

# **GEX613 – Programação II**

**JavaScript**



1100/1101 – CIÊNCIA DA COMPUTAÇÃO

Giancarlo Salton & Edimar Junior

- **JavaScript®** (às vezes abreviado para **JS**) é uma linguagem leve, interpretada e baseada em objetos.
- O JavaScript é uma linguagem baseada em protótipos, multi-paradigma e dinâmica, suportando estilos de orientação à objetos, imperativos e declarativos.
- O padrão JavaScript é ECMAScript

Em 17 de Junho de 2015, a ECMA Internacional publicou a sexta versão do ECMAScript, que é oficialmente chamado de ECMAScript 2015, e foi inicialmente conhecido como ECMAScript 6 ou ES6.
- Não se deve confundir o JavaScript com a linguagem de programação Java.

Tanto *Java* quanto *JavaScript* são marcas registradas da Oracle nos Estados Unidos da América e em outros países.

Console

Variáveis

Operadores

Condicionais

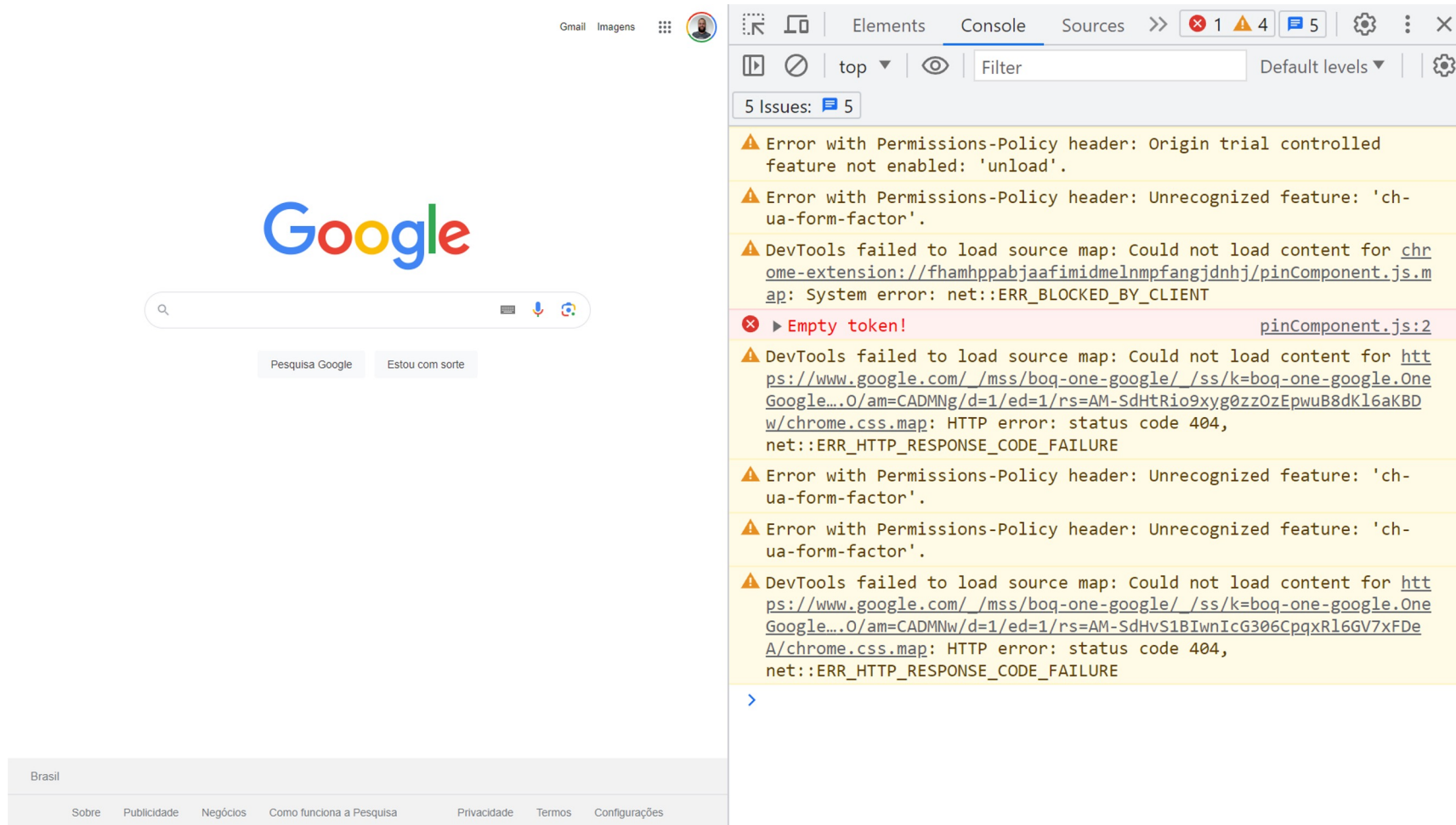
Laços

Funções

Extras

# Console

## Console



- O console é uma ferramenta para exibir mensagens e depurar código.
- Ele é parte integrante do ambiente de desenvolvimento do navegador e fornece várias funções que permitem imprimir informações no console do navegador.

## Principais funções do objeto console:

```
console.log("X é: " + x); //exibir mensagens ou valores no console
```

```
console.error("Algo de errado!"); //exibir mensagens de erro no console
```

```
console.warn("Cuidado!"); //exibir mensagens de aviso no console
```

```
console.info("Algo de errado!"); //exibir informações relevante no console,  
semelhante a log
```

```
console.clear("Algo de errado!"); // limpa a área de visualização do console
```

# Variáveis



```
let variavel = 25; // variável “normal”  
variavel = 'vinte e cinco'; // OK
```

```
const constante = 12; // variável imutável  
constante = 'doze';
```

```
✖ ▶ Uncaught TypeError: Assignment to constant  
   variable.  
     at <anonymous>:1:11 VM158:1
```

```
var varAntiga = 0; // estilo de declaração legado - evite a utilização  
// var se comporta como o 'let'
```

Regras de nomenclatura de variáveis seguem o padrão da maioria das linguagens de programação.

```
let variavel = 25; // variável "normal"  
variavel = 'vinte e cinco'; // OK
```

```
const constante = 12; // variável imutável  
constante = 'doze';
```

```
✖ ▶ Uncaught TypeError: Assignment to constant  
   variable.  
     at <anonymous>:1:11 VM158:1
```

```
var varAntiga = 0; // estilo de declaração legado - evite a utilização  
// var se comporta como o 'let'
```

Não pode ter espaços,  
hifens e nem iniciar com  
números.

```
let variavel = 25; // variável "normal"  
variavel = 'vinte e cinco'; // OK
```

```
const constante = 12; // variável imutável  
constante = 'doze';
```

```
✖ ▶ Uncaught TypeError: Assignment to constant  
   variable.  
     at <anonymous>:1:11 VM158:1
```

```
var varAntiga = 0; // estilo de declaração legado - evite a utilização  
// var se comporta como o 'let'
```

## Variáveis

Variáveis definidas com a keyword `let` são variáveis “mutáveis”.

```
let variavel = 25; // variável “normal”  
variavel = 'vinte e cinco'; // OK
```

```
const constante = 12; // variável imutável  
constante = 'doze';
```

```
✖ ▶ Uncaught TypeError: Assignment to constant  
   variable.  
     at <anonymous>:1:11 VM158:1
```

```
var varAntiga = 0; // estilo de declaração legado - evite a utilização  
// var se comporta como o 'let'
```

```
let variavel = 25; // variável "normal"  
variavel = 'vinte e cinco'; // OK
```

Variáveis definidas com a keyword `const` são variáveis "imutáveis". Em outras palavras, são constantes.

```
const constante = 12; // variável imutável  
constante = 'doze';
```

```
✖ ▶ Uncaught TypeError: Assignment to constant  
   variable.  
     at <anonymous>:1:11 VM158:1
```

```
var varAntiga = 0; // estilo de declaração legado - evite a utilização  
// var se comporta como o 'let'
```

```
let variavel = 25; // variável "normal"  
variavel = 'vinte e cinco'; // OK
```

```
const constante = 12; // variável imutável  
constante = 'doze';
```

```
✖ ▶ Uncaught TypeError: Assignment to constant  
   variable.  
     at <anonymous>:1:11 VM158:1
```

Variáveis definidas com a *keyword* `var` são semelhantes às definidas `let`. Esta definição foi mantida por compatibilidade e não deve ser usada.

```
var varAntiga = 0; // estilo de declaração legado - evite a utilização  
// var se comporta como o 'let'
```

## Tipos nativos

- Boolean: `true` e `false`
- Null: `null`
- Undefined: variável declarada mas que ainda não recebeu valor (e.g., `let und;`)
- Number: engloba inteiros e ponto flutuante
  - inteiros:  $-(2^{53} - 1)$  (Number.MIN\_SAFE\_INTEGER) para  $2^{53} - 1$  (Number.MAX\_SAFE\_INTEGER)
  - ponto flutuante:
    - positivos:  $2^{-1074}$  (Number.MIN\_VALUE) e  $2^{1024}$  (Number.MAX\_VALUE)
    - negativos:  $-(2^{-1074})$  e  $-(2^{1024})$
- BigInt
- String
- Symbol

# Operadores



```
let x = 10; //atribuição (recebe)  
x + 7; //soma  
x - 5; //subtração  
x * 3; //multiplicação  
x / 2; //divisão  
x % 2; //módulo  
x++; //incremento  
x--; //decremento
```

```
let x = 10;  
let y = 5;  
x == y; //igual  
x != y; //diferente  
x === y; //estritamente igual (confere valor e tipo)  
x !== y; //estritamente diferente (confere valor e tipo)  
x > y; //maior que  
x < y; //menor que  
x >= y; //maior ou igual  
x <= y; //menor ou igual
```

```
let x = 10;  
let y = 5;  
x && y; //and  
x || y; //or  
!x ; //not
```

# Condicionais

```
let x = 11;
if (x > 5){
    console.log("X é maior ou igual do que 5");
} else{
    console.log("X é menor do que 5");
}
// saída esperada: X é maior ou igual do que 5
```

Condicionais do tipo `if/else` funcionam de forma semelhante à outras linguagens de programação.

```
let x = 11;  
if (x > 5){  
    console.log("X é maior ou igual do que 5");  
} else{  
    console.log("X é menor do que 5");  
}  
  
// saída esperada: X é maior ou igual do que 5
```

## if / else

Note que os blocos são delimitados por { e }. As regras de escopo de visibilidade das variáveis é semelhante à outras linguagens de programação como o C e o Java.

```
let x = 11;  
if (x > 5){  
    console.log("X é maior ou igual do que 5");  
} else{  
    console.log("X é menor do que 5");  
}  
// saída esperada: X é maior ou igual do que 5
```

## switch / case

```
const expr = "limão";
switch (expr) {
  case "laranja": {
    console.log("Laranjas estão R$ 3,00/kg.");
    break;
  }
  case "manga":
  case "limão": {
    console.log("Mangas e limões estão R$4,79/kg.");
    break;
  }
  default:
    console.log(`Desculpe, estamos sem ${expr}.`);
}
// saída esperada: "Mangas e limões estão $4,79/kg."
```



## switch / case

```
const expr = "limão";
switch (expr) {
  case "laranja": {
    console.log("Laranjas estão R$ 3,00/kg.");
    break;
  }
  case "manga":
  case "limão": {
    console.log("Mangas e limões estão $4,79/kg.");
    break;
  }
  default:
    console.log(`Desculpe, estamos sem ${expr}.`);
}

// saída esperada: "Mangas e limões estão $4,79/kg."
```

Há também o condicional `switch/case`.  
que também se comporta de forma  
semelhante à outras linguagens de  
programação como o C.

## switch / case

```
const expr = "limão";
switch (expr) {
  case "laranja": {
    console.log("Laranjas estão R$ 3,00/kg.");
    break;
  }
  case "manga":
  case "limão": {
    console.log("Mangas e limões estão $4,79/kg.");
    break;
  }
  default:
    console.log(`Desculpe, estamos sem ${expr}.`);
}

// saída esperada: "Mangas e limões estão $4,79/kg."
```

Note a utilização do comando `break` para que o condicional encerre antecipadamente caso o condicional seja aceito.

## switch / case

```
const expr = "limão";
switch (expr) {
  case "laranja": {
    console.log("Laranjas estão R$ 3,00/kg.");
    break;
  }
  case "manga":
  case "limão": {
    console.log("Mangas e limões estão $4,79/kg.");
    break;
  }
  default:
    console.log(`Desculpe, estamos sem ${expr}.`);
}

// saída esperada: "Mangas e limões estão $4,79/kg."
```

Caso dois ou mais condicionais ocorram de forma contígua, com apenas um bloco de execução, todos os condicionais compartilham este bloco.

## switch / case

```
const expr = "limão";
switch (expr) {
  case "laranja": {
    console.log("Laranjas estão R$ 3,00/kg.");
    break;
  }
  case "manga":
  case "limão": {
    console.log("Mangas e limões estão $4,79/kg.");
    break;
  }
  default:
    console.log(`Desculpe, estamos sem ${expr}.`);
}

// saída esperada: "Mangas e limões estão $4,79/kg."
```

O comando `switch/case` ainda permite um caso padrão que será executado se nenhum dos outros condicionais for acionado.

# Laços

## while

```
let i = 0
let text = '';
while (i < 5) {
    text += "0 número é " + i + "\n";
    i++;
}
console.log(text);
```

O comando `while` é a forma mais simples de se criar um laço de repetição. Note, mais uma vez, como a sintaxe é idêntica à outras linguagens como C e Java.

```
/* Saída esperada:
0 número é 0
0 número é 1
0 número é 2
0 número é 3
0 número é 4
*/
```

for

```
for (let i = 1; i < 5; i++) {  
    console.log(i);  
}
```

/\* Saída esperada:

1

2

3

4

\*/

O comando `for` possui a sintaxe  
é similar à outras linguagens  
como C e Java.

```
for (let i = 1; i < 5; i++) {  
    console.log(i);  
}
```

/\* Saída esperada:

1

2

3

4

\*/



for / in

```
let j = [1, 2, 3, 4, 5];  
for (let i in j) {  
    console.log(i);  
}
```

O comando **for** também pode ser usado com um vetor usando a keyword **in**.

```
/* Saída esperada:  
0  
1  
2  
3  
4  
*/
```

for / in

```
let j = [1, 2, 3, 4, 5];  
for (let i in j) {  
    console.log(i);  
}
```

Perceba que ao usar `in`, a variável `i` recebe o índice do vetor e não o valor. Para acessar o valor, utilizamos `j[i]`.

/\* Saída esperada:

0

1

2

3

4

\*/

```
let j = [1, 2, 3, 4, 5];  
for (let i of j) {  
    console.log(i);  
}
```

/\* Saída esperada:

```
1  
2  
3  
4  
5  
*/
```

## for / of

```
let j = [1, 2, 3, 4, 5];  
for (let i of j) {  
    console.log(i);  
}
```

O comando **for** também pode ser usado com um vetor usando a *keyword of*.

/\* Saída esperada:

1

2

3

4

5

\*/

```
let j = [1, 2, 3, 4, 5];  
for (let i of j) {  
    console.log(i);  
}
```

Perceba que ao usar `of`, a variável `i` recebe o valor que está em `j`.

/\* Saída esperada:

```
1  
2  
3  
4  
5  
*/
```

# Funções

usando a keyword 'function'

Funções podem ser definidas de forma similar à linguagem Python usando a *keyword* **function**.

```
function soma(a, b) {  
    return a + b;  
}  
console.log(soma(1, 7));  
// saída esperada: 8
```

```
function somaSemRetorno(a, b) {  
    console.log(a + b);  
}  
somaSemRetorno(3, 2);  
// saída esperada: 5
```

usando a keyword 'function'

Os argumentos são definidos entre ( e ), sem necessidade de definir o "tipo" da variável.

```
function soma(a, b) {  
    return a + b;  
}
```

```
console.log(soma(1, 7));  
// saída esperada: 8
```

```
function somaSemRetorno(a, b) {  
    console.log(a + b);  
}
```

```
somaSemRetorno(3, 2);  
// saída esperada: 5
```



usando a keyword 'function'

```
function soma(a, b) {  
    return a + b;  
}  
console.log(soma(1, 7));  
// saída esperada: 8
```

Se desejarmos retornar um valor das funções, utilizamos a keyword `return`.

```
function somaSemRetorno(a, b) {  
    console.log(a + b);  
}  
somaSemRetorno(3, 2);  
// saída esperada: 5
```

usando a keyword 'function'

```
function soma(a, b) {  
    return a + b;  
}  
console.log(soma(1, 7));  
// saída esperada: 8
```

```
function somaSemRetorno(a, b) {  
    console.log(a + b);  
}  
somaSemRetorno(3, 2);  
// saída esperada: 5
```

Funções não precisam necessariamente retornar valores. Nestes casos, o retorno é definido como `undefined`.

usando 'arrow functions'

```
const somaComArrowReturn = (a, b) => {  
    return a + b;  
}  
console.log(somaComArrowReturn(4, 3));  
// saída esperada: 7
```

```
const somaComArrowSemReturn = (a, b) => {  
    console.log(a + b);  
}  
somaComArrowSemReturn(3, 6);  
// saída esperada: 9
```

usando 'arrow functions'

Uma forma mais “moderna”  
de definir funções é usando a  
sintaxe conhecida como  
“*arrow functions*”.

```
const somaComArrowReturn = (a, b) => {  
    return a + b;  
}  
console.log(somaComArrowReturn(4, 3));  
// saída esperada: 7
```

```
const somaComArrowSemReturn = (a, b) => {  
    console.log(a + b);  
}  
somaComArrowSemReturn(3, 6);  
// saída esperada: 9
```

usando 'arrow functions'

```
const somaComArrowReturn = (a, b) => {  
    return a + b;  
}  
console.log(somaComArrowReturn(4, 3));  
// saída esperada: 7
```

```
const somaComArrowSemReturn = (a, b) => {  
    console.log(a + b);  
}  
somaComArrowSemReturn(3, 6);  
// saída esperada: 9
```

Arrow functions são  
definidas usando  
( ) => {};  
onde os parâmetros são  
definidos entre ( ) e o  
código definido entre {}.

usando 'arrow functions'

```
const somaComArrowReturn = (a, b) => {  
    return a + b;  
}
```

```
console.log(somaComArrowReturn(4, 3));
```

```
// saída esperada: 7
```

Sempre devemos atribuir uma *arrow function* à uma variável. Uma boa prática é sempre atribuir à uma constante.

```
const somaComArrowSemReturn = (a, b) => {  
    console.log(a + b);  
}
```

```
somaComArrowSemReturn(3, 6);
```

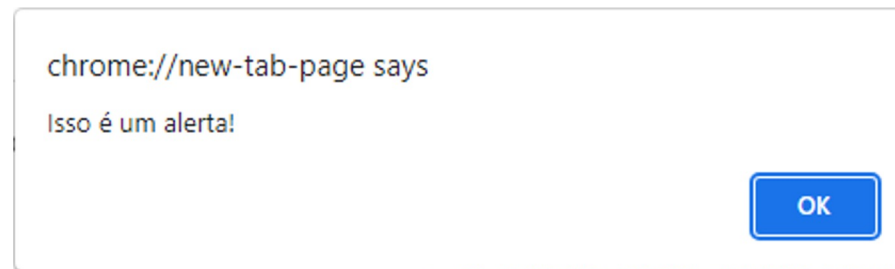
```
// saída esperada: 9
```

# Extras

- As janelas de diálogo em JavaScript são caixas de mensagem que permitem interagir com o usuário exibindo informações, solicitando entrada ou confirmando ações.
- Existem três tipos principais de janelas de diálogo.

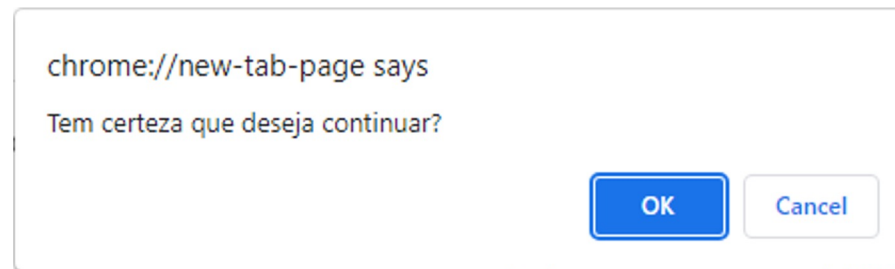


```
alert("Isso é um alerta!");
```

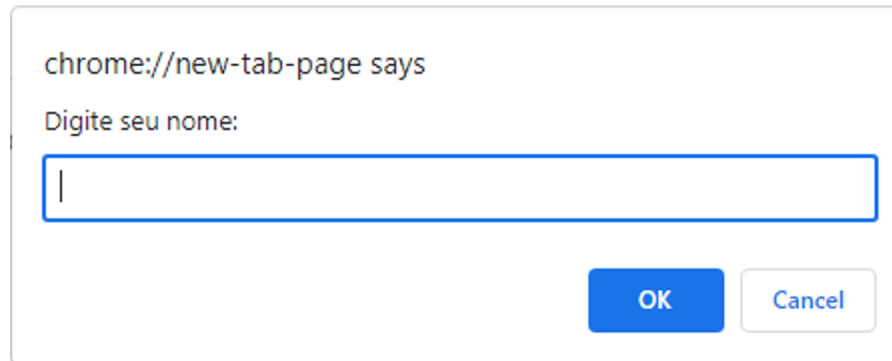


## Principais de janelas de diálogo

```
let opcao = confirm("Tem certeza que deseja continuar?");
```



```
let nome = prompt("Digite seu nome:");
```



A screenshot of a JavaScript prompt dialog box. The title bar reads "chrome://new-tab-page says". The main text inside the dialog is "Digite seu nome:". Below the text is a single-line text input field with a blue border and a vertical cursor. At the bottom right of the dialog are two buttons: a blue "OK" button and a light gray "Cancel" button.