# Estruturas de controle de fluxo JavaScript

DEVinHouse

Parcerias para desenvolver a sua carreira

SENAI

<LAB365>

# AGENDA | M1S03-A1

- Revisão e tira dúvidas (Introdução à JavaScript)
  - Variáveis, tipos de dados, operadores, DOM
  - Atividade
- Estruturas de controle de fluxo
  - Condicionais ( if / else / switch )
  - Repetição ( while / do while / for )

# **REVISÃO** | Introdução à CSS P.1

- **Variáveis**: Espaço de memória nomeado (começa com letras, $, _)
- **Tipos de dados**: string, number, boolean, object, function, undefined
  array (object), null (object), NaN (number)
- **Operadores**:
  - Aritméticos: `+` `-` `*` `/` `%` `**`
  - Atribuição: `=` `+=` `-=` `*=` `/=` `%=` `++` `--`
  - Relacionais: `<` `>` `<=` `>=` `==` `!=` `===` `!==`
  - Lógicos: `!` `||` `&&`
  - Ternário: `a ? b : c`
- **DOM**: getElement...() querySelector...()

# ATIVIDADE

# ATIVIDADE

## Are all strawberries red?

15

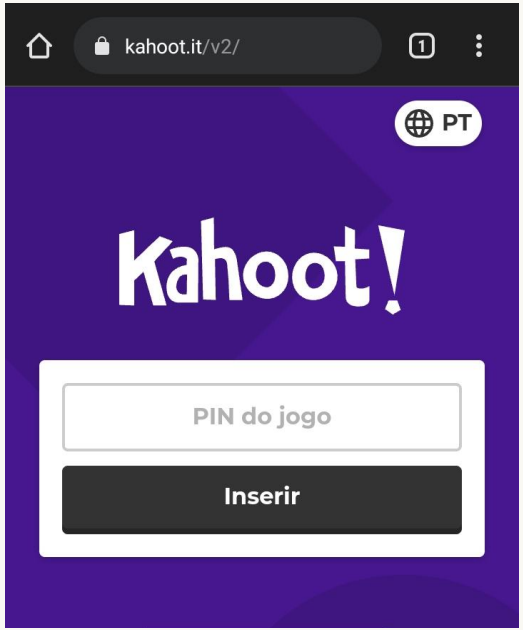
- ▲ Yes
- ◆ No, white strawberries exist too
- ● No, black strawberries exist too
- ■ No, yellow strawberries exist too

# ATIVIDADE

Acesse **kahoot.it**

e digite o **PIN** que aparecerá aqui na transmissão

(**acessar pelo smartphone**)

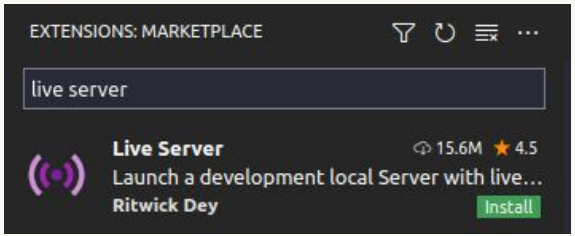



# PARA A MÃO NA MASSA

- Instalar **VS Code**
  (ou outro editor que se sentir mais confortável)
  https://code.visualstudio.com

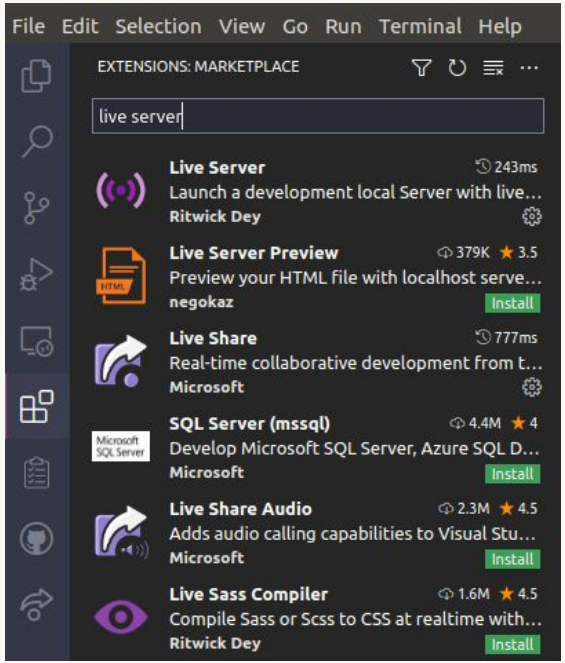
- Sugestão: Instalar extensão **Live Server** no **VS Code**

- Criar um arquivo **index.html** no seu editor

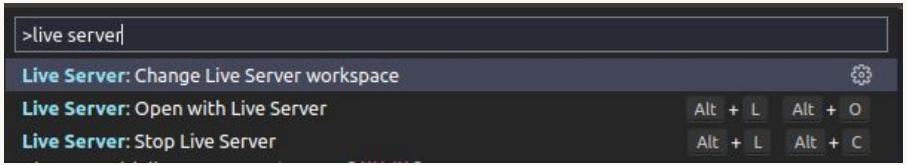




Code Sandbox | https://codesandbox.io
PlayCode | https://playcode.io/new
CodePen | https://codepen.io/pen
JSFiddle | https://jsfiddle.net

# PARA A MÃO NA MASSA




- **Ctrl** + **Shift** + **P**
  Live Server: Open with Live Server

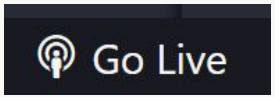



Start

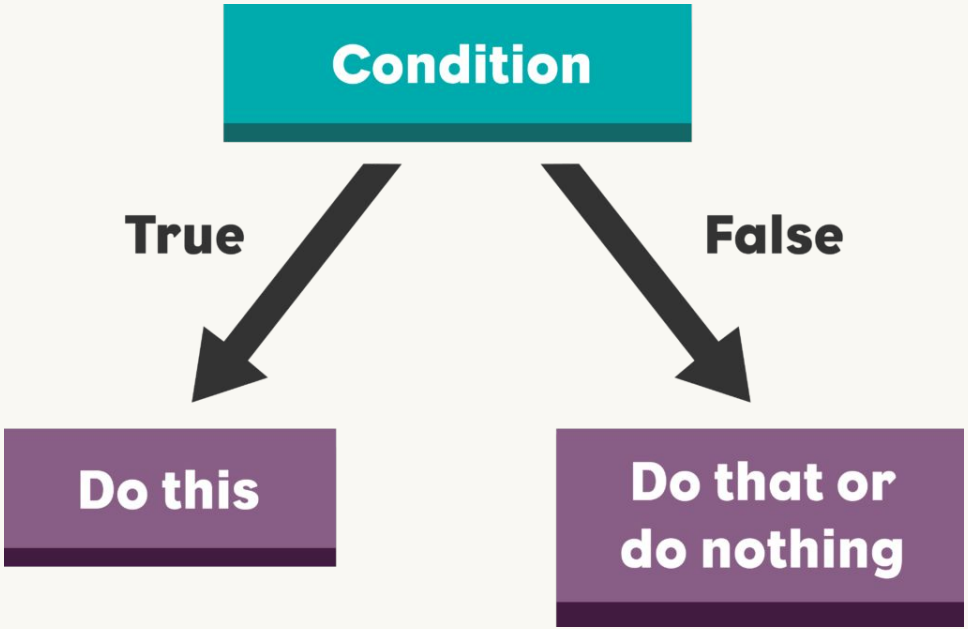
- **Alt** + **L**
- **Alt** + **O**

Stop

- **Alt** + **L**
- **Alt** + **C**

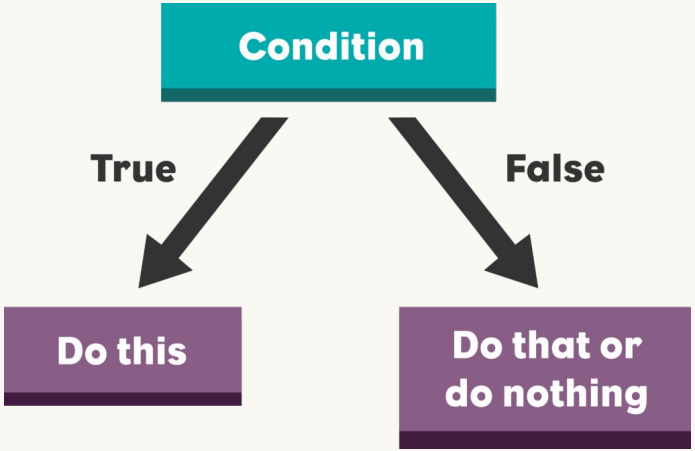



# ESTRUTURAS DE CONTROLE DE FLUXO



# ESTRUTURAS DE CONTROLE DE FLUXO

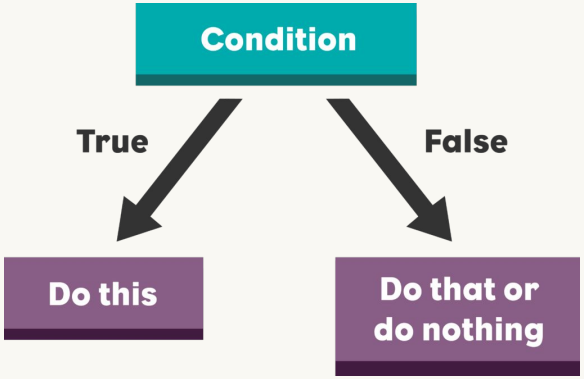
- **Estruturas de controle de fluxo**
  - Execução condicional de blocos de código



# ESTRUTURAS DE CONTROLE DE FLUXO

Até agora vimos execuções sequenciais, uma instrução executa após a outra, sem pular nenhuma, na ordem em que foram escritas.

- **E se eu quiser executar uma das instruções apenas em uma condição específica?**



# ESTRUTURAS DE CONTROLE DE FLUXO

```
var nome =  'Ada';
var sobrenome = 'Lovelace';
var nomeCompleto = nome + ' ' + sobrenome;
var idade = 36;
alert(`A usuária ${nomeCompleto} é adulta.`);
```

- **Exemplo:** Verificar a idade antes de executar o alert e alterar o texto a ser exibido, caso o usuário seja menor de idade.

# ESTRUTURAS DE CONTROLE DE FLUXO | if ... else ...

```
var nome =  'Ada';
var sobrenome = 'Lovelace';
var nomeCompleto = nome + ' ' + sobrenome;
var idade = 36;

if (idade >= 18) {
  alert(`A usuária ${nomeCompleto} é adulta.`);
} else {
  alert(`A usuária ${nomeCompleto} é menor de idade.`);
}
```

- A estrutura de controle de fluxo mais básica que podemos utilizar é a **`if`** / **`else`**.

# ESTRUTURAS DE CONTROLE DE FLUXO | if … else …

- A palavra reservada **`if`** seguida de um par de parênteses. *`if ()`*
- Dentro dos parênteses podemos colocar qualquer condição (teste) a ser verificada (ex.: valor a é maior que b, é diferente, é zero, é igual)
- Essa condição pode empregar os operadores condicionais que aprendemos na última semana, assim como os operadores lógicos **`or`** (ou `||`) e **`and`** (e &&)
- Após isso, utilizamos um par de chaves para delimitar um bloco de código que só será executado se a condição do **`if`** for equivalente a **`true`** (verdadeira).

# ESTRUTURAS DE CONTROLE DE FLUXO | if … else …

- A palavra reservada **`else`** seguida de um par de chaves.
- Após o bloco de código do **`if`** podemos definir um bloco de código a ser executado caso a condição do **`if`** não seja equivalente a **`true`**.
- `if (condição) { executa isso } else { executa isso }`
  `se (condição) { executa isso } senão { executa isso }`

# ESTRUTURAS DE CONTROLE DE FLUXO | if ... else ...

```
var nome =  'Ada';
var sobrenome = 'Lovelace';
var nomeCompleto = nome + ' ' + sobrenome;
var idade = 36;

if (idade >= 18) {
  alert(`A usuária ${nomeCompleto} é adulta.`);
}
alert(`A usuária ${nomeCompleto} é menor de idade.`);
```

- O uso do **else** não é obrigatório, mas devemos prestar atenção na sua necessidade. No caso do exemplo, o **alert** pode executar duas vezes.

# ESTRUTURAS DE CONTROLE DE FLUXO | if ... else ...

**Quando uma condição é `true`?**

- Quando o resultado da condição for **diferente** de qualquer um desses:
  - **`false`**, **`0`** (zero), **`null`**, **`undefined`**, **`NaN`**, **`""`** (string vazia)
- Quando o resultado for **`true`** ou um número positivo ou negativo, um objeto, uma string com pelo menos uma letra, teremos então nossa condição satisfeita
- Podemos também simplesmente colocar o nome de uma variável para testar

```
false
0 (zero)
null
undefined
NaN
'' (string vazia)
// equivalentes a false
```

# ESTRUTURAS DE CONTROLE DE FLUXO | if ... else ...

- Exemplos

```
if (nomeCompleto) {
  alert('O nome é: ' + nomeCompleto);
}
```

```
if (!sobrenome) {
  alert('Informe também o sobrenome');
}
```

# ESTRUTURAS DE CONTROLE DE FLUXO | if ... else ...

- Exemplos

```
if (nomeCompleto) {
  alert(`O nome completo é: ${nomeCompleto}`);
} else {
  alert('Nome não informado');
}
```

# ESTRUTURAS DE CONTROLE DE FLUXO | if ... else ...

- Exemplos

```
if (a > b) {
  alert(`${a} é maior que ${b}`);
} else {
  alert(`${a} não é maior que ${b}`);
}
```

```
if (a > c && b > c) {
  alert(`${a} e ${b} são maiores que ${c}`);
} else {
  alert(`${a} e ${b} não são maiores que ${c}`);
}
```

# ESTRUTURAS DE CONTROLE DE FLUXO | if ... else ...

- E se quisermos mais de duas opções? Podemos utilizar **`if`** dentro de **`if`**

```
if (idade >= 18) {
  if (idade < 60) {
    alert(`A usuária ${nomeCompleto} é adulta.`);
  } else {
    alert(`A usuária ${nomeCompleto} é idosa.`);
  }
} else {
  alert(`A usuária ${nomeCompleto} é menor de idade.`);
}
```

# ESTRUTURAS DE CONTROLE DE FLUXO | if ... else ...

- Mas também podemos utilizar **`else if`**

```
if (idade >= 18 && idade < 60) {
  alert(`A usuária ${nomeCompleto} é adulta.`);
} else if (idade >= 60) {
  alert(`A usuária ${nomeCompleto} é idosa.`);
} else {
  alert(`A usuária ${nomeCompleto} é menor de idade.`);
}
```

# ESTRUTURAS DE CONTROLE DE FLUXO | if ... else ...

- Uma maneira mais otimizada de obter o mesmo resultado com **`else if`**

```
if (idade >= 60) {
  alert(`A usuária ${nomeCompleto} é idosa.`);
} else if (idade >= 18) {
  alert(`A usuária ${nomeCompleto} é adulta.`);
} else {
  alert(`A usuária ${nomeCompleto} é menor de idade.`);
}
```

# ESTRUTURAS DE CONTROLE DE FLUXO

**E se quisermos testar valores exatos ao invés de intervalos?**

- Para grandes quantidades de possibilidades **if...else...** acaba se tornando muito verboso

**Existe outra estrutura?**

```
if (a === 0) {
  alert('zero');
} else if (a === 1) {
  alert('um');
} else if (a === 2) {
  alert('dois');
} else if (a === 3) {
  alert('três');
} else {
  alert('diferente de 0, 1, 2 e 3');
}
```

# ESTRUTURAS DE CONTROLE DE FLUXO | switch

**Para estes casos, podemos utilizar a estrutura switch:**

- Utilizamos a palavra reservada **`switch`** seguida de um par de parênteses como o **`if`**, com alguma variável ou operação dentro.
- Para cada caso (**`case`**), colocamos o valor que esperamos após a palavra reservada **`case`**, seguido de dois pontos.
- Serão executadas todas instruções após os dois pontos, até ser encontrada a palavra **`break`**.

```
switch (num) {
  case 0:
    alert('zero');
    break;
  case 1:
    alert('um');
    break;
  case 2:
    alert('dois');
    break;
  case 3:
    alert('três');
    break;
  default:
    alert('diferente de 0, 1, 2 e 3');
}
```

# ESTRUTURAS DE CONTROLE DE FLUXO | switch

- Caso nenhum dos **cases** seja satisfeito, serão executadas as instruções do caso **default**.
- Se não colocarmos as instruções **break**, a execução do caso não será interrompida e seguirá executando até o final do bloco **switch**, ou até encontrar algum **break**.
- No exemplo ao lado:
  - Caso **num** seja **0** (zero), todos **alert** serão executados
  - Caso **num** seja **2** (dois), os três últimos **alert** serão executados

```
switch (num) {
  case 0:
    alert('zero');

  case 1:
    alert('um');

  case 2:
    alert('dois');

  case 3:
    alert('três');

  default:
    alert('diferente de 0, 1, 2 e 3');
}
```

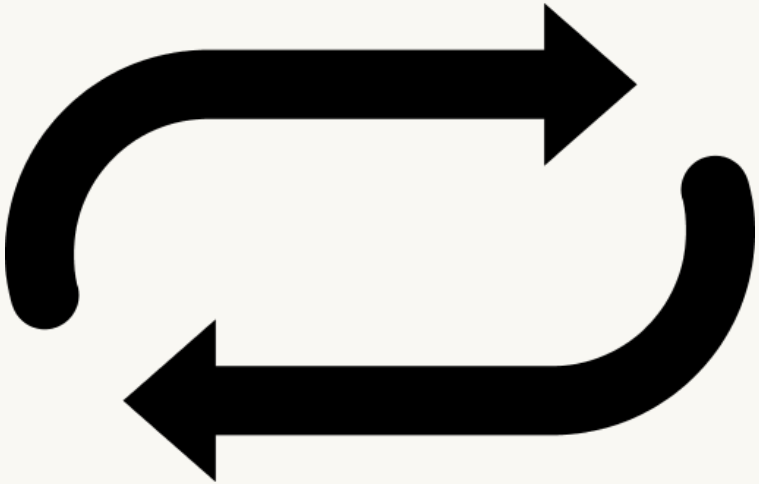
# ESTRUTURAS DE CONTROLE DE FLUXO | switch

```
if (a === 0) {
  alert('zero');
} else if (a === 1) {
  alert('um');
} else if (a === 2) {
  alert('dois');
} else if (a === 3) {
  alert('três');
} else {
  alert('diferente de 0, 1, 2 e 3');
}
```

```
switch (num) {
  case 0:
    alert('zero');
    break;
  case 1:
    alert('um');
    break;
  case 2:
    alert('dois');
    break;
  case 3:
    alert('três');
    break;
  default:
    alert('diferente de 0, 1, 2 e 3');
}
```

Dois códigos diferentes, mas com o mesmo resultado prático

# ESTRUTURAS DE REPETIÇÃO



# ESTRUTURAS DE REPETIÇÃO

Às vezes nos deparamos com situações em que precisamos realizar uma **mesma ação várias vezes**.

**Exemplo:** Exibir na tela todos os elementos de uma lista/array.

```
const listaDeCarros = ['uno', 'fusca', 'escort', 'gol'];

console.log(listaDeCarros[0]);
console.log(listaDeCarros[1]);
console.log(listaDeCarros[2]);
console.log(listaDeCarros[3]);
```

# ESTRUTURAS DE REPETIÇÃO

- Mas não é sempre que sabemos exatamente quantas vezes queremos repetir essa ação.
- Muitas vezes, a quantidade de repetições que precisamos fazer é dinâmica.
- Nessas ocasiões, podemos utilizar as estruturas de repetição que as linguagens de programação nos disponibilizam.

# ESTRUTURAS DE REPETIÇÃO

Alguém sabe dar exemplos de estruturas de repetição?

# ESTRUTURAS DE REPETIÇÃO

- **while** ("enquanto")
- **for** ("para")

# ESTRUTURAS DE REPETIÇÃO | while

Exemplo de **while**:

```
const listaDeCarros = ['uno', 'fusca', 'escort', 'gol'];
let cont = 0;

while (cont < 4) { // ENQUANTO a condição entre parênteses for VERDADEIRA
  alert(listaDeCarros[cont]); // EXECUTA o bloco de instruções entre chaves
  cont++;
}
```

E se a lista sofrer alterações, e novos carros forem adicionados?

# ESTRUTURAS DE REPETIÇÃO | while

Melhorando nosso **while**:

```
const listaDeCarros = ['uno', 'fusca', 'escort', 'gol', 'hb20', 'corsa'];
let cont = 0;

while (cont < listaDeCarros.length) { // ENQUANTO a condição for VERDADEIRA
  alert(listaDeCarros[cont]); // EXECUTA o bloco de instruções entre chaves
  cont++;
}
```

Agora, nossa repetição é dinâmica, variando de acordo com o tamanho da lista.

# ESTRUTURAS DE REPETIÇÃO | do while

**While** invertido:

```
const listaDeCarros = ['uno', 'fusca', 'escort', 'gol', 'hb20', 'corsa'];
let cont = 0;

do { // EXECUTA o bloco de instruções entre chaves
  console.log(listaDeCarros[cont]);
  cont++;
} while (cont < listaDeCarros.length) // ENQUANTO a condição for VERDADEIRA
```

Primeiro executa o bloco de código,
depois testa a condição de repetição.

# ESTRUTURAS DE REPETIÇÃO | for

For é uma estrutura de repetição que já contém uma variável de controle (o “**cont**” que utilizamos nos exemplos anteriores).

Exemplo:

```
const listaDeCarros = ['uno', 'fusca', 'escort', 'gol', 'hb20', 'corsa'];
// (início ; condição_de_repetição ; incremento)
for (let i = 0; i < listaDeCarros.length; i++) { // EXECUTA o bloco de instruções
  console.log(listaDeCarros[cont]);
  // ao FINAL de CADA repetição executa o código em INCREMENTO e TESTA a CONDIÇÃO
}
```

Facilita a escrita quando precisarmos de uma variável de controle para definir o final da nossa repetição.

# ESTRUTURAS DE REPETIÇÃO | for in

For In é um método de iteração em objetos, vetores e strings.

```
const listaDeCarros = ['uno', 'fusca', 'escort', 'gol'];

for (let index in listaDeCarros) { // index = índice do ítem do vetor
  console.log(listaDeCarros[index]);
}
```

```
const listaDeCarros = { marca: 'ford', modelo: 'escort', ano: '1996' };

for (let key in listaDeCarros) { // key = chave do ítem do objeto
  console.log(`${chave} - ${listaDeCarros[key]}`);
}
```

# ESTRUTURAS DE REPETIÇÃO | for of

- **break** "quebra" o loop, "sai fora"
- **continue** apenas "quebra" a iteração atual, continuando o loop

```
const listaDeCarros = ['uno', 'fusca', 'escort', 'gol'];

for (let carro of listaDeCarros) {
  if (carro === 'uno') continue;
  if (carro === 'escort') break;
  console.log(carro);
}
```

# ESTRUTURAS DE REPETIÇÃO | forEach

ForEach é um método de iteração em arrays (vetores).

```
const listaDeCarros = ['uno', 'fusca', 'escort', 'gol', 'hb20', 'corsa'];

listaDeCarros.forEach(
  function (nomeDoCarro, indice) {
    console.log(`${indice} - ${nomeDoCarro}`);
  }
)
```

Facilita a escrita do código quando não precisamos manipular a variável de controle.

# MATERIAL COMPLEMENTAR

JavaScript if else (tutorial) | https://youtu.be/IsG4Xd6LlsM

Condições (Parte 1) - Curso JavaScript #11 | https://youtu.be/cOdG4eACN2A

Condições (Parte 2) - Curso JavaScript #12 | https://youtu.be/EEStcIe8rAM

Condições (Parte 2) - Curso JavaScript #12 | https://youtu.be/EEStcIe8rAM

JavaScript Loops | https://youtu.be/s9wW2PpJsmQ

Repetições (Parte 1) - Curso JavaScript #13 | https://youtu.be/5rZqYPKIwkY

Repetições (Parte 2) - Curso JavaScript #14 | https://youtu.be/eX-lkN_Zahc

for in / for of - Beau teaches JavaScript | https://youtu.be/a3KHBqH7njs

8 Formas de usar Looping em Arrays no JavaScript | https://youtu.be/NfHVPEzo5Ik

# MATERIAL COMPLEMENTAR

JavaScript Conditionals: The Basics with Examples | https://www.javascript.com/learn/conditionals

Tomando decisões no seu código | developer.mozilla.org/docs/Learn/JavaScript/Building_blocks/conditionals

JavaScript if else else if | https://www.w3schools.com/js/js_if_else.asp

JavaScript Switch Statement | https://www.w3schools.com/js/js_switch.asp

Laços e iterações | https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Loops_and_iteration

Estruturas condicionais e de repetição em JS | treinaweb.com.br/blog/estruturas-condicionais-e-estruturas-de-repeticao-em-javascript

JavaScript While Loop | https://www.w3schools.com/js/js_loop_while.asp

JavaScript For Loop | https://www.w3schools.com/js/js_loop_for.asp

JavaScript For In | https://www.w3schools.com/js/js_loop_forin.asp

JavaScript For Of | https://www.w3schools.com/js/js_loop_forof.asp

JavaScript Break and Continue | https://www.w3schools.com/js/js_break.asp

# DEVinHouse

Parcerias para desenvolver a sua carreira

## OBRIGADO!

SENAI

<LAB365>