaptações, caso necessário, e verifique os resultados.



3 MATRIZES



Uma matriz é uma estrutura de dados tabular ou multidimensional, com o mesmo nome e alocada, sequencialmente, em memória. O acesso aos elementos da matriz é utilizando índices, os quais podem ser referenciados diretamente ou por meio de uma expressão que resulte em um valor inteiro. Para cada dimensão da matriz, devemos ter um índice (LEAL; OLIVEIRA, 2020).

Em Python, é usual utilizar “listas de listas” para se representar matrizes. É como se cada linha (ou coluna) de uma matriz fosse uma lista à parte. Ainda assim, o Python reconhece as “listas de listas” e, para cada dimensão, é possível realizar o acesso a posições individuais dessa estrutura de dados, assim como fizemos com as listas simples. Para realizar o acesso a um elemento individual de uma matriz de múltiplas dimensões (N dimensões), temos a seguinte sintaxe:

```
<nome_da_matriz>[<índice1>] [<índice2>] ... [<índiceN>] = <dado>
```

Para ficar mais claro, observe o programa do Quadro 8, no qual se instancia uma lista denominada mat. Em seguida, pede-se para que o usuário preencha os dados da matriz, linha a linha. Ao final, acesse cada elemento da estrutura de dados de maneira individualizada.

```
01 mat = []
02
03 for i in range(3):
04     linha = []
05     for j in range(3):
06         print("Insira o elemento da linha",
07               i, "coluna", j)
08         dado = int(input())
09         linha.append(dado)
10
11     mat.append(linha)
12
13 for i in range(3):
14     for j in range(3):
15         print("O elemento da linha", i, "co-
luna", j, "é:",
16               mat[i][j])
```

Quadro 8 - Utilizando matrizes em Python / Fonte: o autor.

Observe a Figura 3, a qual o usuário insere os dados 10, 20, 30, 40, 50, 60, 70, 80 e 90, sequencialmente, quando solicitado pelo programa.

The screenshot shows a terminal window with the following interface elements:

- Top bar: OUTPUT TERMINAL ... 2: Python Debug Consc
- Right side: +, -, ^, X
- Bottom bar: ↻ Ln 1, Col 9 Spaces: 4 UTF-8 CRLF Python ⚙️ 🗑️

The terminal content is as follows:

```
Insira o elemento da linha 0 coluna 0
10
Insira o elemento da linha 0 coluna 1
20
Insira o elemento da linha 0 coluna 2
30
Insira o elemento da linha 1 coluna 0
40
Insira o elemento da linha 1 coluna 1
50
Insira o elemento da linha 1 coluna 2
60
Insira o elemento da linha 2 coluna 0
70
Insira o elemento da linha 2 coluna 1
80
Insira o elemento da linha 2 coluna 2
90
0 elemento da linha 0 coluna 0 é: 10
0 elemento da linha 0 coluna 1 é: 20
0 elemento da linha 0 coluna 2 é: 30
0 elemento da linha 1 coluna 0 é: 40
0 elemento da linha 1 coluna 1 é: 50
0 elemento da linha 1 coluna 2 é: 60
0 elemento da linha 2 coluna 0 é: 70
0 elemento da linha 2 coluna 1 é: 80
0 elemento da linha 2 coluna 2 é: 90
```

Figura 3 - Resultado da execução do programa do Quadro 8 / Fonte: o autor.

Descrição da Imagem: na imagem, o resultado da execução do programa do Quadro 8 resulta nas seguintes mensagens e interações com o usuário: primeiramente, o usuário é solicitado a inserir o elemento da linha 0, coluna 0, da matriz, e, então, é inserido o dado 10; na sequência, o usuário é convidado a preencher o dado da posição de linha 0, coluna 1, e, então, digita 20; em seguida, quando solicitado a inserir o dado da linha 0, coluna 2, o usuário insere o valor 30, preenchendo, assim, toda a primeira linha da matriz. De forma similar, o programa solicita que o usuário informe a segunda linha da matriz, que é preenchida com os valores 40, 50 e 60, para o primeiro, segundo e terceiro elementos da segunda linha, respectivamente. Por fim, o usuário insere os dados 70, 80 e 90 para a última linha. Ao final, o programa exibe todos os dados que o usuário inseriu, na mesma ordem, mostrando, para cada dado, qual é sua linha e coluna.

```
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0  
0 1 1 1 0 1 0 0 0 1 1 1 0 0 0 0  
1 0 1 0 0 1 0 1 1 0 1 1 0 1 1 0  
0 1 0 1 0 1 0 1 1 0 1 0 0 1 0 1  
1 0 0 0 0 0 0 1 0 1 1 1 0 1 1 1 0  
0 1 0 1 1 0 1 1 0 0 0 1 1 0 0 0 1  
0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 1 1  
0 0 1 1 0 1 1 0 0 1 1 1 1 1 1 1 1  
1 0 0 1 0 0 0 0 0 0 1 0 1 1 1 1 1 0  
0 1 1 0 0 1 0 0 1 0 0 0 1 0 0 1 1 0  
0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
0 0 1 1 0 1 1 0 1 1 1 0 1 1 1 0 1 1  
1 1 1 0 1 0 0 0 1 1 0 1 0 1 0 1 0 1  
1 1 1 0 1 0 0 0 1 1 0 1 0 1 0 1 0 1  
1 1 1 0 0 1 1 0 0 0 1 0 1 0 1 0 1 0 1  
0 1 1 0 0 1 0 1 1 0 0 0 0 0 0 0 0 0 0  
0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1  
0 1 1 0 0 0 1 1 0 1 1 1 0 1 0 1 0 1 0  
1 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0  
0 0 1 1 0 0 0 1 1 1 1 0 1 1 0 1 1 0 1  
1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
0 0 1 0 0 0 0 0 1 0 0 1 0 0 1 0 0 1 0 1  
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
```

Na linha 1 do Quadro 8, temos a declaração de `mat`. Em seguida, das linhas de código 3 a 9, tem-se os laços de repetição que solicitam ao usuário o preenchimento dos dados. Repare que, na linha de código 4, temos a instanciação da lista de nome `linha` cujo objetivo é compor toda uma linha da matriz `mat`. Dentro do `for`, que vai das linhas de código 5 a 8, temos a lógica que permite que o usuário insira cada dado da lista `linha`. Repare no comando `linha.append(dado)`, da linha de código 8. Quando o laço, que vai das linhas de código de 5 a 8, encerra-se, significa que toda uma linha da matriz foi preenchida, e, então, executamos o comando da linha de código 9, `mat.append(-linha)`, que adiciona os dados recém preenchidos à `mat`, criando nossa “lista de listas”, ou seja, a matriz propriamente dita. Ao final, das linhas 11 a 14, temos um exemplo de como é possível acessar elemento a elemento da matriz. Repare, na linha 14, o uso do termo `mat[i][j]`, que significa que estamos realizando o acesso ao elemento que se encontra na linha `i`, coluna `j`, da matriz `mat`.

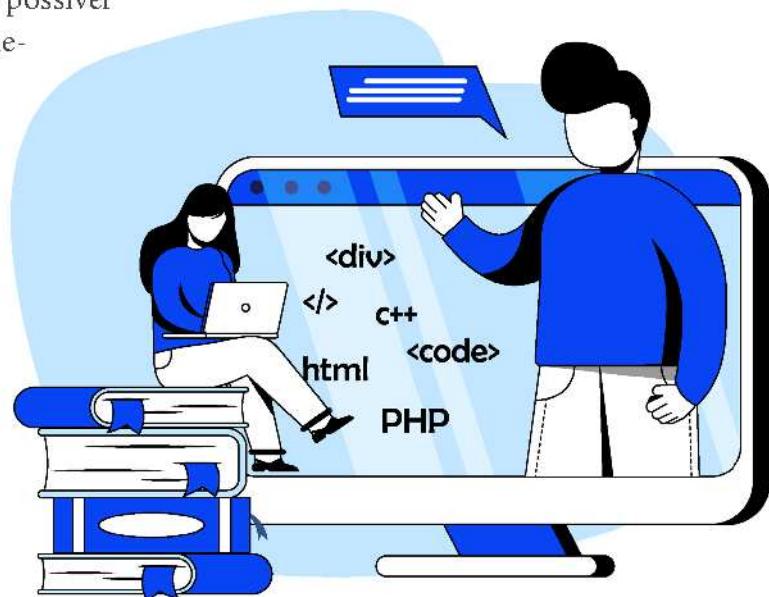


4 DICIONÁRIOS



Existem situações nas quais o problema é melhor resolvido quando, ao invés de acessar dados por meio de índices, o(a) programador(a) prefere obter acesso a um elemento de uma estrutura de dados através de “nomes”. Fique em paz, pois, caso não tenha entendido direito, vamos explicar.

Com a estrutura de dados denominada dicionário, em Python, é possível criar uma região de memória que faz a associação de um nome, um termo, a um dado respectivo. Assim como nas listas, teríamos que o índice i dá acesso ao dado desta posição, e os dicionários permitem que um termo ‘ x ’ dê acesso a um elemento salvo na respectiva estrutura de dados.



Dicionários permitem que dados de distintos tipos sejam armazenados em memória, em uma variável de mesmo identificador, porém com vários “campos”. Isso significa que cada “campo” de um dicionário pode armazenar um dado de qualquer tipo. Para criar um dicionário, devemos obedecer à seguinte sintaxe:

```
<nome_do_dicionário> = {  
    '<termo1>' : <dado1>,  
    '<termo2>' : <dado2>,  
    ...  
    '<termoN>' : <dadoN>  
}
```

O item `<nome_do_dicionário>`, da sintaxe anterior, deve ser substituído pelo identificador da variável que o(a) programador(a) deseja criar. Já o item `<termo1>`, delimitado entre aspas simples, deve ser substituído pelo termo do qual se deseja fazer uso quando o(a) programador(a) for realizar acesso às posições do dicionário. Já o item `<dado1>` deve ser substituído pelo conteúdo, pelo dado que se deseja armazenar de fato, assim como fazíamos com as listas.

Repare que, nesse caso, o item `<termo1>` está, diretamente, associado ao item `<dado1>`. Ou seja, `<termo1>` é como uma chave de acesso ao elemento `<dado1>`. É possível armazenar tantos termos/dados quanto necessário, separando-os por vírgula e contendo-os dentro das chaves. No exemplo do Quadro 9, temos um programa cujo objetivo é guardar, em uma mesma região de memória, os dados de um produto qualquer. Em uma situação generalista, cada produto possui um código, uma descrição, um preço e uma quantidade em estoque. Nesse caso, vamos, apenas, criar um único produto de teste, com os campos anteriormente listados. Em seguida, o programa pede para que o usuário preencha os dados do produto e, ao final, o algoritmo irá imprimir, na tela, os dados recém informados, via teclado.

```
01 produto = {  
02     'codigo': 0,  
03     'descricao': "",  
04     'preco': 0.0,  
05     'qtde': 0.0  
06 }  
07  
08 produto['codigo'] = int(input("Digite o có-  
digo do  
09 produto:\n"))  
10 produto['descricao'] = input("Digite a des-  
crição do  
11 produto:\n")  
12 produto['preco'] = float(input("Digite o pre-  
ço do  
13 produto:\n"))  
14 produto['qtde'] = float(input("Digite a quan-  
tidade em estoque do produto:\n"))  
15  
16  
17  
18 print("Código: ", produto['codigo'])  
19 print("Descrição: ", produto['descricao'])  
20 print("Preço: ", produto['preco'])  
21 print("Quantidade estoque: ", produto['qt-  
de'])
```

Quadro 9 - Exemplo de utilização de um dicionário em Python / Fonte: o autor.

Experimente executar esse programa para aferir seu resultado. Ocorre que, durante a execução, você irá notar que, quando as linhas, que vão de 8 a 15, forem executadas, os campos codigo, descricao, preco e qtde do dicionário produto serão populados. Em seguida, das linhas 17 a 20, o programa encerra, mostrando, na tela, o que o usuário preencheu anteriormente. Observe esse comportamento na Figura 4.

```
Digite o código do produto:  
1  
Digite a descrição do produto:  
Macarrão  
Digite o preço do produto:  
4.60  
Digite a quantidade em estoque do produto:  
130  
Código: 1  
Descrição: Macarrão  
Preço: 4.6  
Quantidade estoque: 130.0
```

Figura 4 - Resultado da execução do programa do Quadro 9 / Fonte: o autor.

Descrição da Imagem: na imagem, o programa inicia sua execução com a seguinte mensagem para o usuário: "Digite o código do produto:", e, então, o usuário insere um código igual a 1. Em seguida, a máquina imprime: "Digite a descrição do produto:", e, então, o usuário insere a palavra "Macarrão". Depois, a máquina continua com a mensagem: "Digite o preço do produto:", e o usuário informa 4.60. Por fim, a máquina solicita: "Digite a quantidade em estoque do produto:", e, com isso, o usuário digita 130. Ao término do programa, o computador imprime, na tela, a seguinte sequência de mensagens: "Código: 1", "Descrição: Macarrão", "Preço: 4.6", "Quantidade estoque: 130.0".



explorando Ideias

Em muitas situações, você será tentado a procurar, na internet, por soluções que te ajudem a resolver seu problema. Todavia você perceberá que, mesmo que o resultado copiado da internet “sirva”, ao se deparar com situações complexas, você precisará desenvolver seu próprio raciocínio para desenvolver uma solução. Assim sendo, quando for copiar um código, seja da internet, seja de livros, esteja certo de que entende o que o código está fazendo. Teste, modifique, adapte. Caso o código não funcione de primeira, procure pelos erros que podem ter surgido do processo de cópia/cola. Utilizar códigos de terceiros é uma prática comum entre programadores, todavia, além de dar o devido crédito, é inteligente que você saiba o que está fazendo.

Ainda que o exemplo acima seja meramente ilustrativo, no mundo real, deve-se lembrar de que as coisas são um pouco mais complexas. Na realidade, imagine um sisteminha de registro de transações em estoque. Não teríamos, apenas, um único produto, mas, sim, toda uma lista de produtos. Assim, o programa do Quadro 10, a seguir, modifica o código do exemplo anterior, para permitir que o usuário informe não 1, mas 10 produtos, em sequência. Ao final, de maneira similar ao exemplo de antes, os dados são informados na tela.

```
01 lista = []
02
03 for i in range(10):
04     produto = {
05         'codigo': 0,
06         'descricao': '',
07         'preco': 0.0,
08         'saldo': 0.0
09     }
10     print("Dados do produto da posição", i)
11     produto['codigo'] = int(input("Digite o código do
12         produto:\n"))
13     produto['descricao'] = input("Digite a descrição do
14         produto:\n")
15     produto['preco'] = float(input("Digite o preço do
16         produto:\n"))
17     produto['saldo'] = float(input("Digite o saldo do
18         produto:\n"))
19     lista.append(produto)
20
21 for i in range(10):
22     print("Dados do produto da posição", i)
23     print("Código: ", lista[i]['codigo'])
24     print("Descrição: ", lista[i]['descricao'])
25     print("Preço: ", lista[i]['preco'])
26     print("Saldo: ", lista[i]['saldo'])
```

Quadro 10 - Programa que combina o uso de dicionários e listas simultaneamente.
Fonte: o autor.

Experimente executar o programa do Quadro 10 e verifique o resultado. Caso se sinta confortável, altere os parâmetros das funções `range()` das linhas de código 3 e 22 para um número menor, fazendo com que seus testes não sejam tão extensos. Lembre-se: você só vai fixar seu conhecimento em programação colocando a mão na massa. Assim, pratique com força!

CONSIDERAÇÕES FINAIS

Ao longo desta unidade, tivemos contato com estruturas de dados mais elaboradas do que as variáveis simples. Vimos estruturas de dados cujas dimensões vão de uma até quantas dimensões forem necessárias. A primeira estrutura estudada foi a lista. Em Python, listas muito populares, pois são bastante flexíveis, podendo armazenar tanto dados homogêneos quanto tipos distintos. Dita unidimensional, em uma lista, podemos acessar elementos individualmente através do valor do índice da posição do respectivo dado.

Vimos, também, as *strings*, que nada mais são que cadeias de caracteres alfanuméricos. Basicamente, quando programadores(as) desejam trabalhar com informações textuais, em Python, recorre-se ao tipo de dados `str`. Foi possível ver exemplos de manipulação de *strings*, nos quais concatenamos, alteramos o texto de caixa alta para caixa baixa etc.

Pudemos ver, ainda, como as matrizes em Python, na realidade, podem ser representadas por listas. Ao considerar que uma linha individual é uma lista, tivemos a ideia de criar uma “lista de linhas”, ou seja, uma “lista de listas”. Para cada dimensão da matriz, como vimos, há a possibilidade de acessá-lo diretamente, informando qual é valor do índice de sua linha e, também, de sua coluna, por meio de colchetes.

Os dicionários, como estudado anteriormente, são estruturas de dados poderosas, no que diz respeito à associação de termos a valores. Cada par termo-valor é como uma posição dentro dessa estrutura. Ao tentar acessar uma posição de um dicionário, basta especificar, entre colchetes, qual é o termo cujo dado deve ser acessado.

Assim sendo, tivemos condições de avançar, ainda mais, em nossos conhecimentos sobre lógica de programação. Agora, já temos ferramentas para, além de realizar entrada e saída, além de criar estruturas de desvio de fluxo, poder armazenar diversos dados em uma mesma região de memória. Mas não esqueça do mais importante: pratique!



1. Escreva um programa que leia uma lista com um total de 30 elementos numéricos do tipo inteiro. Em seguida, apresente a mesma lista impressa em ordem inversa à ordem de entrada.
2. Faça um programa que leia duas listas A e B, ambas com 20 números inteiros. Efetue a soma entre as duas listas em uma terceira lista C e imprima C em ordem crescente.
3. Faça um programa que leia um nome e apresente as letras que se encontram nas posições pares.
4. Construa um programa que leia uma palavra e a escreva de trás para frente.
5. Faça um programa que leia uma palavra e a imprima quantas vezes forem o número de caracteres.
6. Construa um programa que efetue a leitura das quatro notas de 20 alunos, calcule e mostre a média de cada aluno e a média da turma.
7. Construa um programa que leia informações (matrícula, nome, setor e salário) de 20 funcionários. Deve ser permitido executar quantas consultas o operador desejar, em que ele digita a matrícula e são apresentados o setor e o salário. Se a matrícula digitada não existir, informar o usuário.
8. A computação, frequentemente, é utilizada para servir de ferramenta na identificação de diferenças ou semelhanças entre objetos. Dessa forma, faça um programa que leia duas listas A e B, cada uma com 5 elementos. Para cada elemento presente na lista A, mostre quantos elementos iguais ao respectivo elemento existem na lista B.
9. É comum que um programa seja capaz de realizar a ordenação de elementos ou dados de forma a facilitar as operações futuras e a apresentação deste conjunto de dados. Assim sendo, escreva um programa que leia uma lista A e a apresente em ordem decrescente.



10. O processamento de dados textuais, atualmente, é bastante avançado, e já existem bibliotecas capazes de auxiliar neste tipo de situação. Elabore um programa que leia uma palavra e, se ela tiver número ímpar de caracteres, imprima todas as suas vogais.
11. Um dos principais benefícios de aprender a programar é o fato de que é possível automatizar tarefas repetitivas. Dessa forma, faça um programa que leia uma palavra e o número de vezes que se deseja imprimi-la. Em seguida, realize as impressões da respectiva palavra, de acordo com o desejo do usuário.
12. Um programador que se preze necessita dominar vários tipos de estruturas de dados. Uma das estruturas mais importantes, em programas, são as matrizes. Assim, construa um programa que recebe duas matrizes inteiras de ordem 5 e imprima a soma e a diferença entre as matrizes.
13. Já pensou em facilitar a vida do seu professor, criando um programa que automatize os cálculos das notas de seus alunos? Sendo assim, faça um programa que efetue a leitura dos nomes de cinco alunos e, também, de suas quatro notas bimestrais. Calcule a média de cada aluno e apresente os nomes classificados em ordem crescente de média.
14. É possível criar sistemas para organização e controle de estoque nas mais diversas áreas. Assim sendo, elabore um programa para efetuar o cadastro de 20 livros e imprimi-los. O cadastro deve conter as seguintes informações: título, autor, editora, edição e ano.
15. Com o surgimento dos smartphones, a funcionalidade de agenda eletrônica passou a fazer parte do cotidiano das pessoas, de modo geral. Pensando em um protótipo de agenda, faça um programa para efetuar o cadastro de 30 contatos. O cadastro deve conter as seguintes informações: nome, telefone e e-mail. Apresente todos os cadastros.



TUPLAS E SEQUÊNCIAS

Vimos que listas e *strings* têm muitas propriedades em comum, como indexação e operações de fatiamento. Elas são dois exemplos de sequências (veja Tipos sequências — `list`, `tuple`, `range`). Como Python é uma linguagem em evolução, outros tipos de sequências podem ser adicionados. Existe ainda um outro tipo de sequência padrão na linguagem: a tupla. Uma tupla consiste em uma sequência de valores separados por vírgulas, por exemplo:

```
>>>
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
>>> # Tuples are immutable:
... t[0] = 88888
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> # but they can contain mutable objects:
... v = ([1, 2, 3], [3, 2, 1])
>>> v
([1, 2, 3], [3, 2, 1])
```

Fonte: adaptada de Documentação Python 3.9.1 ([2021], on-line).



Como você pode ver no trecho acima, na saída do console as tuplas são sempre envolvidas por parênteses, assim tuplas aninhadas podem ser lidas corretamente. Na criação, tuplas podem ser envolvidas ou não por parênteses, desde que o contexto não exija os parênteses (como no caso da tupla dentro de uma expressão maior). Não é possível atribuir itens individuais de uma tupla, contudo é possível criar tuplas que contenham objetos mutáveis, como listas.

Apesar de tuplas serem similares a listas, elas são frequentemente utilizadas em situações diferentes e com propósitos distintos. Tuplas são imutáveis, e usualmente contém uma sequência heterogênea de elementos que são acessados via desempacotamento (ver a seguir nessa seção) ou índice (ou mesmo por um atributo no caso de namedtuples). Listas são mutáveis, e seus elementos geralmente são homogêneos e são acessados iterando sobre a lista.

Um problema especial é a criação de tuplas contendo 0 ou 1 itens: a sintaxe usa certos truques para acomodar estes casos. Tuplas vazias são construídas por um par de parênteses vazios; uma tupla unitária é construída por um único valor e uma vírgula entre parênteses (não basta colocar um único valor entre parênteses). Feio, mas funciona. Por exemplo:

```
>>>
>>> empty = ()
>>> singleton = 'hello',          # <-- note trailing comma
>>> len(empty)
0
>>> len(singleton)
1
>>> singleton
('hello',)
```

Fonte: adaptada de Documentação Python 3.9.1 ([2021], on-line).



A instrução `t = 12345, 54321, 'bom dia!'` é um exemplo de empacotamento de tupla: os valores 12345, 54321 e 'bom dia!' são empacotados em uma tupla. A operação inversa também é possível:

```
>>>  
>>> x, y, z = t
```

Fonte: adaptada de Documentação Python 3.9.1 ([2021], on-line).

Isso é chamado, apropriadamente, de sequência de desempacotamento e funciona para qualquer sequência no lado direito. O desempacotamento de sequência requer que haja tantas variáveis no lado esquerdo do sinal de igual, quanto existem de elementos na sequência. Observe que a atribuição múltipla é, na verdade, apenas uma combinação de empacotamento de tupla e desempacotamento de sequência.

Fonte: Documentação Python 3.9.1 ([2021], on-line).



eu recomendo!



filme

The Social Dilemma (O Dilema das Redes)

Ano: 2020

Sinopse: a era da informação, pautada na internet, nos dispositivos móveis e em uma massa de dados nunca vista antes na história da humanidade, tem seus custos. Neste documentário, os autores mostram como os dados que todos os indivíduos produzem podem ser utilizados para fins duvidosos e questionáveis. A oferta de anúncios, consumismo, manipulação midiática e ideológica são temas abordados neste vídeo que toda pessoa que possui smartphone deveria assistir.





anotações



SUB-ROTINAS

PROFESSOR
Me. Pietro Martins de Oliveira

PLANO DE ESTUDO

A seguir, apresentam-se as aulas que você estudará nesta unidade:

- Funções
- Escopo de variáveis
- Parâmetros com objetos mutáveis e imutáveis
- Recursividade.

OBJETIVOS DE APRENDIZAGEM

Conhecer a sintaxe de criação e invocação de funções

- Compreender a diferença entre os escopos global e local
- Entender como se comportam parâmetros em funções
- Estudar funções recursivas.

INTRODUÇÃO



Caro(a) estudante, estamos nos direcionando ao fim do ciclo de estudos em Lógica de Programação e Algoritmos. Todavia é apenas o início do fim deste livro, e não de sua rotina de evolução no conhecimento em programação. Para fechar com chave de ouro, falaremos sobre as sub-rotinas, que, em Python, se traduzem em funções.

Problemas de algoritmos, geralmente, são mais complexos do que parecem. Durante a vida de um programador ou de um cientista de dados, haverá situações em que é mais simples “quebrar” um problema maior em problemas menores, cuja solução é mais simples. Ao decompor todo um programa em subprogramas, utilizamos, na verdade, o conceito de funções.

Em se tratando de funções, é preciso compreender que elas são como um subalgoritmo, dentro de um algoritmo maior, dessa forma, surgem as definições de variáveis globais e variáveis locais. Estas são, na realidade, o conceito de escopo de variáveis, que, por sua vez, tem relação com a acessibilidade de uma variável dentro de todo o programa, a partir de onde essa variável foi declarada.

Indo além e analisando como o Python permite a organização e a manipulação de dados, temos o conceito de mutabilidade. Especialmente, quando definimos funções, temos, também, que pensar em como serão passados dados, parâmetros e argumentos às respectivas funções. Por isso, é importante entender como a alteração dos conteúdos dos parâmetros influencia o programa como um todo.

Por fim, fechamos com os estudos voltados à recursividade. A recursão trata da invocação de uma função e esta, por sua vez, invoca a si mesma, direta ou indiretamente, para resolver um problema complexo a partir de problemas menores e de uma solução mais simples.

Com isso, você estará preparado(a) para avançar em seus estudos que envolvam programação em Python. Desejo a você ótimo progresso!



1 FUNÇÕES

Para solucionar problemas complexos e abrangentes, em geral, temos que dividir o problema em subproblemas mais simples e específicos e, com isso, dividimos a sua complexidade e facilitamos o processo de resolução. Este processo de decomposição é denominado refinamento sucessivo ou abordagem *top-down* (FORBELLONE; EBERSPACHER, 2005).

No particionamento dos problemas complexos, utilizamos funções para resolver cada subproblema, permitindo a modularização. Uma função é uma sub-rotina que tem, como objetivo, desviar a execução do programa principal para realizar uma tarefa específica e, geralmente, retornar um valor. São estruturas que possibilitam o programador separar os seus programas em blocos (ASCÊNCIO; CAMPOS, 2012).

Basicamente, toda função tem um nome, um identificador que segue as mesmas regras de nomenclatura de variáveis. Uma função pode, ou não, ter parâmetros, dependendo, exclusivamente, da lógica utilizada pelo desenvolvedor da respectiva sub-rotina. Ainda, por meio do comando `return`, é possível fazer com que a função produza, explicitamente, algum resultado de saída ao término de seu processamento. Todavia vale lembrar que o retorno de dados por uma função não é obrigatório em Python.

Observe a sintaxe de como criar uma função do zero:

```
def <nome_da_função>(<parâmetros>):
    <corpo_da_função>
    return <valor_de_retorno>
```

Neste caso, substituiremos o item <nome_da_função> pelo identificador que deve ser utilizado para invocar a função, em tempo de execução. O item <parâmetros> pode ser removido por completo, caso o programador não deseje que a sua nova função receba dados de entrada via parâmetros. Ainda, pode-se substituir o item <parâmetros> por um ou mais nomes de variáveis, separados por vírgula, em caso de mais de um. O item <corpo_da_função> deve ser substituído por todo o código que implementa a lógica desejada para resolver o subproblema. Por fim, o item <valor_de_retorno> deve ser substituído por algum dado, variável, literal ou expressão que produza a saída esperada para aquela função. Sempre tomar cuidado com a indentação, que, em Python, é o que define quais comandos pertencem, ou não, ao corpo da função criada.

No Quadro 1, temos um programa que utiliza uma função denominada `soma()` para efetuar leitura, processamento e saída de dados. No corpo principal de nosso programa, na linha de código 7, temos a invocação da respectiva função. Isso significa que o interpretador Python avaliará as linhas de código de 1 a 5, apenas para ter ciência de que, quando invocada aquela função, é o código contido nessas linhas que deve ser executado.

```
01 def soma():
02     num1 = float(input("Digite o primeiro número:"))
03     num2 = float(input("Digite o segundo número:"))
04     total = num1 + num2
05     print("A soma é", total)
06
07 soma()
```

Quadro 1 - Declarando uma função e, em seguida, invocando-a / Fonte: o autor.

Ainda, no Quadro 1, devemos reparar que, na linha de código 1, essa função não define parâmetro algum e, por isso, na linha 7, ao invocar a função, não informamos dado algum entre parênteses. Nas linhas 2 e 3, realizam-se operações de entrada e saída com o usuário; na linha 4, há o processamento, e na linha 5, diretamente de dentro do código da função, informamos ao usuário qual é o resultado encontrado. Experimente executar esse código, passo-a-passo, para compreender a ordem na qual os comandos são executados.

A esta altura, você já deve ter confirmado a informação de que, para um problema algorítmico, pode haver diversas soluções distintas que entregam o mesmo resultado. Assim sendo, observe o Quadro 2. Agora, na linha de código 1, tem-se a definição de dois parâmetros de entrada para a nova versão da função `soma()`. Neste caso, quando a função for invocada, é necessário passar dois valores entre parênteses, para que as variáveis de entrada `num1` e `num2` sejam populadas. Outro aspecto a se notar é que as operações de interação com o usuário foram removidas do corpo da função e levadas ao corpo principal do programa (repare na ordem e na indentação das operações).

```
01 def soma(num1, num2):  
02     total = num1 + num2  
03     return total  
04  
05 n1 = float(input("Digite o primeiro número:"))  
06 n2 = float(input("Digite o segundo número:"))  
07  
08 resultado = soma(n1, n2)  
09  
10 print("A soma é", resultado)
```

Quadro 2 - Versão alternativa para a função soma() / Fonte: o autor.

Observe a Figura 1, a seguir.

Figura 1 - Resultado da execução do programa do Quadro 2 / Fonte: o autor.

Descrição da Imagem: Na Figura 1, a máquina solicita ao usuário que informe o primeiro número e, então, o usuário informa o valor 10. Em seguida, a máquina solicita a digitação do segundo número e, assim, o usuário insere o número 20. Por fim, a máquina encerra a execução com a seguinte mensagem: "A soma é 30.0".

Observando a execução do programa do Quadro 2, na Figura 1, reparamos que, na realidade, ao encontrar a linha de código 8, o interpretador Python desvia o fluxo da execução para a linha 1, fazendo com que o resultado da soma seja calculado na linha 2. Ao encontrar a linha de código 3, o interpretador retorna, desviando o código, novamente, para a linha 8, na qual o resultado da execução da função `soma()` é atribuído à variável `resultado`.

AULA

2 ESCOPO DE VARIÁVEIS



Uma questão interessante, em relação a funções, é que elas podem alterar a acessibilidade de uma variável em outros pontos do código. O escopo de uma variável é o que deve ser analisado quando se quer entender onde uma variável é “visível”, ou não, dentro do programa. Observe o Quadro 3, a seguir.

```
01 def areaTriangulo(r):  
02     return 3.14*r**2  
03  
04 raio = float(input("Digite o raio do triângulo:"))  
05  
06 area = areaTriangulo(raio)  
07  
08 print("A área do triângulo é", area)
```

Quadro 3 - Declaração de função com variável local / Fonte: o autor.

A função `areaTriangulo()` foi criada para receber o raio de um círculo e retornar à sua área. Neste contexto, a variável de entrada `r` é uma variável local, ou seja, da forma como foi escrito o código, essa variável pertence ao escopo local à função `areaTriangulo()` e só pode ser invocada dentro da respectiva sub-rotina. Qualquer tentativa de usar o valor de `r` fora da função, de forma direta, poderia resultar em erro sintático e/ou semântico. Resumindo, `r` é uma variável local.

Para entender o escopo global, observe o Quadro 4, a seguir.

```
01 raio = 0  
02  
03 def areaTriangulo():  
04     return 3.14*raio**2  
05  
06 raio = float(input("Digite o raio do triângulo:"))  
07  
08 area = areaTriangulo()  
09  
10 print("A área do triângulo é", area)
```

Quadro 4 - Declaração de variáveis globais / Fonte: o autor.

Nesse segundo exemplo, a nova versão da função `areaTriangulo()` não recebe parâmetro. Agora, ao invés disso, a operação da linha de código 4 faz o uso da variável global `raio`. Ocorre que, quando se realiza a instanciação da variável `raio`, na linha 1, o interpretador Python já sabe que aquela é uma variável que deve estar visível em qualquer parte do programa, a partir dali. Ou seja, quando o interpretador atinge a linha de código 6, o usuário informa o valor que fica armazenado em `raio`. Na linha 8, a função `areaTriangulo()` é invocada e o mesmíssimo valor do `raio`, informado na linha 6, será utilizado na execução do comando da linha 4.

Assim sendo, o resultado do cálculo da área é retornado pela função `areaTriangulo()`. Na linha de código 8, a atribuição do respectivo resultado do cálculo recém-realizado é armazenada na variável `area` e, ao término do programa, na linha 10, o usuário recebe a solução na tela.

AULA

PARÂMETROS COM OBJETOS

Mutáveis e Imutáveis

Antes de falar em parâmetros propriamente ditos, falaremos, um pouco, sobre a forma com a qual a linguagem Python foi criada. Na realidade, ela é uma linguagem que dá suporte à programação orientada a objetos. Por isso, no fundo, toda variável, aqui é, na realidade, um objeto com atributos, métodos etc. Fique em paz, não precisa se preocupar com a orientação a objetos e os seus conceitos.



A orientação a objetos é um paradigma de programação focado em permitir que a abstração do mundo real, dentro do virtual, seja mais eficiente. Sistemas maiores e complexos, em algum ponto, exigem os recursos desse paradigma. Para programas simples, costuma-se utilizar o paradigma estruturado.

Basta saber o seguinte, em Python: alguns tipos de dados nos permitem criar variáveis que podem alterar o seu estado (seu conteúdo), sem que o interpretador realoque o seu endereço em memória; outras variáveis são mais “rígidas” e toda vez que alteramos o seu conteúdo em nossos códigos, o interpretador realoca a sua posição em memória.

Na prática, funciona assim: os objetos (variáveis) imutáveis, são aqueles que, a cada vez que precisarmos alterar o seu conteúdo, serão realocados em endereços de memória que não, necessariamente, serão sempre os mesmos; os objetos mutáveis, por outro lado, permitem que seu conteúdo seja alterado, em tempo de execução, sem que ele tenha de ser realocado, em memória. São exemplos de tipos de dados que geram objetos imutáveis: `int`, `float`, `str`, `bool`, `tuple`. Objetos do tipo `list`, por exemplo, são mutáveis.

O termo mutável/imutável, portanto, está associado à capacidade de alterar o conteúdo do objeto, sem alterar o seu endereço em memória. Isso faz com que a passagem de parâmetros em funções possa ser feita de formas distintas.

No programa do Quadro 5, a seguir, temos, na definição da função `soma()`, a declaração de duas variáveis de entrada, `num1` e `num2`. Quando invocamos a função, na linha 9, são passados os argumentos `n1` e `n2`. Isso fará com que se copie o conteúdo dessas variáveis e os coloque dentro das regiões de memória de `num1` e `num2`, respectivamente. Neste caso, `n1` tem um endereço em memória que é diferente de `num1`, e `n2` também tem endereço diferente de `num2`. Assim sendo, dizemos que os parâmetros `num1` e `num2` são “somente entrada”, pois, caso os seus conteúdos fossem alterados, dentro da função `soma()`, os conteúdos dos argumentos relacionados a eles, fora da função `soma()`, não seriam alterados simultaneamente.

```
01
02     def soma(num1, num2):
03
04         total = num1 + num2
05
06         return total
07
08
09     n1 = float(input("Digite o primeiro número:"))
10
11     n2 = float(input("Digite o segundo número:"))
12
13
14     resultado = soma(n1, n2)
15
16
17     print("A soma é", resultado)
```

Quadro 5 - Objeto imutável como parâmetro de uma função / Fonte: o autor.

Agora, observe o exemplo do Quadro 6, a seguir.

```
01     def soma(num1, num2, total):
02
03         total.append(num1 + num2)
04
05     n1 = float(input("Digite o primeiro número:"))
06
07     n2 = float(input("Digite o segundo número:"))
08
09     resultado = []
10
11     soma(n1, n2, resultado)
12
13
14     print("A soma é", resultado[0])
```

Quadro 6 - Objeto mutável como parâmetro de uma função / Fonte: o autor.

Repare que, nesta segunda versão, a função `soma()` é declarada com três parâmetros ao invés de dois, na linha de código 1 do Quadro 6. Além disso, não há comando `return`, na função `soma()`.

O que está acontecendo, neste caso (Quadro 6), é que a variável `resultado`, instanciada na linha 7, é, na verdade uma lista, um objeto que é mutável. Assim, quando se invoca a função `soma()`, na linha 9, não é necessário atribuir a sua execução a nenhuma variável, diferentemente do que era feito no programa do Quadro 5, na linha de código 9. Na realidade, na nova versão da função `soma()`, quando ela é invocada, passamos o objeto `resultado` como argumento, o que faz com que a variável `total`, declarada na linha 1, seja, na realidade, a mesmíssima variável `resultado`.



Caso você tenha interesse em entender melhor como funcionam os modelos de objetos em Python (mutáveis, imutáveis), acesse este link.



Em suma, no Quadro 6, quando a expressão `total.append(num1 + num2)` da linha 2 é executada, é como se estivéssemos executando o comando `resultado.append(num1 + num2)`, diretamente. Pelo fato de a variável `resultado` ser uma lista, tudo o que fazemos com a variável `total`, dentro da função `soma()`, afeta, diretamente, a mesma região de memória da variável `resultado`. E é por isso que podemos dispensar o comando `return`. Assim, ao imprimir o conteúdo de `resultado[0]`, na linha 11, estamos recuperando o cálculo que foi feito na linha 2.

Observe a Figura 2, a seguir.

```
TERMINAL ... 2: Python Del + □ ⌛ ^ ×
Digite o primeiro número:5
Digite o segundo número:6
A soma é 11.0
Ln 11. Col 32 Spaces: 4 UTF-8 CRLF Python ⌂ ⌂
```

Figura 2 - Resultado de execução idêntico para os programas dos Quadros 5 e 6 / Fonte: o autor.

Descrição da Imagem: Na Figura 2, a máquina imprime a mensagem "Digite o primeiro número:", então, o usuário insere o número 5. Depois, a máquina imprime "Digite o segundo número:", então, o usuário insere o número 6. Por fim, a máquina encerra o programa com a mensagem "A soma é 11.0".

A Figura 2 mostra o resultado da execução dos programas dos Quadros 5 e 6. Em suma, o resultado final, para o usuário, é o mesmo. Todavia, internamente, o interpretador Python trabalhou, de forma distinta, em cada uma das versões. Em uma aplicação de maior porte, o desenvolvimento da solução poderia exigir que ora o programador utilizasse uma forma de passagem de parâmetro, ora outra. Basicamente, isso tem implicações maiores quando se trata de trabalhar com listas e outros objetos mutáveis.

AULA

4 RECURSIVIDADE



$$\Phi = \frac{1 + \sqrt{5}}{2}$$

O conceito de recursividade não é, estritamente, do mundo da programação, mas, na verdade, é importado da matemática. Ao desenvolver um algoritmo, é possível elaborar uma função que chama a si mesma, isto é, uma função recursiva. A recursividade é um mecanismo que permite uma função chamar a si mesma, direta ou indiretamente. Uma função é recursiva quando possui uma chamada a si própria (ZIVIANE, 2004).

Ao construir funções recursivas, devemos nos certificar de que há um critério de parada, o qual determinará o momento que a função parará de fazer chamadas a si mesma, impedindo que entre em um loop. No Quadro 7, temos a implementação da função recursiva para cálculo do fatorial (LEAL; OLIVEIRA, 2020).

```
01 def factorial(x):  
02     if x == 0:  
03         return 1  
04     else:  
05         return x * factorial(x-1)  
06  
07 num = int(input("Digite um número inteiro:"))  
08  
09 resultado = factorial(num)  
10  
11 print("O factorial de", num, "é", resultado)
```

Quadro 7 - Exemplo de função recursiva para calcular o factorial de um número x qualquer, informado pelo usuário / Fonte: o autor.

O Quadro 8 apresenta a função recursiva para cálculo da série de Fibonacci.

```
01 def fibonacci(x):  
02     if x == 0 or x == 1:  
03         return x  
04     else:  
05         return fibonacci(x-1) + fibonacci(x-2)  
06  
07 num = int(input("Digite um número inteiro:"))  
08  
09 resultado = fibonacci(num)  
10  
11 print("O termo de fibonacci é", resultado)
```

Quadro 8 - Exemplo da criação e uso de uma função recursiva para calcular o “n-ésimo” termo da série de Fibonacci / Fonte: o autor.

Em ambos os casos, nos Quadros 7 e 8, tem-se a função que está sendo declarada invocando a ela própria, porém, passando, como parâmetro, uma instância menor do problema. Na prática, a função invocará a si própria tantas vezes que, em certo ponto, o critério de parada é atingido e o resultado é propagado (retornado) para o trecho de código que invocou a função originalmente.

CONSIDERAÇÕES FINAIS

Nesta reta final, falamos sobre sub-rotinas que, em Python, se traduzem em funções. Toda solução algorítmica complexa exige que programadores se organizem para “quebrar” problemas em subproblemas menores, cuja solução é mais simples. Ao decompor todo um programa em subprogramas, é muito usual que nos apropriemos do uso de funções.

Anteriormente, havíamos aprendido, apenas, como utilizar funções prontas, como `input()`, `print()`, `range()` e assim por diante. Todavia, nesta unidade, podemos criar nossas próprias funções e invocá-las em nossos programas, de acordo com a nossa necessidade.

Em se tratando de funções, é, particularmente, importante compreender que ela é como se fosse um sub-algoritmo, dentro de um algoritmo maior. Por isso, compreendemos que existem variáveis de escopos distintos: globais e locais.

Indo além e analisando a natureza de como o Python permite a organização e a manipulação de dados, estudamos o conceito de mutabilidade. Ao tratar da passagem de parâmetros em funções, pudemos ter a visualização de como o conceito de mutabilidade pode impactar a execução dos programas, sobretudo, quando se trata do uso de funções.

Por fim, fechamos com os estudos voltados à recursividade. Como visto, a recursão trata da invocação de uma função por ela mesma, direta ou indiretamente, para resolver o problema complexo, a partir de problemas menores e de solução mais simples.

A partir de tudo o que foi estudado até então, você já deve ter desenvolvido a capacidade de continuar os seus estudos de forma autônoma. Ser programador Python exige que você esteja sempre atualizado, além de entender que, na realidade, o processo de aprendizagem merece muito mais atenção do que o resultado que se deseja alcançar. O resultado é apenas consequência! Desejo sucesso a você!



1. Escreva um programa utilizando uma função que converta dada temperatura lida de Celsius para Fahrenheit.
2. Escreva um programa utilizando uma função que receba o peso de um peso em quilogramas e o converta para libras.
3. Faça uma função que receba, como parâmetro, uma lista com três números inteiros e retorne-os, ordenados, em forma crescente.
4. Elabore uma função que receba uma *string* e retorne a quantidade de consoantes.
5. É comum, em uma aplicação, ter de determinar quais números são pares ou ímpares dentre todos os valores de um conjunto de dados. Dessa forma, faça um programa que verifique se determinado número é positivo ou negativo, por meio de uma função.
6. Provavelmente, você já deve ter se deparado com uma situação na qual é preciso calcular o somatório de valores compreendidos dentro de um intervalo determinado. Por isso, elabore uma função que receba dois números positivos por parâmetro e retorne a soma dos N números inteiros existentes entre eles.



7. Imagine que você está desenvolvendo uma pequena funcionalidade dentro de um programa que será utilizado como processador de texto. Assim sendo, escreva uma função que receba um caractere e retorne 0 se for vogal, ou 1 se for uma consoante, um número ou um caractere especial. Considere que não é possível inserir letras maiúsculas e que o usuário deve respeitar esta regra.
8. É muito comum que programas tenham de implementar funcionalidades para identificar características específicas e, assim, realizar operações sobre um conjunto de dados. Dessa forma, faça um programa com uma função que apresente o somatório dos N primeiros números inteiros pares a partir de 1, definidos por um operador. O valor de N será informado pelo usuário.
9. No jardim de infância, durante o processo de ensino e aprendizagem, um professor costuma ensinar o tema “vogais e consoantes” de maneira bastante lúdica. Para auxiliar o professor, construa uma função que receba um nome e retorne o número de consoantes. Considere que o usuário inserirá apenas palavras com letras minúsculas, sem acentos, nem números, nem caracteres especiais.
10. O mundo das finanças é fascinante. Dentro deste contexto, existem diversas casas de câmbio que realizam compra e venda de moedas estrangeiras. Desse modo, elabore um programa que receba o valor da cotação do dólar e o valor em reais e que apresente o valor em dólares.



4 LINGUAGENS EM CRESCIMENTO NO MERCADO DE DATA SCIENCE

Scala

A Scala ganha destaque por ser utilizada pelo Apache Spark, um *framework* de processamento distribuído muito usado por engenheiros de dados. Além disso, é orientada a objetos, funcional e tem ampla biblioteca nativa que expande as suas capacidades.

Esta linguagem de programação tem muitas similaridades com o Java, o que a torna fácil de aprender e útil para aqueles que querem fazer mais, escrevendo menos. Scala também oferece outra vantagem importante: compatibilidade com a JVM, a máquina virtual Java, que é, amplamente, utilizada.

ScalaNLP (biblioteca de *machine learning*), Epic (*framework* para análise preditiva), Saddle (biblioteca de manipulação de dados) e Chalk (biblioteca para processamento de linguagem natural) são algumas de suas principais bibliotecas.

Linguagem R

Muito apreciada por matemáticos e estatísticos, a linguagem R é uma das mais utilizadas em Data Science, atualmente. Ela encanta os trabalhadores de Exatas por conta de seu suporte para cálculos e análises complexas, além do fato de ter sido criada por profissionais de Estatística.

É utilizada para modelagem linear e não linear, análises temporais, agrupamento e muitas outras funções. Esta tecnologia se caracteriza por exigir um computador bastante poderoso para executá-la, corretamente, em especial, no que diz respeito à memória RAM. Sistemas que não sejam 64 bits, por exemplo, não têm capacidade para rodar essa linguagem.

Outra característica bem marcante é a sua curva de aprendizagem bem acentuada, então, R não é uma boa opção para quem quer agilidade inicial, pois começar a utilizá-la é, realmente, complexo. Apesar disso, grandes empresas, como Microsoft e Oracle, já desenvolvem soluções de Data Science em R, o que tem ajudado a popularizar a linguagem.



Julia

Julia é a caçula desta lista com foco em Data Science, tendo sido, oficialmente, divulgada, apenas, em 2012. Contudo ela não fica atrás das outras opções no quesito desempenho, tendo sido, especificamente, criada para computação científica, processamento distribuído, *machine learning* e álgebra linear de grande escala.

Seu ponto positivo é ser *open source*. Porém a escassez de bibliotecas – algo natural, dada a idade da Julia – é um dos motivos pelos quais muitos ainda não resolveram apostar suas fichas nesta linguagem, embora vários cientistas de dados defendam o seu uso para lidar com aplicações diversas, especialmente, por conta de seu desempenho com *clusters* (processamento distribuído).

Python

A lista não ficaria completa sem Python, atualmente, a linguagem mais utilizada para Data Science, no Brasil. Além de sua popularidade, esta tecnologia tem forte apelo acadêmico, é bastante usada em cursos de Matemática e Estatística, tem tipagem dinâmica, funcional e, como base, a orientação a objetos.

A sintaxe simples, a facilidade no aprendizado e a gigantesca comunidade que ajuda no aprimoramento constante da linguagem contribuem, ainda mais, para a popularidade do Python. Outro ponto positivo é a fácil integração com outras linguagens. Para um cientista de dados que precisa trabalhar com máquinas virtuais de diferentes sistemas operacionais (Linux e Windows, por exemplo), esta é uma grande vantagem.

Por fim, não poderíamos deixar de mencionar a grande variedade de bibliotecas e pacotes exclusivos para ciência de dados. Scikit-learn (para *machine learning*), NumPy e Pandas (para análise de dados) são alguns dos mais populares.

Fonte: adaptado de Tecmundo (2020, on-line).



eu recomendo!



livro

Python para Análise de Dados

Autor: Wes McKinney

Editora: Novatec

Sinopse: obtenha instruções para manipular, processar, limpar e extrair informações de conjuntos de dados em Python. Você conhecerá bibliotecas como Pandas, NumPy, IPython e Jupyter. Este livro contém uma introdução às ferramentas de ciência de dados em Python. Ideal para analistas e para programadores Python iniciantes nas áreas de Ciência de Dados e Processamento Científico. Os arquivos de dados e os materiais relacionados ao livro estão disponíveis no GitHub.





Neste livro, você teve acesso aos fundamentos primordiais para aprender lógica de programação e desenvolver algoritmos. Todo problema com solução computacional pode ser resolvido a partir de conceitos, como variáveis, expressões, atribuição e execução sequencial de instruções.

Vimos como é possível utilizar operações de entrada e saída, como o `print()` e o `input()`, para que a máquina seja capaz de realizar interações com o usuário. Para que o usuário de nossos algoritmos compreenda o propósito daquele programa, ficou claro que é primordial que sejamos capazes de entender o enunciado do problema e articular uma solução lógica coerente.

Em se tratando das estruturas de desvio de fluxo, o primeiro contato foi feito por meio do comando `if`. Com ele, é possível instruir a máquina para que decida entre executar, ou não, um bloco de comandos. Em conjunto com o `if`, é possível, ainda, utilizar o comando `else`. No caso do par de comandos `if-else`, viu-se que é possível decidir entre a execução de dois blocos de comandos distintos. Ainda, vimos que é possível combinar um `if` seguido de um `else` com o comando `elif`.

As estruturas de repetição `while` e `for`, em Python, são utilizadas para que o programador permita a iteração de um mesmo bloco de comandos por várias vezes seguidas. Em especial, devemos lembrar da função `range()`, que, em conjunto com o comando `for`, permite que criemos laços de repetição contados.

Para finalizar, pudemos conhecer algumas estruturas de dados mais elaboradas, como as listas, as matrizes e os dicionários. E, ainda, aprendemos a criar e a utilizar nossas próprias funções, em linguagem Python.

A jornada rumo ao domínio pleno da lógica de programação deve continuar. Por isso, mantenha-se sempre programando, resolvendo exercícios, revisando conceitos e assumindo novas responsabilidades. Assim, desejo sucesso a você, aluno(a), nessa caminhada!

UNIDADE 1

ASCÊNCIO, A. F. G.; CAMPOS, E. A. V.de. **Fundamentos da Programação de Computadores: Algoritmos, Pascal e C/C++**. São Paulo: Pearson Prentice Hall, 2012.

FORBELLONE, A. L. V.; EBERSPACHER, H. F. **Lógica de Programação**: a construção de algoritmos e estruturas de dados. 3. ed. São Paulo: Pearson Prentice Hall, 2005.

LEAL, G. C. L.; OLIVEIRA, P. M. de. **Algoritmos e Lógica de Programação II**. Maringá: Unicesumar, 2020.

LUIZ, M.; ASCHER, D. **Aprendendo Python**. Porto Alegre: Bookman, 2007.

PHYTON BRASIL. **Python e Programação Orientada a Objeto**. 26 set. 2008. Disponível em: https://wiki.python.org.br/ProgramacaoOrientadaObjetoPython#Python_e_Programa.2BAOcA4w-o_Orientada_a_Objeto. Acesso em: 18 jan. 2021.

PYTHON.ORG. **A Referência da Linguagem Python**. [2021]. Disponível em <https://docs.python.org/pt-br/3/reference/index.html>. Acesso em: 16 fev. 2021.

UNIDADE 2

ASCÊNCIO, A. F. G.; CAMPOS, E. A. V.de. **Fundamentos da Programação de Computadores: Algoritmos, Pascal e C/C++**. São Paulo: Pearson Prentice Hall, 2012.

FORBELLONE, A. L. V.; EBERSPACHER, H. F. **Lógica de Programação**: a construção de algoritmos e estruturas de dados. 3. ed. São Paulo: Pearson Prentice Hall, 2005.

LEAL, G. C. L.; OLIVEIRA, P. M. de. **Algoritmos e Lógica de Programação II**. Maringá: Unicesumar, 2020.

LUIZ, M.; ASCHER, D. **Aprendendo Python**. Porto Alegre: Bookman, 2007.

MENDES, R. D. Inteligência Artificial: Sistemas Especialistas no Gerenciamento da Informação. **Ciência da Informação**, Brasília, v. 26, n. 1, jan./abr. 1997. Disponível em: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0100-19651997000100006. Acesso em: 18 jan. 2021.

PYTHON.ORG. **A Referência da Linguagem Python**. [2021]. Disponível em <https://docs.python.org/pt-br/3/reference/index.html>. Acesso em: 16 fev. 2021.



UNIDADE 3

ASCÊNCIO, A. F. G.; CAMPOS, E. A. V. de. **Fundamentos da Programação de Computadores: Algoritmos, Pascal e C/C++**. São Paulo: Pearson Prentice Hall, 2012.

FORBELLONE, A. L. V.; EBERSPACHER, H. F. **Lógica de Programação**: a construção de algoritmos e estruturas de dados. 3. ed. São Paulo: Pearson Prentice Hall, 2005.

LEAL, G. C. L.; OLIVEIRA, P. M. de. **Algoritmos e Lógica de Programação II**. Maringá: Unicesumar, 2020.

LUIZ, M.; ASCHER, D. **Aprendendo Python**. Porto Alegre: Bookman, 2007.

PYTHON.ORG. **A Referência da Linguagem Python**. [2021]. Disponível em <https://docs.python.org/pt-br/3/reference/index.html>. Acesso em: 16 fev. 2021.

SANTOS, W. L. dos. **Algoritmos Recursivos**. Dourados: UFGD-FACET, 2013. Disponível em <https://www.ic.unicamp.br/~ripolito/peds/mc102z/material/Recursividade.PDF>. Acesso em: 21 jan. 2021.

UNIDADE 4

DOCUMENTAÇÃO PYTHON 3.9.1. 5. **3. Tuplas e sequências**. 2021. Disponível em: <https://docs.python.org/pt-br/3/tutorial/datastructures.html>. Acesso em: 8 fev. 2021.

LEAL, G. C. L.; OLIVEIRA, P. M. de. **Algoritmos e Lógica de Programação II**. Maringá: Unicesumar, 2020.

UNIDADE 5

ASCÊNCIO, A. F. G.; CAMPOS, E. A. V. de A. **Fundamentos da Programação de Computadores: Algoritmos, Pascal e C/C++**. São Paulo: Pearson Prentice Hall, 2012.

FORBELLONE, A. L. V.; EBERSPACHER, H. F. **Lógica de Programação**: A construção de algoritmos e estruturas de dados. 3. ed. São Paulo: Pearson Prentice Hall, 2005.

LEAL, G. C. L.; OLIVEIRA, P. M. de. **Algoritmos e Lógica de Programação II**. Maringá: Unicesumar, 2020.

TECMUNDO. **4 linguagens em crescimento no mercado de Data Science**. 17 dez. 2020. Disponível em: <https://www.tecmundo.com.br/software/208403-4-linguagens-crescimento-mercado-data-science.htm>. Acesso em: 9 fev. 2021.

ZIVIANE, N. **Projeto de Algoritmos com implementações em Pascal e C**. 2. ed. São Paulo: Pioneira Thomson Learning, 2004.

UNIDADE 1

1. A.

2.

```
01 #Entrada de dados
02 #Solicitando o primeiro número
03 num1 = int(input("Digite o primeiro número:\n"))
04 #Solicitando o segundo número
05 num2 = int(input("Digite o segundo número:\n"))
06
07 #Processamento
08 #Cálculo do ano de nascimento
09 total = num1 + num2
10
11 #Saída de dados
12 print("A soma dos números é:", total)
```

3.

```
01 num = int(input("Digite um número:\n"))

02

03 ant = num - 1

04 suc = num + 1

05

06 print("O antecessor:", ant)

07 print("O sucessor:", suc)
```

4.

```
01 n1 = float(input("Digite a nota 1:\n"))

02 n2 = float(input("Digite a nota 2:\n"))

03 n3 = float(input("Digite a nota 3:\n"))

04 n4 = float(input("Digite a nota 4:\n"))

05

06 media = (n1 + n2 + n3 + n4)/4

07

08 print("A média é:", media)
```

5.

```
01 deposito = float(input("Informe o valor do depósito:\n"))
02 taxa = float(input("Informe a taxa de juros:\n"))
03
04 rendimento = deposito * (taxa/100)
05 total = deposito + rendimento
06
07 print("O rendimento é", rendimento)
08 print("O total é", total)
```

6.

```
01 num1 = float(input("Informe o primeiro número:\n"))
02 num2 = float(input("Informe o segundo número:\n"))
03
04 total = pow(num1, num2)
05
06 print(num1, "elevado a", num2, "é igual a", total)
```

7.

```
01 basel = float(input("Informe o valor da base maior:\n"))
02 base2 = float(input("Informe o valor da base menor:\n"))
03 altura = float(input("Informe o valor da altura:\n"))
04
05 area = ((basel + base2) * altura)/2
06
07 print("A área do trapézio é:", area)
```

8.

```
01 nome = input("Digite seu nome:\n")
02
03 print("Bem-vindo(a) à disciplina de Lógica de
Programação e Algoritmos,", nome)
```

9.

```
01 valor = int(input("Digite um número inteiro:\n"))
02
03 print("O quadrado de", valor, "é", pow(valor,2))
04 print("A raiz quadrada de", valor, "é", pow(va-
lor,1/2))
```

10.

```
01 n1 = float(input("Digite o valor 1:\n"))
02 p1 = float(input("Digite o peso para o valor 1:\n"))
03 n2 = float(input("Digite a valor 2:\n"))
04 p2 = float(input("Digite o peso para o valor 2:\n"))
05 n3 = float(input("Digite a valor 3:\n"))
06 p3 = float(input("Digite o peso para o valor 3:\n"))
07
08 media = (n1*p1 + n2*p2 + n3*p3)/(p1+p2+p3)
09
10 print("A média é", media)
```

11.

```
01 r = float(input("Insira o raio:\n"))

02

03 A = 3.14 * r**2

04 P = 2 * 3.14 * r

05

06 print("A área é", A)

07 print("O perimetro é", P)
```

UNIDADE 2

1.

```
01 num = int(input("Informe o número:\n"))

02

03 if num % 5 == 0:

04     print("O número", num, "é divisivel por 5.")

05 else:

06     print("O número", num, "não é divisível por 5.")
```

2.

```
01 nome = input("Informe o nome:\n")
02 idade = int(input("Informe a idade:"))
03
04 print("Nome:", nome)
05 print("Idade:", idade)
06
07 if idade <= 18:
08     print("O valor do plano é: R$ 50,00.")
09 elif idade >= 19 and idade <= 29:
10     print("O valor do plano é: R$ 70,00.")
11 elif idade >= 30 and idade <= 45:
12     print("O valor do plano é: R$ 90,00.")
13 elif idade >= 46 and idade <= 65:
14     print("O valor do plano é: R$ 130,00.")
15 else:
16     print("O valor do plano é: R$ 170,00.)
```

3.

```
01 idade = int(input("Informe a idade:\n"))

02

03 if idade < 16:
04     print("Não eleitor.")

05 elif idade < 18 or idade > 65:
06     print("Eleitor facultativo.")

07 else:
08     print("Eleitor obrigatório.")
```

4.

```
01 sexo = input("Informe o sexo (M/F):\n")

02 altura = float(input("Informe a altura:\n"))

03

04 if sexo == "F" or sexo == "f":
05     peso = (61.2 * altura) - 44.7
06 else:
07     peso = (72.7 * altura) - 58
08

09 print("O sexo é:", sexo)
10 print("A altura é:", altura)
11 print("O peso ideal é:", peso)
```

5.

```
01 total = 0
02 print("1 - italiano; 2 - japonês; 3 - Salvadorenho")
03 op = int(input("Informe o número do prato desejado:\n"))
04
05 if op == 1:
06     total = total + 750
07 elif op == 2:
08     total = total + 324
09 elif op == 3:
10     total = total + 545
11
12
13 print("1 - chá; 2 - suco de laranja; 3 - refrigerante")
14 op = int(input("Informe o número da bebida desejada:\n"))
15
16 if op == 1:
17     total = total + 30
18 elif op == 2:
19     total = total + 80
20 elif op == 3:
21     total = total + 90
22
23
24 print("O total de calorias é:", total)
```

6.

```
01 num = int(input("Digite o 1º número:\n"))
02 maior = num
03 menor = num
04
05 num = int(input("Digite o 2º número:\n"))
06 if num > maior:
07     maior = num
08 if num < menor:
09     menor = num
10
11 num = int(input("Digite o 3º número:\n"))
12 if num > maior:
13     maior = num
14 if num < menor:
15     menor = num
16
17 num = int(input("Digite o 4º número:\n"))
18 if num > maior:
19     maior = num
20 if num < menor:
21     menor = num
22
23 num = int(input("Digite o 5º número:\n"))
24 if num > maior:
25     maior = num
26 if num < menor:
27     menor = num
28
29 print("O maior é:", maior)
30 print("O menor é:", menor)
```

7.

```
01 num = int(input("Digite o número:\n"))

02

03 if num % 3 == 0:
04     print("O número é divisível por 3.")

05 if num % 7 == 0:
06     print("O número é divisível por 7.")
```

8.

```
01 num = int(input("Digite o número do mês:\n"))
02
03 if num == 1:
04     print("Janeiro.")
05 elif num == 2:
06     print("Fevereiro.")
07 elif num == 3:
08     print("Março.")
09 elif num == 4:
10     print("Abril.")
11 elif num == 5:
12     print("Maio.")
13 elif num == 6:
14     print("Junho.")
15 elif num == 7:
16     print("Julho.")
17 elif num == 8:
18     print("Agosto.")
19 elif num == 9:
20     print("Setembro.")
21 elif num == 10:
22     print("Outubro.")
23 elif num == 11:
24     print("Novembro.")
25 elif num == 12:
26     print("Dezembro.")
27 else:
28     print("Mês inválido.")
```

9.

```
01 cargo = int(input("Digite o número do cargo do\n02 funcionário:\n"))
03 salario = float(input("Digite o valor do salário do\n04 funcionário:\n"))
05
06 if cargo == 1:
07     aumento = salario * 0.4
08     salario_final = salario + aumento
09     print("O servente teve aumento de R$", aumento,
10          "e"
11          "agora recebe R$", salario_final)
12 elif cargo == 2:
13     aumento = salario * 0.35
14     salario_final = salario + aumento
15     print("O pedreiro teve aumento de R$", aumento,
16          "e"
17          "agora recebe R$", salario_final)
18 elif cargo == 3:
19     aumento = salario * 0.20
20     salario_final = salario + aumento
21     print("O mestre de obras teve aumento de R$",
22           aumento,
23           "e agora recebe R$", salario_final)
24 elif cargo == 4:
25     aumento = salario * 0.1
26     salario_final = salario + aumento
27     print("O técnico de segurança teve aumento de
R$",
28           aumento, "e agora recebe R$", salario_final)
```

10.

```
01 cod_estado = int(input("Digite o código do esta-  
02 do:\n"))  
03 peso = int(input("Digite o peso da carga em tonela-  
04 das:\n"))  
05 cod_carga = int(input("Digite o código da car-  
06 ga:\n"))  
07  
08 if cod_estado == 1:  
09     taxa_imp = 0.2  
10 elif cod_estado == 2:  
11     taxa_imp = 0.15  
12 elif cod_estado == 3:  
13     taxa_imp = 0.1  
14 elif cod_estado == 4:  
15     taxa_imp = 0.05  
16  
17 if cod_carga >= 10 and cod_carga <= 20:  
18     preco_quilo = 180  
19 elif cod_carga >= 21 and cod_carga <= 30:  
20     preco_quilo = 120  
21 elif cod_carga >= 31 and cod_carga <= 40:  
22     preco_quilo = 230  
23  
24 imposto = peso * 1000 * preco_quilo * taxa_imp  
25 preco = peso * 1000 * preco_quilo  
26 total = preco + imposto  
27  
28 print("Peso em kg:", peso*1000)  
29 print("Preço: R$", preco)  
30 print("Imposto: R$", imposto)  
31 print("Total: R$", total)
```

UNIDADE 3

1.

```
01 num = int(input("Digite o número:\n"))

02

03 fat = 1
04 for i in range(1, num+1, 1):
05     fat = fat * i
06
07 print("O fatorial é:", fat)
```

2.

```
01 for i in range(3, 100, 3):

02     print(i)
```

3.

```
01 num = int(input("Informe o número:\n"))

02

03 qtdade = 0
04
05 for i in range(1, num+1, 1):
06     if num % i == 0:
07         qtdade = qtdade + 1
08
09 if qtdade == 2:
10     print("O número é primo.")
11 else:
12     print("Não é primo.")
```

4.

```
01 npessoas = 0
02 somaf = 0
03 somas = 0
04 msalario = 0
05
06 salario = float(input("Informe o salário:\n"))
07
08 while salario != -1:
09     filhos = int(input("Informe o número de fi-
10     lhos:\n"))
11     npessoas += 1
12     if salario > msalario:
13         msalario = salario
14     somaf += filhos
15     somas += salario
16     salario = float(input("Informe o salário:\n"))
17
18 print("A média de salários é: R$", somas/npessoas)
19 print("A média de filhos é:", somaf/npessoas)
20 print("O maior salário é: R$", msalario)
```

5.

```
01 somaa = 0
02 somai = 0
03 npessoas = 0
04
05 idade = int(input("Informe a idade:\n"))
06
07 while idade != 0:
08     altura = float(input("Informe a altura:\n"))
09     npessoas += 1
10     somai += idade
11     somaa += altura
12     idade = int(input("Informe a idade:\n"))
13
14 print("A média de altura é", somaa/npessoas)
15 print("A média de idade é", somai/npessoas)
```

6.

```
01 npessoas = 0
02 npessoass = 0
03 npessoasi = 0
04 somai = 0
05
06 idade = int(input("Informe a idade:\n"))
07
08 while idade != 0:
09     while True:
10         opiniao = int(input("Informe a opinião:\n"))
11         if opiniao == 1 or opiniao == 2 or
12             opiniao == 3:
13             break
14
15         npessoas += 1
16         if opiniao == 1:
17             somai += idade
18             npessoass += 1
19         elif opiniao == 3:
20             npessoasi += 1
21         idade = int(input("Informe a idade:\n"))
22
23 media = somai/npessoass
24
25 print("O número de pessoas insatisfeitas é",
26       npessoasi)
27 print("O número de pessoas satisfeitas é", npes-
28       soass)
29 print("A média de idade das pessoas satisfeitas é",
30       media)
```

7.

```
01 primeiro = True
02 quantidade = 0
03 soma = 0
04 pares = 0
05 impares = 0
06
07 while True:
08     numero = int(input("Digite um número:\n"))
09
10     if numero == 0:
11         break
12
13     else:
14         if primeiro:
15             maior = numero
16             menor = numero
17             primeiro = False
18         else:
19             if numero > maior:
20                 maior = numero
21             if numero < menor:
22                 menor = numero
23
24         quantidade += 1
25         soma += numero
26
27         if numero % 2 == 0:
28             pares += 1
29         else:
30             impares += 1
31
32 media = soma/quantidade
33 print("Média:", media)
34 print("Maior:", maior)
35 print("Menor:", menor)
36 print("Pares:", pares)
37 print("Ímpares:", impares)
```

8.

```
01 S = 0
02
03 N = int(input("Digite um número:\n"))
04
05 for i in range(1,N+1,1):
06     S += 1/i
07
08 print("O valor de S é", S)
```

9.

```
01 for i in range(1,11,1):
02     print("\nTabuada do", i)
03     for j in range (1,11,1):
04         valor = i*j
05         print(i, "x", j, "=", valor)
```

10.

```
01 soma = 0
02
03 for i in range(200,501,1):
04     if i % 2 != 0:
05         soma += i
06
07 print("A soma de todos os inteiros ímpares entre 200
e 500 é", soma)
08
```

11.

```
01 for i in range(1, 31, 1):
02     if i % 3 == 0:
03         print("O número é divisível por 3")
04     if i % 7 == 0:
05         print("O número é divisível por 7")
```

12.

```
01 frase = input("Digite a frase:\n")
02 N = int(input("Insira o número de repetições:\n"))
03
04 for i in range(0,N,1):
05     print(frase)
```

13.

```
01 preco_total = 0
02
03 while True:
04     pedido = int(input("Digite o numero do
05         pedido:\n"))
06     if pedido != 0:
07         dia = int(input("Digite o dia do pedi-
08             do:\n"))
09         mes = int(input("Digite o mês do pedi-
10             do:\n"))
11         ano = int(input("Digite o ano do pedi-
12             do:\n"))
13         preco_unit = int(input("Digite o preço
14             unitário: \n"))
15         quantidade = int(input("Digite a
16             quantidade:\n"))
17         preco_total += preco_unit * quantidade
18     else:
19         break
20
21 print("O preço total dos pedidos é: R$", preco_to-
22 tal)
```

14.

```
01 t_peso = 0
02 t_idade = 0
03 q_casadas = 0
04 q_soltiras = 0
05 q_separadas = 0
06 q_viuvas = 0
07 quantidade = 0
08
09 while True:
10     idade = int(input("Digite a idade:\n"))
11     if idade != 0:
12         t_idade += idade
13         quantidade += 1
14         peso = float(input("Digite o peso:\n"))
15         t_peso += peso
16         sexo = int(input("Digite o sexo (1 = M; 2 = F):\n"))
17         print("Digite o estado civil")
18         print("1 - casadas;")
19         print("2 - solteiras;")
20         print("3 - separadas;")
21         print("4 - viúvas;")
22         est_civil = int(input())
23         if est_civil == 1:
24             q_casadas += 1
25         elif est_civil == 2:
26             q_soltiras += 1
27         elif est_civil == 3:
28             q_separadas += 1
29         elif est_civil == 4:
30             q_viuvas += 1
31         else:
32             print("Estado civil inválido.")
33     else:
34         break
35
36
37 m_idade = t_idade / quantidade
38 m_peso = t_peso / quantidade
39 print("Casadas: ", q_casadas)
40 print("Solteiras: ", q_soltiras)
41 print("Separadas: ", q_separadas)
42 print("Viúvas: ", q_viuvas)
43 print("Média de peso:", m_peso)
44 print("Média de idade:", m_idade)
```

15.

```
01 area = 0
02
03 while True:
04     comodo = input("Digite o nome do cômodo:\n")
05     if comodo != "FIM":
06         largura = float(input("Digite a largura do
07         cômodo:\n"))
08         comprimento = float(input("Digite o
09         comprimento do cômodo:\n"))
10         area += (largura*comprimento)
11     else:
12         break
13 print("Área total da casa:", area)
14
```

UNIDADE 4

1.

```
01 lista = []
02
03 for i in range(30):
04     print("Digite o elemento da posição", i, ":")
05     lista.append(int(input()))
06
07 print("Lista invertida:")
08 for i in range(29, -1, -1):
09     print(lista[i])
```

2.

```
01 listaA = []
02 listaB = []
03 listaC = []
04
05 for i in range(20):
06     print("Digite o elemento da posição", i, "da lis-
07         ta A:")
08     listaA.append(int(input()))
09
10 for i in range(20):
11     print("Digite o elemento da posição", i, "da lis-
12         ta B:")
13     listaB.append(int(input()))
14
15 for i in range(20):
16     listaC.append(0)
17
18 for i in range(20):
19     listaC[i] = listaA[i] + listaB[i]
20
21 for i in range(20):
22     for j in range(i+1, 20):
23         if(listaC[i] > listaC[j]):
24             troca = listaC[i]
25             listaC[i] = listaC[j]
26             listaC[j] = troca
27
28 print(listaC)
```

3.

```
01 nome = input("Digite o nome:\n")
02
03 for i in range(len(nome)):
04     if i % 2 == 0:
05         print(nome[i])
```

4.

```
01 palavra = input("Digite a palavra:\n")
02
03 for i in range(len(palavra)-1, -1, -1):
04     print(palavra[i], end="")
```

5.

```
01 palavra = input("Digite a palavra:\n")
02
03 for i in range(len(palavra)):
04     print(palavra)
```

6.

```
01 media = []
02 notas = []
03
04 somat = 0
05 mediat = 0
06
07 for i in range(3):
08     soma = 0
09     notasAluno = []
10     for j in range(4):
11         print("Informe a nota", j+1, "do aluno", i+1,
12             ":")
13         nota = float(input())
14         notasAluno.append(nota)
15         soma += nota
16     media.append(soma/4)
17     somat += media[i]
18
19 mediat = somat/3
20
21 for i in range(3):
22     print("A média do aluno", i+1, "é", media[i])
23
24 print("A média da turma é", mediat)
```

7.

```
01 lista = []
02
03 TAM = 20
04
05 for i in range(TAM):
06     f = {
07         'matricula': 0,
08         'nome': '',
09         'setor': '',
10         'salario': 0.0
11     }
12     print("Dados do funcionário", i+1, "de um total de", TAM
13     , ",")
14     f['matricula'] = int(input("Informe a matrícula do
15         funcionário:\n"))
16     f['nome'] = input("Informe o nome do funcionário:\n")
17     f['setor'] = input("Informe o setor do funcionário:\n")
18     f['salario'] = float(input("Informe o salário do
19         funcionário:\n"))
20     lista.append(f)
21
22 while True:
23     chave = int(input("Informe a matrícula que deseja
24         buscar:\n"))
25     i = 0
26     achou = False
27     while i < TAM and achou == False:
28         if(lista[i]['matricula'] == chave):
29             achou = True
30         else:
31             i += 1
32
33     if achou:
34         print("O setor é", lista[i]['setor'])
35         print("O salário é R$", lista[i]['salario'])
36     else:
37         print("Matrícula não cadastrada.")
38
39     while True:
40         op = input("Deseja realizar nova busca (s/n)?\n")
41         if(op == "s" or op == "S" or op == "n"
42             or op == "N"):
43             break
44
45     if( op == "n" or op == "N"):
46         break
```

8.

```
01 listaA = []
02 listaB = []
03
04 TAM = 5
05
06 for i in range(TAM):
07     print("Insira o valor de A na posição", i, ":")
08     dado = int(input())
09     listaA.append(dado)
10
11 for i in range(TAM):
12     print("Insira o valor de B na posição", i, ":")
13     dado = int(input())
14     listaB.append(dado)
15
16 for i in range(TAM):
17     iguais = 0
18     for j in range(TAM):
19         if listaA[i] == listaB[j]:
20             iguais += 1
21     if iguais > 0:
22         print("Número de elementos em B que são iguais
23         a",
24             listaA[i], ":", iguais)
```

9.

```
01 lista = []
02
03 for i in range(3):
04     print("Digite o valor da posição", i, ":")
05     dado = int(input())
06     lista.append(dado)
07
08 for i in range(2):
09     for j in range(i+1, 3):
10         if lista[i] < lista[j]:
11             troca = lista[i]
12             lista[i] = lista[j]
13             lista[j] = troca
14
15 print(lista)
```

10.

```
01 palavra = input("Digite uma palavra:\n")
02
03 palavra = palavra.lower()
04
05 if len(palavra) % 2 == 1:
06     for i in range(len(palavra)):
07         palavra[i]
08         if palavra[i] == "a" or palavra[i] == "e"
09             or palavra[i] == "i" or palavra[i] == "o"
10             or palavra[i] == "u":
11                 print(palavra[i], end="")
```

11.

```
01 palavra = input("Digite uma palavra:\n")
02 n = int(input("Digite o número de repetições:\n"))
03
04 for i in range(n):
05     print(palavra)
```

12.

```
01 matA = []
02 matB = []
03
04 N = 5
05
06 print("Matriz A:")
07 for i in range(N):
08     linha = []
09     for j in range(N):
10         print("[", i, "][", j, "]:")
11         linha.append(int(input()))
12     matA.append(linha)
13
14 print("Matriz B:")
15 for i in range(N):
16     linha = []
17     for j in range(N):
18         print("[", i, "][", j, "]:")
19         linha.append(int(input()))
20     matB.append(linha)
21
22 print("Soma das matrizes A + B:")
23 for i in range(N):
24     for j in range(N):
25         print(matA[i][j]+matB[i][j], end="\t")
26     print()
27
28 print("Subtração das matrizes A - B:")
29 for i in range(N):
30     for j in range(N):
31         print(matA[i][j]-matB[i][j], end="\t")
32     print()
```

13.

```
01 boletim = []
02
03 N = 5
04 M = 4
05 notas = 0
06
07 for i in range(N):
08     dados = {
09         'nome': '',
10         'notas': [],
11         'media': 0.0
12     }
13     print("Insira o nome da posição", i+1, ":")
14     dados['nome'] = input()
15     soma = 0
16     for j in range(M):
17         print("Digite a nota de número", j+1, ":")
18         nota = int(input())
19         dados['notas'].append(nota)
20         soma += nota
21     dados['media'] = soma/M
22     print("*** Dados inseridos ***")
23     print("Nome:", dados['nome'])
24     print("Nota 1:", dados['notas'][0])
25     print("Nota 2:", dados['notas'][1])
26     print("Nota 3:", dados['notas'][2])
27     print("Nota 4:", dados['notas'][3])
28     print("*** Dados inseridos ***")
29     boletim.append(dados)
30
31 print()
32 print("-- Ordenando --")
33 print()
34
35 for i in range(N):
36     for j in range(i+1, N):
37         if boletim[i]['media'] > boletim[j]['media']:
38             troca = boletim[i]
39             boletim[i] = boletim[j]
40             boletim[j] = troca
41
42 for i in range(N):
43     print("###")
44     print("Nome:", boletim[i]['nome'])
45     print("Média:", boletim[i]['media'])
46     print("Nota 1:", boletim[i]['notas'][0])
47     print("Nota 2:", boletim[i]['notas'][1])
48     print("Nota 3:", boletim[i]['notas'][2])
49     print("Nota 4:", boletim[i]['notas'][3])
```

14.

```
01 lista = []
02
03 N = 20
04
05 for i in range(N):
06     livro = {
07         'titulo': '',
08         'autor': '',
09         'editora': '',
10         'edicao': 0,
11         'ano': 0
12     }
13     print("Insira os dados do livro", i+1, ":")
14     livro['titulo'] = input("Título:\n")
15     livro['autor'] = input("Autor:\n")
16     livro['editora'] = input("Editora:\n")
17     livro['edicao'] = int(input("Edição:\n"))
18     livro['ano'] = int(input("Ano:\n"))
19     lista.append(livro)
20
21 print("\n\n*** Livros Cadastrados ***\n")
22 for i in range(N):
23     print("Título:", lista[i]['titulo'])
24     print("Autor:", lista[i]['autor'])
25     print("Editora:", lista[i]['editora'])
26     print("Edição:", lista[i]['edicao'])
27     print("Ano:", lista[i]['ano'])
28     print()
```

15.

```
01 lista = []
02
03 N = 3
04
05 for i in range(N):
06     dados = {
07         'nome': '',
08         'telefone': '',
09         'email': ''
10     }
11     print("Insira os dados do cadastro", i+1, ":")
12     dados['nome'] = input("Nome:\n")
13     dados['telefone'] = input("Telefone:\n")
14     dados['email'] = input("E-mail:\n")
15     lista.append(dados)
16
17 print("\n\n*** Pessoas Cadastradas ***\n")
18 for i in range(N):
19     print("Nome:", lista[i]['nome'])
20     print("Telefone:", lista[i]['telefone'])
21     print("E-mail:", lista[i]['email'])
22     print()
```

UNIDADE 5

1.

```
01 def convertet(celcius):  
02     temp = celcius * 1.8 + 32  
03     return temp  
04  
05 c = float(input("Informe a temperatura em graus Cel-  
sius"))  
06  
07 resposta = convertet(c)  
08  
09 print("Em Fahrenreit é", resposta)
```

2.

```
01 def convertep(peso):  
02     peso = peso * 2.68  
03     return peso  
04  
05 p = float(input("Informe o peso em quilogramas:"))  
06  
07 resposta = convertep(p)  
08  
09 print("Em libras é", resposta)
```

3.

```
01 def ordenaLista(lista):
02     for i in range(2):
03         for j in range(i+1,3):
04             if lista[i] > lista[j]:
05                 troca = lista[i]
06                 lista[i] = lista[j]
07                 lista[j] = troca
08
09 numeros = []
10
11 for i in range(3):
12     num = int(input("Insira um elemento:"))
13     numeros.append(num)
14
15 print("Lista original:", numeros)
16
17 ordenaLista (numeros)
18
19 print("Lista ordenada:", numeros)
```

4.

```
01 def contaVogais(palavra):  
02     qtd = 0  
03     for i in range(len(palavra)):  
04         if palavra[i] in ('a', 'e', 'i', 'o', 'u'):  
05             qtd += 1  
06     return qtd  
07  
08 string = input("Insira uma palavra:")  
09  
10 print("O numero de vogais é", contaVogais(string))
```

5.

```
01 numero = 0  
02  
03 def verifica():  
04     if numero > 0:  
05         print("Positivo.")  
06     elif numero < 0:  
07         print("Negativo.")  
08     else:  
09         print("Zero.")  
10  
11 numero = int(input("Insira um número inteiro:"))  
12  
13 verifica()
```

6.

```
01 def soma(num1, num2):
02     resultado = 0
03     for i in range(num1, num2+1):
04         resultado += i
05     return resultado
06
07 nl = int(input("Insira o primeiro número inteiro:"))
08 n2 = int(input("Insira o segundo número inteiro:"))
09
10 print("Somatório:", soma(nl, n2))
```

7.

```
01 def caractere(letra):
02     if letra in ('a', 'e', 'i', 'o', 'u'):
03         return 0
04     else:
05         return 1
06
07 l = input("Insira uma letra:")
08
09 if caractere(l) == 0:
10     print("Essa letra é uma vogal.")
11 else:
12     print("Essa letra é uma consoante, número ou
caractere
especial.")
```

8.

```
01 def somatorio(valor):  
02     soma = 0  
03     for i in range(valor+1):  
04         soma += i  
05     return soma  
06  
07 numero = int(input("Insira um número:"))  
08  
09 resultado = somatorio(numero)  
10  
11 print("O somatório é:", resultado)
```

9.

```
01 def contaConsoantes(palavra):  
02     qtd = 0  
03     for i in range(len(palavra)):  
04         if palavra[i] not in ('a', 'e', 'i', 'o',  
05 'u'):  
06             qtd += 1  
07     return qtd  
08  
09 string = input("Insira uma palavra:")  
10  
11 print("O numero de consoantes é:", contaConsoantes(s-  
tring))
```

10.

```
01 def converte(reais, cotacao):  
02     return reais/cotacao  
03  
04 cot = float(input("Insira a contação de um dolar em  
05 reais:""  
06 ))  
07 qtde = float(input("Insira a quantdade de dinheiro em  
08 reais  
09 :"))  
10  
11 conv = converte(cot, qtde)  
  
    print("A conversão de reais para dólares é: US$",
conv)
```



anotações