SENAI – Serviço Nacional de Aprendizagem Industrial

Diego Rufino, Gabriell Rodrigues, Hellen de Jesus e Mariane Bertrile

**FUNÇÃO SWITCH CASE EM JAVA SCRIPT**

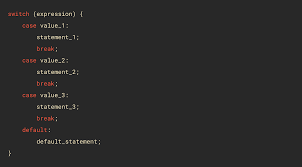
Uma Análise Detalhada

Ferraz de Vasconcelos

2025

1. **O que é um switch-case?**

O **switch-case** é uma estrutura de controle de fluxo muito utilizada em linguagens de programação para gerenciar múltiplas condições de forma organizada e eficiente. Seu principal objetivo é avaliar o valor de uma variável ou expressão e executar diferentes blocos de código com base nesse valor. É uma alternativa ao uso repetitivo de estruturas **if-else**, especialmente quando há várias opções a serem verificadas, tornando o código mais claro e fácil de manter.

O funcionamento básico do switch-case começa com uma variável ou expressão que é avaliada. Cada possível valor dessa variável é representado por um "case", que contém o código a ser executado caso o valor corresponda ao esperado. Além disso, existe a opção de um "default", que é executado caso nenhum dos casos definidos seja atendido. Essa estrutura evita repetições desnecessárias, como várias verificações de igualdade que ocorrem em if-else, e organiza o código em um formato mais linear.

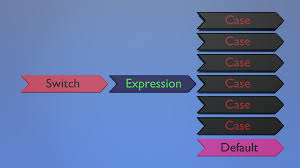
Uma das principais vantagens de utilizar o switch-case está na sua legibilidade. Quando há diversas condições que dependem de um único valor, ele apresenta uma sintaxe limpa e lógica, facilitando a compreensão do fluxo do programa. No entanto, é importante lembrar que o switch-case tem suas limitações: ele normalmente não é capaz de lidar com intervalos ou condições complexas, sendo restrito a verificações de igualdade em muitos casos.

Com suporte em várias linguagens, como C, Java, JavaScript e outras, o switch-case se adapta bem a situações comuns como criação de menus interativos, mapeamento de opções ou ações baseadas em comandos específicos. Seu uso é indicado sempre que se deseja organizar de maneira eficiente em múltiplas escolhas que dependem de um único valor ou parâmetro. Essa estrutura continua sendo uma ferramenta fundamental para programadores que desejam criar códigos mais organizados e eficientes.

1. **Funcionamento básico**

O funcionamento básico do **switch-case** é bastante simples e eficiente, oferecendo uma alternativa clara e organizada para lidar com múltiplas condições que dependem de uma variável ou expressão. Ele começa com a avaliação de uma variável que, normalmente, representa um valor como número, string ou outro tipo suportado pela linguagem de programação. Com base nesse valor, o switch-case direciona o fluxo do programa para o bloco de código correspondente ao "case" que corresponde a essa variável.

Cada **case** dentro do switch-case representa uma possível condição. Por exemplo, se a variável que está sendo avaliada tem o valor "A", o programa executará o bloco de código do case "A". Caso nenhum dos cases definidos seja atendido, o switch-case pode incluir uma opção **default** – que serve como uma "carta na manga" para tratar valores inesperados ou situações que não foram previstas nos cases. Isso garante que, mesmo que o valor da variável não esteja listado, o programa tenha uma saída padrão.



Um dos aspectos fundamentais do funcionamento do switch-case é o uso do comando **break**, que encerra a execução do bloco de código atual e evita que os próximos cases sejam executados. Sem o break, o programa pode continuar executando os códigos dos cases seguintes, mesmo que eles não sejam correspondentes ao valor avaliado. Esse comportamento é conhecido como **fall-through** e, dependendo da situação, pode ser útil ou pode levar a erros inesperados.

A estrutura do switch-case tem uma sintaxe clara e padronizada na maioria das linguagens, permitindo que o programador implemente múltiplas condições de forma mais eficiente e com menor repetição de código. Por exemplo, em um cenário de criação de menu interativo, o switch-case permite que cada opção seja tratada individualmente, mantendo o código organizado e fácil de entender.

1. **Benefícios**

O **switch-case** oferece benefícios que tornam seu uso atraente para programadores que precisam lidar com múltiplas condições de maneira organizada e eficiente. Um dos principais pontos fortes dessa estrutura é a **legibilidade** que ela proporciona ao código. Ao lidar com diversas condições baseadas em um único valor, o switch-case apresenta uma organização mais clara do que o uso excessivo de estruturas if-else, que podem se tornar confusas e difíceis de manter em projetos maiores. Com o switch-case, o desenvolvedor consegue visualizar rapidamente as opções disponíveis e identificar o que cada bloco de código executa.

Outro benefício importante é a **eficiência**. Em linguagens como C ou Java, o switch-case pode ser implementado internamente com tabelas de busca ou algoritmos otimizados, o que pode melhorar o desempenho em comparação com várias condições encadeadas de if-else, especialmente quando há um grande número de opções. Essa eficiência é valiosa em cenários onde a performance é uma prioridade, como aplicativos que precisam de respostas rápidas em tempo real.

Além disso, o switch-case facilita o **gerenciamento de múltiplas condições**. Por exemplo, ao criar menus interativos ou mapeamento de comandos, essa estrutura evita repetições de código e torna o fluxo mais linear. Cada opção é definida como um case individual, o que permite adicionar ou modificar condições específicas de maneira simples e direta.

Um ponto que não pode ser ignorado é a **utilização do bloco default**, que atua como uma espécie de "plano B". Essa funcionalidade assegura que, mesmo que o valor avaliado não corresponda a nenhum dos cases especificados, o programa terá uma resposta padrão. Isso é especialmente útil para lidar com situações inesperadas ou erros.

Por fim, o switch-case contribui para a **manutenção do código**. Ao lidar com condições que envolvem diversos valores possíveis, a estrutura ajuda o programador a organizar e atualizar o código com mais facilidade. Caso uma nova condição precise ser adicionada, basta incluir um novo case, sem precisar alterar toda a lógica como aconteceria em estruturas complexas de if-else.

1. **Linguagens de programação que suportam e limitações**

O **switch-case**, apesar de ser uma ferramenta poderosa e prática, também possui suas limitações que precisam ser consideradas ao decidir utilizá-lo. Um dos principais pontos fracos é a restrição quanto aos tipos de dados suportados, que podem variar de acordo com a linguagem de programação. Em muitas linguagens, o switch-case só funciona com tipos de dados simples, como números inteiros ou strings, e não permite condições mais complexas que envolvam intervalos ou operadores lógicos. Isso limita a aplicação em cenários onde as condições dependem de comparações mais elaboradas.

Outro ponto é que o switch-case é baseado, em sua maioria, em verificações de igualdade. Isso significa que ele só é eficiente quando o valor da variável corresponde exatamente ao que está definido em um dos cases. Para situações em que é necessário lidar com intervalos ou condições relacionadas, como "maior que" ou "menor que", outras abordagens, como if-else, tornam-se inevitáveis.

Além disso, o comportamento de **fall-through** pode ser um problema, caso o programador se esqueça de inserir o comando "break" ao final de cada case. Isso faz com que o código continue a executar os blocos seguintes, o que pode levar a bugs inesperados. Embora algumas linguagens modernas tenham implementado medidas para evitar isso (como exigir "break" por padrão), ainda é um risco em linguagens mais tradicionais como C.

Por outro lado, o switch-case tem amplo suporte em linguagens de programação como C, Java, JavaScript, entre outras, mas sua implementação pode variar. Em linguagens como Python, por exemplo, ele não existe nativamente, e alternativas como dicionários são usadas para simular seu comportamento. Isso demonstra que, mesmo sendo uma ferramenta útil, o switch-case não é universal e pode exigir adaptações dependendo da linguagem escolhida.

1. **"Fall-through" e como evitá-lo, e Switch-case versus Polimorfismo**

O **switch-case** possui uma característica importante chamada **fall-through**, que ocorre quando o programa continua executando os próximos blocos de código após um case, mesmo que eles não correspondam ao valor avaliado. Esse comportamento acontece nas linguagens que não exigem o uso obrigatório do comando **break** ao final de cada case. O fall-through pode ser útil em situações específicas, como quando você deseja que múltiplas condições compartilhem o mesmo bloco de código. No entanto, na maioria das vezes, ele pode levar a erros inesperados e execução indesejada de comandos. Para evitar problemas, é sempre recomendado utilizar o comando **break** corretamente ao final de cada case, interrompendo a execução e garantindo que o fluxo do programa fique limitado ao case correspondente.

Por outro lado, o uso do switch-case pode ser comparado ao **polimorfismo**, especialmente em situações onde muitas condições precisam ser tratadas. Enquanto o switch-case é uma solução prática e fácil para lidar com múltiplos valores de uma variável, o polimorfismo oferece uma abordagem orientada a objetos para tratar diferentes comportamentos em uma estrutura mais extensível. Com o polimorfismo, é possível criar métodos que se comportam de maneiras distintas de acordo com o tipo de objeto ou classe, tornando o código mais flexível e escalável, especialmente em projetos maiores.

A escolha entre o switch-case e o polimorfismo depende do contexto. Se o objetivo for tratar uma variável com múltiplos valores fixos, o switch-case pode ser mais eficiente e direto. Já em cenários onde é necessário lidar com comportamentos mais complexos e relacionamentos entre classes, o polimorfismo se torna a melhor opção. Apesar de terem propósitos diferentes, ambas as abordagens são ferramentas valiosas e, ao compreender suas vantagens e limitações, os programadores conseguem aplicá-las de maneira inteligente para criar códigos mais robustos e organizados.

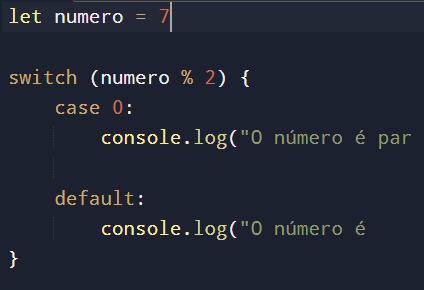
1. **Casos reais de aplicação e implementação prática**

A implementação prática do **switch-case** é uma das maneiras mais eficazes de ilustrar como essa estrutura pode ser utilizada no dia a dia do desenvolvimento de software. Ele é amplamente aplicado em cenários que exigem organização de múltiplas condições ou opções baseadas em um único valor. Um exemplo clássico é a criação de menus interativos em sistemas de interface de linha de comando, onde o usuário escolhe uma opção e o programa executa o código correspondente. Com o switch-case, cada escolha é representada por um **case**, e a execução segue de forma clara e direta, proporcionando uma experiência mais fluida tanto para o desenvolvedor quanto para o usuário final.

Boas práticas são fundamentais ao implementar o switch-case. Uma delas é o uso correto do comando **break** para evitar o comportamento de **fall-through**, onde blocos de código de cases consecutivos podem ser executados acidentalmente. Além disso, a inclusão de um **case default** é altamente recomendada, pois garante que o programa trate valores inesperados de forma previsível, reduzindo a chance de erros. Outra dica é manter os blocos de código dos cases sucintos e modularizados, evitando confusão e facilitando a manutenção.

No mundo real, o switch-case é usado em diversas situações. Por exemplo, em um sistema de e-commerce, ele pode ser utilizado para verificar o status de um pedido e direcionar diferentes ações, como confirmar envio, processar devoluções ou cancelar o pedido. Em jogos, o switch-case pode mapear comandos do jogador, como movimentações ou ataques, para as ações correspondentes no código. Sua versatilidade o torna uma ferramenta indispensável para lidar com múltiplos cenários de maneira clara e eficiente.

1. Código – Gabriell Rodrigues



**Explicação:**

**1° Linha:** cria a variável numero com o valor 7. Esse numero que vamos verificar se é par ou ímpar.

**2° Linha:** usa o operador módulo (%) que retorna o resto da divisão.

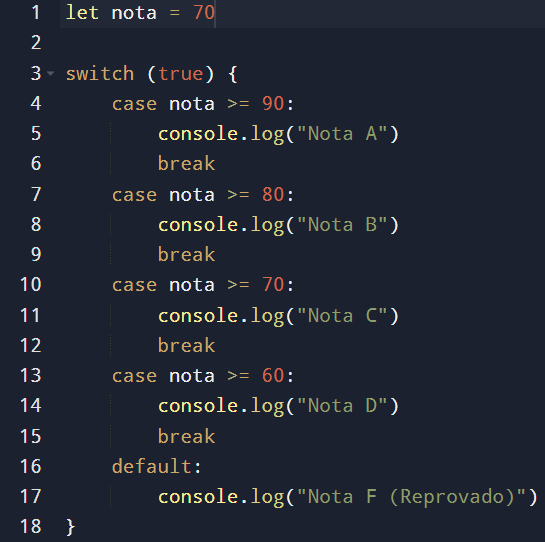
**3° Linha:** numero % 2 vai ser: 0 (se o numero for par 1 se o numero for ímpar) neste caso, como 7 % 2 = 1, ele vai procurar o case 1, mas como não tem, ele vai cair no default.

→ Se o resultado de numero % 2 for 0, significa que o número é par.

→ Executa console.log("O número é par.") Se nenhum case corresponder (ou seja, não foi 0), o default é executado.

→ Neste exemplo, se o número for ímpar, numero % 2 será 1, então o código entra no default. Exibe "O número é ímpar." no console.

1. Código – Diego da Silva



**Explicação:**

**1. let nota = 30;**

→ Aqui está sendo definida a variável nota com valor 30, representando uma pontuação (por exemplo, em uma prova).

**2. switch (true) { ... }**

→ Normalmente usamos switch com valores exatos, mas aqui está sendo usado com a expressão true. Isso permite fazer verificações condicionais nos case, funcionando de forma parecida com uma cadeia de if...else if.

**3. case nota >= 90:**

→ Aqui o case verifica se a nota é maior ou igual a 90. Como o switch está checando true, ele executa o primeiro case cuja condição retorna true.

**4. default:**

→ Se nenhuma das condições for satisfeita (ou seja, nenhuma das case retornar true), o bloco default é executado. Nesse caso, para qualquer nota abaixo de 60, ele exibe:

javascript

console.log("Nota F (Reprovado)");

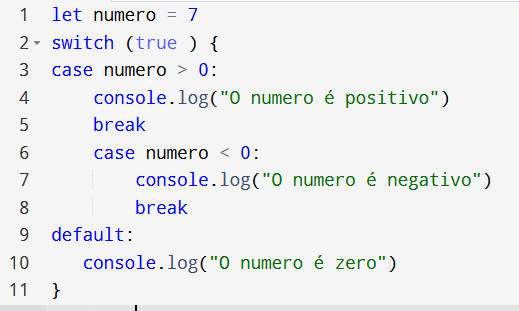
**Resultado para nota = 30:**

→ A nota 30 não atende a nenhuma das condições (>= 60), então o código cai no default, e a saída será:

java

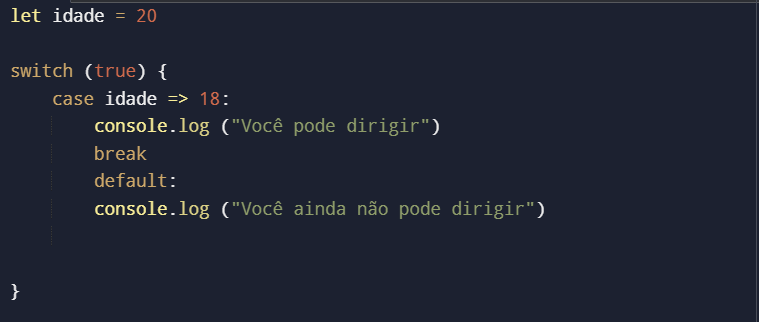
Nota F (Reprovado)

1. Código – Mariane Bertrile



**Explicação:**

1. **Definição da variável**  
   let numero = -5; → Cria a variável numero com valor -5
2. **Início do switch(true)**  
   switch (true) { → Usa true para testar expressões booleanas
3. **Case 1: Número positivo**  
     
   case numero > 0: → Verifica se numero é maior que 0  
     
   console.log("O número é positivo"); → Exibe a mensagem  
     
   break; → Encerra a execução se esse caso for verdadeiro
4. **Case 2: Número negativo**  
     
   case numero < 0: → Verifica se numero é menor que 0  
     
   console.log("O número é negativo"); → Exibe a mensagem  
     
   break; → Encerra a execução se esse caso for verdadeiro
5. **Caso padrão: Número zero**  
     
   default: → Executa este bloco se nenhum dos casos anteriores for verdadeiro  
     
   console.log("O número é zero"); → Exibe a mensagem
6. Código – Hellen de Jesus

****

**Explicação:**

1. l**et idade = 17**  
   → Cria uma variável chamada idade e coloca o valor 17 nela
2. **switch (true)**  
   → Começa a estrutura de decisão. Aqui estamos testando condições booleanas (que podem ser true ou false)
3. **case idade >= 18:**  
   → Verifica se a idade é maior ou igual a 18  
   → Se for true, o código dentro desse bloco será executado
4. **console.log("Você pode dirigir.")**  
   → Mostra essa mensagem **se** a idade for 18 ou mais
5. **break**  
   → Para a execução do switch se a condição acima for verdadeira, pra ele não continuar testando
6. **default:**  
   → Esse bloco será executado **se nenhuma condição anterior for verdadeira**
7. **console.log("Você ainda não pode dirigir.")**  
   → Mostra essa mensagem **se a idade for menor que 18**