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-elif`s, 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)

Semana 13 - Aula 2

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

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, d
# 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 '
        # 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 dicior
tarefas = [
    ("texto", "Olá"),
    ("numero", 10),
    ("desconhecido", None)
]

for tipo, valor_tarefa in tarefas: # Renomeado 'valor' para 'valor_tarefa' para
    # resultado = switch.get(tipo, padrao)(valor_tarefa) # O slide tem um erro
    # 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_comand

# resultado = comandos.get(comando, padrao)(dados_comando) # Corrigindo para ch
funcao_comando = comandos.get(comando, padrao)
if comando in comandos: # As funções adicionar_usuario e remover_usuario espera
  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=Yvo1lHk3QmA&t =953s, que parece incompleto. Sugiro pesquisar o título no YouTube)