

A horizontal bar with a blue segment on the left and an orange segment on the right.

4954/Programador Web

Qualificação Profissional

Eixo Tecnológico: Informação e Comunicação
Segmento: Tecnologia da Informação (TI)



Prof. Davi Gomes

Funções (*Functions*)

O que são Funções?

Ao aprender a calcular a área de um retângulo, há uma sequência de etapas para calcular a resposta correta:

- Meça a largura do retângulo.
- Meça a altura do retângulo.
- Multiplique a largura e a altura do retângulo.

Área do Retângulo



$$A = b \cdot h$$



O que são Funções?

E se quiséssemos calcular a área de três retângulos diferentes?

```
// Área do primeiro retângulo
const largura1 = 10;
const altura1 = 6;
const area1 = largura1 *
altura1;
// Área do segundo retângulo
const largura2 = 4;
const altura3 = 9;
const area2 = largura2 *
altura3;
// Área do terceiro retângulo
const largura3 = 10;
const altura3 = 10;
const area3 = largura3 *
altura3;
```

Functions

example 1

Declaração de Funções

Em JavaScript, existem muitas maneiras de criar uma função. Uma maneira de criar uma função é usando uma *declaração de função*. Assim como uma declaração de variável vincula um valor a um nome de variável, uma declaração de função vincula uma função a um nome ou um *identificador*. Dê uma olhada na anatomia de uma declaração de função ao lado:

FUNCTION
KEYWORD

IDENTIFIER

```
function greetWorld() {  
    console.log('Hello, World!');  
}
```

KEY

- Function body

Declaração de Funções

Uma declaração de função consiste em:

- A palavra- chave *function*.
- O nome da função, ou seu identificador, seguido de parênteses.
- Um corpo de função ou o bloco de instruções necessárias para executar uma tarefa específica, entre as chaves da função, { } .

FUNCTION
KEYWORD IDENTIFIER

```
function greetWorld() {  
    console.log('Hello, World!');  
}
```

KEY

● Function body

Chamando uma função

Uma declaração de função vincula uma função a um identificador.

No entanto, uma declaração de função não solicita que o código dentro do corpo da função seja executado, apenas declara a existência da função. O código dentro do corpo de uma função é executado, ou executa, somente quando a função é *chamada*.

IDENTIFIER

```
graph TD; IDENTIFIER --- greetWorld["greetWorld()"]; style greetWorld fill:none,stroke:#800080,stroke-width:2px
```

`greetWorld()`;

Essa *chamada de função* executa o corpo da função ou todas as instruções entre as chaves na declaração da função.

```
graph TD; 1["① function greetGreeting() {"] --- 3["③ console.log('Hello, World!');"}; 3 --- 2["② greetGreeting();"]; 2 --- 4["④ // Code after function call"]; 4 --> 3;
```

① `function greetGreeting() {`
③ `console.log("Hello, World!");`
}
② `greetGreeting();`
④ `// Code after function call`

Parâmetros e Argumentos

Até agora, as funções que criamos executam uma tarefa sem uma entrada. No entanto, algumas funções podem receber entradas e usar as entradas para executar uma tarefa. Ao declarar uma função, podemos especificar seus *parâmetros*.

```
graph TD; P[PARAMETERS] --- F[function calculateArea(width, height) {  
  console.log(width * height);  
}]; F --- W[width]; F --- H[height]; W --- V[PARAMETERS ARE TREATED LIKE  
VARIABLES WITHIN A FUNCTION]; H --- V;
```

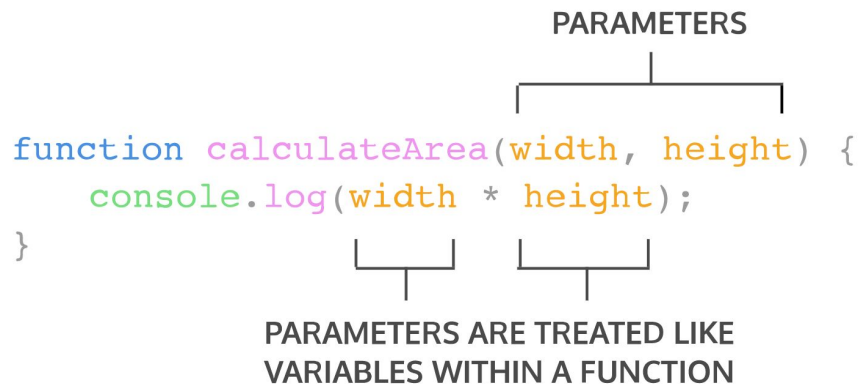
PARAMETERS

```
function calculateArea(width, height) {  
  console.log(width * height);  
}
```

PARAMETERS ARE TREATED LIKE
VARIABLES WITHIN A FUNCTION

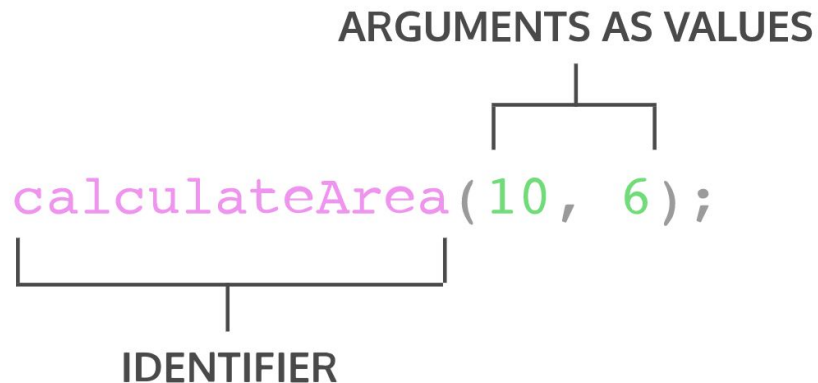
Parâmetros e Argumentos

No diagrama ao lado, `calculateArea()`, calcula a área de um retângulo, com base em duas entradas, `width` e `height`. Os parâmetros são especificados entre parênteses como `width` e `height`, e dentro do corpo da função, eles agem como variáveis regulares. `width` e `height` agem como espaços reservados para valores que serão multiplicados juntos.



Parâmetros e Argumentos

Na chamada de função acima, o número 10 é passado como `width` e 6 é passado como `height`. Observe que a ordem na qual os argumentos são passados e atribuídos segue a ordem em que os parâmetros são declarados.



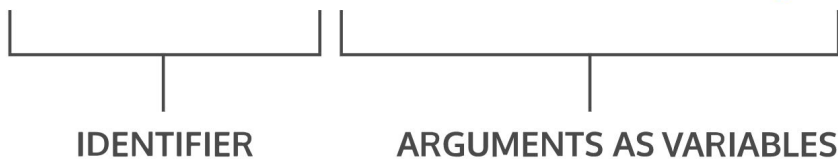


Parâmetros e Argumentos

As variáveis `rectWidth` e `rectHeight` são inicializadas com os valores de altura e largura de um retângulo antes de serem usadas na chamada da função.

```
const rectWidth = 10;  
const rectHeight = 6;
```


```
calculateArea(rectWidth, rectHeight);
```





Parâmetros Padrão

Os parâmetros padrão permitem que os parâmetros tenham um valor predeterminado caso não haja nenhum argumento passado para a função ou se o argumento for `undefined` quando chamado.




```
function greeting (name = 'stranger') {  
  console.log(`Hello, ${name}!`)  
}  
  
greeting('Nick') // Output: Hello, Nick!  
greeting() // Output: Hello, stranger!
```



Return

Quando uma função é chamada, o computador executa o código da função e avalia o resultado da chamada da função. Por padrão, esse valor resultante é `undefined`.



```
function areaDoRetangulo(largura, altura) {  
  let area = largura * altura;  
}  
console.log(areaDoRetangulo(5, 7)) // Imprime undefined
```

Return

Para devolver informação da chamada de uma função, nós utilizamos uma **instrução de retorno**. Para criar uma função de retorno, nós usamos a palavra-chave `return`. Como vimos anteriormente, se o valor é omitido, o retorno será `undefined`.


```
function calculateArea(width, height) {  
    const area = width * height;  
    return area;  
}
```

RETURN KEYWORD RETURN VALUE



Return

Quando uma instrução `return` é usada no corpo de uma função, a execução da função é interrompida e o código que a segue não será executado.

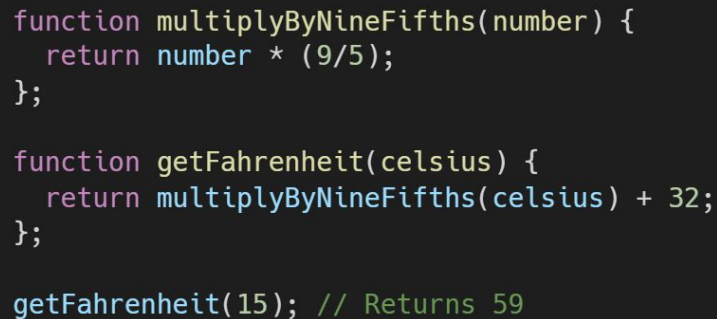


```
function areaDoRetangulo(largura, altura) {  
  if (largura < 0 || altura < 0) {  
    return 'Os valores precisam ser maiores que zero!';  
  }  
  return largura * altura;  
}
```




Funções Auxiliares

Também podemos usar o valor de retorno de uma função dentro de outra função. Essas funções sendo chamadas dentro de outra função geralmente são chamadas de *funções auxiliares*. Como cada função está realizando uma tarefa específica, torna nosso código mais fácil de ler e depurar, se necessário.



```
function multiplyByNineFifths(number) {  
  return number * (9/5);  
};  
  
function getFahrenheit(celsius) {  
  return multiplyByNineFifths(celsius) + 32;  
};  
  
getFahrenheit(15); // Returns 59
```



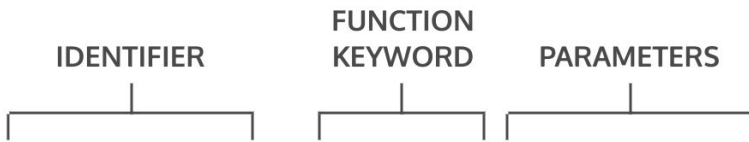
Funções Auxiliares

Podemos usar funções para separar pequenos pedaços de lógica ou tarefas e, em seguida, usá-los quando necessário. Escrever funções auxiliares pode ajudar a pegar tarefas grandes e difíceis e dividi-las em tarefas menores e mais gerenciáveis.

```
function multiplyByNineFifths(number) {  
  return number * (9/5);  
};  
  
function getFahrenheit(celsius) {  
  return multiplyByNineFifths(celsius) + 32;  
};  
  
getFahrenheit(15); // Returns 59
```

Expressões de Função

Outra maneira de definir uma função é usar uma expressão de função . Podemos usar a palavra- chave *function*. Em uma expressão de função, o nome da função geralmente é omitido. Uma função sem nome é chamada de *função anônima* . Uma expressão de função geralmente é armazenada em uma variável para se referir a ela.



```
const calculateArea = function(width, height) {  
    const area = width * height;  
    return area;  
};
```

The diagram illustrates the components of a function expression. It shows three categories: IDENTIFIER, FUNCTION KEYWORD, and PARAMETERS. The code example below shows how these components are used in a function expression.

Expressões de Função

Para declarar uma expressão de função:

1. Declare uma variável para fazer com que o nome da variável seja o nome ou identificador de sua função.
2. Atribua como valor dessa variável uma função anônima criada usando a palavra-chave `function` seguida de um conjunto de parênteses com possíveis parâmetros. Em seguida, um conjunto de chaves que contém o corpo da função.

```
graph TD
    A[IDENTIFIER] --- B[FUNCTION KEYWORD]
    A --- C[PARAMETERS]
```

```
const calculateArea = function(width, height) {
  const area = width * height;
  return area;
};
```



Expressões de Função

Para invocar uma expressão de função, escreva o nome da variável na qual a função é armazenada, seguido de parênteses envolvendo todos os argumentos que estão sendo passados para a função.



```
variableName(argument1, argument2)
```



Arrow Functions

O ES6 introduziu a sintaxe chamada *arrow function* `() =>`, *arrow functions* eliminam a necessidade de digitar a palavra-chave `function` toda vez que você precisa criar uma função. Em vez disso, você primeiro inclui os parâmetros dentro de `()` e, em seguida, adiciona uma seta `=>` que aponta para o corpo da função cercado `{ }`



```
const areaDoRetangulo = (largura, altura) => {  
  let area = largura * altura;  
  return area;  
};
```



Arrow Functions

(Formas Concisas)

O JavaScript também fornece várias maneiras de refatorar a sintaxe de *arrow function*.

1.

As funções que usam apenas um único parâmetro não precisam que esse parâmetro seja colocado entre parênteses. No entanto, se uma função tiver zero ou vários parâmetros, os parênteses serão necessários.

ZERO PARAMETERS

```
const functionName = () => {};
```

ONE PARAMETER

```
const functionName = paramOne => {};
```

TWO OR MORE PARAMETERS

```
const functionName = (paramOne, paramTwo) => {};
```



Arrow Functions (Formas Concisas)

2.

Um corpo de função composto por um bloco de linha única não precisa de chaves. Sem as chaves, o que quer que a linha avalie será retornado automaticamente. O conteúdo do bloco deve seguir imediatamente a seta => e a palavra-chave `return` pode ser removida. Isso é conhecido como retorno implícito.

SINGLE-LINE BLOCK

```
const sumNumbers = number => number + number;
```

MULTI-LINE BLOCK

```
const sumNumbers = number => {  
    const sum = number + number;  
    return sum; } — RETURN STATEMENT  
};
```

Revisão

