



Python I - fundamentos

36 horas (9 aulas de 4h horas)

16/09 com previsão de término no dia **14/10/24**,
Segunda e Quarta das 13:30 às 17:30 horas

Prof Diego Rodrigues

Quem sou?

DIEGO RODRIGUES

Docente e Desenvolvedor Fullstack

11 99145-7584

<https://github.com/diegoartcode/python1-2024>

O que vou aprender?

- **Ambiente de desenvolvimento:** instalação e configuração de ambientes locais e utilização de ambientes na nuvem.
- **Lógica de programação:** comandos de entrada, processamento e saída de dados, tipos de dados dinâmicos, variáveis e constante, expressões e operadores.
- **Estruturas de programação:** controle de fluxo (condicional simples e composta), repetição, cláusulas break, continue e pass e função range.
- **Estruturas de dados:** listas, tuplas e dicionários.
- **Funções:** criação, acesso e parâmetros.

O que vou aprender?

- **Otimização de código:** tratamento de exceções.
- **Módulos de terceiros:** instalação e uso, leitura e escrita de arquivos CSV e JSON.
- **Beautiful soup:** pesquisa, busca e extração de dados de páginas web.

Senac

O que é linguagem Python?

Python é uma linguagem de programação de alto nível, criada para ser fácil de ler e escrever. Ela é uma das linguagens mais populares no mundo da programação, por ser simples, versátil e eficiente.

Características principais:

Sintaxe limpa e intuitiva: Código mais legível, o que facilita o aprendizado.

Multiparadigma: Suporta diferentes estilos de programação, como a programação orientada a objetos, funcional e estruturada.

Interpretada: O código Python é executado linha por linha pelo interpretador, o que facilita o teste e a depuração.

Portabilidade: Funciona em diferentes sistemas operacionais (Windows, Linux, macOS) sem a necessidade de modificações.

Como Python foi criada?

Python foi criado por Guido van Rossum no final dos anos 1980, e lançado pela primeira vez em 1991. Guido queria desenvolver uma linguagem que fosse acessível tanto para programadores iniciantes quanto para os mais experientes.

Motivações:

- Ele queria que a linguagem fosse uma evolução das linguagens que existiam na época, como o ABC.
- O nome Python foi inspirado no grupo de comédia britânico Monty Python, que Guido era fã, e não tem relação com a cobra.

Python foi projetada para ser uma linguagem que combinasse:

- Simplicidade.
- Produtividade.
- Flexibilidade.

A linguagem rapidamente cresceu em popularidade e, com o tempo, diversas melhorias foram feitas. Hoje, a linguagem é mantida pela Python Software Foundation (PSF) e a comunidade global de desenvolvedores.

Onde e como Python pode ser utilizada?

Python é uma linguagem extremamente versátil, usada em diversas áreas da tecnologia e ciências. Aqui estão alguns dos principais campos em que Python é amplamente utilizado:

Desenvolvimento Web: Frameworks como Django e Flask permitem criar sites e aplicativos web de forma rápida e eficiente.

Ciência de Dados e Machine Learning: Python é amplamente usado em análises de dados, aprendizado de máquina e inteligência artificial, com bibliotecas poderosas como Pandas, NumPy, Scikit-learn, e TensorFlow.

Automação de Tarefas: Python é excelente para criar scripts que automatizam processos repetitivos, como mover arquivos, processar dados e interagir com APIs.

Desenvolvimento de Jogos: A biblioteca Pygame permite criar jogos em 2D de maneira simples.

Onde e como Python pode ser utilizada?

Administração de Sistemas: Administradores de sistemas usam Python para escrever scripts que automatizam tarefas de rede e servidores.

Aplicativos Desktop: Com bibliotecas como Tkinter e PyQt, você pode criar interfaces gráficas de usuário (GUIs) para aplicativos desktop.

Big Data e Processamento de Dados: Python é amplamente usado em aplicações de big data devido à sua simplicidade e eficiência em manipular grandes volumes de dados.

Internet das Coisas (IoT): Python é utilizado para controlar dispositivos conectados e sensores no desenvolvimento de IoT.

Ambiente de desenvolvimento Python – instalação e configuração de ambientes locais e utilização de ambientes na nuvem

O que é um Ambiente de Desenvolvimento?

Um ambiente de desenvolvimento Python é onde o código é escrito, testado e executado. Ele pode ser configurado localmente (no computador do desenvolvedor) ou em um servidor na nuvem (acessível via internet).

Senac

Tipos de ambiente de desenvolvimento

Ambiente Local:

Descrição: O desenvolvimento e a execução do código Python ocorrem diretamente no computador do desenvolvedor.

Ferramentas principais:

- **Python:** Linguagem de programação.
- **Editor de Texto/IDE:** VSCode, PyCharm, Sublime Text, entre outros.
- **Gerenciador de Pacotes:** pip ou conda.
- **Virtual Environments:** Utilizado para isolar dependências de projetos.

Vantagens:

- Total controle sobre o ambiente e as versões de bibliotecas.
- Funciona offline.

Desvantagens:

- Pode ser difícil replicar em outros dispositivos ou para outros desenvolvedores.
- Requer gerenciamento manual de versões de pacotes.

Tipos de ambiente de desenvolvimento

Ambientes na Nuvem:

Descrição: O código Python é desenvolvido e executado em servidores acessados via internet.

Ferramentas principais:

- **Google Colab:** Oferece Jupyter Notebooks com execução na nuvem.
- **Replit:** Ambiente online para várias linguagens, incluindo Python.
- **GitHub Codespaces:** Oferece um ambiente de desenvolvimento online completo com integração ao GitHub.

Vantagens:

- Configuração simplificada, acessível de qualquer dispositivo.
- Facilita o desenvolvimento colaborativo.

Desvantagens:

- Depende de uma conexão com a internet.
- Pode ter limitações de desempenho ou tempo de execução (em versões gratuitas).

Utilizando um ambiente de desenvolvimento local

Prática

Configuração de um Ambiente Local de Desenvolvimento Python

Passo 1: Instalação do Python

- Windows/Mac/Linux: Baixar o instalador em python.org/downloads e seguir as instruções da plataforma.

Verifique a instalação com o comando no terminal



```
python --version
```

Utilizando um ambiente de desenvolvimento local

Passo 2: Configuração de um Editor/IDE

VSCode:

- Baixar e instalar o VSCode ([link](#)).
- Instalar a extensão Python no VSCode (buscar "Python" no marketplace de extensões).

Utilizando um ambiente de desenvolvimento local

Passo 3: Gerenciamento de Pacotes

Usar o pip para instalar bibliotecas e dependências:

Documentação: pip.pypa.io

Usando o pip



```
pip install nome_do_pacote
```

Ver versão do pip



```
pip --version
```

Utilizando um ambiente de desenvolvimento local

Passo 4: Criação de um Ambiente Virtual

Criar um ambiente virtual para isolar as dependências



```
python -m venv nome_do_ambiente
```


Utilizando um ambiente de desenvolvimento local

Ativar o ambiente:

Windows

```
nome_do_ambiente\Scripts\activate
```

Mac/Linux

```
source nome_do_ambiente/bin/activate
```

Utilizando um ambiente de desenvolvimento local

Passo 5: Teste com um Script Python Simples

Criar um arquivo `app.py` com o seguinte código



```
print("Olá, mundo!")
```

Rodar o script:



```
python app.py
```

Utilizando um Ambiente de Desenvolvimento na Nuvem

Opção 1: Google Colab

Acesse Google Colab e crie um novo notebook.

No primeiro bloco de código, digite:



```
print("Olá, mundo no Colab!")
```

Executar o bloco (CTRL + ENTER).

Utilizando um Ambiente de Desenvolvimento na Nuvem

Opção 2: Replit

Acesse [Replit](#) e crie um novo projeto em Python.

No editor, adicione o código:



```
print("Olá, mundo no Replit!")
```

Execute o código diretamente pelo navegador.

Vantagens e Desvantagens

	Ambiente Local	Ambiente na Nuvem
Vantagens	Controle total sobre dependências	Acessível de qualquer dispositivo
	Não depende da internet	Configuração automática
Desvantagens	Pode ser mais rápido em alguns casos	Ideal para colaboração em tempo real
	Mais trabalhoso de configurar	Limitações de recursos na versão gratuita

Variáveis

Uma variável é um espaço na memória do computador utilizado para armazenar dados que podem ser modificados durante a execução de um programa, como números, textos ou valores lógicos.

nome	←	"Diego"	str - string
altura	←	1,72	float - números com casas decimais ou chamado de número de ponto flutuante
idade	←	32	int - números inteiros
tem_pet	←	true	boolean - true ou false / verdadeiro ou falso

Variáveis

Regras para Criar Variáveis no Python

Ao criar variáveis, é importante seguir algumas regras e boas práticas. Vamos detalhar essas regras para garantir que suas variáveis sejam válidas e compreensíveis no código.

Não Podemos Usar **Palavras Reservadas**

Python tem uma série de **palavras reservadas**, que são comandos ou funções especiais usadas pela própria linguagem. Você não pode usar essas palavras para nomear suas variáveis, pois isso causaria conflitos.

Exemplo de palavras reservadas:

and, if, else, for, while, True, False, None, return, try, import, entre outras.

Variáveis

Exemplo inválido:



```
if = 10 # Isso causará um erro, pois "if" é uma palavra reservada no Python.
```

Para verificar todas as palavras reservadas em Python, você pode usar o seguinte código:



```
import keyword  
print(keyword.kwlist)
```

Variáveis

Não Podemos Iniciar com Números

O nome de uma variável não pode começar com um número. Isso porque o Python espera que o nome da variável comece com uma letra ou um underscore (_).

Exemplo inválido:



```
1variavel = 100 # Erro: uma variável não pode começar com um número.
```

Forma correta:



```
variavel1 = 100 # Isso é válido, pois o número está no final do nome da variável.
```

Variáveis

Não Podemos Usar Caracteres Especiais

Os nomes de variáveis não podem conter caracteres especiais como @, #, %, &, etc. O único caractere especial permitido é o underscore (_), que muitas vezes é usado para separar palavras no nome da variável.

Exemplo inválido:

```
meu@nome = "Carlos" # Erro: o caractere '@' não é permitido no nome de variáveis.
```

Forma correta:

```
meu_nome = "Carlos" # Válido: usamos o underscore para separar palavras.
```

Variáveis

Letras Maiúsculas e Minúsculas São Diferentes

Python diferencia letras maiúsculas de minúsculas, o que significa que `variavel` e `Variavel` são consideradas duas variáveis diferentes.

Exemplo:

```
variavel = 10
Variavel = 20

print(variavel) # Saída: 10
print(Variavel) # Saída: 20
```

Variáveis

Boas práticas ao criar variáveis

Agora que você sabe as regras básicas, vamos ver algumas boas práticas que ajudam a tornar seu código mais organizado e fácil de entender.

Escolha nomes descritivos

Sempre que possível, use nomes de variáveis que descrevam o propósito ou o conteúdo da variável. Isso torna seu código mais legível e fácil de entender.



#Exemplo ruim:

```
a = 10
```

```
b = 20
```

#Exemplo bom:

```
idade_aluno = 10
```

```
numero_de_aulas = 20
```

Variáveis

Use o Formato Snake Case

Uma convenção comum no Python é usar **snake_case** para nomear variáveis, onde as palavras são separadas por **underscores** (_). Isso ajuda na legibilidade.



```
media_salarial = 1500.50  
nome_completo = "Jhenny Silva"
```

Constante

O que é uma Constante?

No Python, uma constante é um valor que, em teoria, não deveria ser alterado ao longo da execução do programa. Diferente de variáveis, que podem mudar de valor, as constantes são usadas para valores fixos que permanecem os mesmos durante a execução do código.

Por exemplo, o valor de PI (3.14159) é algo que não muda, então, faz sentido defini-lo como uma constante.

Constante

Boas Práticas para Constantes

Mesmo que o Python permita modificar constantes, a boa prática é que, uma vez definidas, elas não devem ser alteradas no decorrer do código. Isso ajuda na organização e legibilidade, permitindo que outros desenvolvedores saibam que esses valores são fixos.

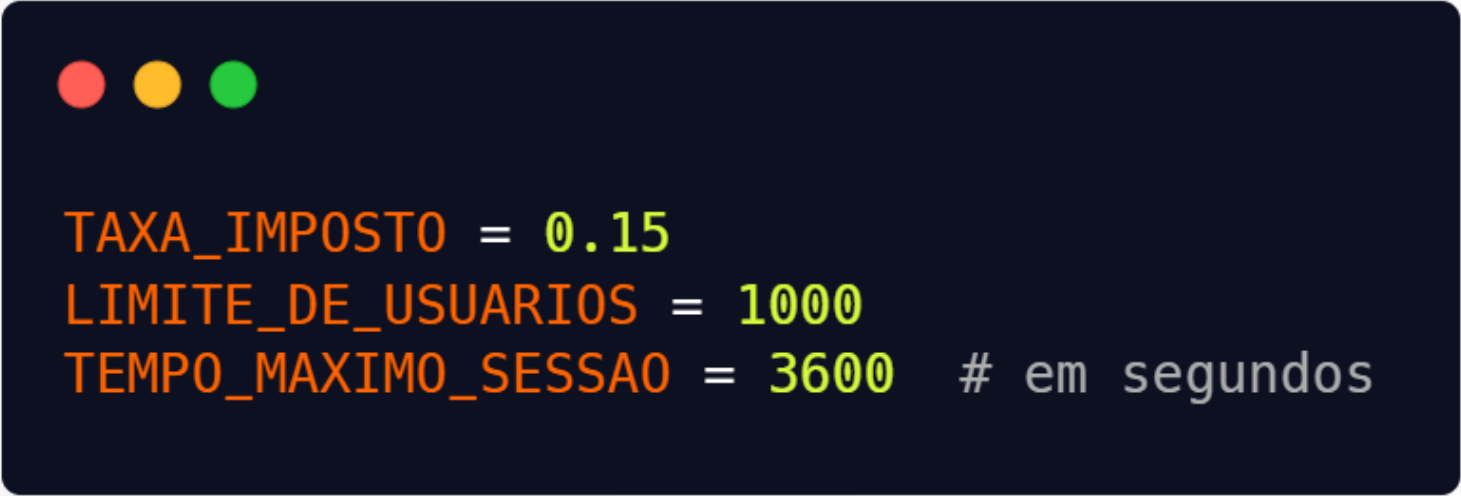
Convenção para Constantes:

Nomes em maiúsculas: O nome de uma constante deve ser todo em letras maiúsculas para que seja imediatamente reconhecida.

Usar underscore para separação: Se o nome for composto por mais de uma palavra, utilize o underscore (_) para separar palavras.

Constante

Exemplo de boas práticas:



```
TAXA_IMPOSTO = 0.15  
LIMITE_DE_USUARIOS = 1000  
TEMPO_MAXIMO_SESSAO = 3600 # em segundos
```

Diferença entre constantes e variáveis

Constante

O valor não deve mudar

Nomeada com letras maiúsculas

Usada para valores fixos

Variável

O valor pode ser alterado

Nomeada com letras minúsculas e, geralmente, no formato snake_case

Usada para armazenar dados que mudam durante a execução

Variável - String - str

Uma variável do tipo **string** em programação é usada para armazenar textos, ou seja, uma sequência de caracteres. Em Python, strings são delimitadas por aspas simples ('...') ou aspas duplas ("..."), e podem conter letras, números, espaços e símbolos.

Variável - String - str

No exemplo, a variável **nome** armazena o texto "Diego", e a variável **mensagem** contém uma saudação. Também é possível **concatenar (juntar) strings**, como mostrado na variável **saudacao**.

```
# Declarando uma variável string
nome = "Diego"
mensagem = 'Olá, bem-vindo ao curso de Python!'

# Usando a variável string
print(nome)           # Saída: Diego
print(mensagem)       # Saída: Olá, bem-vindo ao curso de Python!

# Concatenando strings
saudacao = "Olá, " + nome + "!"
print(saudacao)       # Saída: Olá, Diego!
```

As strings podem ser manipuladas de várias formas, como extrair partes do texto, modificar o conteúdo, ou até realizar operações como formatação.

Variável - Float

A variável do tipo float em Python é usada para armazenar números com casas decimais, também conhecidos como números de ponto flutuante. Esse tipo de variável é útil para representar valores que não são inteiros, como 3.14 ou 2.718.

Variável - Float

Variável **altura** armazena o valor 1.75 (metros), e a variável **peso** armazena 68.5 (quilogramas).

Calculamos o IMC (Índice de Massa Corporal) dividindo o peso pela altura ao quadrado.

O resultado do IMC é arredondado para duas casas decimais usando **round(imc, 2)**.

```
# Declarando variáveis do tipo float
altura = 1.75
peso = 68.5

# Realizando uma operação com float
imc = peso / (altura ** 2)

# Exibindo o resultado
print("Altura:", altura)          # Saída: Altura: 1.75
print("Peso:", peso)              # Saída: Peso: 68.5
print("IMC:", round(imc, 2))      # Saída: IMC: 22.37
```

Números do tipo **float** são frequentemente usados em cálculos que envolvem precisão decimal, como medidas, porcentagens e valores financeiros.

Variável - Int

A **variável do tipo int** em Python é usada para armazenar números inteiros, ou seja, números sem casas decimais, podendo ser positivos, negativos ou zero.

Variável - Int

A variável **idade** armazena o valor inteiro 25.

A variável **ano_atual** armazena 2024.

Usamos uma operação matemática para calcular o ano de nascimento (**ano_nascimento**) subtraindo a idade do ano atual.

```
# Declarando variáveis inteiras
idade = 25
ano_atual = 2024
ano_nascimento = ano_atual - idade

# Exibindo os valores
print("Idade:", idade) # Saída: Idade: 25
print("Ano de nascimento:", ano_nascimento) # Saída: Ano de
nascimento: 1999
```

Variáveis do tipo **int** são usadas sempre que precisamos trabalhar com contagem, índices, ou cálculos inteiros.

Variável - Boolean - bool

Uma **variável booleana** (ou do tipo `boolean`) em Python é usada para armazenar apenas dois valores possíveis: **True (verdadeiro)** ou **False (falso)**. Essas variáveis são comumente utilizadas em operações de controle de fluxo, como condicionais e loops, para representar estados binários, como "**ligado/desligado**" ou "**verdadeiro/falso**".

Variável - Boolean - bool

A variável **esta_logado** armazena o valor booleano **True** (indica que o usuário está logado).

A variável **tem_permissao** armazena o valor **False** (indica que o usuário não tem permissão).

A estrutura condicional **if** verifica se o usuário está logado e tem permissão; caso ambas as condições sejam verdadeiras, o acesso é permitido. Caso contrário, o acesso é negado.

```
# Declarando variáveis booleanas
esta_logado = True
tem_permissao = False

# Usando as variáveis booleanas em uma condicional
if esta_logado and tem_permissao:
    print("Acesso permitido.")
else:
    print("Acesso negado.")

# Exibindo os valores booleanos
print("Está logado:", esta_logado) # Saída: Está logado: True
print("Tem permissão:", tem_permissao) # Saída: Tem permissão: False
```

As variáveis booleanas são muito úteis para controlar o fluxo de um programa e tomar decisões com base em condições lógicas.

Função - print()

O print() em Python é uma função embutida usada para exibir informações no console (ou em qualquer outro fluxo de saída padrão). Ele pode imprimir texto, números, variáveis, listas, tuplas, entre outros, e é uma das funções mais simples e amplamente utilizadas no Python.

```
nome = "Diego"
print(nome) # Saída: Diego
print("oi") # Saída: oi
print(1 + 1) # Saída: 2
print(10 / 2) # Saída: 5
```

Função - input()

A função `input()` em Python é usada para capturar dados inseridos pelo usuário através do teclado. Ela permite que o programa pause e aguarde a entrada do usuário, retornando o valor como uma string. Se for necessário trabalhar com outros tipos de dados, como inteiros ou floats, o valor retornado deve ser convertido.

Função - input()

Exemplo básico de uso do input():

```
# Solicitando entrada de dados do usuário
nome = input("Qual é o seu nome? ")

# Exibindo o valor inserido pelo usuário
print("Olá, " + nome + "! Bem-vindo ao curso de Python.")
```

Explicação: O usuário é solicitado a inserir o nome, e a função **input()** captura essa informação como uma string e guarda essa informação na variável nome.

Função - input()

Exemplo com conversão para outro tipo de dado:

```
# Solicitando a idade do usuário e convertendo para inteiro
idade = int(input("Quantos anos você tem? "))

# Realizando uma operação com a entrada
ano_nascimento = 2024 - idade

# Exibindo o resultado
print("Você nasceu no ano de", ano_nascimento)
```

Explicação: A função input() captura a idade como uma string, e usamos int() para convertê-la em um número inteiro, permitindo fazer cálculos com o valor inserido.

Função - type()

A função `input()` em Python é usada para capturar dados inseridos pelo usuário através do teclado. Ela permite que o programa pause e aguarde a entrada do usuário, retornando o valor como uma string. Se for necessário trabalhar com outros tipos de dados, como inteiros ou floats, o valor retornado deve ser convertido.

Senac

Função - type()

type(nome): Retorna `<class 'float'>`, indicando que nome é uma **string**.

type(idade): Retorna `<class 'float'>`, indicando que idade é um **inteiro**.

type(altura): Retorna `<class 'float'>`, indicando que altura é um número de **ponto flutuante**.

type(esta_logado): Retorna `<class 'bool'>`, indicando que esta_logado é um valor **booleano**.

```
# Exemplos de variáveis
nome = "Jhenny"           # String
idade = 25                 # Inteiro
altura = 1.68              # Float
esta_logado = True         # Booleano

# Usando a função type() para verificar o tipo de cada
# variável
print(type(nome))          # Saída: <class 'str'>
print(type(idade))         # Saída: <class 'int'>
print(type(altura))        # Saída: <class 'float'>
print(type(esta_logado))   # Saída: <class 'bool'>

# Verificando o tipo de uma expressão
numero = 5 + 3.0
print(type(numero))        # Saída: <class 'float'> (porque 5
# + 3.0 resulta em um float)
```

Operadores

Em Python, operadores são símbolos que permitem realizar operações em variáveis e valores. Aqui estão alguns dos principais operadores em Python.

Operadores aritméticos

Operadores Aritméticos	
Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Divisão inteira	//
Resto da divisão	%
Exponenciação	**

Operador de adição



```
a = 5
```

```
b = 3
```

```
soma = a + b
```

```
print(soma) # Saída: 8
```

Operador de subtração



```
a = 10  
b = 4  
diferenca = a - b  
print(diferenca) # Saída: 6
```

Operador de multiplicação



```
a = 7
```

```
b = 3
```

```
produto = a * b
```

```
print(produto) # Saída: 21
```

Operador de divisão



```
a = 10
```

```
b = 2
```

```
divisao = a / b
```

```
print(divisao) # Saída: 5.0
```


Operador de divisão inteira



```
a = 10
```

```
b = 3
```

```
divisao_inteira = a // b
```

```
print(divisao_inteira) # Saída: 3
```

Operador de resto da divisão



```
a = 10
```

```
b = 3
```

```
resto = a % b
```

```
print(resto) # Saída: 1
```

Operador de exponenciação



```
a = 2
```

```
b = 3
```

```
potencia = a ** b
```

```
print(potencia) # Saída: 8
```

Operador de comparação

Operadores de Comparação	
Igual a	==
Diferença	!=
Maior que	>
Menor que	<
Maior ou igual a	>=
Menor ou igual a	<=

Operador de igualdade



```
a = 5  
b = 5  
resultado = a == b  
print(resultado) # Saída: True
```

Operador de diferença



```
a = 5
```

```
b = 3
```

```
resultado = a != b
```

```
print(resultado) # Saída: True
```

Operador maior que



```
a = 7
```

```
b = 4
```

```
resultado = a > b
```

```
print(resultado) # Saída: True
```

Operador menor que



```
a = 3
```

```
b = 6
```

```
resultado = a < b
```

```
print(resultado) # Saída: True
```


Operador maior ou igual a



```
a = 5  
b = 5  
resultado = a >= b  
print(resultado) # Saída: True
```

Operador menor ou igual a



```
a = 4
```

```
b = 7
```

```
resultado = a <= b
```

```
print(resultado) # Saída: True
```

Operadores Lógicos

Operadores Lógicos	
E lógico	and
Ou lógico	or
Não lógico	not

Operadores Lógicos

AND: Retorna True apenas quando todas as suas entradas são True.

OR: Retorna True quando pelo menos uma das suas entradas é True.

NOT: Inverte o valor da entrada; retorna False para entrada True e True para entrada False.

And

False	False	False
False	True	False
True	False	False
True	True	True

Or

False	False	False
False	True	True
True	False	True
True	True	True

Not

False	True
True	False

Operador E lógico (and)



```
a = True  
b = False  
resultado = a and b  
print(resultado) # Saída: False
```

Operador OU lógico (or)



```
a = True  
b = False  
resultado = a or b  
print(resultado) # Saída: True
```

Operador NÃO lógico (not)



```
a = True  
resultado = not a  
print(resultado) # Saída: False
```

Operador de atribuição

Operadores de Atribuição	
Atribuição simples	=
Atribuição com adição	+=
Atribuição com multiplicação	*=

Operador atribuição simples (=)



```
a = 10
```

Operador de atribuição com adição (+=)



```
a = 5
```

```
a += 3 # Equivalente a a = a + 3
```

```
print(a) # Saída: 8
```

Operador de atribuição com multiplicação (*=)



```
a = 4  
a *= 2 # Equivalente a a = a * 2  
print(a) # Saída: 8
```

Condicionais

O que são condicionais?

Condicionais são instruções que permitem tomar decisões com base em condições lógicas. Elas controlam o fluxo do programa, executando diferentes blocos de código dependendo do resultado de uma expressão booleana (verdadeiro ou falso).

As principais instruções condicionais em Python são: **if**, **elif**, e **else**.

Senac

Condicional – if (se)

Sintaxe básica do if

A instrução **if** testa uma condição. Se a condição for verdadeira, o bloco de código dentro do **if** é executado.

- A condição **idade >= 18** é avaliada como verdadeira.
- Como a condição é verdadeira, a mensagem "Você é maior de idade." é exibida.

```
idade = 18

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

Condicional – if (se) e else (se não)

Usando else para lidar com o caso contrário

O **else** é usado para definir um bloco de código que será executado se a condição **if** for falsa.

- Como a condição **idade >= 18** é falsa, o bloco **else** é executado, e a mensagem "Você é menor de idade." é exibida.

```
idade = 16

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

Condicional – elif – composta

Usando elif para múltiplas condições

O **elif** (abreviação de "else if") permite testar várias condições em sequência. Se a condição do **if** for falsa, o Python testa a condição do **elif**. Podemos usar quantos **elif** forem necessários.

- A primeira condição ($\text{nota} \geq 90$) é falsa, então o Python verifica a próxima ($\text{nota} \geq 80$).
- Como a condição $\text{nota} \geq 80$ é verdadeira, a mensagem "Nota B" é exibida.

```
nota = 85

if nota >= 90:
    print("Nota A")
elif nota >= 80:
    print("Nota B")
elif nota >= 70:
    print("Nota C")
else:
    print("Nota D")
```

Aninhando condicionais

Também podemos aninhar condicionais, ou seja, usar um **if** ou **else** dentro de outro **if**.

- A primeira condição **idade >= 18** é verdadeira, então o programa entra no segundo **if**.
- Como **tem_cnh** é verdadeiro, a mensagem "Você pode dirigir." é exibida.

```
idade = 20
tem_cnh = True

if idade >= 18:
    if tem_cnh:
        print("Você pode dirigir.")
    else:
        print("Você não pode dirigir sem CNH.")
else:
    print("Você é menor de idade.")
```


Operadores lógicos com condicionais

Podemos combinar múltiplas condições usando operadores lógicos como and, or, e not.

Operador lógico com condicional - and

- A condição **idade >= 18 and tem_cnh** avalia se ambas as condições são verdadeiras.
- Se ambas forem verdadeiras, a mensagem "Você pode dirigir." é exibida.



```
idade = 20
tem_cnh = True

if idade >= 18 and tem_cnh:
    print("Você pode dirigir.")
else:
    print("Você não pode dirigir.")
```

Operador lógico com condicional - or

- A condição **tem_habilitacao** or **tem_permicao** avalia se uma ou ambas as condições são verdadeiras.
- Se uma delas for verdadeira, a mensagem "Você pode dirigir." é exibida.



```
tem_habilitacao = False
tem_permicao = True

if tem_habilitacao or tem_permicao:
    print("Você pode dirigir.")
else:
    print("Você não pode dirigir.")
```

Próximas aulas

- Laços de repetição
- Break, continue e pass
- Função range
- Listas, tuplas e dicionários
- Função
- Tratamento de exceções.
- Leitura e escrita de arquivos CSV e JSON.
- Pesquisa, busca e extração de dados de páginas web