MÓDULO

JAVASCRIPT/AJAX

UNIDADE

OPERAÇÕES E CONDIÇÕES EM JAVASCRIPT

Instituto Tecnológico

ÍNDICE

OBJETIVOS **3**
INTRODUÇÃO **4**
1. OPERADORES **5**
1.1. OPERADORES DE ATRIBUIÇÃO 5
1.2. OPERADORES ARITMÉTICOS 6
1.3. OPERADORES RELACIONAIS 9
1.4. OPERADORES LÓGICOS 10
1.5. OPERADORES DE STRINGS 11
2. CONDIÇÕES **12**
2.1. IF... ELSE 12
2.2. SWITCH 17
3. LOOP SENTENCES **20**
3.1. FOR 20
3.2. DO... WHILE 23
3.3. WHILE 24
3.4. QUEBRAR E CONTINUAR AS INSTRUÇÕES DE LOOP 26
4. OUTROS COMANDOS **28**
4.1. FORMA ABREVIADA DO IF 28
4.2. FOR... IN 29
CONCLUSÃO **31**

AUTOAVALIAÇÃO 33
SOLUÇÕES 37
PROPOSTAS DE DESENVOLVIMENTO DO ESTUDO 38
BIBLIOGRAFIA.....39

OBJETIVOS

Com esta unidade didática, pretende-se que desenvolva os seguintes objetivos de aprendizagem:

- Aprender a usar os operadores em JavaScript.
- Saber como utilizar condições.
- Aprender a usar loops.

INTRODUÇÃO

Nesta unidade aprenderá como o JavaScript permite operar e controlar dados. Para se poder criar uma calculadora ou exibir dados de uma folha de Excel na página web, utilizam-se operadores e condições explicadas nesta unidade. Aprenderá a controlar os dados e a informação que mostra na página web, assim como loops, condições e operadores e criará a primeira página com código JavaScript. Para isso, precisará de um editor de texto e um navegador da web.

É recomendado ter um dos seguintes editores gratuitos:

- Sublime Text 3.
- Visual Studio Code.

1. OPERADORES

Operadores são palavras e símbolos que o JavaScript usa para trabalhar com valores diferentes. Pode-se dividir a lista de operadores em cinco tipos:

1.1. OPERADORES DE ATRIBUIÇÃO

Este tipo de operador tem apenas um sinal que é o igual (=). Este operador permite dar valores às variáveis, passar valores de uma variável para outra, etc. Como o seu nome indica, "atribui um valor".

Noutras linguagens de programação é usado o sinal de igual para estabelecer relações de igualdade, ou seja, para fazer perguntas como “este valor é igual a outro?”.

Em JavaScript não é esse o caso; o sinal de igual é apenas usado como uma forma de atribuir valores. Seguem-se alguns exemplos:

```
var contador;
contador = 1;
var texto;
texto = ‘olá, eu sou uma variável’;
var valor1 , valor2;
valor1 = 30;
```

```
valor2 = valor1;
var soma;
soma = soma(3,7); // neste caso, teria que criar uma função de soma
```

1.2. OPERADORES ARITMÉTICOS

Estes operadores modificam ou geram um novo valor, e dividem-se em dois tipos:

- **Unitários.**

  Estes operadores modificam o valor a que estão a ser aplicados, ou seja, alteram o valor de uma variável sem necessitar de um segundo valor para o substituir.

  Existem três tipos de operadores aritméticos unitários:

  - **Incremento**: dois sinais de mais (++); aumenta o valor a que é aplicado em uma unidade.
  - **Decremento**: dois sinais de menos (--); diminui o valor a que é aplicado em uma unidade.
  - **Negação**: um sinal de menos (-); torna o valor a que é aplicado negativo (ao aplicar-se a um número negativo, torna-o positivo).

    Seguem-se alguns exemplos:

    ```
    contador = 10;
    contador++;
    ```

    Neste caso, o contador será 11 no final.

```
contador = 10;
contador--;
```

Neste caso, o contador será 9 no final.

```
contador = 10;
contador = -contador;
```

Neste caso, o contador será, no final, -10 (negativo).

Deverá ter em mente que os dois primeiros operadores unitários (++) e (--) podem ser colocados antes ou depois do valor a ser alterado e afetam o resultado final. Ao colocar antes, alteram o valor antes de ser utilizado, enquanto, ao colocar depois, alteram-no após ser utilizado.

Segue-se um exemplo para esclarecer:

```
Contador = 10;
contador2 = contador++;
```

Neste caso, o contador2 = 10 e o contador = 11.

```
Contador = 10;
contador2 = ++contador;
```

Neste caso, o contador2 = 11 e o contador = 11.

- **Binários.**

  Os operadores binários não alteram o valor diretamente, mas permitem criar um terceiro valor a partir de dois ou mais valores.

  Existem cinco tipos de operadores binários:

  - **Adição**: é o sinal de mais (+) e faz a soma entre os valores indicados.
  - **Subtração**: é o sinal de menos (-) e subtrai os valores indicados.
  - **Multiplicação**: é um asterisco (*) e multiplica os valores indicados.

- **Divisão**: é uma barra simples (/) e divide o primeiro valor indicado pelo segundo valor.
- **Restante**: é uma percentagem (%) e devolve o resto de uma divisão entre dois valores indicados.

Os operadores binários utilizam-se da seguinte forma:

```
variavel = valor1 operador valor2
```

Exemplos:

```
variavel = 4 + 5
variavel = 13 * 5
variavel = contador / 5
```

Caso deseje utilizar um operador binário para alterar o valor da variável, sendo um dos dois valores da operação a variável, pode utilizar-se a seguinte forma abreviada:

```
variavel operador= valor
```

Segue-se um exemplo:

- Forma completa: contador = contador + 25
- Forma abreviada: contador += 25

É importante referir que esta forma abreviada só é útil quando um dos valores que vai ser alterado é a própria variável onde irá ser guardado o resultado.

1.3. OPERADORES RELACIONAIS

Os operadores relacionais são aqueles que se encarregam de fazer comparações entre valores, devolvendo sempre um valor verdadeiro ou falso (0 ou 1).

Existem seis tipos de operadores relacionais:

- **Maior do que:** é um sinal de maior (>) e devolve um resultado verdadeiro se o primeiro valor (à esquerda) for maior do que o segundo valor.
- **Menor do que**: é um sinal de menor (<) e devolve um resultado verdadeiro se o primeiro valor for menor do que o segundo valor.
- **Maior ou igual a:** é um sinal de maior e um igual (>=) e devolve um resultado verdadeiro se o primeiro valor for maior ou igual ao segundo valor.
- **Menor ou igual a:** é um sinal de menor e um igual (<=) e devolve um resultado verdadeiro se o primeiro valor for menor ou igual ao segundo valor.
- **Igual:** dois sinais de igual (==); devolve verdadeiro se ambos os valores forem iguais.
- **Diferente:** é um ponto de exclamação e um igual (!=) e devolve verdadeiro se os dois valores não coincidirem.

A forma correta de uso é a seguinte:

```
variavel = valor operador valor
```

Alguns exemplos:

```
5 > 6 //devolve false (falso)
5 >= 5 //devolve true (verdadeiro)
3 < 8 //devolve true
3 <= 2 //devolve false
```

```
3 == 3 //devolve true
4 != 2 //devolve true
3 == ‘3’ //devolve true
```

No último exemplo, na comparação 3 == '3', em que é comparado o número 3 com a string '3', devolve verdadeiro; isto ocorre porque o JavaScript compara os valores em todas as suas formas possíveis.

1.4. OPERADORES LÓGICOS

Os operadores lógicos permitem concatenar (unir) diferentes tipos de operações para evitar o uso de loops.

Existem três tipos principais de operadores lógicos:

- **AND**: dois “e” comerciais (&&); verifica se as operações à esquerda e à direita do operador são verdadeiras; caso não sejam, devolve falso.
- **OR**, duas barras verticais (||); verifica se a operação à esquerda ou à direita são verdadeiras, caso contrário, devolve falso.
- **NOT**: é um ponto de exclamação (!)e é usado para fazer negações.

Seguem-se alguns exemplos:

```
(5 > 3) && (13 < =13 )        //devolve true
(5 < 3) || (13 > 13)          //devolve false
!(5 > 3)                      //devolve false
```

Segue-se uma tabela com valores para duas variáveis x e y, que têm valores binários 0 e 1, e suas combinações possíveis.

x	y	x && y	x || y	!x
0	0	0	0	1
0	1	0	1	1
1	0	1	1	0
1	1	0	0	0

1.5. OPERADORES DE STRINGS

Os operadores de strings são utilizados para trabalhar com texto:

- **Soma**: é um sinal de mais (+) e serve para concatenar strings.
- **Acrescentar**: é um sinal de mais e um igual (+=) e é utilizado para adicionar texto a uma string já existente.
- **Igualdade**: dois iguais (==); é utilizado para comparar strings.
- **Desigualdade**: é um ponto de exclamação e um igual (!=) e serve para comparar negativamente duas strings, ou seja, para saber se são diferentes.

Seguem-se alguns exemplos:

```
variavel = “casas ”  + “de madeira”
//insere na variável os dois textos

variavel += “casas”
//adiciona à variável o texto

(“casas” == “casa”)
//devolve false
```

2. CONDIÇÕES

As frases condicionais são as ferramentas mais utilizadas em JavaScript; servem para saber qual a parte de código a executar após uma pergunta que irá devolver verdadeiro ou falso.

As duas condições em JavaScript são:

2.1. IF... ELSE

Utilizado para executar uma parte de código se o resultado devolvido pela condição for verdadeira. Caso queira executar uma parte do código diferente se a condição devolver falsa, utiliza-se o else.

Uma condição **if... else** (que consiste na palavra reservada **if**) tem entre parênteses a expressão condicional, que será verdadeira ou falsa; dependendo do resultado devolvido, é executado o código que está entre chavetas({}).

Lembre-se de que a condição que utiliza pode ser simples ou múltipla. Veja estes dois exemplos:

```
If (3 < 4) {}
//condição simples
```

```
If ((3 < 4) && (5 > 3) && (2==4)) {}
//condição múltipla
```

Pode criar condições de três maneiras diferentes:

- **Simples.**

```
If (3 > 5) {
alert('3 é maior que 5');
}
```

Esta condição simples verifica apenas se a condição é verdadeira e, caso seja, executa o alerta.

- **Com resultado falso.**

```
If (3 > 5) {
…
código
…
}else{
…
código
}
```

Esta condição **if** permite executar um código se a condição for verdadeira e, caso contrário, executa o código entre as chavetas a seguir ao **else**.

- **Condições em cadeia.**

```
If (3 > 5) {
…
código
…
}else if (5> 2){
```

```
…
código
 …
}else if (8 == 8){
…
código
…
}else{
…
código
…
}
```

Como pode observar, este exemplo permite criar condições em cadeia. Neste caso, quando o código for executado ele irá verificar a primeira condição: se for verdadeira, executa o código e ignora o resto do if; se for falsa, irá verificar a segunda condição e assim por diante até encontrar uma condição verdadeira ou alcançar o fim das condições.

Exemplo 01

No exemplo seguinte, será criado um código simples que compara dois números, que o utilizador pode escrever dentro de duas caixas de texto diferentes e devolver o maior dos dois.

Exemplo:

```
<html>

<head>
    <script type="text/javascript">

        function compararNum() {
```

```
            var var1 = +document.getElementById('campo1').value;
            var var2 = +document.getElementById('campo2').value;

            if (var1 > var2) {
                alert('o valor mais alto é ' + var1);
            }

            else if (var1 < var2) {
                alert('o valor mais alto é ' + var2);
            }

            else {
                alert('os valores são iguais');
            }
        }

    </script>

</head>

<body>

    Número 1 :<input type="text" name="campo1" id="campo1" />
    Número 2 :<input type="text" name="campo2" id="campo2" />
    <input type="submit" value="Comparar" onclick="compararNum()" /
>

</body>

</html>
```

Neste exemplo, foi criada uma função que é ativada com o evento onClick do rato ao clicar no botão. Mais tarde vai ser demonstrado como criar funções em JavaScript. Para aceder aos dados nas caixas, foi utilizada uma função JavaScript designada document.getElementById, que permite aceder aos valores e atributos de um elemento através do seu id.

O sinal + incluído antes do excerto de código
+document.getElementById('campo1').value; e
+document.getElementById('campo2').value; serve para garantir que o resultado seja reconhecido como um número.

Resultados do exemplo.

2.2. SWITCH

Esta condição permite que o JavaScript avalie uma determinada expressão e a compare com os valores indicados. Se eles forem iguais, o código associado a esse valor será executado.

Pode criar uma condição switch da seguinte forma:

```
switch (expressao) {
case valor1:
… código …
break; //não é necessário
case valor2:
… código …
break; //não é necessário
…
default:
… código …
break; //não é necessário
}
```

Neste exemplo poderá colocar os valores de "case" que desejar. Ao executar o JavaScript, ele irá verificar cada valor; se corresponder, ele “entra” para executar o código associado; caso nenhum valor corresponda, “entra” para o valor padrão e executará o código associado (este valor padrão não é obrigatório e, se não estiver presente e não corresponder, a execução do código simplesmente continuará).

A instrução break não é obrigatória. Pode ser utilizada no final de cada “case” e é escrita no final do código associado. O break faz com que a o JavaScript ignore o resto do código na chaveta e vá diretamente para o final dela, continuando com o resto do código que a página possui (evitando assim que ele insira outro dos valores).

Segue-se um exemplo completo para entender como o switch funciona.

Exemplo 02

No exemplo seguinte, pretende-se que o utilizador insira o seu signo do zodíaco e, de acordo com o signo inserido, seja exibido um texto sobre o seu futuro.

Exemplo:

```
<html>
<head>
<script type="text/javascript">
function compararNum(){
switch (document.getElementById('campo1').value)
 {
 case "capricórnio":
    alert('Saúde e negócios são a sua praia esta semana!');
    break;
 case "leão":
    alert('Uma oportunidade difícil de recusar aparecerá no seu
local de trabalho!');
    break;
 case "caranguejo":
    alert('Cuide um pouco melhor de si, ou vai ficar doente neste
inverno');
    break;
 case "carneiro":
    alert('A nível pessoal, deixe os seus amigos aconselharem-no');
    break;
 case "balança":
    alert('Tudo bem! Continue assim.');
    break;
 case "touro":
```

```
    alert('Tem muito trabalho, tire umas férias. A sua saúde vai
agradecer');
    break;
 default:
    alert('Não temos esse signo do zodíaco, tente outro');
 }
}
</script>
</head>
<body>
 Signo:<input type="text" name="campo1" id="campo1"/>
<input type="submit" value="Ver a sorte" onclick="compararNum()"/>
</body>
</html>
```

Resultado do exemplo.

Neste exemplo, também é utilizada uma função a ser executada pelo switch quando o botão é pressionado.

3. LOOP SENTENCES

Um loop é uma série de comandos de código executados enquanto ocorrer uma condição específica ou até que outra condição aconteça. Os loops são muitas vezes utilizados em programação para executar vetores, gerar listas, etc.

As operações loop suportadas por JavaScript são:

3.1. FOR

Um for é um loop que se repete até que a condição atribuída a ele seja falsa. É formado por um valor inicial, uma condição e um incremento para atingir a condição. Como no caso das declarações condicionais, o conteúdo da expressão for é colocado entre parênteses e o código a ser executado é colocado entre chavetas.

```
for (valor inicial; condição; incremento)
{
 …código …
}
```

Quando chega a execução do for, primeiro é verificada a expressão do valor inicial: que pode ser o valor atribuído a uma variável ou a criação da mesma com um valor inicial.

De seguida, o valor devolvido é verificado pela condição, se for falso, o loop é encerrado; e se for verdadeiro, ele irá entrar no for e executar o código associado.

Por fim, o valor da variável é atualizado com o incremento atribuído na declaração do for e volta à segunda etapa, ou seja, volta a verificar o valor devolvido pela condição. E assim sucessivamente até que a condição seja falsa.

```
for (i = 1; i < 10; i++){
    …
    código
    …
}
```

O exemplo acima irá executar o código associado 9 vezes, pois a condição é verdadeira desde que "i" seja menor que 10.

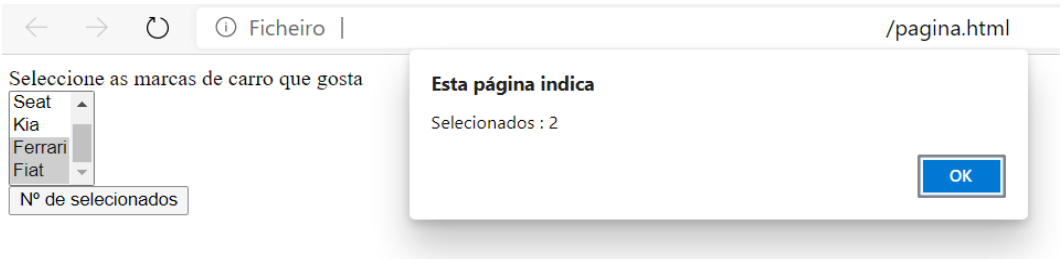
Exemplo 03

No exemplo seguinte, será colocado na página um formulário com uma lista de valores (seleção múltipla) para o utilizador selecionar os valores que desejar; ao pressionar o botão, dirá o número de seleções que o utilizador fez.

Exemplo:

```
<html>
<head>
<script type="text/javascript">
function quantos(objSelect) {
 var ns = 0;
 for (var i = 0; i < objSelect.options.length; i++) {
 if (objSelect.options[i].selected)
 ns++;
 }
 alert('Selecionados : ' + ns);
```

```
 }
</script>
</head>
<body>
<form name="Fselect">
 Seleccione as marcas de carro que gosta<br/>
<select name="carros" multiple="multiple">
<option>Audi</option>
<option>Seat</option>
<option>Kia</option>
<option>Ferrari</option>
<option>Fiat</option>
</select>
<br/>
<input type="button" value="Nº de selecionados" onclick =
"quantos(document.Fselect.carros)" />
</form>
</body>
</html>
```



Resultado do exemplo.

3.2. DO... WHILE

O do... while executa o código associado até que a condição se torne falsa. Este comando tem uma característica especial que o diferencia do for: qualquer que seja a condição, o código é executado pelo menos uma vez.

Quando o código é executado pela primeira vez, a condição é verificada: se for verdadeira, é executada novamente e assim sucessivamente até que a condição seja falsa.

Este tipo de declarações deve ser utilizado com atenção, pois se não criar bem a condição ou se não houver possibilidade de que a condição se torne falsa, ela ficará num loop infinito.

Um loop infinito é uma repetição do mesmo ciclo sem fim (a página web não avança na leitura do resto do código até o loop terminar).

Segue-se um exemplo de uma declaração do... while:

```
do {
    …
    código
    …
} while (condição);
```

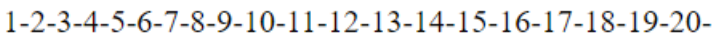
Exemplo 04

Neste exemplo, pretende-se escrever uma sequência numérica de 1 a 20 usando uma declaração do... while.

Exemplo:

```
<html>
<head></head>
<body>
```

```
<script type="text/javascript">
 i = 0;
 do {
    i += 1;
    document.write(i + '-');
 } while (i < 20);
</script>
</body>
</html>
```



Resultado do exemplo.

3.3. WHILE

O while é bastante idêntico ao do... while, com a exceção de que, neste caso, é verificado se a condição é verdadeira antes de executar o código associado. Segue-se um exemplo da sua utilização:

```
while (condição){
    …
    código
    …
}
```

Deve-se utilizar com atenção, à semelhança do anterior, pois se a condição não se tornar falsa, entrará num loop infinito que irá bloquear a página.

Exemplo 05

Neste exemplo, pretende-se escrever uma sequência numérica de 1 a 20 usando uma declaração while.

Exemplo:

```
<html>
<head></head>
<body>
<script type="text/javascript">
i = 0;
 while (i < 20) {
    i += 1;
    document.write(i + '-');
 }
</script>
</body>
</html>
```

1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-19-20-

Resultado do exemplo.

3.4. QUEBRAR E CONTINUAR AS INSTRUÇÕES DE LOOP

- **Break.**

  O break pode ser utilizado em loops para terminar a execução. Para fazer isso, basta colocar break no lugar do código onde se deseja que termine. Segue-se um exemplo:

```
for (i = 1;i < 5;i ++){
if (i == 3){
   break; // quando i for igual a 3, sairá do loop
}
document.write(i + '-')
}
```

- **Continue.**

  O continue pode ser utilizado dentro de loops para terminar a iteração atual do loop sem o encerrar, ou seja, saltar para a próxima iteração. Para fazer isso, basta colocar continue no lugar do código que quer que seja ignorado.

Exemplo 06

Neste exemplo, pretende-se escrever uma sequência numérica de 1 a 5, ignorando o número 3, usando um **loop** e uma declaração while e continue.

Exemplo:

```
<html>
<head></head>
<body>
<script type="text/javascript">
 i = 0;
 while (i < 5) {
```

```
 i++;
 if (i == 3)
 continue;
 document.write(i + '-');
 }
</script>
</body>
</html>
```

1-2-4-5-

Resultado do exemplo.

4. OUTROS COMANDOS

Segue-se uma lista de frases menos utilizadas, mas que podem ser úteis.

4.1. FORMA ABREVIADA DO IF

Existe uma maneira abreviada de criar uma declaração if... else: deve ter em mente que ela ajuda a atribuir um valor a uma variável e não apenas para comparar dois valores. Utiliza-se da seguinte forma:

```
variável = (condição) ? valor1 : valor2
```

Quando esta declaração é executada, verifica a condição que existe antes do ponto de interrogação (?); se essa instrução for verdadeira, irá atribuir o valor1 à variável, se for falsa, atribuirá o valor2. Segue-se um exemplo:

```
var variavel = (5 > 6) ? “maior” : “menor”;
```

Neste caso, a variável será "menor".

4.2. FOR... IN

O **for... in** é utilizado em programas mais específicos e permite percorrer certas propriedades de um objeto com um número (o índice). Este objeto pode ser um objeto criado pelo utilizador, um array de valores ou pode até ser um objeto HTML.

Exemplo 07

Neste exemplo, pretende-se percorrer o array que foi utilizado anteriormente com marcas de carros desportivos, mostrando os valores no ecrã.

Exemplo:

```
<html>
<head></head>
<body>
<script type="text/javascript">
var x;
 var carros = new Array();
carros[0] = "porsche";
 carros[1] = "ferrari";
 carros[2] = "lamborghini";
 carros[4] = "mercedes";
 for (x in carros)
 {
 document.write(carros[x] + "<br/>");
 }
</script>
</body>
</html>
```

porsche
ferrari
lamborghini
mercedes

Resultado do exemplo.

CONCLUSÃO

Um operador JavaScript é uma ferramenta que permite criar páginas mais complexas e úteis. Juntamente com as condições e os loops apresentados nesta unidade, permite criar uma série de novas funcionalidades para a sua página web.

AUTOAVALIAÇÃO

1. **Qual é o sinal utilizado em operadores de atribuição?**

   a) Um sinal de igual (=).

   b) Dois sinais de igual (==).

   c) Um hífen e o sinal de maior (->).

   d) Um ponto de exclamação e um igual (!=).

2. **O que são os operadores aritméticos unitários?**

   a) São operadores que criam um valor a partir de dois outros valores indicados.

   b) São operadores que apenas comparam valores.

   c) São operadores que modificam de valores fornecidos a partir de um outro valor.

   d) São operadores que modificam um valor sem a necessidade de um segundo valor.

3. **Qual dos seguintes operadores não é do tipo aritmético binário?**

   a) Adição.

   b) Igualdade.

   c) Multiplicação.

   d) Restante.

4. Qual é o operador que permite verificar se dois valores são diferentes?

a) O sinal de menor e o sinal de maior juntos (<>).

b) Um ponto de exclamação e um sinal de igual (!=).

c) A palavra reservada NOT.

d) Dois sinais de igual (==).

5. O que é que os operadores lógicos permitem fazer?

a) Comparar valores numéricos.

b) Gerar um valor a partir de dois valores.

c) Concatenar diferentes operações.

d) Alterar strings.

6. O que é utilizado para criar uma string nova a partir de outras duas?

a) Um "e" comercial (&).

b) Uma barra vertical (|).

c) Um sinal de mais (+).

d) Dois sinais de igual (==).

7. Qual das seguintes opções é uma condição?

a) for.

b) while.

c) for... in.

d) if.

8. O que é que a condição if... else permite fazer?

a) Verificar uma condição e executar um código se for verdadeiro ou outro se for falso.

b) Percorrer uma série de números em loop.

c) Executar um pedaço de código enquanto verifica uma condição.

d) Verificar uma condição e executar uma condição verdadeira.

9. Para que é utilizada a condição else if?

a) Criar condições if em loop.

b) Executar else para valores verdadeiros em vez de falsos.

c) Inverter o valor da condição.

d) Executar o código else e, em seguida, o código if (executando ambos).

10. Qual é a palavra utilizada para avaliar uma expressão e comparar o valor com uma série de resultados?

a) for.

b) switch.

c) while.

d) break.

11. De que é feita uma declaração for?

a) Valor inicial e condição.

b) Condição.

c) Valor inicial, condição e incremento.

d) Condição e incremento.

12. Qual é a declaração loop que permite executar um código antes de verificar a condição?

a) do... while.

b) while.

c) for... in.

d) for.

13. O que é que um break permite fazer?

a) Saltar uma linha de código.

b) Ir até a próxima iteração do loop.

c) Sair do loop.

d) Fazer uma pausa no loop.

14. Qual é a declaração utilizada para saltar uma iteração de um loop e ir para a próxima?

a) break.

b) continue.

c) go to.

d) jump.

15. Qual a declaração utilizada para percorrer os elementos de um objeto, (por exemplo, um array)?

a) for.

b) switch.

c) while.

d) for... in.

SOLUÇÕES

1.	a	**2.**	d	**3.**	b	**4.**	b	**5.**	c
6.	c	**7.**	d	**8.**	a	**9.**	a	**10.**	b
11.	c	**12.**	a	**13.**	c	**14.**	b	**15.**	d

PROPOSTAS DE DESENVOLVIMENTO DO ESTUDO

Para saber mais sobre operadores e estruturas de controlo e ver outros exemplos, visite o seguinte site:

- https://www.w3schools.com/js/js_operators.asp

BIBLIOGRAFIA

- MDN Web Docs (2021). "Controle de Fluxo e Manipulação de Erro". Disponível em:

  https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Declara%C3%A7%C3%B5es. Consultado a 18 de janeiro de 2021.

- MDN Web Docs (2021). "Laços e Iterações". Disponível em:

  https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Lacos_e_iteracoes. Consultado a 20 de janeiro de 2021.

- VV. AA. (2010), *JavaScript*. Madrid: AnayaMultimedia.