Módulo 2 - Fundamentos de programação

Olá, **futuro dev**!

Neste módulo, vamos mergulhar no fascinante mundo da lógica e ver como ela se aplica à programação. Isso mesmo! Estamos prestes a começar a escrever nossas primeiras linhas de código. Mas antes, vamos dar um passo atrás e explorar o surgimento da lógica como ciência e a álgebra booleana. Preparado? Vamos lá!

Lógica: Um pouco de filosofia

Provavelmente, você já ouviu falar de Aristóteles, certo? Ele foi um dos três grandes filósofos da Grécia Antiga, juntamente com Sócrates e Platão. Uma de suas principais contribuições para o mundo da filosofia e do pensamento ocidental foi a criação da lógica. Para Aristóteles, a lógica é uma ferramenta que nos permite identificar erros e estabelecer verdades.

Para entender melhor, vamos aprender sobre a noção de **silogismo**. O silogismo é um tipo de raciocínio em que uma conclusão é deduzida a partir de uma série de premissas ou suposições específicas.

Por exemplo:

- Todas as pessoas gregas são humanas.
- Todos os humanos são mortais.
- Sendo assim, todos os gregos são mortais.

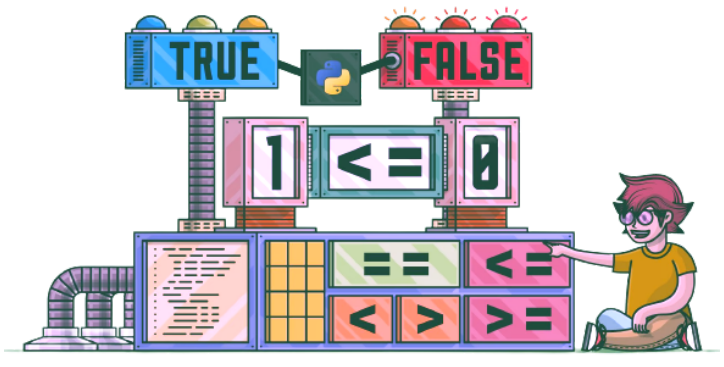
Para explicar melhor o que é um silogismo, o raciocínio pode ser sintetizado da seguinte maneira:

- Se todo X é Y e todo Y é Z, então, todo X é Z.

Agora, você deve estar se perguntando: "O que isso tem a ver com programação?". Bem, a lógica é a base de todos os programas de computador. Quando programamos, estamos essencialmente escrevendo instruções lógicas para o computador seguir. Cada linha de código é uma instrução que diz ao computador o que fazer, e essas instruções são baseadas em princípios lógicos.

Agora que temos uma compreensão básica da lógica, vamos introduzir dois conceitos fundamentais na programação: a álgebra booleana e a tabela verdade.

Álgebra Booleana: A Matemática da Lógica



A álgebra booleana é o sistema matemático que nos permite manipular e resolver expressões lógicas. Ela foi nomeada em homenagem a George Boole, um matemático britânico que a desenvolveu no século XIX.

Existem três operações básicas na álgebra booleana:

1. **AND (E)**: Esta operação retorna verdadeiro se ambas as suas entradas forem verdadeiras. Caso contrário, retorna falso.
2. **OR (OU)**: Esta operação retorna verdadeiro se pelo menos uma de suas entradas for verdadeira. Caso contrário, retorna falso.

3. **NOT (NÃO)**: Esta operação inverte o valor da entrada. Se a entrada for verdadeira, retorna falso. Se a entrada for falsa, retorna verdadeiro.

> 💡 Essas operações podem ser combinadas para formar **expressões lógicas** mais complexas. Por exemplo, a expressão "**(A AND B) OR NOT C**" é verdadeira se A e B forem ambos verdadeiros, ou se C for falso.

> 💻 Na programação, usamos a lógica booleana o tempo todo. Por exemplo, quando verificamos se uma condição é verdadeira ou falsa para decidir qual ação o programa deve tomar.

Tabela Verdade: Verdadeiro ou Falso?

A tabela verdade é uma ferramenta simples, mas poderosa, que nos ajuda a entender como diferentes declarações lógicas interagem. Em uma tabela verdade, cada linha representa uma possibilidade diferente. Vamos ver um exemplo simples com duas variáveis booleanas: **A** e **B**.

A	B	A e B	A ou B
V	V	V	V
V	F	F	V
F	V	F	V
F	F	F	F

Nesta tabela, "**V**" representa **verdadeiro** e "**F**" representa **falso**. A coluna "**A e B**" mostra o resultado da operação **AND** (do inglês, significa “e” em português) entre A e B, enquanto a coluna "**A ou B**" mostra o resultado da operação **OR** (do inglês, significa “ou” em português).

Como você pode ver, "**A e B**" é **verdadeiro** apenas quando ambos A e B são verdadeiros. Por outro lado, "**A ou B**" é **verdadeiro** se pelo menos um de A ou B for verdadeiro.

Exemplos

Vamos usar exemplos do dia a dia para ilustrar a tabela verdade. Considere as seguintes declarações:

- A: "Está chovendo."
- B: "Eu tenho um guarda-chuva."

Agora, vamos aplicar a lógica booleana a essas declarações:

A (Está chovendo)	B (Eu tenho um guarda-chuva)	A AND B (Está chovendo e eu tenho um guarda-chuva)	A OR B (Está chovendo ou eu tenho um guarda-chuva)
V	V	V	V
V	F	F	V
F	V	F	V
F	F	F	F

Aqui está o que cada linha da tabela significa:

1. **Primeira linha (V, V):** Está chovendo e eu tenho um guarda-chuva. Como ambas as declarações são verdadeiras, "A AND B" é verdadeiro (eu estou preparado para a chuva). Da mesma forma, "A OR B" também é verdadeiro.

2. **Segunda linha (V, F):** Está chovendo, mas eu não tenho um guarda-chuva. Neste caso, "A AND B" é falso (eu não estou preparado para a chuva), mas "A OR B" é verdadeiro (uma das condições é verdadeira).
3. **Terceira linha (F, V):** Não está chovendo, mas eu tenho um guarda-chuva. Aqui, "A AND B" é falso (não está chovendo, então não importa se eu tenho um guarda-chuva), mas "A OR B" é verdadeiro (uma das condições é verdadeira).
4. **Quarta linha (F, F):** Não está chovendo e eu não tenho um guarda-chuva. Como ambas as declarações são falsas, tanto "A AND B" quanto "A OR B" são falsos.

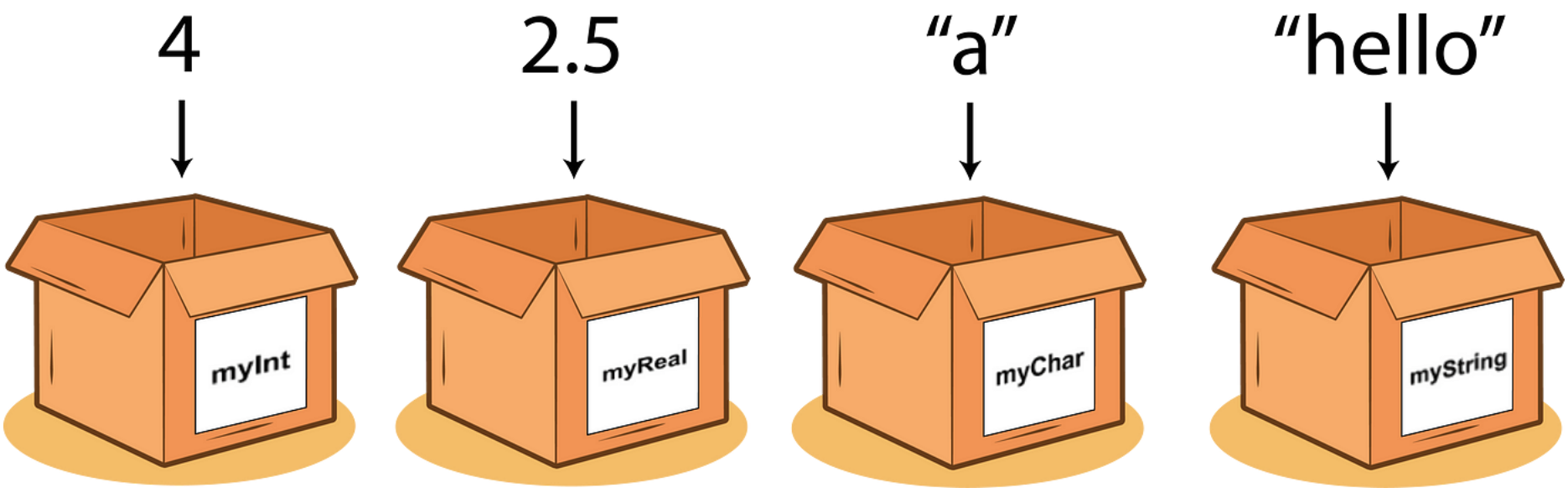
Espero que esses exemplos ajudem a ilustrar como a lógica booleana funciona na prática!

Primeira linha de código: Python 🐍

A primeira linguagem de programação que vamos explorar é o Python. Considerada uma linguagem de alto nível, ela é notável pela sua legibilidade e facilidade de aprendizado. As aplicações do Python são vastas, abrangendo desde o desenvolvimento de programas simples, jogos e sites, até campos avançados como aprendizado de máquina (*Machine Learning*), inteligência artificial (IA) e análise de dados (*Data Analysis*).

Para facilitar a compreensão dos principais elementos da sintaxe do Python, faremos algumas associações com o português. Embora as linguagens de programação sejam escritas predominantemente em inglês, a estrutura do Python é intuitiva e, com um pouco de prática, você será capaz de entender e escrever código, mesmo sem um conhecimento profundo de inglês. Vamos começar?

Variáveis



As variáveis são como caixas onde guardamos nossos dados. Elas têm nomes e guardam valores. No Python, podemos criar uma variável simplesmente atribuindo um valor a ela. Por exemplo:

```
idade = 25
nome = "João"
altura = 1.75
tem_cachorro = True
mora_sozinho = False
```

Aqui, `idade` e `nome` são variáveis. `idade` guarda o valor `25` e `nome` guarda o valor `"João"`.

Tipos de Dados

Os tipos de dados são como as diferentes formas que nossos dados podem assumir. No Python, temos vários tipos de dados, incluindo:

- `int` (inteiro): `idade = 25`
- `float` (número de ponto flutuante): `altura = 1.75`
- `str` (string, ou cadeia de caracteres): `nome = "João"`
- `bool` (booleano, que pode ser **Verdadeiro** ou **Falso**): `tem_cachorro = True`

Assimilando conhecimento

🧵 O tipo `str`, sendo a abreviação do inglês *string* (corda, barbante), na programação é uma cadeia de caracteres. Ou seja, é composta por letras, números e/ou símbolos, que são sempre tratados como texto, independentemente do seu conteúdo.

💯 O tipo `int`, sendo a abreviação do inglês *integer* (inteiro), é como um número inteiro. Isso significa que são números que não possuem casas decimais.

🪂 O tipo `float`, sendo a abreviação do inglês *floating point* (ponto flutuante), é como um número flutuante. Isso significa que, ao contrário dos números inteiros, são números que possuem casas decimais ou “números quebrados”.

📐 O tipo `bool` corresponde aos valores **True** (Verdadeiro) ou **False** (Falso). O nome foi dado em homenagem a Álgebra Booleana de George Boole.

Operadores

Os operadores são como os **verbos** da nossa linguagem de programação. Eles realizam ações em nossos dados. No Python, temos vários tipos de operadores:

Operadores Aritméticos

Os operadores aritméticos são como as operações matemáticas que aprendemos na escola. Eles incluem:

- `+` (Adição): `5 + 3` é igual a `8`.
- `-` (Subtração): `10 - 7` é igual a `3`.
- `*` (Multiplicação): `4 * 2` é igual a `8`.
- `/` (Divisão): `9 / 3` é igual a `3`.
- `%` (Módulo): `10 % 3` é igual a `1`, pois é o resto da divisão de 10 por 3.
- `**` (Exponenciação): `2 ** 3` é igual a `8`, pois é 2 elevado à potência de 3.

Operadores de Comparação

Os operadores de comparação são usados para comparar valores. Eles são como perguntas que fazemos ao Python. Eles incluem:

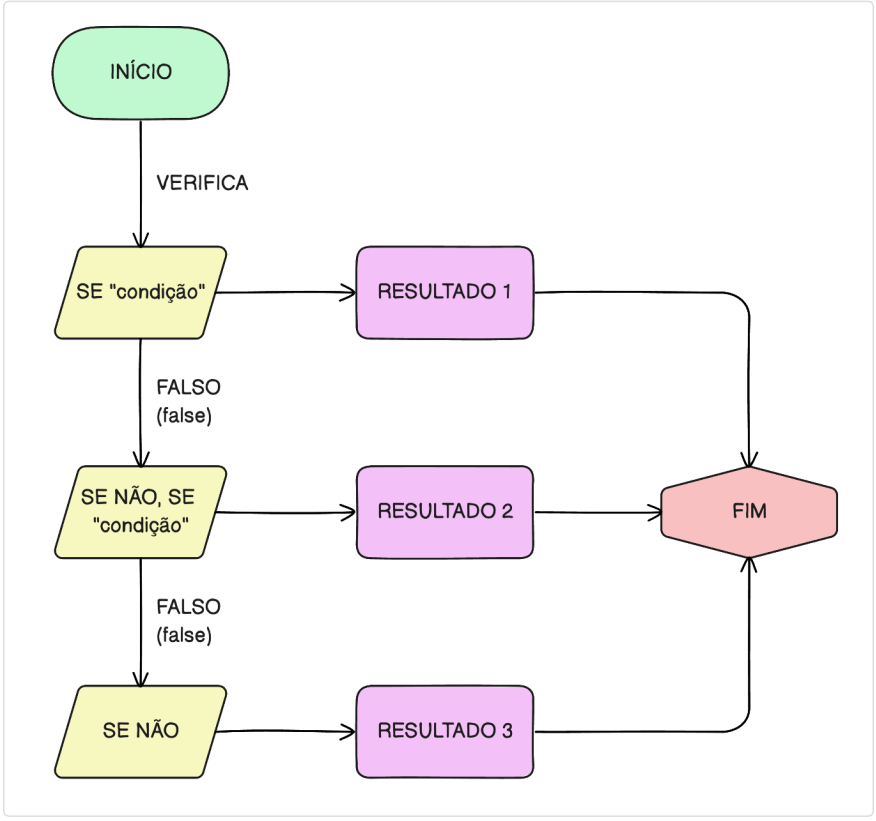
- `==` (Igual a): `5 == 3` retorna `False`, pois 5 **não é igual** a 3.
- `!=` (Diferente de): `5 != 3` retorna `True`, pois 5 **é diferente** de 3.
- `>` (Maior que): `5 > 3` retorna `True`, pois 5 **é maior** que 3.
- `<` (Menor que): `5 < 3` retorna `False`, pois 5 **não é menor** que 3.
- `>=` (Maior ou igual a): `5 >= 5` retorna `True`, pois 5 **é maior ou igual** a 5.
- `<=` (Menor ou igual a): `5 <= 3` retorna `False`, pois 5 **não é menor ou igual** a 3.

Operadores Lógicos

Os operadores lógicos são usados para combinar condições. Eles são como conjunções que usamos para combinar frases. Eles incluem:

- `and` (**E**): `5 > 3 and 5 > 4` retorna `True`, pois ambas as condições são verdadeiras.
- `or` (**OU**): `5 > 3 or 5 < 4` retorna `True`, pois pelo menos uma das condições é verdadeira.
- `not` (**NÃO**): `not 5 < 3` retorna `True`, pois 5 não é menor que 3.

Estruturas de Decisão



As estruturas de decisão são como encruzilhadas na estrada. Dependendo de uma condição, o programa pode seguir um caminho ou outro. No Python, usamos `if`, `elif` e `else` para criar essas estruturas.

Primeiro, em **português**:

```
idade = 20
SE idade < 18:
    ESCREVA("Menor de idade.")
SE NÃO, SE idade == 18:
    ESCREVA("Acabou de se tornar maior de idade.")
SENÃO:
    ESCREVA("Maior de idade.")
```

Agora, em **Python**:

```
idade = 20
if idade < 18:
    print("Menor de idade.")
elif idade == 18:
    print("Acabou de se tornar maior de idade.")
else:
    print("Maior de idade.")
```

Nesse exemplo, se a `idade` for menor que `18`, o programa imprimirá 'Menor de idade'. Se a `idade` for exatamente `18`, o programa imprimirá 'Acabou de se tornar maior de idade'. Caso contrário, ou seja, se a idade for maior que `18`, o programa imprimirá 'Maior de idade'.”

> O exemplo acima irá mostrar “Maior de idade”, pois a `idade` foi definida como `20`.

✏️ O `print` é uma função nativa da linguagem Python. Isso significa que ela é um recurso da linguagem que já está implementado e que tem a função de imprimir/mostrar os resultados na tela do computador.

📦 *Pensando fora da caixa:* O `elif` é uma facilidade na escrita e funciona como a junção dos termos `else` + `if`.

Estruturas de Repetição

As estruturas de repetição são como voltas em uma corrida. Elas permitem que o programa repita um bloco de código várias vezes. No Python, temos `for` e `while` para criar essas estruturas. Por exemplo:

```
for i in range(5):
    print(i)
```

Nesse exemplo, o programa imprimirá os números de `0` a `4`. A função `range(5)` gera uma sequência de números de `0` a `4`, e o `for` percorre essa sequência, imprimindo cada número.

💡 Normalmente, a contagem é feita começando do número 0 (zero) em diante e não do número 1 (um). Porém é possível que você defina que a contagem deva começar a partir do número 1, basta escrever como `range(1,5)`. Isso irá escrever os números `1,2,3,4` ao invés de `0,1,2,3,4`.

Um exemplo de uso do `while`. Primeiro, em **português**:

```
i = 0
ENQUANTO i < 5:
    ESCREVA(i)
    i += 1
```

Agora, em **Python**:

```
i = 0
while i < 5:
    print(i)
    i += 1
```

Nesse exemplo, o programa também imprimirá os números de `0` a `4`. A diferença é que, em vez de gerar uma sequência de números com `range`, estamos iniciando `i` em `0` e incrementando/acrescentando `i` em `1` a cada volta do loop. O loop `while` continuará enquanto a condição `i < 5` for verdadeira.

🔛 O `while`, do inglês *while* (em português significa enquanto), é uma estrutura de repetição que permite executar um bloco de código **enquanto** uma **condição for verdadeira**. No exemplo acima, a **condição** é `i < 5`, então o bloco de código será executado enquanto `i` for menor que `5`. A cada volta do loop, `i` é incrementado em `1` com `i += 1`, então eventualmente `i` será igual a `5` e o loop terminará.

🔁 A palavra `loop`, em inglês, significa laço ou ciclo. No contexto da programação, um loop é uma estrutura de controle de fluxo que permite repetir um bloco de código várias vezes. Existem diferentes tipos de loops, como o `for` e o `while`, que permitem controlar o número de repetições ou a condição para a repetição, respectivamente.

Funções

As funções são como pequenas fábricas que pegam alguns *inputs*, fazem algo com eles e então retornam um *output*. No Python, definimos uma função com a palavra-chave `def`. Por exemplo:

```
def saudacao(nome):
    return "Olá, " + nome + "!"
```

Nesse exemplo, `saudacao` é uma função que recebe um `nome` como *input* e retorna uma saudação personalizada. Esse *input* da função também é chamado de “parâmetro da função”, onde ela pode receber um ou mais parâmetros.

🧩 As palavras `input` e `output`, em inglês, significam **entrada** e **saída**, respectivamente. No contexto da programação, *input* se refere aos dados ou informações que são fornecidos para um programa, enquanto *output* se refere aos dados ou informações que um programa produz.

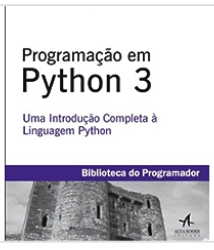
Por exemplo, se você tem um programa que soma dois números, os números seriam o *input* e o resultado da soma seria o *output*. Em Python, você pode usar a função `input()` para **receber** dados do usuário (*input*) e a função `print()` para **mostrar** dados para o usuário (*output*).

Bibliografia

Programação em Python 3

Python 3 é a melhor versão já lançada no que diz respeito à linguagem Python. Ela é muito mais poderosa, conveniente, consistente e expressiva do que jamais foi. Agora, no comando do programador Mark Summerfield, você poderá aprender como escrever códigos que se beneficiam totalmente das vantagen...

https://www.amazon.com.br/Programação-em-Python-Mark-Summerfield/dp/8576083841/ref=asc_df_8576083841/?tag=googleshopp00-20&linkCode=df0&hvadid=379748659420&hvpos=&hvnetw=g&hvrand=12341491743603922798&h



vpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9101531&hvtargid=pla-812887615857&psc=1
Tudo o que você precisa saber sobre filosofia: O guia completo da filosofia para você abrir a mente sem sofrer

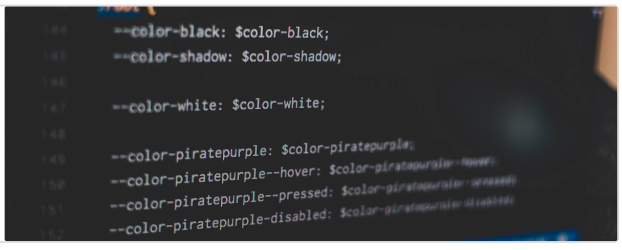
Compre online Tudo o que você precisa saber sobre filosofia: O guia completo da filosofia para você abrir a mente sem sofrer, de Kleinman, Paul na Amazon. Frete GRÁTIS em milhares de produtos com o Amazon Prime. Encontre diversos livros escritos por Kleinman, Paul com ótimos preços.

https://www.amazon.com.br/Tudo-precisa-saber-sobre-filosofia/dp/8573129727

What is a Variable?

I mean it varies. But what else? What is a constant?

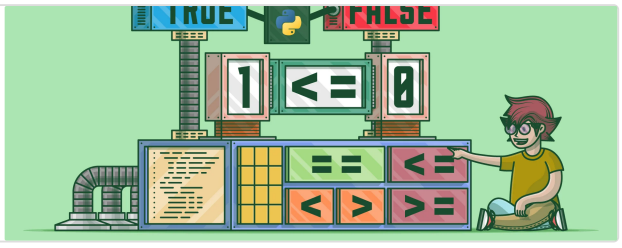
https://stevenpcurtis.medium.com/what-is-a-variable-3447ac1331b9



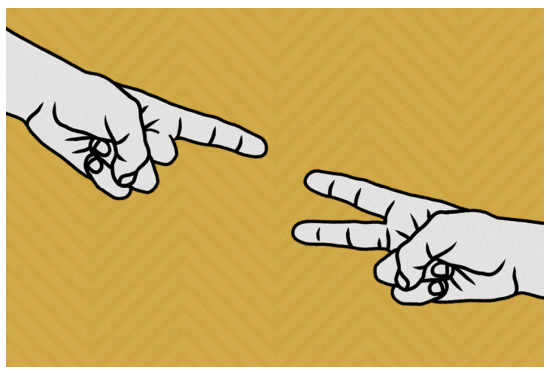
Python Booleans: Use Truth Values in Your Code – Real Python

In this tutorial, you'll learn about the built-in Python Boolean data type, which is used to represent the truth value of an expression. You'll see how to use Booleans to compare values, check for identity and membership, and control the flow of your programs with conditionals.

https://realpython.com/python-boolean/



Projeto - Hands on 👋💻



Jogo de Ímpar ou Par

Neste exemplo, vamos criar um jogo simples de ímpar ou par. Dois jogadores irão escolher seus números e decidir se querem ímpar ou par. O programa então irá somar os números e determinar o vencedor.

```
# Primeiro, vamos pedir os nomes dos jogadores
jogador1 = input("-> Digite o nome do primeiro jogador: ")
jogador2 = input("-> Digite o nome do segundo jogador: ")

# Agora, vamos perguntar quem escolhe primeiro
print(f"// Quem escolhe primeiro, {jogador1} ou {jogador2}?")
primeiro_jogador = input(f"-> Digite 1 para escolher o {jogador1} ou 2 para escolher o {jogador2}.")
segundo_jogador = None

if primeiro_jogador == "1":
    primeiro_jogador = jogador1
    segundo_jogador = jogador2
elif primeiro_jogador == "2":
    primeiro_jogador = jogador2
    segundo_jogador = jogador1

# O jogador que escolhe primeiro decide se quer ímpar ou par
print("// Digite 0 para escolher PAR ou 1 para escolher ÍMPAR")
escolha = input(f"-> {primeiro_jogador}, você quer ímpar ou par? ")
escolha2 = None

# A escolha do segundo jogador é automaticamente definida
if escolha == "ímpar" or escolha == "impar" or escolha == "1":
    escolha = "ímpar"
    escolha2 = "par"
elif escolha == "par" or escolha == "0":
    escolha = "par"
    escolha2 = "ímpar"

print(f"// {segundo_jogador}, você ficou com {escolha2}")

# Agora, cada jogador escolhe um número
numero1 = int(input(f"-> {primeiro_jogador}, escolha um número: "))
numero2 = int(input(f"-> {segundo_jogador}, escolha um número: "))

# A soma dos números é calculada
soma = numero1 + numero2
```

```
# O vencedor é determinado com base na soma
if soma % 2 == 0:
    vencedor = "par"
else:
    vencedor = "ímpar"

if vencedor == escolha:
    print(f"// {primeiro_jogador} venceu!")
else:
    print(f"// {segundo_jogador} venceu!")
```

No código acima, usamos a função `input()` para receber os nomes dos jogadores e suas escolhas, e a função `print()` para mostrar os resultados. A operação `%` é o operador de módulo, que retorna o resto da divisão de dois números. Se a soma dos números for divisível por 2 (ou seja, se o resto da divisão por 2 for 0), então a soma é par. Caso contrário, a soma é ímpar.

Explicando a lógica 🧠

A lógica do jogo é simples. Primeiro, pedimos os nomes dos jogadores e quem escolhe primeiro se quer "ímpar" ou "par". A escolha do segundo jogador é automaticamente definida com base na escolha do primeiro jogador.

Depois, cada jogador escolhe um número. A soma dos números é calculada e, em seguida, verificamos se a soma é par ou ímpar. Se a soma for par e o primeiro jogador tiver escolhido "par", ele é o vencedor. Se a soma for ímpar e o primeiro jogador tiver escolhido "ímpar", ele também é o vencedor. Caso contrário, o segundo jogador vence.

Este é um exemplo simples de como você pode usar Python para criar um jogo interativo. Com um pouco de criatividade, você pode expandir este código para criar jogos mais complexos e interessantes!

Espero que este exemplo tenha sido útil para entender como criar um jogo simples em Python. Lembre-se, a prática é a chave para se tornar um bom programador. Então, continue praticando e experimentando com diferentes tipos de projetos!

🚩 Bônus: *Template Strings*, o que são?

Vamos falar sobre as `template strings` em Python, também conhecidas como `f-strings`.

As `f-strings` são uma maneira de formatar strings em Python que permite incorporar expressões dentro de strings de maneira literal. Elas são chamadas de `f-strings` **porque são prefixadas com a letra 'f'**. As `f-strings` foram introduzidas no Python 3.6.

Aqui está um exemplo simples de uma `f-string`:

```
nome = "Maria"
print(f"Olá, {nome}!")
```

Neste exemplo, a variável `nome` é incorporada diretamente na string. Quando o programa é executado, ele substitui `{nome}` pelo valor da variável `nome`, resultando na string **"Olá, Maria!"**.

No código do jogo acima, as `f-strings` são usadas para incorporar os nomes dos jogadores e suas escolhas diretamente nas mensagens impressas. Por exemplo:

```
print(f"// Quem escolhe primeiro, {jogador1} ou {jogador2}?")
```

Nesta linha, `{jogador1}` e `{jogador2}` são substituídos pelos nomes dos jogadores. Isso torna o código mais legível e fácil de escrever, pois você não precisa concatenar strings manualmente.

As `f-strings` também podem incorporar expressões mais complexas, não apenas variáveis. Por exemplo, você pode fazer algo assim:

```
numero = 5
print(f"O quadrado de {numero} é {numero ** 2}.")
```

Neste exemplo, `{numero ** 2}` é uma expressão que calcula o quadrado do número. Quando o programa é executado, ele substitui a expressão pelo seu resultado.

As `f-strings` são uma ferramenta poderosa para formatar strings em Python. Elas são fáceis de usar e tornam o código mais legível e conciso.

Próximo módulo 🔜

Parabéns por ter chegado até aqui! É por meio de cada linha de código escrita que moldamos o futuro. Programação é para algo divertido e dinâmico. Então, caso tenha alguma dúvida, aproveite para revisar o módulo e as aulas.

No próximo módulo vamos falar sobre “Estrutura de dados e algoritmos”, então daremos mais um passo rumo a aprender novos conceitos e colocar a mão na massa, programando mais código. Espero que tenha gostado do nosso primeiro projeto “Jogo de Ímpar ou Par”. Até o próximo módulo!