LICENCIATURA

EM COMPUTAÇÃO

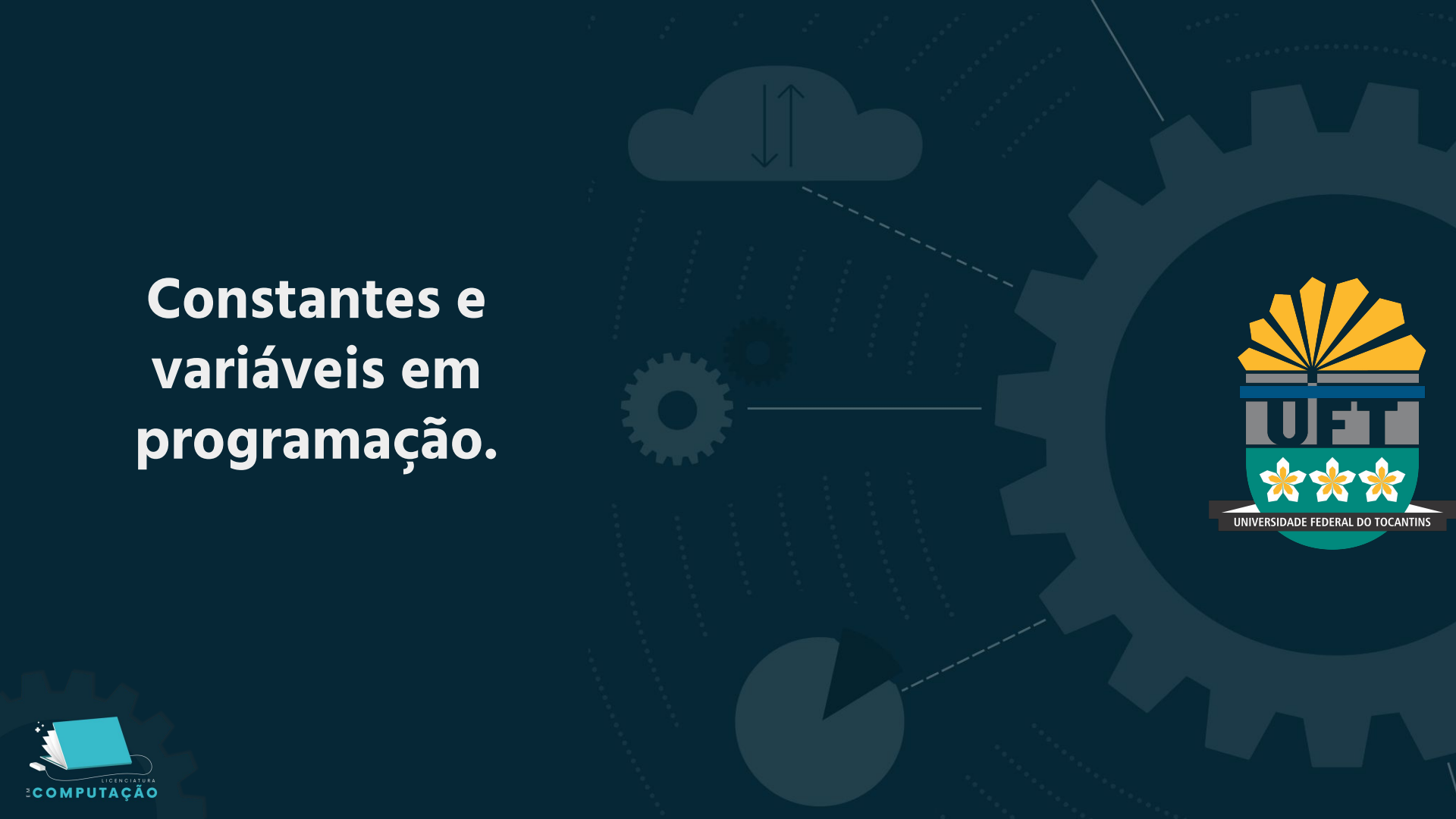
Programação de Computadores

Prof. Dr. Eduardo Ribeiro

UFT
UNIVERSIDADE FEDERAL DO TOCANTINS

LICENCIATURA EM COMPUTAÇÃO

Constantes e variáveis em programação.



UNIVERSIDADE FEDERAL DO TOCANTINS

LICENCIATURA EM COMPUTAÇÃO

O que veremos nessa aula?

- Conceitos fundamentais: Constantes e variáveis em programação.
- Declaração e atribuição de variáveis em Python, tipos de dados e operadores de atribuição.
- Uso de constantes para valores fixos e reutilização no código.
- Manipulação de variáveis: operações matemáticas, concatenação de strings e manipulação de texto.
- Conversão de tipos de dados para interoperabilidade.
- Boas práticas no uso de constantes e variáveis, incluindo nomeação adequada.
- Aplicação prática em Python através de exemplos, exercícios e desafios.

Variáveis em Python

- Declaração de variáveis em Python: Reservando espaço na memória para armazenar dados.
- Sintaxe básica para declarar variáveis: Escolhendo um nome significativo e atribuindo um valor usando o operador "=".
- Exemplo de declaração de variáveis com diferentes tipos de dados: Strings, números inteiros e números de ponto flutuante.

Variáveis em Python

- **Ao escolher nomes para variáveis, existem algumas regras a serem seguidas:**
  - Os nomes de variáveis podem conter letras (maiúsculas ou minúsculas), números e underscores (_).
  - O nome da variável deve começar com uma letra ou um underscore, mas não pode começar com um número.
  - Caracteres especiais, como espaços, pontos e símbolos matemáticos, não são permitidos em nomes de variáveis.
  - Python diferencia maiúsculas e minúsculas, portanto, "nome" e "Nome" seriam considerados variáveis diferentes.
  - Além das regras, existem também algumas convenções que são amplamente seguidas para nomear variáveis em Python:
  - Utilize nomes significativos e descritivos para as variáveis, que reflitam o propósito ou o conteúdo da informação armazenada.
  - Utilize letras minúsculas para nomes de variáveis.
  - Para nomes compostos, utilize o estilo snake_case, ou seja, palavras separadas por underscores (exemplo: meu_nome, idade_pessoa).
  - Evite nomes genéricos ou muito curtos, como "a", "x" ou "var".

Variáveis em Python

```python
nome_completo = "Maria Silva"

ano_nascimento = 1990

altura_cm = 165.5
```

```python
PI = 3.14159

raio = 5

area = PI * raio * raio

print("A área do círculo é:", area)
```

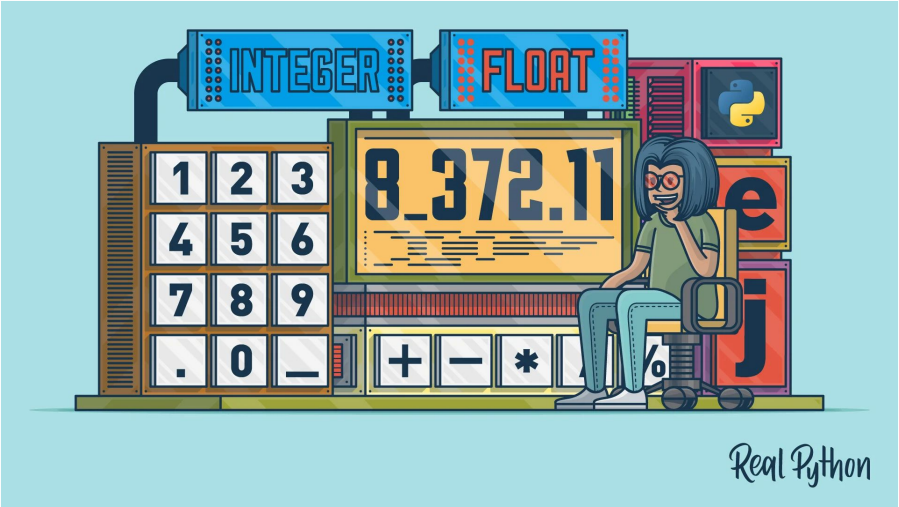
```python
nome = "João"

idade = 25

altura = 1.75
```

```python
nome = input("Digite seu nome: ")

mensagem = "Olá, " + nome + "! Bem-vindo ao nosso programa."

print(mensagem)
```

Tipos de Dados

- **Inteiros:**
  - Os inteiros representam números inteiros sem parte fracionária.
  - Eles podem ser positivos ou negativos.
  - Podemos atribuir um valor inteiro a uma variável da seguinte forma:

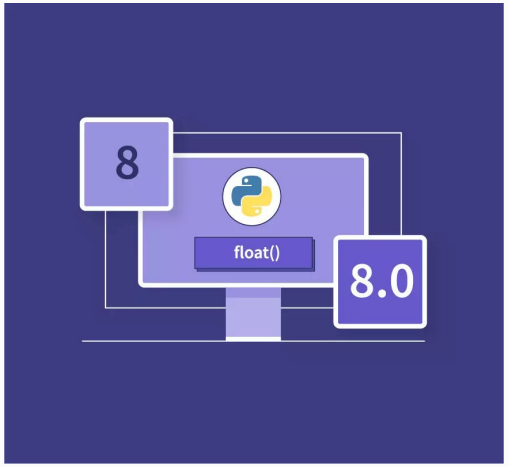
```
idade = 25

ano_nascimento = 1995
```



Tipos de Dados

- **Números de Ponto Flutuante:**
  - Os números de ponto flutuante representam números com parte fracionária.
  - Eles são usados para representar valores decimais.
  - Podemos atribuir um valor de ponto flutuante a uma variável da seguinte forma:

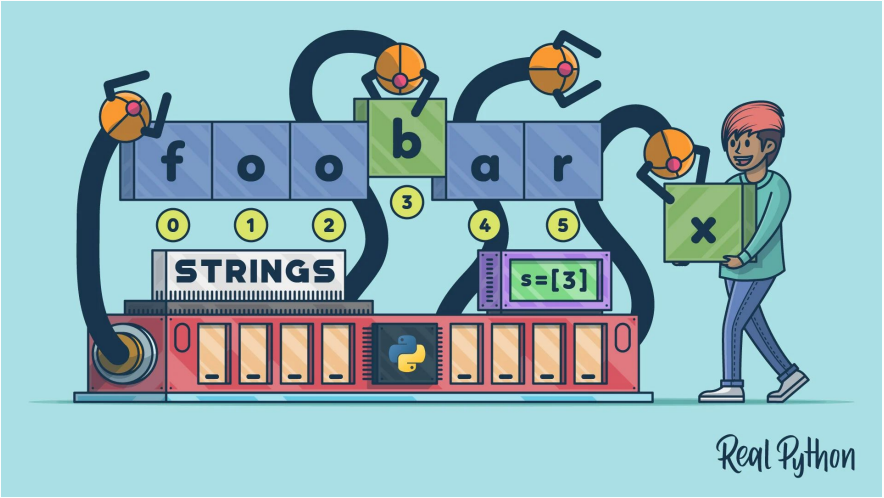
```
altura = 1.75

peso = 68.5
```



Tipos de Dados

- **Strings:**
  - As strings representam sequências de caracteres.
  - Elas são usadas para armazenar texto e são delimitadas por aspas simples (') ou aspas duplas ("").
  - Podemos atribuir um valor de string a uma variável da seguinte forma:

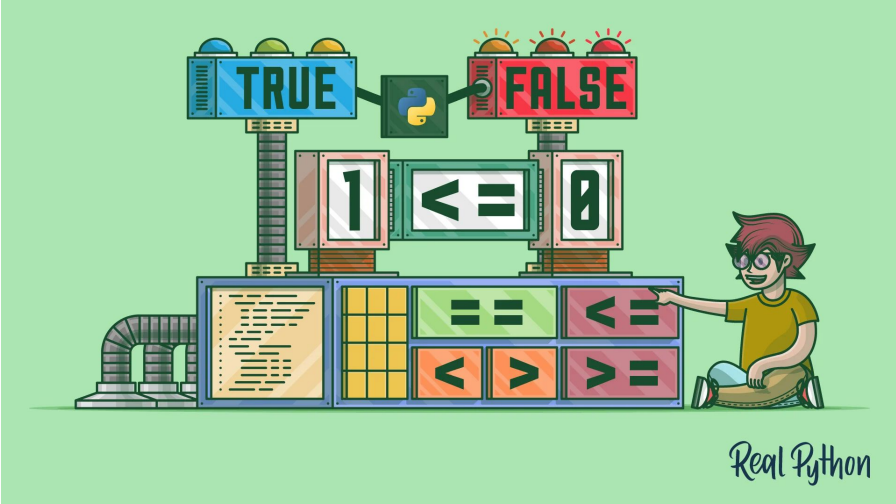
```
nome = "Maria"

frase = "Python é uma linguagem poderosa."
```



Tipos de Dados

- **Booleanos:**
  - Os booleanos representam os valores de verdadeiro (True) e falso (False).
  - Eles são usados para expressar condições lógicas.
  - Podemos atribuir um valor booleano a uma variável da seguinte forma:

```
temperatura_alta = TRUE

pessoa_autorizada = FALSE
```



Tipos de Dados

- Ao atribuir um valor a uma variável, Python infere automaticamente o tipo de dado com base no valor atribuído.
- Por exemplo, se atribuirmos um número inteiro a uma variável, ela será do tipo inteiro.
- Se atribuirmos um valor de string, a variável será do tipo string.
- No entanto, é importante lembrar que em Python as variáveis são dinamicamente tipadas, o que significa que o tipo de dado pode ser alterado posteriormente:

```
x = 10

x = "Olá"
```

Exemplos Práticos

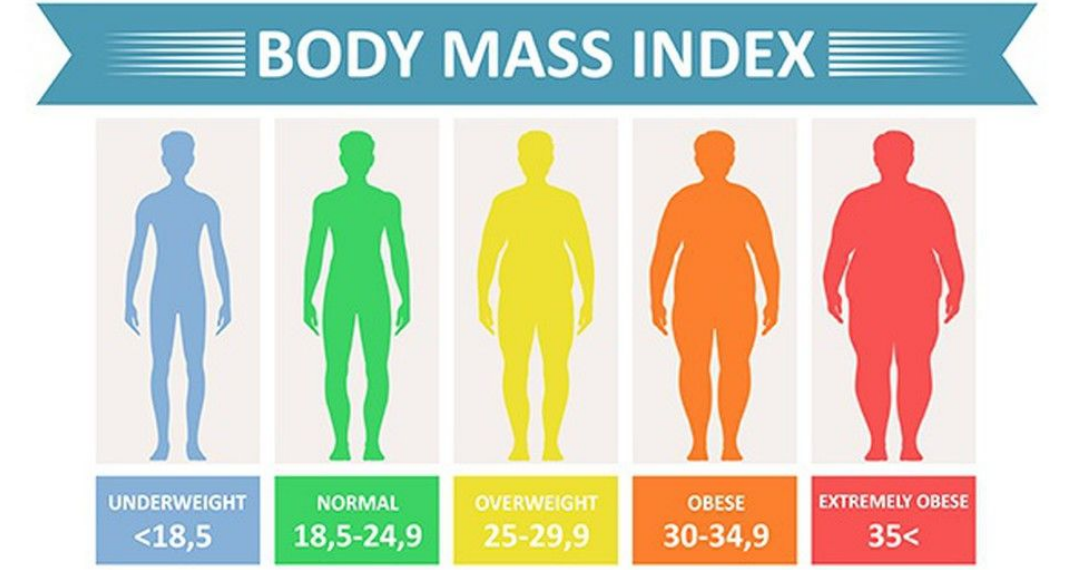
```
# Calculando o ano de nascimento
```

Exemplos Práticos

```
# Calculando o ano de nascimento

idade = 25

ano_nascimento = 1998

ano_atual = 2023

ano_nascimento = ano_atual - idade

print("Ano de nascimento:", ano_nascimento)
```

Exemplos Práticos

```
# Calculando o índice de massa corporal (IMC)
```

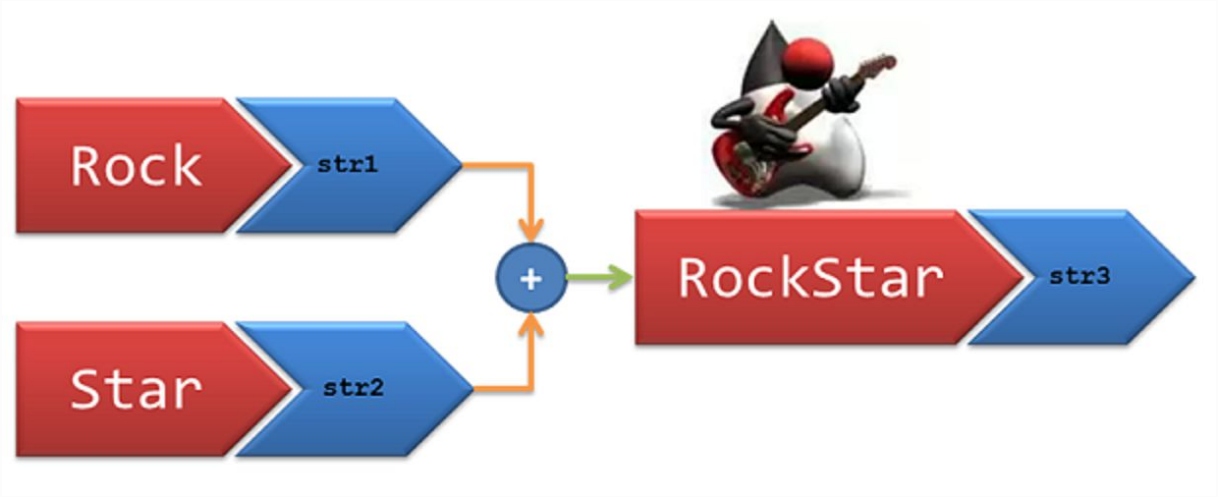


Exemplos Práticos

```
# Calculando o índice de massa corporal (IMC)


altura = 1.75

peso = 68.5


imc = peso / (altura ** 2)

print("IMC:", imc)
```

Exemplos Práticos

```
# Concatenando strings
```



Exemplos Práticos

```
# Concatenando strings


nome = "João"

sobrenome = "Silva"


nome_completo = nome + " " + sobrenome

print("Nome completo:", nome_completo)
```

Exemplos Práticos

```
# Verificando se é um bom dia para passear
```

Exemplos Práticos

```
# Verificando se é um bom dia para passear


temperatura = 25

chovendo = True


if temperatura > 20 and not chovendo:

    print("É um bom dia para passear!")

else:

    print("Melhor ficar em casa.")
```

Constantes

- **O papel importante das constantes na programação:**
  - Armazenar valores fixos que não mudam durante a execução do programa.
- **Vantagens das constantes:**
  - Melhor legibilidade, facilitação da manutenção e evitar repetição de valores no código.
- **Exemplos práticos de constantes em Python:**
  - Utilização de letras maiúsculas para nomear as constantes.

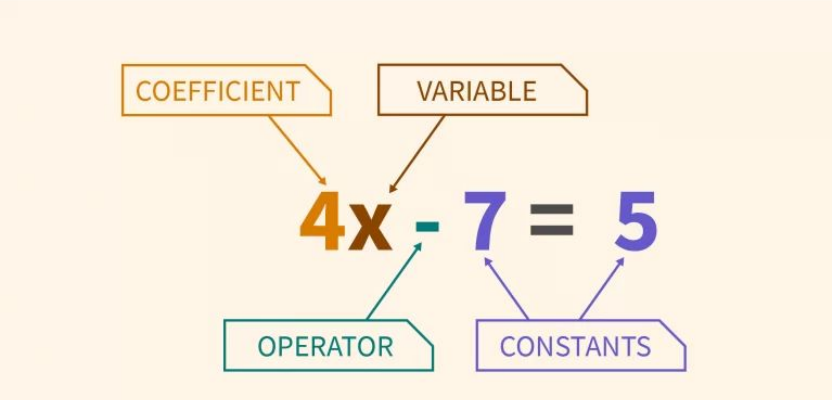
```
PI = 3.14159

TAXA_JUROS = 0.05

DIAS_SEMANA = 7
```

```
raio = 5

area = PI * raio ** 2

print("A área do círculo é:", area)
```

LICENCIATURA EM COMPUTAÇÃO

Constantes

- **Vantagens de utilizar constantes:**
  - Legibilidade e Manutenção: Nomes significativos facilitam a compreensão e a alteração de valores em um único local.
  - Reutilização de Valores: Evita repetição de valores, reduzindo erros de digitação e tornando o código mais eficiente.
  - Facilidade de Alterações: Modificar valores é rápido e fácil apenas na declaração da constante, sem buscar ocorrências no código.



Manipulação de Variáveis

Operações Matemáticas e Aritméticas:

```
# Operações Matemáticas

x = 5

y = 3

soma = x + y

subtracao = x - y

multiplicacao = x * y

divisao = x / y

resto = x % y

potencia = x ** y
```

```
print("Soma:", soma)

print("Subtração:", subtracao)

print("Multiplicação:",
multiplicacao)

print("Divisão:", divisao)

print("Resto:", resto)

print("Potência:", potencia)
```

Manipulação de Variáveis

Concatenação de Strings e Manipulação de Texto:

```
nome = "Maria"

sobrenome = "Silva"

idade = 30


nome_completo = nome + " " + sobrenome

mensagem = "Olá, " + nome_completo + "! Você tem " + str(idade) + " anos."


print(nome_completo)

print(mensagem)
```

Manipulação de Variáveis

Atualização de Variáveis com Base em Seus Valores Anteriores:

```
contador = 0
while contador < 5:
    print("Contagem:", contador)
    contador += 1
print("Fim do loop!")
```

Conversão de Tipos de Dados

Conversão de Inteiro para String:

```
idade = 25

idade_str = str(idade)


print("Idade:", idade_str)

print("Tipo de Dado:", type(idade_str))
```

Conversão de Tipos de Dados

Conversão de String para Inteiro ou Ponto Flutuante:

```
numero_str = "100"

numero_int = int(numero_str)

print("Número Inteiro:", numero_int)

print("Tipo de Dado:", type(numero_int))

numero_str = "3.14"

numero_float = float(numero_str)

print("Número de Ponto Flutuante:", numero_float)

print("Tipo de Dado:", type(numero_float))
```

Expressões Aritméticas

Precedência e Uso de Parênteses:

```
resultado = 2 + 3 * 4
print("Resultado sem parênteses:", resultado)  # Resultado: 14
resultado = (2 + 3) * 4
print("Resultado com parênteses:", resultado)  # Resultado: 20
```

Expressões Aritméticas

- Funções auxiliares com números de ponto flutuante em Python
- Módulo de matemática (math) e sua importação
- Exemplo de uso da função math.ceil(x) para encontrar o valor máximo inteiro de um número x

```
from math import ceil, sqrt
```

```
floor(x) #Retorna o maior inteiro menor ou igual a x.

ceil(x) #Retorna o menor inteiro maior ou igual a x.
```

```
sqrt(x) # Retorna a raiz quadrada de x.

log(x) # Retorna o logaritmo natural de x. Com dois argumentos, é possível especificar a base do logaritmo.

sin(x) #Retorna o seno de x em radianos.

asin(x) #Retorna o arcseno de x em radianos.
```

Expressões Lógicas

Operadores Lógicos:

```
a = 5

b = 3

c = 7

resultado = (a > b) and (b < c)

print("Resultado da expressão 'and':", resultado)  # Resultado: True
```

Expressões Lógicas

Operadores Lógicos:

```
a = 5

b = 3

c = 7

resultado = (a < b) or (b < c)

print("Resultado da expressão 'or':", resultado)  # Resultado: True
```

Expressões Lógicas

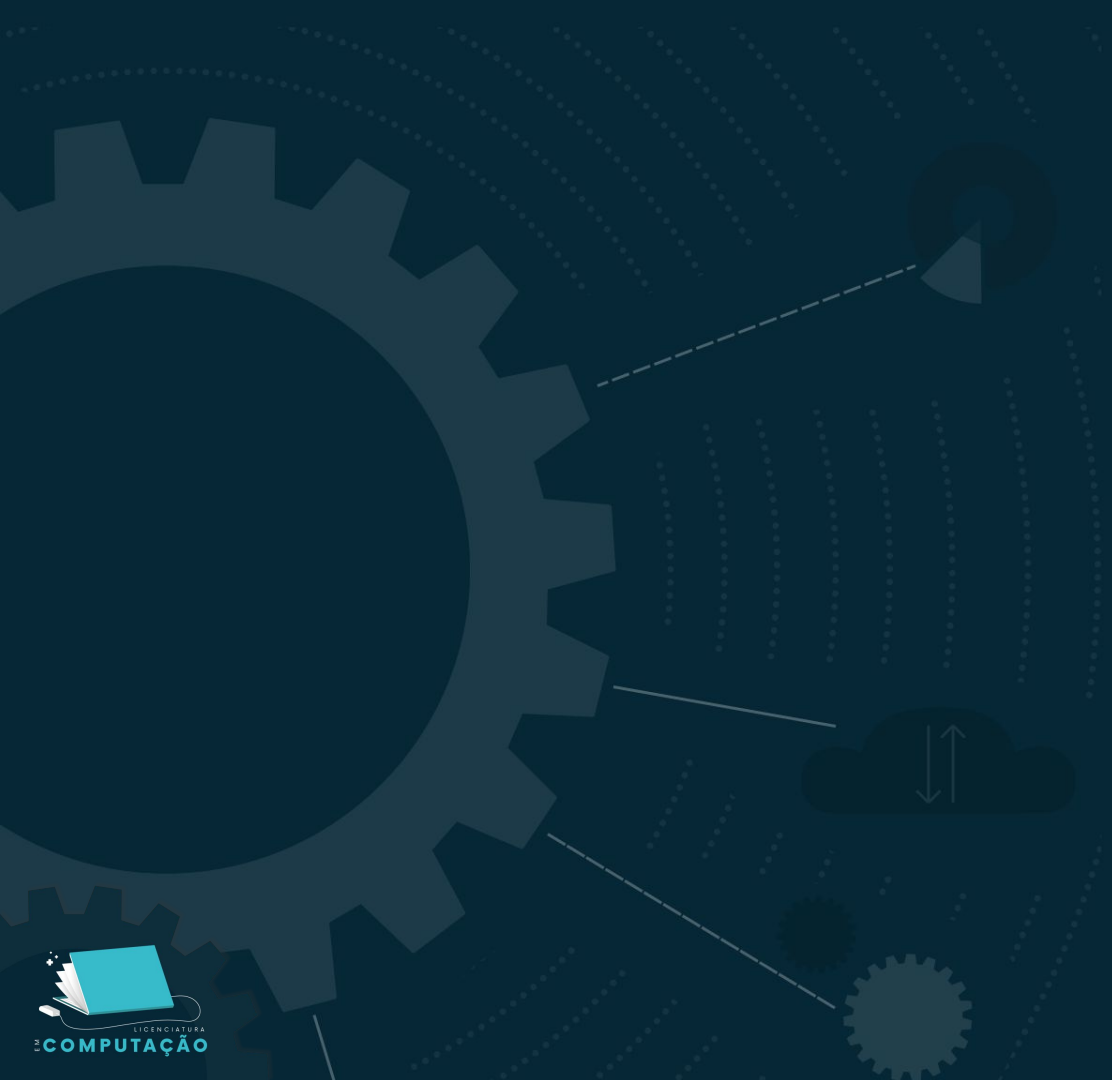
Operadores Lógicos:

```
a = 5

b = 3

resultado = not (a == b)

print("Resultado da expressão 'not':", resultado)  # Resultado:
```

Expressões Lógicas

Operadores Lógicos:

```
a = 5

b = 3

c = 7

resultado = ((a > b) and (b < c)) or (a == c)

print("Resultado da combinação de expressões lógicas:", resultado)  #
Resultado: True

resultado = not ((a > b) or (b < c))

print("Resultado da combinação de expressões lógicas:", resultado)  #
Resultado: False
```


LICENCIATURA
EM
COMPUTAÇÃO