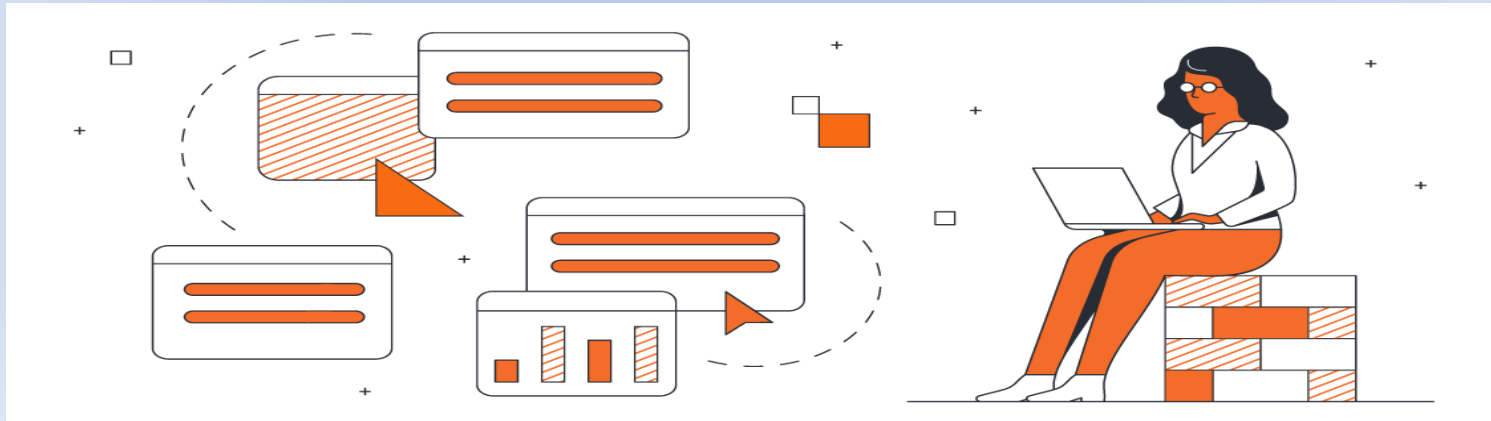


Modularização

- ✓ É a capacidade de dividir o programa em partes distintas que desempenham tarefas específicas e que podem ser combinadas para resolver problemas mais complexos.
- ✓ A modularização nos permite escrever programas mais legíveis, mais fáceis de manter e reusar e, muitas vezes, com melhor desempenho.
- ✓ Em JavaScript, a modularização é feita geralmente através da criação e do uso de funções, pois são estruturas de código que permitem que o usuário organize seus programas em partes menores e mais simples.



JS



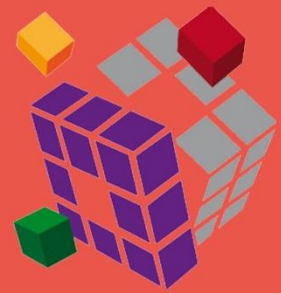
Uma função é um conjunto de instruções que recebem entradas, fazem alguns cálculos específicos e produzem saída. Basicamente, uma função é um conjunto de instruções que executa algumas tarefas ou alguns cálculos e, em seguida, retorna o resultado ao usuário.

Declaração de Funções

A declaração de uma função consiste em

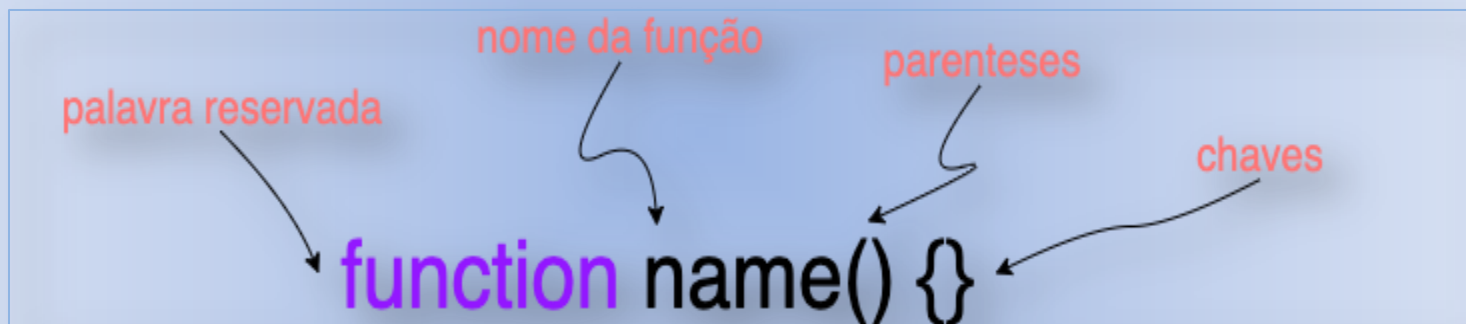
- ✓ A palavra-chave `function`;
- ✓ O nome da função, ou seu identificador, seguido de parênteses;
- ✓ O corpo da função, ou o bloco de instruções necessárias para desempenhar uma tarefa, delimitado pelas chaves.

Função em JavaScript



Declaração de Funções

A maneira mais primitiva é usando declarações a partir da palavra-chave `function`.



Exemplo:

```
function olaMundo() {  
    console.log('Olá Mundo!');  
}
```

Chamada de Funções

Para que o código dentro de uma função seja executado, é necessário realizar o processo de **chamar a função**.

Exemplo:

```
> // declaração
function saudacao(){
    console.log("Olá!");
}


// chamada
saudacao()

Olá!
```

Chamada de Funções

Uma característica interessante do Javascript é a possibilidade de realizar chamadas de função antes da efetiva declaração dessa mesma função.

Exemplo:

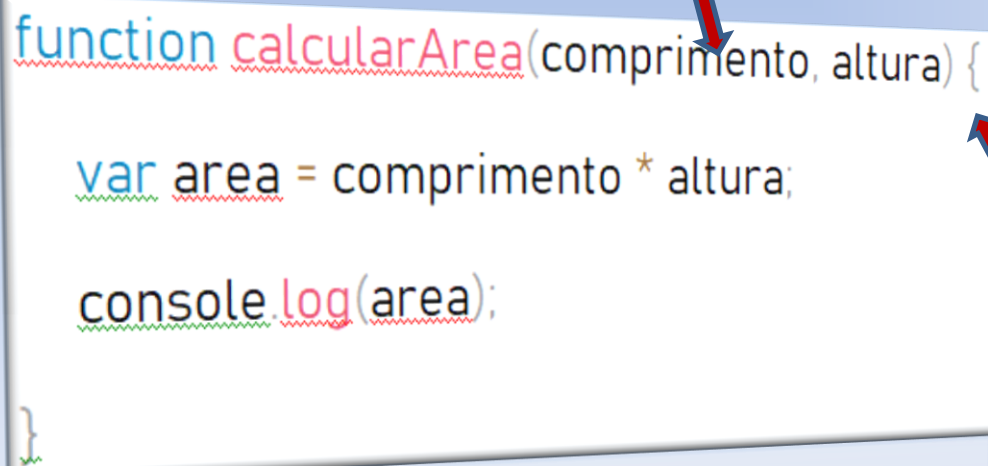


```
olaMundo();  
  
function olaMundo() {  
    console.log('Olá Mundo!');  
}
```

Parâmetros

- Quando estamos declarando uma função, podemos especificar **parâmetros**.
- Parâmetros permitem que funções aceitem entradas e manipulem essas entradas para desempenhar alguma tarefa.
- Usamos parâmetros para definir qual a informação deve ser passada para a função quando ela é chamada.

Exemplo:



```
function calcularArea(comprimento, altura) {  
    var area = comprimento * altura;  
    console.log(area);  
}
```

The image shows a code snippet for a JavaScript function named `calcularArea`. The function takes two parameters: `comprimento` and `altura`. Inside the function, a variable `area` is calculated as the product of `comprimento` and `altura`, and then logged to the console. Two red arrows with blue outlines point to the parameters `comprimento` and `altura` in the function signature.

Argumentos

- Os valores que são passados para uma função quando ela é chamada são chamados de **argumentos**. Argumentos podem ser passados como valores ou como variáveis.

Exemplo:

```
function calcularArea(comprimento, altura) {  
    var area = comprimento * altura;  
    console.log(area);  
}
```



```
calcularArea(10, 6); // chamando a função com valores como argumentos
```

```
var comprimento_retangulo = 5;
```

```
var altura_retangulo = 6;
```




```
calcularArea(comprimento_retangulo, altura_retangulo); // chamando a  
função com variáveis como argumentos
```


Parâmetro Padrão ou Pré-definidos

- Permitem que parâmetros regulares sejam inicializados com valores iniciais caso undefined ou nenhum valor seja passado.

Exemplo:



```
function ola(nome = 'Estranho') {  
    console.log('Olá, ' + nome + '!');  
}  
  
ola('Maria'); // retorna "Olá Maria!"  
ola(); // retorna "Olá Estranho!"
```

return

- A **declaração return** finaliza a execução de uma função e especifica os valores que devem ser retonados para onde a função foi chamada.

Exemplo:

```
function calcularArea(comprimento, altura) {  
    var area = comprimento * altura;  
    return area; // retorna o valor atribuído a variável 'area' e encerra a  
    execução da função.  
}  
  
console.log(calcularArea(10, 6));
```

Funções Auxiliares

- São funções que são chamadas dentro de outras funções.
- Podemos usar o valor de retorno de uma função dentro de outra função.
- Funções auxiliares podem ajudar a quebrar tarefas grandes e complexas em tarefas menores e mais simples.

Exemplo:

```
// Fórmula para conversão:  
// fahrenheit = 9/5*(celsius) + 32  
  
function multiplicaPorNoveQuintos(numero) {  
    return numero * (9 / 5);  
}  
  
function converteParaFahrenheit(celsius) {  
    return multiplicaPorNoveQuintos(celsius) + 32; // invocação da nossa  
    função auxiliar!  
}  
  
console.log(converteParaFahrenheit(15)); // Retorna 59
```

Expressão de função

- Permite criar uma **função** anônima que não tem nenhum nome de **função**, que é a principal diferença entre **Expressão de Função** e **Declaração de Função**.
- Para invocar uma expressão de função, basta escrever o nome da variável na qual a função foi armazenada, seguido dos argumentos envoltos de parênteses




```
var eFimDeSemana = function(dia) {  
    if (dia === 'Sábado' || dia === "Domingo") {  
        return true;  
    } else {  
        return false;  
    }  
}  
  
console.log(eFimDeSemana('Segunda-Feira'))
```

Exemplo:

Arrow function

- O Javascript também permite escrever funções usando a sintaxe conhecida como **função flecha**, ou *Arrow function*. Esse tipo de sintaxe permite omitir a palavra-chave `function`, deixando a definição de uma função bem mais reduzida.
- Quando falamos de Arrow Functions, elas são sempre expressões, e portanto, são sempre funções anônimas.

Exemplo:



```
var calcularArea = (comprimento, altura) => {  
    return comprimento * altura;  
}  
  
console.log(calcularArea(10, 6));
```

Arrow function

Comparando a Arrow Function com as outras funções, podemos perceber vários benefícios:

- Ela **pode** ser escrita em apenas em 1 linha de código.
- Sem a palavra-chave `function`.
- Sem a palavra-chave `return`.
- Sem o uso de chaves `{ }` (quando tiver apenas uma linha de comando)



Exemplos

Tipos de declarações

```
// declaração da função  
function sayHelloWorld() {  
  return 'Hello World!'  
}
```

```
// declaração da função como expressão  
const sayHelloWorld2 = function() {  
  return 'Hello World!'  
}
```

```
// declaração da arrow function  
const sayHelloWorld3 = () => 'Hello World!'
```

Chamadas

```
// chamado da função  
console.log(sayHelloWorld())  
console.log(sayHelloWorld2())  
console.log(sayHelloWorld3())
```

Saídas

```
// saída:  
// Hello World!  
// Hello World!  
// Hello World!
```