

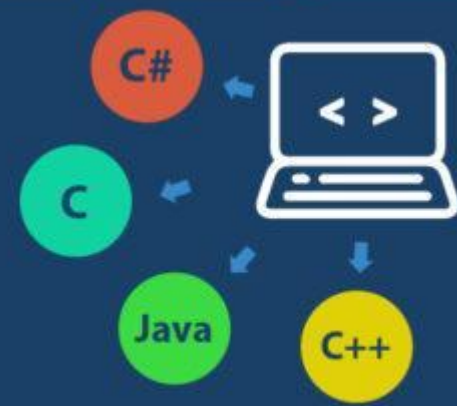
Linguagem de Programação

Professora: Letícia Pieper Jandt

Análise e Desenvolvimento de Sistemas

60 Horas

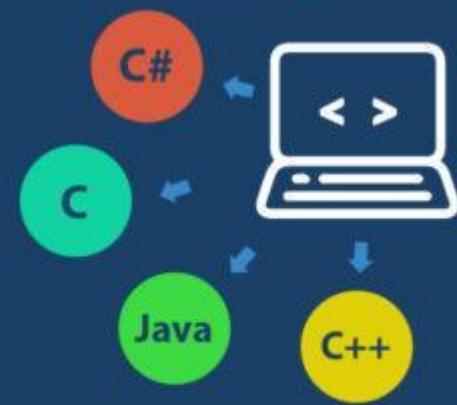
Javascript



JavaScript é uma linguagem de programação que adiciona interatividade ao seu site.

Isso acontece em jogos, no comportamento das respostas quando botões são pressionados ou com entrada de dados em formulários; com estilo dinâmico; com animação, etc.

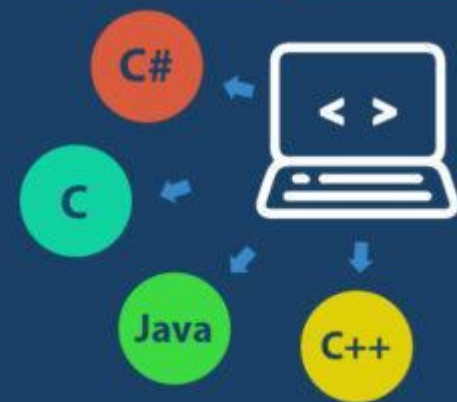
Javascript



O próprio JavaScript é relativamente compacto, mas muito flexível.

Os desenvolvedores escreveram uma variedade de ferramentas sobre a linguagem JavaScript principal, desbloqueando uma grande quantidade de funcionalidades com o mínimo de esforço.

Esses incluem:



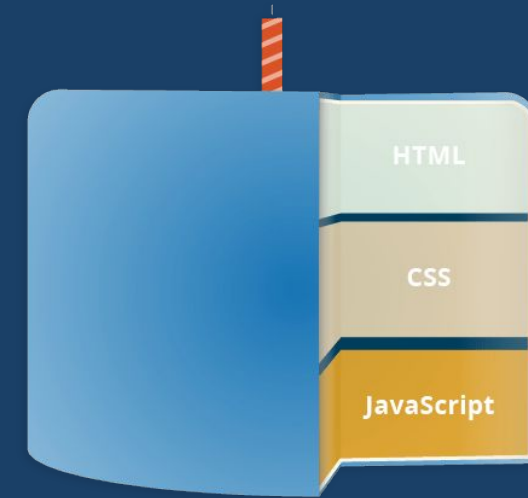
- Interfaces de programação de aplicativos de navegador (APIs) incorporadas a navegadores da Web, fornecendo funcionalidades como criação dinâmica de HTML e definição de estilos CSS; coletar e manipular um fluxo de vídeo da webcam de um usuário ou gerar gráficos 3D e amostras de áudio.
- APIs de terceiros que permitem aos desenvolvedores incorporar funcionalidades em sites de outros provedores de conteúdo, como Twitter ou Facebook.
- Estruturas e bibliotecas de terceiros que você pode aplicar ao HTML para acelerar o trabalho de construção de sites e aplicativos.

Definição de alto nível

JavaScript é uma linguagem de programação que permite a você implementar itens complexos em páginas web.

Toda vez que uma página da web faz mais do que simplesmente mostrar a você informação estática, mostrando conteúdo que se atualiza em um intervalo de tempo, mapas interativos ou gráficos 2D/3D animados, etc.

você pode apostar que o JavaScript provavelmente está envolvido. É a terceira camada do bolo das tecnologias padrões da web (HTML e CSS)



O que JS pode fazer?

Armazenar conteúdo útil em variáveis.

Operações com pedaços de texto (conhecidos como "strings" em programação). – Concatenação.

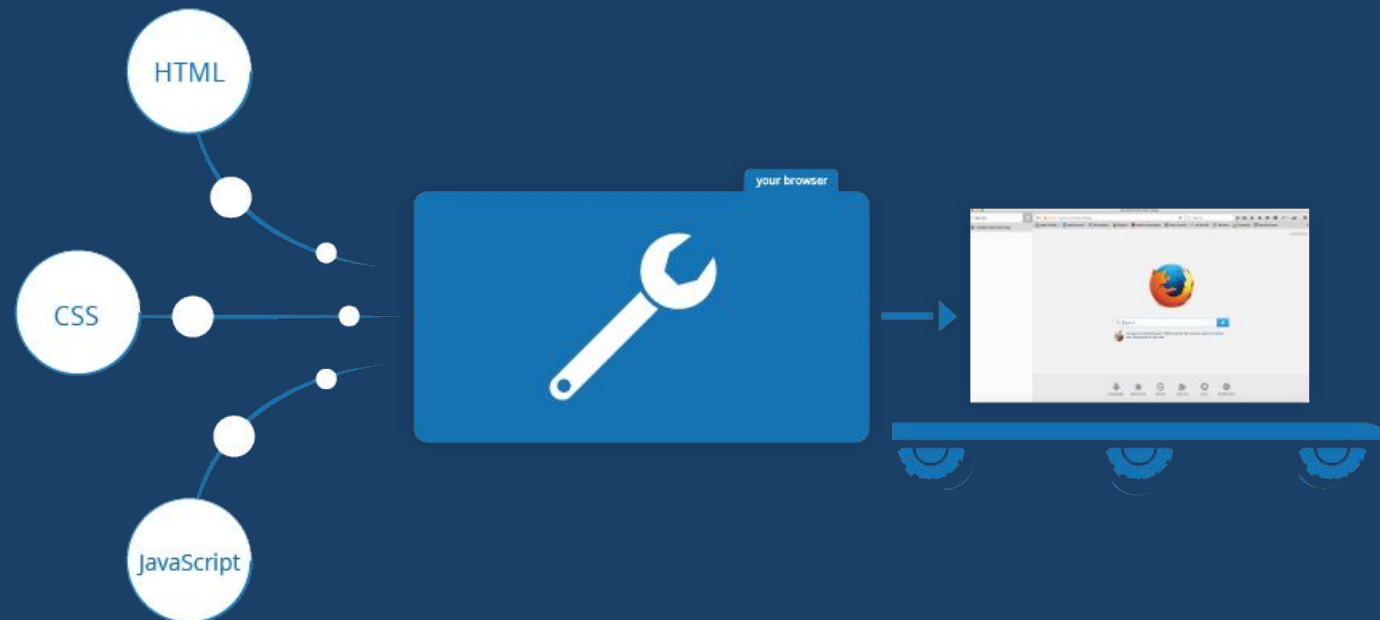
Executar o código em resposta a determinados eventos que ocorrem em uma página da Web.

Manipular jsons, envio de formulários, montar apis...

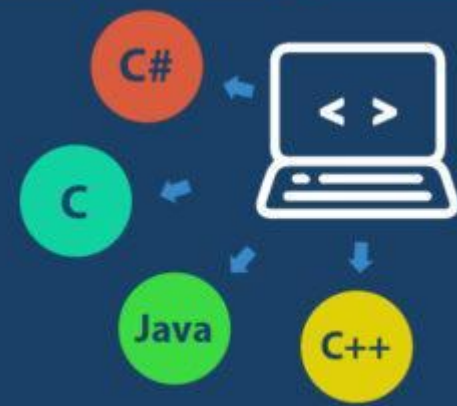
E muito mais!

O que JavaScript está fazendo na sua página web?

Quando você carrega uma página web no seu navegador, você está executando seu código (o HTML, CSS e JavaScript) dentro de um ambiente de execução (a guia do navegador). Isso é como uma fábrica que pega a matéria prima (o código) e transforma em um produto (a página web).



Ordem de execução do JavaScript



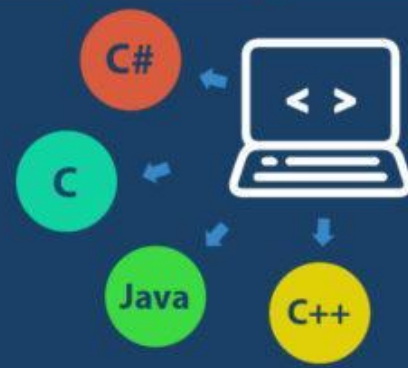
Quando o navegador encontra um bloco de código JavaScript, ele geralmente executa na ordem, de cima para baixo. Isso significa que você precisa ter cuidado com a ordem na qual você coloca as coisas.

Ordem de execução do JavaScript

```
const para = document.querySelector("p");
```

```
para.addEventListener("click", atualizarNome);
```

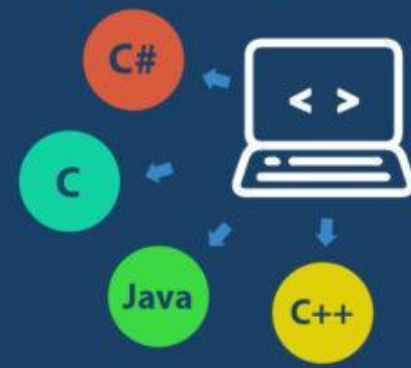
```
function atualizarNome() {  
  let nome = prompt("Informe um novo nome:");  
  para.textContent = "Jogador 1: " + nome;  
}
```



Aqui nós estamos selecionando um parágrafo (linha 1) e anexando a ele um event listener (linha 3). Então, quando o parágrafo recebe um clique, o bloco de código atualizarNome() (linhas 5 a 8) é executado. O bloco de código atualizarNome() pede ao usuário que informe um novo nome, e então insere esse nome no parágrafo, atualizando-o.

Se você inverte a ordem das duas primeiras linhas de código, ele não funcionaria — em vez disso, você receberia um erro no console do navegador — TypeError: para is undefined. Isso significa que o objeto para não existe ainda, então nós não podemos adicionar um event listener a ele.

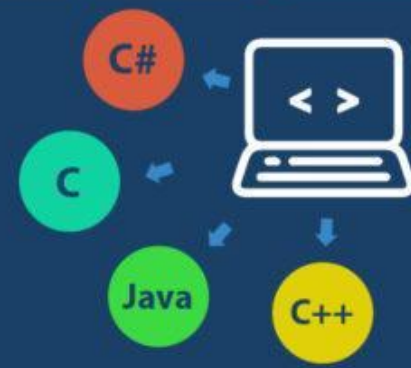
Código interpretado x compilado



JavaScript é uma linguagem interpretada — o código é executado de cima para baixo e o resultado da execução do código é imediatamente retornado. Você não tem que transformar o código em algo diferente antes do navegador executa-lo.

JavaScript é uma linguagem de programação leve e interpretada. O navegador recebe o código JavaScript em sua forma de texto original e executa o script a partir dele.

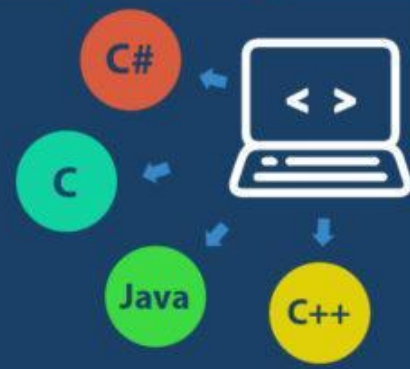
Código interpretado x compilado



Do ponto de vista técnico, a maioria dos intérpretes modernos de JavaScript realmente usa uma técnica chamada compilação just-in-time para melhorar o desempenho; o código-fonte JavaScript é compilado em um formato binário mais rápido enquanto o script está sendo usado, para que possa ser executado o mais rápido possível. No entanto, o JavaScript ainda é considerado uma linguagem interpretada, pois a compilação é manipulada em tempo de execução, e não antes.

Há vantagens em ambos os tipos de linguagem, mas nós não iremos discutir no momento.

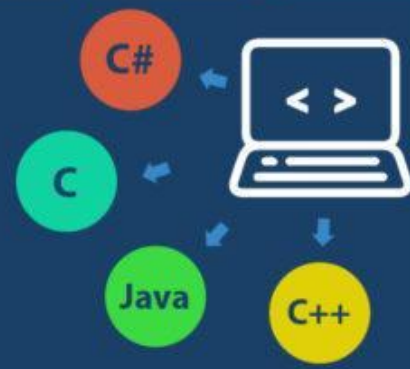
Lado do servidor x Lado do cliente



Códigos do lado do cliente são executados no computador do usuário — quando uma página web é visualizada, o código do lado do cliente é baixado, executado e exibido pelo navegador.

JavaScript também pode ser usada como uma linguagem server-side, por exemplo, no popular ambiente Node.js

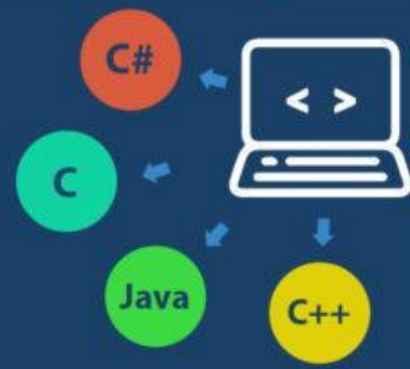
Código dinâmico x estático



A palavra dinâmico é usada para descrever tanto o JavaScript client-side como o server-side — essa palavra se refere a habilidade de atualizar a exibição de uma página web/app para mostrar coisas diferentes em circunstâncias diferentes, gerando novo conteúdo como solicitado.

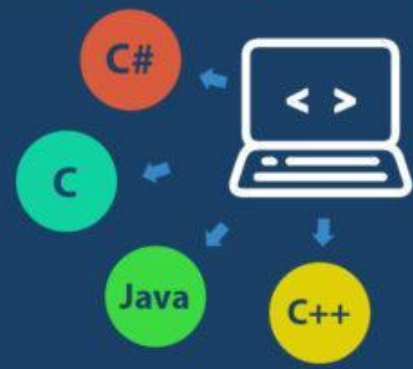
JavaScript do lado do cliente dinamicamente gera novo conteúdo dentro do navegador do cliente, como criar uma nova tabela HTML com dados recebidos do servidor e mostrar a tabela em uma página web exibida para o usuário. Os significados são ligeiramente diferente nos dois contextos, porém relacionados, e ambos (JavaScript server-side e client-side) geralmente trabalham juntos.

Como você adiciona JavaScript na sua página?



O JavaScript é inserido na sua página de uma maneira similar ao CSS. Enquanto o CSS usa o elemento `<link>` para aplicar folhas de estilo externas e o elemento `<style>` para aplicar folhas de estilo internas, o JavaScript só precisa de um amigo no mundo do HTML — o elemento `<script>`.

Como você adiciona JavaScript na sua página?



JavaScript interno

Esse conceito é utilizar o js dentro da própria página html.

Você pode adicionar de duas formas no html.

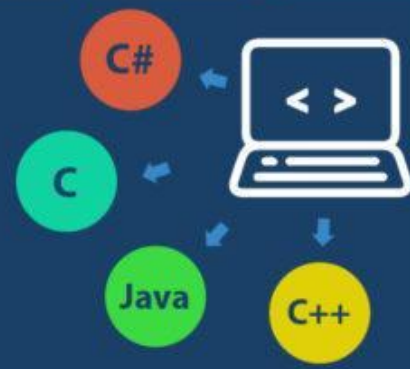
Dentro do head ou antes de `</body>` utilizando a tag script

```
<script>
```

```
// O JavaScript fica aqui
```

```
</script>
```

Como você adiciona JavaScript na sua página?



JavaScript externo

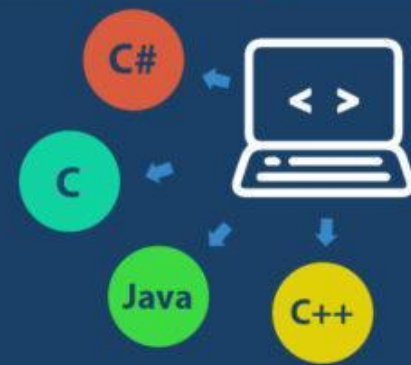
Esse conceito é criar um arquivo externo e vincular ao html.

Precisamos de um arquivo com arquivo tem a extensão .js, pois é assim que ele será reconhecido como JavaScript.

Para linkar no html utilizamos dessa forma:

```
<script src="script.js" defer></script>
```


Variáveis

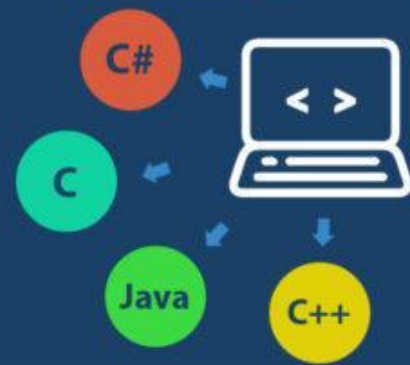


O nome das variáveis, chamados de identificadores, obedecem determinadas regras.

Variáveis devem começar com uma letra, underline (_), ou cifrão (\$); os caracteres subsequentes podem também ser números (0-9). Devido JavaScript ser case-sensitive, letras incluem caracteres de "A" a "Z" (maiúsculos) e caracteres de "a" a "z" (minúsculos).

Alguns exemplos de nomes legais são `Numeros_visitas`, `temp99`, e `_nome`.

Variáveis



Declarações – Existem três tipos de declarações em JavaScript.

`var`

Declara uma variável, opcionalmente, inicializando-a com um valor.

`let`

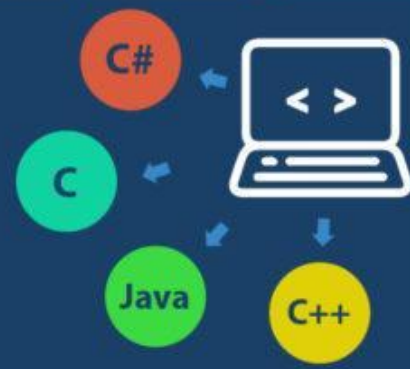
Declara uma variável local de escopo do bloco, opcionalmente, inicializando-a com um valor.

`const`

Declara uma constante de escopo de bloco, apenas de leitura.

Variáveis

Declarando variáveis



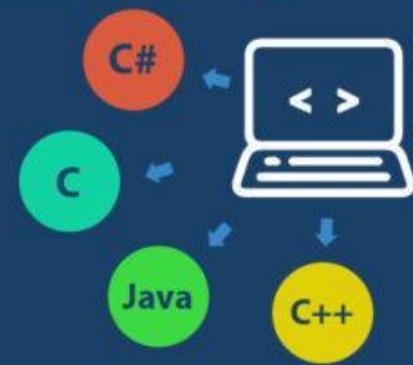
Você pode declarar uma variável de três formas:

Com a palavra chave `var`. Por exemplo, `var x = 42`. Esta sintaxe pode ser usada para declarar tanto variáveis locais como variáveis globais.

Com a palavra chave `let`. Por exemplo, `let y = 13`. Essa sintaxe pode ser usada para declarar uma variável local de escopo de bloco.

Variáveis

Escopo de variável



Quando você declara uma variável fora de qualquer função, ela é chamada de ***variável global***, porque está disponível para qualquer outro *código no documento atual*.

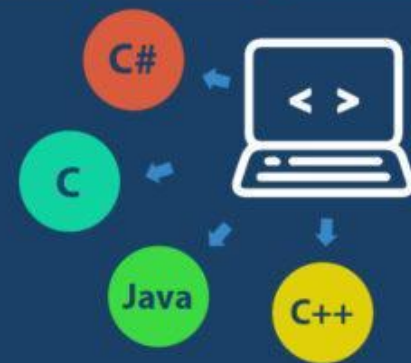
Quando você declara uma variável dentro de uma função, é chamada de variável local, pois ela está *disponível somente dentro dessa função*.

Variáveis

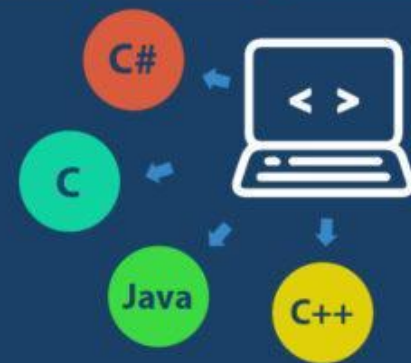
Tipos de dados

Seis tipos de dados são os chamados primitivos:

- Boolean. true e false.
- null. Uma palavra-chave que indica valor nulo. Devido JavaScript ser case-sensitive, null não é o mesmo que Null, NULL, ou ainda outra variação.
- undefined. Uma propriedade superior cujo valor é indefinido.
- Number. 42 ou 3.14159.
- String. "Howdy"
- Object (json, vetor, matriz)



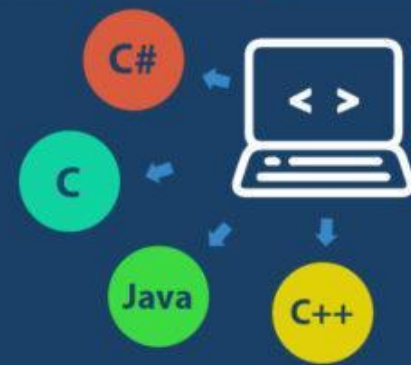
Variáveis



Conversão de tipos de dados

JavaScript é uma linguagem dinamicamente tipada. Isso significa que você não precisa especificar o tipo de dado de uma variável quando declará-la, e tipos de dados são convertidos automaticamente conforme a necessidade durante a execução do script.

Variáveis



Então, por exemplo, você pode definir uma variável da seguinte forma:

```
var answer = 42;
```

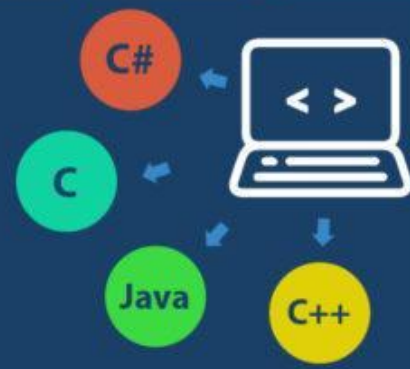
E depois, você pode atribuir uma string para a mesma variável, por exemplo:

```
answer = "Obrigado pelos peixes...";
```

Devido JavaScript ser dinamicamente tipado, essa declaração não gera uma mensagem de erro.

Variáveis

Vamos praticar.



```
var nome = "Letícia";
```

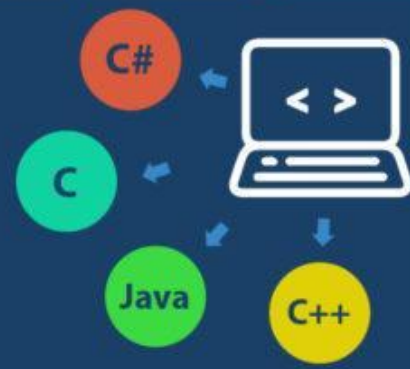
```
var numerico = 1;
```

```
var bol = true;
```

```
var num= 1.5;
```

```
console.log(nome, numerico, bol, num)
```

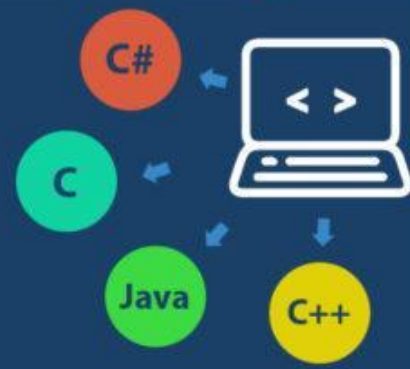

Laços e iterações



Laços oferecem um jeito fácil e rápido de executar uma ação repetidas vezes.

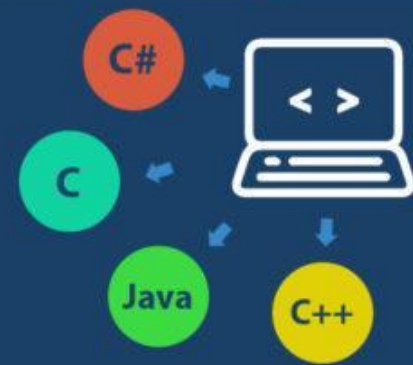
Você pode pensar em um laço de repetição como um jogo onde você manda o seu personagem andar X passos em uma direção e Y passos em outra; por exemplo, a ideia "vá 5 passos para leste"

Laços e iterações



```
var passo;  
for (passo = 0; passo < 5; passo++) {  
    // Executa 5 vezes, com os valores de passos de 0 a 4.  
    console.log("Ande um passo para o leste");  
}
```

Laços e iterações

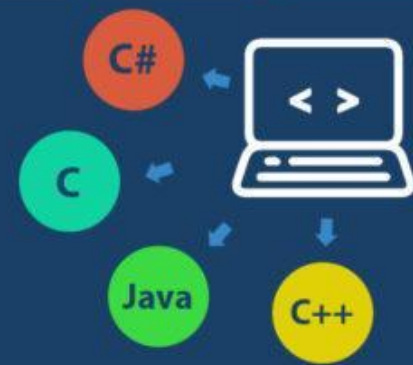


Existem várias formas diferentes de laços, mas eles essencialmente fazem a mesma coisa: repetir uma ação múltiplas vezes (inclusive você poderá repetir 0 vezes). Os vários mecanismos diferentes de laços oferecem diferentes formas de determinar quando este irá começar ou terminar.

Há várias situações em que é mais fácil resolver um problema utilizando um determinado tipo de laço do que outros.

Laços e iterações

Declaração for



Um laço for é repetido até que a condição especificada seja falsa.

```
for ([expressaoInicial]; [condicao]; [incremento]){  
    declaracao  
}
```

Laços e iterações

Declaração do...while

A instrução do...while repetirá até que a condição especificada seja falsa.

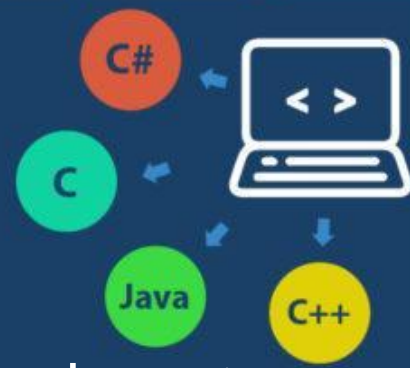
do

declaracao

while (condicao);

No exemplo a seguir, o laço é executado pelo menos uma vez e irá executar até que i seja menor que 5.

```
do {  
    i += 1;  
    console.log(i);  
} while (i < 5);
```



Laços e iterações

Declaração do...while

A instrução do...while repetirá até que a condição especificada seja falsa.

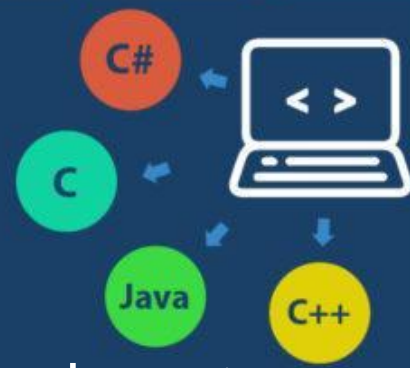
do

declaracao

while (condicao);

No exemplo a seguir, o laço é executado pelo menos uma vez e irá executar até que i seja menor que 5.

```
do {  
    i += 1;  
    console.log(i);  
} while (i < 5);
```



Laços e iterações

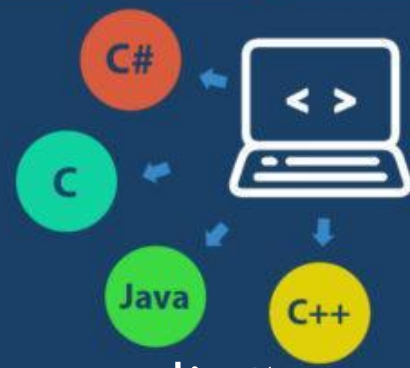
Declaração while

Uma declaração while executa suas instruções, desde que uma condição especificada seja avaliada como verdadeira. Segue uma declaração while:

```
while (condicao)
```

```
    declaracao
```

Se a condição se tornar falsa, a declaração dentro do laço para a execução e o controle é passado para a instrução após o laço.



Laços e iterações



O teste da condição ocorre antes que o laço seja executado. Desta forma se a condição for verdadeira o laço executará e testará a condição novamente. Se a condição for falsa o laço termina e passa o controle para as instruções após o laço.

O while a seguir executará enquanto n for menor que três:

```
n = 0;  
x = 0;  
while (n < 3) {  
    n++;  
    x += n;  
}
```


Funções

Declarando uma função



```
function square(numero) {  
    return numero * numero;  
}
```

A função `square` recebe um argumento chamado `numero`. A função consiste em uma instrução que indica para retornar o argumento da função: `var numero` multiplicado por si mesmo. A declaração *return* especifica o valor retornado pela função.

Funções

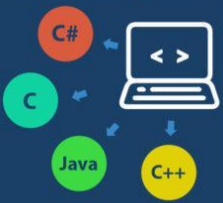


Expressão de função

Embora a declaração de função acima seja sintaticamente uma declaração, funções também podem ser criadas por uma expressão de função. Tal função pode ser anônima; ele não tem que ter um nome. Por exemplo, a função square poderia ter sido definida como:

```
var square = function (numero) {  
    return numero * numero;  
};  
  
var x = square(4); //x recebe o valor 16
```

Expressões e operadores



Operadores – O JavaScript possui os tipos de operadores a seguir

Operadores de atribuição

Operadores de comparação

Operadores aritméticos

Operadores lógicos

Operadores de string

Operador condicional (ternário)

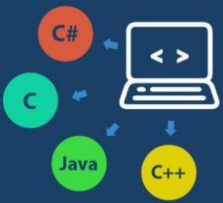
Operador vírgula

Operadores unário

Operadores relacionais

Expressões e operadores

Operadores de atribuição



Um operador de atribuição atribui um valor ao operando à sua esquerda baseado no valor do operando à direita. O operador de atribuição básico é o igual (=), que atribui o valor do operando à direita ao operando à esquerda. Isto é, $x = y$ atribui o valor de y a x .

Expressões e operadores

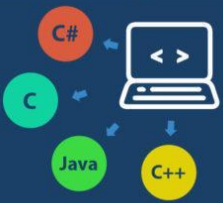


Os outros operadores de atribuição são encurtamentos de operadores padrão, como mostrado na tabela a seguir.

Nome	Operador encurtado	Significado
Atribuição	<code>x = y</code>	<code>x = y</code>
Atribuição de adição	<code>x += y</code>	<code>x = x + y</code>
Atribuição de subtração	<code>x -= y</code>	<code>x = x - y</code>
Atribuição de multiplicação	<code>x *= y</code>	<code>x = x * y</code>
Atribuição de divisão	<code>x /= y</code>	<code>x = x / y</code>
Atribuição de resto	<code>x %= y</code>	<code>x = x % y</code>
Atribuição exponencial	<code>x **= y</code>	<code>x = x ** y</code>

Expressões e operadores

Operadores de comparação



Um operador de comparação compara seus operandos e retorna um valor lógico baseado em se a comparação é verdadeira. *Os operandos podem ser numéricos, strings, lógicos ou objetos.*

Strings são comparadas com base em ordenação lexográfica utilizando valores Unicode.



Expressões e operadores

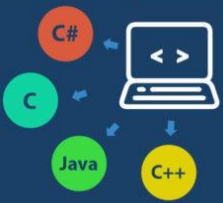
Operadores de comparação

Operador	Descrição	Exemplos que retornam verdadeiro
Igual (==)	Retorna verdadeiro caso os operandos sejam iguais.	<code>3 == var1</code> <code>"3" == var1</code> <code>3 == '3'</code>
Não igual (!=)	Retorna verdadeiro caso os operandos não sejam iguais.	<code>var1 != 4</code> <code>var2 != "3"</code>
Estritamente igual (===)	Retorna verdadeiro caso os operandos sejam iguais e do mesmo tipo. Veja também Object.is e igualdade em JS (en-US) .	<code>3 === var1</code>
Estritamente não igual (!==)	Retorna verdadeiro caso os operandos não sejam iguais e/ou não sejam do mesmo tipo.	<code>var1 !== "3"</code> <code>3 !== '3'</code>
Maior que (>)	Retorna verdadeiro caso o operando da esquerda seja maior que o da direita.	<code>var2 > var1</code> <code>"12" > 2</code>
Maior que ou igual (>=)	Retorna verdadeiro caso o operando da esquerda seja maior ou igual ao da direita.	<code>var2 >= var1</code> <code>var1 >= 3</code>
Menor que (<)	Retorna verdadeiro caso o operando da esquerda seja menor que o da direita.	<code>var1 < var2</code> <code>"12" < "2"</code>
Menor que ou igual (<=)	Retorna verdadeiro caso o operando da esquerda seja menor ou igual ao da direita.	<code>var1 <= var2</code> <code>var2 <= 5</code>

Expressões e operadores

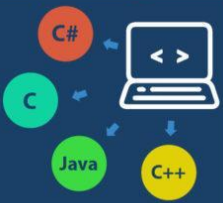
Operadores aritméticos

Operadores aritméticos tomam valores numéricos (sejam literais ou variáveis) como seus operandos e retornam um único valor numérico. Os operadores aritméticos padrão são os de soma (+), subtração (-), multiplicação (*) e divisão (/).



Expressões e operadores

Operadores aritméticos




Estes operadores trabalham da mesma forma como na maioria das linguagens de programação quando utilizados com números de ponto flutuante (em particular, repare que divisão por zero produz um NaN (en-US)). Por exemplo:

```
console.log(1 / 2); /* imprime 0.5 */
```

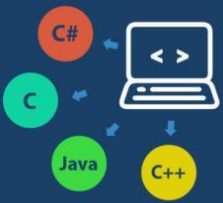
```
console.log(1 / 2 == 1.0 / 2.0); /* isto também é verdadeiro */
```

Em complemento às operações aritméticas padrões (+, -, * /), o JavaScript disponibiliza os operadores aritméticos listados na tabela a seguir.

Operador	Descrição	Exemplo
Módulo (%)	Operador binário. Retorna o inteiro restante da divisão dos dois operandos.	12 % 5 retorna 2.
Incremento (++)	Operador unário. Adiciona um ao seu operando. Se usado como operador prefixado (++x), retorna o valor de seu operando após a adição. Se usado como operador pósfixado (x++), retorna o valor de seu operando antes da adição.	Se x é 3, então ++x define x como 4 e retorna 4, enquanto x++ retorna 3 e, somente então, define x como 4.
Decremento (--)	Operador unário. Subtrai um de seu operando. O valor de retorno é análogo àquele do operador de incremento.	Se x é 3, então --x define x como 2 e retorna 2, enquanto x-- retorna 3 e, somente então, define x como 2.
Negação (-)	Operador unário. Retorna a negação de seu operando.	Se x é 3, então -x retorna -3.
Adição (+)	Operador unário. Tenta converter o operando em um número, sempre que possível.	+ "3" retorna 3. +true retorna 1.
Operador de exponenciação (**) 	Calcula a base elevada á potência do expoente, que é, base <code>expoente</code>	2 ** 3 retorna 8. 10 ** -1 retorna 0.1

Expressões e operadores

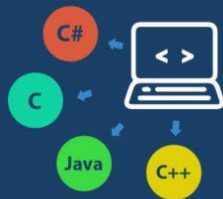
Operadores lógicos



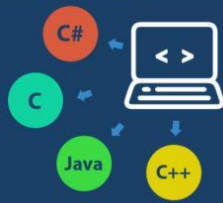
Operadores lógicos são utilizados tipicamente com valores booleanos (lógicos); neste caso, retornam um valor booleano.

Expressões e operadores

Operadores lógicos



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



Expressões e operadores

Operadores lógicos

O código a seguir mostra exemplos do operador && (E lógico).

```
var a1 = true && true; // retorna true
```

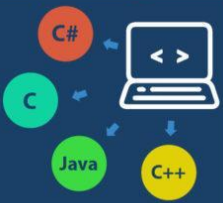
```
var a2 = true && false; // f retorna false
```

```
var a3 = false && true; // retorna false
```

```
var a4 = false && 3 == 4; // retorna false
```


Expressões e operadores

Operadores lógicos

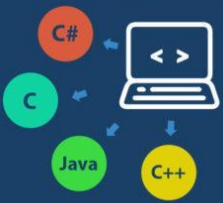


O código a seguir mostra exemplos do operador || (OU lógico).

```
var o1 = true || true; // retorna true  
var o2 = false || true; // retorna true  
var o3 = true || false; // retorna true  
var o4 = false || 3 == 4; // retorna false
```

Expressões e operadores

Operadores lógicos



O código a seguir mostra exemplos do operador ! (negação lógica).

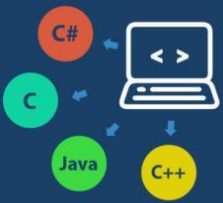
```
var n1 = !true; // retorna false
```

```
var n2 = !false; // retorna true
```

```
var n3 = !"Gato"; // retorna false
```

Expressões e operadores

Operadores de string



Além dos operadores de comparação, que podem ser utilizados em valores string, o operador de concatenação (+) concatena dois valores string, retornando outra string que é a união dos dois operandos.

Por exemplo,

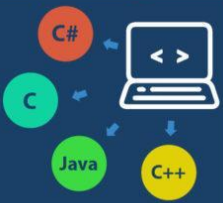
```
console.log("minha " + "string"); // exibe a string "minha string".
```


Expressões e operadores

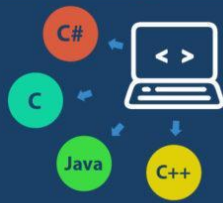
Operador condicional (ternário)

O operador condicional é o único operador JavaScript que utiliza três operandos. O operador pode ter um de dois valores baseados em uma condição. A sintaxe é:

`condicao ? valor1 : valor2`



Expressões e operadores



Se condicao for verdadeira, o operador terá o valor de valor1. Caso contrário, terá o valor de valor2. Você pode utilizar o operador condicional em qualquer lugar onde utilizaria um operador padrão.

```
var status = idade >= 18 ? "adulto" : "menor de idade";
```

Esta declaração atribui o valor "adulto" à variável status caso idade seja dezoito ou mais. Caso contrário, atribui o valor "menor de idade".

PRATICANDO



Vamos começar com o famoso hello word;

interação com o usuário:

- Uma forma de mostrar dados para o usuário é com:

```
console.log("hello word")
```

```
ou alert("hello word");
```