


# Capítulo 2 - Constantes e Variáveis

*"Olhar para o código que você escreveu há mais de duas semanas é como olhar para o código que você está vendo pela primeira vez."*

*Dan Hurvitz*



Neste capítulo, vamos explorar um dos conceitos fundamentais da programação: constantes e variáveis. Ao entender esses conceitos, você estará um passo mais próximo de se tornar um programador competente e capaz de criar soluções eficientes.

As constantes desempenham um papel importante ao definir valores fixos que não podem ser alterados durante a execução do programa. Elas nos permitem armazenar informações essenciais, como valores matemáticos ou configurações estáticas. Ao utilizar constantes, tornamos nosso código mais legível e evitamos a repetição desnecessária de valores em todo o programa.

Por outro lado, as variáveis são espaços na memória que nos permitem armazenar valores que podem ser alterados ao longo do programa. Elas são flexíveis e nos dão a liberdade de armazenar e manipular informações conforme necessário. Ao declarar uma variável, podemos atribuir-lhe um valor inicial e atualizá-la sempre que necessário, permitindo-nos realizar cálculos, armazenar entradas do usuário e muito mais.


Neste capítulo, você aprenderá a declarar variáveis corretamente em Python, seguindo as regras e convenções adequadas. Também exploraremos os diferentes tipos de dados disponíveis, como inteiros, números de ponto flutuante, strings e booleanos. Através de exemplos práticos, você descobrirá como atribuir valores a variáveis e utilizar operadores de atribuição para atualizá-las.

Além disso, vamos mergulhar no uso de constantes, que nos permite armazenar valores fixos e reutilizá-los em todo o código. Você entenderá as vantagens de utilizar constantes em vez de valores fixos diretos no código, tornando-o mais legível, facilitando a manutenção e permitindo alterações rápidas quando necessário.

Manipular variáveis é uma habilidade essencial em programação, e neste capítulo, você aprenderá como realizar operações matemáticas e aritméticas utilizando variáveis. Além disso, exploraremos a concatenação de strings e a manipulação de texto, permitindo-nos criar mensagens dinâmicas e personalizadas.

Outro aspecto importante é a conversão de tipos de dados. Você aprenderá como converter um tipo de dado para outro, permitindo a interoperabilidade entre diferentes tipos e facilitando a manipulação de informações em diversos contextos.

Por fim, abordaremos boas práticas no uso de constantes e variáveis, incluindo recomendações sobre a nomeação adequada de variáveis para tornar o código mais legível. Também discutiremos o uso apropriado de variáveis globais e como evitar seu uso excessivo.



Através de exemplos práticos, exercícios e desafios, você terá a oportunidade de aplicar os conceitos de constantes e variáveis em Python. Essa prática constante permitirá que você aprimore suas habilidades e se familiarize com a declaração, atribuição e manipulação desses elementos fundamentais da programação.

As constantes e variáveis são elementos fundamentais na programação, desempenhando papéis importantes na manipulação de dados e no controle do fluxo do programa. Vamos entender melhor a importância desses conceitos e como eles são aplicados em Python.

As constantes são valores fixos que não podem ser alterados durante a execução do programa. Elas são utilizadas para armazenar informações que não devem ser modificadas, como valores matemáticos constantes, configurações padrão ou códigos de identificação. Ao utilizar constantes, tornamos o código mais legível e facilitamos a manutenção, pois podemos fazer alterações em um único local, caso seja necessário.

Um exemplo prático de constante em Python é a constante matemática Pi. Podemos definir uma constante chamada "PI" e atribuir-lhe o valor aproximado de 3.14159. Dessa forma, podemos utilizá-la em cálculos de geometria ou física, como o cálculo da área de um círculo.

Python

```
PI = 3.14159

raio = 5

area = PI * raio * raio

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

Já as variáveis são espaços na memória que podem armazenar valores que podem ser alterados ao longo do programa. Elas são flexíveis e nos permitem armazenar informações, como resultados de cálculos, entradas do usuário ou dados obtidos de outras fontes. As variáveis são fundamentais para a criação de programas dinâmicos e interativos.

Um exemplo prático de variável em Python é o armazenamento do nome de um usuário. Podemos solicitar ao usuário que digite seu nome e atribuir esse valor a uma variável chamada "nome". Em seguida, podemos exibir uma mensagem personalizada utilizando essa variável.

Python

```
nome = input("Digite seu nome: ")  
  
mensagem = "Olá, " + nome + "! Bem-vindo ao nosso programa."  
  
print(mensagem)
```

No exemplo acima, o valor digitado pelo usuário é armazenado na variável "nome" e concatenado com a string "Olá, " e "Bem-vindo ao nosso programa." para formar uma mensagem personalizada. Essa mensagem é exibida na tela através do comando print().

Perceba como as variáveis nos permitem armazenar informações e utilizá-las de forma flexível, tornando o programa mais interativo e adaptável às necessidades dos usuários.

Portanto, as constantes e variáveis desempenham papéis cruciais na programação. As constantes nos permitem armazenar valores fixos, facilitando a manutenção e a legibilidade do código. Já as variáveis nos fornecem a flexibilidade de armazenar e manipular dados ao longo do programa, tornando-o dinâmico e interativo.

## Seção 2.1 - Declaração de Variáveis

A declaração de variáveis é um passo fundamental na programação, pois permite que você reserve espaço na memória para armazenar dados. Em Python, a sintaxe para declarar variáveis é simples e direta. Vamos explorar a sintaxe, as regras e os nomes de variáveis válidos e convencionais.

A sintaxe básica para declarar uma variável em Python é bastante simples. Basta escolher um nome significativo para a variável e atribuir um valor a ela usando o operador de atribuição "=", conforme o exemplo abaixo:

Python

```
nome = "João"  
  
idade = 25
```

```
altura = 1.75
```

No exemplo acima, declaramos três variáveis: "nome", "idade" e "altura". A variável "nome" armazena uma string, enquanto "idade" e "altura" armazenam um número inteiro e um número de ponto flutuante, respectivamente.

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".

Seguindo essas regras e convenções, você tornará seu código mais legível e compreensível tanto para você quanto para outros programadores.

Agora, vamos a alguns exemplos práticos:

Python

```
nome_completo = "Maria Silva"

ano_nascimento = 1990
```

```
altura_cm = 165.5
```

No exemplo acima, utilizamos nomes significativos para as variáveis. "nome\_completo" indica que estamos armazenando o nome completo de uma pessoa, "ano\_nascimento" representa o ano em que a pessoa nasceu, e "altura\_cm" indica a altura em centímetros.

Ao seguir as regras e convenções de nomeação de variáveis, você estará criando um código mais legível, facilitando a compreensão do seu programa.

Lembre-se de que a escolha adequada de nomes para variáveis é uma prática importante na programação, pois contribui para a clareza, a manutenção e a colaboração em projetos.

## Seção 2.2 - Tipos de Dados

Na programação, lidamos com diferentes tipos de dados que representam os diversos tipos de informações que podemos manipular. Em Python, alguns dos tipos de dados mais comuns são inteiros, números de ponto flutuante, strings e booleanos. Vamos explorar esses tipos de dados e aprender como atribuí-los a variáveis.

### 1) 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:

Python

```
idade = 25
```

```
ano_nascimento = 1995
```

No exemplo acima, "idade" e "ano\_nascimento" são variáveis que armazenam valores inteiros. Podemos realizar operações matemáticas com essas variáveis, como soma, subtração, multiplicação e divisão.

### 2) 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:

Python

```
altura = 1.75
```

```
peso = 68.5
```

No exemplo acima, "altura" e "peso" são variáveis que armazenam valores de ponto flutuante. Podemos realizar operações matemáticas com essas variáveis da mesma forma que com os inteiros.

### 3) 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:

Python

```
nome = "Maria"
```

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

No exemplo acima, "nome" e "frase" são variáveis que armazenam valores de string. Podemos manipular strings realizando operações como concatenação, fatiamento e muito mais.

### 4) 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:

Python

```
temperatura_alta = TRUE  
pessoa_autorizada = FALSE
```

No exemplo acima, "temperatura\_alta" e "pessoa\_autorizada" são variáveis que armazenam valores booleanos. Eles são frequentemente utilizados em estruturas condicionais e loops para tomar decisões com base em condições lógicas.

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.

Por exemplo:

Python


```
x = 10  
x = "Olá"
```

No exemplo acima, inicialmente atribuímos o valor 10 à variável "x", tornando-a do tipo inteiro. Em seguida, atribuímos a string "Olá" à mesma variável, mudando o tipo para string.

Compreender os diferentes tipos de dados disponíveis em Python é essencial para manipular informações de forma adequada em nossos programas. Ao atribuir um valor a uma variável, é importante considerar o tipo de dado que estamos armazenando para garantir que as operações e manipulações subsequentes sejam realizadas corretamente.

Os tipos de dados primitivos são a base para representar informações em um programa. Eles nos permitem armazenar diferentes tipos de valores e realizar operações com eles. Os principais tipos de dados primitivos que estudaremos são os inteiros, números de ponto flutuante, strings e booleanos.





Os inteiros representam números inteiros, como 1, -5 e 100, que são utilizados em diversas situações, desde contagens até cálculos matemáticos complexos. Os números de ponto flutuante representam valores com casas decimais, como 3.14, -0.5 e 2.0, permitindo a realização de cálculos mais precisos. As strings são sequências de caracteres, como "Olá, mundo!", "Python" e "123", utilizadas para representar texto em um programa. Os booleanos representam os valores True (verdadeiro) e False (falso), que desempenham um papel fundamental nas expressões lógicas.

Vamos explorar mais alguns exemplos práticos que ilustram o uso dos principais tipos de dados primitivos em Python:

a) Inteiros:

Os inteiros são utilizados para representar números inteiros, como 10, -5 e 100. Podemos realizar operações matemáticas simples com eles. Vejamos um exemplo:

Python

```
idade = 25

ano_nascimento = 1998

ano_atual = 2023

# Calculando o ano de nascimento

ano_nascimento = ano_atual - idade

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

b) Números de Ponto Flutuante:

Os números de ponto flutuante representam valores com casas decimais. Eles são úteis para cálculos que envolvem valores fracionados. Vejamos um exemplo:

Python

```
altura = 1.75  
peso = 68.5  
  
# Calculando o índice de massa corporal (IMC)  
imc = peso / (altura ** 2)  
print("IMC:", imc)
```

#### c) Strings:

As strings são utilizadas para representar sequências de caracteres. Elas são usadas para trabalhar com texto em um programa. Vejamos um exemplo:

Python

```
nome = "João"  
sobrenome = "Silva"  
  
# Concatenando strings  
nome_completo = nome + " " + sobrenome  
print("Nome completo:", nome_completo)
```

#### d) Booleanos:

Os booleanos representam valores verdadeiros (True) e falsos (False). Eles são usados em expressões lógicas e estruturas de controle. Vejamos um exemplo:

Python

```
temperatura = 25

chovendo = True

# Verificando se é um bom dia para passear

if temperatura > 20 and not chovendo:

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

else:

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

Esses exemplos ilustram como podemos utilizar os tipos de dados primitivos em Python para realizar operações matemáticas, manipular texto e avaliar condições. É importante entender como cada tipo de dado funciona e quais operações são aplicáveis a eles.

Lembre-se de praticar e experimentar com diferentes valores e operações para fortalecer sua compreensão dos tipos de dados primitivos em Python. Com o tempo, você se sentirá cada vez mais confortável ao utilizá-los em seus programas.

## Seção 2.3 - Uso de Constantes

As constantes desempenham um papel importante na programação, pois nos permitem armazenar valores fixos que não serão alterados durante a execução do programa. Elas ajudam a tornar o código mais legível, facilitam a manutenção e oferecem várias vantagens em relação ao uso de valores fixos diretos no código. Vamos explorar esses conceitos e ver exemplos práticos de constantes em Python.

As constantes são valores que atribuímos a variáveis que não pretendemos alterar durante a execução do programa. Elas são utilizadas para armazenar informações como valores matemáticos constantes, configurações estáticas ou códigos de identificação. Ao utilizar constantes, evitamos a repetição de valores em todo o código e melhoramos a legibilidade e a manutenção do programa.

Em Python, é comum nomear as constantes em letras maiúsculas para distingui-las das variáveis. Vejamos alguns exemplos práticos de constantes em Python:

Python

```
PI = 3.14159  
  
TAXA_JUROS = 0.05  
  
DIAS_SEMANA = 7
```

No exemplo acima, "PI", "TAXA\_JUROS" e "DIAS\_SEMANA" são constantes que armazenam valores fixos. Essas constantes podem ser utilizadas em várias partes do programa sem a necessidade de repetir o valor em cada ocorrência. Isso torna o código mais legível, fácil de manter e permite alterações rápidas se necessário.

Vamos considerar um exemplo em que precisamos calcular a área de um círculo. Podemos usar a constante "PI" para realizar o cálculo sem a necessidade de digitar o valor de Pi em cada ocorrência:

Python

```
raio = 5  
  
area = PI * raio ** 2  
  
print("A área do círculo é:", area)
```

Ao utilizar a constante "PI" em vez do valor direto de Pi, o código se torna mais legível e qualquer alteração necessária no valor de Pi pode ser feita facilmente modificando apenas a declaração da constante.

Existem várias vantagens em utilizar constantes em vez de valores fixos diretos no código:

- ☐ **Legibilidade e Manutenção:** Ao utilizar constantes, o código se torna mais legível, pois os valores fixos são representados por nomes significativos. Se houver a necessidade de

modificar esses valores, a alteração pode ser feita em um único local, facilitando a manutenção do código.

- ☐ Reutilização de Valores: O uso de constantes permite que valores fixos sejam reutilizados em todo o código sem a necessidade de repetição. Isso reduz erros de digitação e torna o código mais eficiente.
- ☐ Facilidade de Alterações: Se houver a necessidade de modificar um valor fixo, como uma taxa de juros ou uma configuração padrão, essa alteração pode ser feita de forma rápida e fácil apenas na declaração da constante, sem a necessidade de procurar todas as ocorrências desse valor no código.

Ao utilizar constantes, tornamos o código mais robusto, legível e fácil de manter. Elas nos ajudam a evitar erros de digitação, facilitam a reutilização de valores e permitem alterações rápidas e eficientes.

## Seção 2.4 - Manipulação de Variáveis

A manipulação de variáveis é uma parte essencial da programação, permitindo que realizem operações matemáticas, concatenação de strings e atualização de valores com base em seus estados anteriores. Vamos explorar exemplos práticos dessas manipulações utilizando variáveis em Python.

### a) Operações Matemáticas e Aritméticas:

Em Python, podemos realizar uma ampla variedade de operações matemáticas e aritméticas utilizando variáveis. Vamos considerar alguns exemplos práticos:

Python

#### # 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)
```

No exemplo acima, utilizamos as variáveis "x" e "y" para realizar operações matemáticas básicas. Armazenamos os resultados em variáveis separadas, como "soma", "subtracao", "multiplicacao" e assim por diante. Ao exibir os resultados, utilizamos a função print() para mostrar os valores na tela.

#### b) Concatenação de Strings e Manipulação de Texto:

Em Python, podemos usar as variáveis para manipular texto, realizar concatenação de strings e personalizar mensagens. Vejamos um exemplo:

```
Python

nome = "Maria"
sobrenome = "Silva"
idade = 30

nome_completo = nome + " " + sobrenome
```

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

No exemplo acima, utilizamos as variáveis "nome", "sobrenome" e "idade" para criar uma mensagem personalizada. A variável "nome\_completo" concatena as strings "nome" e "sobrenome" com um espaço entre elas. A variável "mensagem" concatena várias strings, incluindo a conversão do valor inteiro "idade" para string utilizando a função str().

#### c) Atualização de Variáveis com Base em Seus Valores Anteriores:

Em Python, podemos atualizar os valores de variáveis com base em seus estados anteriores. Vamos considerar um exemplo de atualização de variável em um loop:

Python

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

No exemplo acima, utilizamos a variável "contador" para acompanhar o progresso de um loop. A cada iteração do loop, exibimos o valor atual da variável "contador" e, em seguida, atualizamos seu valor adicionando 1 (contador += 1). Isso faz com que o loop seja executado cinco vezes antes de ser interrompido.

A manipulação de variáveis em Python é uma poderosa ferramenta que nos permite realizar operações matemáticas, concatenar strings, personalizar mensagens e atualizar valores com

base em seus estados anteriores. Essas manipulações são fundamentais para criar programas dinâmicos e interativos.

## Seção 2.5 - Conversão de Tipos de Dados

Em programação, muitas vezes precisamos converter um tipo de dado para outro. Em Python, é possível realizar a conversão de tipos utilizando funções específicas. Vamos explorar exemplos práticos de conversão de tipos de dados em Python.

### a) Conversão de Inteiro para String:

Para converter um número inteiro para uma string em Python, podemos utilizar a função `str()`. Vejamos um exemplo:

```
Python

idade = 25

idade_str = str(idade)

print("Idade:", idade_str)

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

No exemplo acima, convertemos a variável "idade" de um número inteiro para uma string utilizando a função `str()`. Em seguida, exibimos o valor da variável "idade\_str" na tela, bem como seu tipo de dado, que será uma string.

### b) Conversão de String para Inteiro ou Ponto Flutuante:

Para converter uma string para um número inteiro ou ponto flutuante, podemos utilizar as funções `int()` e `float()`, respectivamente. Vejamos alguns exemplos:

```
Python

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))
```

No primeiro exemplo, convertamos a string "100" em um número inteiro utilizando a função `int()`. Em seguida, exibimos o valor da variável "numero\_int" na tela, bem como seu tipo de dado, que será um número inteiro.

No segundo exemplo, convertamos a string "3.14" em um número de ponto flutuante utilizando a função `float()`. Exibimos o valor da variável "numero\_float" na tela, bem como seu tipo de dado, que será um número de ponto flutuante.

É importante notar que, ao realizar conversões de tipos de dados, devemos garantir que os valores sejam compatíveis. Caso contrário, podemos obter erros durante a execução do programa.

Ao trabalhar com entradas do usuário ou com dados de arquivos, a conversão de tipos de dados é uma habilidade útil para garantir que as informações sejam manipuladas corretamente.

A conversão de tipos de dados em Python nos oferece flexibilidade para manipular informações em diferentes formatos. Compreender como realizar essas conversões é essencial para trabalhar com dados de maneira eficiente e precisa em nossos programas.

## Seção 2.6 - Expressões Aritméticas

As expressões aritméticas desempenham um papel fundamental na programação, permitindo que realizemos cálculos matemáticos dentro dos nossos programas. Vamos explorar alguns

exemplos práticos que ilustram a utilização dos operadores matemáticos e a importância da precedência e do uso de parênteses nas expressões aritméticas em Python.

a) Adição e Subtração:

Os operadores de adição (+) e subtração (-) são utilizados para somar e subtrair valores. Vejamos um exemplo:

Python

```
a = 5
b = 3

resultado = a + b
print("Resultado da adição:", resultado)

resultado = a - b
print("Resultado da subtração:", resultado)
```

b) Multiplicação e Divisão:

Os operadores de multiplicação (\*) e divisão (/) são utilizados para multiplicar e dividir valores. Vejamos um exemplo:

Python

```
a = 4
b = 2

resultado = a * b
print("Resultado da multiplicação:", resultado)

resultado = a / b
```

```
print("Resultado da divisão:", resultado)
```

#### c) Resto da Divisão:

O operador de resto da divisão (%) retorna o resto da divisão entre dois valores. Vejamos um exemplo:

Python

```
a = 10
b = 3

resto = a % b

print("Resto da divisão:", resto)
```

#### d) Exponenciação:

O operador de exponenciação (\*\*) é utilizado para elevar um valor a uma determinada potência. Vejamos um exemplo:

Python

```
a = 2
b = 3

resultado = a ** b

print("Resultado da exponenciação:", resultado)
```

#### e) Precedência e Uso de Parênteses:

A precedência dos operadores define a ordem em que as operações são realizadas em uma expressão. É possível utilizar parênteses para controlar a ordem das operações. Vejamos um exemplo:

Python

```
resultado = 2 + 3 * 4

print("Resultado sem parênteses:", resultado) # Resultado: 14

resultado = (2 + 3) * 4

print("Resultado com parênteses:", resultado) # Resultado: 20
```

Neste último exemplo, o uso dos parênteses altera a precedência da operação, fazendo com que a adição seja realizada antes da multiplicação.

Ao utilizar expressões aritméticas, é importante estar ciente da precedência dos operadores e usar parênteses quando necessário para garantir que as operações sejam executadas na ordem desejada.

O Python oferece um conjunto abrangente de funções auxiliares para cálculos com números de ponto flutuante, e elas estão agrupadas no módulo de matemática (`math`). Para utilizar esse módulo, é necessário importá-lo no início do programa utilizando a instrução `import math`.

Por exemplo, se desejamos encontrar o valor máximo inteiro para um número `x`, ou seja, o menor inteiro não menor que `x`, podemos chamar a função adequada do módulo de matemática: `math.ceil(x)`.

A sintaxe para chamar funções de módulos é sempre a mesma: `nome_do_módulo.nome_da_função(argumento_1, argumento_2, ...)`.

Outra forma de utilizar as funções dos módulos é importá-las individualmente, atribuindo-lhes um nome específico:

Python

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

Dessa forma, podemos utilizar diretamente as funções `ceil` e `sqrt` sem precisar referenciar o módulo.

Além disso, algumas funções que lidam com números, como `int()`, `round()` e `abs()` (valor absoluto), são integradas ao Python e não requerem a importação do módulo de matemática.

Todas as funções de qualquer módulo Python padrão estão documentadas no site oficial do Python. No caso do módulo de matemática, é possível encontrar a descrição completa de cada função. Algumas delas incluem:

Arredondamento:

Python

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

Raízes e logaritmos:

Python

```
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.
```

Trigonometria:

Python

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

Ao utilizar o módulo de matemática, você tem acesso a um conjunto de funções poderosas que facilitam cálculos matemáticos complexos. É importante consultar a documentação oficial do Python para obter informações detalhadas sobre cada função.

Explore essas funcionalidades e aproveite os recursos do módulo de matemática para aprimorar seus programas e lidar com tarefas matemáticas de forma eficiente.

O Python oferece um conjunto abrangente de funções auxiliares para cálculos com números de ponto flutuante, e elas estão agrupadas no módulo de matemática (`math`). Para utilizar esse módulo, é necessário importá-lo no início do programa utilizando a instrução `import math`.

Por exemplo, se desejamos encontrar o valor máximo inteiro para um número `x`, ou seja, o menor inteiro não menor que `x`, podemos chamar a função adequada do módulo de matemática: `math.ceil(x)`. A sintaxe para chamar funções de módulos é sempre a mesma: `nome_do_módulo.nome_da_função(argumento_1, argumento_2, ...)`.

Outra forma de utilizar as funções dos módulos é importá-las individualmente, atribuindo-lhes um nome específico:

Python

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

Dessa forma, podemos utilizar diretamente as funções `ceil` e `sqrt` sem precisar referenciar o módulo.

Além disso, algumas funções que lidam com números, como `int()`, `round()` e `abs()` (valor absoluto), são integradas ao Python e não requerem a importação do módulo de matemática.

Todas as funções de qualquer módulo Python padrão estão documentadas no site oficial do Python. No caso do módulo de matemática, é possível encontrar a descrição completa de cada função.

Ao utilizar o módulo de matemática, você tem acesso a um conjunto de funções poderosas que facilitam cálculos matemáticos complexos. É importante consultar a documentação oficial do Python para obter informações detalhadas sobre cada função.

Explore essas funcionalidades e aproveite os recursos do módulo de matemática para aprimorar seus programas e lidar com tarefas matemáticas de forma eficiente.

## Seção 2.7- Expressões Lógicas

As expressões lógicas desempenham um papel importante na programação, permitindo que avaliemos condições e tomemos decisões com base em seus resultados. Vamos explorar alguns exemplos práticos que ilustram a utilização dos operadores lógicos e a combinação de expressões lógicas em Python.

### a) Operadores Lógicos:

Operador "and" (e): retorna True se ambas as expressões forem verdadeiras, caso contrário, retorna False.

Python

```
a = 5
b = 3
c = 7

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

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

Operador "or" (ou): retorna True se pelo menos uma das expressões for verdadeira, caso contrário, retorna False.

Python

```
a = 5
```

```
b = 3
c = 7
resultado = (a < b) or (b < c)
print("Resultado da expressão 'or':", resultado) # Resultado:
True
```

Operador "not" (não): inverte o valor de uma expressão lógica, retornando True se a expressão for falsa e False se a expressão for verdadeira.

```
Python
a = 5
b = 3
resultado = not (a == b)
print("Resultado da expressão 'not':", resultado) # Resultado: True
```

#### b) Combinação de Expressões Lógicas:

É possível combinar expressões lógicas utilizando parênteses e operadores de comparação, como igual a (==), diferente de (!=), maior que (>), menor que (<), maior ou igual a (>=) e menor ou igual a (<=). Vejamos alguns exemplos:

```
Python
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
```


Nesses exemplos, estamos combinando expressões lógicas utilizando parênteses para controlar a ordem das operações. O resultado final será determinado pelas condições especificadas nas expressões.

As expressões lógicas são amplamente utilizadas para avaliar condições em programas e tomar decisões com base nesses resultados. Pratique a utilização dos operadores lógicos e a combinação de expressões lógicas em Python, aplicando-os em situações reais do seu programa. Quanto mais você se familiarizar com esses conceitos, mais eficiente será ao lidar com condições e tomadas de decisão em seus programas.

## EXERCÍCIOS DE FIXAÇÃO

Aqui está uma lista de seis exercícios para você praticar em casa utilizando o Google Colab:

1. Declaração de Variáveis:
  - a) Declare uma variável chamada "idade" e atribua a ela o valor 25.
  - b) Declare uma variável chamada "nome" e atribua a ela seu nome completo.
  - c) Declare uma variável chamada "nota" e atribua a ela o valor 8.5.
2. Atribuição de Valores:
  - a) Atribua o valor 10 à variável "x" e o valor 5 à variável "y".
  - b) Atribua a string "Olá, mundo!" à variável "mensagem".



c) Atribua o valor booleano True à variável "ativo".

3. Uso de Constantes:

a) Declare uma constante chamada "PI" e atribua a ela o valor de 3.14159.

b) Declare uma constante chamada "DIAS\_SEMANA" e atribua a ela o valor de 7.

4. Manipulação de Variáveis:

a) Declare uma variável chamada "quantidade" e atribua a ela o valor 10. Em seguida, incremente seu valor em 5 unidades.

b) Declare uma variável chamada "nome\_completo" e atribua a ela seu primeiro nome. Em seguida, concatene seu sobrenome à variável.

c) Declare uma variável chamada "saldo" e atribua a ela o valor 100. Em seguida, atualize o valor adicionando 50.

5. Conversão de Tipos de Dados:

a) Converta a variável "idade" (inteiro) para uma string e armazene o resultado na variável "idade\_str".

b) Converta a string "3.14" para um número de ponto flutuante e armazene o resultado na variável "numero\_float".

c) Converta a string "25" para um número inteiro e armazene o resultado na variável "idade".

6. Exercícios Combinados:

a) Escreva um programa que solicite ao usuário um número inteiro, multiplique esse número por 2 e exiba o resultado na tela. Utilize variáveis para armazenar os valores.

b) Cálculo do Volume de uma Esfera

Desenvolva um algoritmo em Python que solicite ao usuário o raio de uma esfera e calcule o seu volume. O volume de uma esfera é dado pela fórmula:  $V = (4/3) * \pi * r^3$ , onde V é o volume e r é o raio da esfera.

Utilize os conceitos estudados, como declaração de variáveis, atribuição de valores, uso do módulo de matemática (math) e exibição de resultados na tela.

Passos do algoritmo:



Importe o módulo de matemática utilizando a instrução `import math`.

Solicite ao usuário que digite o raio da esfera.

Armazene o valor digitado em uma variável chamada "raio".

Calcule o volume utilizando a fórmula:  $\text{volume} = (4/3) * \text{math.pi} * \text{raio}^3$ .

Exiba na tela uma mensagem informando o volume calculado.

Exemplo de saída esperada:

Digite o raio da esfera: 5

O volume da esfera é: 523.5987755982989

Neste exercício, você estará aplicando os conceitos de declaração de variáveis, atribuição de valores, cálculos matemáticos utilizando o módulo de matemática (`math`) e exibição de resultados na tela. Essa é uma aplicação prática dos conceitos estudados, pois o cálculo do volume de uma esfera é utilizado em diversas áreas, como física e engenharia.


Lembre-se de testar seu algoritmo com diferentes valores de raio para garantir que os cálculos estejam corretos.

d) Faça um programa para uma loja de tintas. O programa deverá pedir o tamanho em metros quadrados da área a ser pintada. Considere que a cobertura da tinta é de 1 litro para cada 3 metros quadrados e que a tinta é vendida em latas de 18 litros, que custam R\$ 80,00. Informe ao usuário a quantidades de latas de tinta a serem compradas e o preço total.

7. Faça um programa em Python que solicite ao usuário o seu nome e idade. Em seguida, exiba na tela uma mensagem contendo o nome e a idade informados pelo usuário.

8. Escreva um programa em Python que solicite ao usuário dois números inteiros e realize as seguintes operações:

- a) Calcule a soma dos dois números.
- b) Calcule a diferença entre o primeiro número e o segundo número.
- c) Calcule o produto dos dois números.
- d) Calcule a divisão do primeiro número pelo segundo número (certifique-se de tratar o caso de divisão por zero).



9. Escreva um programa em Python que solicite ao usuário dois números inteiros e realize as seguintes operações:

- e) Calcule a soma dos dois números.
- f) Calcule a diferença entre o primeiro número e o segundo número.
- g) Calcule o produto dos dois números.
- h) Calcule a divisão inteira do primeiro número pelo segundo número.
- i) Calcule o resto da divisão do primeiro número pelo segundo número.
- j) Calcule o resultado da potenciação do primeiro número elevado ao segundo número.

10. Escreva um programa em Python que solicite ao usuário o valor de um ângulo em graus e calcule o seno, cosseno e tangente desse ângulo.

11. Escreva um programa em Python que solicite ao usuário a quantidade de horas trabalhadas por mês e o valor da hora de trabalho, e calcule o salário mensal.

12. Escreva um programa em Python que solicite ao usuário se ele deseja ativar o modo noturno (sim ou não). O programa deve exibir a mensagem "Modo noturno ativado!" se o usuário responder "sim", e a mensagem "Modo noturno desativado!" se o usuário responder "não"