Fundamentos de JavaScript

Olá, mundo!

Para escrever o primeiro programa em JS, usaremos um arquivo HTML com um arquivo JavaScript. Aqui também usaremos CSS para personalizar a página, além do Google Chrome para usar o console para debugs e mostrar novas aplicações usando console.log ().

Exemplo:

```
console.log("Olá, mundo!");
```

O que é JavaScript?

É uma linguagem de programação orientada a objetos de alto nível, multiparadigma. Também é possível dizer que ela é uma linguagem de programação Client-side, que roda no cliente, ou seja, roda na máquina da pessoa que está acessando o site, e não em um servidor remoto. Essa linguagem é interpretada, podendo ser utilizada junto com HTML e não compilada, uma vez que o navegador irá interpretar os códigos à medida que as ações do site forem acontecendo.

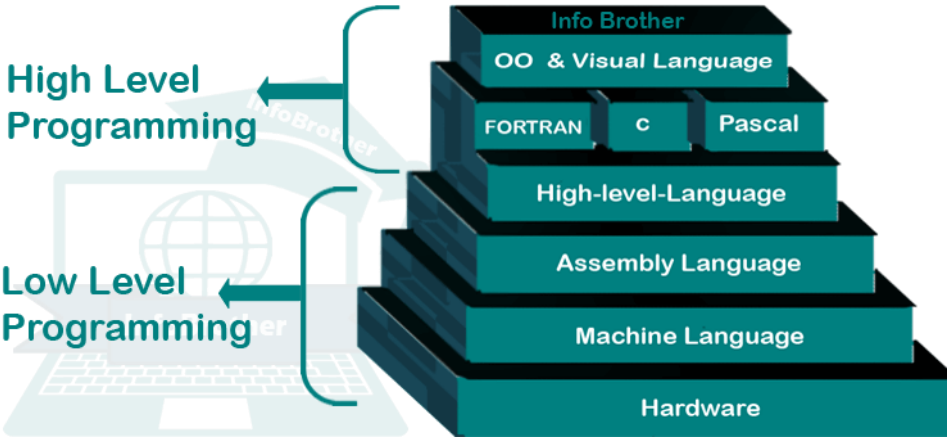
- Orientada a objetos – usa objetos para melhor compreensão do código.
- Alto nível - fácil de ser compreendida e é semelhante a língua humana.
- Multiparadigma – JavaScript pode usar diferentes tipos de paradigmas de programação para desenvolver um código como Imperativo e Declarativo. A ideia é fornecer um framework no qual o programador possa trabalhar com vários estilos, misturando livremente diferentes paradigmas. De modo geral, o objetivo em se projetar linguagens deste tipo, é reservar aos programadores a melhor ferramenta para determinado trabalho, admitindo que nenhum paradigma resolve todos os problemas da maneira mais elegante, ou mesmo eficiente.

As 5 gerações da linguagem de programação

Desde que a programação iniciou em 1945, há desde então 5 gerações de linguagem de programação, sendo elas:

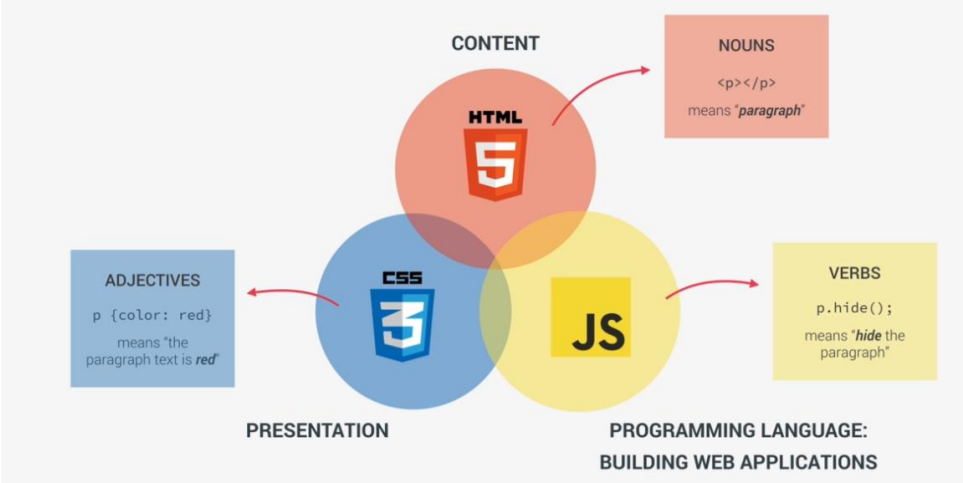
1. Machine Language - também conhecida como a representação de dados 1's e 0's: a única língua que o computador consegue realmente entender.
2. Assembly Language – linguagem de programação de baixo nível. Isso quer dizer que Assembly está próximo de Machine Language, porém mais fácil de ser entendida e desenvolvida.
3. High Level Languages - são linguagens de programação que são semelhantes a linguagem humana. São facilmente compreensíveis, lidas e escritas. É o contrário de Assembly Language.
4. Problem Oriented Languages – linguagens que foram criadas para resolver determinado problema do mundo real.
5. Natural Languages – principalmente usado por inteligência artificial.

Exemplo de linguagem de alto nível:

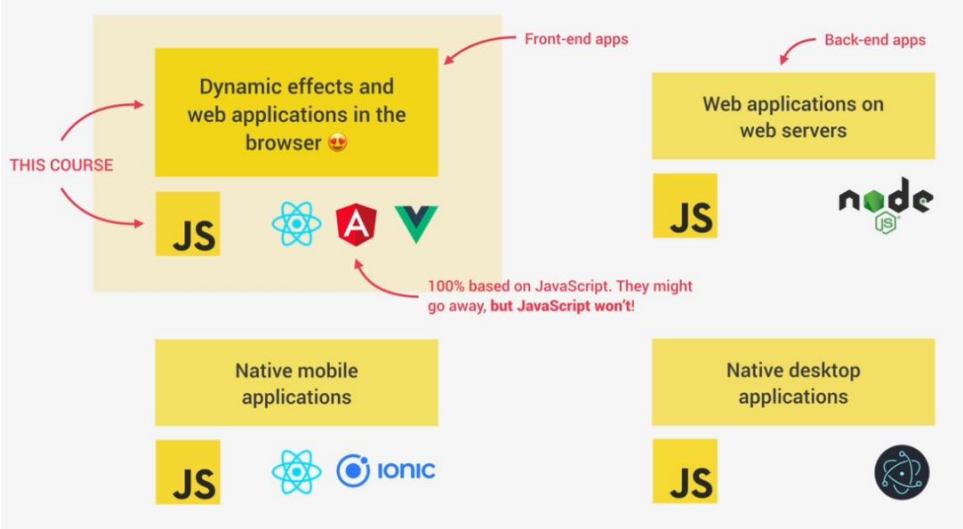


O espaço do JavaScript no Desenvolvimento Web

O JavaScript permite inserir vários efeitos, fazendo com que o site fique mais dinâmico, principalmente porque permite ao programador Web melhorar a página do site.

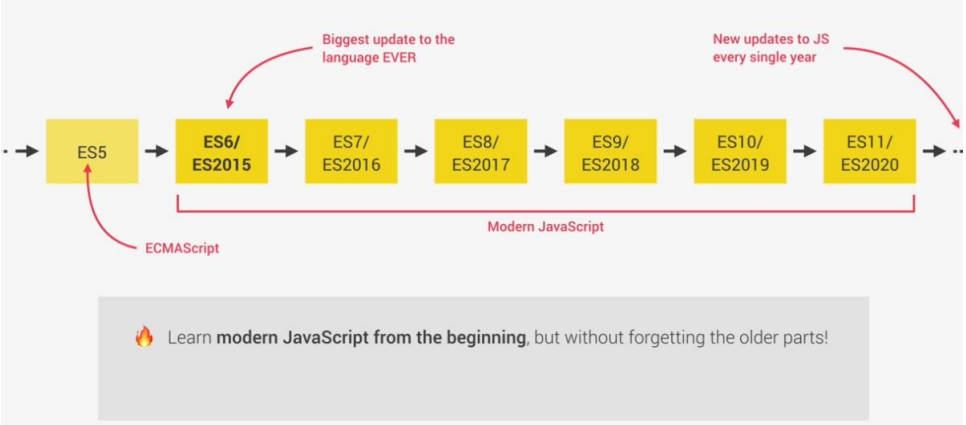


O que JavaScript pode fazer?



ES significa ECMASript.

Versões do JavaScript



Usando JavaScript com HTML

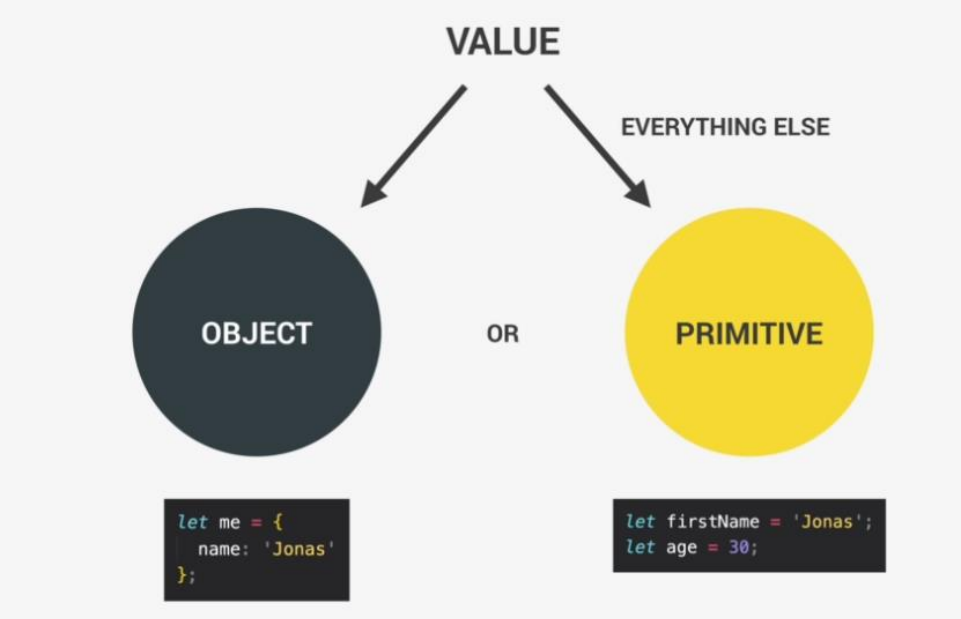
Para usar JavaScript com HTML, linkamos o arquivo.js com a tag <script> antes da tag de fechamento </body>. Linkar um arquivo.js na tag <head> pode ocasionar problemas.

Variáveis e Valores

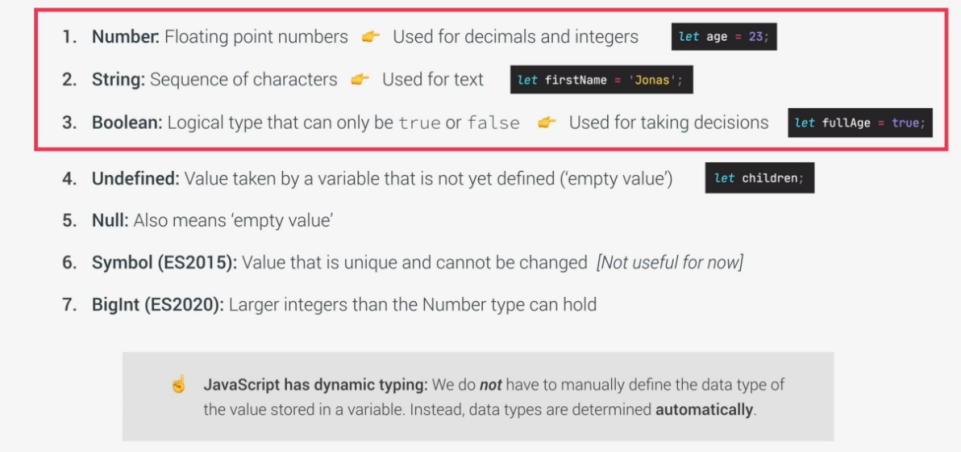
Variáveis em JavaScript são uma forma para se salvar dados que passam ser acessados e/ou manipulados posteriormente, independentemente do tipo desses dados – **strings, números, booleanos, arrays, objetos ou funções**. Sua utilização proporciona um código mais limpo e fácil de ser mantido. Vale ressaltar também que JavaScript é **Case Sensitive**, ou seja, ela faz distinção entre letras maiúsculas e minúsculas. Outra coisa, variáveis não podem começar com número.

Tipos de Dados

Em JavaScript, o valor pode ser um objeto ou um primitivo. São chamados de primitivo porque seus valores podem conter apenas uma coisa (uma string ou um número ou qualquer coisa). Diferente de objeto que podem conter uma coleção de valores e entidades mais complexas.



Tipos Primitivos



Resumidamente teremos:

- **Strings** – uma string nada mais é que um texto puro;
- **Numbers** - são os números, seja eles integer, float, double, etc.
- **Booleans** - são operadores booleanos (true ou false);
- **Arrays** - É uma estrutura de dado para armazenar uma coleção de valores, sendo eles de qualquer tipo.
- **Objects** – Conjunto de atributos aninhados a uma variável denomina-se um objeto.
- **Functions** – Em JavaScript é possível declarar uma variável como uma função, podendo fazer operações e retornando o valor para a variável de declaração.

Observe: Java Script tem **Tipagem Dinâmica (Dynamic Typing)**, ou seja, o própria JS consegue saber que tipo é o valor atribuído à variável e fazer o casting automático. Por exemplo, em C# precisamos declarar o tipo de variável (por exemplo, string) para depois dar um nome à ela e finalmente um valor. Em JavaScript, basta que o temos um nome e um valor.

Let, Const e Var

Além da possibilidade de declarar uma variável com **Let**, temos **Const** e **Va**r. Em JavaScript, podemos reatribuir um valor de uma variável, se quisermos, depois de já estabelecidas ou deixadas sem.

No entanto, **reatribuir um tipo de valor para uma variável do tipo const não irá funcionar**, pois trata-se de uma variável que não pode mudar o seu valor em via de regra, por ser constante. Assim também não podemos declarar variáveis do tipo **const sem atribuir a ela um valor**, igual como aconteceu com let ou null. Para desenvolver um código limpo e funcional, usamos **const** em vez de **let** para minimizar problemas e erros futuros com códigos, já que **const** não pode ser alterado ou ser reatribuído ou ser usado fora do escopo em que foi criado.

Var é a última opção e deve ser evitada. Não é mais utilizada hoje em dia, pois ela não possui Block Scop. Basicamente podemos acessar **var** depois de {}. Ou seja, qualquer linha de código depois de uma função fechada, podemos acessa-las sem problema nenhum. Sempre temos que declarar uma variável.

Escopo

O escopo em JavaScript define a limitação e visibilidade de um bloco de código. Eles são:

- **Escopo Global** – quando a variável é declarada fora qualquer bloco, sua visibilidade fica disponível em todo o código;
- **Escopo Local** – quando a variável é declarada dentro de um bloco, sua visibilidade pode ficar disponível ou não.

Operações básicas

Operações matemáticas:

- + = Adições;
- - = Subtrações;
- / = Divisão;
- * = Multiplicação;
- ** = Potenciação;

Assignment Operators:

- += : reutiliza o mesmo valor;
- *= : reutiliza o mesmo valor, mas com multiplicação;
- ++ : adiciona mais 1 valor a um valor anterior;
- -- : subtrai 1 valor de um valor anterior.

Operadores Relacionais

São tipos de operadores que consultam a relação entre valores:

- **>** maior que;
- **<** menor que;
- **==** igual a **-** para valores não relacionados (‘13’ == 13);
- **===** igual a - para valores relacionados (13 === 13);
- **>=** maior ou igual que;
- **<=** menor ou igual que;
- **!=** diferente que - abstração;
- **!==** diferente que – literal.

Operadores Lógicos

Utilizamos os operadores lógicos quando precisamos realizar operações sobre um ou dois valores booleanos:

- **&& - Operador AND –** precisa que ambos os valores booleanos sejam verdadeiros para que o resultado seja validado como verdadeiro. Caso um valor booleano seja falso, então o resultado será falso.
- **|| - Operador OR –** precisa que apenas um valor seja verdadeiro para que o resultado seja verdadeiro. Não importa quantos variáveis forem falsas, se existir um valor booleano verdadeiro, o resultado será validado como verdadeiro.
- **! - Operador NOT –** diferente de todos acima, NOT inverte um valor booleano. Por exemplo: A = True; se usarmos NOT, o valor de A será falso.

Precedência de Operadores

A procedência de operadores determina a ordem em que os operadores são processados.

Um exemplo simples:

```
3 + 4 * 5 // returns 23
```

O operador de multiplicação ("`*`") tem maior precedência que o operador de adição ("`+`") e por isso será executado primeiro.

Podemos usar **PEMDAS** (Please Excuse My Dear Aunt Sally):

> With PEMDAS, you first work with what is inside the parenthesis, then exponents, next is multiplication, then division, after its addition, and lastly, it is subtraction.

Strings Templates Literals

O JavaScript tem uma grande característica chamada interpolação de string que lhe permite injetar uma variável, uma chamada de função, e uma expressão aritmética diretamente numa string.

O ES6 introduziu o conhecido Template String ou Template Literals, onde conseguimos criar cadeias de caracteres, utilizando outra forma de interpolação que não fosse o sinal de +.

```
const jonas = "I'm " + firstName + ', a ' + (year -
birthYear) + ' years old ' + job + '!';
console.log(jonas);
```

Para utilizar um Template Literal, é necessário utilizarmos backtick (acento grave da língua portuguesa), para realizar a interpolação, utilizando notação de ${}

```
const jonasNew = `I'm ${firstName}, a ${year -
birthYear} year old ${job}!`;
console.log(jonasNew);
```

Tomando Decisões e IF-ELSE

- **IF** – Usamos IF para testar se uma condição booleana. Se as condições forem aceitas, ela executará um método, por exemplo.
- **ELSE IF** – Usamos para confirmar outra condição booleana. Basicamente uma repetição do IF. Se a condição de IF não for aceita, então um segundo IF irá aparecer.
- **ELSE** – Usamos ELSE para notificar que uma condição booleana de IF não foi aceita, então ELSE irá aparecer para dizer isso.

```
const idade = 17;

if (idade >= 18){
  console.log("Você é maior de idade.");
} else {
  console.lo("Você não tem idade suficiente.");
}
```

Tipos de Conversão e Coerção.

Tipo de conversão (ou typecasting) significa transferir dado para outra tipo de dado. Ou seja, fazer a conversão de um dado(string) para (number), por exemplo. Qualquer tipo de dado, seja ele primitivo ou objeto, é suscetível a coerção.

Podemos fazer isso automaticamente (coerção do JavaScript) ou manualmente. Por exemplo, usando os métodos Number ou String.

Valores Verdadeiros ou Falsos

Em JavaScript, temos 5 valores que irão representar a condição falsa:

- **0, ‘ ’(vazio), undefined, null e NaN(Not a Number)**.

Agora, todo valor que não for 0 ou vazio será verdadeiro.

Operadores de Igualdade == e ===

Usamos operadores de igualdade para comparar se um valor A é igual ao valor B. No entanto, existem diferenças ao redor dessas operações.

- = quando temos um símbolo de igualdade, na verdade estamos atribuindo valor a algo. Como exemplo, atribuindo um dado a uma variável;
- == quando usamos dois símbolos de igualdade, na verdade estamos usando um operador de igualdade, no entanto, para dizer que um valor é diferente de outro valor. Isso quer dizer que comparar uma string com um number, nesse caso, usaremos o (==) também conhecido como loose. Aqui a variável é convertida para a mesma que está sofrendo a comparação. No final, estamos fazendo uma comparação abstrata.
- === quando usamos três símbolos de igualdade, na verdade estamos comparando se um valor de A é igualmente(number === number e não string == number) com o valor de B. No final, estamos fazendo uma comparação literal.

Prompt

Usamos prompt para receber dados dos usuários. Podemos armazenar essas informações em uma variável. **Vale lembrar que o dado recebido aqui será uma String**, ou seja, precisamos usar Number para converter nesse processo.

Lógica Booleana

- AND – Usamos && para comparar se duas condições booleanas são verdadeiras. Aqui, é necessário que as duas operações sejam verdadeiras para que o resultado seja validado como verdadeiro. Se uma operação for falso, então o resultado será falso, já que é preciso de duas condições para isso.

- OR – Usamos || para comparar uma condição booleana for verdadeira ou falsa. Aqui precisamos que apenas uma operação seja verdadeira para que ela seja verdadeira. Não importa quantas variáveis negativas forem, um resultado verdadeiro é suficiente.
- NOT – Usamos ! para reverter um valor booleano.

Switch

Usamos a Switch quando temos em nosso código diversas condições IF-ELSE. A função do Switch é a mesma de IF-ELSE, só que mais organizado e possui mais performance.

```
const day = 'monday';

switch (day) {
  case 'monday': // day === 'monday'
    console.log('Plan course structure');
    console.log('Go to coding meetup');
    break;
  case 'tuesday':
    console.log('Prepare theory videos');
    break;
  case 'wednesday':
  case 'thursay':
    console.log('Write code examples');
    break;
  case 'friday':
    console.log('Record videos');
    break;
  case 'saturday':
  case 'sunday':
    console.log('Enjoy the weekend :D');
    break;
  default:
    console.log('Not a valid day!');
}
```

Statements and Expressions

- **Expressions** - é um pedaço de código que produz um valor, por exemplo: 3 + 4 vai produzir um valor. Strings e Numbers são expressions também, além de condições booleanas.
- **Statements** - é um pedaço de código que não produz um valor. Ele apenas dita o que deve ser feito, mas não gerará um valor. Tudo que termina com “;” é um statement. Aliás, um statement pode conter uma expression.

Vale ressaltar que o contrário disso não pode acontecer. Uma expression não pode conter um statement.

Conditional Ternary Operator

O Operador Condicional Ternário é o único operador JavaScript que possui três operandos. Este operador é frequentemente usado como um atalho para a instrução IF-ELSE.

In general, the syntax of the ternary operator is as follows:

```
condition ? expression_1 : expression_2;
```

The JavaScript ternary operator is the only operator that takes three operands.