


# Capítulo 1: Introdução à programação em Python

## Contents

- O que é programação de computadores?
- O que você precisa para começar?
- Escrevendo seu primeiro programa em Python
- Como um programa em Python funciona?
- Estrutura básica de um programa em Python
- Exemplo aprimorado: capturando nome e idade
- Estratégias para Estudar Programação
-  Exercícios
- Referências



# O que é programação de computadores?

Programar computadores é a arte e ciência de conceber e criar conjuntos de instruções que capacitam computadores a realizar tarefas específicas. Esse processo envolve a expressão lógica de algoritmos por meio de uma linguagem de programação, atuando como a ponte entre a mente humana e a máquina.

Essa habilidade é fundamental para aqueles que buscam atuar no universo da computação, desempenhando um papel essencial em diversas disciplinas, como engenharia, ciência, negócios, saúde, educação e entretenimento. A capacidade de programar não apenas possibilita a automação de processos, mas também estimula a resolução criativa de problemas e impulsiona a inovação tecnológica.

Na prática da programação, os desenvolvedores convertem conceitos abstratos em linguagem compreensível pelos computadores, proporcionando-lhes a habilidade de executar tarefas complexas. Essa interação entre humanos e máquinas desempenha um papel fundamental na

contínua evolução da sociedade digital, moldando desde avanços científicos até transformações sociais significativas.

A habilidade de programar transcende a mera condição técnica, transformando-se em uma ferramenta importante para explorar novas ideias e o aprimoramento pessoal. Filosoficamente falando, programar é também uma forma de enxergar o mundo sob diferentes perspectivas, processos e abstrações.

Dentro do contexto do Python, esta é uma linguagem de alto nível, interpretada e multiparadigma. Isso significa que o Python se destaca pela facilidade de aprendizado e uso, sendo aplicável a uma ampla gama de propósitos. Sua versatilidade é reforçada pela capacidade de suportar diversos paradigmas de programação, proporcionando aos desenvolvedores uma abordagem flexível e adaptável para resolver problemas em diferentes domínios [[Foundation, 2024](#)].

O Python desempenha papéis significativos em diversas áreas. A seguir, um breve resumo de algumas dessas possibilidades:

- **Engenharia e Ciências:** Utilizado em simulação, análise e visualização de dados, além de ser amplamente empregado em projetos de aprendizado de máquina. Sua sintaxe clara e concisa, juntamente com uma extensa biblioteca de módulos científicos, o torna uma escolha popular para essas aplicações [[McKinney, 2017](#)].
- **Negócios:** Ferramenta essencial para análise de dados, automação de processos e desenvolvimento de aplicativos web. A flexibilidade e eficiência do Python o tornam versátil para soluções empresariais [[Sweigart, 2020](#)].
- **Educação:** Considerado a linguagem de programação mais popular para o ensino em escolas e universidades, sua sintaxe simples e intuitiva facilita o aprendizado, mesmo para iniciantes [[Matthes, 2019](#)].
- **Entretenimento:** Empregado na criação de jogos, aplicativos móveis e outros softwares de entretenimento. A robustez e flexibilidade do Python permitem o desenvolvimento de aplicações de alta qualidade em diversas áreas [[Lutz, 2013](#)].
- **Saúde:** Amplamente utilizado em análise de dados médicos, desenvolvimento de softwares especializados e pesquisa médica. Sua capacidade analítica e adaptabilidade o tornam uma ferramenta valiosa para inovação e pesquisa em saúde [[Foundation, n.d.](#)].

Além dessas áreas, o Python atende a uma ampla gama de necessidades, desde a automatização de tarefas repetitivas até desafios avançados. Suas aplicações abrangem:

- **Automação de tarefas e processos**
- **Análise e visualização de dados**

- **Desenvolvimento de jogos**
- **Inteligência artificial e aprendizado de máquina**
- **Automação de redes e segurança cibernética**
- **Desenvolvimento de aplicativos de desktop e web**
- **Construção de APIs**
- **Simulações científicas e matemáticas**
- **Internet das Coisas (IoT)**
- **Produção e manipulação de mídia**

A flexibilidade do Python o torna uma ferramenta indispensável em diversas áreas, oferecendo uma base sólida para a inovação no cenário tecnológico atual. Dominar Python não é apenas uma habilidade essencial, mas também uma forma de explorar as constantes inovações e desafios deste mundo tecnológico em rápida evolução. De acordo com o ranking atualizado da IEEE Spectrum para 2024, o Python continua consolidando sua posição entre as linguagens de programação mais influentes e utilizadas, impulsionado por bibliotecas e frameworks que atendem áreas emergentes como a inteligência artificial ([spectrum.ieee.org/top-programming-languages-2024](https://spectrum.ieee.org/top-programming-languages-2024))

## O que você precisa para começar?

Para iniciar seu aprendizado em Python, além desta documentação, é fundamental contar com os seguintes requisitos:

### 1. **Computador com Acesso à Internet:**

Recomenda-se utilizar um computador com conexão à internet para facilitar o download de pacotes adicionais e o acesso à documentação online, enriquecendo sua experiência de aprendizado. Contudo, é possível programar em Python mesmo em ambientes offline, o que pode ser útil quando a conexão não estiver disponível.

### 2. **Editor de Texto ou IDE (Ambiente de Desenvolvimento Integrado):**

Escolha um editor ou IDE que atenda às suas preferências e necessidades. Pode ser algo simples, como o Notepad, ou opções mais avançadas, como o [Sublime Text](#), [Visual Studio Code](#), [Cursor](#), ou editores online como o [Replit](#), [Google Colab](#) e [Jupyter Notebook](#). Além disso, o [PyCharm](#) é uma poderosa IDE específica para Python, que oferece recursos avançados e é amplamente utilizada por desenvolvedores. Se você prefere uma experiência altamente customizável e baseada em terminal, o [NeoVim](#) é uma excelente alternativa.

### 3. **Interpretador Python:**

Faça o download do interpretador Python diretamente do site da Python Software

Foundation ([python.org](https://python.org)). Alternativamente, você pode utilizar ambientes online, como o Replit, que já incluem um interpretador Python integrado a um editor de texto.

Equipado com esses recursos, você estará pronto para explorar e aprimorar suas habilidades em Python. Seja trabalhando localmente em seu computador ou em ambientes online, você terá a flexibilidade necessária para mergulhar no mundo da programação, adaptando-se ao seu estilo de aprendizado.

**Observação:** Por ser interpretada, o código Python é executado diretamente pelo interpretador sem necessidade de compilação, tornando o aprendizado mais rápido e prático, principalmente para iniciantes.

## Escrevendo seu primeiro programa em Python

Vamos criar um programa simples em Python que soma dois números e mostra o resultado. O arquivo pode ser chamado de "soma.py" e o código é este:

```
a = 1
b = 2
soma = a + b
print(soma)
```

Ao rodar o programa, ele vai exibir:

```
3
```

### Explicação do código:

- `a = 1`: Define o valor 1 para a variável `a`.
- `b = 2`: Define o valor 2 para a variável `b`.
- `soma = a + b`: Soma os valores de `a` e `b` e guarda o resultado (3) na variável `soma`.
- `print(soma)`: Mostra o valor de `soma` na tela (3).

### O que são variáveis?

Variáveis guardam dados para serem usados no programa. Aqui, `a`, `b` e `soma` são variáveis.

# O que são funções?

Funções realizam tarefas específicas. A função `print` mostra algo na tela.

## Como rodar o programa:

1. Escreva o código em um editor de texto.
2. Salve como "soma.py".
3. Abra o terminal ou prompt de comando.
4. Vá até a pasta onde o arquivo foi salvo.
5. Execute o código com o comando:

```
python soma.py
```

6. O resultado será o número 3, que é a soma de 1 e 2.

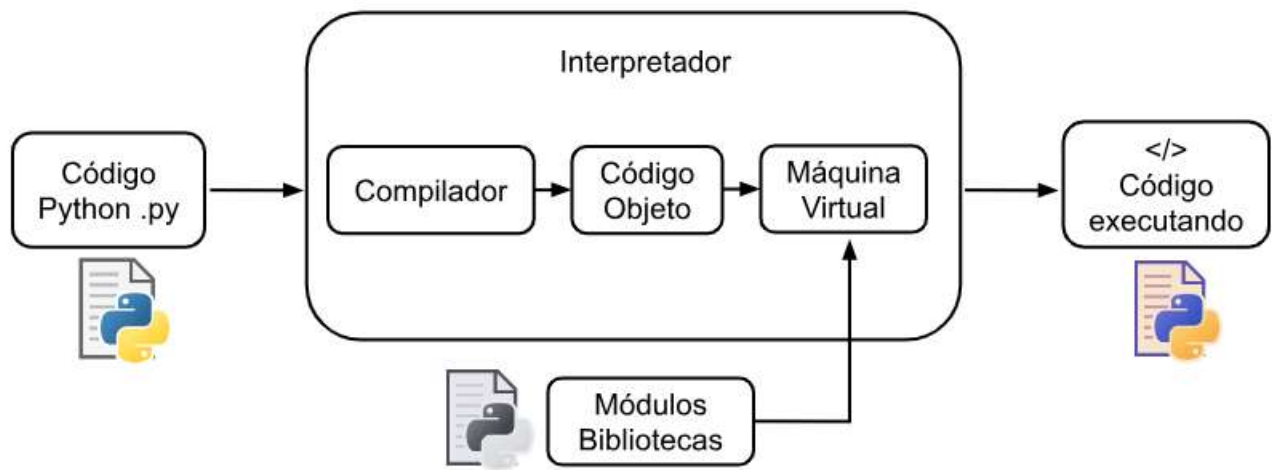
## Como um programa em Python funciona?

A execução de um programa em Python passa por várias etapas importantes. Tudo começa com o desenvolvimento do código-fonte, que é normalmente salvo em arquivos com extensão ".py". Esse código é, então, enviado ao interpretador Python.

O **interpretador** é responsável por ler e processar o código Python. Ele transforma o código em **bytecode**, uma forma intermediária que pode ser executada em diferentes tipos de hardware. O bytecode é uma representação mais simples, que será executada pela Máquina Virtual Python (PVM).

A **PVM (Máquina Virtual Python)** é o componente que realmente roda o programa. Ela executa o bytecode, gerencia a memória e interage com o sistema operacional. Em algumas versões do Python, como o **CPython**, o interpretador pode usar um **Compilador Just-In-Time (JIT)**, que melhora o desempenho do programa durante a execução [[Lutz, 2013](#)].





**Figura: Funcionamento Interno do Python.**

## Estrutura básica de um programa em Python

Um programa em Python segue uma estrutura simples, composta por um algoritmo, que é uma sequência de passos definidos para realizar uma tarefa. Assim como uma receita de cozinha, o algoritmo envolve **entrada** (dados), **processamento** (etapas a serem seguidas) e **saída** (resultado).

A estrutura básica de um programa Python é:

```
def main():  
    # Bloco de código principal  
  
if __name__ == "__main__":  
    main()
```

- `main()`: A função principal do programa, chamada quando o código é executado.
- **Bloco de código principal**: É onde o programa faz seu trabalho, e o código é indentado para indicar que pertence ao bloco principal.
- `__name__`: Variável especial que identifica o contexto de execução do arquivo. Se o arquivo está sendo executado diretamente, `__name__` será `"__main__"`, mas se for importado como um módulo, `__name__` terá o nome do arquivo sem a extensão `.py`.
- `if __name__ == "__main__":`: Verifica se o arquivo está sendo executado diretamente como o programa principal. Se sim, a função `main()` é chamada.

Essa verificação (`if __name__ == "__main__":`) evita que código indesejado seja executado ao importar o arquivo como um módulo, garantindo que a função `main()` só rode quando o script for executado diretamente.

## Exemplo de um programa simples:

```
def main():  
    print("Hello, world!")  
  
if __name__ == "__main__":  
    main()
```

Para executar o programa, siga estes passos:

- Salve o código em um arquivo com extensão “.py” (por exemplo, “hello\_world.py”).
- No terminal ou prompt de comando, execute o seguinte:

```
python hello_world.py
```

## Dica: A importância da indentação

Em Python, a indentação não é apenas uma questão de estilo, mas uma parte essencial da sintaxe. Ela define a estrutura do programa e o agrupamento dos blocos de código. O uso de quatro espaços por nível de indentação é recomendado pela [PEP 8](#), o guia oficial de estilo do Python. Seguir essa recomendação não só melhora a legibilidade, como também mantém o código organizado e fácil de entender.

## Exemplo aprimorado: capturando nome e idade

Vamos melhorar nosso código inicial para aprender a interagir com o usuário. Neste exemplo, o programa vai solicitar o nome e a idade do usuário, armazenar essas informações e depois exibi-las na tela. Siga os passos abaixo:

- Crie um novo arquivo para o código.
- Insira o seguinte código no arquivo:

```
nome = input("Qual é o seu nome? ")  
idade = input("Qual é a sua idade? ")  
print("Seu nome é:", nome)  
print("Você tem", idade, "anos.")
```



O código usa a função `input` para pedir que o usuário insira seu nome e idade. As informações são guardadas nas variáveis `nome` e `idade`. Depois, o programa imprime esses dados formatados.

- Salve o arquivo e execute-o com o Python.
- Quando você rodar o programa, ele vai pedir o nome e a idade:

```
Qual é o seu nome? Ana Maria
Qual é a sua idade? 25
```

- Após inserir as informações, o programa vai mostrar a saída:

```
Seu nome é: Ana Maria
Você tem 25 anos.
```

Neste exemplo, você aprende a usar a função `input` para receber dados do usuário e a função `print` para mostrar essas informações. Isso é essencial para criar programas interativos que respondem ao que o usuário insere.

## Comentando seu código

Comentar o código é essencial para facilitar sua compreensão, tanto para você quanto para outros que possam ler ou trabalhar no mesmo projeto. Em Python, usamos o símbolo `#` para comentários de uma única linha, e `'''` ou `"""` para comentários que ocupam várias linhas.

## Exemplos:

### Comentário de uma linha:

```
# Solicita o nome do usuário
nome = input("Qual é o seu nome? ")
```

### Comentário de várias linhas com aspas simples:

```
'''
Este bloco de código solicita a idade do usuário
e armazena o valor na variável 'idade'.
'''
idade = input("Qual é a sua idade? ")
```

### Comentário de várias linhas com aspas duplas:

```
"""
A seguir, exibimos o nome e a idade inseridos pelo usuário.
Esta parte do código é responsável pela exibição dos dados.
"""
print("Nome:", nome)
print("Idade:", idade)
```

### Comentário explicando a lógica do código:

```
# Verifica se o usuário é maior de 18 anos para determinar a elegibilidade
elegivel = int(idade) > 18
```

## Dicas importantes:

- **Seja claro e objetivo:** Use os comentários para explicar a lógica e as intenções do código, não para descrever ações óbvias que o código já explica por si só.
- **Evite excessos:** Comentários são úteis, mas comentários em excesso podem poluir o código. Use-os com moderação para explicar partes mais complexas ou decisões importantes.
- **Atualize seus comentários:** Sempre que modificar o código, lembre-se de atualizar os comentários para evitar inconsistências.

Com boas práticas de comentário, o código se torna mais fácil de manter, entender e colaborar, especialmente em projetos maiores ou quando compartilhado com outras pessoas.

## Estratégias para Estudar Programação

Aprender a programar envolve não só compreender a lógica e a sintaxe, mas também otimizar a maneira como nosso cérebro processa e retém informações. Entender o funcionamento da **memória de trabalho** (curto prazo) e da **memória de longo prazo** pode transformar sua forma de estudar [[Oakley and Sejnowski, 2021](#)]

- **Memória de trabalho:** Responsável por armazenar temporariamente informações enquanto realizamos uma tarefa. Tem capacidade limitada e é essencial para o raciocínio lógico e resolução de problemas.
- **Memória de longo prazo:** Onde as informações são armazenadas de forma mais duradoura, permitindo a recuperação do conhecimento adquirido ao longo do tempo.

Além disso, a Inteligência Artificial (IA) e ferramentas de anotação, como o [Obsidian](#), podem ser grandes aliadas nesse processo.

# Repetição Espaçada e Consolidação da Memória

Revisar conteúdos em intervalos crescentes permite que as informações sejam transferidas da memória de trabalho para a memória de longo prazo, consolidando o aprendizado.

**Aplicação em Programação:** Revise seus códigos e materiais periodicamente para fixar o aprendizado e evitar o esquecimento.

[\[Cepeda et al., 2006\]](#)

## Interleaving: Mistura de Tópicos

Alternar entre diferentes assuntos evita a sobrecarga da memória de trabalho e fortalece a criação de conexões duradouras.

**Aplicação em Programação:** Alterne entre linguagens ou paradigmas, combinando teoria e prática para desenvolver uma aprendizagem mais flexível.

[\[Rohrer, 2012\]](#)

## Prática Ativa e Uso Estratégico da IA

Programar ativamente, em vez de apenas ler, fortalece a memória de trabalho e, com a revisão, a memória de longo prazo.

**Aplicação em Programação:** Desenvolva pequenos projetos e resolva problemas reais. Utilize a IA para obter exemplos práticos, esclarecer dúvidas e sugerir melhorias em seu código, sempre tentando resolver os desafios por conta própria primeiro.

[\[Ericsson and Pool, 2016\]](#)

## Compreensão Profunda com Apoio da IA

Buscar entender o “porquê” e o “como” dos algoritmos favorece a fixação do conhecimento na memória de longo prazo.

**Aplicação em Programação:** Analise a lógica por trás do código e dos algoritmos e peça à IA explicações alternativas e exemplos para reforçar sua compreensão.

[\[Chi and Wylie, 2014\]](#)

## Foco e Minimização de Interrupções

Manter a concentração ajuda a preservar a limitada capacidade da memória de trabalho e facilita a transferência de informações para a memória de longo prazo.

**Aplicação em Programação:** Use técnicas como o método Pomodoro para períodos

concentrados de estudo e permita que a IA monitore seu tempo, sugerindo pausas estratégicas. Além disso, evite manter o smartphone ao alcance da visão durante os estudos, pois mesmo sua presença pode comprometer a atenção e reduzir o foco. [[Baumann and Kuhl, 2005](#)] [[Ward et al., 2017](#)]

## Aprendizagem Baseada em Problemas

Resolver desafios práticos ativa tanto a memória de trabalho quanto a de longo prazo, ajudando a fixar o aprendizado.

**Aplicação em Programação:** Envolver-se com problemas reais em plataformas como [LeetCode](#). [[Hmelo-Silver, 2004](#)]

## Recursos Diversificados, Reflexão e Ferramentas de Anotação

Combinar múltiplas fontes, como livros, vídeos e cursos online, enriquece o aprendizado e estimula a retenção.

**Aplicação em Programação:**

- Explore diferentes formatos e, em seguida, reflita sobre o que aprendeu escrevendo resumos ou discutindo com colegas.
- Utilize o [Obsidian](#) para organizar suas anotações.
- A IA pode ajudar a integrar informações de diversas fontes em resumos concisos.

[[Mayer, 2014](#)] [[Boud et al., 1985](#)]

## Revisão Regular e Aprendizagem Colaborativa

Revisar continuamente o material combate o esquecimento e fortalece as conexões neurais, enquanto a colaboração amplia a compreensão dos temas.

**Aplicação em Programação:**

- Planeje revisões periódicas com ferramentas de repetição espaçada, como [RemNote](#).
- Participe de grupos de estudo ou fóruns (por exemplo, [Stack Overflow](#)).
- Utilize a IA para mediar discussões e fornecer feedback adicional.

[[Karpicke and Blunt, 2011](#)] [[Dillenbourg, 1999](#)]

Integrar essas estratégias com o entendimento dos mecanismos da memória, o uso inteligente da IA e o aproveitamento de ferramentas pode acelerar seu aprendizado em programação. A

prática constante, a revisão e a aplicação ativa dessas técnicas são fundamentais para transformar a teoria em habilidades duradouras.



## Exercícios

Antes de iniciar os exercícios, é importante esclarecer que todos seguem um padrão específico de entradas e saídas. O sistema irá fornecer entradas simulando a interação de um usuário, e seu programa deverá produzir as saídas corretas para garantir que a lógica e o processamento foram implementados adequadamente.

1. Escreva um programa que imprime a famosa mensagem do mundo da programação.

Neste exercício, você deve simplesmente exibir uma mensagem na tela. Não é necessário ler nenhuma entrada do usuário, apenas utilizar o comando print para exibir o texto desejado.

```
# Teste 1
Saída: Olá Mundo!
```

2. Neste exercício, você deve ler duas entradas: o nome de um aluno e sua matrícula. Em seguida, exiba uma mensagem de boas-vindas formatada com esses dados.

```
# Teste 1
Entrada:
Python da Silva
2024123456
Saída: Olá Python da Silva Matrícula: 2024123456 Seja bem vindo!
```

3. Informações de um Pedido. Crie um programa que deve ler quatro entradas do usuário:

- Nome do cliente
- Produto comprado
- Quantidade adquirida
- Valor unitário do produto

Em seguida, exiba uma mensagem formatada informando os detalhes da compra, incluindo o valor total.

```
# Teste 1
Entrada:
Ana Souza
Livro de Python
1
45.50
Saída:
Pedido confirmado: Livro de Python
Valor total: R$ 45.50
Obrigado pela preferência!
```

📌 **Observação Importante:** Neste exercício, perceba que todas as entradas do usuário são inicialmente tratadas como strings. Para realizar cálculos corretamente, é necessário converter os valores apropriados para números.

🏆 **Desafio:** E se fossem 3 livros em vez de 1? Como calcular o valor final corretamente?

## Referências

- [BK05] Nicole Baumann and Julius Kuhl. How to resist temptation: the effects of external control versus autonomy support on self-regulatory dynamics. *Journal of Personality*, 73(2):443–470, 2005.
- [BKW85] David Boud, Rosemary Keogh, and David Walker. *Reflection: Turning Experience into Learning*. Kogan Page, 1985.
- [CPV+06] Nicholas J. Cepeda, Harold Pashler, Edward Vul, John T. Wixted, and Doug Rohrer. Distributed practice in verbal recall tasks: a review and quantitative synthesis. *Psychological Bulletin*, 132(3):354–380, 2006.
- [CW14] Michelene T. H. Chi and Ruth Wylie. The icap framework: linking cognitive engagement to active learning outcomes. *Educational Psychologist*, 49(4):219–243, 2014.
- [Dil99] Pierre Dillenbourg. What do you mean by collaborative learning? In *Collaborative-learning: Cognitive and Computational Approaches*, pages 1–19. Elsevier, 1999.
- [EP16] K. Anders Ericsson and Robert Pool. *Peak: Secrets from the New Science of Expertise*. Houghton Mifflin Harcourt, 2016.
- [Fou] Python Software Foundation. Python in healthcare. <https://www.python.org/about/success/#healthcare>.
- [Fou24] Python Software Foundation. *Python 3 Documentation*. 2024. Accessed: 2024-09-29. URL: <https://docs.python.org/3/tutorial/index.html>.
- [HS04] Cindy E. Hmelo-Silver. Problem-based learning: what and how do students learn? *Educational Psychology Review*, 16(3):235–266, 2004.



- [KB11] Jeffrey D. Karpicke and Janell R. Blunt. Retrieval practice produces more learning than elaborative studying with concept mapping. *Science*, 331(6018):772–775, 2011.
- [Lut13](1,2) Mark Lutz. *Learning Python*. O'Reilly Media, Inc., 2013.
- [Mat19] Eric Matthes. *Python Crash Course*. No Starch Press, 2nd edition, 2019.
- [May14] Richard E. Mayer. Cognitive theory of multimedia learning. In *The Cambridge Handbook of Multimedia Learning*, pages 43–71. Cambridge University Press, 2014.
- [McK17] Wes McKinney. *Python for Data Analysis*. O'Reilly Media, Inc., 2nd edition, 2017.
- [OS21] Barbara Oakley and Terrence J Sejnowski. *Uncommon sense teaching: Practical insights in brain science to help students learn*. Penguin, 2021.
- [Roh12] Doug Rohrer. Interleaving helps students distinguish among similar concepts. *Educational Psychology Review*, 24(3):355–367, 2012.
- [Swe20] Al Sweigart. *Automate the Boring Stuff with Python*. No Starch Press, 2nd edition, 2020.
- [WDGB17] A. F. Ward, K. Duke, A. Gneezy, and M. W. Bos. Brain drain: the mere presence of one's own smartphone reduces available cognitive capacity. *Journal of the Association for Consumer Research*, 2(2):140–154, 2017.