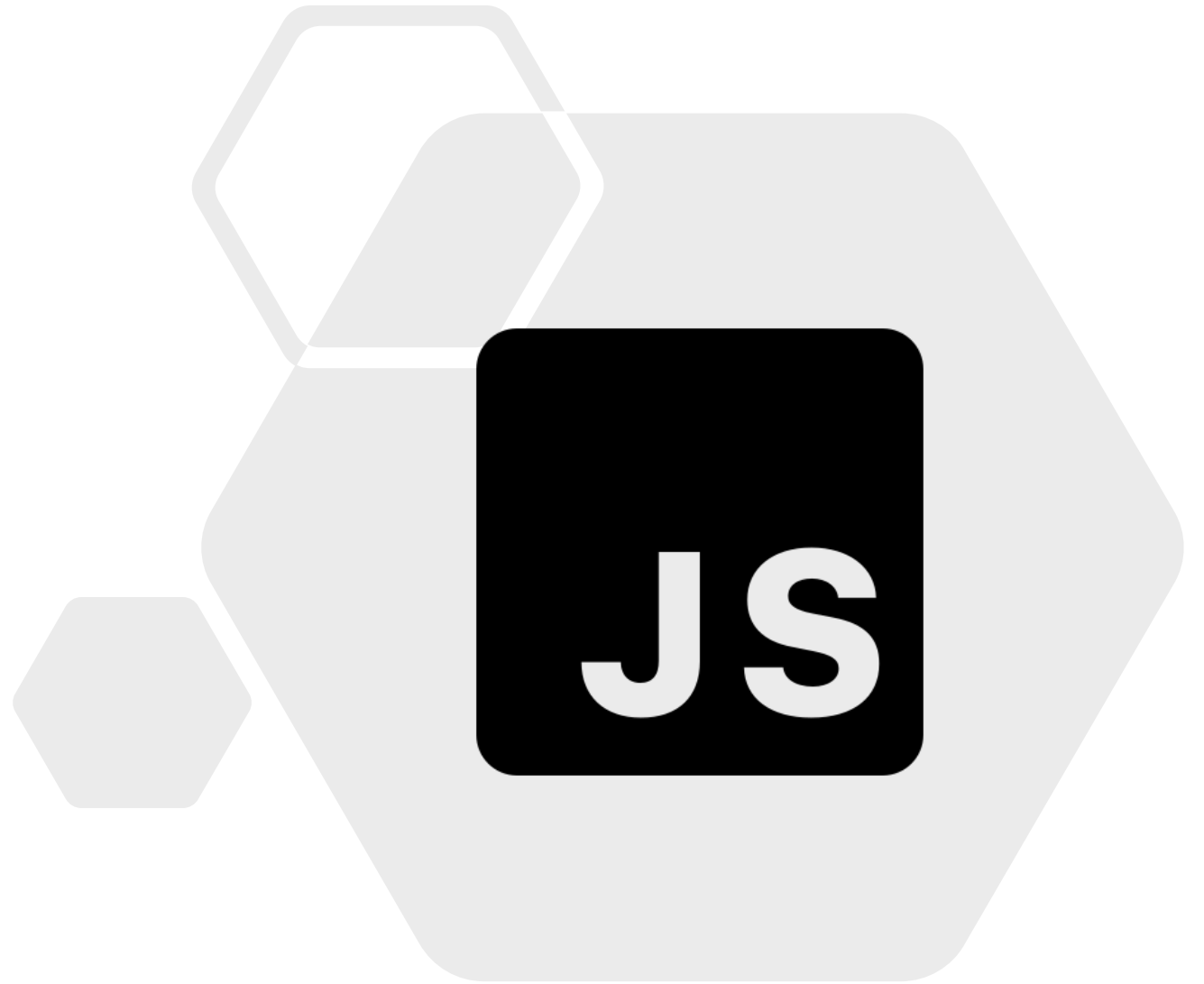
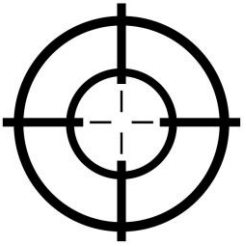


JavaScript



# O que é o JavaScript?

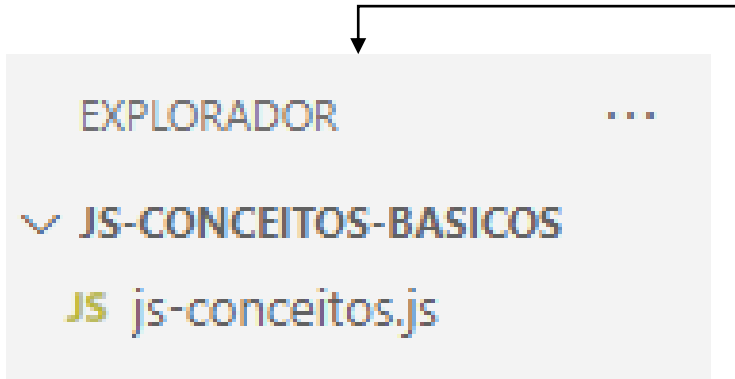
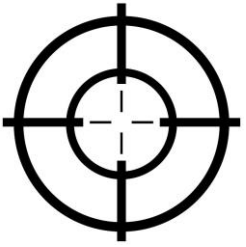


JavaScript é uma linguagem de programação interpretada multiparadigma, de script em alto nível, com tipagem dinâmica.

É uma linguagem de programação em conformidade com a especificação ECMAScript.

JavaScript é uma das principais tecnologias da World Wide Web.

# Iniciando a programação com JavaScript



Pasta e arquivo de código fonte JavaScript, atenção que a extensão é ".js".

Para executar, digite o comando “node js-conceitos.js”

```
console.log('iniciando com o JavaScript em back-end')
```

O `console.log()` é uma função em JavaScript que é usada para imprimir qualquer tipo de variável definida anteriormente ou apenas para imprimir qualquer mensagem que precise ser exibida ao usuário.

# Tipos das variáveis no JavaScript?



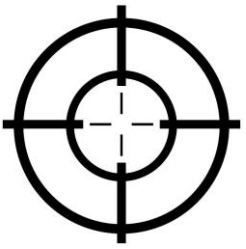
Variáveis JavaScript podem ser do tipo: Números, Strings, Objetos, Arrays ou Funções (Numbers, Strings, Objects, Arrays ou Functions).

JavaScript usa as palavras-chave "var", "let" e "const" para declarar variáveis.

```
let steveRogers = 'Capitão América'  
var tonyStark = "Homen de Ferro"  
console.log("Super-heróis da Marvel: "+steveRogers+" "+tonyStark)
```

```
let pagamento = 1500.50  
var conta = 2356  
console.log("Fazer pagamento de: "+pagamento+" Na conta: "+conta)
```

# Declaração de variáveis com “var”



A palavra-chave "var" é usada, desde o principio, para declarar variáveis em JavaScript.

Quando se declara uma variável usando o "var", esta pode ter seu valor reatribuído, comportamento comum em várias linguagens de programação. Também pode ser redeclarada.

```
var nome = "Príncipe T'Challa"  
console.log("Meu nome é: "+nome)  
nome = "Black Panther"  
console.log("Meu nome é: "+nome)
```

Neste exemplo a variável nome teve seu valor alterado.

# Hoisting



```
nome = "Príncipe T'Challa"  
console.log("Meu nome é: "+nome)  
nome = "Black Panther"  
console.log("Meu nome é: "+nome)
```

```
var nome;
```

Importante! Variáveis declaradas com “var” não precisam ser declaradas antes do uso!

Em JavaScript, toda variável é “elevada/içada” (hoisting) até o topo do seu contexto de execução. Esse mecanismo move as variáveis para o topo do seu escopo antes da execução do código.

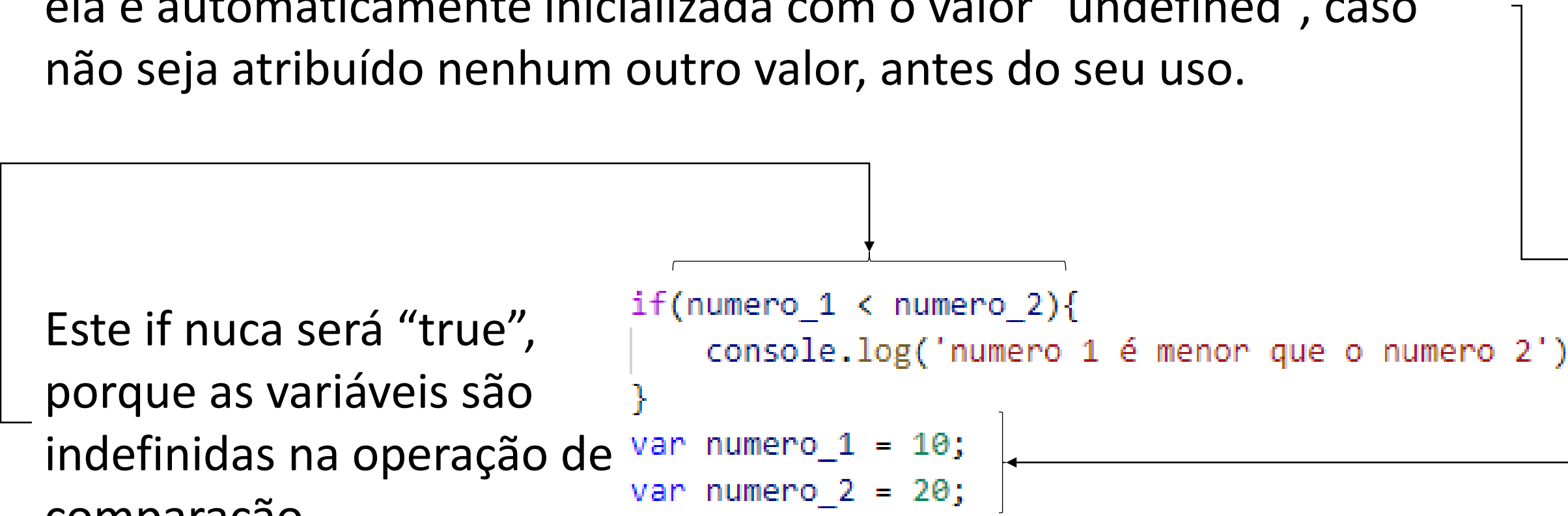
# Hoisting



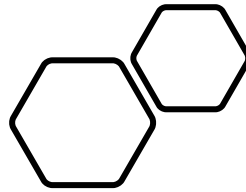
No caso da palavra-chave “var”, além da variável ser içada (hoisting) ela é automaticamente inicializada com o valor “undefined”, caso não seja atribuído nenhum outro valor, antes do seu uso.

Este if nunca será “true”, porque as variáveis são indefinidas na operação de comparação.

```
if(numero_1 < numero_2){  
    console.log('numero 1 é menor que o numero 2')  
}  
var numero_1 = 10;  
var numero_2 = 20;
```

A diagram illustrating variable hoisting. A bracket on the left groups the variable declarations 'var numero\_1 = 10;' and 'var numero\_2 = 20;'. An arrow points from this bracket to the 'if' statement in the code block above. Another arrow points from the 'if' statement to the text on the left, which explains that the comparison will never be true because the variables are undefined at that point in the execution.

# Hoisting



- Por causa do hoisting, variáveis que são declaradas com “var” podem não respeitar seu escopo, fazendo com que fiquem ainda disponíveis fora do seu escopo.

```
if(true){  
  var hulk = 'Eric Bana'  
  console.log(hulk)  
}  
console.log(hulk)
```



# Qual o problema com o “var”?



Imagine que o código contenha muitas linhas e que sua complexidade não seja trivial de compreender. Muitas vezes, deseja-se declarar variáveis que serão utilizadas apenas dentro de um pequeno trecho do nosso código.

Ter que lidar com o escopo das variáveis declaradas com “var” (escopo abrangente) pode confundir o desenvolvedor.

# Declaração de variáveis com “let”



A palavra-chave “let” foi introduzida no ES6 (ECMAScript 6) em 2015.

Importante! Variáveis declaradas com “let” não podem ser redeclaradas e devem ser declaradas antes do uso.

Importante! Variáveis declaradas com "let" possuem escopo de bloco.

```
let hulk = 'Eric Bana'  
let hulk = 'Mark Ruffalo'  
console.log(hulk)
```

```
if(true){  
    let hulk = 'Mark Ruffalo'  
    console.log(hulk)  
}  
console.log(hulk)
```

# E o const?

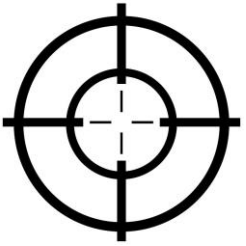


A palavra-chave “const” foi introduzida no ES6 (ECMAScript 6) em 2015.

Variáveis declaradas com const não podem ser redeclaradas nem reatribuídas. Também possuem escopo de bloco, como com "let".

Importante! Precisam ter seu valor atribuído no momento da declaração!

# Dado do tipo objeto



É importante lembrar que variáveis podem conter diferentes tipos de dados como: números, strings, objetos e mais.

```
let idade = 80 // Number
let nome = "Steve Rogers" // String
let capitao = { nome: "Steve Rogers", idade: 80 } // Object
console.log('Nome do personagem: ' + capitao.nome + " idade: " + capitao['idade'])
```

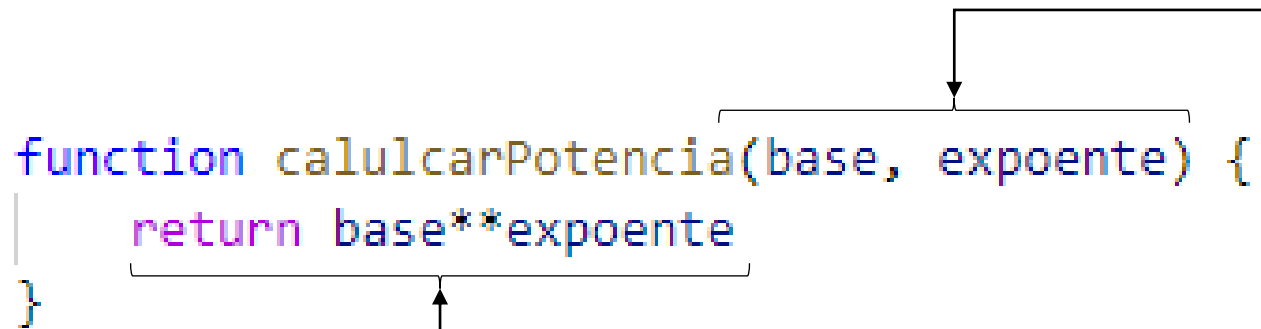
Importante! Perceba o detalhe ao acesso dos atributos (também chamados de propriedades) de um objeto.

# Funções



Uma função é um bloco de código projetado para executar uma tarefa específica. Uma função é executada quando “algo” a invoca (chama).

```
function calcularPotencia(base, expoente) {  
  return base**expoente  
}
```

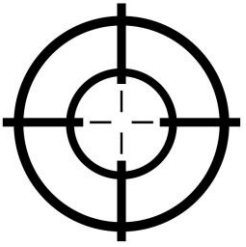
A diagram with two lines. One line starts from the text 'Estes elementos são chamados de parâmetros ou argumentos.' and points to the parameters 'base' and 'expoente' in the function definition. Another line starts from the same text and points to the arguments '2' and '3' in the function call.

```
let resultado = calcularPotencia(2, 3)  
console.log('resultado: ' + resultado)
```

Estes elementos são chamados de parâmetros ou argumentos.

Uma função pode retornar ou não, algo a partir do seu processamento.

# Parâmetros ou argumentos?



Os termos parâmetro e argumento podem ser usados para a mesma coisa: informações que são passadas para uma função. Contudo, significam coisas diferentes.

Um parâmetro é a variável listada entre parênteses na definição da função.

```
function calcularPotencia(base, expoente) {  
    return base**expoente  
}
```

```
let resultado = calcularPotencia(2, 3)  
console.log('resultado: ' + resultado)
```

Um argumento é o valor enviado para a função quando ela é chamada.

# Sobre funções...



As definições de função, em JavaScript, não especificam tipos de dados para parâmetros.

As funções, em JavaScript, não executam verificação de tipo nos argumentos passados.

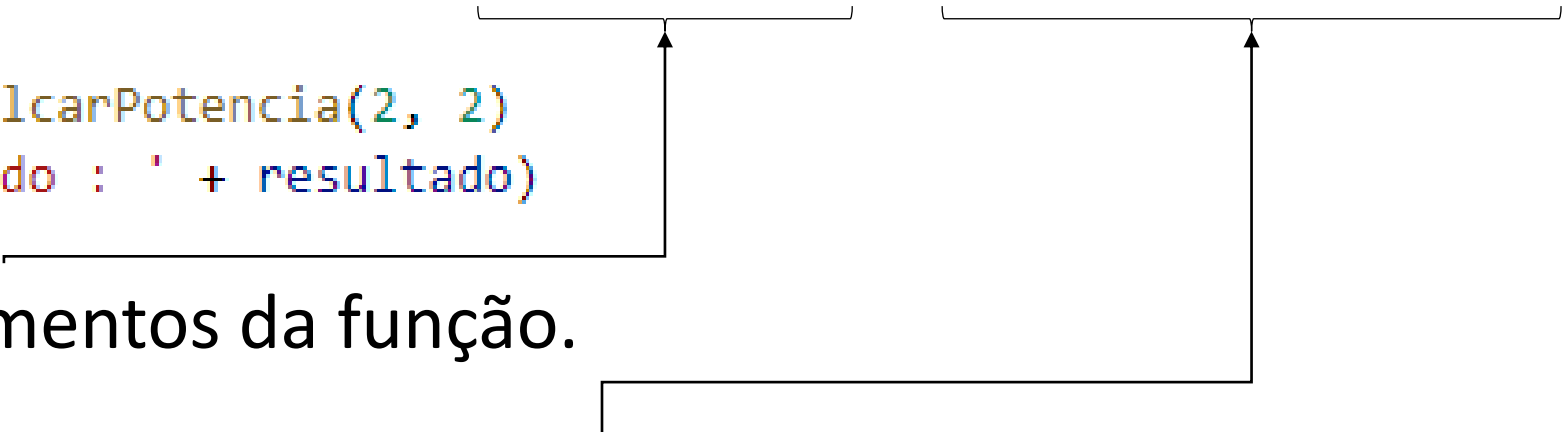
As funções, em JavaScript, não verificam o número de argumentos recebidos. Se uma função for chamada com argumentos ausentes (menos do que declarados), os valores ausentes serão definidos como indefinidos.

# Funções também são objetos



Uma função pode ser tratada com um objeto. Pode ser criada através de um método especial, chamado de construtor!

```
let calcularPotencia = new Function("base,expoente", "return base**expoente")  
  
let resultado = calcularPotencia(2, 2)  
console.log('resultado : ' + resultado)
```

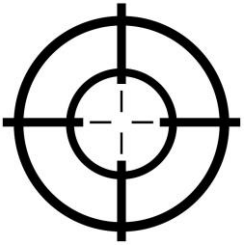
A diagram with two horizontal curly braces above the first two lines of code. The first brace is under "base,expoente" and the second is under "return base\*\*expoente". Two vertical arrows point from these braces down to the arguments "2" and "2" in the function call "calcularPotencia(2, 2)". A horizontal line connects the two vertical arrows, and a vertical line extends down from its center to the text "Estes são os argumentos da função." below.

Estes são os argumentos da função.

Logo após os argumentos deve ser declarado o corpo da função, ou seja, o que ela deve fazer.



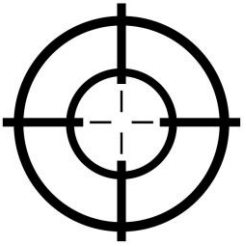
# O que é um método?



Os objetos podem ter métodos e métodos são ações que podem ser executadas pelos objetos.

Importante! Os métodos são armazenados nas propriedades como definições de função. Portanto, um método é uma função armazenada como uma propriedade.

# Como usar métodos?



```
const personagem = {  
  apelido: "Capitão America",  
  nome: "Steve Rogers",  
  habilidades: "força e velocidade",  
  descricao: function () {  
    return "Nome: " + this.nome + " Apelido: " + this.apelido + " poderes: " + this.habilidades  
  }  
}
```

```
console.log(personagem.descricao())
```

O “this” em uma função, refere-se ao “proprietário” da função, ou seja, o objeto. No exemplo acima, é o objeto personagem que “possui” a função “descricao”.

# Variáveis podem “ser” funções



Uma função também pode ser definida usando uma expressão, que pode ser armazenada em uma variável.

```
soma = function (valor1, valor2) {  
    return valor1 + valor2  
}  
console.log("Valor da soma: " + soma(3, 5))
```

Depois que uma função foi armazenada em uma variável, a variável pode ser usada como uma função.

A função acima é na verdade uma função anônima (uma função sem um nome). Funções armazenadas em variáveis não precisam de nomes, elas são sempre invocadas (chamadas) usando o nome da variável!

# Arrow functions (Funções de seta)

As “arrow functions” permitem uma sintaxe curta para escrever de funções.

```
soma = (valor1, valor2) => {  
  |   return valor1 + valor2  
  |  
}  
console.log("Valor da soma: " + soma(3, 5))
```

Perceba que é uma forma de fazer funções anônimas.

Pode-se omitir a palavra-chave "return" e as chaves se a função for uma única instrução.

```
soma = (valor1, valor2) => valor1 + valor2  
console.log("Valor da soma: " + soma(3, 5))
```

# Sobre arrow functions...



Não são adequadas para definir métodos de objeto. Eles devem ser definidos antes de serem usados.

Geralmente, os desenvolvedores declaram este tipo de função como valor de variáveis usando o “const”.

JavaScript

