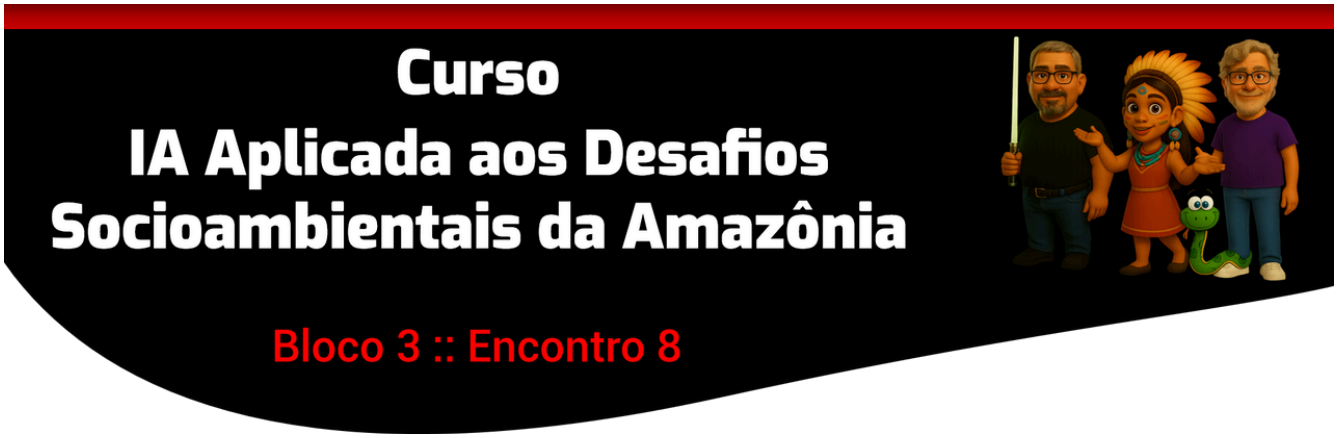
Mostrar código



Sobre o curso

O curso Inteligência Artificial Aplicada aos Desafios Socioambientais da Amazônia, promovido pelo Instituto de Inteligência Artificial Aplicada (I2A2), é uma iniciativa pioneira voltada para capacitar moradores do Pará e da região Norte. Com duração de seis meses, combina aulas online, atividades assíncronas, workshops práticos, encontros com especialistas e mentoria contínua. O objetivo é aplicar a IA de forma prática em problemas ambientais reais, em alinhamento com as diretrizes e temas da COP30.

Ao longo do curso, os participantes exploram desde fundamentos de Machine Learning até IA Generativa, trabalhando com dados ambientais da Amazônia para enfrentar questões como desmatamento, queimadas, qualidade da água e riscos climáticos. Cada grupo finaliza sua jornada com um projeto integrador, apresentando soluções sustentáveis e socialmente viáveis. A formação busca fortalecer competências técnicas e o protagonismo de lideranças locais, articulando ciência, tecnologia e justiça ambiental, consolidando a Amazônia como polo de inovação para o desenvolvimento sustentável.

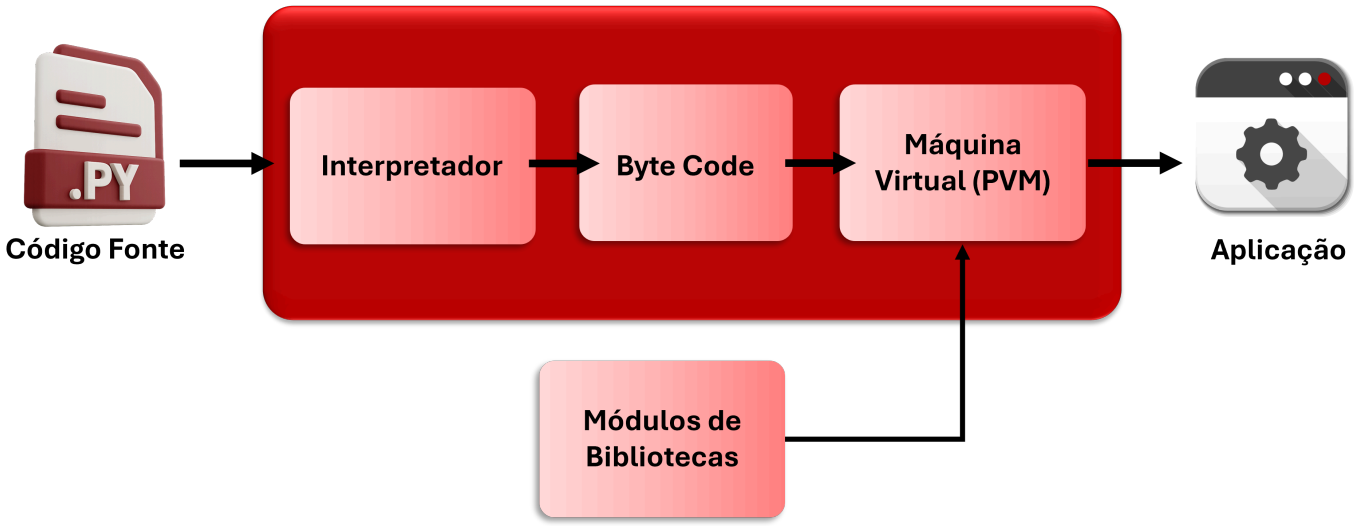
Linguagem Python

☰ Como Funciona um programa de computador

Este curso tem como objetivo principal ensinar do zero como funciona o Python e para que ele serve. Parte do princípio de que um computador, por mais poderoso que seja, não faz nada sem instruções: é o programa que transforma uma máquina inerte em uma ferramenta útil, assim como um pianista transforma um piano em música. Sem código, um computador é apenas um conjunto de peças sem propósito.

Os computadores, embora consigam executar tarefas muito complexas, só o fazem porque combinam operações básicas como somas, subtrações ou divisões de forma extremamente rápida e repetitiva. Eles não compreendem conceitos por conta própria: não sabem o que é distância ou tempo sem receber instruções claras. Por isso, para resolver problemas, como calcular a velocidade média de uma viagem, precisam de passos bem definidos.

Um programa, portanto, é um conjunto de instruções que diz exatamente à máquina o que fazer: receber dados, processá-los por meio de operações simples e apresentar um resultado. Para se comunicar com o computador é necessário um linguagem, como o Python, que traduz nossas ideias para uma forma que a máquina possa executar sem erros, permitindo resolver problemas reais de forma precisa e eficiente.



☰ Linguagens naturais versus linguagens de programação

Uma linguagem é um meio para expressar e registrar pensamentos, e há muitas ao nosso redor. Algumas nem exigem fala ou escrita, como a linguagem corporal, que permite transmitir sentimentos profundos sem dizer uma única palavra. Já a língua materna é usada diariamente para manifestar vontades e refletir sobre a realidade. Da mesma forma, os computadores também possuem seu próprio idioma: a linguagem de máquina, extremamente básica. Mesmo os computadores mais avançados não têm inteligência; são como cães bem treinados, capazes apenas de responder a um conjunto limitado de comandos simples, como pegar um número, dividir por outro e guardar o resultado.

Esse conjunto de comandos é chamado de **lista de instruções, ou IL** (***Instruction List***). Diferentes computadores podem ter ILs de tamanhos variados, com instruções distintas entre modelos. Vale lembrar: as linguagens de máquina são criadas por humanos — até hoje, nenhum computador consegue criar um novo idioma, embora isso possa mudar em breve. Por outro lado, os humanos inventam suas próprias

línguas, que evoluem constantemente com novas palavras surgindo e antigas desaparecendo. Essas são as chamadas linguagens naturais.

☰ Elementos de uma Linguagem

Podemos dizer que todo idioma — seja máquina ou natural — é composto basicamente por quatro elementos: **um alfabeto**, que é o conjunto de símbolos usados para formar palavras (como o alfabeto latino no inglês, o cirílico no russo ou os kanjis no japonês); **um léxico** (ou dicionário), que é o conjunto de palavras que fazem sentido dentro desse idioma; **uma sintaxe**, que define as regras de combinação dessas palavras para formar frases válidas; e **uma semântica**, que determina se uma frase construída faz sentido de fato. No caso das máquinas, a lista de instruções (IL) funciona como o alfabeto do seu idioma, sendo o conjunto básico de símbolos que usamos para dar comandos diretos a um computador — a chamada língua materna da máquina.

No entanto, essa língua é muito distante da linguagem natural que usamos no dia a dia. Por isso, tanto humanos quanto computadores precisam de um meio-termo, um tipo de linguagem que funcione como ponte entre o raciocínio humano e a execução computacional. Assim surgiram os linguagens de programação, que permitem ao programador escrever instruções de forma mais compreensível, enquanto ainda são capazes de ser traduzidas em comandos de máquina.

Esses são os chamados linguagens de programação de alto nível, que usam símbolos, palavras e convenções próximas do idioma humano. Com eles, podemos dar ordens muito mais complexas aos computadores do que seria possível usando apenas a IL. **Um programa escrito em uma linguagem de alto**

nível recebe o nome de código-fonte, e o arquivo onde ele é armazenado é chamado de **arquivo-fonte** — diferente do código de máquina que, de fato, é executado pela máquina.

☰ Compilação vs Interpretação - Vantagens e Desvantagens

	COMPILADORES	INTERPRETADORES
VANTAGENS	▪ A execução do código traduzido costuma ser mais rápida. ▪ Apenas o programador precisa ter o compilador; o usuário final pode usar o código sem ele. ▪ O código traduzido é armazenado em linguagem de máquina e, por ser muito difícil de entender, é provável que suas próprias invenções e truques de programação permaneçam em segredo.	▪ Você pode executar o código assim que o concluir; não há fases adicionais de tradução. ▪ O código é armazenado usando a linguagem de programação, nã a linguagem de máquina; isso significa que ele pode ser executado em computadores que utilizam diferentes linguagens de máquina, sem precisar ser compilado separadamente para cada arquitetura diferente.
DESVANTAGENS	▪ A compilação em si pode levar bastante tempo; pode ser que você não consiga executar seu código imediatamente após qualquer modificação. ▪ Você precisa ter tantos compiladores quanto plataformas de hardware nas quais deseja que seu código seja executado.	▪ Não espere que a interpretação execute seu código em alta velocidade: seu código dividirá a potência da máquina com o interpretador, portanto, não pode ser realmente rápido. ▪ Tanto você quanto o usuário final precisam ter o interpretador para executar o código.

☰ Seu pirmeiro Programa - Olá, Mundo!

Chegou o momento de começar a escrever um código real e funcional em Python — por enquanto, algo bem simples. Como vamos apresentar alguns conceitos e termos fundamentais, os exemplos não serão complexos nem difíceis. Você pode executar o código direto no notebook, se tudo estiver certo, verá a saída ao pressionar as teclas [SHIFT] + [ENTER].

Agora vamos dedicar um momento para explicar o que você está vendo e por que é assim. O primeiro programa é formado por partes importantes: a palavra **print**, um parêntese de abertura, uma aspa, a frase Olá, Mundo!, outra aspa e o parêntese de fechamento. Cada elemento cumpre um papel essencial na estrutura do código e juntos permitem que o Python saiba exatamente o que exibir na tela.

```
1 print("Olá, Mundo!")
```

```
Olá, Mundo!
```

☰ A Função print()

```
print("Olá, Mundo!")
```

1) Definição e Uso

A função print() exibe a mensagem especificada na tela ou em outro dispositivo de saída padrão.

A mensagem pode ser uma string ou qualquer outro objeto; o objeto será convertido em uma string antes de ser exibido na tela.

2) Sintaxe

```
print(object(s), sep=separator, end=end, file=file, flush=flush)
```

3) Valores dos Parâmetros

Parâmetro	Descrição
`object(s)`	Qualquer objeto, e quantos quiser. Serão convertidos em string antes de serem exibidos
`sep='separator'`	Opcional. Especifica como separar os objetos, se houver mais de um. O padrão é ' '
`end='end'`	Opcional. Especifica o que imprimir no final. O padrão é '\n' (quebra de linha)
`file`	Opcional. Um objeto com método de escrita. O padrão é sys.stdout
`flush`	Opcional. Um Booleano que define se a saída é liberada imediatamente (True) ou armazenada em buffer (False). O padrão é False

```
# Exemplo de separador personalizado para listar ações de sustentabilidade
print("Reflorestamento", "Energia Solar", "Educação Ambiental", sep=" - ")
```

```
Reflorestamento - Energia Solar - Educação Ambiental
```

```
# Imprimir duas informações na mesma linha
print("Projeto:", end=" ")
print("Proteção dos Rios Amazônicos")
```

```
Projeto: Proteção dos Rios Amazônicos
```

```
import time

# Simulação de leitura de dados com flush
for dado in ["Monitorando CO2", "Analisando Biodiversidade", "Gerando Relatório"]:
    print(dado, end=" | ", flush=True)
    time.sleep(1)
```

```
Monitorando CO2 | Analisando Biodiversidade | Gerando Relatório |
```

☰ Desafio usando a função print()

🌍Desafio ODS

Você faz parte de um projeto educacional que quer conscientizar os alunos sobre os ODS. Crie um programa em Python que imprima o seguinte relatório, com a mesma estrutura:

```
Relatório ODS - Consumo Responsável

Meta: Garantir padrões de produção e de consumo sustentáveis.

Dica: Reduza, Reutilize e Recicle sempre que possível.
```

Regras

• Use apenas a função print.

•Utilize quebras de linha (\n) se precisar.

• Mantenha a formatação exatamente igual: título, meta e dica, cada parte separada.

```
# Coloque aqui a sua resposta do seu desafio
```

☰ Variáveis

Variáveis são contêineres para armazenar valores de dados.

Criando Variáveis

O Python não possui um comando para declarar uma variável. Uma variável é criada no momento em que você atribui um valor a ela pela primeira vez.

Exemplo

```
# declarando as variáveis
nr_ods = 17
ods_13 = "Ação Contra a Mudança Global do Clima"
# imprimindo as variáveis
print("Número de ODS:", nr_ods)
print(f"O que trata o ODS 13: {ods_13}")
```

Número de ODS: 17

O que trata o ODS 13: Ação Contra a Mudança Global do Clima

Variáveis não precisam ser declaradas com nenhum tipo específico e podem até mesmo mudar de tipo depois de serem definidas.

Exemplo

```
# declarando as variáveis
x = 17 # x é do tipo inteiro
x = "Ação Contra a Mudança Global do Clima" # x agora é uma string
# imprimindo as variáveis
print("Valor de x:", x)
```

Valor de x: Ação Contra a Mudança Global do Clima

Casting

Se você quiser especificar o tipo de dados de uma variável, isso pode ser feito com conversão.

Exemplo

```
# declarando as variáveis
nr_ods = str(17) # nr_ods é uma string '17'
nr_ods = int(17) # nr_ods é um inteiro 17
nr_ods = float(17) # nr_ods é um po 17
```

```
# imprimindo as variáveis
print("Número de ODS:", nr_ods)
```

Número de ODS: 17.0

```
# Exemplo 1
# declarando as variáveis
nr_ods = 17
ods_13 = "Ação Contra a Mudança Global do Clima"
# imprimindo as variáveis
print("Número de ODS:", nr_ods)
print(f"O que trata o ODS 13: {ods_13}")
```

```
Número de ODS: 17
O que trata o ODS 13: Ação Contra a Mudança Global do Clima
```

```
# Exemplo 2
# declarando as variáveis
x = 17 # x é do tipo inteiro
x = "Ação Contra a Mudança Global do Clima" # x agora é uma string
# imprimindo as variáveis
print("Valor de x:", x)
```

```
Valor de x: Ação Contra a Mudança Global do Clima
```

```
# Exemplo 3
# declarando as variáveis
nr_ods = str(17) # nr_ods é uma string '17'
nr_ods = int(17) # nr_ods é um inteiro 17
nr_ods = float(17) # nr_ods é um po 17
# imprimindo as variáveis
print("Número de ODS:", nr_ods)
```

```
Número de ODS: 17.0
```

Obtendo o tipo de dado de uma variável

Você pode obter o tipo de dados de uma variável com a função **type()**.

Exemplo

```
# declarando as variáveis
nr_ods = 17
ods_13 = "Ação Contra a Mudança Global do Clima"
# imprimindo as variáveis
print("Tipo de dado da variável nr_ods:", type(nr_ods))
print(f"Tipo de dado da variável ods_13: {type(ods_13)}")
```

Tipo de dado da variável nr_ods: int

Tipo de dado da variável ods_13: str

```
# declarando as variáveis
nr_ods = 17
ods_13 = "Ação Contra a Mudança Global do Clima"
# imprimindo as variáveis
print("Tipo de dado da variável nr_ods:", type(nr_ods))
print(f"Tipo de dado da variável ods_13: {type(ods_13)}")
```

```
Tipo de dado da variável nr_ods: <class 'int'>
Tipo de dado da variável ods_13: <class 'str'>
```

Aspas simples ou duplas?

Variáveis de string podem ser declaradas usando aspas simples ou duplas:

Exemplo

```
ods_13 = "Ação Contra a Mudança Global do Clima"
# é a mesma coisa
ods_13 = 'Ação Contra a Mudança Global do Clima'
```

☰ Desafio usando Variáveis

🌱Desafio: Monitoramento de Área Queimada

Crie um programa para reportar o progresso do monitoramento de queimadas:

1. Armazene em duas variáveis:

• area_total (float): área total monitorada, por exemplo, 1500.75 hectares.

• area_queimada (float): área queimada até agora, por exemplo, 123.45 hectares.

2. Calcule a porcentagem de área queimada (use casting para mostrar em inteiro).

3. Use print para exibir um relatório assim:

```
Relatório de Queimadas:
Área total monitorada: 1500.75 ha
Área queimada até agora: 123.45 ha
Porcentagem de área queimada: 8%

"Atenção": é urgente redobrar os esforços!
```

```
1 # Digite a sua solução aqui
2
```

☰ Nome de Variáveis

Uma variável pode ter um nome curto (como x e y) ou um nome mais descritivo (como idade, nr_ods, nome_completo)

Regras para nomes de variáveis em Python:

• O nome deve começar com uma letra ou um underscore (_)

• Não pode começar com número

• Pode conter apenas caracteres alfanuméricos e underscores (A–Z, 0–9, e _)

• Sensível a maiúsculas e minúsculas (areaQueimada, areaqueimada e AREAQUEIMADA são variáveis diferentes)

• Não pode ser uma palavra reservada do Python (como if, for, class)

```
# nome de variáveis permitidas
minhavariavel = "Celso"
minha_variavel = "Celso"
_minha_variavel = "Celso"
minhaVariavel = "Celso"
MINHAVARIAVEL = "Celso"
minhavariavel2 = "Celso"
```

```
# nome de variáveis não permitidas
2minhavariavel = "Celso"
minha-variavel = "Celso"
minha variavel = "Celso"
```

observação: O PEP 8, que é o guia oficial de estilo para Python, recomenda o uso do snake_case para nomes de variáveis, funções e métodos. **Exemplo**: area_total, calcula_media(), tempo_de_execucao (snake_case: minúsculas + underscore).

☰ Vários valores para múltiplas variáveis

O Python permite que você atribua valores a várias variáveis em uma única linha.

```
#Exemplo de vários valores para múltiplas variáveis
reservatorio_1, reservatorio_2, reservatorio_3 = "Limpo", "Monitorado", "Em Tratamento"

print(reservatorio_1)
print(reservatorio_2)
print(reservatorio_3)
```

```
Limpo
Monitorado
Em Tratamento
```

Atenção : Certifique-se de que o número de variáveis corresponde ao número de valores, caso contrário, você receberá um erro.

☰ Um valor para múltiplas variáveis

Você pode atribuir o mesmo valor a várias variáveis em uma linha.

```
# Exemplo de um  valor para múltiplas variáveis
area_paragominas = area_maraba = area_tailandia = "Reflorestamento Ativo"

print(area_paragominas)
print(area_maraba)
print(area_tailandia)
```

```
Reflorestamento Ativo
Reflorestamento Ativo
Reflorestamento Ativo
```

☰ Desempacotar uma Coleção

Se você tiver uma coleção de valores em uma lista, tupla, etc., o Python permite extrair esses valores para variáveis. Isso é chamado de desempacotamento

```
# Exemplo de desempacotamento
reservatorios = ["Limpo", "Monitorado", "Em Tratamento"]

reservatorio_1, reservatorio_2, reservatorio_3 = reservatorios

print(reservatorio_1)
print(reservatorio_2)
print(reservatorio_3)
```

```
Limpo
Monitorado
Em Tratamento
```

☰ Tipos de Dados em Python

Em programação, tipo de dado é um conceito muito importante. Variáveis podem armazenar dados de diferentes tipos, e cada tipo permite realizar operações específicas e adequadas à sua natureza.

O Python possui vários tipos de dados integrados por padrão, organizados em categorias como números, texto, listas, tuplas, conjuntos, dicionários, booleanos e tipos especiais. Esses tipos facilitam o armazenamento, a manipulação e o processamento de informações de forma flexível e eficiente.

Tipo de Texto:

- `str` (string) → usado para textos.

Tipos Numéricos:

- `int` (inteiro) → números inteiros.
- `float` (ponto flutuante) → números decimais.
- `complex` (complexo) → números complexos.

Tipos de Sequência:

- `list` (lista) → sequência ordenada e mutável.
- `tuple` (tupla) → sequência ordenada e imutável.
- `range` (intervalo) → sequência de números, normalmente usada em loops.

Tipo de Mapeamento:

- `dict` (dicionário) → coleção de pares chave-valor.

Tipos de Conjunto:

- `set` (conjunto) → coleção não ordenada de valores únicos.
- `frozenset` (conjunto imutável) → igual ao set, mas imutável.

Tipo Booleano:

- `bool` (booleano) → valores `True` ou `False`.

Tipos Binários:

- `bytes` (bytes) → sequência imutável de bytes.

- `bytearray` (array de bytes) → sequência mutável de bytes.
- `memoryview` (visão de memória) → acessa internamente os dados de outros objetos binários sem copiá-los.

Tipo Nulo:

- `NoneType` → representa a ausência de valor ou um valor nulo (`None`).

```
# Texto
texto = "Reflorestamento no Pará"
print(texto)
print(type(texto))
```

```
Reflorestamento no Pará
<class 'str'>
```

```
# Número inteiro
num_arvores = 1500
print(num_arvores)
print(type(num_arvores))
```

```
1500
<class 'int'>
```

```
# Número decimal
area_hectares = 350.75
print(area_hectares)
print(type(area_hectares))
```

```
350.75
<class 'float'>
```

```
# Lista
municipios = ["Paragominas", "Marabá", "Tailândia"]
print(municipios)
print(type(municipios))
```

```
['Paragominas', 'Marabá', 'Tailândia']
<class 'list'>
```

```
# Booleano
meta_atingida = True
print(meta_atingida)
print(type(meta_atingida))
```

```
True
<class 'bool'>
```

☰ Número Aleatório em Python

O Python não possui uma função random() para gerar um número aleatório, mas possui um módulo embutido chamado random que pode ser usado para criar números aleatórios.

```
import random

# Gera um número aleatório de árvores plantadas em um projeto de reflorestamento
arvores_plantadas = random.randrange(1000, 5000)

# Gera um número aleatório de hectares recuperados
hectares_recuperados = random.uniform(10.5, 50.75)

# Faz o cast para inteiro das árvores e arredonda os hectares para 2 casas decimais
arvores_plantadas = int(arvores_plantadas)
```

```
hectares_recuperados = round(hectares_recuperados, 2)

# Imprime o relatório usando print com 'end' para manter na mesma linha
print("Projeto de Reflorestamento no Pará:", end=" ")
print("Árvores plantadas:", arvores_plantadas, end=" | ")
print("Hectares recuperados:", hectares_recuperados)
```

```
Projeto de Reflorestamento no Pará: Árvores plantadas: 1923 | Hectares recuperados: 19.34
```

☰ Strings em Python

Strings multilinha

Você pode usar três aspas duplas ou três aspas simples.

```
mensagem = """Projeto de Reflorestamento no Pará:
- Plantio de espécies nativas
- Recuperação de áreas degradadas
- Monitoramento sustentável"""

print(mensagem)
```

```
Projeto de Reflorestamento no Pará:
- Plantio de espécies nativas
- Recuperação de áreas degradadas
- Monitoramento sustentável
```

Strings são matrizes

Assim como em muitas outras linguagens de programação populares, as strings em Python são arrays de bytes que representam caracteres Unicode. No entanto, o Python não possui um tipo de dado específico para caractere; um único caractere é, na verdade, apenas uma string com comprimento igual a 1. É possível usar colchetes para acessar elementos individuais dessa string.

```
# Exemplo: Acessando um caractere específico em uma string

# Declarando uma string
mensagem = "COP30!"

# Usando colchetes [] para acessar o segundo caractere da string (posição 1)
print(mensagem[1])  # Saída: 'O'
```

```
O
```

Percorrendo uma String

Como as strings são arrays, podemos percorrer os caracteres de uma string usando um laço **for**.

```
# Exemplo: Percorrendo uma string

for letra in "Energia Solar":
    print(letra, end=",")
```

```
E,n,e,r,g,i,a, ,S,o,l,a,r,
```

Comprimento da String

Para obter o comprimento de uma string, use a função **len()**.

```
fonte_energia = "Hidrelétrica"

# Usando len() para descobrir o comprimento da string
comprimento = len(fonte_energia)

print("A palavra:", fonte_energia)
print("Comprimento da string:", comprimento)
```

```
A palavra: Hidrelétrica
Comprimento da string: 12
```

Verificar String

Para verificar se uma determinada frase ou caractere está presente em uma string, podemos usar a palavra-chave **in**.

```
# Verifica se a palavra Clima está presente no textp
ods_texto = "ODS 13: Ação Contra a Mudança Global do Clima"

# Verificar se a palavra 'Clima' está na string
print("A palavra Clima está presente no texto:", "Clima" in ods_texto)
```

```
A palavra Clima está presente no texto: True
```

```
# Verifica se a palavra Clima não está presente no textp
ods_texto = "ODS 13: Ação Contra a Mudança Global do Clima"

# Verificar se a palavra 'Clima' não está na string
print("A palavra Clima não está presente no texto:", "Clima" not in ods_texto)
```

```
A palavra Clima não está presente no texto: False
```

Slicing

Você pode retornar um intervalo de caracteres usando a sintaxe de slice (fatiamento). Especifique o índice inicial e o índice final, separados por dois-pontos, para retornar uma parte da string.

```
#Fatiar do índice 0 ao 6
texto = "Energia Solar Renovável"
parte = texto[0:6]
print(parte)  # Saída: 'Energi'
```

```
Energi
```

```
# Fatiar do índice 8 até o final
texto = "Reflorestamento Sustentável"
parte = texto[8:]
print(parte)  # Saída: 'stamento Sustentável'
```

```
tamento Sustentável
```

```
# Fatiar usando índices negativos
texto = "Mudança Climática"
parte = texto[-9:-1]
print(parte)  # Saída: 'Climátic'
```

```
Climátic
```

```
# Fatiar com passo
texto = "Desenvolvimento Sustentável"
parte = texto[0:12:2]
print(parte)  # Saída: 'Dsvlvmet'
```

☰ Modificar Strings em Python

O Python tem um conjunto de métodos integrados que você pode usar em strings.

.upper()

Deixa todos os caracteres em maiúsculo.

```
texto = "energia limpa"
print(texto.upper())  # 'ENERGIA LIMPA'
```

```
ENERGIA LIMPA
```

.lower()

Deixa todos os caracteres em minúsculo.

```
texto = "SUSTENTABILIDADE"
print(texto.lower())  # 'sustentabilidade'
```

```
sustentabilidade
```

.capitalize()

Coloca a primeira letra em maiúsculo e o resto em minúsculo.

```
texto = "desenvolvimento sustentável"
print(texto.capitalize())  # 'Desenvolvimento sustentável'
```

```
Desenvolvimento sustentável
```

.title()

Coloca a primeira letra de cada palavra em maiúsculo.

```
texto = "recursos naturais renováveis"
print(texto.title())  # 'Recursos Naturais Renováveis'
```

```
Recursos Naturais Renováveis
```

.strip()

Remove espaços em branco do início e do final.

```
texto = "   economia verde   "
print(texto.strip())  # 'economia verde'
```

```
economia verde
```

.replace()

Substitui um trecho por outro.

```
texto = "energia fóssil"
print(texto.replace("fóssil", "renovável"))  # 'energia renovável'
```

```
energia renovável
```

.split()

Divide a string em uma lista, usando o separador indicado.

```
texto = "solar,eólica,biomassa"
print(texto.split(","))  # ['solar', 'eólica', 'biomassa']
```

```
['solar', 'eólica', 'biomassa']
```

.join()

Une uma lista de strings em uma única string.

```
fontes = ["solar", "eólica", "biomassa"]
print(", ".join(fontes))  # 'solar, eólica, biomassa'
```

```
solar, eólica, biomassa
```

.find()

Retorna o índice onde uma substring aparece (ou -1 se não encontrar).

```
texto = "mudança climática"
print(texto.find("climática"))  # 8
```

```
8
```

.startswith() / .endswith()

Verifica se a string começa ou termina com certo trecho.

```
texto = "reciclagem inteligente"
print(texto.startswith("rec"))  # True
print(texto.endswith("ente"))   # True
```

```
True
True
```

☰ Caractere de escape

Para inserir caracteres que seriam inválidos dentro de uma string em Python, usamos um caractere de escape. O caractere de escape é uma barra invertida (****) seguida do caractere que se deseja inserir. Um exemplo de caractere inválido é uma aspa dupla dentro de uma string delimitada por aspas duplas — nesse caso, usamos o caractere de escape para que o Python interprete corretamente o símbolo dentro da string.

```
# Exemplo de string com aspas duplas dentro de aspas duplas usando caractere de escape

mensagem = "O instrutor disse: \"Pratique Python todos os dias!\""
print(mensagem)
```

```
O instrutor disse: "Pratique Python todos os dias!"
```

☰ Valores Booleanos no Python

Os booleanos representam um de dois valores: **True** ou **False**.

Valores Booleanos

Em programação, muitas vezes é necessário saber se uma expressão é verdadeira ou falsa.

No Python, você pode avaliar qualquer expressão e obter uma de duas respostas: True (Verdadeiro) ou False (Falso).

Quando você compara dois valores, a expressão é avaliada e o Python retorna o resultado booleano dessa comparação.

```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

```
True
False
False
```

A função bool() permite que você avalie qualquer valor e devolva True ou False.

```
print(bool("Hello"))
print(bool(15))
```

```
True
True
```

☰ DESAFIO ENCONTRO 8

🌱 Monitoramento Sustentável de Consumo no Pará

A preservação dos recursos naturais é uma das grandes prioridades para o Pará, que abriga extensas áreas de floresta, rios e comunidades que dependem diretamente do equilíbrio ambiental. À medida que o Estado avança em políticas de sustentabilidade, surge também a necessidade de pequenas automações locais, que ajudem a organizar informações básicas sobre consumo de água, energia e reflorestamento.

Você, como aprendiz de programação em Python, faz parte dessa mudança: o conhecimento que você adquiriu — variáveis, tipos de dados, strings, impressão formatada, slicing, métodos de string, comparação de valores e uso do print() — pode ser aplicado de forma simples, porém útil, para melhorar a organização e monitorar ações ligadas à sustentabilidade.

Imagine que, até hoje, uma pequena associação comunitária mantinha os dados de consumo de água anotados em cadernos, de forma desorganizada. Agora, com Python, é possível criar relatórios simples que

mostram claramente quanto cada setor consome, ajudam a definir metas de redução e alertam quando o consumo ultrapassa um limite. Essa é uma forma de aplicar tecnologia de forma acessível, ajudando diretamente a comunidade e, em maior escala, o Estado do Pará a avançar nos compromissos com os ODS (Objetivos de Desenvolvimento Sustentável).

Observações: Este é um desafio de prática! Não tem valor de nota, não precisa entregar para ninguém, serve apenas para você testar seu conhecimento e ganhar confiança em resolver problemas reais usando Python de forma aplicada à sustentabilidade.

```
# DESAFIO: Preencha as partes em branco para criar um relatório simples de consumo de água.

# Crie as variáveis com os nomes dos setores
setor_residencial = "Residencial"
setor_agricola = "Agrícola"
setor_industrial = "Industrial"
```