



Introdução desenvolvimento web javaScript

É necessário saber tudo?



Stacks – Front-End



Stacks – Back-End



Respectivos
Frameworks, se necessário



Stack – Banco de Dados

Na maioria dos casos é necessário **só uma STACK** para cada papel

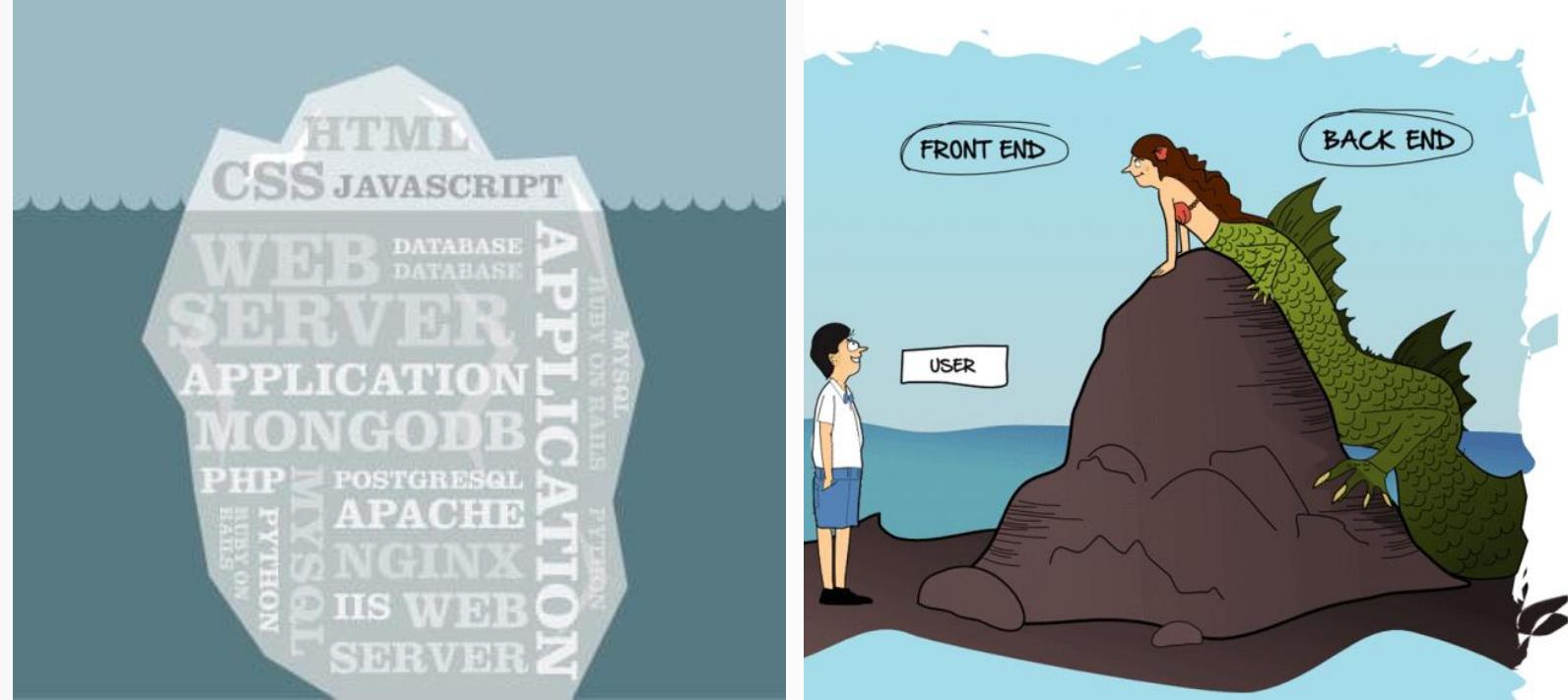
Front-End x Back-End



BACK-END



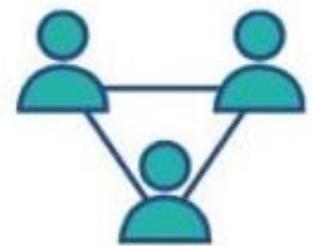
FRONT-END



Front-End



Site



Cliente

Back-End



Script Back-End



Banco de dados



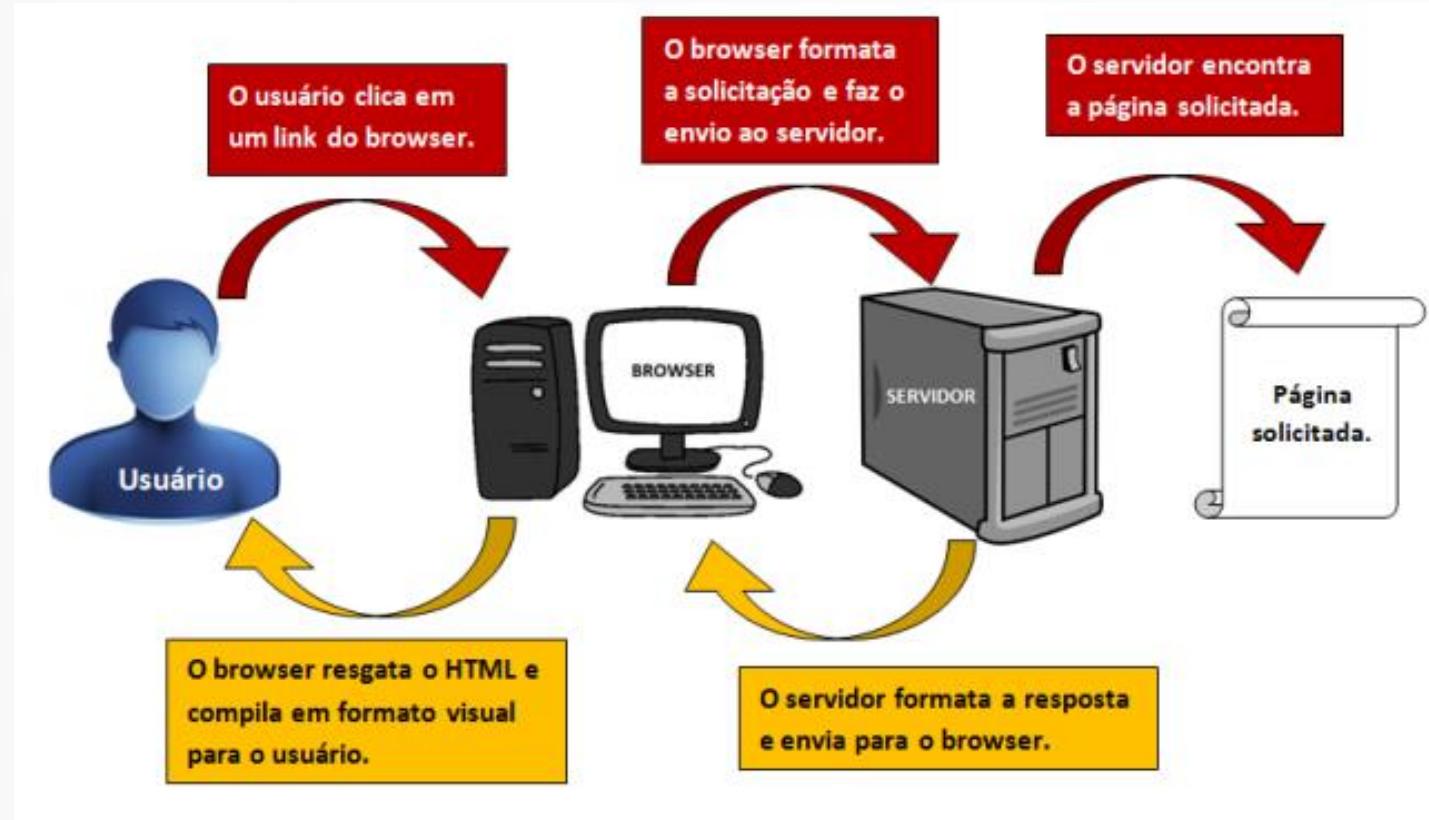
Internet



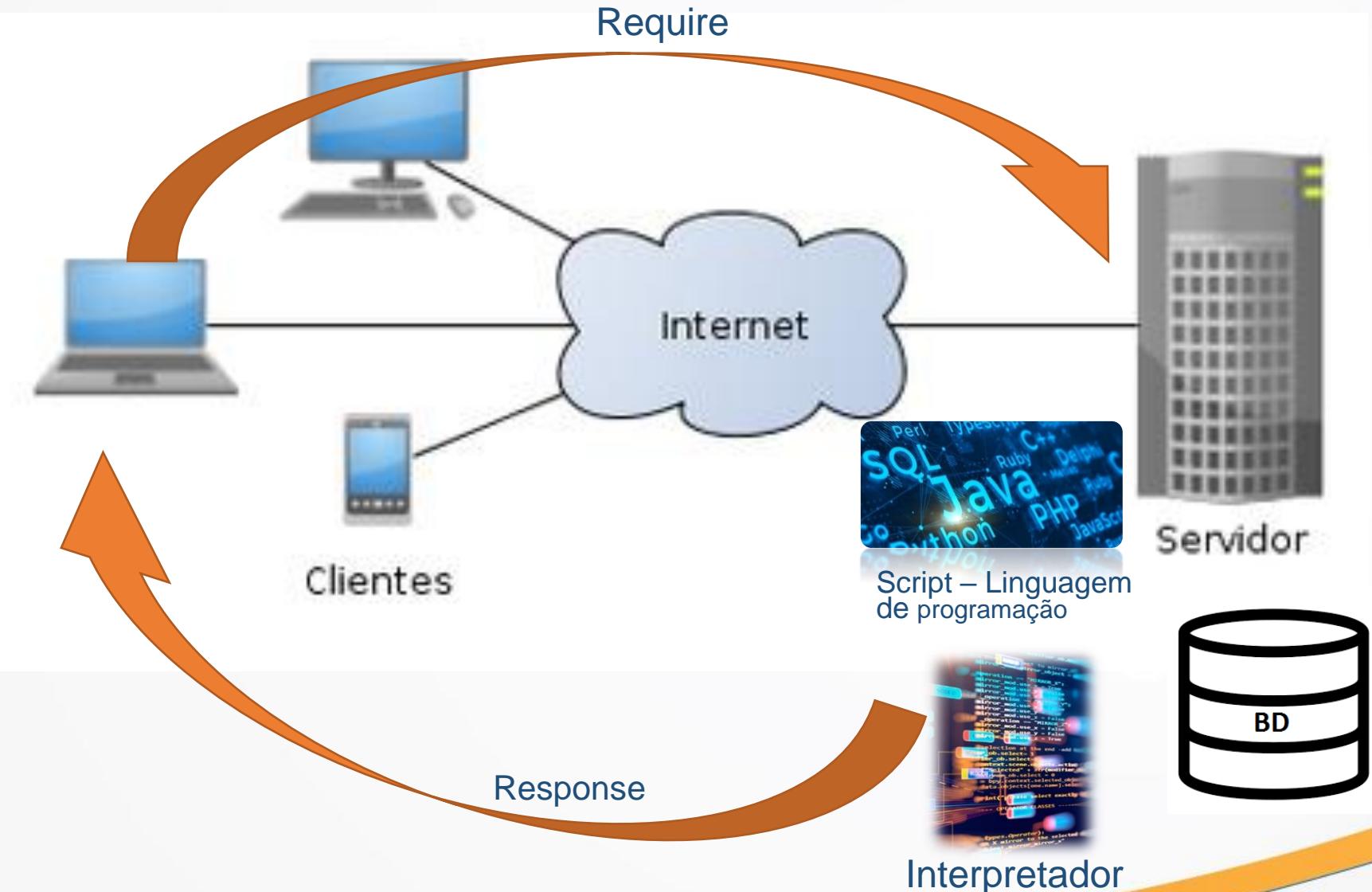
Servidor



Cliente/Servidor

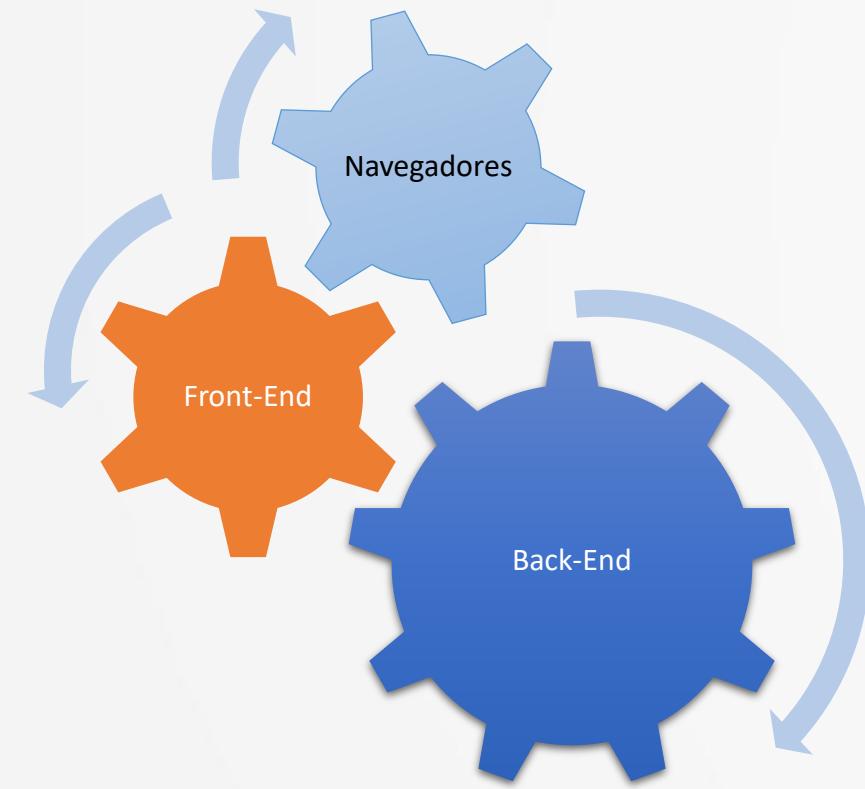


Cliente/Servidor

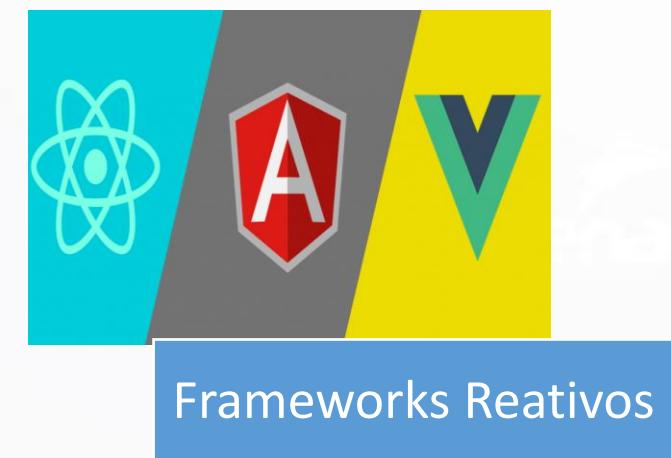
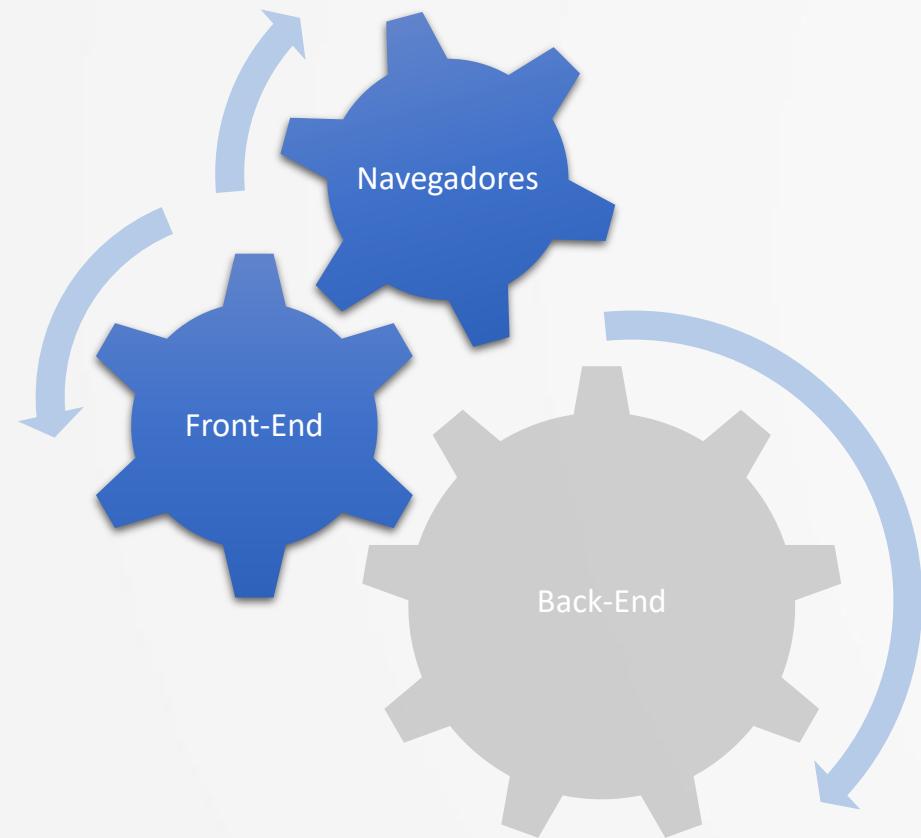


Web Stacks

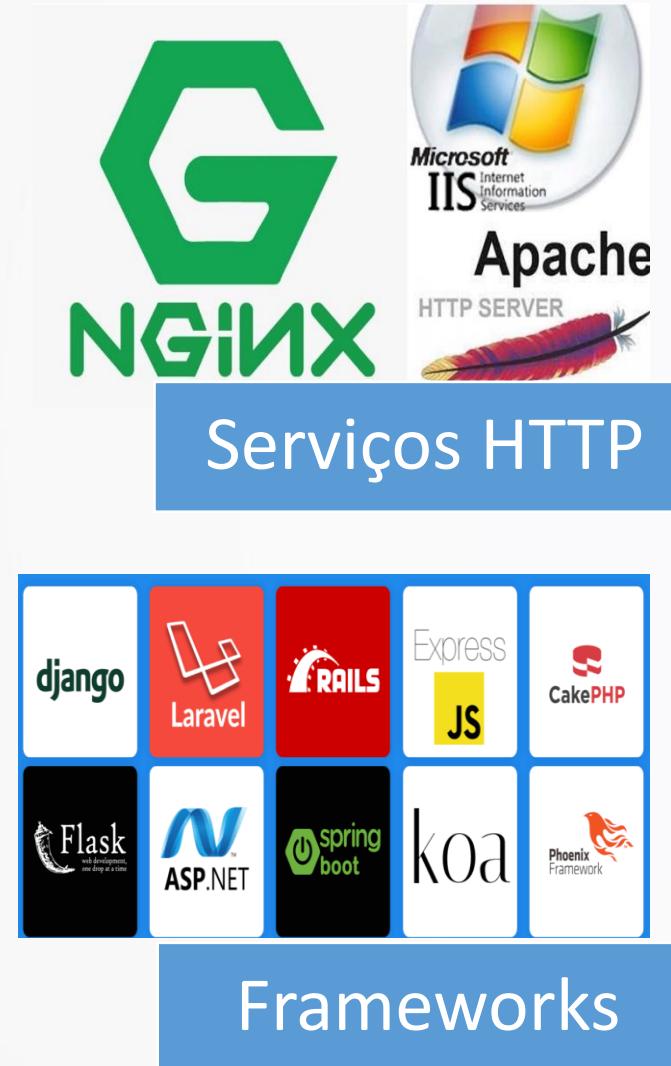
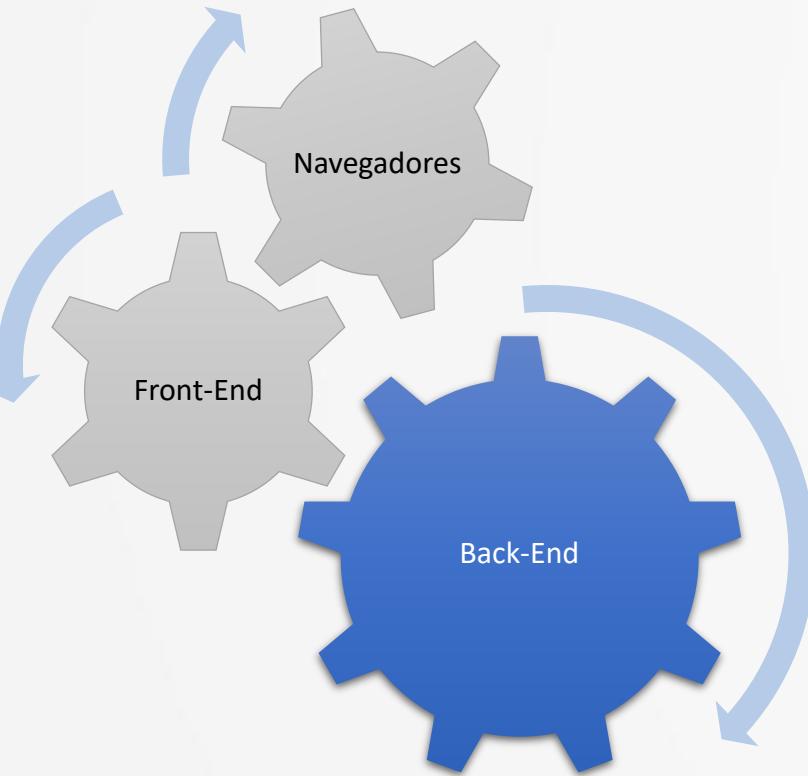
Stack é o nome que damos ao **conjunto de tecnologias** que podem ser usadas para o **desenvolvimento** de aplicações web.



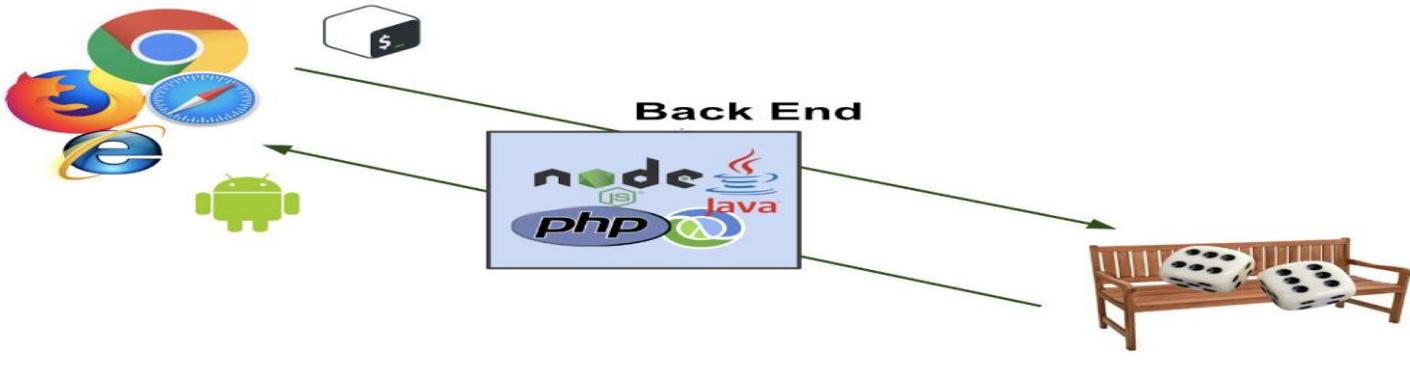
Stacks Front-End



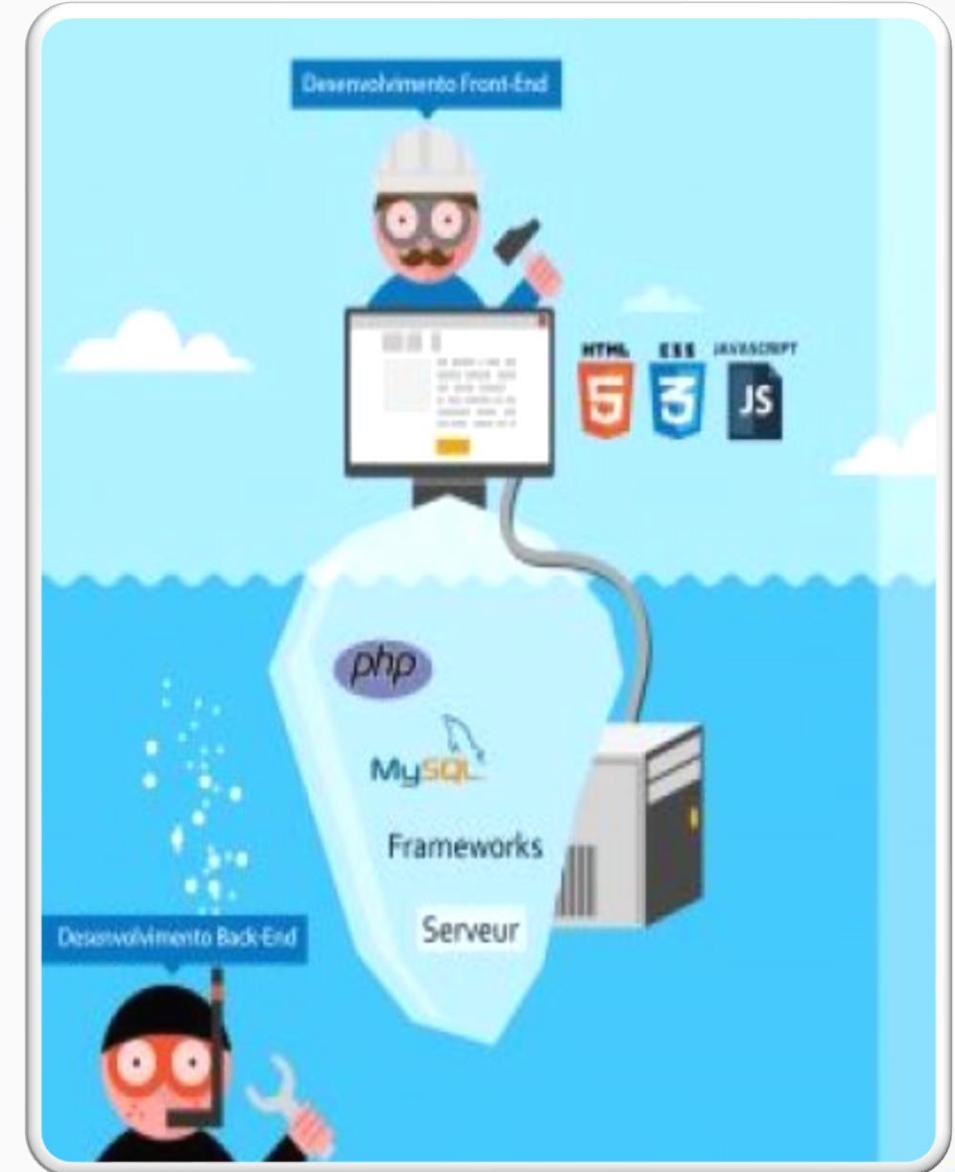
Stacks Back-End

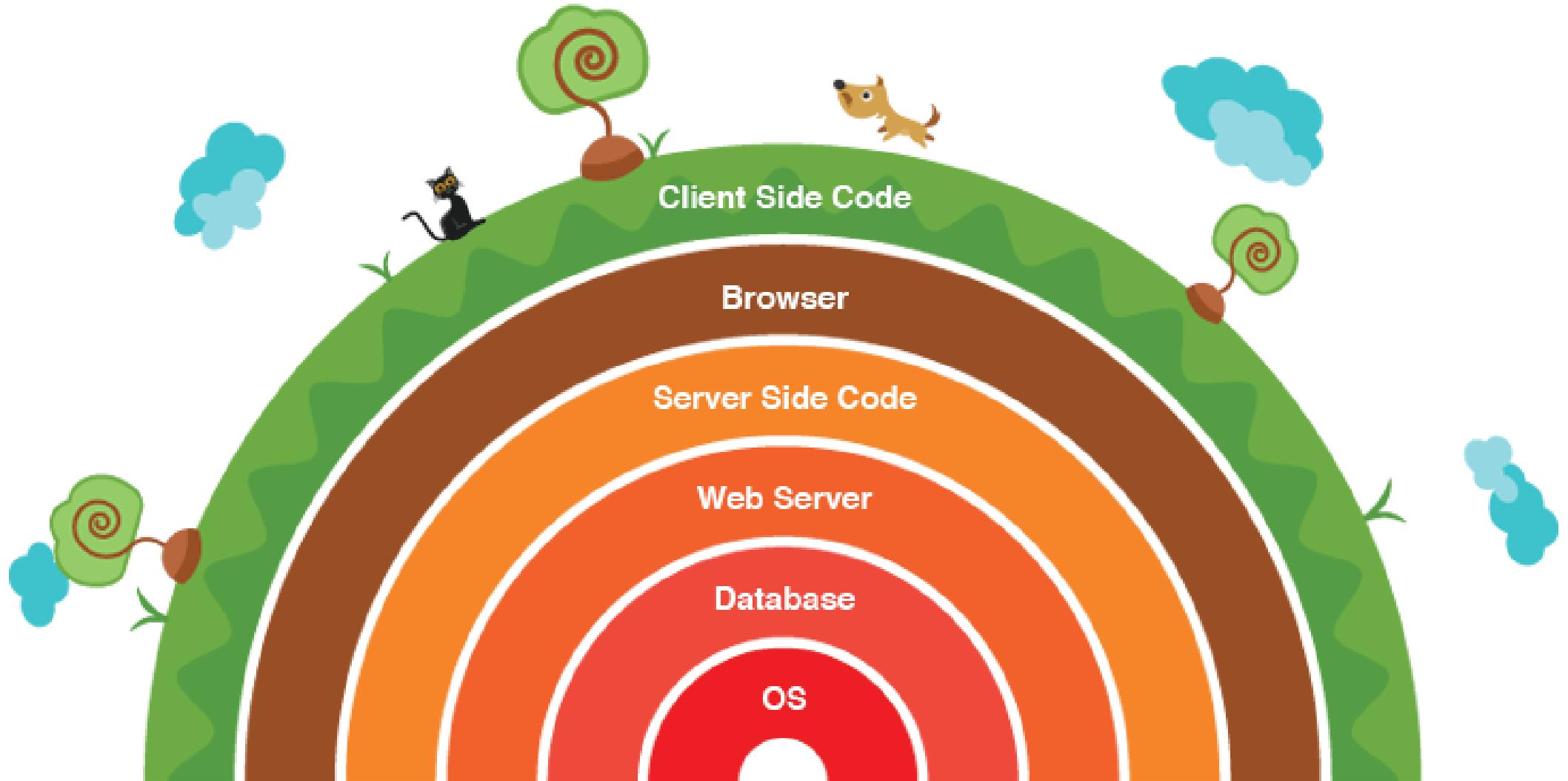


Front End: Podemos classificar como a parte visual de um site, aquilo que conseguimos interagir. Quem trabalha com Front End é responsável por desenvolver por meio de código uma interface gráfica



Back End: O próprio nome sugere, vem da ideia do que tem por trás de uma aplicação. Envolve um banco de dados e linguagens de programação que cuidam das regras de inserção, atualização e exibição das informações registradas







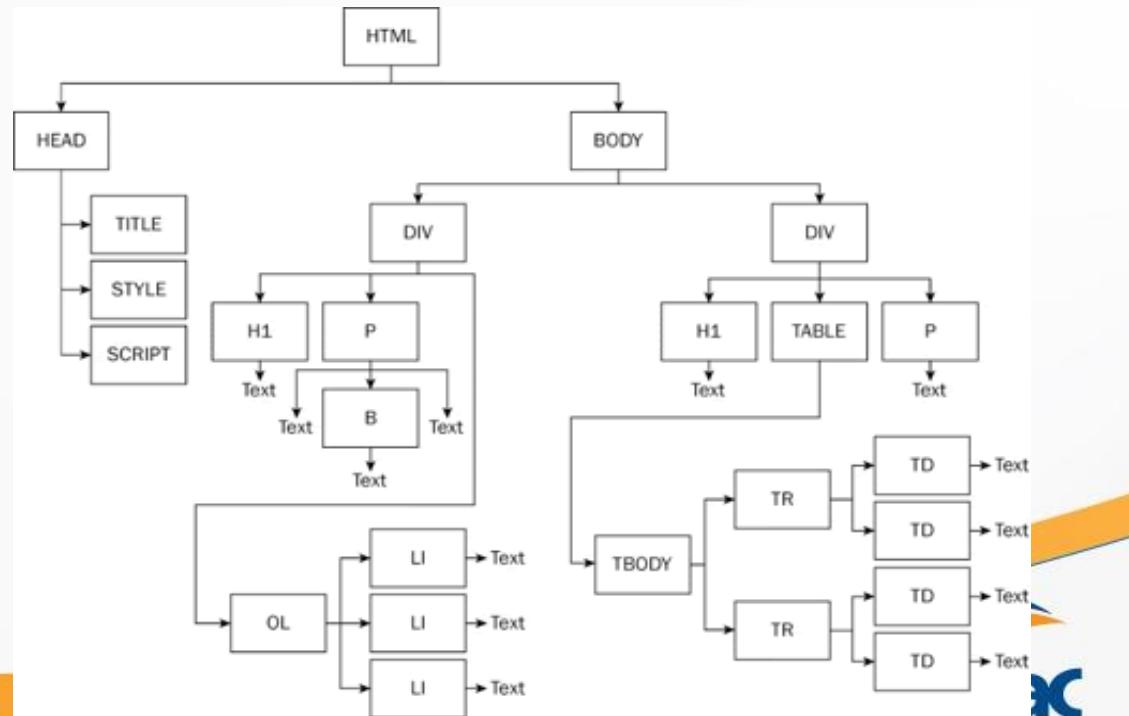
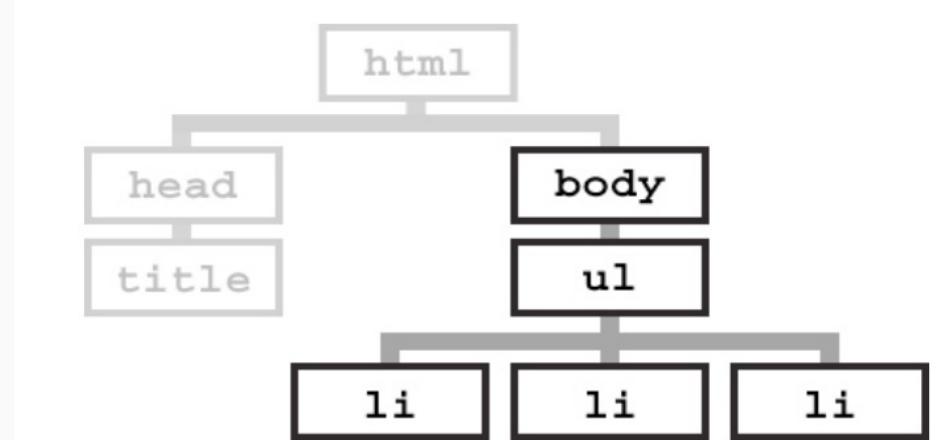
Java Script

DOM – Document Object Model

DOM – Árvore de elementos HTML

Dignifica “**Modelo de Objeto de Documento**” (**DOM**) é um modelo hierárquico de elementos de linguagens de marcação (**HTML**, **XML** e **SVG**) que **estabelece uma interface com linguagens de programação de scripts**.

O DOM define métodos que permitem acesso à árvore, para que eles possam alterar a estrutura e seus elementos.



Manipulando o DOM com Java Script

Principais seletores da API do DOM javascript:

Seletores para recuperar os valores:

```
document.getElementById()  
document.getElentsByTagName()  
document.getElementsByClassName()  
document.getElentsByName()  
document.querySelector() *  
document.querySelectorAll()
```

Todos os elementos do DOM possuem atributos que podem ser acessados ou manipulados pelo DOM.

Exemplo:

```
type="text"  
placeholder="digite um texto"  
class="campo"  
name = "txtBox"
```

Exemplo:

```
document.getElementById('id').atributo  
document.getElementById('id').getAttribute('Atributo')
```

*Para utilizar o **querySelector** é necessário utilizar um símbolo para o tipo de seletor, exemplo:

#Id | .Class | [propriedade]

Eventos - Java Script

Captura ações que acontecem no browser.

Podem ou não ser estimulados pelo usuários;

Podemos citar 4 tipos principais:

Mouse: onclick → ondblclick → onmouseup → onmouseover → onmouseout

Teclado: onkeydown → onkeypress → onkeyup

Janela: onresize → onscroll

Formulário: onfocus → onblur → onchange

Exemplo de utilização de seletores

```
9 <body>
10   <div id="box">
11     <ul id="lista">
12       <li class="item">Item1</li>
13       <li class="item">Item2</li>
14       <li class="item">Item3</li>
15       <li class="item">Item4</li>
16     </ul>
17   </div>
18 </script>
19 //Seleção de parentes:
20 var lista = document.querySelector("#lista") // Seleção da lista
21 var todosFilhos = lista.children; // Seleciona todos os elementos filhos (HTML Collection)
22 // todosFilhos = document.querySelectorAll('div ul li') - outra forma de selecionar todos os itens
23 var pai = lista.parentNode; //Seleciona o elemento PAI anterior
24 var primeiro = lista.firstElementChild; //Seleciona o primeiro elemento filho
25 var segundo = primeiro.nextElementSibling; //A partir de um elemento, seleciona o proximo
26 var ultimo = lista.lastElementChild; // Seleciona o ultimo elemento filho
27 var penultimo = ultimo.previousElementSibling; // A partir de um elemento, seleciona o anterior
28 //Exibição dos elementos selecionados
29   console.log(lista)
30   console.log(todosFilhos)
31   console.log(pai)
32   console.log(primeiro)
33   console.log(segundo)
34   console.log(ultimo)
35   console.log(penultimo)
36 </script>
37 </body>
```

The screenshot shows the browser's developer tools open to the 'Console' tab. The output of the script execution is displayed, showing the selection of elements and their properties.

```
• Item1
• Item2
• Item3
• Item4

<ul id="lista">
  <li class="item">...</li>
  <li class="item">...</li>
  <li class="item">...</li>
  <li class="item">...</li>
</ul>

HTMLCollection(4) [li.item, li.item, li.item, li.item]
  ▷ 0: li.item
  ▷ 1: li.item
  ▷ 2: li.item
  ▷ 3: li.item
  length: 4
  [[Prototype]]: HTMLCollection

<div id="box">
  <ul id="lista">...</ul>
</div>

<li class="item">
  ::marker
  "Item1"
</li>
<li class="item">
  ::marker
  "Item2"
</li>
<li class="item">
  ::marker
  "Item4"
</li>
<li class="item">
  ::marker
  "Item3"
</li>
```

Manipulando estilos e classes do DOM

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <style>
    .padrao {
      width: 100px;
      height: 100px;
      margin: auto;
      border: solid 2px black;
    }
    .grande {
      width: 200px;
      height: 200px;
      margin: auto;
      border: solid 2px black;
    }
    button {
      margin: 5px 2px 5px 2px;
    }
  </style>
```

Código cores hexadecimais:

https://www.w3schools.com/colors/colors_picker.asp

```
<body>
  <button onclick="mudarCor('#ffffff')>Branco</button>
  <button onclick="mudarCor('#000000')>Preto</button>
  <button onclick="mudarCor('#ff0000')>Vermelho</button>
  <button onclick="mudarCor('#00ff00')>Verde</button>
  <button onclick="mudarCor('#0000ff')>Azul</button><br>
  <button onclick="mudarTamanho('padrao')>Pequeno</button>
  <button onclick="mudarTamanho('grande')>Grande</button><hr>
  <div id="frame" class="padrao"></div>
  <script>
    function mudarCor(cor){
      document.getElementById("frame").style.backgroundColor = cor;
    }
    function mudarTamanho (tamanho){
      let div = document.getElementById("frame");
      div.classList.remove('padrao', 'grande');
      div.classList.add(tamanho);
      /* div.classList.toggle('padrao');
       | div.classList.toggle('grande');
       Se existe remove, se não adiciona;
       Apenas um botão e sem parâmetro
      */
    }
  </script>
</body>
```

Manipulando o DOM com Java Script

Manipulando estilo dos elementos:

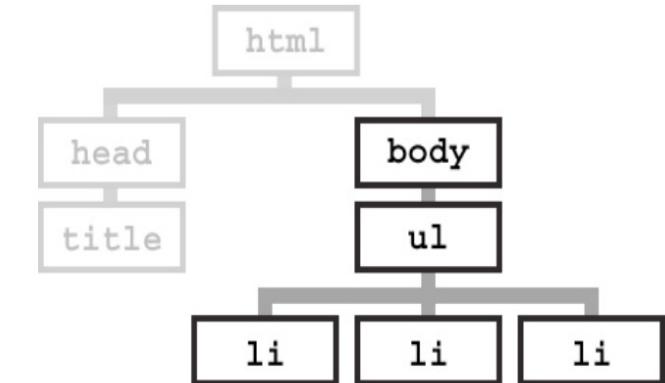
Alteração de propriedades CSS:

```
getElementById('idElemento').style.propriedade = "valor"
```

Aplicar e remover classes:

```
getElementById('idElemento').className = "classe"
```

```
getElementById('idElemento').classList.remove("classe");
```



Modificar o HTML – Javascript

```
//Escrever HTML no elemento de forma programática
var div = document.getElementById("frame"); // Seleciona o elemento
//Formas de "escrever" HTML em um elemento
div.innerHTML = `<h3>Olá mundo!!!</h3>`;
div.insertAdjacentHTML('beforebegin','<p>Antes do início</p>');
div.insertAdjacentHTML('afterbegin','<p>Depois do início</p>');
div.insertAdjacentHTML('beforeend','<p>Antes do fim</p>');
div.insertAdjacentHTML('afterend','<p>Depois do fim</p>');

//Retorna o texto de um elemento
var texto = div.innerText;
console.log(texto);
```



Lógica de Programação

Conceitos | Variáveis | Estruturas | Exercícios de exemplo

Entrada | Processamento | Saída – Java Script

Entradas

Pode-se utilizar diversos métodos JS e/ou elementos HTML do DOM para se realizar a entrada de DADOS em um ambiente.

Inicialmente para fins didáticos, utilizaremos o seguinte:

```
prompt('Orientação para o usuário...')
```

Processamento

Depende da regra de negócio, ou seja, a lógica do algoritmo que pode ser construída por meio da manipulação de variáveis, cálculos e comparações. Esta parte do algoritmo pode ser estimulada por eventos dentro. Os eventos que pode-se utilizar no Java Script será alvo de outras aulas posteriores.

Saída

Assim como no caso das entradas, as saídas também podem utilizar elementos HTML do DOM para realizar as saídas. Mas também para fins didáticos utilizaremos o:

```
document.write('Exibição na tela...')
```

Estrutura de decisão - Java Script

```
<script>
    //Sintaxe if else
    var a=0, b=5, c=7;
    if (a==0){
        a=b+c;
    } else if (a>0){
        a+=1;
    } else{
        a=0;
    }
</script>
```

```
switch (new Date().getDay()) {
    case 0:
        day = "Sunday";
        break;
    case 1:
        day = "Monday";
        break;
    case 2:
        day = "Tuesday";
        break;
    case 3:
        day = "Wednesday";
        break;
    case 4:
        day = "Thursday";
        break;
    case 5:
        day = "Friday";
        break;
    case 6:
        day = "Saturday";
}
```

If ternário:

O Operador Ternário (`? : no javascript`) verifica uma condição e retorna um dentre dois valores pré-definidos em sua estrutura. Esta é uma notação inline para as vezes em que seja necessário avaliar expressões e decidir por um dentre 2 valores.

```
<script>
    //Sintaxe
    var a=0, b=0;
    var c;
    c = a==b ? "caso verdadeiro": "caso falso";
    /* If ternário: Váriavel = teste lógico
     * ? do possível caso verdadeiro
     * : seguido do valor caso falso;*/
</script>
```

Fazer um programa para ler as medidas da largura e comprimento de um terreno retangular, e o preço (valor venal) do m² do terreno. O programa deve calcular e exibir o tamanho do terreno em m² e também o preço de venda do terreno e retorne os dois valores como saída.

- Exemplo de execução.

```
Console simulando o modo texto do MS-DOS

Digite a largura do terreno: 12.0
Digite o comprimento do terreno: 20.0
Digite o valor do metro quadrado: 150.00
Area do terreno = 240.00
Preco do terreno = 36000.00

>>> Fim da execução do programa !
```

```
1 Algoritmo "terreno"
2
3 Var
4
5     largura, comprimento, metroQuadrado, area,
6
7 Inicio
8
9     escreva("Digite a largura do terreno: ")
10    leia(largura)
11    escreva("Digite o comprimento do terreno: ")
12    leia(comprimento)
13    escreva("Digite o valor do metro quadrado: ")
14    leia(metroQuadrado)
15
16    area <- largura * comprimento
17    preco <- area * metroQuadrado
18
19    escreval("Area do terreno = ", area:8:2)
20    escreval("Preco do terreno = ", preco:8:2)
21
22 Fimalgoritmo
23
```

53
53 Finalgoritmo
54
55

Fazer um programa para ler três números inteiros e mostra qual é o menor número. Em caso de empate deve mostrar o número apenas uma vez.

Console simulando o modo texto do MS-DOS

```
Primeiro valor: 9
Segundo valor: 9
Terceiro valor: 9
MENOR = 9

>>> Fim da execução do programa !
```

```
1 Algoritmo "menor_de_tres"
2
3 Var
4
5     a, b, c, menor : inteiro
6
7 Inicio
8
9     escreva("Primeiro valor: ")
10    leia(a)
11    escreva("Segundo valor: ")
12    leia(b)
13    escreva("Terceiro valor: ")
14    leia(c)
15
16    se (a < b) e (a < c) entao
17        menor <- a
18    senao
19        se b < c entao
20            menor <- b
21        senao
22            menor <- c
23        fimse
24    fimse
25
26    escreval("MENOR = ", menor)
27
28 Fimalgoritmo
29
```

4-) Fazer um programa para ler um valor inteiro de 1 a 7 representando um dia da semana (sendo 1=domingo, 2=segunda, e assim por diante). Escrever na tela o dia da semana correspondente, conforme exemplos.

```
Console simulando o modo texto do MS-DOS

1
Dia da semana: domingo

>>> Fim da execução do programa !
```

```
1 Algoritmo "teste_dias"
2
3 Var
4   x : inteiro
5   dia : caractere
6
7 Inicio
8   leia(x)
9
10  escolha x
11  caso 1
12    dia <- "domingo"
13  caso 2
14    dia <- "segunda"
15  caso 3
16    dia <- "terca"
17  caso 4
18    dia <- "quarta"
19  caso 5
20    dia <- "quinta"
21  caso 6
22    dia <- "sexta"
23  caso 7
24    dia <- "sabado"
25  outrocaso
26    dia <- "valor invalido"
27  fimescolha
28
29  escreval("Dia da semana: ", dia)
30 Fimalgoritmo
31
32 Entrada
33   escreval("Dia da semana: ", qqq)
34
```

Repetição ou Loop

Permitem executar mais de uma vez um mesmo trecho de código. Trata-se de uma forma de executar blocos de comandos somente sob determinadas condições, mas com a opção de repetir o mesmo bloco quantas vezes forem necessárias.

As estruturas de repetição são úteis, por exemplo, para repetir uma série de operações semelhantes que são executadas para todos os elementos de uma lista ou de uma tabela de dados, ou simplesmente para repetir um mesmo processamento até que uma certa condição seja satisfeita.

Há três modelos básicos de estrutura de repetição:
repetição com variável de controle, repetição com teste no início e repetição com teste no final.

Resumo

| PARA → FAÇA → FIMPARA
| ENQUANTO → FAÇA → FIMENQUANTO
| REPITA → ATÉ

Exemplo:

```
...
Var
n, i, soma de inteiro
Inicio
...
soma <- 0
...
Enquanto (i<=7) faca
    escreva("Entre com o número: ")
    leia(n)
    soma <- soma + n
    i=i+1
FimEnquanto

escreva("Soma: ",soma)
escreva("Media: ",soma/7)
```

Exemplo:

```
...
Var
n, i, soma de inteiro
Inicio
...
soma <- 0
...
para i de 1 ate 7 faca
    escreva("Entre com o número: ")
    leia(n)
    soma <- soma + n
FimPara

escreva("Soma: ",soma)
escreva("Media: ",soma/7)
```

Exemplo:

```
...
Var
n, i, soma de inteiro
Inicio
...
i<-1
soma <- 0
Repita
    escreva("Entre com o número: ")
    leia(n)
    soma <- soma + n
    i=i+1
Até (i>=7)

escreva("Soma: ",soma)
escreva("Media: ",soma/7)
```

Java Script - While

```
<script>
  //Sintaxe Laço com teste no início
  var x = 1
  document.write('inicio <hr>')
  //Código...
  while(x<=10){
    //Código...
    document.write(x + '<br>');
    x++
  }
  //Código...
  document.write('fim <hr>')
</script>
```

```
<script>
  //Formas de escape dos Laços
  // continue e break
  var x = 1
  document.write('inicio <hr>')
  //Código...
  while(x<=10){
    //Código...
    if(x==5){
      x++; //Necessário - se não irá criar laços infinitos
      continue;
    }
    document.write(x + '<br>');
    x++
  }
  //Código...
  document.write('fim <hr>')
</script>
```

```
<script>
  //Formas de escape dos Laços
  // continue e break
  var x = 1
  document.write('inicio <hr>')
  //Código...
  while(x<=10){
    //Código...
    if(x==5){
      break;
    }
    document.write(x + '<br>');
    x++
  }
  //Código...
  document.write('fim <hr>')
</script>
```

Java Script – Do While

```
<script>
    // Sintaxe do laço com teste no final
    var x = 1;
    do {
        //Código...
        document.write(x + '<br>')
        x++;
        //break e continue funcionam de mesma forma.
    } while(x <= 10)
</script>

</head>
```

```
<script>
    // Sintaxe do laço com teste no final
    var x = 1;
    do {
        //Código...
        document.write(x + '<br>')
        x+=5; //Incrementação variada
        //break e continue funcionam de mesma forma.
    } while(x <= 100)
</script>
```

```
<script>
    var x=10;
    document.write('Inicio <hr>')
    do {
        document.write(x+'<br>')
        x-=1; //Decrementação
    } while(x>=0)
    document.write('Final <hr>')
</script>
```

Java Script – For

```
//Sintaxe de laço com quantidade de loops determinados

for (var i=1; i<=10; i++){ //Var. de controle declarada na estrutura
    //Código
    document.write(x+'<br>')
}
div.appendChild(lista);
```

```
<script>
//Sintaxe de laço com quantidade de loops determinados
for (var x=1; x>=100; x+=5){ //Variável incrementação variada
    //Código...
    document.write(x + '<br>')
    //Não tem variavel contadora
}

</script>
```

```
<script>
//Sintaxe de laço com quantidade de loops determinados
for (var x=10; x>=0; x-=1){ //Exemplo com decrementação
    //Código...
    document.write(x + '<br>')
    //Não tem variavel contadora
}

</script>
```

Criando novos elementos HTML no DOM

```
<script>
  var div = document.querySelector('#divLista');
  var lista = document.createElement('ul');

  for (var i=1; i<=10; i++){
    //Cria um nó de texto
    let texto = document.createTextNode(`Item${i}`);

    //Cria um Elemento HTML - No exemplo um item de lista
    let item = document.createElement('li');

    //Insere o nó de texto em um elemento
    item.appendChild(texto);

    //Insere um elemento como filho
    lista.appendChild(item); //No exemplo um item em uma lista
  }
  //Insere um elemento como filho
  div.appendChild(lista); //No exemplo uma lista em uma div
</script>
```

The screenshot shows the browser's developer tools with the 'Elements' tab selected. The title bar says 'Exemplo AppendChild'. The DOM tree on the right side shows the following structure:

- <!DOCTYPE html>
- <html lang="pt-br">
- <head>...</head>
- <body>
- <div id="divLista">
-
- Item1
- Item2
- Item3
- Item4
- Item5
- Item6
- Item7
- Item8
- Item9
- Item10

The 'Elements' tab has a toolbar with icons for file, refresh, search, and other options. The status bar at the bottom shows the path 'Arquivo | D:/Desktop/exploJS/exemplo-app...'.

Ler um número inteiro N, daí mostrar na tela a tabuada de N para 1 a 10.

Exemplo de execução.

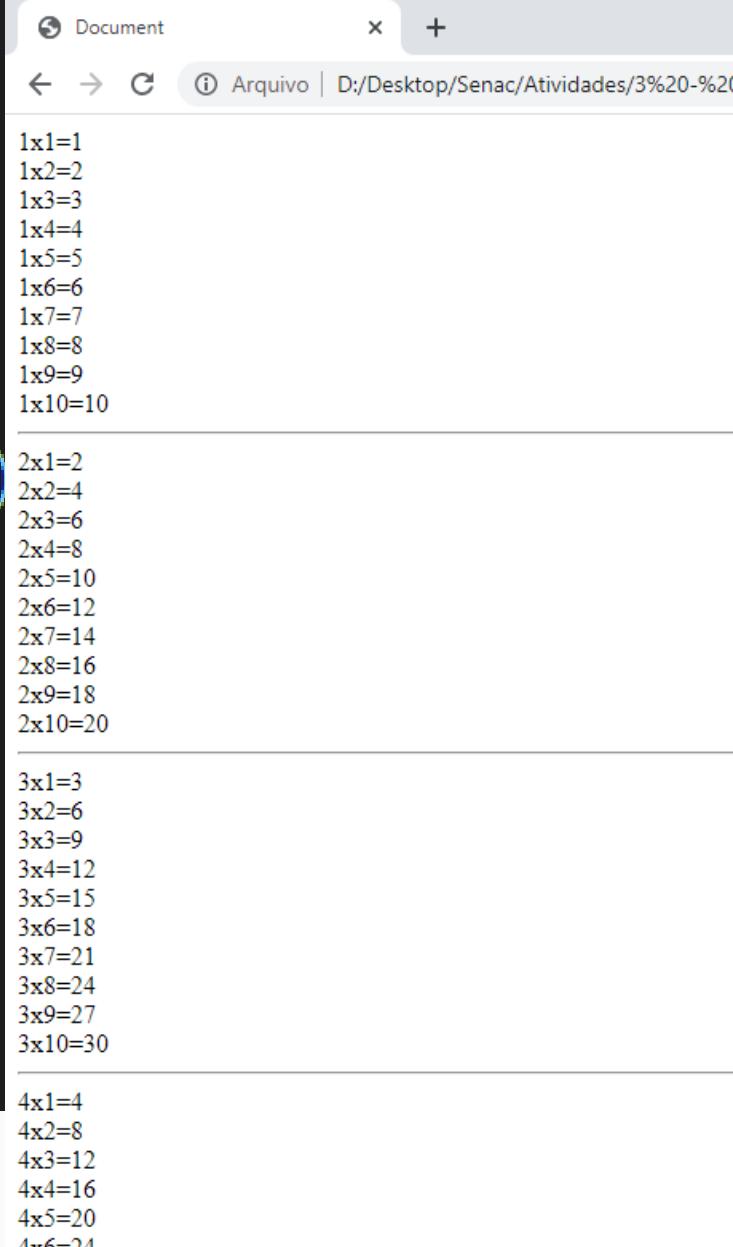
C:\ Console simulando o modo texto do MS-DOS

```
Valor que deseja calcular a tabuada:  
6  
6X 1= 6  
6X 2= 12  
6X 3= 18  
6X 4= 24  
6X 5= 30  
6X 6= 36  
6X 7= 42  
6X 8= 48  
6X 9= 54  
6X 10= 60  
  
>>> Fim da execução do programa !
```

```
2 Algoritmo "tabuada"  
3  
4 Var  
5 n, i, r : inteiro  
6  
7 Inicio  
8 //Entrada  
9 escreval("Valor que deseja calcular a tabuada: ")  
10 leia (n)  
11 para i de 1 ate 10 faca  
12 //Processamento  
13     r<- n* i  
14 //saída  
15     escreval (N, "X", i, "=", r)  
16 fimPara  
17  
18 Fimalgoritmo
```

```
<script>
  for (var y = 1; y <= 10; y++) {
    for (var x = 1; x <= 10; x++) {
      document.write(y + 'x' + x + ' = ' + (y * x) + '<br>')
    }
    document.write('<hr>')
  }
</script>
```

Laços encadeados - Mostrar a tabuada de 1 a 10



The screenshot shows a browser window with the title "Document". The address bar indicates the file is located at "D:/Desktop/Senac/Atividades/3%20-%20". The page content displays the multiplication table for numbers 1 through 10, separated by horizontal lines and newlines.

1x1=1	1x2=2	1x3=3	1x4=4	1x5=5	1x6=6	1x7=7	1x8=8	1x9=9	1x10=10
2x1=2	2x2=4	2x3=6	2x4=8	2x5=10	2x6=12	2x7=14	2x8=16	2x9=18	2x10=20
3x1=3	3x2=6	3x3=9	3x4=12	3x5=15	3x6=18	3x7=21	3x8=24	3x9=27	3x10=30
4x1=4	4x2=8	4x3=12	4x4=16	4x5=20	4x6=24				

UTILIZAÇÃO DE VETORES

É uma estrutura de dados que armazena uma sequência consecutiva de valores do mesmo tipo. Associado a estruturas de repetição se torna algo muito poderoso na otimização de rotinas de programação.

Sintaxe:

Var
VariavelControle de **inteiro**
NomeVetor : **vetor [IndiceInicial .. IndiceFinal] de TipoDado**

Para VariavelDeControle de ValorInicial ate ValorFinal faca

...

NomeVetor[VariavelControle]

...

FimPara

Exemplo:

...

Var

nome : **vetor [0..9] de Caracter**

nota: **vetor [0..9] de real**

i de inteiro

Inicio

...

para i de 0 ate 9 faca

...

escreva("Informe o nome: ")

leia(nome[i])

escreva("Informe a nota: ")

leia(nota[i])

....

FimPara

...

FimAlgoritmo

Java Script – Sintaxe da declaração de Arrays

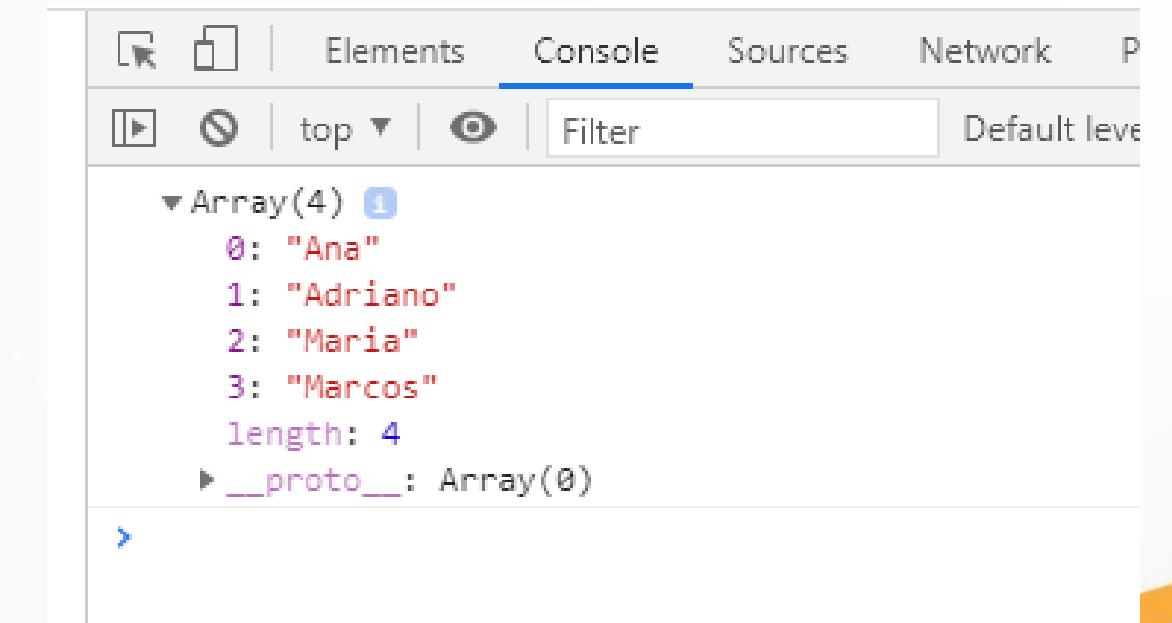
```
<script>
//Formas semântica, utilização e valoração de ARRAYS
var listaAlunos = Array();
listaAlunos[0] = 'Ana';
listaAlunos[1] = 'Adriano';
listaAlunos[2] = 'Maria';
listaAlunos[3] = 'Marcos';
</script>
```

```
<script>
//Formas semântica, utilização e valoração de ARRAYS
var listaAlunos = Array(
  'Ana',
  'Adriano',
  'Maria', 'Marcos'
)

</script>
```

```
<script>
//Formas semântica, utilização e valoração de ARRAYS
var listaAlunos = ['Ana', 'Adriano', 'Maria', 'Marcos']

</script>
```



Personalização dos índices e atributo length - Arrays em Java Script

```
<script>
//Formas semântica, utilização e val
var listaAlunos = Array();
listaAlunos[100] = 'Ana';
listaAlunos[200] = 'Adriano';
listaAlunos[300] = 'Maria';
listaAlunos[400] = 'Marcos';
length: 401
console.log(listaAlunos)
</script>
```

```
<script>
//Formas semântica, utilização e val
var listaAlunos = Array();
listaAlunos[0] = 'Ana';
listaAlunos[1] = 'Adriano';
listaAlunos[3] = 'Marcos';
x: "Maria"
length: 4
__proto__: Array(0)
>
```

```
<script>
//Formas semântica, utilização e val
var listaAlunos = Array();
listaAlunos[0] = 'Ana';
listaAlunos[1] = 'Adriano';
listaAlunos['x'] = 'Maria';
listaAlunos[3] = 'Marcos';
console.log(listaAlunos)
</script>
```

```
<script>
//Formas semântica, utilização e val
var listaAlunos = Array(
  'Ana',
  'Adriano',
  'Maria',
  'Marcos'
);
console.log(listaAlunos)
console.log(listaAlunos.length)
>
```

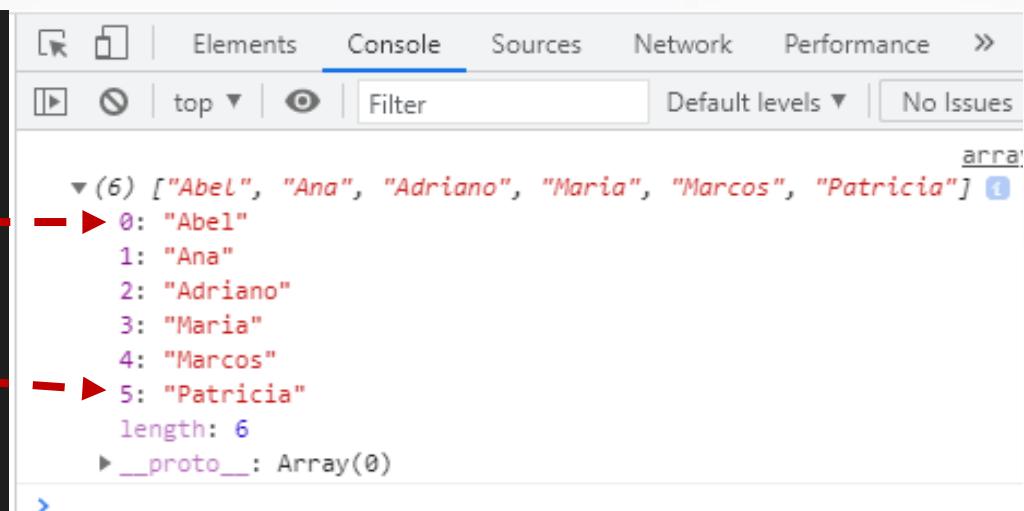
```
<script>
//Formas semântica, utilização e val
var listaAlunos = Array();
listaAlunos[1] = 'Ana';
listaAlunos[2] = 'Adriano';
listaAlunos[3] = 'Maria';
listaAlunos[4] = 'Marcos';
console.log(listaAlunos)
</script>
```

```
<script>
//Formas semântica, utilização e val
var listaAlunos = Array();
1: "Ana"
2: "Adriano"
3: "Maria"
4: "Marcos"
length: 5
__proto__: Array(0)
>
```

Inserir Elementos Dinamicamente

```
<script>
//Multidimensionais ARRAYS
var listaAlunos = Array('Ana', 'Adriano', 'Maria', 'Marcos');
//Adiciona item no inicio do array
listaAlunos.unshift('Abel');
//Adiciona item do final do array
listaAlunos.push('Patricia');
console.log(listaAlunos)

</script>
```



The screenshot shows the Chrome DevTools interface with the 'Console' tab selected. A red dashed box highlights the console output where the array is logged. To the right, the array's properties are expanded, showing elements from index 0 to 5, a length of 6, and a prototype.

```
array(6) ["Abel", "Ana", "Adriano", "Maria", "Marcos", "Patricia"]
  0: "Abel"
  1: "Ana"
  2: "Adriano"
  3: "Maria"
  4: "Marcos"
  5: "Patricia"
  length: 6
  __proto__: Array(0)
```

Excluir Elementos Dinamicamente

```
Elements      Console      Sources
top ▾ Filter
array(2) ["Adriano", "Maria"]
  0: "Adriano"
  1: "Maria"
  length: 2
  __proto__: Array(0)
```

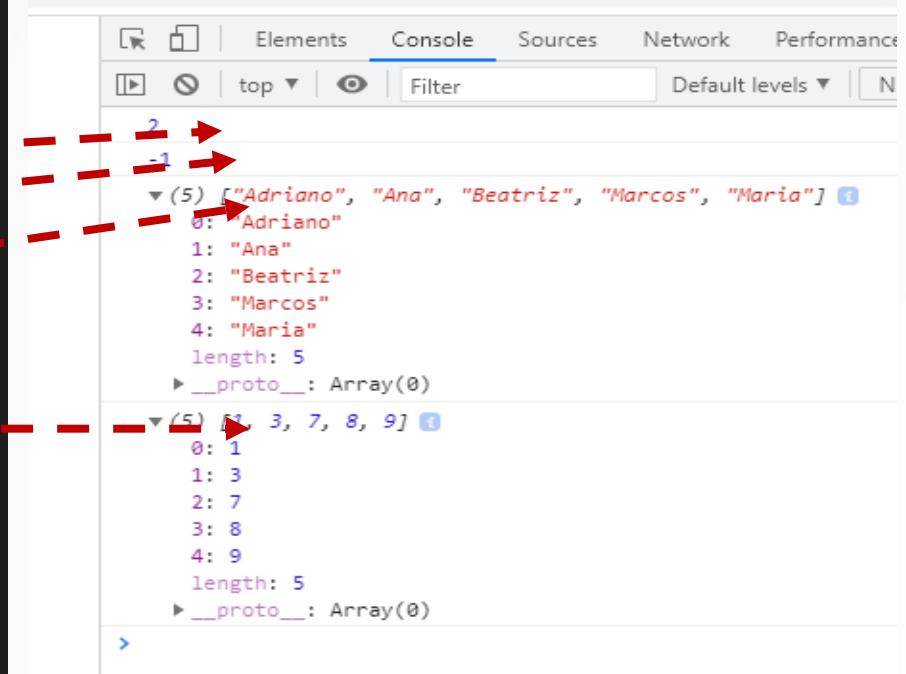
```
<script>
//Multidimensionais ARRAYS
var listaAlunos = Array('Ana', 'Adriano', 'Maria', 'Marcos');
//Exclui item no inicio do array
listaAlunos.shift();
//Exclui item do final do array
listaAlunos.pop();
console.log(listaAlunos)
</script>
```

Pesquisa e Ordenação de Elementos

```
<script>
//Pesquisa e ordenação de elementos no array
var listaAlunos = Array('Ana', 'Adriano', 'Maria', 'Marcos', 'Beatriz');
//Retorna o índice
console.log(listaAlunos.indexOf('Maria'));
//Caso não exista, retorna -1
console.log(listaAlunos.indexOf('Aline'));

//Ordenação em ordem alfabética
console.log(listaAlunos.sort())

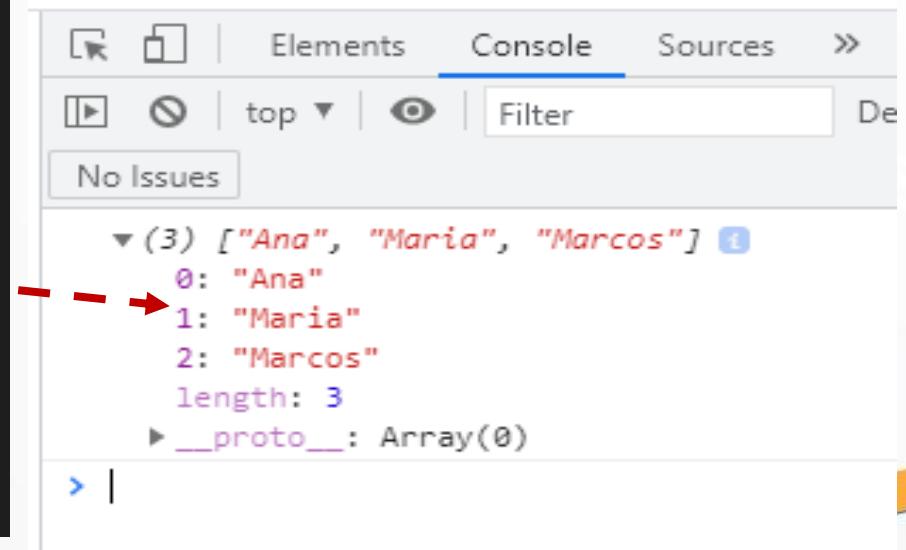
//Ordenação numérica
var numerosAleatorios = Array(3, 1, 9, 7, 8)
console.log(numerosAleatorios.sort())
/*function ordemNumerica(a, b){return a-b}
   Pode ser necessários em alguns navegadores um
   ajuste com esta função sendo chamada dentro
   do método sort*/
</script>
```



```
(5) ["Adriano", "Ana", "Beatriz", "Marcos", "Maria"]
  0: "Adriano"
  1: "Ana"
  2: "Beatriz"
  3: "Marcos"
  4: "Maria"
  length: 5
  ▶ __proto__: Array(0)

(5) [1, 3, 7, 8, 9]
  0: 1
  1: 3
  2: 7
  3: 8
  4: 9
  length: 5
  ▶ __proto__: Array(0)
```

```
<script>
  //Excluir elemento em outras posições dentro do vetor
  var listaAlunos = Array('Ana', 'Adriano', 'Maria', 'Marcos');
    /*O método indexOf encontra o índice do elemento a ser apagado e o
     | método splice que apaga o elemento.*/
    listaAlunos.splice(listaAlunos.indexOf('Adriano'), 1);
    /*O 1º parâmetro a ser passado para o splice é o índice
     (no exemplo indexOf('Adriano') retornará 1 ) do elemento e
     o segundo é o número(quantidade) de elementos a serem apagados,
     no exemplo 1*/
</script>
```



```
(3) ["Ana", "Maria", "Marcos"]
  0: "Ana"
  1: "Maria"
  2: "Marcos"
  length: 3
  ▶ __proto__: Array(0)
```

Arrays Multidimensionais(matrizes) – Java Script

```
<script>
//Multidimensionais ARRAYS
var cursos = Array();
//Sintaxe 1
cursos['TecInformatica'] = Array();
    cursos['TecInformatica'][0] = 'Ana';
    cursos['TecInformatica'][1] = 'Adriano';
    cursos['TecInformatica'][3] = 'Maria';
    cursos['TecInformatica'][4] = 'Marcos';
//Sintaxe 2
cursos['TecRedes'] = Array('Beatriz','Fábio','Mariana','Marcelo');

console.log(cursos)
</script>
```

```
console.log(cursos['TecInformatica'])
```

```
console.log(cursos['TecRedes'])
```

```
Array(0)
  TecInformatica: Array(5)
    0: "Ana"
    1: "Adriano"
    3: "Maria"
    4: "Marcos"
    length: 5
  > __proto__: Array(0)
  TecRedes: Array(4)
    0: "Beatriz"
    1: "Fábio"
    2: "Mariana"
    3: "Marcelo"
    length: 4
  > __proto__: Array(0)
  > __proto__: Array(0)
```

cursos
TecInformatica Array()
TecRedes Array()

TecInformatica

Ana

Adriano

Maria

Marcos

TecRedes

Beatriz

Fabio

Mariana

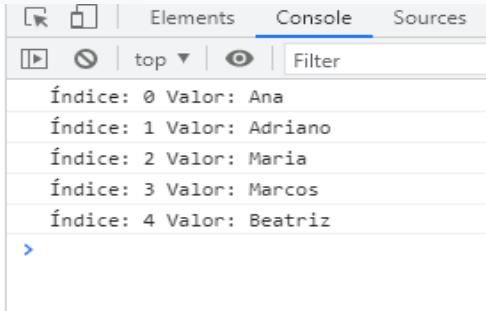
Marcelo

Percorrendo Arrays

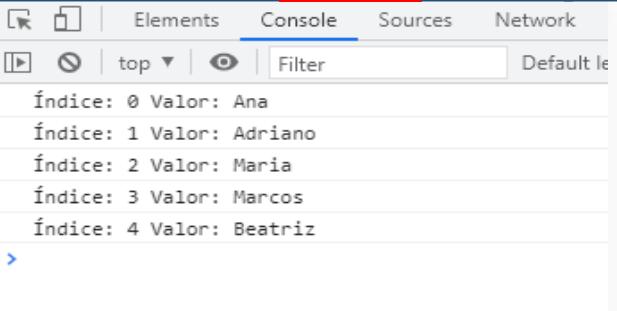
```
<script>
//Varrendo vetor com For In
var listaAlunos = Array('Ana', 'Adriano', 'Maria', 'Marcos', 'Beatriz');

for (var x in listaAlunos){
    console.log('Índice: ' + x + ' Valor: ' + listaAlunos[x]);
}
</script>
```

For In – Java Script



ForEach – Java Script

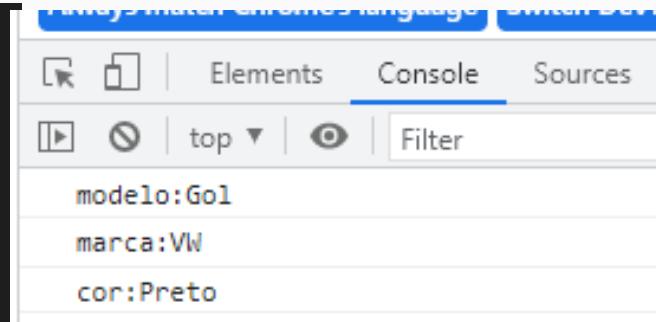


```
<script>
//Varrendo vetor com forEach
var listaAlunos = Array('Ana', 'Adriano', 'Maria', 'Marcos', 'Beatriz');

listaAlunos.forEach(function(valor, indice){
    console.log('Índice: ' + indice + ' Valor: ' + valor)
})
</script>
```

ForEach para Objetos – Java Script

```
var carro = {modelo:'Gol', marca:'VW', cor:'Preto'};
Object.keys(carro).forEach((item) => {
    console.log(item+':'+carro[item]);
});
```





Java Script

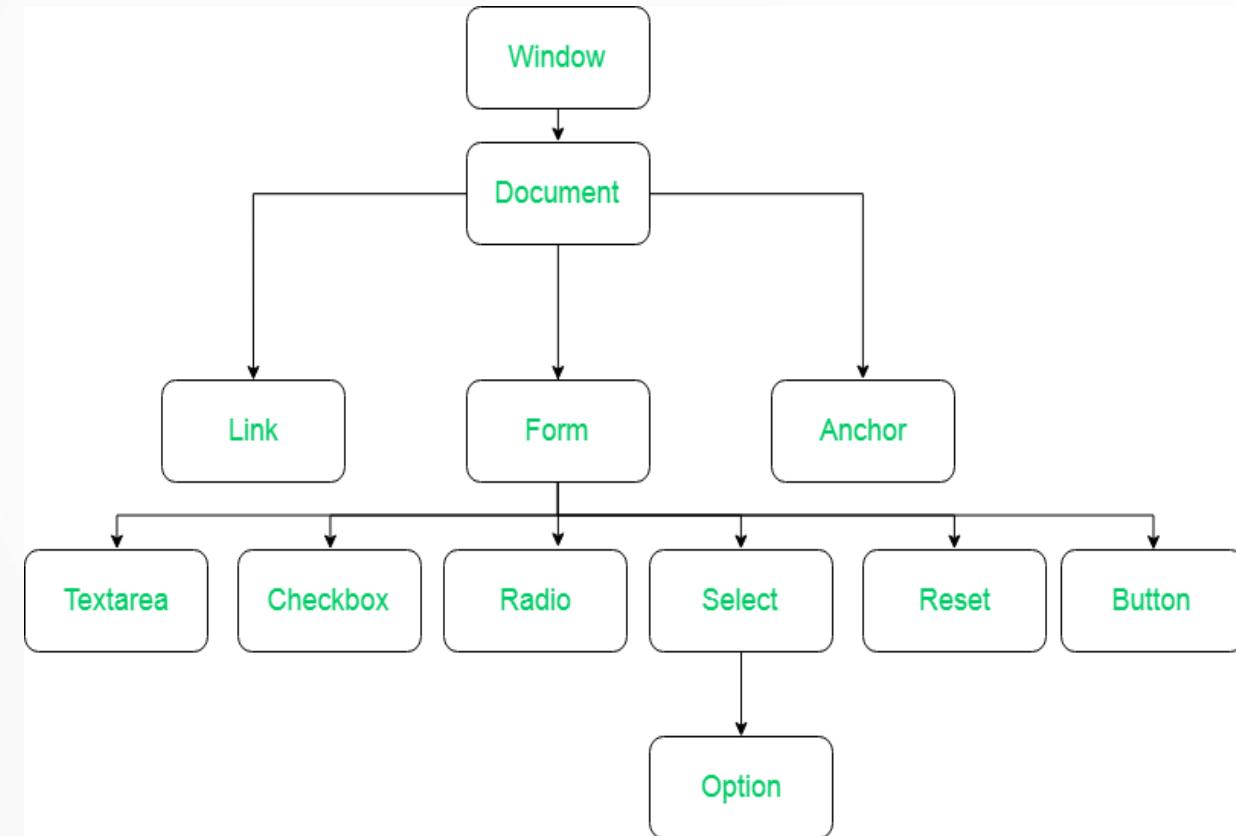
BOM – Browser Object Model

BOM – Browser Object Model

BOM – API que permite o acesso e disparar recursos do browser, abrir e fechar da janelas, forçar cookies forçar o carregamento de URLs, dentre outras funcionalidades;

https://www.w3schools.com/js/js_window.asp

https://www.w3schools.com/jsref/obj_window.asp



```
//Método open(), close() e print() do BOM
// - open() - Abrir uma janela PopUp
var janela; //Variavel de controle da janela criada
function abrirPopUp (){
    janela = window.open(
        'http://sp.senac.br/catanduva', //URL interna ou externa
        'Senac Catanduva', //Nome da janela
        'width=400', // Largura da janela em pixels
        'height=200' // Altura da janela em pixels
    )
}
// - close() - Fechar janela que foi aberta
function fecharJanela (){
    janela.close(); //Nativo de janelas
}
// - print() - Imprimir paginas
function imprimirPagina(){
    window.print();
}
```

Exemplos de utilização - BOM

Objeto Windows

Exemplos de utilização - BOM

Objeto Screen

```
/* Metodos e propriedades para recuperar  
a altura e largura da janela */  
var larguraMax = window.screen.availWidth;  
var alturaMax = window.screen.availHeight;  
var larguraAtual = window.screen.width;  
var alturaAtual = window.screen.height;  
/*Pode-se fazer teste simulando nas ferramentas  
de desenvolvedor tamanho de telas de dispositivos  
diferentes */
```

Exemplos de utilização - BOM

Objeto Location

```
/*Objeto location - redirecionamento de URL, atualizar */
//redirecionamento de URL de forma programática
window.location.href="http://sp.senac.br/catanduva"
//atualizar página
function atualizarPagina(){
    window.location.reload();
}
```

Exemplos de utilização - BOM

Método:
setTimeout()

Método:
setInterval();

```
//Função utilizando intervalos de tempo

/*Função setTimeout() executa a ação apenas uma única vez
uma função dentro de um intervalo de tempo */
/* Sintaxe:
setTimeout(function(){//ações}, tempo) */
function eventoExecutado() {
    setTimeout(function() {
        //Rotina a ser disparada 5 segundos após o evento;
    }, 5000)
}
/*Função setInterval() executa a ação repetidamente
uma função dentro de um intervalo de tempo */
/* Sintaxe:
setInterval(function(){//ações}, tempo) */
function eventoExecutado(){
    setInterval(function(){
        //Rotina a ser repetida a cada 1 segundo após o evento;
    }, 1000)
}
```

Exemplos de utilização - BOM

Método: SetTimeout()

```
//Expirar a sessão após 5 segundos
setTimeout(function() {
    window.location.reload();
}, 5000)
```

```
//Lógica para um cronômetro
var i=10;
/*Necessário encapsular a função em uma variável
para executar o método clearInterval que finaliza a execução*/
cronometro = setInterval(function(){
    document.querySelector('#cronometro').innerHTML = i;
    i--;
    if (i==0){
        clearInterval(cronometro);
    }
}, 1000)
```

Método: SetInterval();



Objetos Literais e JSON

Java Script

Objetos

Objetos Literais do JavaScript são coleções dinâmicas de dados ordenados por pares de chave/Valor, formando assim as propriedades do objeto.

As propriedades de um objeto definem as características do objeto. Podem ser explicadas como uma variável que é ligada ao objeto. São basicamente as mesmas que variáveis normais em JavaScript, exceto pelo fato de estarem ligadas a objetos.

As propriedades de um objeto com uma simples notação de ponto:

```
nomeDoObjeto.nomeDaPropriedade
```

https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Working_with_Objects

Formas de declarar/criar Objetos Literais

```
//Declaração por meio da classe Object  
var calsado = new Object  
calsado.descricao = 'Tenis Masculino';  
calsado.marca='Pizante';  
calsado.tamanho='41';  
calsado.cor='Branco';  
calsado.preco=300;  
console.log(calsado);
```

```
//Removendo atribuutos do Objeto  
delete calsado.cor;  
delete calsado.preco;  
console.log(calsado);
```

poo.html:22

```
{descricao: 'Tenis Masculino', marca: 'Pizante', tamanho:  
  '41', cor: 'Branco', preco: 300} ⓘ  
  descricao: "Tenis Masculino"  
  marca: "Pizante"  
  tamanho: "41"  
  ► [[Prototype]]: Object
```

poo.html:27

```
{descricao: 'Tenis Masculino', marca: 'Pizante', tamanho:  
  '41'} ⓘ  
  descricao: "Tenis Masculino"  
  marca: "Pizante"  
  tamanho: "41"  
  ► [[Prototype]]: Object
```

Formas de declarar/criar Objetos Literais

```
var carro = {
    marca:'Ferrari',
    modelo:'488 PISTA SPIDER',
    cor:'Vermelha',
    valor:4000000.00,
    proprietario: {
        nome: 'Silvio Santos',
        cpf:'333.333.333-33',
        idade: 90,
        endereco: {
            logradouro:'Rua dos Pombos, n.000',
            cidade:'São Paulo'
        }
    },
    condutores: [
        {nome:'Silvia Santos',
        idade:60},
        {nome:'Patricia Santos',
        idade:45}
    ]
}
console.log(carro);
```

poo.html:37

```
{marca: 'Ferrari', modelo: '488 PISTA SPIDER', cor: 'Vermelha', valor: 4000000,
proprietario: {...}, ...} ⓘ
▼condutores: Array(2)
▶ 0: {nome: 'Silvia Santos', idade: 60}
▶ 1: {nome: 'Patricia Santos', idade: 45}
length: 2
▶ [[Prototype]]: Array(0)
cor: "Vermelha"
marca: "Ferrari"
modelo: "488 PISTA SPIDER"
▼proprietario:
    cpf: "333.333.333-33"
▼endereco:
    cidade: "São Paulo"
    logradouro: "Rua dos Pombos, n.000"
    ▶ [[Prototype]]: Object
    idade: 90
    nome: "Silvio Santos"
    ▶ [[Prototype]]: Object
    valor: 4000000
    ▶ [[Prototype]]: Object
```

Formas de acessar Objetos

```
//Ler(get) os valores de um objeto
console.log(carro);
console.log(carro.marca);
console.log(carro.modelo);
console.log(carro.proprietario.nome);
console.log(carro.proprietario.endereco.logradouro);
console.log(carro.condutores[0].nome)
```

```
//Modificar(set) os valores de um objeto
carro.marca = 'Lamborgini';
carro.modelo = 'Aventador';
carro.valor = 3000000;
carro.proprietario.nome = 'Leonardo';
carro.proprietario.endereco.cidade = 'Catanduva';
console.log(carro);
```

poo.html:38

```
{marca: 'Ferrari', modelo: '488 PISTA SPIDER', cor: 'Vermelha', valor: 4000000, proprietario: {...}, ...}
```

Ferrari
488 PISTA SPIDER
Silvio Santos
Rua dos Pombos, n.000
Patricia Santos

poo.html:39

poo.html:40

poo.html:41

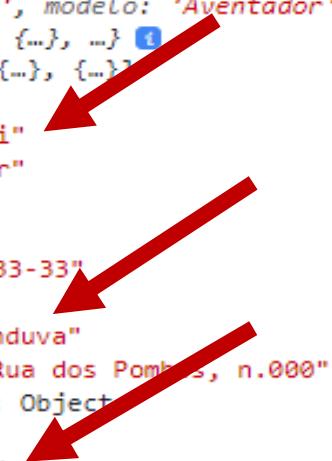
poo.html:42

poo.html:43

```
> |
```

poo.html:43

```
{marca: 'Lamborgini', modelo: 'Aventador', cor: 'Vermelha', valor: 4000000, proprietario: {...}, ...} i
  ▶ condutores: (2) [{...}, {...}]
  cor: "Vermelha"
  marca: "Lamborgini"
  modelo: "Aventador"
  preco: 3000000
  proprietario:
    cpf: "333.333.333-33"
    endereço:
      cidade: "Catanduva"
      logradouro: "Rua dos Pombos, n.000"
      ▶ [[Prototype]]: Object
      idade: 90
      nome: "Leonardo"
    proprietario:
      [[Prototype]]: Object
      valor: 4000000
    [[Prototype]]: Object
```



Funções para Manipular Objetos

```
11 var heroi = {  
12     Nome:"SuperMan",  
13     IdentidadeSecreta:"Clark Kent",  
14     Poderes:[  
15         "voar",  
16         "visão raio X",  
17         "super força",  
18         "super sopro",  
19         "super velocidade"  
20     ],  
21     Fraqueza:"kriptonita",  
22     TemMedo:"Lois Lane"  
23 }  
24 //Algumas funções interessantes para manipular objetos  
25 console.log('Objeto Inteiro')  
26 console.log(heroi)  
27  
28 console.log('Apenas chaves: Object.keys ')  
29 console.log(Object.keys(heroi))  
30  
31 console.log('Apenas valores: Object.values ')  
32 console.log(Object.values(heroi))  
33  
34 console.log('Vetores com chave/valor: Object.entries')  
35 console.log(Object.entries(heroi))  
36  
37 console.log("Varrendo entries com forEach:")  
38 Object.entries(heroi).forEach(([chave, valor])=>{  
39     console.log(`#${chave}: ${valor}`)  
40 }  
41  
42 //A função freeze de um objeto congela a alteração,  
43 //Tornando o objeto e a referência da memória uma constante  
44 Object.freeze(heroi)
```

Default levels ▾ || No Issues ||

objetoLiteral.html:24
objetoLiteral.html:25
objetoLiteral.html:26
objetoLiteral.html:27
objetoLiteral.html:28
objetoLiteral.html:29
objetoLiteral.html:30
objetoLiteral.html:31
objetoLiteral.html:32
objetoLiteral.html:33
objetoLiteral.html:34
objetoLiteral.html:34
objetoLiteral.html:34
objetoLiteral.html:34

Objeto Inteiro

Nome: "SuperMan", IdentidadeSecreta: "Clark Kent", Poderes: Array(5), Fraqueza: "kriptonita", TemMedo: "Lois Lane"

Poderes: (5) ["voar", "visão raio X", "super força", "super sopro", "super velocidade"]
TemMedo: "Lois Lane"
[[Prototype]]: Object

Apenas chaves: Object.keys

(5) ["Nome", "IdentidadeSecreta", "Poderes", "Fraqueza", "TemMedo"]
0: "Nome"
1: "IdentidadeSecreta"
2: "Poderes"
3: "Fraqueza"
4: "TemMedo"
length: 5
[[Prototype]]: Array(0)

Apenas valores: Object.values

(5) ["SuperMan", "Clark Kent", Array(5), "kriptonita", "Lois Lane"]
0: "SuperMan"
1: "Clark Kent"
2: (5) ["voar", "visão raio X", "super força", "super sopro", "super velocidade"]
3: "kriptonita"
4: "Lois Lane"
length: 5
[[Prototype]]: Array(0)

Vetores com chave/valor: Object.entries

(5) [Array(2), Array(2), Array(2), Array(2), Array(2)]
0: (2) ["Nome", "SuperMan"]
1: (2) ["IdentidadeSecreta", "Clark Kent"]
2: (2) ["Poderes", Array(5)]
3: (2) ["Fraqueza", "kriptonita"]
4: (2) ["TemMedo", "Lois Lane"]
length: 5
[[Prototype]]: Array(0)

Varrendo entries com forEach:

Nome: SuperMan
IdentidadeSecreta: Clark Kent
Poderes: voar,visão raio X,super força,super sopro,super velocidade
Fraqueza: kriptonita
TemMedo: Lois Lane

Funções para Manipular Objetos

```
11 var heroi = {  
12     Nome:"SuperMan",  
13     IdentidadeSecreta:"Clark Kent",  
14     Poderes:[  
15         "voar",  
16         "visão raio X",  
17         "super força",  
18         "super sopro",  
19         "super velocidade"  
20     ],  
21     Fraqueza:"kriptonita",  
22     TemMedo:"Lois Lane"  
23 }  
24 //Algumas funções interessantes para manipular objetos  
25 console.log('Objeto Inteiro')  
26 console.log(heroi)  
27  
28 console.log('Apenas chaves: Object.keys ')  
29 console.log(Object.keys(heroi))  
30  
31 console.log('Apenas valores: Object.values ')  
32 console.log(Object.values(heroi))  
33  
34 console.log('Vetores com chave/valor: Object.entries')  
35 console.log(Object.entries(heroi))  
36  
37 console.log("Varrendo entries com forEach:")  
38 Object.entries(heroi).forEach(([chave, valor])=>{  
39     console.log(` ${chave}: ${valor}`)  
40 })
```

Default levels ▾ || No Issues ||

Objeto Inteiro

Default levels ▾ || No Issues ||

objetoLiteral.html:24
objetoLiteral.html:25
objetoLiteral.html:26
objetoLiteral.html:27
objetoLiteral.html:28
objetoLiteral.html:29
objetoLiteral.html:30
objetoLiteral.html:31
objetoLiteral.html:32
objetoLiteral.html:33
objetoLiteral.html:34
objetoLiteral.html:34
objetoLiteral.html:34
objetoLiteral.html:34
objetoLiteral.html:34

Nome: SuperMan
IdentidadeSecreta: Clark Kent
Poderes: Array(5)
Fraqueza: kriptonita
TemMedo: Lois Lane
[[Prototype]]: Object

Apenas chaves: Object.keys
(5) ['Nome', 'IdentidadeSecreta', 'Poderes', 'Fraqueza', 'TemMedo']
0: "Nome"
1: "IdentidadeSecreta"
2: "Poderes"
3: "Fraqueza"
4: "TemMedo"
length: 5
[[Prototype]]: Array(0)

Apenas valores: Object.values
(5) ['SuperMan', 'Clark Kent', Array(5), 'kriptonita', 'Lois Lane']
0: "SuperMan"
1: "Clark Kent"
2: (5) ['voar', 'visão raio X', 'super força', 'super sopro', 'super velocidade']
3: "kriptonita"
4: "Lois Lane"
length: 5
[[Prototype]]: Array(0)

Vetores com chave/valor: Object.entries
(5) [Array(2), Array(2), Array(2), Array(2), Array(2)]
0: (2) ['Nome', 'SuperMan']
1: (2) ['IdentidadeSecreta', 'Clark Kent']
2: (2) ['Poderes', Array(5)]
3: (2) ['Fraqueza', 'kriptonita']
4: (2) ['TemMedo', 'Lois Lane']
length: 5
[[Prototype]]: Array(0)

Varrendo entries com forEach:
Nome: SuperMan
IdentidadeSecreta: Clark Kent
Poderes: voar,visão raio X,super força,super sopro,super velocidade
Fraqueza: kriptonita
TemMedo: Lois Lane

JSON – Formato de representação de DADOS

JSON é um formato de representação de dados muito utilizado para interoperabilidade(trota de informações) entre sistemas distintos.

Foi baseado na linguagem de programação Javascript, por isso o nome: Java Script Object Notation (Notação de Objeto em Javascript).

Os dois formatos de dados são os mais usados em aplicações WEB são:

JSON

```
1 {  
2     "Heroi":{  
3         "Nome": "SuperMan",  
4         "IdentidadeSecreta": "Clark Kent",  
5         "Poderes": [  
6             "voar",  
7             "visão raio X",  
8             "super força",  
9             "super sopro",  
10            "super velocidade"  
11        ],  
12        "Fraqueza": "kriptonita",  
13        "TemMedo": "Lois Lane"  
14    }  
15 }
```

XML

```
1 <Heroi>  
2     <nome>SuperMan</nome>  
3     <IdentidadeSecreta></IdentidadeSecreta>  
4     <Poderes>  
5         <Poder1>voar</Poder1>  
6         <Poder2>visão raio X</Poder2>  
7         <Poder3>super força</Poder3>  
8         <Poder4>super sopro</Poder4>  
9         <Poder5>super velocidade</Poder5>  
10    </Poderes>  
11    <Fraqueza>kriptonita</Fraqueza>  
12    <TemMedo>Lois Lane</TemMedo>  
13 </Heroi>
```

Vs.

Objeto Vs JSON

```
console.log(carro);
//Converter um objeto(ex. carro) em JSON
var carroJSON = JSON.stringify(carro);
console.log(carroJSON); //Saída de um formato textual(string).
```

```
//Formato JSON
const heroiJSON = `{
    "Heroi": {
        "Nome": "SuperMan",
        "IdentidadeSecreta": "Clark Kent",
        "Poderes": [
            "Voar",
            "Visão Raio X",
            "Super Força",
            "Super Sopro",
            "Super Velocidade"
        ],
        "Fraqueza": "Kriptonita",
        "TemMedo": "Lois Lane"
    }
}`;
//Extra Validador de JSON: https://jsonlint.com/
console.log(heroiJSON);

//Converter um JSON em um objeto JavaScript
var heroiObj = JSON.parse(heroiJSON);
console.log(heroiObj);
```

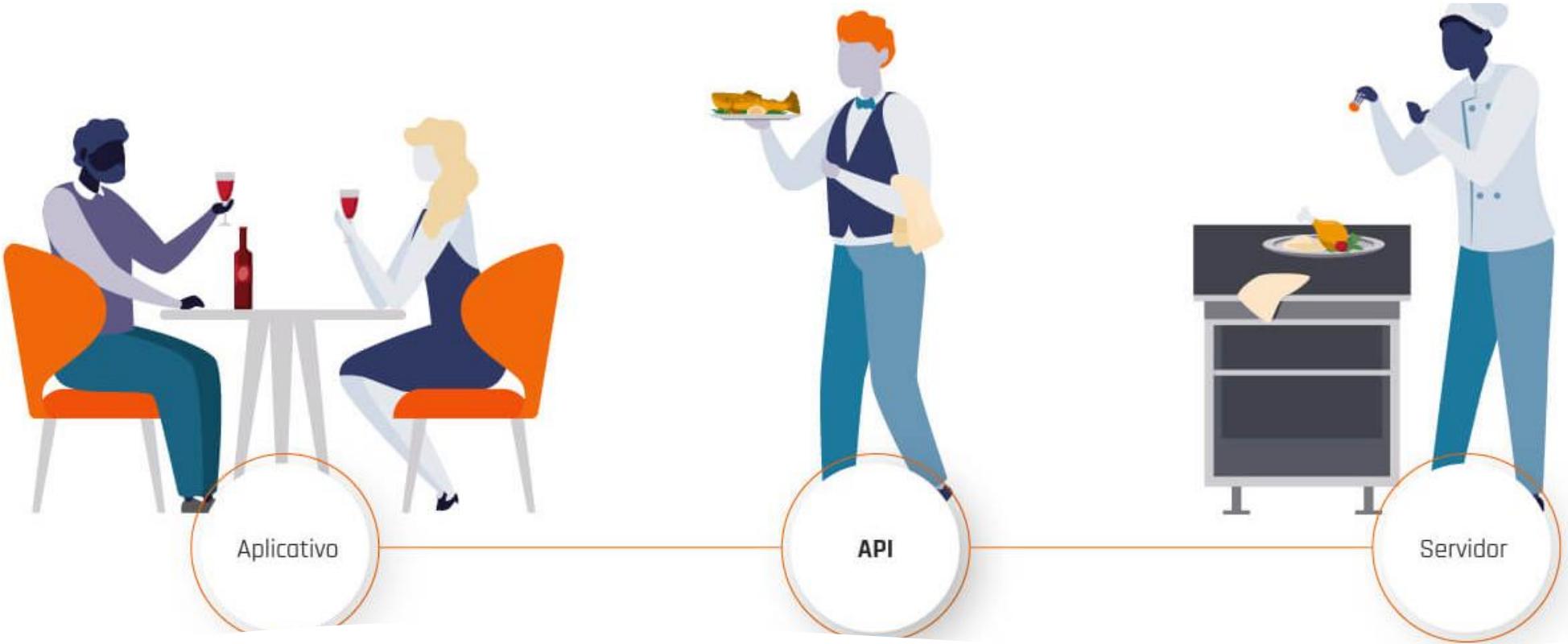
```
{marca: 'Ferrari', modelo: '488 PISTA SPIDER', cor: 'Vermelha', valor: 4000000, proprietario:  
  ↴ {}  
  ↴ ...  
  ↴ condutores: Array(2)  
    ↴ 0: {nome: 'Patricia Santos', idade: 60}  
    ↴ 1: {nome: 'Silvia Santos', idade: 45}  
    ↴ length: 2  
    ↴ [[Prototype]]: Array(0)  
  cor: "Vermelha"  
  marca: "Ferrari"  
  modelo: "488 PISTA SPIDER"  
  proprietario:  
    ↴ cpf: "333.333.333-33"  
    ↴ endereco: {logradouro: 'Rua dos Pombos, n.000', cidade: 'São Paulo'}  
    ↴ idade: 90  
    ↴ nome: "Silvio Santos"  
    ↴ [[Prototype]]: Object  
  valor: 4000000  
  [[Prototype]]: Object  
{"marca": "Ferrari", "modelo": "488 PISTA SPIDER", "cor": "Vermelha", "valor": 4000000, "proprietario": {"nome": "Silvio Santos", "cpf": "333.333.333-33", "idade": 90}, "endereco": {"logradouro": "Rua dos Pombos, n.000", "cidade": "São Paulo"}, "condutores": [{"nome": "Patricia Santos", "idade": 60}, {"nome": "Silvia Santos", "idade": 45}]}  
poo.html:54  
poo.html:57
```

```
[{"Heroi":{ "Nome":"SuperMan", "IdentidadeSecreta":"Clark Kent", "Poderes":[ "Voar", "Visão Raio X", "Super Força", "Supter Sopro", "Super Velocidade" ], "Fraqueza":"Kriptonita", "TemMedo":"Lois Lane" }}, {"Heroi:{...}":{ "Heroi":{ "Fraqueza": "Kriptonita", "IdentidadeSecreta": "Clark Kent", "Nome": "SuperMan", "Poderes": Array(5), "length": 5, "[[Prototype]]": Array(0), "TemMedo": "Lois Lane", "[[Prototype]]": Object, "[[Prototype]]": Object }}, "Heroi":{...}":{ "Heroi":{ "Fraqueza": "Kriptonita", "IdentidadeSecreta": "Clark Kent", "Nome": "SuperMan", "Poderes": Array(5), "length": 5, "[[Prototype]]": Array(0), "TemMedo": "Lois Lane", "[[Prototype]]": Object, "[[Prototype]]": Object }}, "Heroi":{...}":{ "Heroi":{ "Fraqueza": "Kriptonita", "IdentidadeSecreta": "Clark Kent", "Nome": "SuperMan", "Poderes": Array(5), "length": 5, "[[Prototype]]": Array(0), "TemMedo": "Lois Lane", "[[Prototype]]": Object, "[[Prototype]]": Object }}]
```



Java Script

Requisições HTTP – API - REST



API - Interface de Programação de Aplicativos

- APIs são mecanismos que permitem a integração de componentes de software utilizando um conjunto de regras ou protocolos.

Padrão REST - Transferência Representacional de Estado:

- As solicitações do cliente são semelhantes a URLs e a resposta do servidor são dados simples, sem a renderização - geralmente em formato JSON ou XML.

Requisições HTTP

Rest HTTP Verbos

- ✓ **Hypertext Transfer Protocol (HTTP)** é um protocolo para transmissão de documentos hipermídia, como o HTML.
- ✓ Segue o modelo **cliente-servidor** - onde um **cliente executa uma requisição** e espera até receber uma **resposta do servidor**.
- ✓ O HTTP define um conjunto de métodos de requisição responsáveis por indicar a ação a ser executada para um dado recurso são comumente referenciados como **HTTP Verbs (Verbos HTTP)**.

Referencias:

- <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods>
- <https://www.w3schools.in/restful-web-services/intro/>
- <https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>

Resposta

Status Code

100-199 | Respostas informativas

200-299 | Respostas de sucesso

- ✓ **201** - A requisição foi bem sucedida e um novo recurso foi criado como resultado. Esta é uma típica resposta enviada após uma requisição POST.

300-399 | Mensagens de redirecionamento

400-499 | Respostas de erro do Cliente

- ✓ **404** - O servidor não pode encontrar o recurso solicitado. Este código de resposta talvez seja o mais famoso devido à frequência com que acontece na web.

500-599 | Respostas de erro do Servidor

<https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status>

Requisições HTTP

Verbo	Objetivo	Uso	Múlt. Req.	Cache
GET	O método GET solicita a representação de um recurso específico. Requisições utilizando o método GET devem retornar apenas dados.	Links	Yes	Yes
POST	O método POST é utilizado para submeter uma entidade a um recurso específico, frequentemente causando uma mudança no estado do recurso ou efeitos colaterais no servidor.	Forms	No	No
PUT	O método PUT substitui todas as atuais representações do recurso de destino pela carga de dados da requisição.	Forms	Yes	No
PATCH	O método PATCH é utilizado para aplicar modificações parciais em um recurso.	Forms	No/Yes	No
DELETE	O método DELETE remove um recurso específico.	Forms/Links	Yes	No

Padrões de arquitetura API

RESTful

- ✓ **Cliente/Servidor** – A mesma aplicação não pode ser cliente e servidor ao mesmo tempo.
- ✓ **Stateless** – Não guarda estado(nenhum cabeçalho) da aplicação, só recebe e responde solicitações;
- ✓ **“Cacheável”** – Capaz de guardar cache se necessário;
- ✓ **Ignorar Camadas** – Não importa a infraestrutura entre a aplicação cliente e servidor.
- ✓ **Interface uniforme e direta** – Usar o verbo **HTTP** correto para cada operação, sem a necessidade de especificar ação na url(rotas/end points);

Uma API pode utilizar apenas partes do padrão da arquitetura. Neste caso é considerada uma API **REST**. Só é considerada **RESTful** quando utiliza todos os níveis de implementação.

<https://pt.wikipedia.org/wiki/REST>





AJAX - Consumo de API

XMLHttpRequest | Fetch API | Axios

```
<script>
  var btnPesquisar = document.querySelector('[btn-submit]')
Complexity is 4 Everything is cool!
  btnPesquisar.onclick = function(event){ █
    event.preventDefault()
    const form = document.forms[0]
    let cep = form.cep.value
    let url = `https://viacep.com.br/ws/${cep}/json/`
    ajax = new XMLHttpRequest()
    ajax.open("GET", url, true)
    ajax.send(null)
Complexity is 3 Everything is cool!
    ajax.onreadystatechange = ()=>{ █
      if (ajax.readyState==4){
        if(ajax.status==200){
          console.log(ajax.responseText) //Chega como um JSON
          let resposta =JSON.parse(ajax.responseText)
          form.endereco.value = resposta.logradouro
          form.bairro.value = resposta.bairro
          form.cidade.value = resposta.localidade
          form.uf.value = resposta.uf
        }
      }
    }
  }
</script>
```

XMLHttpRequest

```
<script>
  var btnPesquisar = document.querySelector('[btn-submit]')
  Complexity is 4 Everything is cool!
  btnPesquisar.onclick = function(event){ █
    event.preventDefault()
    const form = document.forms[0]
    let cep = form.cep.value
    let url = `https://viacep.com.br/ws/${cep}/json/`
    fetch(url)
      .then(resposta=>resposta.json())
      .then(resposta=>{
        console.log(resposta)
        form.endereco.value = resposta.logradouro
        form.bairro.value = resposta.bairro
        form.cidade.value = resposta.localidade
        form.uf.value = resposta.uf
      })
      .catch(erro => {
        console.log(erro)
      })
  }
</script>
```

Fetch API

Axios

```
<script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>
<script>
  var btnPesquisar = document.querySelector('[btn-submit]')
  btnPesquisar.onclick = async (event)=>{
    event.preventDefault()
    const form = document.forms[0]
    let cep = form.cep.value
    let url = `https://viacep.com.br/ws/${cep}/json/`
    const dados = await axios.get(url)
    console.log(dados)
    let resposta = dados.data
    form.endereco.value = resposta.logradouro
    form.bairro.value = resposta.bairro
    form.cidade.value = resposta.localidade
    form.uf.value = resposta.uf
  }
</script>
```



POO – Programação Orientação a Objetos

Paradigmas de programação:

Paradigma de programação é um meio de se classificar as linguagens de programação baseado em suas funcionalidades. As linguagens podem ser classificadas em vários paradigmas. O paradigma de programação fornece a visão que o programador possui sobre a estruturação e execução do programa.

Mais informações: https://pt.wikipedia.org/wiki/Paradigma_de_programação

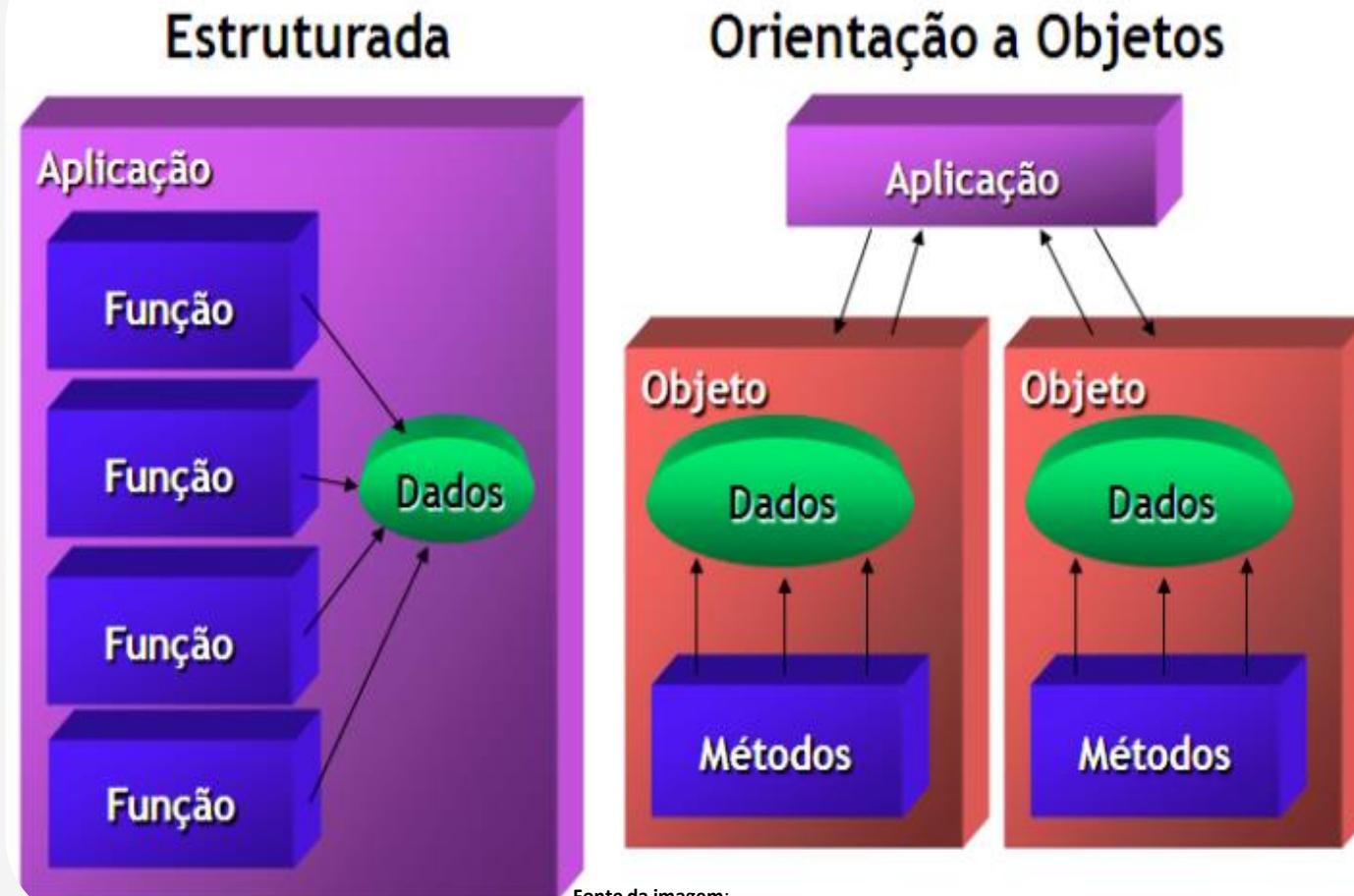
Programação Estruturada/Procedural/Imperativa

- Funcional;
- Lógico;
- Declarativa;
- Orientado a eventos;

Programação Orientada a Objetos (POO)

- Abstração;
- Herança/Polimorfismo;
- Encapsulamento;
- Concorrente;
- Reativa;

Estruturada vs. Orientação a Objetos



Programação Estruturada/Procedural

Chamada de procedimentos (ou funções) para manipulação de dados e variáveis;

Programação Orientada a Objetos - POO

Estruturas de CLASSES com comportamentos;

Estruturada vs. Orientação a Objetos

Programação Estruturada	Programação Orientada a Objetos – POO
Procedimentos e funções	Métodos
Variáveis	Instâncias de atributos
Chamadas a procedimentos e funções	Método construtor e métodos de acesso;
Tipos de dados definidos de forma global	Atributos das Classes
-----	Herança
-----	Polimorfismo

Pilares - POO

Princípio da Abstração:

O conceito de abstração consiste em esconder os detalhes de algo, no caso, os detalhes desnecessários. Na vida, utilizamos abstrações o tempo todo. Em tudo que não sabemos como funciona nos bastidores, pode ser considerado uma abstração.

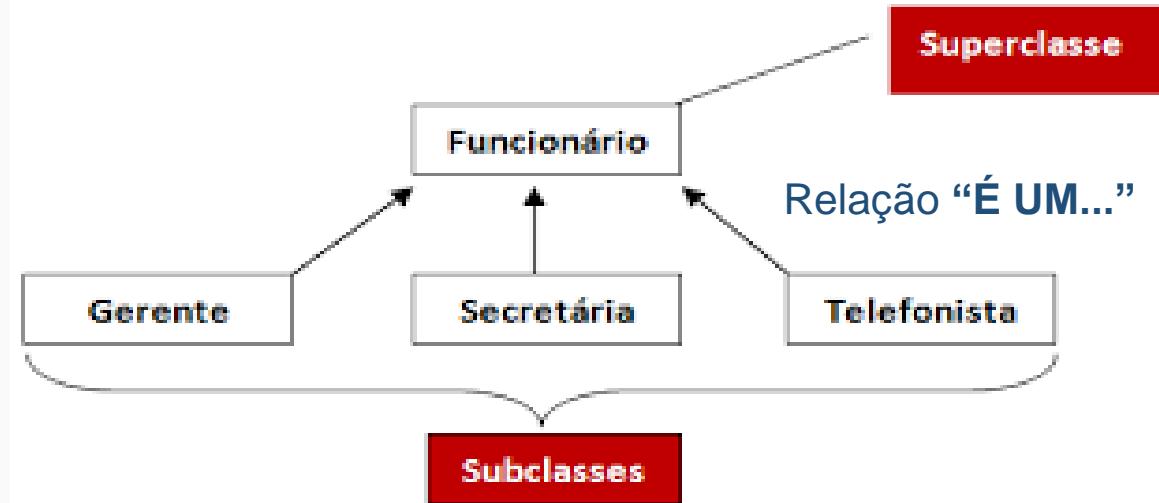
- Encapsulamento;
- Herança;
- Composição;
- Polimorfismo;



<https://materialpublic.imd.ufrn.br/curso/disciplina/2/8/1/4>

Herança

A herança permite que você crie uma nova classe que reutiliza, estende e modifica o comportamento definido em outras classes. A classe cujos membros são herdados é chamada **classe base** e a classe que herda esses membros é chamada **classe derivada**.



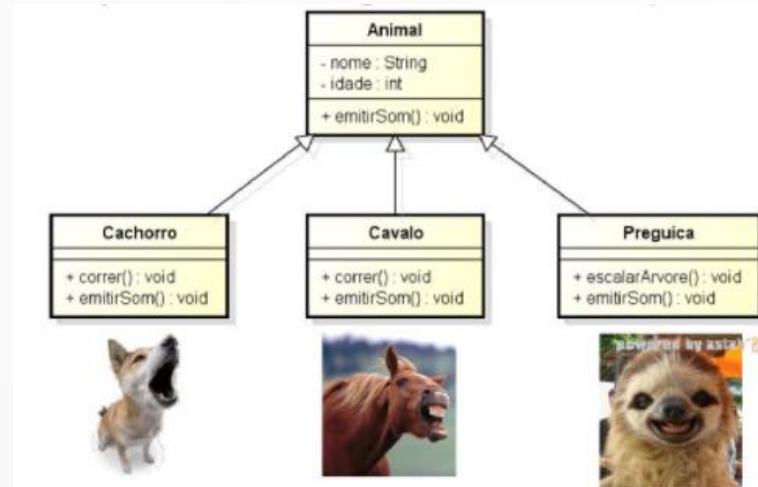
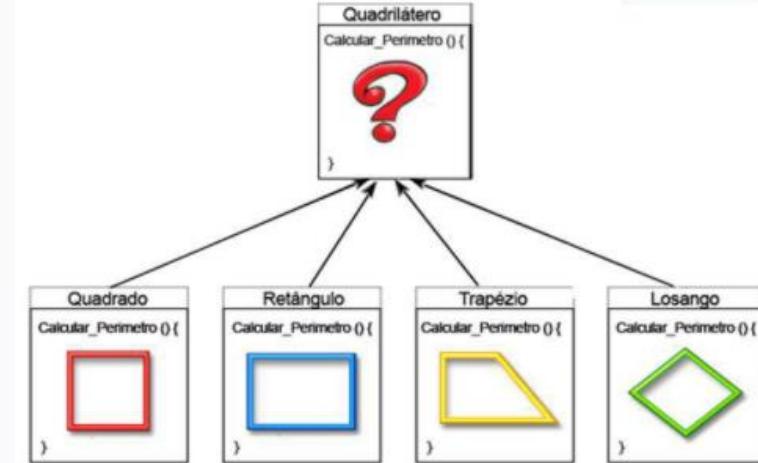
Para herdar de uma classe base:

C#

```
class DerivedClass:BaseClass {}
```

Polimorfismo

Polimorfismo é o princípio pelo qual duas ou mais classes derivadas da mesma superclasse podem invocar métodos que têm a mesma assinatura, mas comportamentos distintos.



Composição

Em linguagens de programação, objetos compostos são normalmente expressos por meio de **referências de um objeto para outro**.

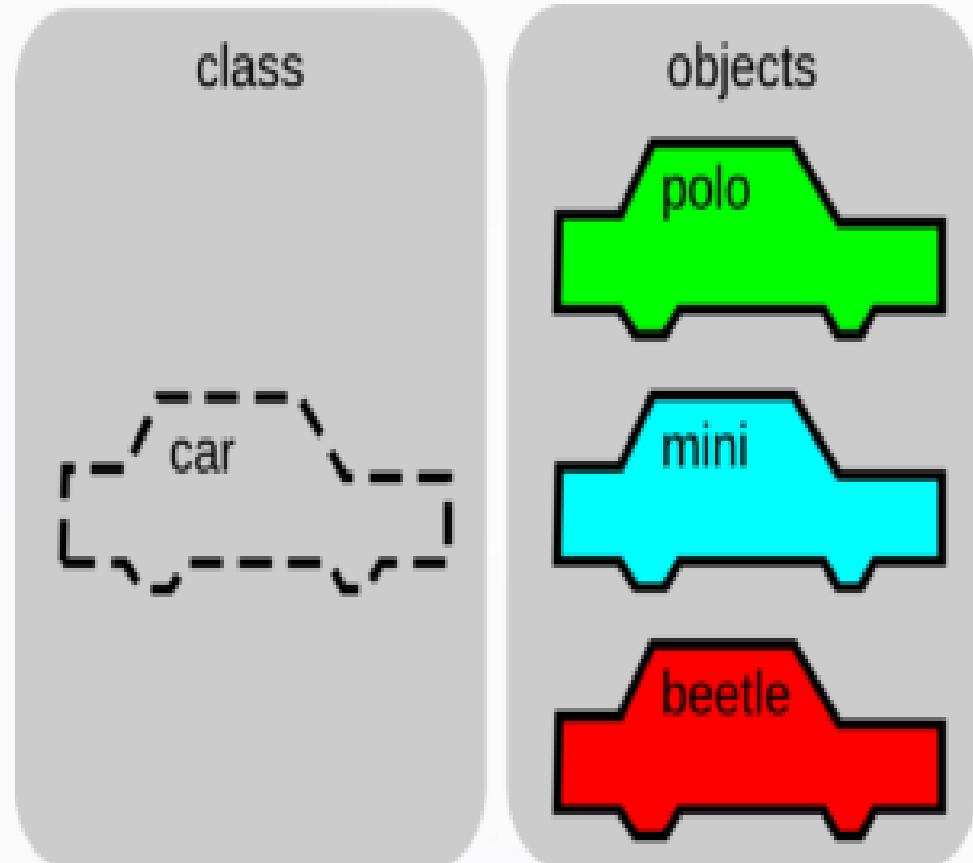
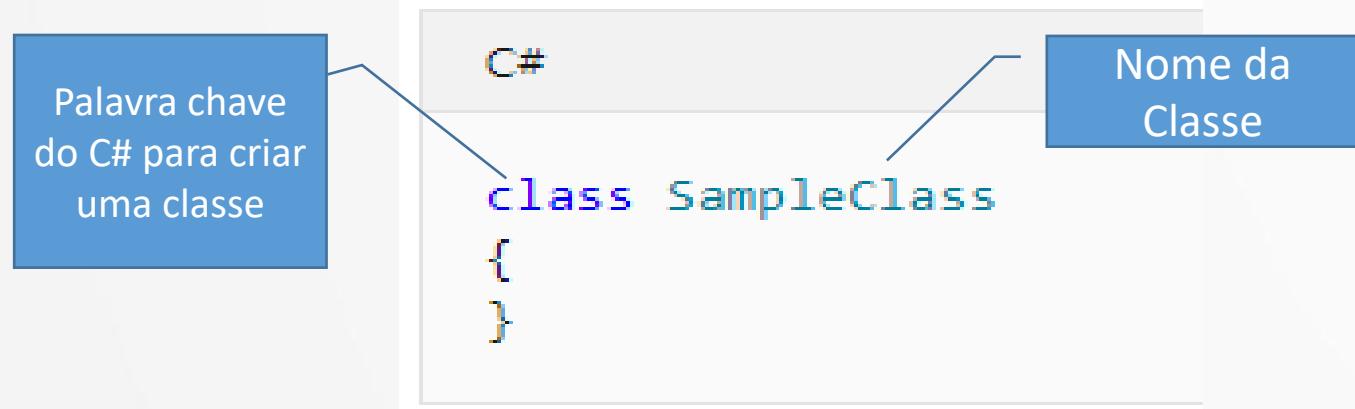
São atribuídos nomes exclusivos aos campos para que cada um possa ser distinguido dos outros. Um elemento só é definido como composto, se os objetos a que ele se refere fazem parte dele de fato, isto é, não existem isoladamente. Para detalhes, veja a seção agregação abaixo.

Composições são blocos de construção críticos de muitas estruturas de dados básicas, incluindo a união rotulada, lista ligada e árvore binária, bem como o objeto utilizado em programação orientada a objetos. Objetos compostos são frequentemente referidos como tendo um relacionamento "**tem um...**".



Classes e Objetos

A *classe* de termos e o *objeto* descrevem o *tipo* de objetos e as *instâncias* de classes, respectivamente. Sendo assim, o ato de criar um objeto é chamado de *instanciação*.



Método Construtor

Um construtor é um método cujo nome é igual ao nome de seu tipo. Sua assinatura do método inclui apenas o nome do método e lista de parâmetros, ele não inclui um tipo de retorno.

```
public class Person
{
    private string last;
    private string first;

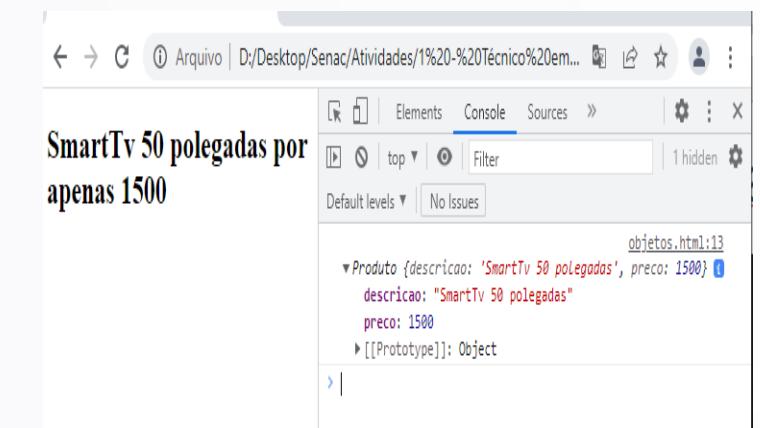
    public Person(string lastName, string firstName)
    {
        last = lastName;
        first = firstName;
    }

    // Remaining implementation of Person class.
}
```

Mão na massa... Instanciando uma classe com JavaScript

```
//Sintaxe para a definição de classe no JavaScript
class Produto {
    constructor(descricao, preco){
        this.descricao = descricao
        this.preco = preco
    }
    verProduto(){
        document.write(`<h2>${this.descricao} por apenas ${this.preco}</h2>`)
    }
}
```

```
//Criando uma estancia com o método construtor
var produto = new Produto('SmartTv 50 polegadas', 1500);
console.log(produto);
//Chamando um método do objeto
produto.verProduto();
```



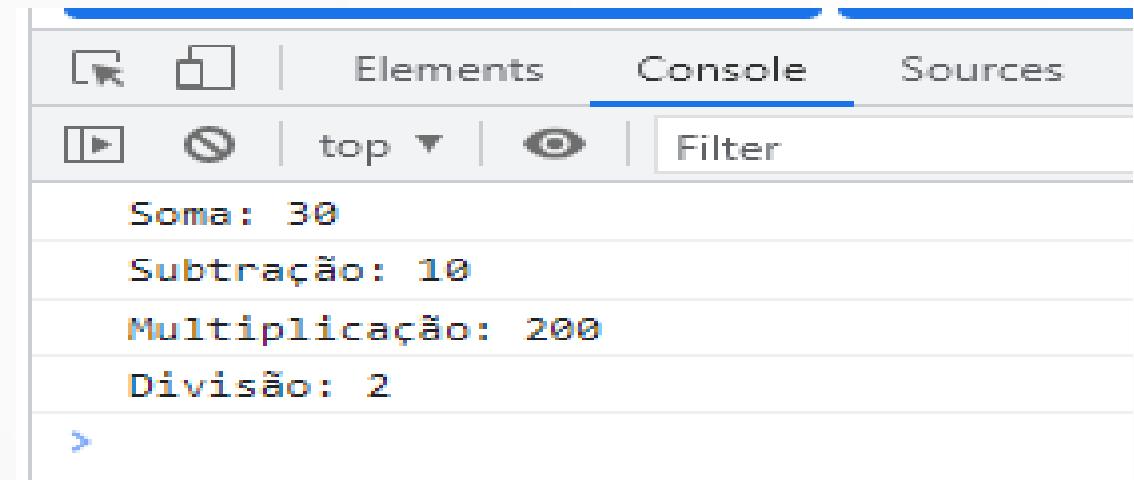
Desafio Calculadora:

Fazer uma calculadora que receba **dois valores** e retorne o resultado de **adição, subtração, multiplicação ou divisão** destes dois valores;

Resposta do desafio

```
class Calculadora{  
    constructor(valor1, valor2){  
        this.valor1 = valor1  
        this.valor2 = valor2  
    }  
    somarValores(){  
        return parseInt(this.valor1) + parseInt(this.valor2)  
    }  
    subtrairValores(){  
        return parseInt(this.valor1) - parseInt(this.valor2)  
    }  
    multiplicarValores(){  
        return parseInt(this.valor1) * parseInt(this.valor2)  
    }  
    dividirValores(){  
        return parseInt(this.valor1) / parseInt(this.valor2)  
    }  
}
```

```
var valor1=20;  
var valor2 =10  
var calculadora = new Calculadora(valor1, valor2)  
console.log('Soma: '+calculadora.somarValores())  
console.log('Subtração: '+calculadora.subtrairValores())  
console.log('Multiplicação: '+ calculadora.multiplicarValores())  
console.log('Divisão: '+calculadora.dividirValores())
```



Mão na massa... Encapsulamento com o JavaScript

```
class AssinaturaTV{  
    constructor (){  
        this.canaisDisponiveis = Array(2, 5, 7, 15, 30, 50)  
        this._canalAtivo = 7  
        this.volume = 10  
    }  
    get canalAtivo(){  
        return this._canalAtivo  
    }  
    set canalAtivo(canal){  
        if (this.canaisDisponiveis.indexOf(canal)>0){  
            this._canalAtivo = canal  
            console.log(`O canal ativo agora é: ${this._canalAtivo}`)  
        } else {  
            console.log(`O Canal ${canal} não está disponível`)  
        }  
    }  
}
```

```
var tv = new AssinaturaTV()  
console.log(tv)  
console.log(tv.canaisDisponiveis)  
console.log(tv.canalAtivo)  
tv.canalAtivo = 50  
tv.canalAtivo = 25
```

```
AssinaturaTV {canaisDisponiveis: Array(6), _canalAtivo: 7, volume: 10} objetos.html:14  
  ▶ canaisDisponiveis: (6) [2, 5, 7, 15, 30, 50]  
    volume: 10  
    _canalAtivo: 50  
    canalAtivo: (...)  
    ▶ [[Prototype]]: Object  
  ▶ (6) [2, 5, 7, 15, 30, 50] objetos.html:15  
  7 objetos.html:16  
  O canal ativo agora é: 50 classes.js:24  
  O Canal 25 não está disponível classes.js:26
```

Desafio Produto:

Construir uma classe Produto que com atributos **descrição**, **precoCusto** e **precoVenda**. O atributo **precoVenda** deve ser protegido e ter um método GET para acesso e SET alterando o valor apenas se o novo valor for maior que o **precoCusto**;

Resposta do desafio

```
class Produto {  
    constructor(descricao, precoCusto, precoVenda){  
        this.descricao = descricao  
        this.precoCusto = precoCusto  
        this._precoVenda = precoVenda  
    }  
    get precoVenda(){  
        return this._precoVenda;  
    }  
    set precoVenda(valor){  
        if (valor >= (this.precoCusto)){  
            this._precoVenda = valor  
            console.log(`O preço de venda foi alterado para ${this._precoVenda}`)  
        } else {  
            console.log('O valor é muito baixo, não pode ser alterado');  
        }  
    }  
}
```

```
var smartTv = new Produto('SmartTV 50 pol', 1500, 3600.50)  
smartTv.precoVenda = 1000  
console.log(smartTv.precoVenda)  
smartTv.precoVenda = 2200.60  
console.log(smartTv.precoVenda)
```

The screenshot shows a browser's developer tools console interface. At the top, there are buttons for play/pause, stop, and filter, followed by a dropdown menu set to 'top'. Below the toolbar, the console displays two lines of code and their corresponding outputs:

- The first line of code is: `var smartTv = new Produto('SmartTV 50 pol', 1500, 3600.50)`. Its output is: `O valor é muito baixo, não pode ser alterado` and `3600.5`.
- The second line of code is: `smartTv.precoVenda = 1000`. Its output is: `O preço de venda foi alterado para 2200.6` and `2200.6`.

Mão na massa... Herança com o JavaScript

```
class Animal {  
  constructor() {  
    this.cor = ''  
    this.tamanho = null  
    this.peso = null  
  }  
  
  dormir() {  
    console.log('Dormir')  
  }  
}
```

```
class Papagaio extends Passaro {  
  constructor() {  
    super()  
    this.sabeFalar = true  
  }  
  
  falar() {  
    console.log('Falar')  
  }  
}
```

```
class Passaro extends Animal {  
  constructor() {  
    super()  
    this.bico = 'Curto'  
  }  
  
  voar() {  
    console.log('Voar')  
  }  
}
```

```
class Cachorro extends Animal {  
  constructor(){  
    super()  
    this.pelo = true  
  }  
  
  correr() {  
    console.log('Correr')  
  }  
  
  latir() {  
    console.log('Au!Au!')  
  }  
}
```

```
let cachorro = new Cachorro()  
let passaro = new Passaro()  
let papagaio = new Papagaio()
```

```
console.log(cachorro)  
console.log(passaro)  
console.log(papagaio)
```

```
cachorro.dormir()  
passaro.dormir()  
papagaio.falar()  
papagaio.dormir()  
papagaio.voar()
```

Herança utilizando outros métodos

Atributo Prototype e
Object.create(obj)

```
// Cadeia de protótipos (prototype chain)
Object.prototype.attr0 = 'Z'
const avo = { attr1: 'A' }
const pai = { __proto__: avo, attr2: 'B' }
const filho = { __proto__: pai, attr3: 'C' }
console.log(filho.attr0)

1 const pai = { nome: 'Pedro', corCabelo: 'preto' }
2
3 const filha1 = Object.create(pai)
4 filha1.nome = 'Ana'
5 console.log(filha1.corCabelo)
6
7 const filha2 = Object.create(pai, {
8   nome: { value: 'Bia', writable: false, enumerable: true }
9 })
10
11 console.log(filha2.nome)
12 filha2.nome = 'Carla'
13 console.log(`${filha2.nome} tem cabelo ${filha2.corCabelo}`)
```

Exemplo de Composição/Agregação

```
//Exemplo de composição por Agregação
var contasMarco22 = new CicloMensal('Março','2022')
contasMarco22.addLancamento('Salario',6000)
contasMarco22.addLancamento('Cartão Alimentação',500)
contasMarco22.addLancamento('Trabalho Extra',1000)
contasMarco22.addLancamento('Supermercado',-2000)
contasMarco22.addLancamento('Cartão de Créditos',-800)
contasMarco22.addLancamento('Farmacia',450)
console.log(contasMarco22.lancamentos)
console.log('O valor total de lançamentos é:' + contasMarco22.calcularTotal())
```

```
▼ (6) [Lancamento, Lancamento, Lancamento, Lancamento, Lancamento, Lancamento] ⓘ
  ► 0: Lancamento {nome: 'Salario', valor: 6000}
  ► 1: Lancamento {nome: 'Cartão Alimentação', valor: 500}
  ► 2: Lancamento {nome: 'Trabalho Extra', valor: 1000}
  ► 3: Lancamento {nome: 'Supermercado', valor: -2000}
  ► 4: Lancamento {nome: 'Cartão de Créditos', valor: -800}
  ► 5: Lancamento {nome: 'Farmacia', valor: 450}
  length: 6
  ► [[Prototype]]: Array(0)
O valor total de lançamentos é:5150
```

Desafio CarrinhoCompras:

Construir uma classe **CarrinhoCompras** que tenha como **atributos**: **data da compra**, um **objeto Cliente** composto por **nome e endereço** e o atributo **produtos** onde serão agregado **N objetos da classe protuto**. A classe produto deve ter os atributos descrição e preco. A classe Carrinho de compras deve ter um método para somar os preços de todos os produtos agregados e exibir o total.

```
Complexity is 3 Everything is cool!
class CicloMensal{ constructor(mes, ano){ this.mes = mes; this.ano = ano; this.lancamentos = [] } addLancamento(nome, valor){ let lancamento = new Lancamento(nome, valor); this.lancamentos.push(lancamento) } //Lógica diferente para criar elemento agregado fora do método addLancamento2(...lancamentos){ lancamentos.forEach(l=>this.lancamentos.push(l)) } //Usar um ou outro. O primeiro tem vantagens Complexity is 3 Everything is cool!
calcularTotal(){ let valorTotal=0; this.lancamentos.forEach(l =>{ valorTotal+=l.valor }) return valorTotal } class Lancamento{ constructor(nome, valor){ this.nome = nome; this.valor = valor } }
```

Resposta do desafio

```
Complexity is 3 Everything is cool!
class CarrinhoCompras{ 
    constructor(dataCompra, nomeCliente, enderecoEntrega){
        this.dataCompra = dataCompra
        this.cliente= new Cliente(nomeCliente, enderecoEntrega)
        this.produtos = []
    }
    addProduto(descricao, preco){
        let produto = new Produto(descricao, preco)
        this.produtos.push(produto)
    }
}
Complexity is 3 Everything is cool!
calcularTotal(){
    let valorTotal=0
    this.produtos.forEach(p =>{
        console.log(p.descricao+' | '+p.preco)
        valorTotal+=p.preco
    })
    return valorTotal
}
```

```
class Cliente{
    constructor(nomeCliente, enderecoEntrega){
        this.nomeCliente=nomeCliente
        this.enderecoEntrega=enderecoEntrega
    }
}

class Produto{
    constructor(descricao, preco){
        this.descricao=descricao
        this.preco=preco
    }
}

var data='15/03/25';
var nome = "Joseph Climber"
var endereco='Rua de Traz, 700';
var carrinho = new CarrinhoCompras(data, nome, endereco)
console.log(carrinho)
carrinho.addProduto('SmartPhone', 6000)
carrinho.addProduto('XBox', 4000)
carrinho.addProduto('Bicicleta', 5000)
//Chamada do total
console.log(carrinho.calcularTotal())
```

```
poo.html:14
CarrinhoCompras {dataCompra: '15/03/25', cliente: Cliente, produtos: Array(0)} ⓘ
  cliente: Cliente
    enderecoEntrega: "Rua de Traz, n.700 Never Land"
    nomeCliente: "Joseph Climber"
  ► [[Prototype]]: Object
  dataCompra: "15/03/2025"
  ► produtos: Array(3)
    ► 0: Produto {descricao: 'SmartPhone', preco: 6000}
    ► 1: Produto {descricao: 'XBox', preco: 4000}
    ► 2: Produto {descricao: 'Bicicleta', preco: 5000}
    length: 3
    ► [[Prototype]]: Array(0)
  ► [[Prototype]]: Object
SmartPhone|6000
XBox|4000
Bicicleta|5000
15000
poo.html:19
```