

Programação Orientada a Objetos

...

M02 - Funções

MO2 - Funções

1. Declaração de funções
2. Nomeação de funções
3. Variáveis locais e globais
4. Parâmetros
5. Retorno de valor
6. Expressão de função
7. Funções arrow

MO2 - Funções

O que são funções?

- Muitas vezes precisamos realizar uma ação semelhante em muitos lugares da script
- Exemplo: mostrar uma mensagem para login, logout e para outros lugares
- Funções são os principais “blocos de construção” de um programa JavaScript
- Elas permitem que o código seja chamado muitas vezes sem repetição
- Já vimos exemplos de funções integradas: **alert**, **prompt** e **confirm**
- Mas também podemos criar funções próprias!

M02 - Funções

1. Declaração de funções

M02 - Funções

1. Declaração de função

- Para **criar uma função**, pode usar uma declaração de função



MO2 - Funções

1. Declaração de função

- Execução de função
 - Para **executar uma função**, use o nome da função com os parêntesis e parâmetros (caso hajam)
 - Este exemplo demonstra claramente um dos principais objetivos das funções: **evitar a duplicação de código**
 - Se precisarmos alterar a mensagem ou a maneira como ela é mostrada, basta modificar o código num lugar: a função que gera a mensagem!



M02 - Funções

2. Nomeação de funções

MO2 - Funções

2. Nomeação de funções

- Funções são ações. Então o nome delas é geralmente um verbo
- Deve ser breve, o mais preciso possível e descrever o que a função faz, de modo que alguém que esteja lendo o código receba uma indicação do que a função faz
- É uma prática generalizada iniciar uma função com um prefixo verbal que descreve vagamente a ação. Deve haver um acordo dentro da equipa sobre o significado dos prefixos
- Por exemplo, funções que começam com "show" geralmente mostram algo

MO2 - Funções

2. Nomeação de funções

- Funções começando com...
 - "get..." - retorna um valor
 - "calc..." - calcula algo
 - "create..." - cria algo
 - "check..." - verifica algo e retorna um booleano, etc.
- Exemplos:

```
showMessage(...) // mostra uma mensagem
getAge(...)      // retorna a idade
calcSum(...)     // calcula a soma e retorna o resultado
createForm(...)  // cria um formulário
checkPermission(...) // verifica uma permissão, devolvendo true ou false
```

MO2 - Funções

2. Nomeação de funções

- Uma função deve fazer exatamente o que é sugerido pelo seu nome, não mais
- Duas ações independentes geralmente merecem duas funções, mesmo que sejam geralmente chamadas juntas (nesse caso, podemos fazer uma terceira função que chama essas duas)
- As funções devem ser curtas e fazer exatamente uma coisa. Se isso for grande, talvez valha a pena dividir a função em algumas funções menores. Às vezes, seguir esta regra pode não ser tão fácil, mas definitivamente é uma boa estratégia:
 - uma função separada é mais fácil de testar e depurar
 - a sua própria existência é um ótimo comentário!

M02 - Funções

3. Variáveis locais e globais

M02 - Funções

3. Variáveis locais e globais

- Uma variável declarada dentro de uma função só é visível dentro dessa função
- Diz-se que é uma variável **local**

Variável local

```
function showMessage() {  
  let message = "Hello, I'm JavaScript!"; // variável local  
  alert(message);  
}
```

Erro: variável não existe
neste âmbito

```
showMessage(); // Hello, I'm JavaScript!  
alert(message); // <-- Erro! A variável é local à função
```

M02 - Funções

3. Variáveis locais e globais

- Uma função também pode acessar uma variável externa (**global**), por exemplo:

```
let userName = 'John';

function showMessage() {
  let message = 'Hello, ' + userName;
  alert(message);
}

showMessage(); // Hello, John
```

M02 - Funções

3. Variáveis locais e globais

- A função tem acesso total à variável externa podendo-a modificá-la

Alteração da
variável global

```
let userName = 'John';

function showMessage() {
  userName = "Bob"; // (1) alteração da variável externa
  let message = 'Hello, ' + userName;
  alert(message);
}

alert(userName); // John (antes da chamada da função)
showMessage();
alert(userName); // Bob (o valor foi modificado pela função)
```

- A variável externa é usada apenas se não houver uma local
- Então, uma modificação ocasional pode acontecer se não usarmos **let**

M02 - Funções

3. Variáveis locais e globais

- Se variável com o mesmo nome é declarada dentro da função, ela é usada em detrimento da externa

```
let userName = 'John';

function showMessage() {
  let userName = "Bob"; // declara uma variável local

  let message = 'Hello, ' + userName; // Bob
  alert(message);
}

showMessage(); // a função irá criar e usar a sua variável userName
alert(userName); // John (inalterado!, a função não acedeu à variável externa)
```

Acesso a
variável local

variável global
inalterada

M02 - Funções

3. Variáveis locais e globais

- Sumário
 - As variáveis globais
 - Declaradas fora de qualquer função
 - Visíveis de qualquer função (a menos que sejam sobrepostas por locais)
 - Armazenam apenas dados ao nível do projeto e acessíveis de qualquer lugar
 - Normalmente, uma função declara todas as variáveis específicas para a sua tarefa
 - O código moderno tem poucos ou nenhuma globais. A maioria das variáveis reside em funções

M02 - Funções

4. Parâmetros

M02 - Funções

4. Parâmetros

- Podemos passar dados arbitrários para funções
 - **Parâmetros** de função são os nomes listados na definição da função
 - **Argumentos** de função são os valores reais passados para (e recebidos pela) função

```
function showMessage(from, text) { // argumentos: from, text
  alert(from + ': ' + text);
}

showMessage('Ann', 'Hello!'); // Ann: Hello! (*)
showMessage('Ann', "What's up?"); // Ann: What's up? (**)
```

- Quando a função é chamada nas linhas (*) e (**), os valores dados são copiados para variáveis locais (**from** e **text**). A partir daí a função usa essas variáveis locais

MO2 - Funções

4. Parâmetros

- Veja mais um exemplo: temos a variável **from** e passamos para a função
- A função muda a variável **from**, mas a mudança não é vista de fora, porque a função recebe sempre uma cópia do valor:

```
function showMessage(from, text) {  
  from = '*' + from + '*'; // alteração da variável local: from  
  alert(from + ': ' + text);  
}  
  
let from = "Ann";  
  
showMessage(from, "Hello"); // *Ann*: Hello  
  
// o valor de "from" é o mesmo, a função modificou uma cópia local  
alert(from); // Ann
```

M02 - Funções

4. Parâmetros

- O objeto **arguments**
 - As funções JavaScript têm um objeto interno chamado de **arguments**
 - contém uma matriz dos argumentos usados quando a função foi chamada (invocada)
 - Exemplo:

```
let x = findMax(1, 123, 500, 115, 44, 88);

function findMax() {
  let i;
  let max = -Infinity;
  for (i = 0; i < arguments.length; i++) {
    if (arguments[i] > max) {
      max = arguments[i];
    }
  }
  return max;
}
```

M02 - Funções

4. Parâmetros

- Parâmetros **Rest**

- Uma função pode ser chamada com qualquer nº de argumentos, não importa como seja definida

```
function soma(a, b) {  
    return a + b;  
}  
  
alert(soma(1, 2, 3, 4, 5));
```

- Não há erro pelos argumentos "excessivos". Mas no resultado apenas os dois primeiros serão contados
 - Os parâmetros **Rest**
 - significam "reunir os parâmetros restantes num array"
 - podem ser mencionados numa definição de função com três pontos ...
 - devem ser sempre os últimos a serem mencionados

MO2 - Funções

4. Parâmetros

- Parâmetros Rest

- Por exemplo, para reunir todos (ou alguns) argumentos num array:

uso do rest para
todos os parâmetros

```
function sumAll(...args) { // args é o nome para o array
  let sum = 0;
  for (let arg of args) sum += arg;
  return sum;
}

alert(sumAll(1)); // 1
alert(sumAll(1, 2)); // 3
alert(sumAll(1, 2, 3)); // 6
```

uso do rest para
apenas alguns
parâmetros

```
function showName(firstName, lastName, ...titles) {
  alert(firstName + ' ' + lastName); // Julius Caesar

  // o resto vai para o array titles
  // i.e. titles = ["Consul", "Imperator"]
  alert(titles[0]); // Consul
  alert(titles[1]); // Imperator
  alert(titles.length); // 2
}

showName("Julius", "Caesar", "Consul", "Imperator");
```

M02 - Funções

4. Parâmetros

- Valores por omissão
 - Se um parâmetro não for fornecido, o seu valor ficará indefinido

```
function showMessage(from, text) {  
    alert(from + ': ' + text);  
}  
  
showMessage("John"); // John: undefined
```

Uma página incorporada em www.google.com diz

John: undefined

OK

M02 - Funções

4. Parâmetros

- Valores por omissão
 - Se quisermos usar um texto "padrão" neste caso, podemos especificá-lo depois de =

```
function showMessage(from, text = "nenhum texto fornecido") {  
    alert(from + ": " + text);  
}  
  
showMessage("John"); // John: nenhum texto fornecido
```

Uma página incorporada em www.google.com diz
John: nenhum texto fornecido

OK

- Pode ser uma expressão mais complexa, que só é avaliada e atribuída se o parâmetro estiver ausente

```
function showMessage(from, text = anotherFunction()) {  
    // anotherFunction() é executada apenas se não for fornecido um valor para text  
    // o resultado da função fica o valor da variável text  
}
```


M02 - Funções

5. Retorno de valor

M02 - Funções

5. Retorno de valor

- Uma função pode retornar um valor de volta ao código de chamada como resultado
- O exemplo mais simples seria uma função que soma dois valores:

```
function sum(a, b) {  
  return a + b;  
}  
  
let result = sum(1, 2);  
alert(result); // 3
```

- A diretiva **return** pode estar em qualquer lugar da função. Quando a execução alcança, a função para e o valor é retornado ao código de chamada (atribuído ao resultado acima).

M02 - Funções

5. Retorno de valor

- Podem haver várias ocorrências de retorno numa única função. Por exemplo:

```
function checkAge(age) {  
  if (age > 18) {  
    return true;  
  } else {  
    return confirm('tens permissões dos teus pais?');  
  }  
}  
  
let age = prompt('Quantos anos tens?', 18);  
  
if (checkAge(age)) {  
  alert('Access concedido');  
} else {  
  alert('Access negado');  
}
```

M02 - Funções

5. Retorno de valor

- É possível usar o retorno sem um valor
- Faz com que a função saia imediatamente

```
function showMovie(age) {  
  if (!checkAge(age)) {  
    return;  
  }  
  alert("Mostrando o filme...");  
}
```

- Uma função com um retorno vazio ou sem ele retorna **undefined**

M02 - Funções

6. Expressão de Função

MO2 - Funções

6. Expressão de Função

- Definição
 - É muito similar e tem quase a mesma sintaxe de uma declaração de função
 - A principal diferença entre ambas é o nome da função, o qual pode ser omitido em expressões de funções para criar funções anônimas

The diagram shows two code snippets side-by-side. The first snippet is a function declaration: `function showMessage() { alert("Olá a todos!"); }`. A yellow box labeled "Declaração de função" has an arrow pointing to the `function` keyword. The second snippet is a function expression: `let showMessage = function() { alert("Olá a todos!"); }`. A yellow box labeled "Expressão de função" has an arrow pointing to the `function` keyword within the assignment.

```
function showMessage() {  
    alert("Olá a todos!");  
}
```

Declaração de função

```
let showMessage = function() {  
    alert("Olá a todos!");  
}
```

Expressão de função

- A chamada da função é idêntica em ambas as abordagens

MO2 - Funções

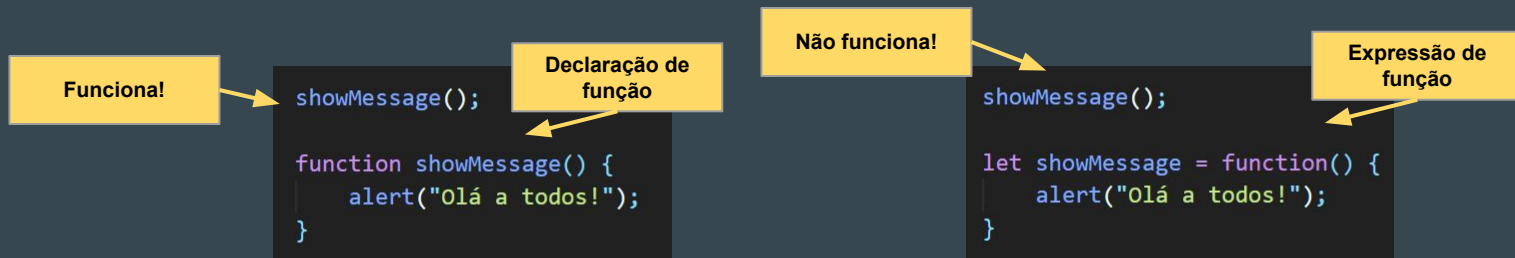
6. Expressão de Função

- Diferenças entre declarações e expressões de funções
 - Uma expressão de função é criada quando a execução chega e SÓ é utilizável a partir de então
 - Declarações de função são diferentes:
 - Uma declaração de função é utilizável em todo o script / bloco de código
 - Em outras palavras, quando o JavaScript se prepara para executar o script ou um bloco de código, ele primeiro procura declarações de função nele e cria as funções. Podemos pensar nisso como um “estágio de inicialização”
 - E depois de todas as declarações de função serem processadas, a execução continua
 - Como resultado, uma função declarada como uma declaração de função pode ser chamada antes do que é definido

M02 - Funções

6. Expressão de Função

- Diferenças entre declarações e expressões de funções



M02 - Funções

6. Expressão de Função

- Quando devo usar declarações e expressões de funções?
 - Considere a sintaxe de declaração de função
 - Dá mais liberdade em como organizar o nosso código, porque podemos chamar tais funções antes que elas sejam declaradas
 - Também é mais fácil procurar a função `f (...) {...}` no código do que `let f = function (...) {...}`

M02 - Funções

6. Expressão de Função

- Expressões de função imediatamente invocadas (IIFE)
 - Forma de criar e invocar imediatamente a função
 - Basta agregar a função dentro de parêntesis e invocá-la com novos parênteses

```
(function () {  
    let message = "Hello";  
    alert(message); // Hello  
})();
```

M02 - Funções

7. Funções arrow

M02 - Funções

7. Funções arrow

- Há uma sintaxe mais simples e concisa para criar expressões de funções
- Chamadas de "funções de seta" (funções arrow)
- Sintaxe:

`let func = (arg1, arg2, ...argN) => expression`

- Significado
 - Cria uma função `func` que possui argumentos `arg1..argN`
 - Avalia a expressão `expression` no lado direito
 - Retorna o seu resultado

M02 - Funções

7. Funções arrow

- Exemplo de uma função arrow e de um expressão de função similar:

Função arrow

```
let sum = (a, b) => a + b;  
alert(sum(1, 2)); // 3
```

Expressão de função

```
let sum = function (a, b) {  
    return a + b;  
};  
  
alert(sum(1, 2)); // 3
```

MO2 - Funções

7. Funções arrow

- Remoções:
 - palavra **function**
 - chavetas
 - palavra **return**
- Adições:
 - seta (**=>**)

Função arrow

```
let sum = (a, b) => a + b;  
alert(sum(1, 2)); // 3
```

Expressão de função

```
let sum = function (a, b) {  
    return a + b;  
};  
alert(sum(1, 2)); // 3
```

M02 - Funções

7. Funções arrow

- Se tivermos apenas um argumento, então os parênteses podem ser omitidos, tornando a escrita da função ainda mais curta:

```
// o mesmo que  
// let double = function(n) { return n * 2 }  
let double = n => n * 2;  
  
alert(double(3)); // 6
```

M02 - Funções

7. Funções arrow

- Se não houver argumentos, os parênteses devem estar vazios (mas devem estar presentes)

```
let sayHi = () => alert("Olá!");  
  
sayHi();
```


M02 - Funções

7. Funções arrow

- Exemplos anteriores receberam argumentos da esquerda de => e avaliaram a expressões simples
- Às vezes temos várias expressões ou declarações
- Para tal envolva tudo em chavetas e use a palavra **return**

```
let sum = (a, b) => { // a chaveta abre um função multi-linha
  let result = a + b;
  return result; // se usar chavetas, use return para devolver resultados
};

alert(sum(1, 2)); // 3
```

M02 - Funções

7. Funções arrow

- As funções **arrow** podem parecer estranhas e pouco legíveis no início, mas isso muda rapidamente à medida que os olhos se acostumam com a estrutura
- Elas são muito convenientes para ações simples de uma linha, quando não queremos escrever muito código