Java Script

Sumário

Variáveis

São espaços em memórias que nos possibilitam armazenar informações por tempo determinado. Possuem tipos distintos, em Java Script são: *string*, *number* e *boolean*.

Em Java Script existe Case Sensitive, ou seja, maiúsculas e minúsculas fazem diferença.

```
var texto = "Teste de variável" //String
var numero = 2.3 //Number
var booleano = true //Boolean
```

No Java Script existe dois tipos de resultado primitivos importantes para variáveis: *null* e *undefined*. O null representa ausência intencional de um valor. Já o undefined, de fato, não possui nenhum valor.

If e Else

É uma estrutura de controle, que nos possibilita, de acordo com uma condição, qual será o fluxo do código. Ela se inicia com a palavra reservada If e em seguida sua condição.

```
if (condicao) {
    //Trecho de código a ser executado
} else {
    //Trecho de código a ser executado
}
```

Existe também os operadores de comparação ou condicionais, veja os oito operadores: igual (==), idêntico (===), diferente (!=), não idêntico (!==), menor (<), maior (>), menor igual (<=) e maior igual (>=).

```
var numero = 5;
if (numero >= 2 ) {
    console.log("Número maior ou igual a 2");
} else {
    console.log("Número menor que 2");
}
```

Operadores lógicos

Existe três tipos de operadores lógicos, eles nos permitem conectar operações de comparação. São eles:

- E (&&) = verdadeiro se todas as expressões forem verdadeiras;

- Ou (||) = verdadeiro se pelo menos umas das expressões for verdadeira;

- Negação (!) = inverte o resultado da expressão de comparação.

```
var faltas = 5;
var media = 8

if(media >= 7 && faltas < 5) {
    console.log("Aprovado"); //Precisa obedecer a lógica de:
                             // média maior ou igual a 7 e
                             //ter faltado menos que 5 vezes
} else {
    console.log("Reprovado");
}
```

Neste exemplo o aluno será reprovado por ter faltado 5 vezes, logo o operador E (&&) não pode ser declarado *true*, devido ao fato de somente uma expressão ser verdadeira.

Existe uma estrutura de decisão com uma sintaxe menor ele é chamado de operador ternário. Ele pode substituir a estrutura de decisão *if/else.* A estrutura é a seguinte

<condição> ? <verdadeiro> : <falso>

```
var res = (media >= 7 && faltas < 5) ? "Aprovado" : "Reprovado";
```

Lembrando que essa estrutura não aceita lógicas complexas.

Operadores Aritméticos

Operadores aritméticos são aqueles que aprendemos na escola, porém no Java Script temos alguns diferentes. São eles: adição (+), subtração (-), multiplicação (*), divisão (/), * módulo (%), incremento (++) e decremento (- -).

O módulo (%) é o resto existente em uma operação de divisão. Ou seja, 4 % 2 resultará em 0 pois nessa operação o resto dessa divisão é 0.

No Java Script também existe a precedência de operadores. Os parênteses também acionam a precedência.

- 1° = multiplicação e divisão

- 2° = soma e subtração

Conversão de tipos

Existe três tipos de métodos que faz com que altere o tipo do resultado da variável, são eles:

- .toString(resultado) = transforma o resultado da variável em string;

- parseInt(resultado) = transforma o resultado da variável em número inteiro;

- parseFloat(resultado) = transforma o resultado da variável em números de ponto flutuante.

```
var n1 = 20
console.log(n1.toString()) //Transformará o 20 em string
var n2 = parseInt(20) //Confirmará a ideia de número inteiro
var n3 = parseFloat //Transfomará em número de ponto flutuante
```

Switch

O *switch* é um condicional, e a partir disso segue diferentes tipos de caminhos. Ele é bem parecido com o *if/else* porém a escrita do comando *switch* é muito mais intuitiva.

```
var opcao = 2
switch (opcao) {
    case 1:
        //Trecho de cógido que será executado
        break;
    case 2:
        //Trecho de cógido que será executado
        break;
    default:
        //Trecho de cógido que será executado
        break;
}
```

Deve-se ter atenção a dois pontos, ao *break* que fará com que não haja uma repetição infinita daquele bloco de código e ao *default* que exibirá um código de valor padrão, caso não tenha nenhum *case* para aquela condição enviada por parâmetro.

Funções

Uma função tem o objetivo de encapsular um trecho de código, que pode ser usado infinitas vezes durante o código somente chamando a função, sem a necessidade de reescrever todo o bloco de código. Existe dois tipos de função, a *void,* que quando chamada irá apenas processar alguma lógica, já a de retorno, que, além de processar uma lógica, também trará um retorno de um valor.

```
function nomeDaFuncao(parametros) {
    //Trecho de código que a função executará
}
```

Os parâmetros ou argumentos são uma entrada de dados para a função, pode ser passado quantos parâmetros quiser ou até mesmo nenhum parâmetro. Os eles são uma espécie de variável, por isso a nomenclatura deve seguir o mesmo padrão.

```
function calcularArea(largura, comprimento) {
    var area = largura * comprimento;

    return area;
}
```

Para a função ser devidamente executada ela precisar sem chamada, isso pode ser feito em qualquer lugar do código. A chamada pode ser feita infinitas vezes e também antes do bloco da função.

```
calcularArea(2,3); //2 = largura e 3 = comprimento
```

Porém nessa função não aparecerá nada para o usuário e para fazer isso terá que implementar algo para que esse valor 'area' apareça para o usuário.

```
function calcularArea(largura, comprimento) {
    var area = largura * comprimento;

    return document.write('A área é de ' + area + ' m²'); //aparecerá
    o resultado para o usuário
}

calcularArea(2,3); //2 = largura e 3 = comprimento
```

A área é de 6 m²

Funções anônimas e técnicas de Wrapper

Funções anônimas são aquelas que não possuem nomes, a sintaxe e o funcionamento são os mesmo de uma função comum. Geralmente elas são atribuídas a alguma variável e isso é chamado de técnica de Wrapper.

```
var saudacao = function(nome) {
    console.log("Olá, " + nome)
}

saudacao("Allan") //Chamar a variável que tem uma função anônima
atribuida
```

Funções de Callback

Elas são funções que são encaminhadas como parâmetros para outras funções.

```
function cadastrarUsuaurio(usuario, senhaSucesso, senhaErro) {
    //lógica de cadastrar o usuário e senha
    //Tratar se senha cumpre os requisitos de segurança
    if (logica == true) {
        //se cumpre os requisitos
        senhaSucesso('Senha válida');
    }else {
        //se não cumpre os requisitos
        senhaErro('Senha inválida pois...');
    }
}
var senhaSucesso = function(validar) {
    //lógica para a senha ser válida
}
var senhaErro = function(erro) {
    //lógica para a senha ser inválida
}

cadastrarUsuaurio(1, senhaSucesso, senhaErro);
```

Na chamada da função 'cadastrarUsuario' os parâmetros das funções *callback* não recebem parênteses pois se for utilizado, indicará que ela deverá ser executada quando for passada como parâmetro, mas queremos que a função seja somente encaminhada e não executada.

Veja este vídeo para facilitar o entendimento.

Funções nativas para manipulação de strings

Essas funções são nativas, ou seja, você não precisa desenvolver uma lógica para elas, pois o Java Script faz isso automaticamente. Para entender um pouco um pouco dessas funções, deve-se ter em mente que cada caractere (até mesmo os espaços entre letras e palavras) ocupam uma posição sequencial específica (índice) e sempre começando pelo 0.

A	**L**	**L**	**A**	**N**		**C**	**O**	**S**	**T**	**A**
0	1	2	3	4	5	6	7	8	9	10

Confira a lista completa aqui.

Propriedade:

- length – retorna a quantidade exata de todos os caracteres de uma string (espaço inclusive).

```
var nome = 'Allan Costa'
console.log(nome.length)
```

Métodos:

* Métodos precisam ter () para funcionar.

- charAt(índice) – retorna um caractere específico no index (posição na cadeia de caracteres);

- indexOf() – retorna o índice da primeira ocorrência dentro da cadeia de caracteres. Caso não tenha o caractere na cadeia de string, o valor retornado será -1.

```
var nome = 'Allan Costa'
console.log(nome.indexOf('a'))
```

- replace() – você precisa passar um valor de pesquisa e outro de modificação, assim o texto de pesquisa será trocado pelo de modificação na cadeia de caracteres.

```
var nome = 'Allan Costa'
console.log(nome.replace('Costa', 'Vigiani'))
```

- toLowerCase() – retorna toda a string e converte em letras minúsculas.

- toUpperCase() – retornar toda a string e converte em letras maiúsculas.

- trim() – remove os espaços em branco nas extremidades da string.

Funções nativas para tarefas matemáticas

Segue o mesmo padrão do tópico anterior e são definidos através do método Math. Por ter uma quantidade muito grande de métodos, deixarei um link do MDN com todos eles.

Métodos Math no Java Script

Funções nativas para manipulação de datas

Como são funções nativas essas seguem os mesmos padrões das anteriores, a única diferença é que se utiliza a palavra Date.

Lista de todos os métodos Date

A única diferença é que essas funções precisam ser instanciadas. Veja como é feita.

```
var data = new Date() //Instancia o Date() a uma variável
console.log(data.getDate())
```

Eventos

Eventos possibilitam captar ações e comportamentos do usuário, e com isso diversas ações e animações podem ser feitas. O mais comum é que através desses eventos, funções serem chamadas e disparar algum tipo de lógica. É importante consultar a documentação para que se tenha conhecido de quais tags aceitam o evento escolhido.

Mouse

Alguns desses eventos podem ser ativados através de comandos feitos pelo mouse. A seguir uma lista será disponibilizada com todos os eventos de mouse existentes.

Eventos de mouse Java Script

Os mais usados são:

- onclick = é ativado quando o usuário clica com o botão esquerdo;
- ondblclick = é ativado quando o usuário dispara um duplo clique;
- onmouseover = é ativado quando o usuário põe o mouse em cima do elemento;
- onmouseout = é ativado quando o usuário tira o mouse de cima do elemento.

Teclado

São eventos relacionados a alguma ação de alguma tecla do teclado. São três eventos possíveis.

- onkeydown = é disparado quando qualquer tecla do teclado é apertada;
- onkeypress = é disparada quando alguma tecla de caractere é pressionada;
- onkeyup = é acionada quando qualquer tecla é liberada.

Janela

São relacionamos com a manipulação de exibição da janela. Os mais comuns são:

- onresize = acionado quando o tamanho da tela muda (acionado pela tag <body>);
- onscroll = acionado quando a barra de rolagem é ativada.

Formulários

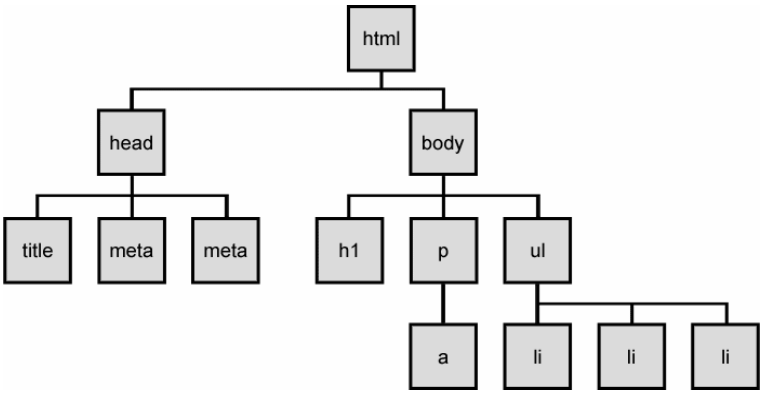
São eventos relacionados ao uso de formulários. Os mais comuns são:

- onfocus = quando recebe o foco do cursor do mouse (clique);
- onblur = quando perde o foco do cursor do mouse;
- onchange = quando o estado do elemento é modificado.

LEMBRE-SE SEMPRE DE CONSULTAR A DOCUMENTAÇÃO PARA VER TODOS OS EVENTOS DE TODOS OS TIPOS.

DOM

DOM é o Document Object Model, ele é uma API multiplataforma que permite o acesso através do Java Script a todos os elementos do HTML da página. Ou seja, acessar e modificar os elementos HTML da página.



Cada elemento HTML é chamado no DOM de nó ou *node*.

Seletores

Existe alguns método que pegam através de Ids, classes, name e outros. Veremos alguns deles:

- getElementById() = somente para a propriedade Id
- getElementsByClasseName() = somente para elementos de propriedade Class
- getElementByName() = somente para elementos de propriedade Name
- querySelector() = serve para qualquer tipo de propriedade

Manipular valores de texto

Existe diversos modos de manipular texto, o mais importante deles é:

- .value = que capta o valor do elemento digitado (em forma de *string*)

Manipular estilos de elementos

Através do atributo .*style* que você conseguirá modificar qualquer coisa baseado nos atributos do CSS desse elemento. Segue a lista completa de todos as propriedades aceitas.

Lista de propriedades DOM style

Array

São listas ordenadas, variáveis que nos permitem associar itens a índices, também conhecido como chaves. Permite o armazenamento em uma única variável N valores.

```
var lista_frutas = Array();
lista_frutas[1] = 'Uva'
lista_frutas[2] = 'Banana'
lista_frutas[3] = 'Maça'
lista_frutas[4] = 'Melancia'
```

Para consulta algum elemento dentro do *Array*, deve ser passado o índice do elemento. Lembrando que o índice de um *Array* não precisa, necessariamente, ser um número

```
console.log(lista_frutas[2])
```

Nesse caso, o item que será retornado é a 'banana', pois ela que está ligada ao índice [2].

Outra forma de escrever um *Array* é usando os colchetes.

```
var lista_coisas =[];
lista_coisas['coisa'] = 2
lista_coisas[143] = 'Banana'
lista_coisas[true] = false
lista_coisas['carro'] = 4.5

console.log(lista_coisas[true])
```

Existe mais duas formas de se escrever um *Array*, porém não é possível manipular o índice.

```
var lista_carros = Array('Gol', 'Toro', 'Prisma');

console.log(lista_carros[0]);
```

```
var lista_carros = ['Gol', 'Toro', 'Prisma'];

console.log(lista_carros[2]);
```

Nestes casos, o próprio Java Script designa os índices em ordem crescente iniciando do 0.

Um ponto importante é o atributo length. Possui diferenças de uma variável do tipo *string* para um objeto do tipo *Array*. Nas variáveis de tipo *string* é simples, retorna o número de elementos dentro da cadeia de caracteres. Já no objeto *Array* ele retornará à quantidade de elementos dentro do *Array*. Deve-se atentar que sempre aparecerá o número posterior do último índice escolhido, uma boa prática é começar pelo índice 0 de forma crescente, pois assim aparecerá a quantidade exata de elementos.

```
var lista_frutas = Array();
lista_frutas[0] = 'Uva'
lista_frutas[1] = 'Banana'
lista_frutas[2] = 'Maça'
lista_frutas[3] = 'Melancia'

console.log(lista_frutas.length)
//Resultará em 4 elementos
```

Array Multidimensional

São os conhecidos como *Array* de *Array, ou seja*, encadeamento de *Array*.

```
var lista = Array();

lista['objetos'] = Array()
lista['objetos'][0] = 'colher'
lista['objetos'][1] = 'faca'
lista['objetos'][2] = 'garfo'

lista['comida'] = Array()
lista['comida'][0] = 'arroz'
lista['comida'][1] = 'feijao'
lista['comida'][2] = 'salada'
lista['comida'][3] = 'pudim'

console.log(lista)
```

```
[
  objetos: [ 'colher', 'faca', 'garfo' ],
  comida: [ 'arroz', 'feijao', 'salada', 'pudim' ]
]
```

E também existindo as duas formas de escreves os *Arrays*.

```
var lista = Array();

lista['objetos'] = Array('colher', 'faca', 'garfo')

lista['comida'] = ['arroz', 'feijao', 'salada', 'pudim']
```

Inclusão e Exclusão de elementos

- push() = incluir o elemento no final do *Array;*

- unshift() = incluir o elemento do início do *Array*;

- pop() = excluir o elemento do final do *Array;*

- shift() = excluir o elemento do início do *Array.*

Métodos de pesquisa

Servem para buscar se existe um elemento específico dentro de um *Array* e com base nisso executar alguma lógica específica.

- indexOf(elemento) = busca determinado elemento dentro de um *Array*, caso ele não seja encontrado, o retorno será -1.

Ordenação de elementos

- sort() = ordena os elementos e os índices de forma alfanumérica.

- sort(ordenarNumeros) = uma função callback é passada para dentro do sort(), segue a função.

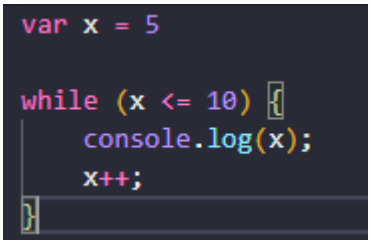
```
sort(ordenarNumeros)

function ordenarNumeros(a, b) {
    //indíce anterior menos o posterior
    return a - b;
    // resultado < 0 == a ordenado antes de b
    // resultado > 0 == a ordenado depois de b
    // resultado == a ordem é mantida
}
```

Estruturas de repetição

São estruturas, também chamados de laços ou *loops*, que permitem a repetição de um comando ou um bloco de código, até atender a uma determinada condição.

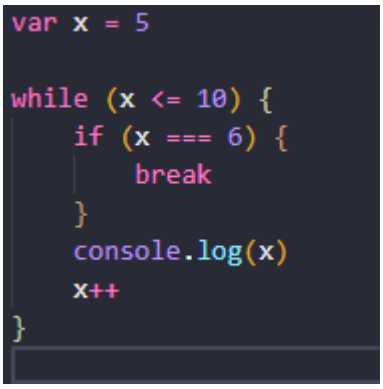
While



Nesse caso específico, o *While* fará uma contagem do valor inicial da variável 'x', que é 5, até que cumpra a condição posta, no caso, 'x <= 10'. O resultado será: 5,6,7,8,9 e 10.

Fique atento com sua condição, pois se for feita de forma errada ela criará um loop infinito e isso acarretará e um erro na sua aplicação.

Nos laços de repetições existem comandos que podem ser utilizados durante a lógica de repetição, um dele é o *break*, que faz com que determinada lógica ou elemento seja interrompido.



Nesta estrutura, o único número que aparecerá no console será o 5, pois o comando *break* designa a parada desde que 'x === 6'.

Existe também o comando *continue*, que finaliza aquele determinado passo e passa para o próximo, não executando somente aquele comando. Nesse comando, deve-se ter muito cuidado, pois o incremento deve vir antes da lógica em que o *continue* está para que não ocasione um loop infinito.

```
var x = 5

while (x <= 10) {
    console.log(x)
    x++

    if (x === 6) {
        continue
    }
}
```

Perceba que o incremento ‘x++’ está localizado antes da lógica que contém o *continue*. Nessa situação o resultado será: 5,7,8,9 e 10 pulando o número 6.

Do While

Este comando é bem parecido com o *While*¸ a única diferença é que a condição e a lógica ficam em locais separados.

```
var x = 5

do {
    console.log(x)
    x++;
}
while(x <= 10)
```

Vejamos o exemplo do *break* nesse novo laço de repetição. Lembrando que o resultado permanece o mesmo.

```
var x = 5

do {
    if (x === 6) {
        break
    }
    console.log(x)
    x++;
}
while(x <= 10)
```

Agora vejamos o *continue*, que se mantem o mesmo resultado.

```
var x = 5

do {
    console.log(x)
    x++;
    if (x === 6) {
        continue
    }
}
while(x <= 10)
```

For

O comando *for* é um pouco diferente dos outros, pois a variável, a condição e a lógica estão juntas.

```
for (var x = 5; x <= 10; x++){
    console.log(x)
}
```

E também é possível usar o *break* e o *continue* na implementação, seguindo as mesmas regras.

Percorrer Array

```
var coisa = ['garfo', 'faca', 'prato']

for (var x = 1; x <= coisa.length; x++){
    console.log(x)
}
```

A partir disso, você pode, por exemplo, saber o tamanho de um *Array.* Ou saber o os itens dentro do *Array*.

```
var coisa = ['garfo', 'faca', 'prato']

for (var x = 0; x < coisa.length; x++){
    console.log(coisa[x])
}
```

For in

Segue a mesma estrutura do *loop for*, porém com algumas mudanças.

```
var convidados = ['Allan', 'Marcelo', 'Carla', 'Ana']

for (var x in convidados) {
    //var x recebe, a cada interação, o valor do indice
    console.log(x)
}
```

Nesse caso, o console.log receberá somente os índices dos objetos dentro do *Array*.

```
var convidados = ['Allan', 'Marcelo', 'Carla', 'Ana']

for (var x in convidados) {
    //var x recebe, a cada interação, o valor do indice
    console.log(convidados[x])
}
```

Já, nesse caso, o resultado resultará nos valores atribuídos os índices. Utilizando o laço *for.. of ..* você terá esse mesmo resultado;

```
var convidados = ['Allan', 'Marcelo', 'Carla', 'Ana']

for (var x of convidados) {
    //var x recebe, a cada interação, o valor do indice
    console.log(x)
}
```

For each

É uma função que é aplicada para *Arrays*. Ele funciona de forma parecida com o *for..in.* Em resumo, executará uma função para cada item do *Array*.

```
var convidados = ['Allan', 'Marcelo', 'Carla', 'Ana']

convidados.forEach(function(valor,indice,array){
    //lógica será executada para cada íncide do Array
    console.log(`valor: ${valor}`)
    console.log(`indice: ${indice}`)
    console.log(`array: ${array}`)
})
```

Atente-se aos parâmetros, pois os nomes podem mudar, mas a ordem sempre será do valor, do índice e do *Array*.

Tratamento de erros

Durante o desenvolvimento é comum ocorrer erros de execução, que são aqueles que interrompem o processamento do *script*. Por isso a importância do tratamento de erros, pois eles não deixam que, mesmo que ocorra o erro, o *script* seja interrompido.

Try, Catch e Finally

A instrução *try* é um bloco e dentro dele deve ir todo código que seja passível de erro e a quantidade de uso é indefinida, ou seja, pode-se usar quantos forem necessários.

O *catch* é implementado logo após o *try* e recebe o erro por parâmetro. Sua função é tratar o possível erro.

O *finally* também é um bloco de códigos e serve para finalizar "alguma coisa de alguma forma".

```
var video = Array()

video[1] = Array()
video[1]["nome"] = 'Se beber não case'
video[1]["categoria"] = 'Comédia'

function pegarVideo() {
    try {
        console.log(video[0]["nome"]) //forçando um erro
    }
    catch(erro){
        console.log(erro)
        console.log("Tratar erro (catch)")
        //Tratar erro de forma criativa
    }
    finally {
        console.log("Passou por aqui (finally)")
    }
}

pegarVideo()
```

```
TypeError: Cannot read property 'nome' of undefined                script2.js:12
    at pegarVideo (script2.js:9)
    at script2.js:21
Tratar erro (catch)                                                script2.js:13
Passou por aqui (finally)                                          script2.js:17
```

Throw

Serve para lançar exceções dentro da aplicação.

```
catch(erro){
    console.log(erro)
    console.log("Tratar erro (catch)")
    throw new Error("Servidor Indisponível! Voltamos em breve.")
    //Tratar erro de forma criativa
}
```

```
▶Uncaught Error: Servidor Indisponível! Voltamos em breve.                script2.js:14
    at pegarVideo (script2.js:14)
    at script2.js:22
```

BOM

Browser Object Model nada mais é que uma forma de se comunicar, através do *Java Script*, com recursos do *browser*.

Window

É toda a janela do *browser*, tudo aquilo que é 'renderizado'. Temos o objeto 'document', que por ele, pode acessar qualquer elemento HTML.

Veja a lista de objetos do Window

```
function abrirPopUp() {
    window.open('http://www.google.com', 'nova_janela', 'width=300 ,
    height=300')
    //url - nome - especificações - substituto
}
```

Screen

Fornece acesso a atributos relativos a tela do *browser*, como largura e altura de onde os elementos HTML são 'renderizados'.

Lista de objetos Screen

```
var altura = window.screen.availHeight
var largura = window.screen.availWidth

document.write("Largura: " + largura)
document.write("Altura: " + altura)
```

Largura: 1360Altura: 738

Location

Fornece acesso a atributos e métodos da URL atual.

Lista de objetos do Location

```
window.location.href="http://www.google.com"
```

No momento que o *Java Script* ler essa linha de código a página será redirecionada para a URL escolhida.

Timming

Existe dois métodos de extrema importância.

- setTimeout(<ação>, <tempo em ms>) = executa uma ação uma vez após tempo informado.
  - clearTimeout() = para a execução.
- setInterval(<ação>, <tempo em ms>) = sempre executa a ação após o tempo informado.
  - clearInterval() = para a execução.

```
setTimeout(function() {
  document.write("Passou 2 segundos")
}, 2000)
```

```
var x = 10

var cronometro = setInterval(function() {
    document.write(x + "<br>");
    x--;
    if(x == 0) {
        clearInterval(cronometro)
    }
}, 1000)
```