PRÁTICAS DE PROGRAMAÇÃO

Dados Internacionais de Catalogação na Publicação (CIP)
(Claudia Santos Costa - CRB 8ª/9050)

Calixto, Gustavo Moreira
Práticas de programação / Gustavo Moreira Calixto. – São Paulo : Editora Senac São Paulo, 2025. – (Série Universitária)

Bibliografia.
e-ISBN 978-85-396-5539-7 (ePub/2025)
e-ISBN 978-85-396-5540-3 (PDF/2025)

1. Programação (Computadores). 2. Python (Linguagem de programação). I. Título. II. Série..

25-2460c

CDD – 005.13
BISAC COM051000
COM051360

Índice para catálogo sistemático:
1. Programação (Computadores) 005.13

PRÁTICAS DE PROGRAMAÇÃO

Gustavo Moreira Calixto

Senac

Administração Regional do Senac no Estado de São Paulo

Presidente do Conselho Regional
Abram Szajman

Diretor do Departamento Regional
Luiz Francisco de A. Salgado

Superintendente Universitário e de Desenvolvimento
Luiz Carlos Dourado

Editora Senac São Paulo

Conselho Editorial
Luiz Francisco de A. Salgado
Luiz Carlos Dourado
Darcio Sayad Maia
Lucila Mara Sbrana Sciotti
Luís Américo Tousi Botelho

Gerente/Publisher
Luís Américo Tousi Botelho

Coordenação Editorial
Verônica Marques Pirani

Prospecção
Andreza Fernandes dos Passos de Paula
Dolores Crisci Manzano
Paloma Marques Santos

Administrativo
Marina P. Alves

Comercial
Aldair Novais Pereira

Comunicação e Eventos
Tania Mayumi Doyama Natal

Coordenação de Arte
Antonio Carlos De Angelis

Coordenação de Revisão de Texto
Marcelo Nardeli

Acompanhamento Pedagógico
Otacília da Paz Pereira

Designer Educacional
Janaina Nascimento Menezes da Silva

Revisão Técnica
Alexandre dos Santos Mignon

Preparação e Revisão de Texto
Allan Moraes
Amanda Carvalho

Projeto Gráfico
Alexandre Lemes da Silva
Emília Corrêa Abreu

Capa
Antonio Carlos De Angelis

Editoração Eletrônica e Ilustrações
Natália da Silva Nakashima
Manuela Ribeiro

Imagens
Adobe Stock Photos

Editora Senac São Paulo
Av. Engenheiro Eusébio Stevaux, 823 – Prédio Editora
Jurubatuba – CEP 04696-000 – São Paulo – SP
Tel. (11) 2187 4450
editora@sp.senac.br
https://www.editorasenacsp.com.br

Sumário

Capítulo 1

Programação estruturada

O aprendizado sobre programação é essencial para os profissionais da área de TIC (tecnologias da informação e comunicação) aperfeiçoarem suas habilidades no uso de ferramentas de elaboração, aplicações e serviços advindas do avanço da computação nos tempos atuais. Programar, hoje em dia, também é uma competência trabalhada em jovens do ensino básico, o que revela que a interação com elementos da computação não está limitada a profissionais de TIC.

Alguns fundamentos de programação são importantes para alcançar um nível básico de conhecimento na área, como a definição de variáveis, entrada e saída de dados, desvios condicionais e laços. No entanto, à medida que os programas se tornam mais complexos, é necessário adotar técnicas que permitam otimizar e reutilizar os artefatos desenvolvidos. De acordo com Forbellone e Eberspächer (2022), o uso de estruturas modulares e a organização do código são fundamentais para garantir a eficiência e a manutenção dos programas, facilitando sua evolução e adaptação a novas demandas.

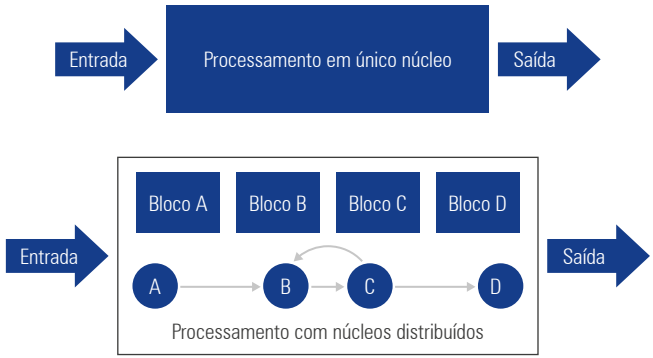
Sem dúvida, a primeira prática para estudantes que desejam alcançar o nível intermediário em programação é a organização de seu código-fonte através de funções e procedimentos, abordagem conhecida como programação estruturada. O primeiro subcapítulo deste livro conceitua e apresenta exemplos de funções e procedimentos, enquanto o segundo mostra os diferentes escopos de variáveis em programas que trabalham com um ou mais subprogramas.

1 Funções e procedimentos

Os algoritmos são conjuntos de instruções finitas organizadas para um propósito específico, como aperfeiçoar tarefas, processar grandes volumes de dados ou monitorar sensores e atuadores em qualquer segmento para uma aplicação ou serviço.

Existem programas desenvolvidos em poucas linhas cuja sequência de instruções é claramente interpretável. Contudo, podem existir situações em que o código-fonte se torna extenso, incorporado a um núcleo único de processamento, com a repetição das mesmas instruções para realizar tarefas idênticas ou similares. Quando um programa chega a esse nível de complexidade, é necessário buscar estratégias para atribuir as devidas responsabilidades a cada trecho de código, evidenciando as etapas do processamento elaborado. A figura 1 ilustra a condição de um programa cujas instruções estão agrupadas em um único núcleo e a organização que se pretende alcançar. Observe que um mesmo programa pode ter sua responsabilidade descentralizada em blocos que podem ser invocados na ordem e quantidade de vezes estabelecidas pelo desenvolvedor. Práticas de programação estruturada favorecem a organização do código e facilitam o entendimento para futuras ações de refatoração (melhora do código-fonte sem alteração de suas funcionalidades).

Figura 1 – Comparação entre programa com núcleo único e programa com núcleos distribuídos



Um exemplo prático é apresentado para um programa que verifica as aprovações de alunos informando o nome e a nota.

```
# Código sem modularização (bloco único)
nome = "João"
nota1 = 8.0
nota2 = 7.5
nota3 = 6.0

media = (nota1 + nota2 + nota3) / 3

if media >= 7.0:
    resultado = "Aprovado"
else:
    resultado = "Reprovado"

print(f"Aluno: {nome}")
print(f"Média: {media:.2f}")
print(f"Resultado: {resultado}")
```

Observe que esse programa utiliza práticas básicas de entrada, variáveis, desvio condicional e saída em bloco único. O código-fonte pode crescer para processar em maior volume e detalhes de dados, exigindo-se para isso a distribuição de responsabilidades. Veja, portanto, o mesmo

código com uma abordagem de programação estruturada reorganizado em funções e procedimentos:

```
# Código otimizado com funções

def calcular_media(n1, n2, n3):
    """Calcula a média de três notas."""
    return (n1 + n2 + n3) / 3

def verificar_aprovacao(media, media_aprovacao=7.0):
    """Verifica se a média é suficiente para aprovação."""
    return "Aprovado" if media >= media_aprovacao else
"Reprovado"

def exibir_resultado(nome, media, resultado):
    """Exibe o nome do aluno, sua média e o status de
aprovação."""
    print(f"Aluno: {nome}")
    print(f"Média: {media:.2f}")
    print(f"Resultado: {resultado}")

# Variáveis
nome = "João"
nota1 = 8.0
nota2 = 7.5
nota3 = 6.0

# Processamento
media = calcular_media(nota1, nota2, nota3)
resultado = verificar_aprovacao(media)
exibir_resultado(nome, media, resultado)
```

Constate que a resolução do programa é dividida em três partes: calcula_media, verificar_aprovacao e exibir_resultados. Logo após, essas partes são invocadas de maneira a efetuar o processamento desejado. Para as práticas de programação em programas complexos, a divisão de responsabilidades em funções e procedimentos resulta em uma melhor organização e, na maioria dos casos, contribui para um desempenho mais eficiente. Segundo Guilhon *et al*. (2022), a modularização do código

é uma abordagem essencial para garantir a escalabilidade e a manutenibilidade das aplicações, permitindo que diferentes partes do programa sejam desenvolvidas, testadas e aprimoradas de forma independente.

Perceba que, no código-fonte do exemplo apresentado, a função "verificar_aprovacao" possui um parâmetro de entrada "media_aprovacao = 7.0". Essa prática é chamada de parâmetros default ou parâmetros padrão e permite adotar o valor informado como padrão para executar a função, caso não seja informado um valor para esse parâmetro. No caso da função "verificar_aprovacao", observe que ela é executada sem que se informe o segundo parâmetro "media_aprovacao", adotando o valor 7.0 como padrão. No entanto, se a média de aprovação fosse diferente de 7.0, o novo valor poderia ser informado no segundo parâmetro.

Conceitualmente, qual é a diferença entre uma função e um procedimento? Ambos são subprogramas na programação estruturada, mas possuem diferenças. Uma função é um bloco de código reutilizável que executa uma tarefa específica. As funções ajudam a organizar o código, tornando-o mais modular, reutilizável e legível. A sintaxe básica para definir uma função em Python é:

```
def nome_da_funcao(parametro1, parametro2, ...):
    """Docstring opcional explicando a função"""
    # Bloco de código da função
    return resultado
```

É importante ressaltar que uma função (como nos ensinamentos da matemática) necessita de um retorno: um resultado do processamento realizado no subprograma. A função também pode receber parâmetros para o fornecimento de elementos a serem utilizados na execução do subprograma. Veja o exemplo a seguir do uso de uma função:

```
# Definição da função
def saudacao(nome):
    """Retorna uma saudação personalizada para o usuário."""
    return f"Olá, {nome}! Bem-vindo ao mundo da programação."
```

```
# Chamando a função e exibindo o resultado
usuario = “Alice”
mensagem = saudacao(usuario)
print(mensagem)
```

Observe que a função retorna uma mensagem de saudação que possui como parâmetro o nome, ou seja, ela depende do nome para a devida execução. Na programação estruturada, é natural que, primeiramente, as funções e os procedimentos sejam projetados e desenvolvidos para depois serem invocados nos pontos desejados do programa principal.

Já o procedimento é uma função que executa uma ação, mas não retorna um valor explícito. A sintaxe de um procedimento é a mesma de uma função, mas sem a expressão *return*.

```
def nome_do_procedimento(parametros):
        '''Executa uma ação sem retornar um valor'''
    # Bloco de código da função
```

Os procedimentos são uma maneira de construir subprogramas quando o propósito é o retorno direto para a saída do programa, como apresenta o exemplo a seguir. Observe que a saudação é exibida diretamente por um comando de saída de terminal sem o uso da palavra reservada *return*.

```
# Definição do procedimento
def exibir_saudacao(nome):
    """Exibe uma saudação na tela sem retornar um valor."""
    print(f"Olá, {nome}! Seja bem-vindo!")

# Chamando o procedimento
exibir_saudacao("Lucas")
exibir_saudacao("Mariana")
```

IMPORTANTE

O uso de funções e procedimentos em Python é fundamental para a organização, a reutilização e a manutenção do código, pois permite dividir um programa em blocos menores e mais compreensíveis. Funções são a opção ideal quando há necessidade de calcular e retornar valores, enquanto procedimentos executam ações sem necessariamente retornar algo. Essa separação melhora a legibilidade, evita repetições desnecessárias e facilita a depuração, tornando o código mais modular e eficiente. Em aplicações grandes, essa abordagem reduz a complexidade, favorece o reaproveitamento de código e permite que diferentes partes do sistema sejam desenvolvidas e testadas de forma independente. É importante reforçar que, conceitualmente, na linguagem de programação Python, os procedimentos são adaptações em uma única estrutura de função. Assim, mesmo que o código-fonte do procedimento não tenha a expressão *return*, o valor "None" é retornado ao final da execução.

2 Escopo de variáveis

Ao trabalhar com funções e procedimentos, é importante entender que os recursos computacionais alocados por subprogramas são liberados ao final da sua execução dentro de um contexto de seu processo. Em linhas gerais, uma função ou procedimento pode ter variáveis declaradas dentro do seu bloco de execução, limitadas ao uso dentro do seu bloco. Todavia, esse subprograma pode ter acesso a variáveis declaradas no bloco principal de execução do programa. Observe o exemplo a seguir:

```
# Definição de uma variável global
mensagem_global = "Olá, bem-vindo ao programa!"

def saudacao(nome):
    """Função que usa uma variável global e uma variável
local."""
```

```
    mensagem_local = f"Olá, {nome}! Tenha um ótimo dia!"  #
Variável local
    print(mensagem_global)  # Acessando a variável global
    print(mensagem_local)   # Acessando a variável local

# Chamando a função
saudacao("Lucas")
```

As variáveis podem ser classificadas em dois tipos: globais e locais. As variáveis globais são declaradas fora das funções e podem ser acessadas de qualquer parte do código, inclusive dentro de funções e procedimentos. Por outro lado, as variáveis locais são definidas dentro de uma função e só podem ser utilizadas no seu escopo. Segundo Lambert (2022), compreender essa distinção é essencial para evitar conflitos no código e garantir que as variáveis sejam manipuladas de forma segura e eficiente.

Veja que, no exemplo apresentado, a variável mensagem_global pode ser acessada dentro da função saudação. Já a variável mensagem_local só pode ser utilizada durante a execução do bloco da função. Se ao final do código do exemplo anterior fosse adicionado o código a seguir, um erro seria gerado por tentativa de acesso a uma variável local no escopo de uma variável global.

```
# Tentando acessar a variável local fora da função (isso
gerará um erro)

# print(mensagem_local)  # Isso resultaria em um erro pois a
variável é local
```

IMPORTANTE

O uso de variáveis globais em Python pode parecer conveniente, mas apresenta diversos riscos que comprometem a qualidade do código. Como podem ser acessadas e modificadas de qualquer parte do programa, elas dificultam a depuração e, consequentemente, a identificação da origem de erros. Além disso, criam dependências ocultas, tornando as funções menos previsíveis e reduzindo sua reutilização. Outro problema é o risco

de sobrescrita acidental, o que pode levar a comportamentos inesperados e difíceis de corrigir. Para evitar esses problemas, é recomendável priorizar o uso de variáveis locais, passar argumentos explicitamente para funções e encapsular estados em classes, garantindo um código mais seguro, organizado e sustentável.

Uma curiosidade interessante sobre funções em Python é que elas são cidadãs de primeira classe (first-class citizens), o que significa que podem ser atribuídas a variáveis, passadas como argumentos para outras funções e até mesmo retornadas por outras funções. Isso permite a criação de funções anônimas (lambda) e a programação funcional com map(), filter() e reduce(), além de tornar o código mais flexível e reutilizável. Esse recurso é amplamente utilizado em frameworks modernos, como Flask e Django, nos quais as funções são passadas como *callbacks* para manipular requisições HTTP de forma dinâmica.

Considerações finais

O uso de funções e procedimentos é essencial para escrever códigos mais organizados, modulares e eficientes. Essas práticas facilitam a reutilização de código, a separação de responsabilidades e reduzem a complexidade dos sistemas. Além disso, a programação estruturada torna o código mais claro, o que simplifica sua manutenção, depuração e futuras melhorias, sem comprometer a lógica central do programa.

Outro ponto fundamental é o conceito de escopo de variáveis, que ajuda a evitar conflitos e erros inesperados, garantindo que cada variável seja usada no contexto certo. A distinção entre variáveis globais e locais é imprescindível para manter o código organizado, permitindo que funções e procedimentos operem de forma independente quando necessário.

No caso do Python, a linguagem oferece suporte a estruturas modulares e funções anônimas e permite a passagem de funções como

argumentos, tornando essa abordagem ainda mais poderosa. Esses conceitos são amplamente aplicados em frameworks modernos e no desenvolvimento de aplicações escaláveis, reforçando a importância de dominá-los desde o início da jornada na programação.

Ao adotar funções e procedimentos da maneira certa, os desenvolvedores conseguem escrever códigos mais eficientes, reutilizáveis e alinhados com as boas práticas da engenharia de software, criando soluções bem estruturadas e fáceis de evoluir.

Referências

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. **Lógica de programação**: a construção de algoritmos e estruturas de dados com aplicações em Python. 4. ed. São Paulo: Bookman, 2022. *E-book*.

GUILHON, André *et al*. (org.). **Jornada Python**: uma jornada imersiva na aplicabilidade de uma das mais poderosas linguagens de programação do mundo. Rio de Janeiro: Brasport, 2022. *E-book*.

LAMBERT, Kenneth A. **Fundamentos de Python**: estruturas de dados. Porto Alegre: Cengage Learning, 2022. *E-book*.