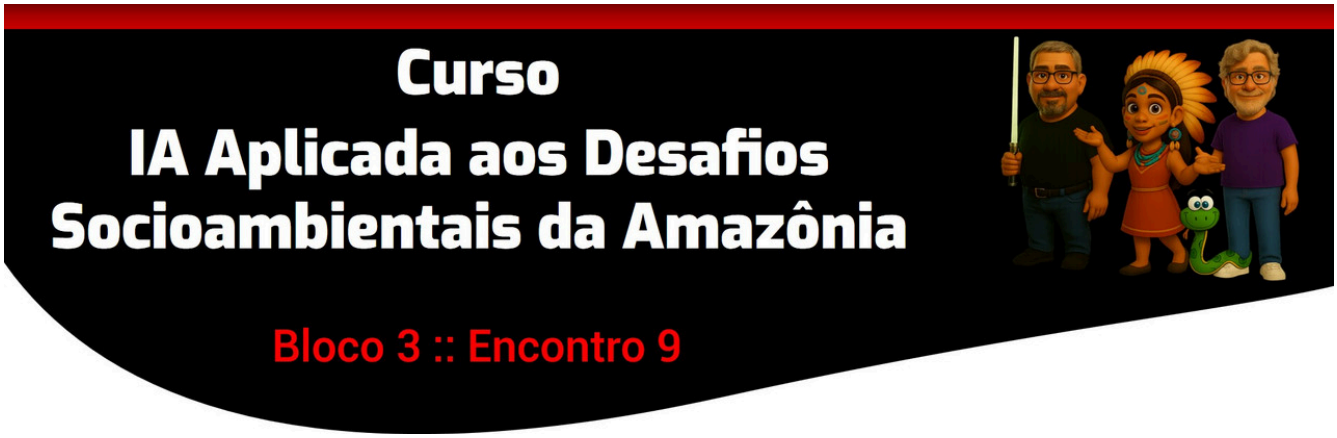
Mostrar código



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

☰ Trilha do Conhecimento

• input

• Operadores

• if..else

• while Loops

• for Loops

• match-case

☰ input()

A função **input()** em Python é utilizada para receber dados digitados pelo usuário durante a execução do programa, permitindo que o código seja mais interativo e dinâmico. Com ela, é possível solicitar informações como textos, números ou opções, exibindo uma mensagem de orientação na tela. **O valor capturado pelo input() é sempre uma string**, mas pode ser convertido para outros tipos de dados, como inteiro ou decimal, usando funções como int() ou float(). Essa funcionalidade é essencial para criar scripts que coletam informações em tempo real, como cadastros, somas ou alertas baseados em dados fornecidos na hora.

```
# Exemplo 1
# Pergunta ao usuário qual estado deseja monitorar
territorio = input("Informe o Estado que será monitorado para buscar áreas queimadas (exemplo: 'DF

# Exibe uma mensagem com a resposta
print("Você escolheu monitorar o território:", territorio)
```

```
Informe o Estado que será monitorado para buscar áreas queimadas (exemplo: 'DF'): DF
Você escolheu monitorar o território: DF
```

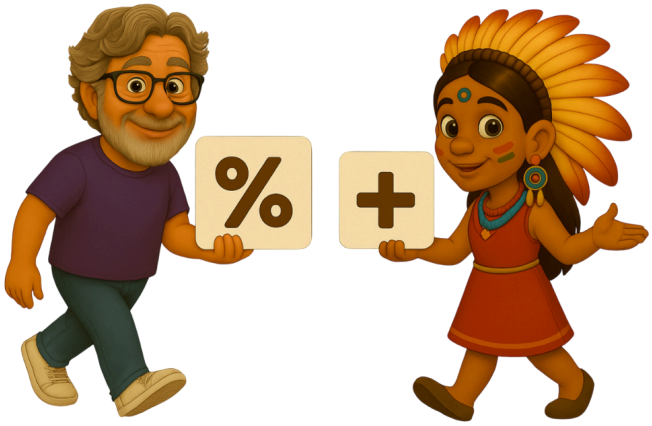
```
# Exemplo 2
# Solicita ao usuário a área de floresta monitorada
area = input("Digite a área monitorada (em hectares): ")

# Converte a entrada para número inteiro
area_int = int(area)

# Mostra a área informada
print("A área monitorada é:", area_int, "hectares")
```

```
Digite a área monitorada (em hectares): 45
A área monitorada é: 45 hectares
```

☰ Operadores

Em Python, **operadores** são símbolos ou palavras-chave que permitem **realizar operações sobre variáveis e valores**, tornando possível executar cálculos, comparações, atribuições, verificações de identidade, filiação em coleções e decisões lógicas. São fundamentais para escrever instruções que manipulem dados, como somar áreas monitoradas, comparar limites de desmatamento, validar condições de alerta ou combinar

expressões em regras de decisão. Dominá-los é essencial para criar códigos funcionais, claros e eficientes, principalmente quando aplicados em cenários reais, como análise de dados ambientais.



```
# Exemplo
area_queimada_para = 20 #m2
area_queimada_amazonas = 40 #m2
print("Área total de queimadas na região norte:", area_queimada_para+area_queimada_amazonas, "m2")

```

```
Área total de queimadas na região norte: 60 m2
```

Python divide os operadores nos seguintes grupos:

- **Operadores Aritméticos**: soma, subtração, multiplicação, divisão, resto, potência.
- **Operadores de Atribuição**: =, +=, -=, *=, /=, etc.
- **Operadores de Comparação**: ==, !=, >, <, >=, <=.
- **Operadores Lógicos**: and, or, not.
- **Operadores de Identidade**: is, is not.
- **Operadores de Filiação**: in, not in.

= Operadores Aritméticos

Operador	Nome	Exemplo
+	Adição	x + y
-	Subtração	x - y
*	Multiplicação	x * y
/	Divisão	x / y
%	Módulo	x % y
**	Exponenciação	x ** y
//	Divisão inteira	x // y

```
# Multiplicação: calcular área total de reflorestamento
# 10 hectares por lote * 5 lotes
area_lote = 10
numero_lotes = 5
area_total = area_lote * numero_lotes
print("Área total de reflorestamento:", area_total, "hectares")
```

```
Área total de reflorestamento: 50 hectares
```

```
# Módulo: verificar se uma meta de plantio é múltipla de 100 mudas
mudas_plantadas = 450
resto = mudas_plantadas % 100
print("Resto da divisão por 100:", resto, "- Se for 0, meta é múltipla de 100.")
```

```
Resto da divisão por 100: 50 - Se for 0, meta é múltipla de 100.
```

```
# Exponenciação: calcular crescimento de área reflorestada dobrando a cada ano
# 2 hectares que dobram por 3 anos
area_inicial = 2
anos = 3
area_após_anos = area_inicial * (2 ** anos)
print("Área após", anos, "anos de crescimento exponencial:", area_após_anos, "hectares")
```

```
Área após 3 anos de crescimento exponencial: 16 hectares
```

```
# Divisão inteira: repartir mudas igualmente entre comunidades
total_mudas = 550
comunidades = 4
mudas_por_comunidade = total_mudas // comunidades
print("Cada comunidade receberá:", mudas_por_comunidade, "mudas (divisão inteira)")
```

```
Cada comunidade receberá: 137 mudas (divisão inteira)
```

= Operadores de Atribuição

Operadores de atribuição são usados para atribuir valores a variáveis:

Operador	Exemplo	Equivalente
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3
:=	print(x := 3)	x = 3; print(x)

```
# Soma acumulada: adicionar árvores plantadas
arvores_plantadas = 50
arvores_plantadas += 10  # equivale a: arvores_plantadas = arvores_plantadas + 10

print("Total de árvores plantadas:", arvores_plantadas)
```

```
Total de árvores plantadas: 60
```

```
# Verificar se a quantidade de mudas cabe em pacotes de 8
mudas = 45
mudas %= 8  # equivale a: mudas = mudas % 8

print("Mudas restantes que não formam pacote de 8:", mudas)
```

```
Mudas restantes que não formam pacote de 8: 5
```

```
# Operação bitwise: simulação de redução de área em unidades binárias
# (exemplo didático, não prático para área real)
x = 16  # valor binário: 10000
x >>= 3  # equivale a: x = x >> 3

print("Resultado do shift à direita:", x)
```

```
Resultado do shift à direita: 2
```

= Operadores de Comparação

Operadores de comparação são usados para comparar dois valores:

Operador	Nome	Exemplo
==	Igual	x == y
!=	Diferente	x != y
>	Maior que	x > y
<	Menor que	x < y
>=	Maior ou igual a	x >= y
<=	Menor ou igual a	x <= y

```
#Verificar se a área desmatada atingiu o limite permitido
area_desmatada = 120
limite_permitido = 100

# Verifica se a área desmatada é maior que o limite
print(area_desmatada > limite_permitido)  # Saída: True
```

```
True
```

```
#Comparar se duas regiões têm o mesmo número de árvores plantadas
arvores_regiao1 = 500
arvores_regiao2 = 500

# Verifica se as duas regiões têm o mesmo valor
print(arvores_regiao1 == arvores_regiao2)  # Saída: True
```

```
True
```

= Operadores Lógicos

Operadores lógicos são usados para combinar instruções condicionais:

Operador	Descrição	Exemplo
and	Retorna True se ambas as condições forem verdadeiras	x < 5 and x < 10
or	Retorna True se pelo menos uma condição for verdadeira	x < 5 or x < 4
not	Inverte o resultado, retorna False se o resultado for True	not(x < 5 and x < 10)

```
# Verifica se a área está entre 50 e 100 hectares
area = 70

resultado = area > 50 and area < 100
print("Área dentro da faixa ideal?", resultado)
```

```
Área dentro da faixa ideal? True
```

```
# Verifica se a área é crítica (muito pequena ou muito grande)
area = 5

resultado_alerta = area < 10 or area > 200
print("Área crítica?", resultado_alerta)

# Usando not para inverter a condição
resultado_invertido = not(area == 50)
print("Área NÃO é exatamente 50 hectares?", resultado_invertido)
```

```
Área crítica? True
Área NÃO é exatamente 50 hectares? True
```

= Operadores de Identidade

Operadores de identidade são usados para comparar os objetos, não se eles são iguais, mas se eles são realmente o mesmo objeto, com o mesmo local de memória:

Operador	Descrição	Exemplo
is	Retorna True se ambas as variáveis forem o mesmo objeto	x is y
is not	Retorna True se ambas as variáveis não forem o mesmo objeto	x is not y

```
# Variável com valor None
agua = None

# Usando 'is' para verificar se é None
print(agua is None)  # True

# Usando 'is not' para verificar se não é None
print(agua is not None)  # False
```

```
True
False
```

= Operadores de Associação

Operadores de associação são usados para testar se uma sequência é apresentada em um objeto:

Operador	Descrição	Exemplo
in	Retorna True se um valor especificado estiver presente no objeto	x in y
not in	Retorna True se um valor especificado não estiver presente no objeto	x not in y

```
# Exemplo de uso do operador de filiação (in e not in) no contexto ambiental
monitoramento = """
Na floresta amazônica
Habita a onça pintada.
Rios cortam a mata densa,
Guardam vida preservada.
"""

print('onça pintada' in monitoramento)
# resultado: True

print('desmatamento' not in monitoramento)
# resultado: True
```

```
True
True
```

= Precedência do Operador

Operator precedence describes the order in which operations are performed.

Operador	Descrição
()	Parênteses
**	Exponenciação
+x, -x, ~x	Mais unário, menos unário e NOT bit a bit
* / // %	Multiplicação, divisão, divisão inteira e módulo
+ -	Adição e subtração
<< >>	Deslocamentos à esquerda e à direita (bitwise)
&	AND bit a bit
^	XOR bit a bit
|	OR bit a bit
== != > < >= <= is is not in not in	Operadores de comparação, identidade e filiação
not	NOT lógico
and	AND lógico
or	OR lógico

```
# Sem parênteses: multiplicação antes da soma
resultado1 = 2 + 3 * 4  # 3 * 4 = 12, depois 2 + 12 = 14

print("Resultado sem parênteses:", resultado1)
```

```
Resultado sem parênteses: 14
```

```
# Com parênteses: soma antes da multiplicação
resultado2 = (2 + 3) * 4  # (2 + 3) = 5, depois 5 * 4 = 20

print("Resultado com parênteses:", resultado2)
```

```
Resultado com parênteses: 20
```

☰ if elif else

A estrutura **if..elif..else** em Python é um dos principais blocos de **controle de fluxo**, permitindo que o programa **tome decisões** com base em condições. Com o if, testamos se uma expressão é **verdadeira**; se for, um bloco de código é executado. Já o else define o que fazer quando essa condição **não é atendida**, garantindo que sempre haja uma ação alternativa. Esse recurso é essencial para criar scripts dinâmicos, capazes de responder a diferentes situações, como gerar alertas, validar dados ou definir caminhos distintos no processamento de informações — algo fundamental, por exemplo, no monitoramento ambiental e na análise de dados reais.

```
if condicao1:
    # Bloco de código se condicao1 for verdadeira
    pass

elif condicao2:
    # Bloco de código se condicao1 for falsa E condicao2 for verdadeira
    pass

else:
```

```
    # Bloco de código se todas as condições anteriores forem falsas
    pass
```

• O **if** verifica a primeira condição.

• O **elif** (pode ter vários) testa outras opções.

• O **else** é o "caminho padrão", quando nada mais for verdadeiro.

= Cenário 1 (if else)

Uma comunidade quer saber se uma área de preservação permanente ainda está protegida ou se já houve desmatamento.

```
# Verificar se uma área de preservação está protegida ou desmatada

area_desmatada = 0  # valor em hectares

if area_desmatada == 0:
    print("Área protegida.")
else:
    print("Atenção: área com desmatamento!")
```

```
Área protegida.
```

= Cenário 2 (if...elif...else)

Uma ONG ambiental monitora o nível de poluição de um rio.

```
# Monitorar nível de poluição de um rio

nivel_poluição = 75  # valor fictício

if nivel_poluição < 50:
    print("Qualidade da água: Boa")
elif nivel_poluição <= 100:
    print("Qualidade da água: Moderada")
else:
    print("Qualidade da água: Ruim")
```

```
Qualidade da água: Moderada
```

= Cenário 3 (if...elif...else)

Um órgão fiscalizador analisa dados de queimadas.

```
# Analisar risco de queimadas considerando área e umidade

area_queimada = 120  # em hectares
umidade = 25  # em porcentagem

if area_queimada > 100 and umidade < 30:
    print("🚨 Risco extremo de queimadas!")
elif area_queimada > 100 and umidade >= 30:
    print("⚠️ Risco alto de queimadas.")
```

```
else:
    print("✅ Situação controlada.")
```

```
🚨 Risco extremo de queimadas!
```

= Cenário 4 (if...elif...else)

Uma equipe monitora um parque para decidir se libera visitação.

```
# Verificar se o parque pode ser visitado usando if aninhado

parque_aberto = True
previsao_chuva = False

if parque_aberto:
    print("Parque está aberto.")
    if previsao_chuva:
        print("Previsão de chuva. Visita adiada.")
    else:
        print("Tempo bom. Visita liberada!")
else:
    print("Parque fechado. Visita cancelada.")
```

```
Parque está aberto.
Tempo bom. Visita liberada!
```

☰ while

Um loop while em Python é usado quando precisamos **repetir a execução de um bloco de código** continuamente, **enquanto uma condição for verdadeira**. Esse laço de repetição avalia a condição antes de cada ciclo e só para quando ela se torna falsa, permitindo criar repetições controladas por variáveis ou entradas do usuário.

```
# sintaxe básica de um while loop
while condição:
    # executa um conjunto de instruções repetidamente enquanto a condição testada for verdadeira
```

• while avalia a **condição** antes de cada repetição.

• Se for **True**, executa o bloco indentado.

• Assim que for **False**, o loop termina.

= Cenário 1

Uma ONG precisa monitorar o plantio de mudas em tempo real. Para isso, o sistema exibe uma árvore 🌳 a cada muda plantada, simulando o progresso até completar dez mudas. Essa visualização ajuda a registrar a meta de reflorestamento de forma simples, interativa e motivadora.

```
# Exemplo cenário 1
import time
```

```
mudas = 0
print("Plantando mudas:")
while mudas < 10:
    print("🌳", flush= True, end= " ")
    mudas += 1
    time.sleep(1)
print(f"\nTotal de mudas plantadas: {mudas}")
```

```
Plantando mudas:
🌳 🌳 🌳 🌳 🌳 🌳 🌳 🌳 🌳 🌳
Total de mudas plantadas: 10
```

= Cenário 2

Uma equipe de monitoramento precisa emitir alertas enquanto houver risco de queimadas na região. Para isso, o sistema repete o aviso de alerta três vezes, simulando uma checagem contínua até o risco ser controlado, ajudando a comunidade a ficar informada e a agir rapidamente quando necessário.

```
# Exemplo cenário 2
risco = True
contador = 0

while risco:
    print("🚨 Alerta de risco de queimadas!")
    contador += 1
    if contador == 3:
        risco = False
```

```
🚨 Alerta de risco de queimadas!
🚨 Alerta de risco de queimadas!
🚨 Alerta de risco de queimadas!
```

= Cenário 3

Uma equipe de fiscalização ambiental precisa vistoriar cinco regiões de preservação. Para organizar esse trabalho, será usado um sistema que percorre automaticamente cada área numerada, registrando a sequência de fiscalização, garantindo que nenhuma região fique sem acompanhamento e facilitando o controle de monitoramento em campo.

```
# Exemplo cenário 3
regiao = 1

while regiao <= 5:
    print("Fiscalizando região:", regiao)
    regiao += 1
```

```
Fiscalizando região: 1
Fiscalizando região: 2
Fiscalizando região: 3
Fiscalizando região: 4
Fiscalizando região: 5
```

= Cenário 4

Uma associação comunitária quer resolver a falta de organização no controle de dados ambientais da região. Para isso, será criado um Sistema de Monitoramento Ambiental com um menu interativo, permitindo que moradores e gestores consultem de forma prática o status de reflorestamento, queimadas, qualidade da água e alertas essenciais.

```
# Exemplo cenário 4
opcao = 0 #criado a variável para entrar na primeira iteração

while opcao != 5:
    print("\n🌿 ==== MENU DE MONITORAMENTO AMBIENTAL ====")
    print("1. Ver status de reflorestamento 🌳")
    print("2. Monitorar queimadas 🔥")
    print("3. Consultar qualidade da água 💧")
    print("4. Ver alertas gerais ⚠️")
    print("5. Sair 🚪")
    print("=========================================\n")

    opcao = int(input("Digite a opção desejada (1 a 5): "))

    if opcao == 1:
        print("Você selecionou: Status de reflorestamento 🌳")
    elif opcao == 2:
        print("Você selecionou: Monitorar queimadas 🔥")
    elif opcao == 3:
        print("Você selecionou: Consultar qualidade da água 💧")
    elif opcao == 4:
        print("Você selecionou: Alertas gerais ⚠️")
    elif opcao == 5:
        print("Saindo do sistema... Você saiu do sistema com sucesso. 👋")
    else:
        print("Opção inválida. Tente novamente.")
```

```
🌿 ==== MENU DE MONITORAMENTO AMBIENTAL ====
1. Ver status de reflorestamento 🌳
2. Monitorar queimadas 🔥
3. Consultar qualidade da água 💧
4. Ver alertas gerais ⚠️
5. Sair 🚪
=========================================

Digite a opção desejada (1 a 5): 3
Você selecionou: Consultar qualidade da água 💧

🌿 ==== MENU DE MONITORAMENTO AMBIENTAL ====
1. Ver status de reflorestamento 🌳
2. Monitorar queimadas 🔥
3. Consultar qualidade da água 💧
4. Ver alertas gerais ⚠️
5. Sair 🚪
=========================================

Digite a opção desejada (1 a 5): 5
Saindo do sistema... Você saiu do sistema com sucesso. 👋
```

☰ for Loops

Um laço for em Python é usado para percorrer **itens de uma sequência** (como listas, tuplas ou strings), executando um bloco de código para **cada elemento**, de forma organizada e automática. É ideal para situações em que já sabemos **quantas vezes** o loop precisa acontecer

```
for item in sequência:
    # Bloco de código que será executado para cada item
    pass
```

• for percorre cada elemento de uma **sequência** (lista, string, range, etc.).

• O **bloco indentado** roda uma vez para **cada item**.

• **item** é uma **variável temporária** que recebe cada valor.

= Cenário 1

Uma comunidade quer exibir o nome de cinco árvores nativas plantadas em uma área de reflorestamento.

```
arvores = ["Castanheira", "Ipê", "Açaí", "Andiroba", "Copaíba"]

for arvore in arvores:
    print("Árvore plantada:", arvore)
```

```
Árvore plantada: Castanheira
Árvore plantada: Ipê
Árvore plantada: Açaí
Árvore plantada: Andiroba
Árvore plantada: Copaíba
```

= Cenário 2

Durante uma inspeção, ao encontrar uma área degradada, a vistoria para imediatamente.

```
areas = ["Área 1", "Área 2", "Área Degradada", "Área 3"]

for area in areas:
    if area == "Área Degradada":
        print("⚠️ Área degradada encontrada! Interrompendo inspeção.")
        break
    print("Inspecionando:", area)
```

```
Inspecionando: Área 1
Inspecionando: Área 2
⚠️ Área degradada encontrada! Interrompendo inspeção.
```

= Cenário 3

Um monitoramento percorre áreas, mas ignora áreas que já foram reflorestadas.

```
areas = ["Área 1", "Área Reflorestada", "Área 2", "Área Reflorestada", "Área 3"]

for area in areas:
    if area == "Área Reflorestada":
        continue
    print("Área em análise:", area)
```

```
Área em análise: Área 1
Área em análise: Área 2
Área em análise: Área 3
```

= Cenário 4

Contar 5 lotes de mudas sendo plantados, numerando cada um.

```
for lote in range(1, 6):
    print(f"Lote {lote} de mudas plantado.")
```

```
Lote 1 de mudas plantado.
Lote 2 de mudas plantado.
Lote 3 de mudas plantado.
Lote 4 de mudas plantado.
Lote 5 de mudas plantado.
```

= Cenário 5

Percorrer 3 setores de uma área de conservação e, ao final, exibir mensagem de que toda a vistoria foi concluída.

```
for setor in range(1, 4):
    print(f"Vistoriando setor {setor}...")
else:
    print("Todos os setores foram vistoriados com sucesso!")
```

```
Vistoriando setor 1...
Vistoriando setor 2...
Vistoriando setor 3...
Todos os setores foram vistoriados com sucesso!
```

= Cenário 6

Para cada região, verificar 2 pontos de monitoramento de queimadas.

```
regioes = ["Norte", "Sul"]

for regiao in regioes:
    for ponto in range(1, 3):
        print(f"Monitorando ponto {ponto} da região {regiao}.")
```

```
Monitorando ponto 1 da região Norte.
Monitorando ponto 2 da região Norte.
Monitorando ponto 1 da região Sul.
Monitorando ponto 2 da região Sul.
```

= Cenário 7

Percorrer uma lista de rios monitorados, mas ainda sem ação programada para eles (bloco reservado).

```
rios = ["Rio Amazonas", "Rio Tapajós", "Rio Xingu"]

for rio in rios:
    pass  # Código futuro para análise da qualidade da água
```

Desafio 1 - Relatório de Monitoramento Ambiental

Você, como TecnoGuarda da Amazônia, precisa resolver a desorganização no relatório diário de monitoramento das áreas de queimadas. O sistema deve receber o nome do território, validar se o usuário digitou algo, perguntar quantas áreas foram vistoriadas e exibir, com um for, os setores analisados. Caso algum setor seja identificado como área de risco, o laudo deve parar de registrar (break). Áreas já reflorestadas devem ser ignoradas (continue). Ao final, o sistema mostra uma mensagem de fechamento com else. Para registrar trechos que ainda serão detalhados no futuro, deve usar pass. O processo só termina quando o usuário decidir sair.

```
# Digite aqui o seu código

```

Solução Desafio 1 - Relatório de Monitoramento Ambiental

Mostrar código

☰ match case

Introduzida no Python 3.10, a estrutura match case trouxe uma forma mais clara e poderosa de corresponder padrões em Python. Ela permite criar **verificações condicionais** de forma mais legível e organizada. Diferente das cadeias tradicionais if-elif-else, que podem ficar confusas em situações complexas, o match case torna o código **mais elegante e flexível**.

```
match variável:
    case valor1:
        # Bloco de código se variável for igual a valor1
        pass
    case valor2:
        # Bloco de código se variável for igual a valor2
        pass
    case _:
        # Bloco de código padrão (equivalente ao else)
        pass
```

• match avalia o valor de uma variável.

• case define os padrões que podem ser verificados.

• O case _: funciona como caso padrão, para quando nenhum outro case for atendido.

= Cenário 1

Imagine um sistema que apresenta um menu de opções ambientais para o usuário (por exemplo: 1 para Florestas, 2 para Rios, 3 para Clima). O programa usa match case para exibir uma mensagem correspondente à opção escolhida.

```
opcao = input("Escolha uma opção do menu (1-Florestas, 2-Rios, 3-Clima): ")

# Uso do match-case para verificar a opção escolhida
match opcao:
    case "1":
        print("Você selecionou a opção: Florestas. Aqui estão os dados sobre florestas.")
    case "2":
        print("Você selecionou a opção: Rios. Aqui estão os dados sobre rios.")
    case "3":
        print("Você selecionou a opção: Clima. Aqui estão os dados sobre o clima.")
    case _:
        # Caso padrão: qualquer valor não previsto acima
        print("Opção inválida. Por favor escolha 1, 2 ou 3.")
```

```
Escolha uma opção do menu (1-Florestas, 2-Rios, 3-Clima): 1
Você selecionou a opção: Florestas. Aqui estão os dados sobre florestas.
```

= Cenário 2

Um sistema de alerta ambiental monitora o desmatamento de uma região. Dependendo da categoria do desmatamento (baixo, moderado, alto ou crítico), o programa usa match case para simular uma resposta automática adequada, como uma mensagem de aviso ou de emergência.

```
nivel_desmatamento = "alto"  # Exemplo de categoria atual de desmatamento (baixo, moderado, alto ou

# Resposta automática do sistema de alerta com base no nível de desmatamento
match nivel_desmatamento:
    case "baixo":
        print("Desmatamento baixo. Nenhuma ação necessária.")
    case "moderado":
        print("Desmatamento moderado. Intensificar monitoramento.")
    case "alto":
        print("Desmatamento alto. Atenção: preparar medidas de contenção.")
    case "crítico":
        print("Desmatamento crítico! Alerta máximo: iniciar ações emergenciais.")
    case _:
        # Caso padrão: qualquer valor não previsto acima
        print("Categoria de desmatamento desconhecida. Verificar dados.")

```

```
Desmatamento alto. Atenção: preparar medidas de contenção.
```

= Cenário 3

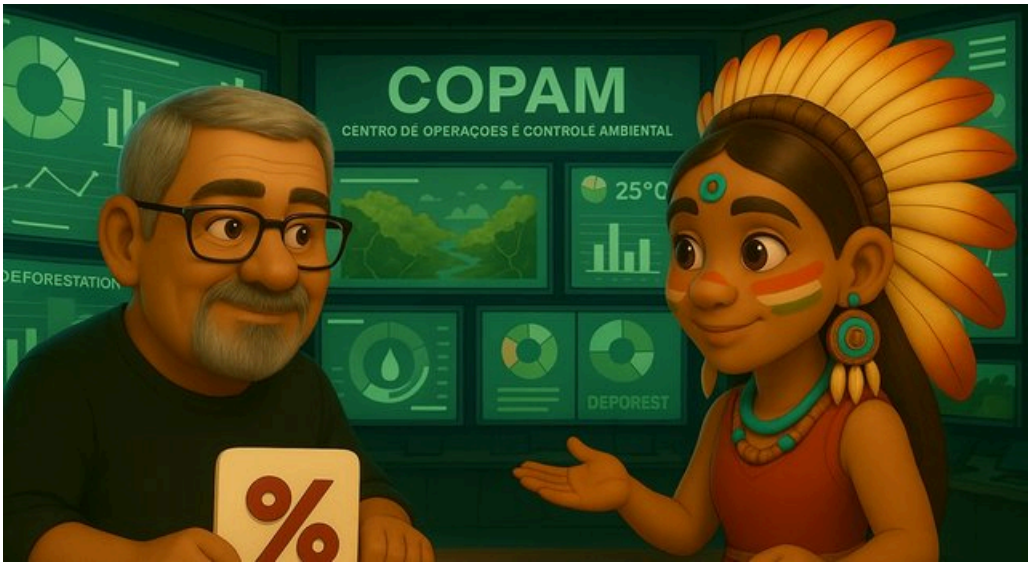
Você é TecnoGuarda e precisa classificar rapidamente áreas monitoradas de acordo com o tipo de vegetação predominante. Para isso, o operador deve digitar o tipo de cobertura do solo, que pode ser “mata”, “floresta”, “cerrado”, “campo” ou “pantanal”. O sistema deve agrupar mata e floresta como Floresta

Densa, cerrado ou campo como Vegetação Aberta, e pantanal como Zona Úmida. Se o operador digitar algo diferente, o sistema mostra “Tipo desconhecido”.

```
# Entrada do operador
tipo_vegetacao = input("Informe o tipo de vegetação (mata, floresta, cerrado, campo, pantanal): ")

# Classificação usando match case com OR
match tipo_vegetacao:
    case "mata" | "floresta":
        print("Classificação: Floresta Densa 🌳")
    case "cerrado" | "campo":
        print("Classificação: Vegetação Aberta 🌾")
    case "pantanal":
        print("Classificação: Zona Úmida 💧")
    case _:
        print("Tipo desconhecido. Verifique a entrada e tente novamente.")
```

```
Informe o tipo de vegetação (mata, floresta, cerrado, campo, pantanal): cerrado
Classificação: Vegetação Aberta 🌾
```

Desafio 2 - COPAM: Monitoramento Integrado de Risco Ambiental



O Centro de Operações de Controle Ambiental (COPAM), órgão estratégico do Ministério do Meio Ambiente, enfrenta um desafio crescente para responder com agilidade a situações de risco ambiental na Amazônia. Hoje, os registros de dados sobre desmatamento, queimadas, qualidade da água e umidade do solo ainda são fragmentados, o que dificulta alertas imediatos e decisões integradas no momento certo.

Para superar essa limitação, o COPAM quer implementar um Sistema Automatizado de Monitoramento Integrado, operado pelo TecnoGuarda, que atuará como chefe de operações. Esse sistema precisa permitir que o operador insira manualmente, em cada ronda, quatro indicadores críticos: Nível de Desmatamento, Áreas Queimadas, Qualidade da Água e Umidade do Solo. Cada indicador será registrado com valores de 0 a 100.

No cálculo do Indicador de Risco Ambiental, os pesos atribuídos devem refletir a prioridade de cada fator. O Nível de Desmatamento e as Áreas Queimadas são os mais críticos e devem ter peso 2 cada. Já a Qualidade da Água e a Umidade do Solo terão peso 1. O índice final será calculado somando os pesos aplicados e dividindo o total pela soma dos pesos, resultando em uma média ponderada.