

# **SISTEMAS DE COMPUTAÇÃO**

## **MÓDULOS DE UM SISTEMA OPERACIONAL**

Os módulos de um sistema operacional podem variar dependendo do sistema operacional em questão, mas de forma geral, um sistema operacional pode incluir os seguintes módulos:

## **NÚCLEO(KERNEL)**

Núcleo (Kernel) - é o módulo central do sistema operacional, responsável por gerenciar recursos de hardware e software, como a CPU, memória, dispositivos de entrada e saída, e processos.

# GERENCIADOR DE MEMÓRIA

Gerenciador de Memória - responsável por gerenciar o acesso à memória do sistema, alocando espaço para processos e gerenciando a memória virtual.



# GERENCIADOR DE PROCESSOS

Gerenciador de Processos - responsável por gerenciar a execução de processos, incluindo a criação, término e escalonamento de processos.

# **GERENCIANDOR DE ARQUIVOS**

Gerenciador de Arquivos - responsável por gerenciar o acesso a arquivos e diretórios no sistema de arquivos.



# **GERENCIADOR DE SEGURANÇA**

Gerenciador de segurança: garante a segurança do sistema operacional, protegendo-o contra ameaças externas e internas. Ele controla o acesso aos recursos do sistema, como arquivos, dispositivos e serviços, e fornece mecanismos de autenticação e criptografia.

# BIBLIOTECAS DO SISTEMA

Bibliotecas do sistema: são coleções de funções e rotinas que os aplicativos podem usar para acessar os recursos do sistema operacional, como a E/S, a rede e a memória.



# DRIVERS DE DISPOSITIVOS

Drivers de dispositivos: são programas que permitem que o sistema operacional se comunique com os dispositivos de hardware, como a placa de som, a placa de rede e a placa de vídeo. Eles fornecem uma interface entre o hardware e o sistema operacional.

# UTILITÁRIOS DE SISTEMA

Utilitários de sistema: são programas que ajudam o usuário a gerenciar e manter o sistema operacional, como ferramentas de backup, gerenciadores de disco, utilitários de diagnóstico e ferramentas de segurança.

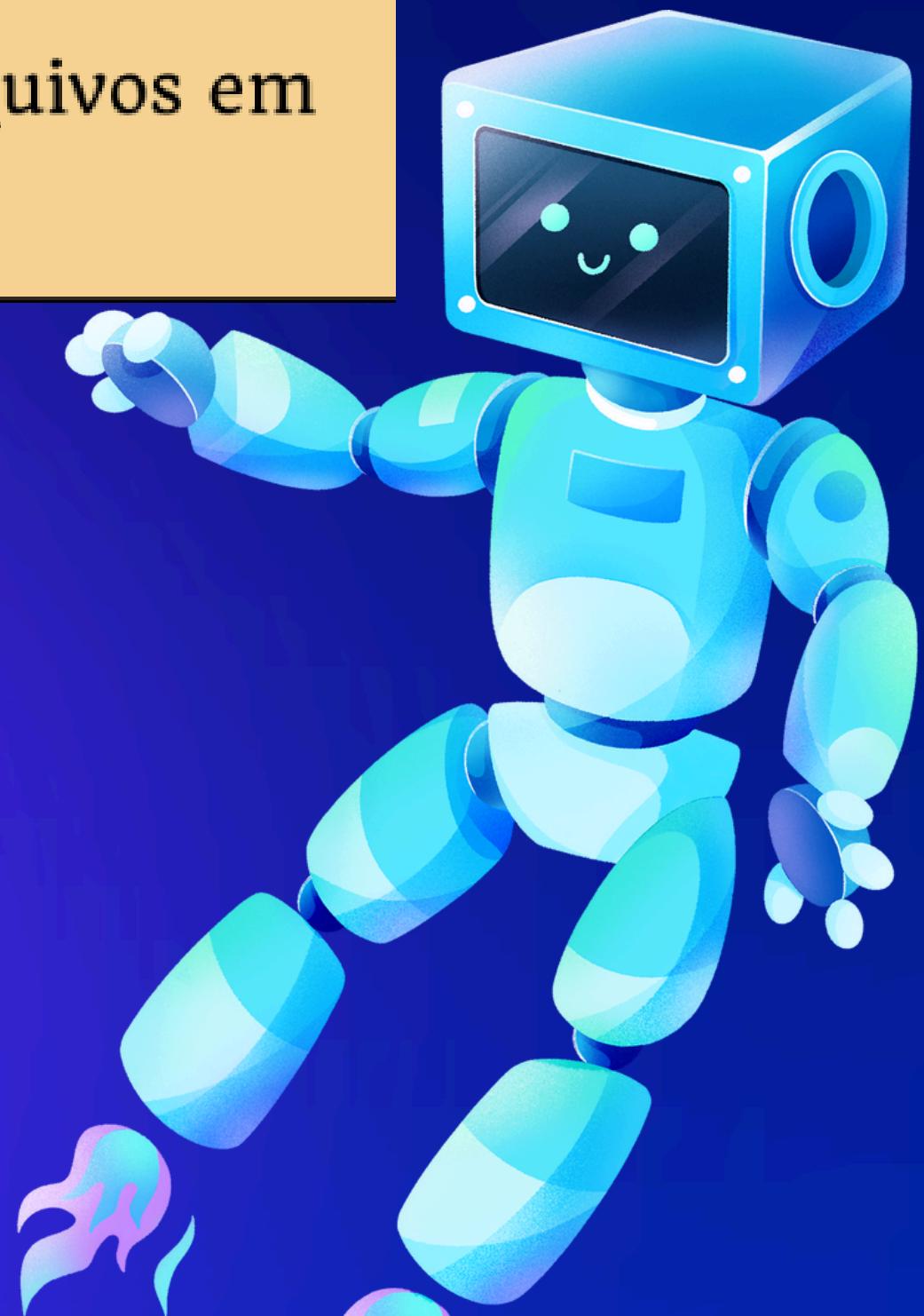


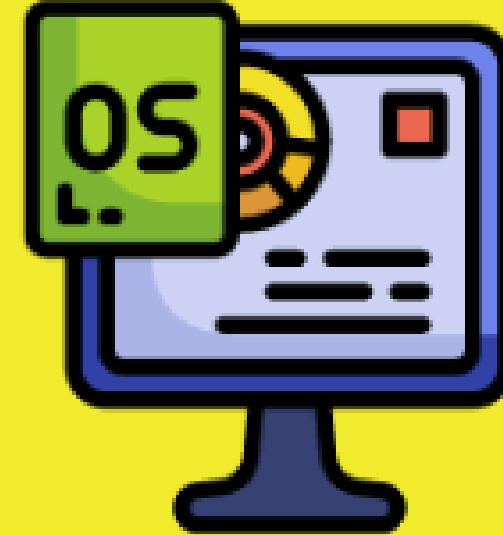
# EXERCÍCIOS

Qual é a função do núcleo do sistema (kernel) em um sistema operacional ?

Quais são as responsabilidades do gerenciador de dispositivos em um sistema operacional ?

Explique como o gerenciador de arquivos organiza e armazena arquivos em um sistema operacional ?





# SISTEMAS OPERACIONAIS



# Sistemas operacionais



Existem várias famílias de sistemas operacionais, cada uma com suas próprias características e abordagens. Principais famílias de sistemas operacionais:

# Unix

# UNIX

Unix é uma família de sistemas operacionais que se originou na década de 1970 e é conhecida por sua estabilidade e segurança. O Unix é usado em servidores, estações de trabalho e dispositivos móveis e é conhecido por sua flexibilidade e facilidade de automação.

# Unix

## VANTAGEM

- Multiusuário e Multitarefa
- Estabilidade e Confiabilidade
- Segurança
- Portabilidade
- Ampla Gama de Ferramentas e Aplicativos

## DESVANTAGEM

- Curva de Aprendizado
- Fragmentação
- Suporte Limitado para Jogos e Software de Desktop
- Gerenciamento de Hardware Específico

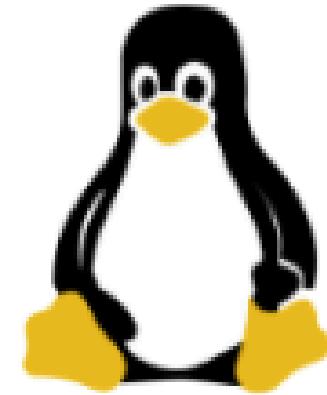
# Conclusão



Apesar de suas desvantagens, o Unix continua sendo uma escolha poderosa e flexível para uma ampla variedade de casos de uso, desde servidores de grande escala até sistemas embarcados e dispositivos móveis. Sua estabilidade, segurança e flexibilidade o tornam uma escolha popular para muitos desenvolvedores e administradores de sistemas.

# Linux

# Linux™



O Linux é um sistema operacional de código aberto baseado em Unix que foi criado na década de 1990 por Linus Torvalds. O Linux é amplamente usado em servidores, dispositivos embarcados e desktops e é conhecido por sua flexibilidade e personalização.

# Linux

## VANTAGEM

- Código Aberto e Gratuito
- Customização e Flexibilidade
- Estabilidade e Segurança
- Desempenho e Eficiência

## DESVANTAGEM

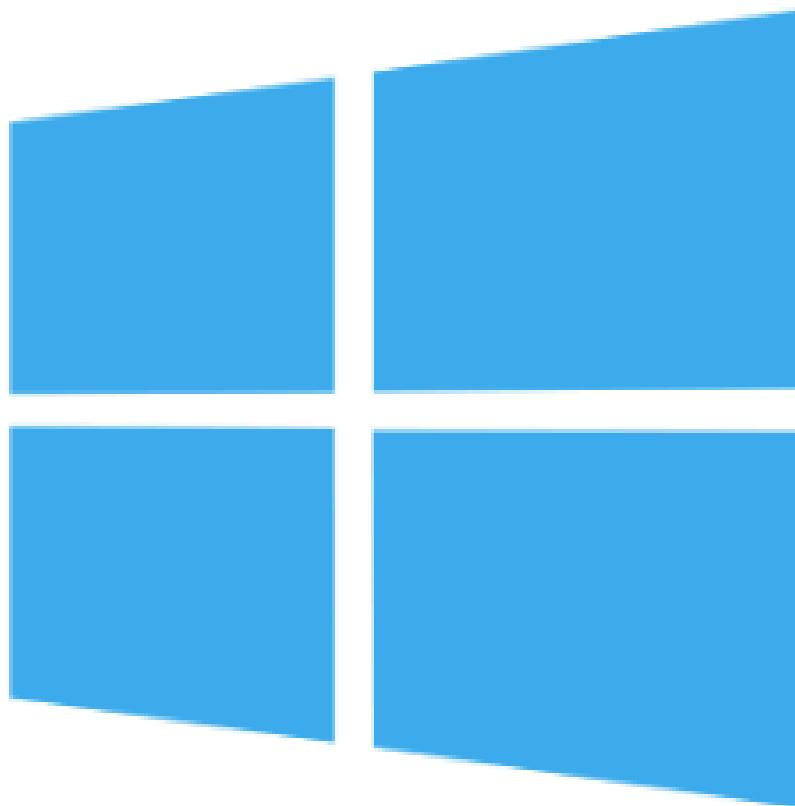
- Compatibilidade de Software
- Curva de Aprendizado
- Suporte ao Hardware

# Conclusão



O Linux oferece uma série de vantagens notáveis, incluindo sua natureza de código aberto, customização, estabilidade, segurança e desempenho eficiente. No entanto, a compatibilidade de software, a curva de aprendizado e a compatibilidade com hardware podem ser desafios para alguns usuários.

# Windows



O Windows é uma família de sistemas operacionais da Microsoft que é usada em desktops, servidores e dispositivos móveis. O Windows é conhecido por sua interface gráfica de usuário e suporte para uma ampla variedade de hardware e software.

# Windows

## VANTAGEM

- Visual Agradável
- Variedade de opções
- Reconhecimento de drivers
- Aplicativos

## DESVANTAGEM

- Preço
- Segurança
- Tamanho

# Conclusão



O sistema operacional Windows possui uma excelente interface gráfica, além de diversas opções aos seus usuários, complementados por uma grande facilidade de operação; entretanto ele possui desvantagens que afetam extremamente em seu funcionamento, além de um preço exorbitante.

# MacOS



O MacOS é um sistema operacional da Apple usado em computadores Mac. O MacOS é conhecido por sua interface de usuário elegante e intuitiva, e é popular entre profissionais de design e criativos.

# MacOS

## VANTAGEM

- Desempenho
- Segurança
- Interface Gráfica
- Aplicativos

## DESVANTAGEM

- Preço
- Compatibilidade
- Assistência técnica
- Gerenciador de arquivos

# Conclusão



Como todos puderam perceber, ter um Mac custa caro e as vezes pode não oferecer aquilo que o usuário realmente precisa. Por sua vez é um ótimo sistema para uso profissional, principalmente na área de design e fotografia.

# Android



O Android é um sistema operacional de código aberto baseado em Linux usado em dispositivos móveis, como smartphones e tablets. O Android é conhecido por sua ampla disponibilidade de aplicativos e personalização.

# Android

## VANTAGEM

- Código aberto
- Variedade de dispositivos
- Liberdade de customização
- Grande variedade de aplicativos
- Maior integração com os serviços do Google

## DESVANTAGEM

- Falta de padronização
- Falta de otimização
- Fragmentação
- Aplicativos desnecessários
- Desvalorização dos dispositivos

# Conclusão



O Android é um sistema robusto, versátil e amado por milhões. Seus prós geralmente superam os contras para muitos usuários. Ao ponderar sobre a aquisição de um dispositivo Android, é essencial considerar o melhor custo-benefício, as características da câmera, a duração da bateria e outras características relevantes para uma decisão bem informada.

# iOS



O iOS é um sistema operacional da Apple usado em dispositivos móveis, como iPhones e iPads. O iOS é conhecido por sua interface de usuário elegante e intuitiva e pela ampla disponibilidade de aplicativos.

# IOS

## VANTAGEM

- Ótimo desempenho
- Integração
- Aparência
- Segurança

## DESVANTAGEM

- Inflexível
- Preço
- Obsolescência
- Dependência
- Limitação

# Conclusão



o iOS é um sistema com várias vantagens e desvantagens, porém, o usuário precisa avaliar bem as necessidades e motivos que levam a buscar um iOS, antes de realizar a comprar ou troca do seu aparelho.

# Chrome OS



O Chrome OS é um sistema operacional da Google usado em dispositivos como Chromebooks. O Chrome OS é baseado em Linux e é conhecido por sua simplicidade e foco em aplicativos baseados na web.

# CHROME OS

## VANTAGEM

- Fácil de usar
- Rápido e eficiente
- Seguro e protegido
- Atualizado automaticamente
- Aplicativos

## DESVANTAGEEM

- Compatibilidade de software limitada
- Utilização offline
- Hardware limitado
- Compatibilidade de ficheiros
- Dependência da Google

# Conclusão



Como podemos ver é um gadget limitado, porém muito bom naquilo que foi projetado para fazer.

Muitos usuários do Chromebook ainda mantem seu MacBook ou Notebook Windows, utilizando o gadget somente para mobilidade e atividades específicas.

# FreeBSD



FreeBSD

O FreeBSD é uma variação do Unix que é conhecida por sua estabilidade e segurança. O FreeBSD é frequentemente usado em servidores e é conhecido por sua facilidade de configuração e administração.

# FreeBSD

## VANTAGEM

- O FreeBSD é um sistema operacional extremamente limpo e previsível
- Pode funcionar realmente como uma boa alternativa às plataformas UNIX tradicionais
- É um núcleo monolítico e seu principal interesse é segurança
- É estável
- Boa documentação
- Licença

## DESVANTAGEM

- Problema de compatibilidade de hardware
- Além disso, há menos suporte ao desenvolvedor
- Um pouco complexo de entender
- Precisa de muita prática

# Conclusão



Certamente, podemos dizer que tem muitas vantagens. É possível tentar com certeza, tentar aprender a ter boa exposição ao maravilhoso sistema operacional.

# Solaris



O Solaris é uma variação do Unix que é conhecida por sua escalabilidade e desempenho. O Solaris é frequentemente usado em servidores e é popular em ambientes empresariais.

# Solaris

## VANTAGEM

- Escalabilidade
- Segurança
- Confiabilidade
- Virtualização e Contêineres

## DESVANTAGEM

- Custo
- Adoção Limitada
- Compatibilidade de Aplicativos
- Curva de Aprendizado

# Conclusão



o Solaris oferece uma plataforma sólida e confiável, especialmente para ambientes corporativos que exigem alta escalabilidade e segurança. No entanto, os custos e a adoção limitada podem ser considerações importantes ao decidir entre o Solaris e outras opções de sistema operacional.

# IBM z/OS



IBM z/OS: É um sistema operacional da IBM que é usado em seus mainframes. O z/OS é conhecido por sua escalabilidade e confiabilidade e é popular em ambientes empresariais e governamentais que exigem alta disponibilidade e segurança.

# IBM z/OS

## VANTAGEM

- Confiabilidade e Disponibilidade Elevadas
- Segurança Avançada
- Eficiência de Recursos
- Escalabilidade Vertical

## DESVANTAGEM

- Custo
- Disponibilidade de Recursos e Suporte
- Compatibilidade de Aplicativos
- Curva de Aprendizado

# Conclusão



O IBM z/OS é uma escolha sólida para organizações que exigem alta disponibilidade, segurança e eficiência em ambientes de mainframe. No entanto, o alto custo, a curva de aprendizado e a complexidade podem ser considerações importantes ao avaliar sua adequação para uma determinada carga de trabalho ou ambiente operacional.

# **Exercícios**

**Qual é a diferença entre o Unix e o Linux?**

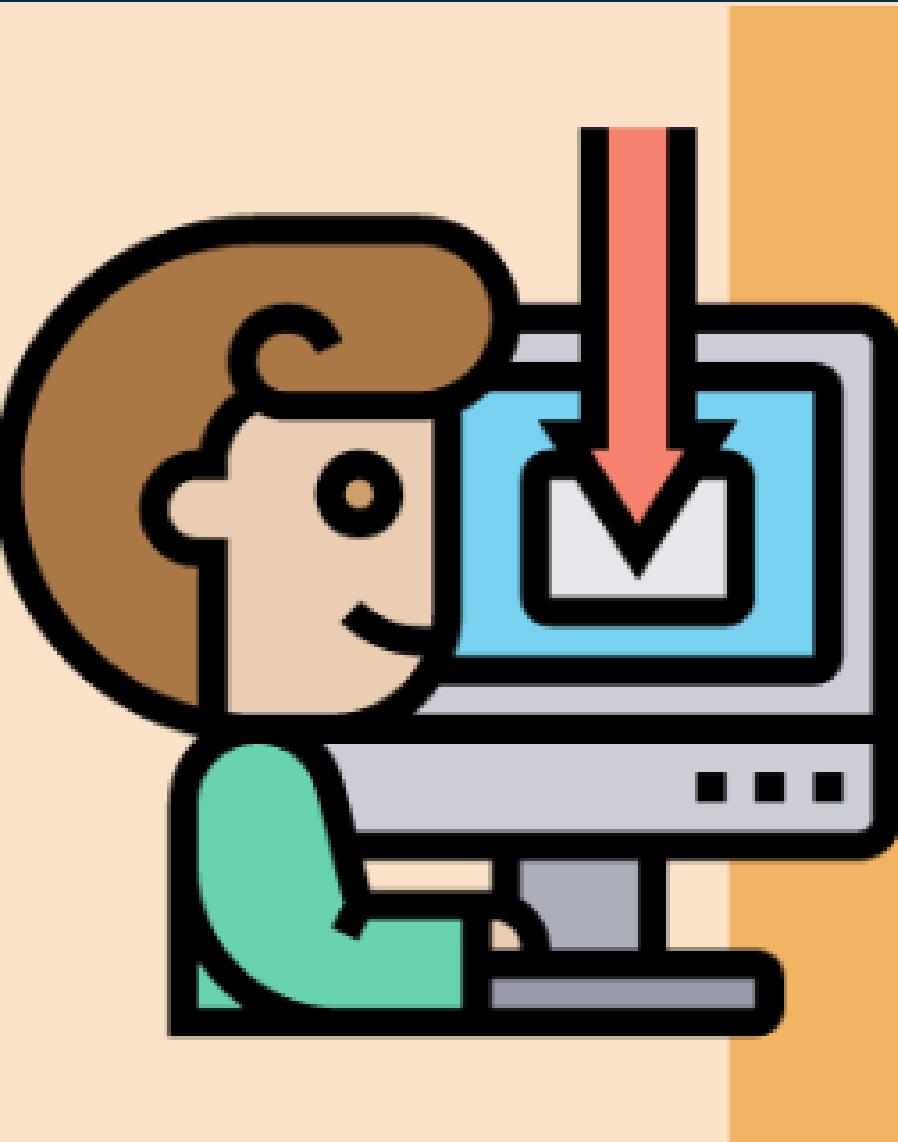
**Quais são as principais características do Windows?**

**Qual é a principal vantagem do Chrome OS?**

**LÓGICA DE  
PROGRAMAÇÃO  
ENTRADA E SAÍDA  
DE DADOS**



# ENTRADA DE DADOS



A entrada de dados é uma parte essencial da lógica de programação. Ela permite que o programa obtenha informações relevantes do usuário ou de outras fontes de dados.



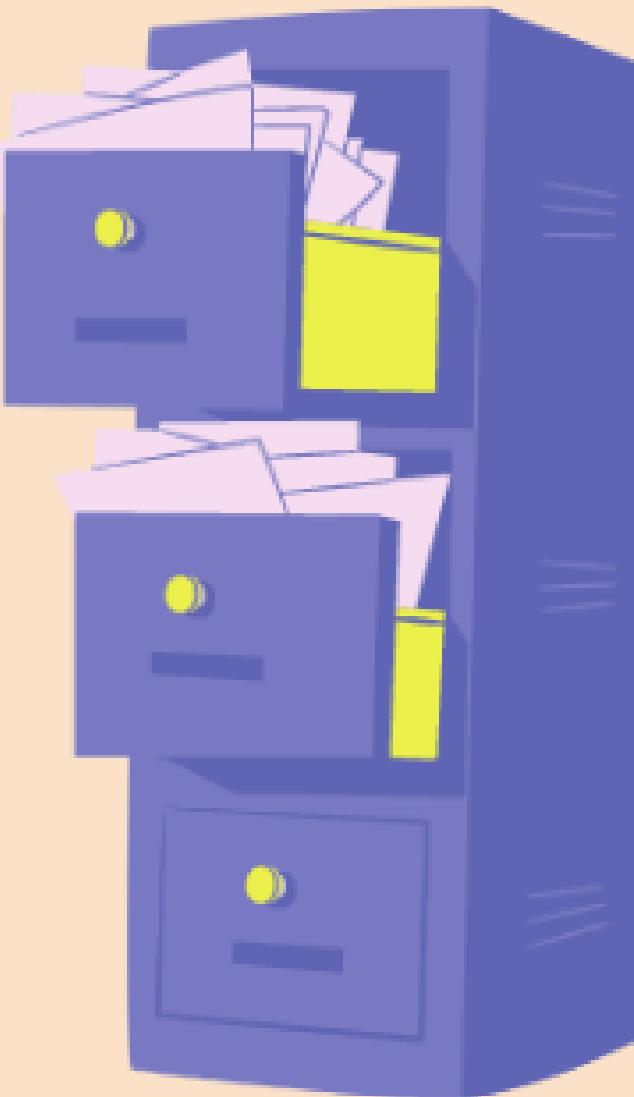
# GUI



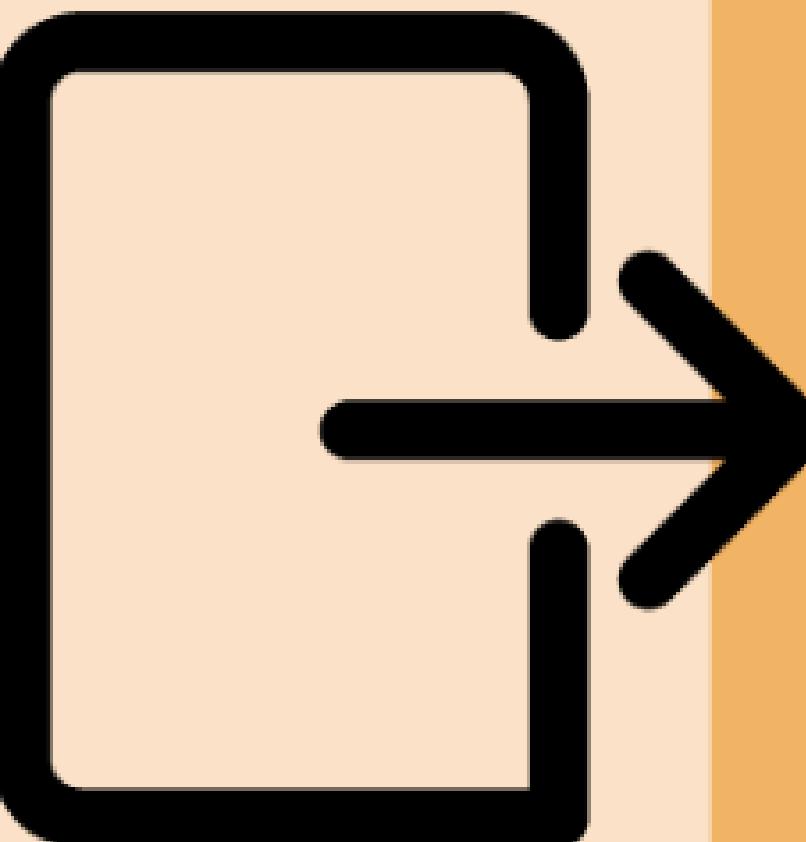
Existem várias formas de entrada de dados em um programa. A mais comum é por meio de uma interface gráfica do usuário (GUI), onde o usuário pode inserir informações em campos de entrada de texto, seletores de opções ou botões de rádio.



# ARQUIVOS



Outra forma de entrada de dados é por meio de arquivos, onde o programa lê informações de um arquivo de texto, planilha ou banco de dados. Isso é útil para processar grandes quantidades de dados ou para automatizar tarefas que envolvem o processamento de informações armazenadas.

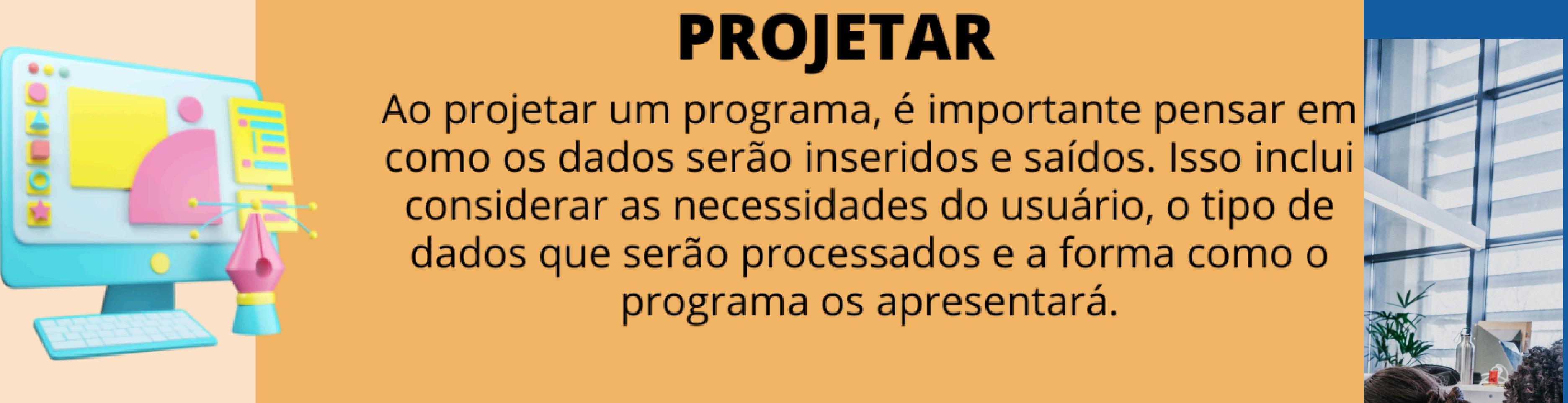


# **SAÍDA DE DADOS**

A saída de dados é a outra face da entrada de dados. Ela permite que o programa apresente resultados ou informações processadas de volta ao usuário ou a outras partes interessadas.

# PROJETAR

Ao projetar um programa, é importante pensar em como os dados serão inseridos e saídos. Isso inclui considerar as necessidades do usuário, o tipo de dados que serão processados e a forma como o programa os apresentará.



# VALIDAÇÃO



Um aspecto importante da entrada e saída de dados é a validação. Isso envolve verificar se as informações inseridas pelo usuário são válidas e se os resultados gerados pelo programa fazem sentido.





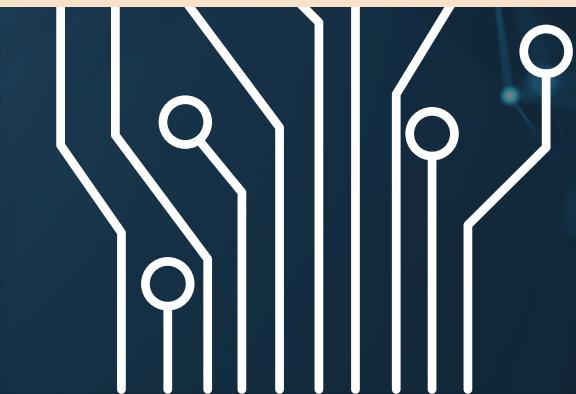
## VERIFICAÇÃO DA SINTAXE

A validação pode incluir a verificação da sintaxe (ou formatação) dos dados inseridos, bem como a verificação de regras de negócios ou restrições específicas. Por exemplo, um programa financeiro pode verificar se um valor inserido é um número válido e se ele não excede um limite específico.



# SEGURANÇA

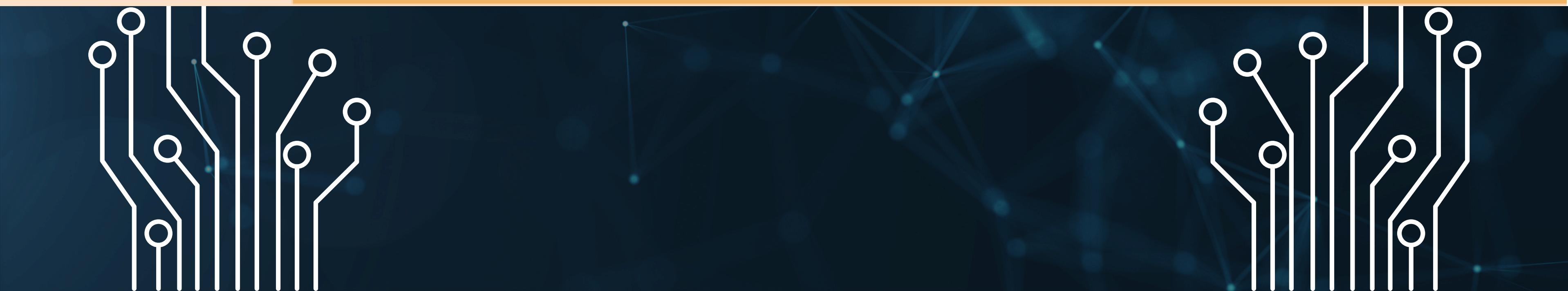
Além disso, a entrada e saída de dados devem ser seguras. Isso significa que o programa deve proteger informações confidenciais, como senhas, números de cartão de crédito e informações pessoais do usuário.



# EFICIÊNCIA NO RESULTADO

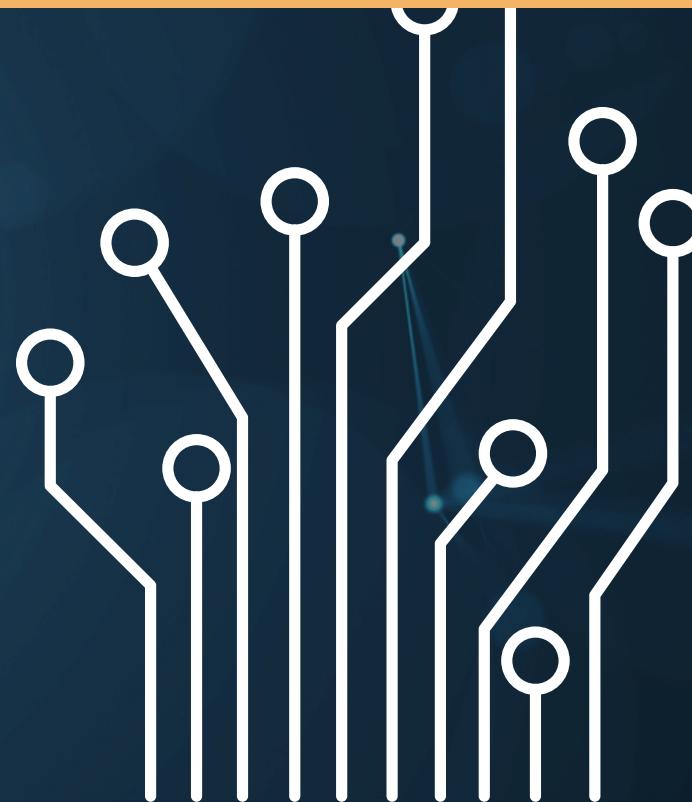


A entrada e saída de dados são fundamentais para a lógica de programação. Um bom programa deve ser capaz de obter informações relevantes do usuário ou de outras fontes de dados, processá-las de maneira eficiente e apresentar resultados claros e úteis de volta ao usuário.





"Programação é como escrever um livro; você começa com um rascunho e, em seguida, revisa e edita até que o produto final seja perfeito." - John Johnson



# Exercícios

1. O QUE É ENTRADA DE DADOS EM PROGRAMAÇÃO ?

2. O QUE É SAIDA DE DADOS EM PROGRAMAÇÃO?

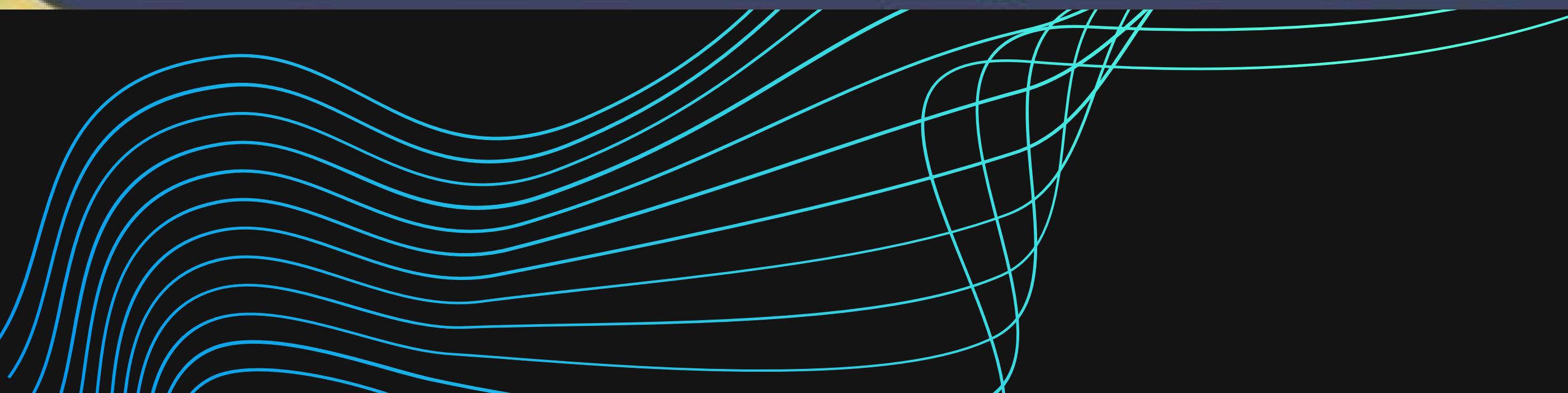
3. PORQUE É IMPORTANTE VALIDAR OS DADOS DE ENTRADA EM UM PROGRAMA?

4. COMO GARANTI QUE A ENTRADA DE DADOS EM UM PROGRAMA SEJA SEGURA?

# LÓGICA DE PROGRAMAÇÃO

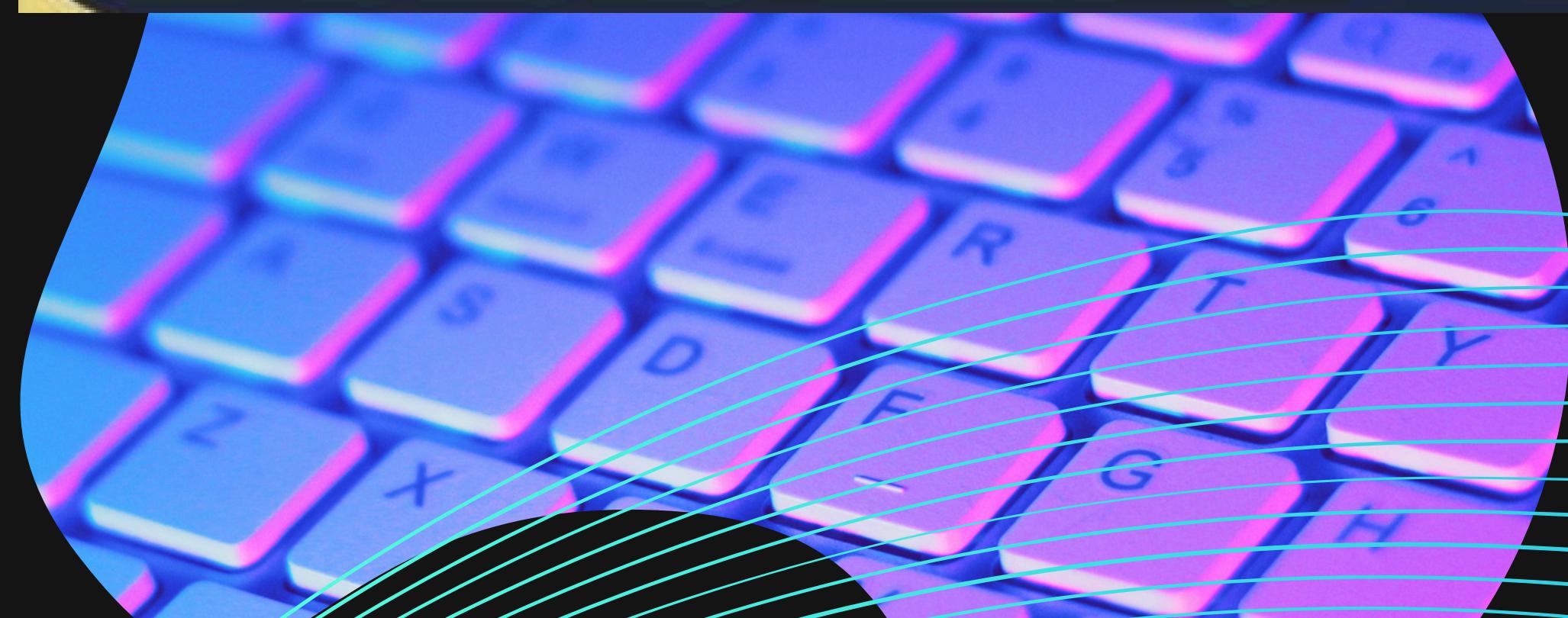
# CONSTANTES E VARIÁVEIS

Constantes e variáveis são elementos fundamentais em programação, pois permitem armazenar e manipular dados em memória. Neste contexto, as constantes representam valores fixos, que não podem ser alterados ao longo da execução do programa, enquanto as variáveis podem ter seus valores modificados durante a execução do programa.



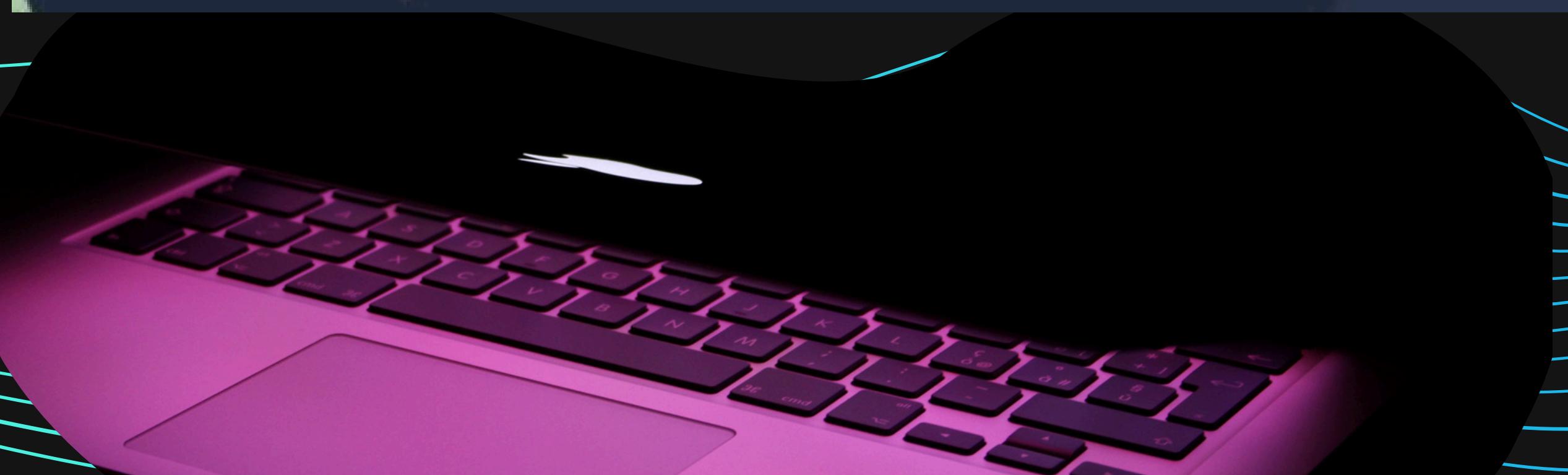
# PALAVRA-CHAVE

As constantes são definidas com a palavra-chave const ou final, dependendo da linguagem de programação utilizada. Já as variáveis são definidas com a palavra-chave var ou apenas com o nome da variável, dependendo da linguagem de programação. É importante destacar que o uso excessivo de constantes pode tornar o código pouco flexível, enquanto o uso excessivo de variáveis pode tornar o código confuso e difícil de entender.



# TIPAGEM

Outro ponto importante a ser considerado é a tipagem das constantes e variáveis. Algumas linguagens de programação possuem tipagem estática, ou seja, é necessário definir o tipo de dado que será armazenado na variável ou constante no momento de sua declaração. Já em outras linguagens, a tipagem é dinâmica, ou seja, o tipo de dado é definido automaticamente pelo valor atribuído à variável ou constante.





## VARIÁVEIS

As variáveis podem ser inicializadas no momento de sua declaração, ou posteriormente, durante a execução do programa. Já as constantes devem ser inicializadas obrigatoriamente no momento de sua declaração. O valor atribuído a uma constante não pode ser alterado ao longo da execução do programa.



# OPERADOR ARITMÉTICOS

As variáveis podem ser utilizadas em expressões matemáticas, como operadores aritméticos, relacionais e lógicos. As constantes, por sua vez, podem ser utilizadas como valores fixos em expressões matemáticas.

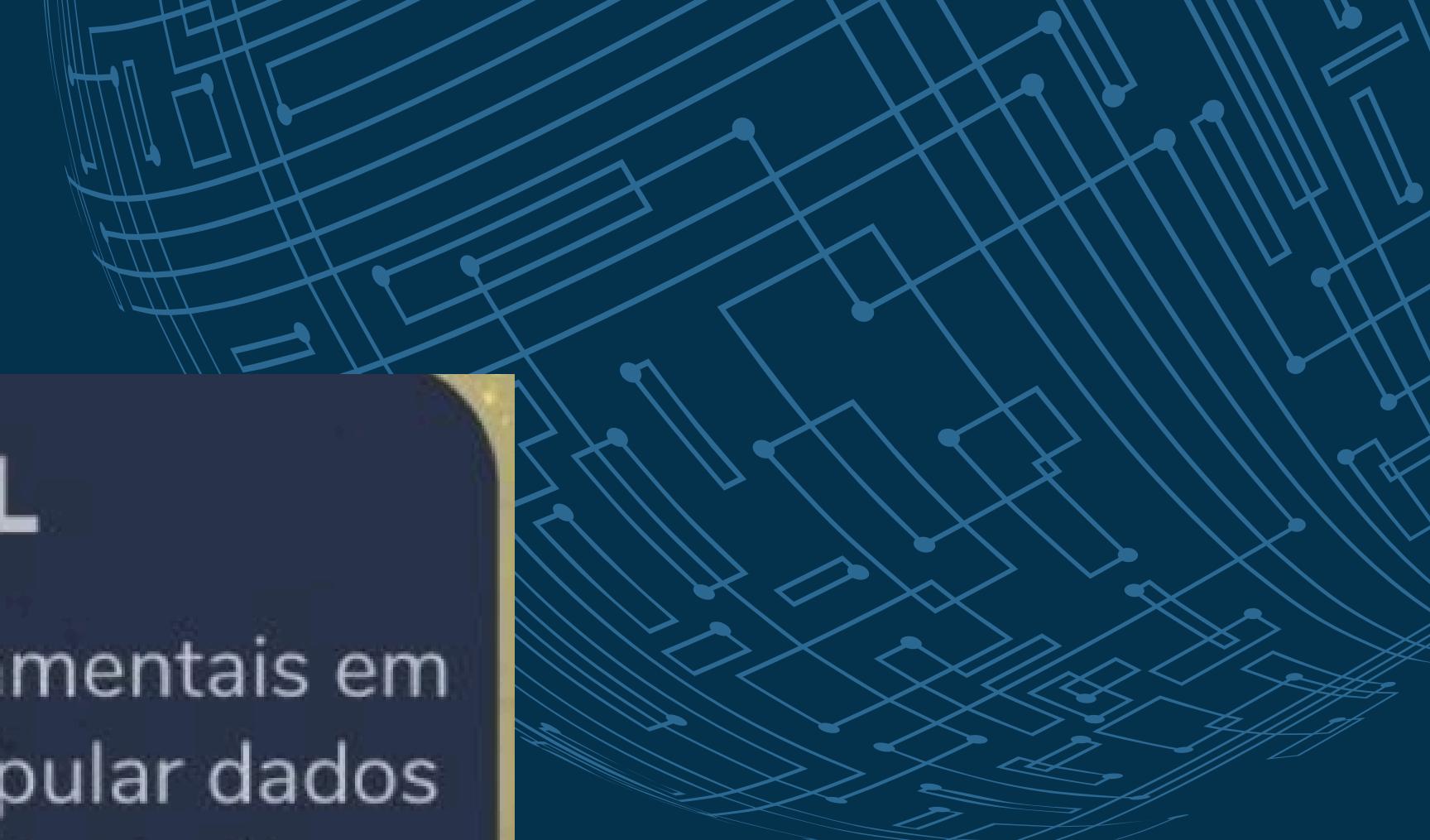


# ATRIBUIÇÃO

Uma variável pode ter seu valor modificado através de atribuição, enquanto uma constante não pode ter seu valor alterado após sua definição. Em algumas linguagens de programação, como o Python, as constantes são definidas utilizando a palavra-chave `const`, mas na prática, essa definição não é realmente aplicada, sendo comum a utilização de variáveis com nomes em maiúsculas para indicar que se trata de uma constante.

## CÓDIGO MAIS FLEXÍVEL

Constantes e variáveis são elementos fundamentais em programação, permitindo armazenar e manipular dados em memória. As constantes representam valores fixos, enquanto as variáveis podem ter seus valores modificados durante a execução do programa. É importante considerar a tipagem das constantes e variáveis, bem como suas formas de inicialização e uso em expressões matemáticas. O uso equilibrado de constantes e variáveis pode tornar o código mais flexível e fácil de entender.



python

```
import math

# Definindo a constante para representar o valor de PI
PI = math.pi

# Pedindo ao usuário para inserir o raio do círculo
raio = float(input("Digite o raio do círculo: "))

# Calculando a área do círculo usando a fórmula: área = PI * raio^2
area = PI * raio ** 2

# Exibindo a área calculada
print("A área do círculo é:", area)
```

Neste programa:

Importamos o módulo math para acessar o valor de PI.  
Definimos a constante PI com math.pi.  
Pedimos ao usuário para inserir o raio do círculo.  
Calculamos a área usando a fórmula matemática para a área de um círculo.  
Exibimos a área calculada.

```
# Definindo a constante para representar o fator de conversão de Celsius para Fahrenheit
FACTOR_DE_CONVERSAO = 9 / 5

# Pedindo ao usuário para inserir a temperatura em Celsius
temperatura_celsius = float(input("Digite a temperatura em Celsius: "))

# Convertendo a temperatura de Celsius para Fahrenheit usando a fórmula: Fahrenheit = Celsius * FACTOR_DE_CONVERSAO + 32
temperatura_fahrenheit = (temperatura_celsius * FACTOR_DE_CONVERSAO) + 32

# Exibindo a temperatura convertida em Fahrenheit
print("A temperatura em Fahrenheit é:", temperatura_fahrenheit)
```



Neste programa:

- Definimos a constante FACTOR\_DE\_CONVERSAO como o fator de conversão de Celsius para Fahrenheit (que é 9/5).
- Pedimos ao usuário para inserir a temperatura em Celsius.
- Calculamos a temperatura em Fahrenheit usando a fórmula de conversão.
- Exibimos a temperatura convertida em Fahrenheit.

## 1. print():

Este comando é usado para exibir uma mensagem ou o valor de uma variável na saída padrão.

Exemplo:

```
python
```

```
nome = "João"  
print("Olá, ", nome)
```

 Copy code



## 2. input():

Este comando permite que o usuário forneça entrada a partir do teclado durante a execução do programa.

Exemplo:

```
python
```

```
nome = input("Qual é o seu nome? ")  
print("Olá, ", nome)
```

 Copy code

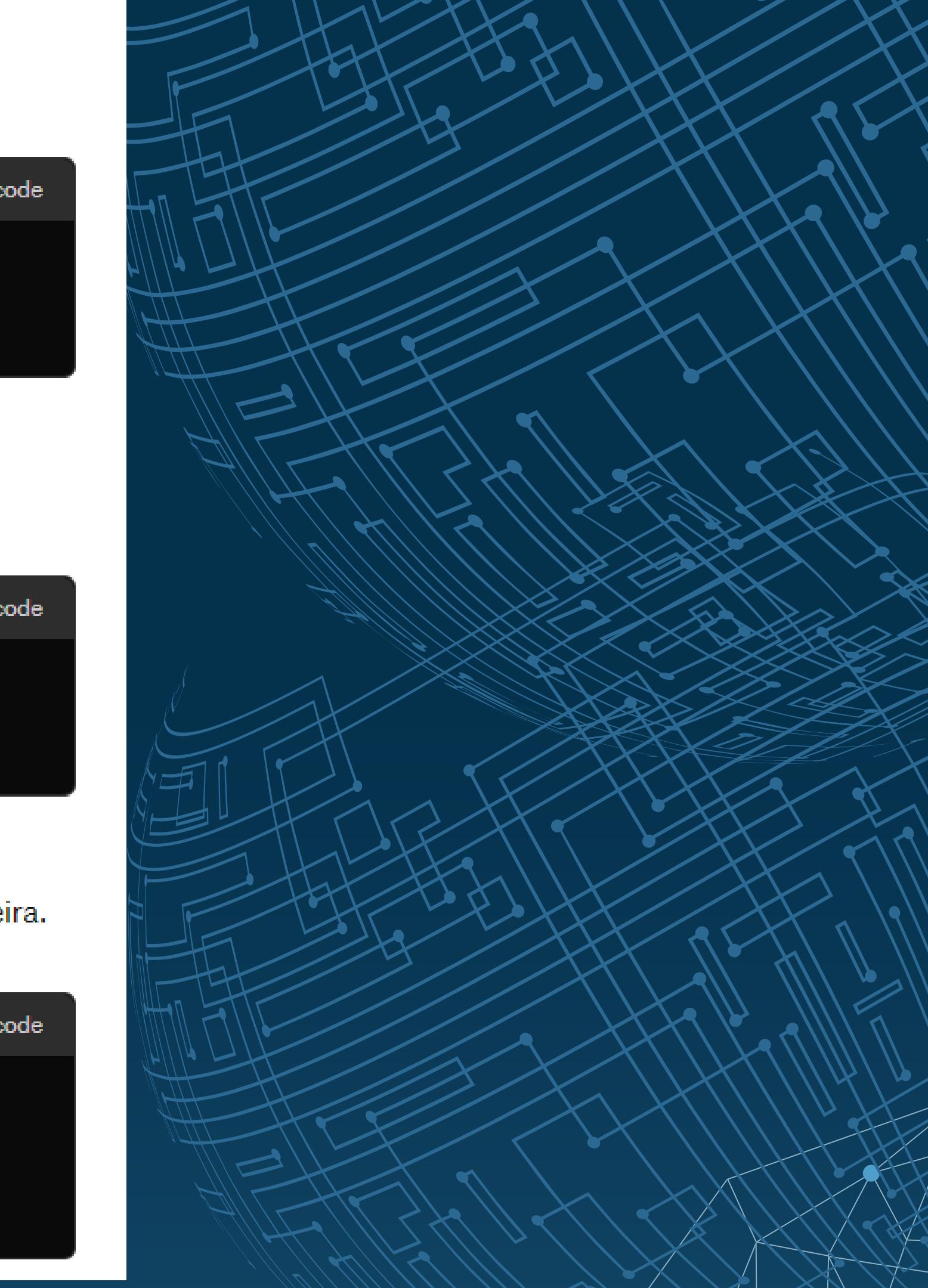


### 3. len():

Retorna o comprimento (número de itens) de um objeto.

Exemplo:

```
python
lista = [1, 2, 3, 4, 5]
print(len(lista)) # Saída: 5
```



### 4. range():

Cria uma sequência de números que são geralmente usados em loops.

Exemplo:

```
python
for i in range(5):
    print(i) # Saída: 0, 1, 2, 3, 4
```

### 5. if:

Este comando é usado para executar um bloco de código se uma condição específica for verdadeira.

Exemplo:

```
python
idade = 18
if idade >= 18:
    print("Você é maior de idade.")
```

## 6. for:

Este comando é usado para iterar sobre uma sequência e executar um bloco de código para cada item na sequência.

Exemplo:

```
python
```

 Copy code

```
lista = [1, 2, 3, 4, 5]
for item in lista:
    print(item) # Saída: 1, 2, 3, 4, 5
```



## 7. while:

Este comando é usado para executar repetidamente um bloco de código enquanto uma condição específica for verdadeira.

Exemplo:

```
python
```

 Copy code

```
contador = 0
while contador < 5:
    print(contador)
    contador += 1 # Incrementa o contador
```

## 8. def:

Define uma função no Python.

Exemplo:

```
python
```

```
def saudacao(nome):  
    print("Olá, ", nome)  
  
saudacao("Maria") # Saída: Olá, Maria
```

 Copy code

## 9. return:

Este comando é usado dentro de uma função para retornar um valor ao chamador da função.

Exemplo:

```
python
```

```
def soma(a, b):  
    return a + b  
  
resultado = soma(3, 4)  
print(resultado) # Saída: 7
```

 Copy code



## 10. import:

Este comando é usado para importar módulos ou pacotes em um programa Python.

Exemplo:

```
python  
  
import math  
print(math.sqrt(16)) # Saída: 4.0
```

 Copy code

## 1. Listas:

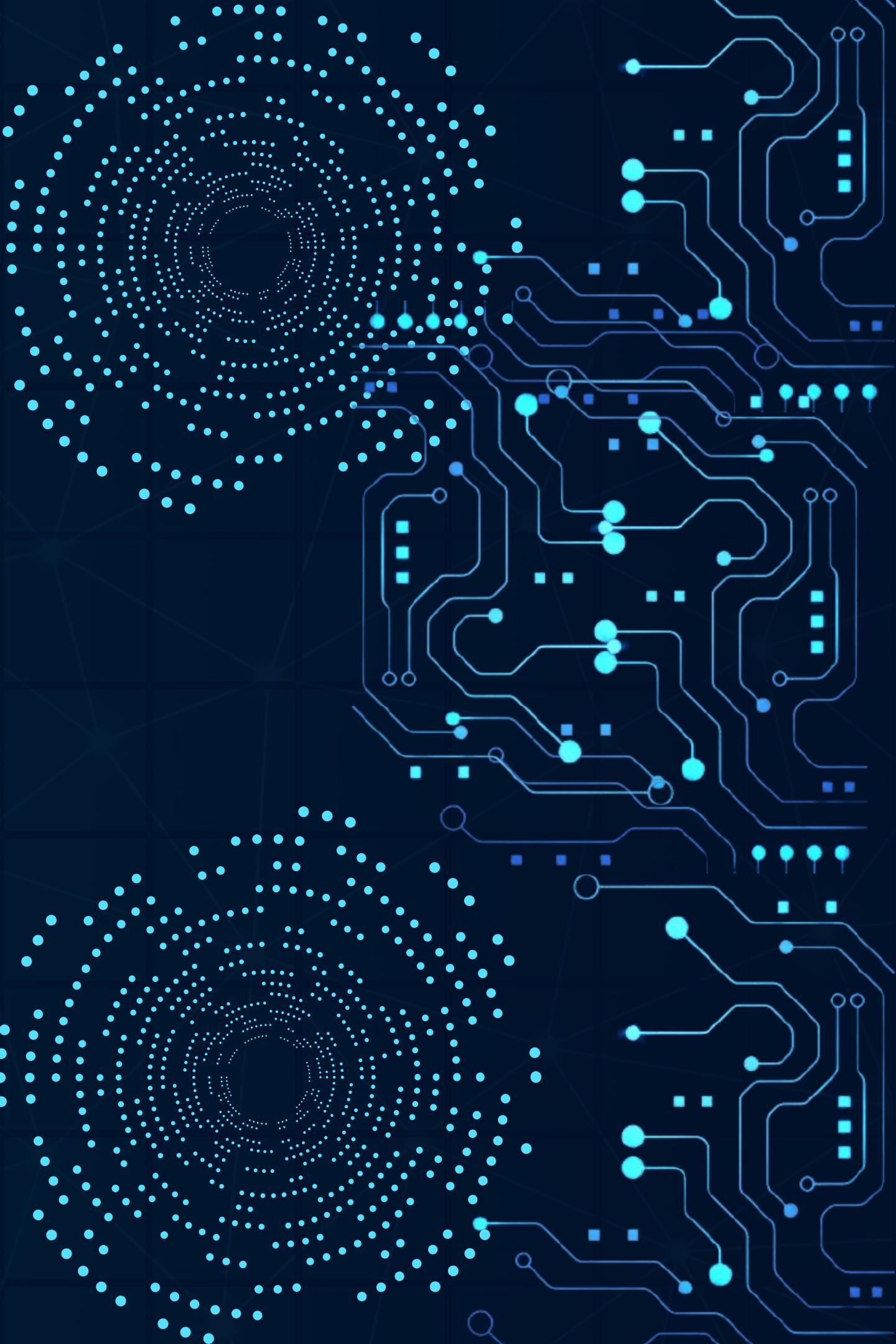
- Uma lista é uma coleção ordenada e mutável de itens.
- Os itens em uma lista são armazenados entre colchetes `[]` e separados por vírgulas.
- As listas podem conter itens de diferentes tipos de dados (inteiros, strings, booleanos, outras listas, etc.).
- Os elementos de uma lista podem ser alterados, adicionados ou removidos após a sua criação.

Exemplo:

python

 Copy code

```
lista = [1, 2, 3, 'quatro', True]
lista.append(5) # Adiciona um novo elemento
lista.remove('quatro') # Remove um elemento
print(lista) # Saída: [1, 2, 3, True, 5]
```



## 2. Tuplas:

- Uma tupla é uma coleção ordenada e imutável de itens.
- Os itens em uma tupla são armazenados entre parênteses `()` e separados por vírgulas.
- Assim como as listas, as tuplas podem conter itens de diferentes tipos de dados.
- A principal diferença em relação às listas é que os elementos de uma tupla não podem ser alterados, adicionados ou removidos após a sua criação.

Exemplo:

python

 Copy code

```
tupla = (1, 2, 3, 'quatro', True)
print(tupla[0]) # Acessando um elemento por índice - Saída: 1
```

Para criar uma lista em Python, podemos simplesmente definir os elementos da lista entre colchetes [ ], separados por vírgulas

```
# Criando uma lista de números inteiros
lista_numeros = [1, 2, 3, 4, 5]

# Criando uma lista de strings
lista_strings = ['maçã', 'banana', 'laranja', 'uva']

# Criando uma lista mista de diferentes tipos de dados
lista_mista = [10, 'vinte', True, 3.14]

# Criando uma lista vazia
lista_vazia = []

# Imprimindo as listas criadas
print(lista_numeros) # Saída: [1, 2, 3, 4, 5]
print(lista_strings) # Saída: ['maçã', 'banana', 'laranja', 'uva']
print(lista_mista) # Saída: [10, 'vinte', True, 3.14]
print(lista_vazia) # Saída: []
```

Você pode adicionar ou remover elementos de uma lista após sua criação usando métodos como `append()`, `insert()`, `remove()`, `pop()`, entre outros. As listas em Python são mutáveis, o que significa que seus elementos podem ser alterados.

Para criar uma tupla em Python, podemos definir os elementos da tupla entre parênteses (), separados por vírgulas. Aqui está um exemplo simples de como criar uma tupla.

```
# Criando uma tupla de números inteiros
tupla_numeros = (1, 2, 3, 4, 5)

# Criando uma tupla de strings
tupla_strings = ('maçã', 'banana', 'laranja', 'uva')

# Criando uma tupla mista de diferentes tipos de dados
tupla_mista = (10, 'vinte', True, 3.14)

# Criando uma tupla com um único elemento
tupla_simples = (42,) # A vírgula é necessária para distinguir uma tupla de um valor

# Imprimindo as tuplas criadas
print(tupla_numeros)    # Saída: (1, 2, 3, 4, 5)
print(tupla_strings)     # Saída: ('maçã', 'banana', 'laranja', 'uva')
print(tupla_mista)       # Saída: (10, 'vinte', True, 3.14)
print(tupla_simples)     # Saída: (42,)
```

As tuplas em Python são imutáveis, o que significa que, após criadas, seus elementos não podem ser alterados, adicionados ou removidos. Se você tentar modificar uma tupla após sua criação, o Python retornará um erro.

# RESUMINDO

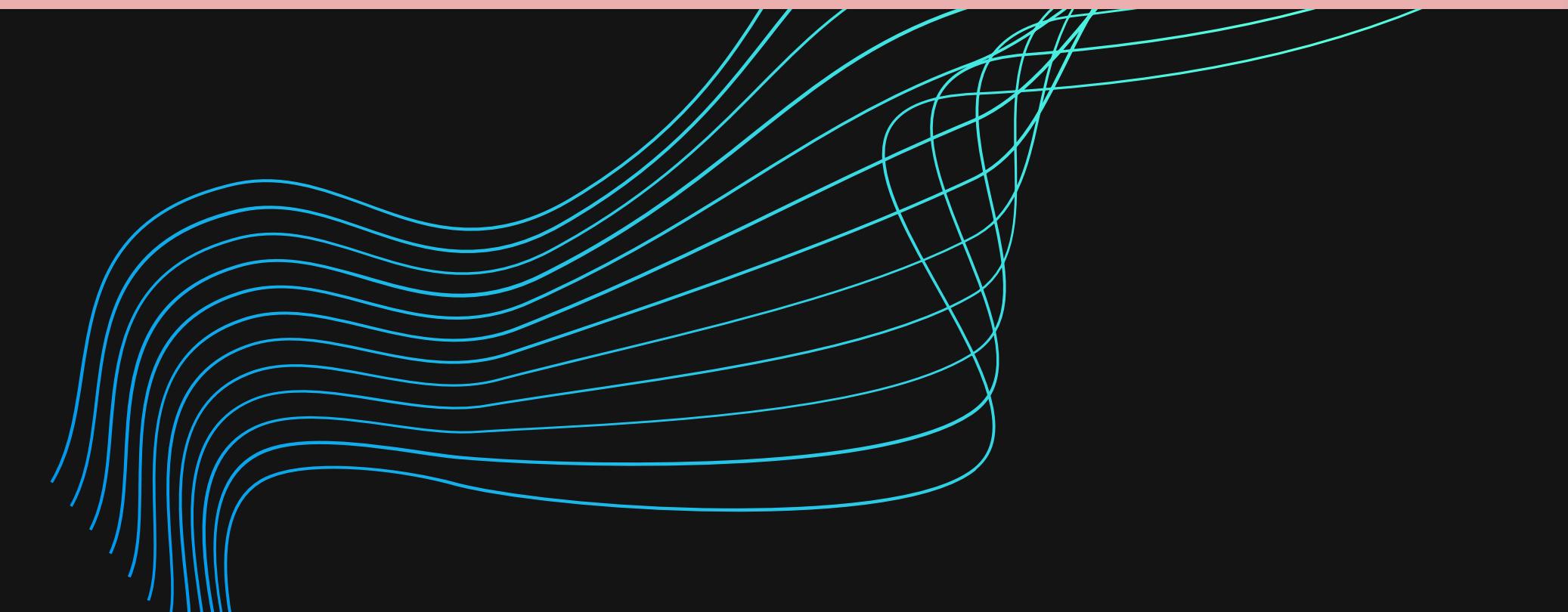
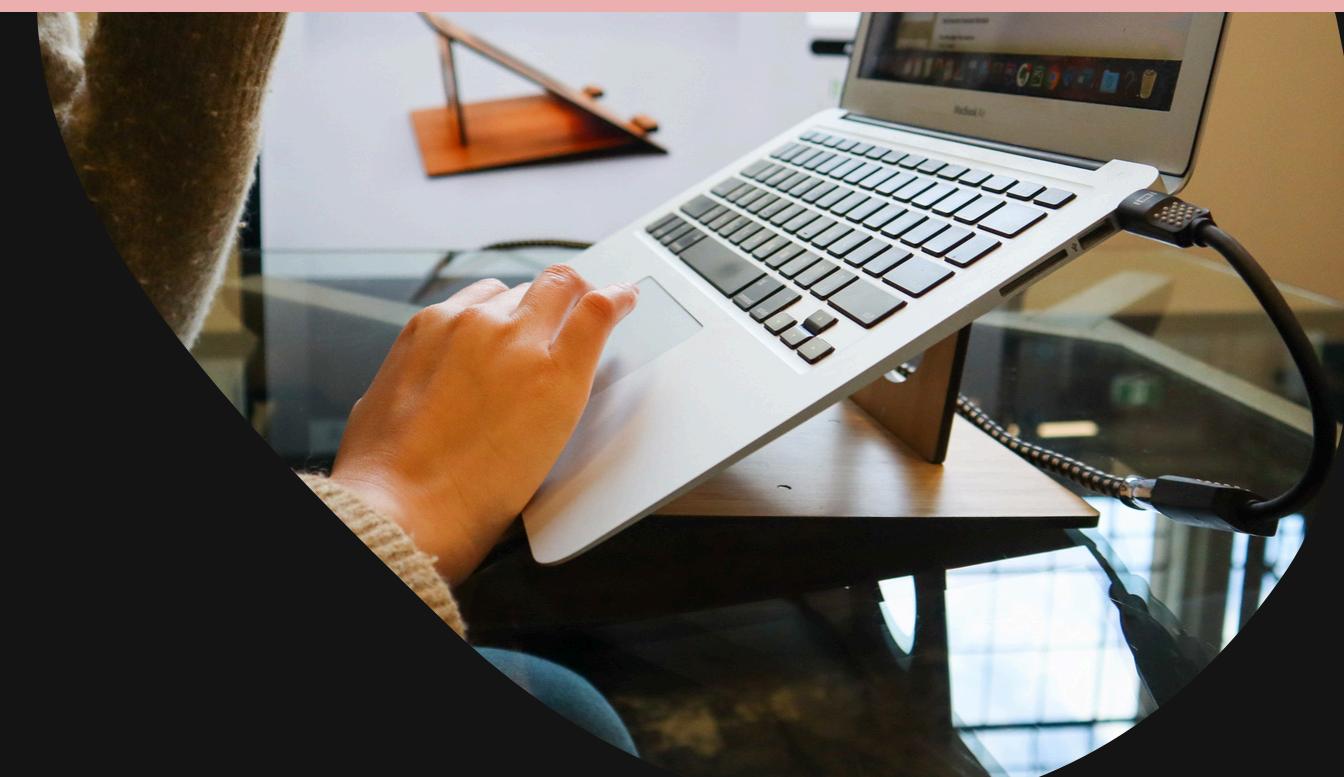
As listas são mutáveis e usam colchetes [ ], enquanto as tuplas são imutáveis e usam parênteses ( ). A escolha entre usar uma lista ou uma tupla depende da necessidade de mutabilidade dos elementos da coleção.

Se você precisar de uma coleção que não possa ser modificada, use uma tupla. Se precisar de uma coleção que possa ser alterada, use uma lista.

# **ESTRUTURAS DE CONTROLE**

# ESTRUTURAS DE CONTROLE

Estruturas de controle são estruturas fundamentais em qualquer linguagem de programação. Elas são responsáveis por controlar o fluxo do programa, ou seja, determinam como o código será executado. No Python, **existem três principais estruturas de controle: Laços de repetição, condicionais e estruturas de repetição.**



# LAÇOS DE REPETIÇÃO

Laços de repetição são usados para repetir um bloco de código até que uma condição seja atingida. O Python oferece três tipos de laços de repetição: for, while e do-while. O laço for é usado para percorrer sequências em Python, como listas, tuplas ou strings. O laço while é usado para executar um bloco de código enquanto uma condição se mantiver verdadeira. O laço do-while é semelhante ao laço while, mas a condição é verificada após a execução do bloco de código.



# LAÇOS DE REPETIÇÃO

Os laços de repetição, condicionais e estruturas de repetição são estruturas fundamentais em Python, que são usadas para controlar o fluxo do programa. Elas permitem que você crie programas mais complexos, pois permitem que você execute certos blocos de código com base em certas condições.



# CONDICIONAIS

Condicionais são usadas para executar um bloco de código se e somente se uma condição específica for atendida. O Python oferece três tipos de condicionais: **if**, **elif** e **else**. O condicional **if** executa um bloco de código se a condição especificada for verdadeira. O condicional **elif** executa um bloco de código se nenhuma das condições anteriores forem verdadeiras. O condicional **else** é executado se nenhuma das condições anteriores forem verdadeiras.

# ESTRUTURA DE REPETIÇÃO

A estrutura de repetição é usada para executar um bloco de código múltiplas vezes, dependendo do número de vezes especificado. O Python oferece a estrutura de repetição **for**, que é usada para executar um bloco de código um número específico de vezes. A estrutura de repetição **for** é muito útil quando é necessário executar um bloco de código várias vezes.

## BREAK

Além dos laços de repetição, condicionais e estruturas de repetição, o Python também oferece outra estrutura de controle muito importante: o comando **break**. O comando **break** é usado para interromper um laço de repetição quando uma determinada condição é atendida.

## CONTINUE

O Python também oferece outra estrutura de controle chamada **continue**. O comando **continue** é usado para interromper a iteração atual de um laço de repetição e iniciar a próxima iteração. Isso é útil quando você deseja ignorar certos blocos de código durante a execução de um laço de repetição.

# RETURN

O Python também oferece as estruturas de controle **pass** e **return**. O comando **pass** é usado para marcar o fim de um bloco de código, enquanto o comando **return** é usado para retornar um valor de uma função para o escopo em que foi chamada.

# FUNÇÕES

Além das estruturas de controle, o Python também oferece outras estruturas de programação úteis, como funções, classes, módulos e exceções. Uma função é um bloco de código que pode ser usado para executar tarefas específicas várias vezes. Uma classe é usada para criar objetos de aplicação específicos. Um módulo é um arquivo contendo códigos Python que podem ser usados em qualquer aplicação. E uma exceção é usada para tratar erros em tempo de execução.



# ESTRUTURAS DE CONTROLE PYTHON

As estruturas de controle Python são fundamentais para qualquer linguagem de programação. Elas permitem que você controle o fluxo de um programa, e também oferecem outras estruturas de programação úteis, como funções, classes, módulos e exceções. Se você aprender a usar essas estruturas de controle e outras estruturas de programação, você terá um amplo conhecimento da linguagem Python.



# EXERCÍCIOS

01

Faça um programa que leia um número inteiro e verifique se é positivo, negativo ou zero?

02

Desenvolva um programa que leia dois números inteiro e verifique qual deles é o maior?

03

Desenvolva um programa que calcule o preço total de uma compra em uma loja. Use variáveis para armazenar o preço de cada item e uma constante para representar a taxa de imposto?

