

Estruturas de Controle: Decisão e Repetição

CATALOGUE

Contents

- Visão geral das estruturas de controle
- Estruturas Condicionais (Tomada de Decisão)
- Operadores na Tomada de Decisão
- Aplicações práticas

Visão geral das estruturas de controle

PART 01

Execução Sequencial

Adaptar às necessidades socioeconômicas

Alinhar-se às estratégias de desenvolvimento nacional e regional para atender às necessidades da vida dos residentes e ao desenvolvimento industrial.

Proteger o ambiente ecológico

Fortalecer a conservação de recursos e a proteção ambiental, melhorando a capacidade de suporte abrangente da cidade.

Disposição racional e desenvolvimento coordenado

Otimizar o layout espacial urbano, fortalecer a delimitação das áreas funcionais urbanas e promover o desenvolvimento regional integrado.



Tomando uma decisão

Estruturas condicionais

Ferramentas como "if-then-else" permitem que o programa avalie condições e execute diferentes blocos de código com base nos resultados.

Decisões múltiplas

A estrutura "switch" facilita o tratamento de múltiplas opções, tornando o código mais organizado e legível em situações complexas.

Flexibilidade lógica

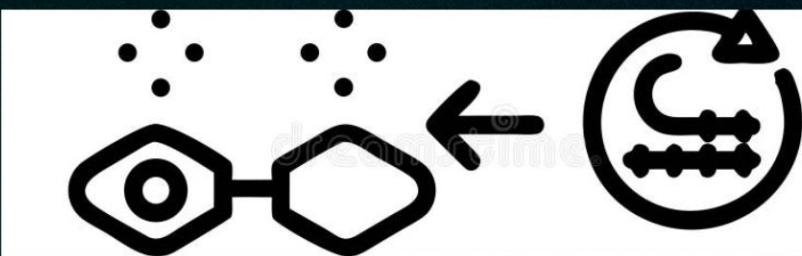
Permite que o programa responda a eventos ou entradas dinâmicas, ajustando sua execução em tempo real.



Mecanismos de Repetição

Repetição eficiente

Estruturas como "for", "while" e "do-while" permitem realizar tarefas recorrentes sem reescrever o código, economizando tempo e esforço.



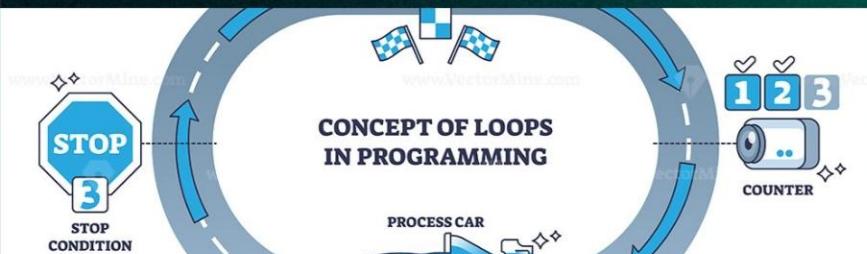
Flexibilidade em algoritmos

Usar repetição é fundamental na criação de processos complexos, como cálculos iterativos e manipulação de grandes volumes de dados.



Controle do fluxo

Condições específicas garantem que os loops não se tornem infinitos, evitando problemas de performance ou falhas no programa.



Estruturas Condicionais (Tomada de Decisão)

PART 02

A instrução "se"

Condição simples

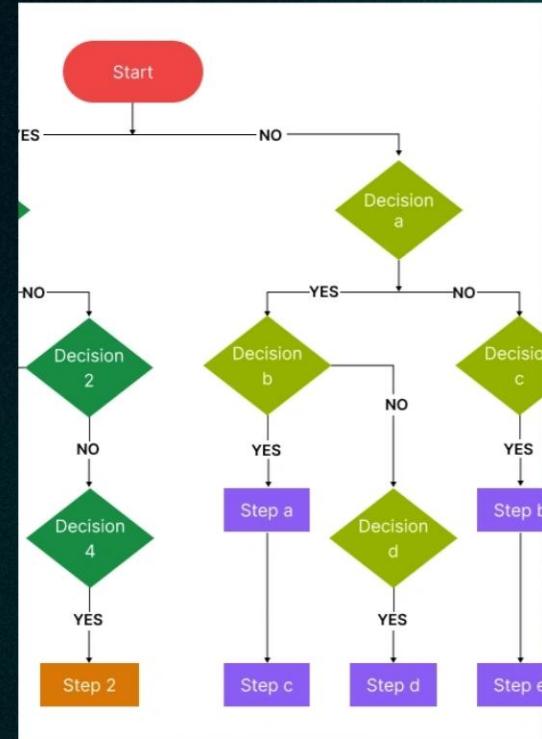
Verifica uma única condição e executa o bloco de código correspondente se o resultado for verdadeiro. Exemplo: `if x > 0:
print("Número positivo")`.

Versatilidade

Facilita a execução de ações específicas, como verificar limites, validar entradas ou iniciar processos quando certo critério é atendido.

Estrutura básica

A simplicidade da sintaxe permite fácil implementação em sistemas que exigem decisões binárias rápidas.



A Cláusula Else

Resposta alternativa



Executa um bloco de código específico caso a condição na instrução `if` seja falsa. Exemplo: `if idade < 18:
print("Menor de idade") else: print("Maior de idade")`.

Complementaridade



Garante que toda possível situação seja tratada, evitando falhas ou saída inválida do programa.

Senso de fluxo



Contribui para a lógica contínua, certificando que o código permaneça funcional mesmo quando a condição inicial falha.

Declaração de Elif

1

Permite verificar várias condições consecutivamente, sem necessidade de aninhar instruções `if`. Exemplo: `if nota >= 90: print("Aprovado com louvor") elif nota >= 60: print("Aprovado") else: print("Reprovado")`.

2

Melhor legibilidade

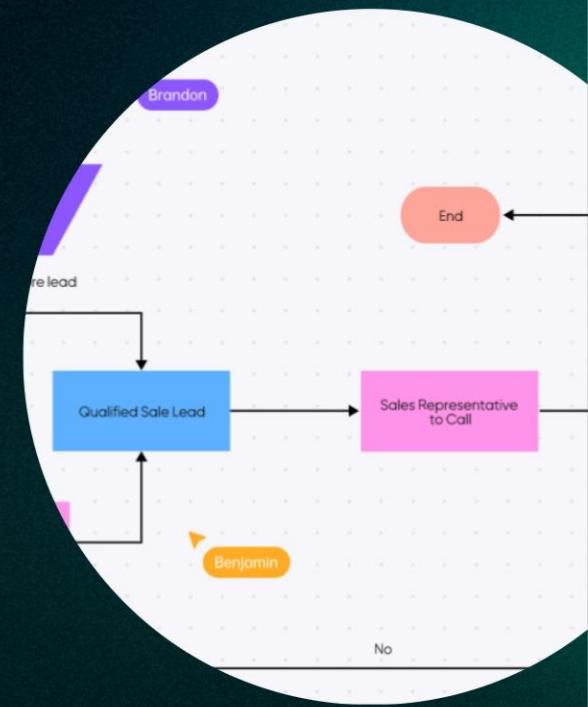
Reduz complexidade e simplifica a interpretação do código ao eliminar excesso de instruções condicionais.

3

Permite criar lógica ordenada onde cada condição é avaliada sequencialmente, garantindo eficiência no fluxo do programa.

Múltiplas condições

Hierarquia de testes



Operadores na Tomada de Decisão

PART 03

Operadores Lógicos

Conjunção

01

O operador AND (`&&`) retorna verdadeiro quando todas as condições avaliadas são verdadeiras, permitindo combinações lógicas mais precisas.

Disjunção

02

O operador OR (`||`) retorna verdadeiro quando pelo menos uma das condições avaliadas é verdadeira, ajudando a ampliar possibilidades de decisão.

Negação

03

O operador NOT (`!`) inverte o estado lógico de uma condição, tornando falso o que era verdadeiro e vice-versa, ideal para verificações complementares.

Operadores Aritméticos

Adição/Subtração

Os operadores + e - permitem somar ou subtrair valores em condições que envolvem cálculos como resultados acumulados.



Multiplicação/Divisão

Os operadores * e / ajudam a calcular proporções e relações numéricas para uso em lógicas condicionais.



Módulo

O operador % retorna o resto de uma divisão entre dois números, sendo frequentemente utilizado para testar divisibilidade ou ciclos em lógica.



Aplicações práticas

PART 04

Operadores Relacionais

Igualdade

O operador == compara se dois valores são exatamente iguais, sendo útil para validações diretas.

01

02

Diferença

O operador != verifica se dois valores são diferentes, indicando contraste entre as variáveis ou entradas.

Maior/Menor

> e < determinam se um valor é maior ou menor que outro, essenciais para decisões baseadas em hierarquias numéricas ou ordens.

03

04

Limites

>= e <= avaliam condições de desigualdade com limites inclusivos, garantindo abrangência ao verificar intervalos.

Tratamento de erros



Validação

Implementar estruturas condicionais para verificar entradas inválidas ou fora do esperado, garantindo que os dados atendam a critérios definidos.



Mensagens de erro

Usar instruções "if-else" para fornecer retornos claros e direcionar o usuário ao corrigir erros, como formatos incorretos de dados.



Exceções

Utilizar controle condicional para lidar com exceções inesperadas no programa e evitar falhas críticas na execução.

Processamento de dados

01.

Iteração

Empregar loops "for" ou "while" para percorrer conjuntos de dados, como análise de listas ou tabelas.

02.

Filtros

Usar lógica condicional dentro de loops para selecionar elementos específicos de grandes conjuntos de dados.

03.

Automação

Aplicar estruturas repetitivas para transformar ou organizar dados repetidamente até que uma condição seja atendida.



Desenvolvimento de Jogos



Decisões dos jogadores

Utilizar lógica condicional para promover respostas diferentes de acordo com ações tomadas pelos jogadores no ambiente do jogo.



Ciclos de repetição

Empregar loops para atualizar os movimentos, rodadas ou fases do jogo continuamente até o objetivo ser alcançado.



Cenários dinâmicos

Incorporar condicionais para criar variações de resultados (como vitória ou derrota), adaptando interações conforme escolhas feitas.

Obrigado