Javascript falando sobre IFs, Ifs implícitos e função lambda

Salve padawan, estive olhando os arquivos dos meus artigos e bateu uma culpa, falei sobre tantas coisas e assumi, veja este jedi renegado com mea culpa, não escrevi, acreditando que era simples. Mas esses povos do ECMA gostam de complicar nossa vida e inventam muita coisa útil, porem complicam os novatos.

Este artigo é um complemento a nossa série de artigos sobre o Javascript, vamos explorar o funcionamento das condicionais e descobrir novos caminhos, juro que eu mesmo não conhecia e algumas vezes debugando programas, até via estas instruções e me perguntava o que queria dizer e como funcionava.

Que loucura, as evoluções da linguagem, vão criando novas construções, otimizando algoritmos e facilitando com caminhos alternativos e que diminuem a codificação, porem perder a legibilidade, e dificulta o entendimento. No passado quando criavam linguagem um dos requisitos eram a legibilidade, pois ao codificar necessitamos ler o código, com o requisito humano primeiro.

O que são operadores condicionais?

Quando codificamos uma das logicas mais utilizadas são os controles de fluxo, afinal fazemos uma pergunta, de acordo com a resposta seguimos um caminho feliz ou um caminho tenebroso. Que cara é essa? Calma padawan, voltou um ponto.

O caminho feliz é aquele caminho dentro da lógica do seu programa que segue TRUE TRUE TRUE em seu algoritmo, atendendo os requisitos e funcionando dentro da premissa esperada.

O caminho tenebroso é aquele caminho dentro da lógica do seu programa que segue FALSE FALSE FALSE em seu algoritmo, sendo situações inesperadas e fora da regra do negócio, obrigando atenção e cuidados especiais. Pense nisso ao programar seguindo TRUE para a lógica principal e o FALSE para exceções que devem ser tratadas com codificação especial.

Voltando ao operador condicional existem dois tipos principais IF e SWITCH, ambos funcionam como filtros e controle do fluxo, indicando qual direção a seguir, qual função ou rotina a ser carregada em primeiro lugar.

Em processamento procedural o IF era um dos comandos mais importantes por indicar o fluxo logico do programa, auxiliando o debug e caça a anomalias na hora do Teste Unitário, com o novo paradigma de Orientação ao Objeto os operadores condicionais ganharam nova forma de implementação, simplificando em alguns casos e confundindo enormemente em outros. Vamos por parte que a história é longa.

Operador condicional IF

A condição “SE”, traduzindo para o português, funciona como um sinaleiro, forçando o fluxo do processamento para uma determinada direção de acordo com as condições testadas. De acordo com a pergunta e sua resposta TRUE ou FALSE o ponteiro continuará até o final da instrução.

Veja o exemplo abaixo:

function testNum(a) {

let result;

if (a > 0) {

result = 'Positivo';

} else {

result = 'Negativo';

}

return result;

}

console.log(testNum(-5));

// Nesta execução a expectativa é imprimir “Negativo”

O IF testa o valor recebido, caso seja True / Verdadeiro executa o primeiro statement, caso o resultado seja FALSE / Falso executar o segundo stement.

Multiples IF

Uma implementação mais complexa do IF é o caso do múltiplo IF, em que são feitas inúmeras perguntas, que somente serão executadas caso as anteriores sejam FALSE, veja o exemplo abaixo:

if (condition1)

statement1

else if (condition2)

statement2

else if (condition3)

statement3

...

else

statementN

Sua utilização mantem a programação mais elegante, evitando a cacofonia dos inúmeros IFs dentro de IFs.

Tenha muito cuidado com os operadores lógicos, uma pergunta feita de forma incorreta, gerara uma resposta anômala, que afetara a qualidade do seu software e poderá ocorrerem abend inexplicáveis à primeira vista.

Operador condicional SWITCH

Pense num IF que tomou toddynho ou comeu espinafre. Eis que surge o SWITCH, um super IF onde podemos comparar inúmeras condições para um grupo de variáveis e cada vez que for verdade executa uma serie de instruções. Um pouco perigoso, pois funciona como um touro bravo, porem usando o BREAK, conseguimos controla-lo e obter o melhor desta ferramenta.

Veja o exemplo:

const expr = 'Papayas';

switch (expr) {

case 'Oranges':

console.log('Oranges are $0.59 a pound.');

break;

case 'Mangoes':

case 'Papayas':

console.log('Mangoes and papayas are $2.79 a pound.');

// expected output: "Mangoes and papayas are $2.79 a pound."

break;

default:

console.log(`Sorry, we are out of ${expr}.`);

}

Reparou que o comando inicia com uma pergunta em SWITCH, depois são apresentadas as situações caso a caso e no fim temos uma condição geral, caso as anteriores não funcionem. Veja o Break controlando o fluxo e encerrando, quando necessário.

Agora que vimos o IF e o SWITCH, vamos avançar e ver um tópico novo, e sendo sincero eu mesmo apanhei e não tenho total segurança sobre os temas abaixo:

Vamos falar sobre operadores condicionais

Alguem em algum dia, com preguiça de escrever código, sugeriu uma nova grafia, bem complexa para os não iniciados, que causa muita confusão e requer atenção máxima do codificador e como não é clara, muitas vezes passa batida na debugaçao e muitas vezes ajudam a causar ABENDs.

O **operador condicional (ternário)** é o único operador JavaScript que possui três operandos. Este operador é frequentemente usado como um atalho para a instrução [if](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/if...else).

[**Sintaxe**](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Conditional_Operator#sintaxe)

*condition* ? *expr1* : *expr2*

[**Parâmetros**](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Operators/Conditional_Operator#par%C3%A2metros)

**condition**

Uma expressão que é avaliada como true ou false.

**expr1, expr2**

Expressões com valores de qualquer tipo.

Dando uma boa olhada, o programador atento notara, que é uma nova maneira de escrever IFs de forma direta e economizando código, porem cuidado com o uso das chaves e as condições.

Exemplo:

"The fee is " + (isMember ? "$2.00" : "$10.00")

var elvisLives = Math.PI > 4 ? "Yep" : "Nope";

var firstCheck = false,

secondCheck = false,

access = firstCheck ? "Access denied" : secondCheck ? "Access denied" : "Access granted";

console.log( access ); // logs "Access granted"

var stop = false, age = 16;

age > 18 ? location.assign("continue.html") : stop = true;

var age = 16;

var url = age > 18 ? (

alert("OK, you can go."),

// alert returns "undefined", but it will be ignored because

// isn't the last comma-separated value of the parenthesis

"continue.html" // the value to be assigned if age > 18

) : (

alert("You are much too young!"),

alert("Sorry :-("),

// etc. etc.

"stop.html" // the value to be assigned if !(age > 18)

);

location.assign(url); // "stop.html"

Função Lambda ou Arrow Function

Mais um golpe de programadores preguiçosos e que servem para confundir os novatos e ou velhotes egressos de outras linguagens procedurais. Brincadeiras e maldades a parte, a Funçao Lambda tem sua vantagem e grande qualidade: economiza código na escrita.

Atualmente temos visto movimentos que minimizam a codificação, fugindo dos primeiros idealizadores das linguagens de programação, cujo o paradigma eram criar LP de fácil leitura, que auxiliam-se o programador a entender a logica legada, sem muita complexidade e podendo debugar mais rapidamente.

O Arrow Function nada mais é que uma uma função anônima minimalista, podendo ser escrita em até uma única linha, claro que encontrara maiores, em que o programador utiliza chaves. Principais vantagens funções mais curtas e a inexistência da palavra chave this.

const materials = [

'Hydrogen',

'Helium',

'Lithium',

'Beryllium'

];

console.log(materials.map(material => material.length));

// expected output: Array [8, 6, 7, 9]

Conheça a sintaxe básica

(param1, param2, …, paramN) => { statements }

(param1, param2, …, paramN) => expression

// equivalente a: => { return expression; }

// Parênteses são opcionais quando só há um nome de parâmetro:

(singleParam) => { statements }

singleParam => { statements }

// A lista de parâmetros para uma função sem parâmetros deve ser escrita com um par de parênteses.

() => { statements }

Recordando, vamos olhar um pouco sobre os operadores

Recapitulando existem inúmeros tipos de operadores, a seguir enumeros os existentes em outro artigo comentei sobre eles, neste iremos nos ater nos de comparação e lógicos, recomendo ler o artigo anterior.

* [**Operadores de atribuição**](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Expressions_and_Operators#operador_atribuicao)
* [**Operadores de comparação**](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Expressions_and_Operators#operador_comparacao)
* [**Operadores aritméticos**](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Expressions_and_Operators#operadores_aritmeticos)
* [**Operadores bit a bit**](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Expressions_and_Operators#operadores_bit_a_bit)
* [**Operadores lógicos**](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Expressions_and_Operators#operadores_logicos)
* [**Operadores de string**](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Expressions_and_Operators#operadores_string)
* [**Operador condicional (ternário)**](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Expressions_and_Operators#operador_condicional_ternario)
* [**Operador vírgula**](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Expressions_and_Operators#operador_virgula)
* [**Operadores unário**](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Expressions_and_Operators#operadores_unario)
* [**Operadores relacionais**](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Expressions_and_Operators#operador_virgula)

Operadores de Comparação

Ira comparar uma condição A com outra condição B, retornando TRUE ou FALSE de acordo com o resultado das condições abaixo:

| **Operadores de comparação** | | |
| --- | --- | --- |
| **Operador** | **Descrição** | **Exemplos que retornam verdadeiro** |
| Igual (==) | Retorna verdadeiro caso os operandos sejam iguais. | 3 == var1  "3" == var1  3 == '3' |
| Não igual (!=) | Retorna verdadeiro caso os operandos não sejam iguais. | var1 != 4 var2 != "3" |
| Estritamente igual (===) | Retorna verdadeiro caso os operandos sejam iguais e do mesmo tipo. Veja também [Object.is](https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Global_Objects/Object/is) e [igualdade em JS (en-US)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness). | 3 === var1 |
| Estritamente não igual (!==) | Retorna verdadeiro caso os operandos não sejam iguais e/ou não sejam do mesmo tipo. | var1 !== "3" 3 !== '3' |
| Maior que (>) | Retorna verdadeiro caso o operando da esquerda seja maior que o da direita. | var2 > var1 "12" > 2 |
| Maior que ou igual (>=) | Retorna verdadeiro caso o operando da esquerda seja maior ou igual ao da direita. | var2 >= var1 var1 >= 3 |
| Menor que (<) | Retorna verdadeiro caso o operando da esquerda seja menor que o da direita. | var1 < var2 "12" < "2" |
| Menor que ou igual (<=) | Retorna verdadeiro caso o operando da esquerda seja menor ou igual ao da direita. | var1 <= var2 var2 <= 5 |

Operador logico

Interliga vários testes em um único teste que podem obter respostas TRUE, TRUE/FALSE, FALSE/FALSE e FALSE/TRUE de acordo com a pergunta feita, olho vivo pequeno padawan, muitos mestres jedis já caíram e cometeram grandes deslizes nas operações logicas.

| **Operador** | **Utilização** | **Descrição** |
| --- | --- | --- |
| AND lógico (&&) | expr1 && expr2 | (E lógico) - Retorna expr1 caso possa ser convertido para falso; senão, retorna expr2. Assim, quando utilizado com valores booleanos, && retorna verdadeiro caso ambos operandos sejam verdadeiros; caso contrário, retorna falso. |
| OU lógico (||) | expr1 || expr2 | (OU lógico) -  Retorna expr1 caso possa ser convertido para verdadeiro; senão, retorna expr2. Assim, quando utilizado com valores booleanos, || retorna verdadeiro caso ambos os operandos sejam verdadeiro; se ambos forem falsos, retorna falso. |
| NOT lógico (!) | !expr | (Negação lógica) Retorna falso caso o único operando possa ser convertido para verdadeiro; senão, retorna verdadeiro. |

Conclusao

PAdawan o tema é mais complexo , que imaginei inicialmente e mais uma vez, o artigo se extendeu acima das mil palavras, espero ter sido claro e auxiliado nesta jornada de descoberta e conhecimento.

Verificamos a utilização do IF e do SWITCH, passamos pelos operadores lógicos, demos uma vista de olhos na Funçoes Lambda / Arrow Function e vimos seu funcionamento e principais vantagens, mas sendo sincero, necessitarei criar um único artigo focado apenas nela e pensando nos preguiçosos revisamos os operadores condicionais, mais conhecidos como ifs sem ifs.