

Semana 12 - Aula 1

Tópico Principal da Aula: Estruturas de decisão compostas – Aninhamento de estruturas de decisão

Subtítulo/Tema Específico: Introdução ao aninhamento de estruturas de decisão e fluxo de execução.

Código da aula: [SIS]ANO1C1B2S12A1

Objetivos da Aula:

- Conhecer estruturas de decisão compostas por meio do fluxo de execução de programas.
- Compreender exemplos de aninhamentos em estruturas de decisão.

Recursos Adicionais:

- Recursos audiovisuais para exibição de vídeos e imagens.
- Caderno, canetas e lápis.

Exposição do Conteúdo:

Referência do Slide: Slides 03, 04 - Objetivos da Aula, Desenvolvimento da aula

- **Definição:** A aula visa introduzir o conceito de aninhamento de estruturas de decisão, explicando como funcionam e como podem ser usadas para criar programas mais complexos e poderosos. Serão abordados o conceito e a utilidade do aninhamento, a análise do fluxo de execução e a habilidade de escrever programas com essas estruturas.
- **Aprofundamento/Complemento:** O aninhamento é fundamental quando uma decisão depende do resultado de uma decisão anterior, permitindo uma lógica condicional mais granular e específica.

Referência do Slide: Slide 05 - Introdução

- **Definição:** Estruturas de decisão compostas permitem tomar decisões baseadas em múltiplas condições. O aninhamento ocorre quando uma estrutura de decisão (como um bloco `if`, `elif` ou `else`) é colocada dentro de outra. Isso é útil para lidar com múltiplos níveis de condições.
- **Aprofundamento/Complemento:** Aninhar estruturas de decisão significa que o bloco de código interno só será considerado se a condição do bloco externo for satisfeita. Isso cria um caminho de execução condicional hierárquico.

- **Exemplo Prático:** Imagine um sistema de login. Primeiro, verifica-se se o usuário existe (primeira condição). Se existir, uma segunda condição aninhada verifica se a senha está correta.

Referência do Slide: Slide 06 - Exemplo de Aninhamento

- **Definição:** O slide apresenta um exemplo genérico de aninhamento:

```
Python

if condition1:
    if condition2:
        action1
    else:
        action2
else:
    action3
```

Copiar o código

- Neste caso, `condition2` só é verificada se `condition1` for verdadeira. Se ambas forem verdadeiras, `action1` é executada. Se `condition1` for verdadeira e `condition2` for falsa, `action2` é executada. Se `condition1` for falsa, `action3` é executada, independentemente de `condition2`.
- **Aprofundamento/Complemento:** Este exemplo ilustra a dependência sequencial das condições. A estrutura `else` associada a `if condition2` só é relevante se `condition1` for verdadeira. Da mesma forma, `action3` é um caminho alternativo que ignora completamente a lógica de `condition2`.
- **Exemplo**

Prático:

```
Python

idade = 20
possui_habilitacao = True

if idade >= 18:
    print("Maior de idade.")
    if possui_habilitacao:
        print("Pode dirigir.")
    else:
        print("Não pode dirigir legalmente ainda.")
else:
    print("Menor de idade. Não pode dirigir.")
```

Referência do Slide: Slide 07 - Fluxo de execução

- **Definição:** O fluxo de execução em estruturas de decisão aninhadas ocorre de cima para baixo e de dentro para fora. O Python primeiro avalia a condição da estrutura externa. Se verdadeira, ele passa para a estrutura interna e avalia sua condição. Esse fluxo permite decisões complexas baseadas em múltiplos níveis de condições.
- **Aprofundamento/Complemento:** É crucial entender a indentação em Python, pois ela define os blocos de código e, consequentemente, a lógica do aninhamento. Uma indentação incorreta pode levar a erros lógicos ou de sintaxe.
- **Exemplo Prático:** No exemplo do slide 06, se `condition1` for falsa, o interpretador Python ignora completamente o bloco indentado que contém `if condition2`, `action1` e `action2`, e vai diretamente para o bloco `else` correspondente a `condition1` para executar `action3`.

Referência do Slide: Slide 08 - Exemplos de aninhamento (Verificação de número)

- **Definição:** O exemplo pede ao usuário para inserir um número e imprime mensagens diferentes dependendo se o número é positivo e par, positivo e ímpar, negativo e par, negativo e ímpar, ou zero.

```
Python

num = int(input("Digite um número: "))
if num > 0:
    if num % 2 == 0: # Em Python, a sintaxe correta é "num % 2 == 0"
        print("O número é positivo e par.")
    else:
        print("O número é positivo e ímpar.")
elif num < 0:
    if num % 2 == 0: # Em Python, a sintaxe correta é "num % 2 == 0"
        print("O número é negativo e par.")
    else:
        print("O número é negativo e ímpar.")
else:
    print("O número é zero.")
```

- **Aprofundamento/Complemento:** Este código demonstra como `if-elif-else` pode ser usado para a estrutura externa (verificar se o número é positivo, negativo ou zero) e, dentro dos blocos `if` (positivo) e `elif` (negativo), uma estrutura `if-else` aninhada verifica a paridade do número. A condição `num % 2 == 0` verifica se o resto da divisão do número por 2 é igual a 0, o que indica um número par. (Nota: A sintaxe correta em Python para a verificação de

paridade é `num % 2 == 0`, como corrigido no bloco de código acima, e não `num % $2==0$` como no slide)

- **Exemplo Prático:** Se o usuário digitar `10`, a primeira condição (`num > 0`) é verdadeira. A condição aninhada (`num % 2 == 0`) também é verdadeira, então "O número é positivo e par." é impresso. Se o usuário digitar `-7`, a primeira condição é falsa, a segunda (`num < 0`) é verdadeira. A condição aninhada (`num % 2 == 0`) é falsa, então "O número é negativo e ímpar." é impresso.

Referência do Slide: Slides 09, 10 - Exemplos de aninhamento (Descontos em loja)

- **Definição:** Um exemplo de uma loja que oferece descontos com base no tipo de produto e no número de unidades compradas.

```
Python

tipo_produto = input("Insira o tipo de produto (tecnologia/vestuario): ")
quantidade = int(input("Insira a quantidade de produtos: "))

if tipo_produto == "tecnologia":
    if quantidade > 2: # No slide está $>2$, o correto em Python é > 2
        print("Você ganhou um desconto de 10%!")
    else:
        print("Infelizmente, você não atingiu a quantidade necessária para um c
elif tipo_produto == "vestuario":
    if quantidade > 3: # No slide está If quantidade > 3, o correto em Python é
        print("Você ganhou um desconto de 20%!")
    else:
        print("Infelizmente, você não atingiu a quantidade necessária para um c
else:
    print("Tipo de produto não reconhecido.")
```

- (Nota: Foram feitas pequenas correções de sintaxe no código acima para conformidade com Python, como `quantidade > 2` em vez de `quantidade >2` e o `if` minúsculo).
- **Aprofundamento/Complemento:** Este exemplo mostra o uso de `if-elif-else` para a condição externa (tipo de produto) e `if-else` aninhados para a condição interna (quantidade). Isso permite criar regras de desconto específicas para cada combinação de tipo de produto e quantidade.
- **Exemplo Prático:** Se `tipo_produto` for "tecnologia" e `quantidade` for 3, a saída será "Você ganhou um desconto de 10%!". Se `tipo_produto` for "vestuario" e `quantidade` for 2, a saída será "Infelizmente, você não atingiu a quantidade necessária para um desconto.".

Semana 12 - Aula 2

Tópico Principal da Aula: Estruturas de decisão compostas – Aninhamento de estruturas de decisão

Subtítulo/Tema Específico: Aplicações práticas e casos de borda em estruturas aninhadas.

Código da aula: [SIS]ANO1C1B2S12A2

Objetivos da Aula:

- Conhecer estruturas de decisão compostas por meio do fluxo de execução de programas.
- Compreender exemplos de aninhamentos em estruturas de decisão.

Recursos Adicionais:

- Recursos audiovisuais para exibição de vídeos e imagens.
- Caderno, canetas e lápis.

Exposição do Conteúdo:

Referência do Slide: Slides 05, 06 - Situação-problema (Acesso por país)

- **Definição:** Apresenta uma situação-problema onde um departamento de tecnologia precisa desenvolver um sistema em Python que verifique o país do usuário para fornecer acessos específicos à localidade. A tarefa é montar um fluxograma para resolver o problema, identificar o parâmetro para identificar o país e quais funções liberar.
- **Aprofundamento/Complemento:** Esta situação-problema introduz a necessidade de lógica condicional para personalizar a experiência do usuário com base em um critério (localização). Parâmetros para identificar o país podem incluir o endereço IP do usuário, seleção manual pelo usuário, ou configurações de idioma do sistema/navegador. As funções liberadas seriam então condicionadas ao país identificado.
- **Exemplo Prático (Fluxograma - Conceitual):**
 1. **Início**
 2. **Obter Localização do Usuário** (Ex: por IP ou pergunta)
 3. **Decisão: País é "Brasil"?**
 - **Sim:** Liberar Funções A, B, C (Específicas do Brasil)
 - **Não: Decisão: País é "EUA"?**
 - **Sim:** Liberar Funções X, Y, Z (Específicas dos EUA)
 - **Não:** Liberar Funções Padrão / Mostrar mensagem de erro de localização.

4. **Fim** O slide 07 apresenta um modelo de fluxograma genérico para exemplificar a representação visual da lógica.

Referência do Slide: Slides 08, 09 - Aninhamento de estruturas (Exemplo do Guarda-Chuva)

- **Definição:** O aninhamento de estruturas de decisão em Python significa ter uma instrução `if-elif-else` dentro de outra. O exemplo decide se deve usar um guarda-chuva com base em duas variáveis: se está chovendo e se a pessoa possui um guarda-chuva.

```
Python

esta_chovendo = True # No slide, a atribuição é feita com '=' e não '$=' [cite:
possui_guarda_chuva = False # No slide, a atribuição é feita com '=' e não '$='

if esta_chovendo:
    if possui_guarda_chuva:
        print("Está chovendo, mas você tem um guarda-chuva. Pode sair.")
    else:
        print("Está chovendo e você não tem um guarda-chuva. Melhor ficar em ca
else:
    print("Não está chovendo. Pode sair.")
```

- Neste exemplo, a primeira condição verifica se está chovendo. Se sim, uma nova estrutura de decisão aninhada verifica se há um guarda-chuva. (*Nota: Correção na atribuição das variáveis booleanas `esta_chovendo` e `possui_guarda_chuva` para o padrão Python, usando `=` em vez de `$=` como parecia estar no slide.*)
- **Aprofundamento/Complemento:** Este é um exemplo clássico de como uma decisão secundária (usar o guarda-chuva ou ficar em casa) depende do resultado de uma condição primária (estar chovendo).
- **Exemplo Prático:** Se `esta_chovendo` for `True` e `possui_guarda_chuva` for `False`, o programa imprimirá "Está chovendo e você não tem um guarda-chuva. Melhor ficar em casa.". Se `esta_chovendo` for `False`, independentemente do valor de `possui_guarda_chuva`, imprimirá "Não está chovendo. Pode sair.".

Referência do Slide: Slides 10, 11 - Casos de Borda (Elegibilidade para Aposentadoria)

- **Definição:** Casos de borda são situações especiais que ocorrem nos extremos das condições testadas e precisam ser cuidadosamente considerados, especialmente em estruturas aninhadas. O exemplo aborda a idade mínima para aposentadoria (65 anos) e o que acontece quando a pessoa tem exatamente 65 anos.

```
Python

idade = 65
if idade < 65:
    print("Você ainda não é elegível para se aposentar.")
elif idade >= 65: # Operador >= trata o caso de borda de idade exatamente 65
    print("Você é elegível para se aposentar.")
```

- O operador `>=` é usado para tratar o caso de borda em que a idade é exatamente 65.
- **Aprofundamento/Complemento:** Casos de borda são fontes comuns de erros lógicos em programas. Testar com valores que estão nos limites das condições (ex: 64, 65, 66 para a condição `>= 65`) é essencial para garantir que o programa se comporte como esperado. Outros exemplos de casos de borda incluem listas vazias, strings vazias, números zero, valores máximos/mínimos permitidos para um tipo de dado.
- **Exemplo Prático:** Se `idade` for 64, a primeira condição (`idade < 65`) é verdadeira. Se `idade` for 65, a primeira é falsa, e a condição `elif idade >= 65` é verdadeira. Se `idade` for 66, a primeira é falsa, e a condição `elif idade >= 65` é verdadeira.

Referência do Slide: Slides 12, 13, 14 - Exemplo prático (Temperatura e Chuva)

- **Definição:** Um programa que pede a temperatura atual e se está chovendo. Ele informa se está quente (>30 graus), agradável (20-30 graus) ou frio (<20 graus). Dentro de cada condição de temperatura, verifica se está chovendo e aconselha sobre levar guarda-chuva ou ficar em casa. Exemplo de código com aninhamento (Slide 13):

```
Python

temperatura = int(input("Qual é a temperatura atual? "))
esta_chovendo = input("Está chovendo? (sim/não) ") == "sim"

if temperatura > 30:
    print("Está quente.")
    if esta_chovendo: # Aninhado
        print("E também está chovendo. Você deve ficar dentro de casa.")
elif 20 <= temperatura <= 30:
    print("A temperatura está agradável.")
    if esta_chovendo: # Aninhado
        print("Mas está chovendo. Leve um guarda-chuva.")
else: # Menor que 20
    print("Está frio.")
    if esta_chovendo: # Aninhado
        print("E também está chovendo. Fique quente e seco dentro de casa.")
```

- exemplo de código utilizando operador `and` (Slide 14) para reduzir aninhamento explícito:

```
Python

temperatura = int(input("Qual é a temperatura atual? "))
esta_chovendo = input("Está chovendo? (sim/não) ") == "sim"

if temperatura > 30 and esta_chovendo:
    print("Está quente. E também está chovendo. Você deve ficar dentro de casa.")
elif temperatura > 30:
    print("Está quente.")
elif 20 <= temperatura <= 30 and esta_chovendo:
    print("A temperatura está agradável. Mas está chovendo. Leve um guarda-chuva.")
elif 20 <= temperatura <= 30:
    print("A temperatura está agradável.")
elif esta_chovendo: # Implícito que está frio aqui (temperatura < 20)
    print("Está frio. E também está chovendo. Fique quente e seco dentro de casa.")
else: # Frio e não chovendo
    print("Está frio.")
```


- **Aprofundamento/Complemento:** O primeiro código usa aninhamento direto. O segundo código mostra como o operador lógico `and` pode ser usado para combinar condições, o que pode, em alguns casos, reduzir a profundidade do aninhamento e tornar o código mais linear. No entanto, para lógicas muito complexas, o aninhamento explícito pode ser mais fácil de ler e depurar. A escolha entre eles depende da clareza e da complexidade das condições.
- **Exemplo Prático:** Para o código do Slide 13:
 - Se `temperatura = 35` e `esta_chovendo = True`: Imprime "Está quente." e depois "E também está chovendo. Você deve ficar dentro de casa."
 - Se `temperatura = 25` e `esta_chovendo = False`: Imprime "A temperatura está agradável.". Para o código do Slide 14:
 -
 - Se `temperatura = 15` e `esta_chovendo = True`: Imprime "Está frio. E também está chovendo. Fique quente e seco dentro de casa."

Semana 12 - Aula 3

Tópico Principal da Aula: Estruturas de decisão compostas – Aninhamento de estruturas de decisão

Subtítulo/Tema Específico: Aninhamento em diferentes cenários e casos de borda complexos.

Código da aula: [SIS]ANO1C1B2S12A3

Objetivos da Aula:

- Conhecer estruturas de decisão compostas por meio do fluxo de execução de programas.
- Compreender exemplos de aninhamentos em estruturas de decisão.

Recursos Adicionais:

- Recursos audiovisuais para exibição de vídeos e imagens.
- Caderno, canetas e lápis.

Exposição do Conteúdo:

Referência do Slide: Slide 05 - Aninhamento em diferentes cenários (App de Streaming)

- **Definição:** O aninhamento é útil em cenários que requerem várias verificações e condições. O exemplo é um aplicativo de streaming de música que verifica preferências do usuário para sugerir músicas. Primeiro, verifica se o usuário gosta de música clássica. Se sim, verifica se gosta de Beethoven (condição aninhada). Se ambas forem verdadeiras, sugere Beethoven.
- **Aprofundamento/Complemento:** Este cenário demonstra como o aninhamento pode refinar progressivamente uma decisão ou recomendação. Cada nível de aninhamento adiciona uma camada de especificidade.
- **Exemplo Prático (Código do Slide 06):**

```
Python

gosta_de_musica_classica = True
gosta_de_beethoven = True

if gosta_de_musica_classica:
    if gosta_de_beethoven:
        print("Sugerindo uma música de Beethoven!")
    else:
        print("Sugerindo uma música clássica.") # Outro compositor clássico
else:
    print("Sugerindo uma música de um gênero diferente.")
```

- Se `gosta_de_musica_classica` for `True` e `gosta_de_beethoven` for `False`, o app sugere "uma música clássica" (que não seja Beethoven).

Referência do Slide: Slide 07 - Exemplo - Python - Utilizando `and`

- **Definição:** Apresenta uma alternativa ao código anterior usando o operador `and` para combinar condições, o que pode reduzir o aninhamento visual.

```
Python

gosta_de_musica_classica = True
gosta_de_beethoven = True

if gosta_de_musica_classica and gosta_de_beethoven:
    print("Sugerindo uma música de Beethoven!")
elif gosta_de_musica_classica: # Executado se a primeira condição for falsa, mas
    print("Sugerindo uma música clássica.")
else:
    print("Sugerindo uma música de um gênero diferente.")
```

- **Aprofundamento/Complemento:** A estrutura `if/elif/else` com `and` avalia as condições de forma sequencial. Se `gosta_de_musica_classica` `and`

`gosta_de_beethoven` for verdadeiro, o primeiro bloco é executado. Caso contrário, ele verifica `elif gosta_de_musica_classica`. Se apenas esta for verdadeira (implicando que `gosta_de_beethoven` era falso ou a combinação não satisfaz o primeiro `if`), o segundo bloco é executado.

- **Exemplo Prático:** Se `gosta_de_musica_classica = True` e `gosta_de_beethoven = False`:
 - A condição `gosta_de_musica_classica and gosta_de_beethoven` é `True and False`, que é `False`.
 - O programa passa para o `elif gosta_de_musica_classica`. Esta condição é `True`.
 - Portanto, "Sugerindo uma música clássica." é impresso.

Referência do Slide: Slides 08, 09 - Aninhamento de estruturas de decisão (Controle Climático)

- **Definição:** Exemplo de um sistema de controle climático para uma casa. O sistema verifica se é dia ou noite (sensor de luz) e, então, a temperatura para decidir se liga o ar-condicionado, aquecedor ou nenhum. Código:

```
Python

é_dia = True # No slide é_dia = True
temperatura = 22 # No slide temperatura = 22 (comentário Em graus Celsius)

if é_dia:
    if temperatura > 25:
        print("Ligando o ar-condicionado.")
    elif temperatura < 20: # No slide está temperatura < 20
        print("Ligando o aquecedor.")
    else: # Temperatura entre 20 e 25 inclusive
        print("Temperatura confortável. Não fazendo nada.")
else: # é_noite
    print("É noite. Controlador de clima em modo de economia de energia.")
```

- *(Nota: Pequenas correções no código para refletir a sintaxe de atribuição e comparação em Python, como `temperatura < 20`.)*
- **Aprofundamento/Complemento:** Este exemplo mostra uma estrutura `if-else` externa baseada no período do dia, e uma estrutura `if-elif-else` aninhada para controlar o clima com base na temperatura, mas apenas se for dia. Durante a noite, a lógica de temperatura é ignorada, e o sistema entra em modo de economia.
- **Exemplo Prático:** Se `é_dia = True` e `temperatura = 28`, o sistema imprimirá "Ligando o ar-condicionado.". Se `é_dia = False` e `temperatura = 28`, imprimirá "É noite. Controlador de clima em modo de economia de energia.".

Referência do Slide: Slides 10, 11 - Casos de Borda em Cenários Complexos

- **Definição:** Casos de borda são situações nos limites extremos das condições. No exemplo do controle climático, um caso de borda é quando a temperatura é exatamente 25°C durante o dia. O sistema original não faria nada, mas talvez queiramos ligar o ar-condicionado. Casos de borda podem se tornar mais complexos com múltiplas condições e estruturas aninhadas. Código modificado para lidar com o caso de borda (temperatura \geq 25):

```
Python

é_dia = True
temperatura = 25 # Caso de borda

if é_dia:
    if temperatura >= 25: # Modificado para incluir 25 graus
        print("Ligando o ar-condicionado.")
    elif temperatura < 20:
        print("Ligando o aquecedor.")
    else: # Temperatura entre 20 (inclusive) e 24 (inclusive)
        print("Temperatura confortável. Não fazendo nada.")
else:
    print("É noite. Controlador de clima em modo de economia de energia.")
```

- **Aprofundamento/Complemento:** A modificação `temperatura >= 25` altera o comportamento no limite. Antes, 25°C resultaria em "Temperatura confortável". Agora, com `>= 25`, 25°C liga o ar-condicionado. Identificar e testar explicitamente esses casos de borda é crucial para a robustez do software.
- **Exemplo Prático:** Com o código modificado, se `é_dia = True` e `temperatura = 25`, a saída será "Ligando o ar-condicionado.".

Referência do Slide: Slides 12, 13, 14 - Atividade Estudo de Caso (App de Entrega de Alimentos)

- **Definição:** Uma atividade de estudo de caso para programar um app de entrega de alimentos que calcula o custo de entrega com base no valor do pedido e se o usuário é membro de um programa de fidelidade. Condições:
 - Pedido < R\$10: taxa R\$5.
 - Pedido \geq R\$10 e < R\$20: taxa R\$3.
 - Pedido \geq R\$20: entrega gratuita.
 - Membro fidelidade: desconto de R\$2 na taxa aplicável (taxa não pode ser negativa). O objetivo é escrever o algoritmo que calcula e imprime a taxa final. Resolução sugerida (Slide 13):

```
valor_pedido = float(input("Digite o valor do pedido: R$"))
eh_membro_fidelidade = input("É membro do programa de fidelidade? (sim/não): ")
taxa_entrega = 0

# Determinando a taxa de entrega com base no valor do pedido
if valor_pedido < 10:
    taxa_entrega = 5
elif 10 <= valor_pedido < 20: # Pode ser simplificado para elif valor_pedido < 20:
    taxa_entrega = 3
# elif valor_pedido >= 20: # Esta condição é implícita se as anteriores forem verdadeiras
# taxa_entrega = 0 # Já inicializada com 0

# Aplicando desconto para membros de fidelidade
if eh_membro_fidelidade:
    taxa_entrega = max(taxa_entrega - 2, 0) # Evita taxas negativas

print(f"A taxa de entrega é: R${taxa_entrega}")
```

- (Nota: Adicionados comentários sobre simplificações possíveis no `elif` e a inicialização de `taxa_entrega`). A explicação detalhada da resolução é fornecida no Slide 14.
- **Aprofundamento/Complemento:** Este estudo de caso combina múltiplas `if/elif` para determinar a taxa base e, em seguida, uma estrutura `if` separada para aplicar o desconto de fidelidade. A função `max(valor - desconto, 0)` é uma forma elegante de garantir que a taxa de entrega não se torne negativa após o desconto.
- **Exemplo Prático:**
 - `valor_pedido = 8`, `eh_membro_fidelidade = False`: `taxa_entrega_base = 5`. Final = R\$5.
 - `valor_pedido = 15`, `eh_membro_fidelidade = True`: `taxa_entrega_base = 3`. Desconto de 2. Final = R\$1.
 - `valor_pedido = 25`, `eh_membro_fidelidade = True`: `taxa_entrega_base = 0`. Desconto de 2 (`max(0-2,0) = 0`). Final = R\$0.
 - `valor_pedido = 8`, `eh_membro_fidelidade = True`: `taxa_entrega_base = 5`. Desconto de 2. Final = R\$3.

Referência do Slide: Slides 15, 16 - Quiz (Estrutura de Decisão para Chá)

- **Definição:** A situação "Ao preparar um chá, você verifica a temperatura da água. Se a temperatura for igual a 100 graus, você desliga o fogão, senão, você continua a aquecer a água" é um exemplo de estrutura de decisão

If/Else. O termo `if` representa a primeira decisão, enquanto `else` é a consequência. Trata-se de uma estrutura simples de tomada de decisão.

- **Aprofundamento/Complemento:** Esta é a forma mais fundamental de estrutura de decisão, permitindo que um programa execute um de dois blocos de código com base em uma única condição booleana.
- **Exemplo** **Prático:**

```
Python

temperatura_agua = int(input("Digite a temperatura da água: "))
if temperatura_agua == 100:
    print("Desligar o fogão.")
else:
    print("Continuar a aquecer a água.")
```

Semana 12 - Aula 4

Tópico Principal da Aula: Estruturas de decisão compostas – Aninhamento de estruturas de decisão

Subtítulo/Tema Específico: Exemplos práticos, jogo de adivinhação e aplicação em problemas cotidianos.

Código da aula: [SIS]ANO1C1B2S12A4

Objetivos da Aula:

- Conhecer estruturas de decisão compostas por meio do fluxo de execução de programas.
- Compreender exemplos de aninhamentos em estruturas de decisão.

Recursos Adicionais:

- Recursos audiovisuais para exibição de vídeos e imagens.
- Caderno, canetas e lápis.

Exposição do Conteúdo:

Referência do Slide: Slide 05 - Jogo de Adivinhação com `elif`

- **Definição:** Propõe relembrar o conceito do jogo de adivinhação, incluindo a aplicação de estruturas de decisão compostas, especificamente o `elif`. Um vídeo da Alura é referenciado para demonstrar a condição `elif`.

- **Aprofundamento/Complemento:** Em um jogo de adivinhação, o jogador tenta acertar um número secreto. O programa dá dicas como "muito alto" ou "muito baixo". A estrutura `if-elif-else` é perfeita para isso:
 - `if chute == numero_secreto:` (Acertou)
 - `elif chute < numero_secreto:` (Muito baixo)
 - `else:` (Muito alto, já que não é igual nem menor) O `elif` (abreviação de "else if") permite testar múltiplas condições em sequência. Assim que uma condição `if` ou `elif` é verdadeira, seu bloco é executado, e as demais condições `elif` ou `else` no mesmo nível são ignoradas.
- **Exemplo Prático (Conceitual, baseado na ideia do jogo):**

```
Python

numero_secreto = 42
chute = int(input("Adivinhe o número: "))

if chute == numero_secreto:
    print("Parabéns! Você acertou!")
elif chute < numero_secreto:
    print("Muito baixo! Tente um número maior.")
else: # chute > numero_secreto
    print("Muito alto! Tente um número menor.")
```

Referência do Slide: Slides 07, 08 - Exemplos práticos (Planejamento de Viagens)

- **Definição:** Apresenta uma situação-problema de um aplicativo de planejamento de viagens que decide o tipo de transporte com base na distância, orçamento e preferência por rapidez. Código:

```
Python

distancia = float(input("Digite a distância da viagem em km: "))
orcamento = float(input("Digite o seu orçamento em R$: ")) # No slide é R$
prioriza_rapidez = input("Você prioriza rapidez? (sim/não): ").lower() == "sim"

if distancia > 1000:
    if prioriza_rapidez:
        meio_transporte = "Avião"
    else:
        meio_transporte = "Ônibus" if orcamento < 300 else "Avião" # Operador t
elif distancia > 300: # Entre 300.01 e 1000
    if orcamento < 100:
        meio_transporte = "Ônibus"
    else: # orcamento >= 100
        meio_transporte = "Carro" if prioriza_rapidez else "Trem" # Operador te
else: # distancia <= 300
    meio_transporte = "Carro" if orcamento >= 50 else "Bicicleta" # Operador te

print(f"Recomendamos utilizar: {meio_transporte}")
```

- (Nota: Ajustes na formatação do input para `orçamento` e `prioriza_rapidez` para melhor conformidade com Python strings e a função `lower()`).
-
- **Aprofundamento/Complemento:** Este exemplo demonstra um aninhamento mais complexo, onde a decisão principal (`if distancia > 1000, elif distancia > 300, else`) ramifica para outras decisões aninhadas. Notavelmente, ele também utiliza o operador ternário (`valor_se_verdadeiro if condicao else valor_se_falso`) como uma forma concisa de expressar uma decisão `if-else` em uma única linha, o que também é uma forma de estrutura de decisão.
- **Exemplo Prático:**
 - `distancia = 1200, orçamento = 500, prioriza_rapidez = True`: `meio_transporte` será "Avião".
 - `distancia = 1200, orçamento = 200, prioriza_rapidez = False`: `meio_transporte` será "Ônibus".
 - `distancia = 500, orçamento = 50, prioriza_rapidez = True`: `meio_transporte` será "Ônibus".
 - `distancia = 200, orçamento = 100, prioriza_rapidez = False`: `meio_transporte` será "Bicicleta" (erro na lógica original do slide, pois `orçamento >= 50` resultaria em "Carro". Corrigindo para exemplo: `distancia = 200, orçamento = 30 -> "Bicicleta"`). Se `distancia = 200, orçamento = 60`, o resultado é "Carro".

Referência do Slide: Slides 09, 10 - Exemplos práticos (Recomendação de Cursos)

- **Definição:** Um website educacional recomenda cursos com base nos interesses (tecnologia, artes, ciências) e nível de experiência (iniciante, intermediário, avançado) do usuário. Código:

```
Python

interesse = input("Qual é o seu interesse principal? (tecnologia/artes/ciência) ")
experiencia = input("Qual é o seu nível de experiência? (iniciante/intermediário/avançado) ")

curso_recomendado = "" # Inicializar a variável

if interesse == "tecnologia":
    if experiencia == "iniciante":
        curso_recomendado = "Introdução à programação"
    elif experiencia == "intermediario": # No slide está experiencia = "intermediário"
        curso_recomendado = "Desenvolvimento Web"
    else: # avançado
        curso_recomendado = "Inteligência Artificial avançada"
elif interesse == "artes":
    if experiencia == "iniciante":
        curso_recomendado = "Fundamentos do desenho" # No slide falta _recomendado
    elif experiencia == "intermediario":
        curso_recomendado = "Técnicas de pintura"
    else: # avançado
        curso_recomendado = "História da Arte Moderna"
else: # Assume-se ciências se não for tecnologia nem artes
    if experiencia == "iniciante":
        curso_recomendado = "Ciências para todos"
    elif experiencia == "intermediario": # No slide está experiencia = "intermediário"
        curso_recomendado = "Química experimental"
    else: # avançado
        curso_recomendado = "Física teórica"

# No slide está Printf, o correto em Python é print() [cite: 205]
```

- (Nota: Correções na sintaxe de comparação (`==` em vez de `=`), nome da variável `curso_recomendado` e a função `print`).
-
- **Aprofundamento/Complemento:** Este é um exemplo claro de múltiplas camadas de decisão. A primeira camada decide pelo interesse e, dentro de cada interesse, uma segunda camada decide pelo nível de experiência. É importante inicializar `curso_recomendado` para evitar erros caso nenhuma condição seja atendida (embora a lógica aqui cubra todos os casos esperados).
- **Exemplo Prático:**

- interesse = "tecnologia", experiencia = "iniciante": Recomenda "Introdução à programação".
- interesse = "artes", experiencia = "avancado": Recomenda "História da Arte Moderna".

Referência do Slide: Slides 11, 12, 13 - Atividade (Decisão de Modo de Transporte para o Trabalho)

- **Definição:** Uma atividade para criar um esquema ou fluxograma para decidir qual modo de transporte usar para o trabalho. A decisão depende do clima, distância e se há reuniões importantes.
- **Passos para o Fluxograma/Esquema:**
 - **Início:** Avaliar condições do dia.
 - **Verificar Clima:**
 - Chovendo -> Passo 3.
 - Ensolarado -> Passo 4.
 - **Clima Chuvoso:**
 - Distância < 5km: Transporte Público.
 - Distância >= 5km (mais longa): Carro.
 - **Clima Ensolarado:**
 - **Verificar Reuniões Importantes:**
 - Sim (reuniões):
 - Carro ou Transporte Público (dependendo da distância - *aqui a lógica pode precisar de mais detalhe, por ex., se distância for longa, carro, senão transporte público*).
 - Não (sem reuniões):
 - Distância curta: Bicicleta ou A Pé.
 - Distância maior: Transporte Público ou Carro.
 - **Decisão Final:** Escolher o meio de transporte.
 - **Fim.** Um modelo de fluxograma é fornecido no Slide 13 para referência visual.
 -
- **Aprofundamento/Complemento:** Esta atividade reforça a tradução de um problema do cotidiano com múltiplas variáveis em uma estrutura lógica de decisão. A criação de um fluxograma ajuda a visualizar os caminhos de decisão antes de codificar. A complexidade surge nas combinações de condições, como "clima ensolarado E sem reuniões E distância curta".
- **Exemplo Prático (Decisão com base no esquema):**
 - Clima: Chovendo, Distância: 3km. -> Transporte Público.
 - Clima: Ensolarado, Reuniões: Sim, Distância: 10km. -> Carro (assumindo que para reunião e distância longa, carro é preferível).

- Clima: Ensolarado, Reuniões: Não, Distância: 2km. -> Bicicleta ou A Pé.

Semana 13 - Aula 1

Tópico Principal da Aula: Estrutura de seleção: Comandos condicionais: SWITCH

Subtítulo/Tema Específico: Introdução ao comando SWITCH

Código da aula: [SIS]ANO1C1B2S13A1

Objetivos da Aula:

- Compreender as principais diferenças entre a estrutura SWITCH e os comandos tradicionais if-else.

Recursos Adicionais:

- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet.
- Recurso audiovisual para exibição de vídeos e imagens.
- Folhas sulfite, canetas coloridas e lápis.

Exposição do Conteúdo:

Referência do Slide: Slide 04 - Introdução ao comando SWITCH

- **Definição:** O comando SWITCH é fundamental para entender a lógica de programação condicional e serve como base para construir estruturas de decisão mais complexas.

Referência do Slide: Slide 04 - O que é o comando SWITCH e como difere do IF-ELSE

- **Definição:** O comando SWITCH é uma estrutura de controle que seleciona entre múltiplas opções com base no valor de uma variável. É frequentemente usado quando há muitas condições a serem verificadas. O IF-ELSE é uma estrutura condicional básica que executa um bloco de código se uma condição for verdadeira e outro bloco se for falsa.

Simulação de "Switch" (usando Dicionário ou `match`):

- **Clareza para Múltiplas Condições Específicas:** É muito mais legível e conciso quando você tem muitas combinações ou valores específicos para verificar (como no seu jogo de Pedra, Papel e Tesoura, onde cada par de jogadas tem um resultado).

- **Melhor Desempenho (Dicionário):** Para um grande número de condições a serem verificadas, buscar em um dicionário pode ser marginalmente mais eficiente do que uma longa cadeia de `if-elifs`, pois é uma busca direta pela chave.
- **Expressividade (Dicionário):** Permite representar mapeamentos de forma clara, onde uma "entrada" (a chave, como `(computador, jogador)`) corresponde diretamente a uma "saída" (o valor, como `'Empate'`).
- **Padrões Complexos (`match`):** Com o `match` (Python 3.10+), você pode não apenas verificar valores exatos, mas também padrões mais complexos em estruturas de dados (como tuplas, listas, objetos).
- **Ausência de 'Break':** Diferente de outras linguagens, em Python, não há a necessidade de um `break` para sair de um "case" quando se usa dicionários ou `match`, pois eles naturalmente "param" após encontrar uma correspondência.

Comando `if-elif-else`:

- **Flexibilidade para Lógicas Variadas:** É mais flexível para condições que envolvem operadores de comparação (maior que, menor que), ranges ou lógica booleana complexa (`and`, `or`).
- **Ordem de Avaliação:** As condições são avaliadas sequencialmente, de cima para baixo. A primeira condição `True` é executada e o restante é ignorado. Isso pode ser importante para lógicas onde a ordem de verificação importa.
- **Simplicidade para Poucas Condições:** Para um número pequeno de condições, o `if-elif-else` é perfeitamente legível e muitas vezes mais direto do que criar um dicionário ou usar `match`.
- **Compatibilidade Universal:** Funciona em todas as versões do Python, enquanto o `match` é exclusivo do Python 3.10+.

Em resumo:

A principal diferença é que a simulação de "switch" (especialmente com dicionários ou `match`) é ideal para quando você precisa verificar um **valor específico** ou uma **combinação de valores** e associá-los a uma ação/resultado. Ela oferece uma forma mais estruturada e, em muitos casos, mais legível de lidar com múltiplos "casos". Já o `if-elif-else` é mais geral e flexível para **qualquer tipo de condição**, sendo a escolha padrão para a maioria das verificações lógicas em Python.

-

- **Aprofundamento/Complemento:**

- **Vantagens do SWITCH:** Mais limpo e legível do que múltiplos IF-ELSE ao lidar com várias condições.

- **Funcionamento do SWITCH:** Baseia-se no valor de uma expressão ou variável, executando o bloco de código correspondente ao valor (denominado "caso" ou "case").
- **Uso do IF-ELSE:** Mais adequado para situações com menos condições ou quando as condições são complexas e não se baseiam apenas no valor de uma única variável.
- **Diferenças-chave:**
 - **Estrutura:** SWITCH é mais estruturado para múltiplas escolhas baseadas em um único valor de variável. IF-ELSE é mais flexível para avaliar condições variadas e complexas.
 - **Legibilidade:** SWITCH tende a ser mais limpo e legível em casos com muitas condições.
 - **Aplicabilidade:** IF-ELSE pode ser usado em quase qualquer situação, enquanto SWITCH é limitado a cenários de comparação de uma variável com valores constantes.
- **Exemplo Prático:**
 - **SWITCH (simulado em Python):** Um sistema de menu onde o usuário escolhe uma opção (1 para "Ver Saldo", 2 para "Fazer Transferência", 3 para "Pagar Conta") e o programa executa a ação correspondente.
 - **IF-ELSE:** Verificar se a idade de um usuário é maior ou igual a 18 para permitir o acesso a um determinado conteúdo.

Referência do Slide: Slide 04, 08 - Sintaxe básica do SWITCH em diferentes linguagens de programação

- **Definição:** A sintaxe do comando SWITCH varia entre as linguagens de programação. Python não possui um comando SWITCH nativo, mas seu comportamento pode ser simulado com dicionários ou estruturas if-elif-else.
-
- **Aprofundamento/Complemento:** Em linguagens como C#, a estrutura `switch` é seguida por `case` para cada valor possível e, opcionalmente, um `default` para casos não previstos. A instrução `break` é usada para sair do bloco `switch` após a execução de um `case`.
- **Exemplo Prático:**

- **C#:**

```
C#  
  
switch (diaDaSemana) {  
    case 1:  
        Console.WriteLine("Domingo");  
        break;  
    case 2:  
        Console.WriteLine("Segunda-feira");  
        break;  
    default:  
        Console.WriteLine("Dia inválido");  
        break;  
}
```

- **Python (simulando SWITCH com dicionário):**

```
Python  
  
def switch_dia(dia):  
    dias = {  
        1: "Segunda-feira",  
        2: "Terça-feira",  
        3: "Quarta-feira",  
    }  
    return dias.get(dia, "Dia inválido") # [cite: 504]  
  
dia_selecionado = switch_dia(3) # [cite: 504]  
print("O dia é:", dia_selecionado) # [cite: 504]
```

- **Vídeo:**

- O MATEMÁTICO COMPUTACIONAL. Programação em C: Estruturas de Seleção. (Link não fornecido no slide, sugiro pesquisar no YouTube com o título)
- VALE, J. C. S. Estruturas de Seleção em Python – #07. DEV, 2022. (<https://dev.to/jcarlosvale/estruturas-de-selecao-em-python-07-2mac>)

Referência do Slide: Slide 04, 10, 11 - Exemplos simples de uso do SWITCH

- **Definição:** O comando SWITCH (ou sua simulação) é útil em diversas situações práticas no desenvolvimento de software.
- **Exemplos Práticos de Mercado:**
 - **Seleção de configurações:** Em um software de configurações, onde cada opção leva a um conjunto diferente de parâmetros, o SWITCH pode selecionar a configuração apropriada.

-
- **Processamento de comandos:** Em sistemas de comando por voz ou texto (ex: "abrir", "salvar", "editar"), o SWITCH direciona para a função correta.
- **Interfaces de usuário:** Em interfaces gráficas, ao lidar com eventos de diferentes botões ou menus, o SWITCH determina a ação correspondente.
- **Exemplo Prático Detalhado (Automação Residencial):**
 - Um aplicativo de automação residencial que controla dispositivos por comandos de voz ou texto. Cada comando requer uma ação específica para um dispositivo.
 - **Por que usar um método tipo SWITCH (dicionário em Python)?**
 - **Clareza e manutenibilidade:** Comandos e ações são claramente mapeados, tornando o código mais legível e fácil de manter.
 - **Eficiência:** Acesso mais rápido aos comandos comparado a múltiplos if-elif-else, especialmente com muitos comandos.
 - **Flexibilidade para expansão:** Adicionar novos comandos é simples (nova entrada no dicionário) sem modificar longas cadeias de if-elif-else.

- Implementação

em

Python:

Python

```
def acender_luzes():
    return "Luzes acesas"

def ajustar_termostato(temperatura):
    return f"Termostato ajustado para {temperatura} graus"

def tocar_musica(musica):
    return f"Tocando {musica}"

def comando_nao_encontrado():
    return "Comando não reconhecido"

comandos = {
    "acender luzes": acender_luzes,
    "ajustar termostato": ajustar_termostato,
    "tocar música": tocar_musica
}

def processar_comando(comando, *args):
    acao = comandos.get(comando, comando_nao_encontrado) # [cite: 512]
    return acao(*args) # [cite: 512]

print(processar_comando("acender luzes"))
print(processar_comando("ajustar termostato", 22))
print(processar_comando("tocar música", "Beethoven"))
print(processar_comando("abrir portas"))
```

- Neste exemplo, cada comando é mapeado para uma função. Se o comando não existir, uma função padrão é chamada. Isso torna o sistema extensível e de fácil manutenção.
- **Vídeo:**
 - CANAL DO OVIDIO. MVC – ASP.Net MVC na Prática – Aula 1. (Link fornecido no slide como <https://www.youtube.com/watch?v=-ciXTC3JZ9U>, que parece incompleto. Sugiro pesquisar o título no YouTube)

Subtítulo/Tema Específico: SWITCH avançado e casos de uso

Código da aula: [SIS]ANO1C1B2S13A2

Objetivos da Aula:

- Conhecer implementações de utilização do SWITCH em cenários mais complexos de desenvolvimento de software.
-

Recursos Adicionais:

- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet.
- Recurso audiovisual para exibição de vídeos e imagens.
- Folhas sulfite, canetas coloridas e lápis.

Exposição do Conteúdo:

Referência do Slide: Slide 04, 06 - Uso avançado do SWITCH em cenários complexos

- **Definição:** Em Python, um padrão comum para simular um SWITCH avançado é usar um dicionário onde as chaves representam os casos e os valores são funções que executam o código de cada caso.

- Exemplo

Prático:

```
Python

def processar_texto(texto):
    return texto.upper()

def processar_numero(numero):
    return numero * 2

def padrao():
    return "Opção inválida"

switch = {
    "texto": processar_texto,
    "numero": processar_numero
}

entrada = "texto"
valor = "Olá Mundo"
# resultado = switch.get(entrada, padrao)(valor) # O slide tem um erro aqui, de
# Corrigindo para o exemplo funcionar como esperado:
if entrada in switch:
    if entrada == "texto":
        resultado = switch[entrada](valor)
    elif entrada == "numero":
        # Para este exemplo, vamos assumir que 'valor' pode ser um número se 'v
        # Se 'valor' fosse sempre string, precisaríamos de conversão e tratame
        # Exemplo: valor_numerico = 5
        # resultado = switch[entrada](valor_numerico)
        resultado = switch[entrada](len(valor)) # Exemplo usando o tamanho da
else:
    resultado = padrao()

print(resultado) # Saída: OLÁ MUNDO (com a correção)
```

- Vídeo:

- ALURA. Python para Data Science: primeiros passos. Começando com Python. Utilizando o Google Colab. (<https://cursos.alura.com.br/course/python-data-science-primeiros-passos/task/122383>)
-

Referência do Slide: Slide 04, 07 - SWITCH em loops e com tipos de dados variados

- **Definição:** É possível combinar a simulação do SWITCH com loops para processar uma lista de tarefas ou dados de diferentes tipos.

- Exemplo

Prático:

```
Python

# (Reutilizando as funções processar_texto, processar_numero, padrao e o dicionario)
tarefas = [
    ("texto", "Olá"),
    ("numero", 10),
    ("desconhecido", None)
]

for tipo, valor_tarefa in tarefas: # Renomeado 'valor' para 'valor_tarefa' para evitar conflito com o resultado
    # resultado = switch.get(tipo, padrao)(valor_tarefa) # O slide tem um erro aqui
    # Corrigindo:
    funcao_a_executar = switch.get(tipo, padrao)
    if tipo == "numero": # A função processar_numero espera um argumento
        resultado = funcao_a_executar(valor_tarefa)
    elif tipo == "texto": # A função processar_texto espera um argumento
        resultado = funcao_a_executar(valor_tarefa)
    else: # A função padrao não espera argumentos
        resultado = funcao_a_executar()
    print(f"Resultado: {resultado}")
```

- Neste código, uma lista de tarefas é processada, onde cada tarefa tem um tipo e um valor. O SWITCH simulado é usado para determinar a função a ser chamada para cada tipo.
- **Vídeo:**
 - (Sem vídeo específico para este subtópico no slide)

Referência do Slide: Slide 04, 08, 09 - Estudo de caso: aplicando SWITCH em um projeto real

- **Definição:** Em um sistema de processamento de comandos, onde cada comando deve ser executado de forma diferente, a simulação do SWITCH é uma abordagem eficaz.
- **Aprofundamento/Complemento:** Esta técnica é particularmente útil em cenários onde diferentes tipos de processamento são necessários com base em alguma entrada, como em sistemas de comandos ou processamento de dados variados. A abordagem com dicionários e funções em Python, embora não seja um SWITCH nativo, é poderosa e flexível.

- Exemplo Prático (Sistema de Processamento de Comandos):

```
Python

# (Reutilizando a função padrao da seção anterior)
def adicionar_usuario(dados):
    # Lógica para adicionar um usuário
    return f"Usuário {dados['nome']} adicionado."

def remover_usuario(dados):
    # Lógica para remover um usuário
    return f"Usuário {dados['nome']} removido."

comandos = {
    "adicionar": adicionar_usuario,
    "remover": remover_usuario
}

# Exemplo de uso em um sistema
comando_entrada = {"tipo": "adicionar", "dados": {"nome": "João"}}
comando = comando_entrada["tipo"] # [cite: 651]
dados_comando = comando_entrada["dados"] # Renomeado 'dados' para 'dados_comando'

# resultado = comandos.get(comando, padrao)(dados_comando) # Corrigindo para chamar a função
funcao_comando = comandos.get(comando, padrao)
if comando in comandos: # As funções adicionar_usuario e remover_usuario esperam argumentos
    resultado = funcao_comando(dados_comando)
else: # a função padrao não espera argumentos
    resultado = funcao_comando()

print(resultado) # Saída: Usuário João adicionado. [cite: 651]
```

- Dependendo do tipo de comando recebido, uma função específica é chamada, facilitando a extensão e manutenção do código.
- Vídeo:
 - CURSOS KANE CHAN. Estruturas de seleção If e Else em Python. (Link fornecido no slide como <https://www.youtube.com/watch?v=zouf7AkISR4>, que parece incompleto. Sugiro pesquisar o título no YouTube)

Semana 13 - Aula 3

Tópico Principal da Aula: Estrutura de seleção: Comandos condicionais: SWITCH

Subtítulo/Tema Específico: Boas práticas e armadilhas com SWITCH

Código da aula: [SIS]ANO1C1B2S13A3

Objetivos da Aula:

- Compreender as boas práticas durante a aplicação do comando SWITCH (ou sua simulação) no contexto de desenvolvimento de software.

Recursos Adicionais:

- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet.
- Recurso audiovisual para exibição de vídeos e imagens.
- Folhas sulfite, canetas coloridas e lápis.

Exposição do Conteúdo:

Referência do Slide: Slide 04, 05 - Boas práticas no uso do SWITCH

- **Definição:** Em Python, como não há um comando SWITCH nativo, a emulação usando dicionários ou if-elif-else é comum. Seguir boas práticas torna o código mais robusto e legível.
- **Aprofundamento/Complemento (Boas Práticas ao Simular SWITCH com Dicionários):**
 - **Clareza:** Mantenha os nomes das chaves do dicionário e das funções associadas descritivos.
 - **Funções Simples:** Idealmente, as funções associadas aos "casos" devem ser concisas e realizar uma tarefa específica.
 - **Tratamento de Caso Padrão:** Sempre inclua uma forma de lidar com entradas que não correspondem a nenhum caso esperado (equivalente ao `default` em outras linguagens), utilizando o método `.get()` com um valor padrão ou uma função padrão.
- **Exemplo Prático (Cafeteria):**

```
Python

def calcular_preco(cafe):
    precos = {
        "espresso": 1.50,
        "latte": 2.00,
        "cappuccino": 2.25
    }
    return precos.get(cafe, "Café não disponível") # [cite: 749]

preco = calcular_preco("espresso") # [cite: 749]
print(preco) # Saída: 1.5
print(calcular_preco("chá")) # Saída: Café não disponível
```

- **Vídeo:**
 - (Sem vídeo específico para este subtópico no slide)

Referência do Slide: Slide 04, 07 - Armadilhas e erros comuns com SWITCH

- **Definição:** Ao simular o SWITCH em Python, especialmente com dicionários, é importante estar ciente de algumas armadilhas.
- **Aprofundamento/Complemento (Armadilhas Comuns):**
 - **Ausência de valor padrão:** Não fornecer um valor ou ação padrão pode levar a erros (KeyError) se uma chave inesperada for acessada diretamente, ou a um retorno `None` indesejado se `.get()` for usado sem um segundo argumento. É crucial sempre fornecer um valor padrão para lidar com entradas inesperadas.
 - **Uso inadequado para lógica complexa:** Se a lógica dentro de cada "caso" se torna muito extensa ou complexa, o uso de if-elif-else pode ser mais legível e apropriado.
 - **Mutabilidade dos dicionários:** Modificar o dicionário que simula o SWITCH durante a execução do programa (especialmente em contextos concorrentes ou complexos) pode levar a comportamentos inesperados. Tenha cuidado ao fazê-lo.
-
- **Exemplo Prático (Demonstrando ausência de valor padrão):**

```
Python

opcoes = {
    1: "Abrir arquivo",
    2: "Salvar arquivo"
}

# Tentativa de acessar uma chave inexistente sem tratamento:
# print(opcoes[3]) # Isso geraria um KeyError

# Usando .get() sem valor padrão:
print(opcoes.get(3)) # Saída: None

# Usando .get() com valor padrão (boa prática):
print(opcoes.get(3, "Opção inválida")) # Saída: Opção inválida
```

- **Vídeo:**
 - (Sem vídeo específico para este subtópico no slide)

Referência do Slide: Slide 04, 08 - Refatoração de código: melhorando o uso do SWITCH

- **Definição:** Refatorar código que utiliza múltiplas condicionais if-elif-else para uma abordagem de SWITCH simulado com dicionário pode simplificar e tornar o código mais legível, especialmente quando há muitos casos.
- **Exemplo Prático (Aplicar Desconto):**

- **Código Original com if-elif-else:**

```
Python

def aplicar_desconto_original(tipo_cliente, valor):
    if tipo_cliente == "estudante":
        return valor * 0.9
    elif tipo_cliente == "membro":
        return valor * 0.85
    elif tipo_cliente == "vip":
        return valor * 0.8
    else:
        return valor

valor_descontado_original = aplicar_desconto_original("membro", 100)
print(valor_descontado_original) # Saída: 85.0
```

- **Código Refatorado com Dicionário (simulando SWITCH):**

```
Python

def aplicar_desconto_refatorado(tipo_cliente, valor):
    descontos = {
        "estudante": lambda v: v * 0.9,
        "membro": lambda v: v * 0.85,
        "vip": lambda v: v * 0.8
    }
    # A função get retorna a função lambda ou uma lambda padrão que não apl
    # Em seguida, a função retornada é chamada com 'valor'.
    return descontos.get(tipo_cliente, lambda v: v)(valor) # [cite: 753]

valor_descontado_refatorado = aplicar_desconto_refatorado("membro", 100)
print(valor_descontado_refatorado) # Saída: 85.0
print(aplicar_desconto_refatorado("normal", 100)) # Saída: 100 (caso padrão)
```

- **Vídeo:**

- KITAMURA, C. If-Elif-Else – Estrutura de Decisão em Python. (Link fornecido no slide como <https://www.youtube.com/watch?v=3fv05eGgRIA>, que parece incompleto. Sugiro pesquisar o título no YouTube)

Semana 13 - Aula 4

Tópico Principal da Aula: Estrutura de seleção: Comandos condicionais: SWITCH

Subtítulo/Tema Específico: Alternativas ao SWITCH e futuro da programação

Código da aula: [SIS]ANO1C1B2S13A4

Objetivos da Aula:

- Compreender quais são as principais alternativas ao uso do comando SWITCH.


Recursos Adicionais:

- Caderno para anotações;
- Acesso ao laboratório de informática e/ou internet.
- Recurso audiovisual para exibição de vídeos e imagens.
- Folhas sulfite, canetas coloridas e lápis.

Exposição do Conteúdo:

Referência do Slide: Slide 04, 05 - Alternativas ao comando SWITCH (exemplo: Pattern Matching)

- **Definição:** Embora o comando SWITCH seja comum em linguagens como C, C++ e Java, Python não possui um equivalente nativo direto. As alternativas em Python incluem:
 - **Instrução if-elif-else:** A forma mais básica, boa para um número limitado de casos.
 - **Dicionários:** Mapeiam chaves a funções ou valores, úteis para muitos casos e ações complexas.
 - **Pattern Matching (Correspondência de Padrões):** Introduzido no Python 3.10, é um conceito da programação funcional que pode substituir if-else, switch-cases, laços e comparações de tipo.
- **Exemplo Prático:**
 - **If-elif-else:**

```
Python   
  
def exemplo_if_else(valor):  
    if valor == 'A':  
        return "Opção A selecionada"  
    elif valor == 'B':  
        return "Opção B selecionada"  
    else:  
        return "Outra opção"  
print(exemplo_if_else('A')) # Saída: Opção A selecionada [cite: 827]
```


- **Dicionários:**

```
Python

def opcao_a():
    return "Opção A executada"

def opcao_b():
    return "Opção B executada"

switch_dict = {
    'A': opcao_a,
    'B': opcao_b
}

valor = 'A'
# A função get retorna a função (opcao_a) ou uma lambda padrão.
# A função retornada é então chamada.
print(switch_dict.get(valor, lambda: "Opção inválida")()) # Saída: Opção A
```

- **Pattern Matching (Python 3.10+):**

```
Python

def exemplo_pattern_matching(valor):
    match valor:
        case 'A':
            return "Opção A"
        case 'B':
            return "Opção B"
        case _: # O underscore é o caso padrão (wildcard)
            return "Outra opção"
    print(exemplo_pattern_matching('A')) # Saída: Opção A [cite: 827]
```

- **Vídeo:**

- (Sem vídeo específico para este subtópico no slide)

Referência do Slide: Slide 04, 07 - Vantagens e desvantagens dessas alternativas

- **Definição:** Cada alternativa ao SWITCH tem seus prós e contras.
- **Aprofundamento/Complemento:**
 - **If-elif-else:**
 - **Vantagens:** Simples, direto e fácil de entender.
 - **Desvantagens:** Pode se tornar verboso e difícil de manter com muitas condições.
 - **Dicionários:**
 - **Vantagens:** Rápido para um grande número de casos, flexível (permite associação de funções).

- **Desvantagens:** Menos intuitivo para alguns, requer a inicialização do dicionário.
- **Pattern Matching:**
 - **Vantagens:** Muito flexível, permite correspondência complexa de padrões.
 - **Desvantagens:** Mais novo em Python (3.10+), pode ser menos familiar para alguns desenvolvedores.
- **Exemplo Prático:** A escolha entre as alternativas dependerá da complexidade do problema, do número de condições e da versão do Python utilizada. Para uma simples verificação de 2-3 condições, if-elif-else é suficiente. Para um roteamento de ações baseado em strings de comando, dicionários são elegantes. Para desempacotamento e verificação de estruturas de dados complexas, Pattern Matching é poderoso.
- **Vídeo:**
 - (Sem vídeo específico para este subtópico no slide)

Referência do Slide: Slide 04, 08 - Tendências futuras em estruturas condicionais

- **Definição:** A programação condicional continua a evoluir.
- **Aprofundamento/Complemento (Tendências):**
 - **Aumento do uso de Pattern Matching:** Espera-se um uso mais amplo devido à sua flexibilidade e poder.
 -
 - **Integração com Inteligência Artificial:** Estruturas condicionais podem se tornar mais dinâmicas e adaptativas, usando IA para otimizar decisões.
 -
 - **Linguagens Específicas de Domínio (DSLs):** Desenvolvimento de linguagens mais orientadas a casos de uso específicos, podendo incluir estruturas condicionais personalizadas.
 -
- **Exemplo Prático:** Imagine um sistema de diagnóstico médico onde o Pattern Matching analisa sintomas complexos (estrutura de dados) e a IA ajusta a probabilidade de diagnósticos com base em novos dados de pesquisa (tendência de IA adaptativa).
- **Vídeo:**
 - LET'S DATA. If, Elif, Else (Estruturas de Decisão) | Python em 30 minutos. (Link fornecido no slide como <https://www.youtube.com/watch?v=Yvo1IHk3QmA&t=953s>, que parece incompleto. Sugiro pesquisar o título no YouTube)