

Function - Parte 1

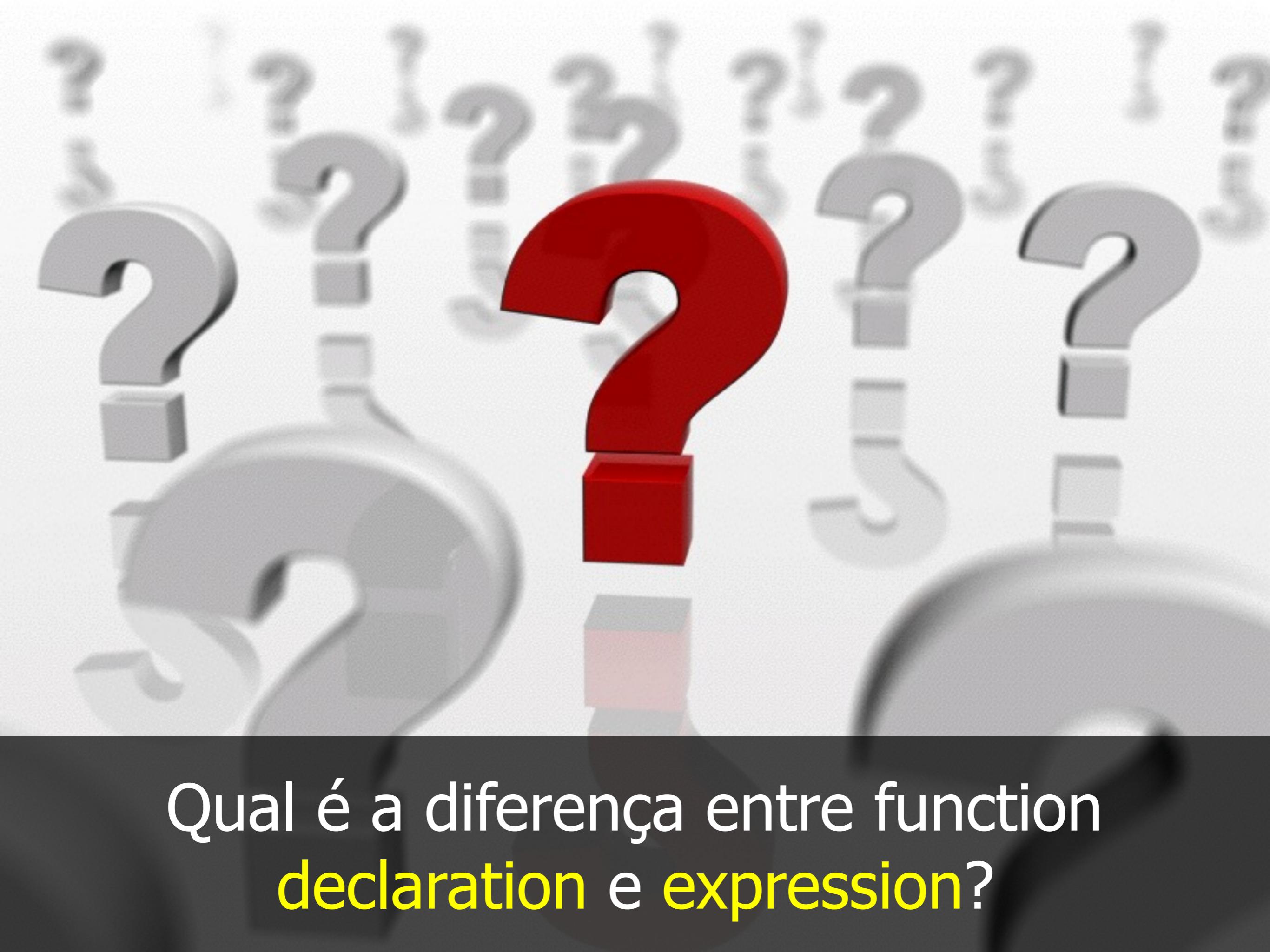
Uma função é um objeto que **contém** um bloco de código **executável** e **isolado**. Além disso, as funções são de primeira classe, ou seja, pode ser atribuída a uma variável, passada por parâmetro ou ser retornada de outra função.

```
1. function sum(a, b) {  
2.     return a + b;  
3. }
```

Function Declaration

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

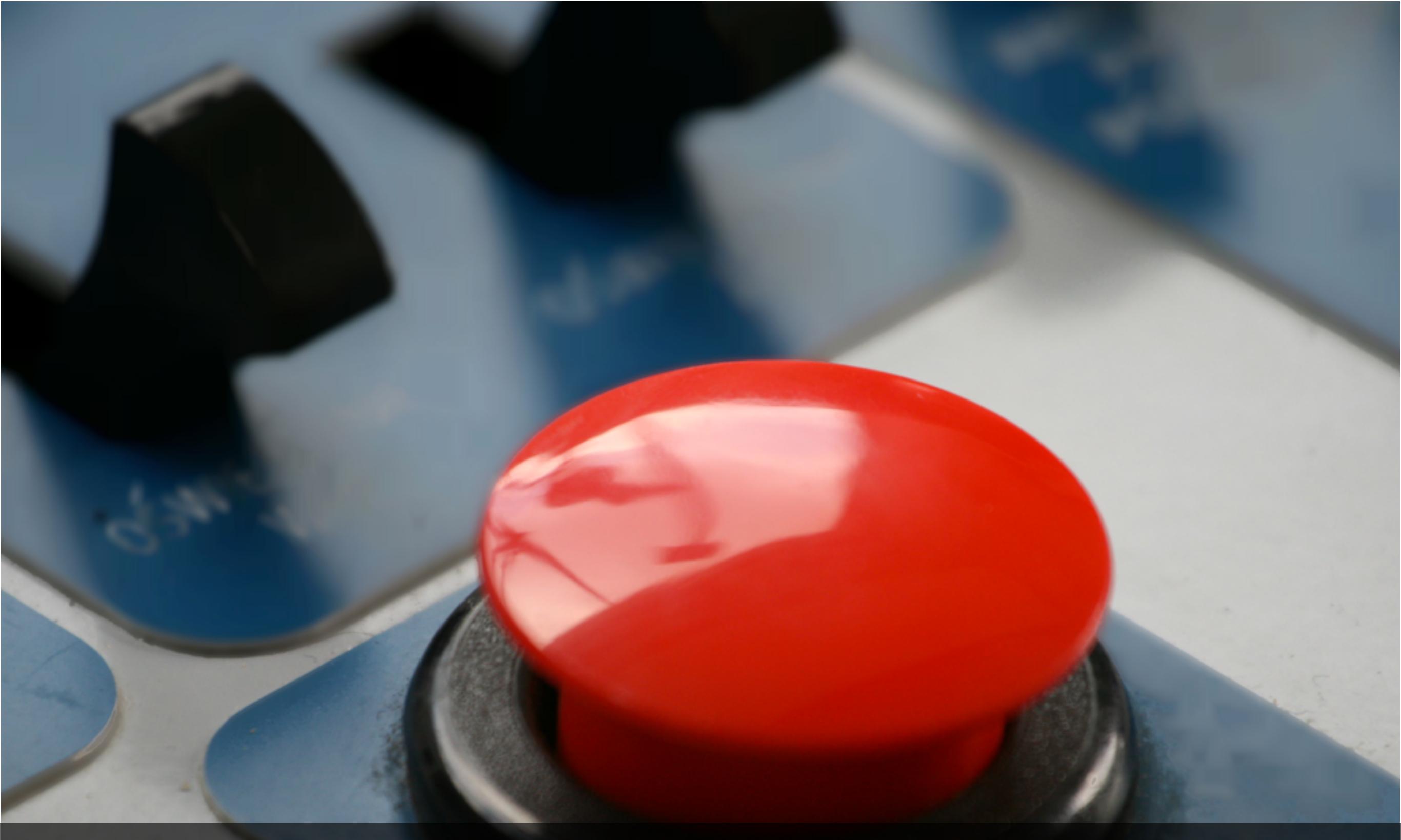
Function Expression



Qual é a diferença entre function
declaration e expression?

```
1. let sum = function (a, b) {  
2.     return a + b;  
3. };  
4. let subtract = function (a, b) {  
5.     return a - b;  
6. };  
7. let calculator = function (fn) {  
8.     return function (a, b) {  
9.         return fn(a, b)  
10.    }  
11.};
```

Entendendo funções de **primeira classe**



Formas de passar parâmetros na
invocação de uma função

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

Invocando uma função diretamente

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

Invocando uma função **default parameter values**

```
1. let sum = function () {  
2.   let total = 0;  
3.   for(let argument in arguments) {  
4.     total += arguments[argument];  
5.   }  
6.   return total;  
7. };
```

Invocando uma função sem parâmetros,
utilizando a variável **ímplicita arguments**

```
1. let sum = function (...numbers) {  
2.   let total = 0;  
3.   for(let number of numbers) {  
4.     total += number;  
5.   }  
6.   return total;  
7. };
```

Invocando uma função com **rest parameter**

```
1. function soma([a,b]) {  
2.     return a + b;  
3. }
```

Invocando uma função com **parameter context matching**

```
1. function soma({a,b}) {  
2.     return a + b;  
3. }  
4. var params = {  
5.     a: 1,  
6.     b: 2  
7. };  
8. soma(params); // 3
```

Invocando uma função com **parameter context matching**

```
1. function soma({c: a, d: b}) {  
2.     return a + b;  
3. }  
4. var params = {  
5.     c: 5,  
6.     d: 5  
7. };  
8. soma(params);
```

Invocando uma função com **parameter context matching**



Invocando uma função por
meio de um **objeto**

```
1. let book = {  
2.   title: "Refactoring",  
3.   author: "Martin Fowler",  
4.   pages: 342,  
5.   getTitle: function () {  
6.     return this.title;  
7.   }  
8.};
```

Invocando uma função por
meio de um **objeto**

```
1. let getTitle = function () {  
2.   return this.title;  
3. };  
4. let book = {  
5.   title: "Refactoring",  
6.   author: "Martin Fowler",  
7.   pages: 342,  
8.   getTitle: getTitle  
9. };
```

Invocando uma função por
meio de um **objeto**

```
1. let book = {  
2.   title: "Refactoring",  
3.   author: "Martin Fowler",  
4.   pages: 342,  
5.   getTitle() {  
6.     return this.title;  
7.   }  
8.};
```

Invocando uma função por
meio de um **objeto**

```
1. let getTitleFunctionName = "getTitle";
2. let book = {
3.   title: "Refactoring",
4.   author: "Martin Fowler",
5.   pages: 342,
6.   [getTitleFunctionName]() {
7.     return this.title;
8.   }
9. };
```

Invocando uma função por
meio de um **objeto**

```
1. let book = {  
2.   get title() {  
3.     return this._title;  
4.   },  
5.   set title(value) {  
6.     if (!value) throw "Invalid title";  
7.     this._title = value;  
8.   }  
9.};
```

Invocando uma função por
meio de um **objeto**



Arrow Functions

```
1. var sum = function (a, b) {  
2.   return a + b;  
3. };  
4.  
5. var sum = (a, b) => {  
6.   return a + b;  
7. };
```

Arrow Functions

```
1. var sum = function (a, b) {  
2.   return a + b;  
3. };  
4.  
5. var sum = (a, b) => a + b;
```

Arrow Functions

```
1. var print = function (value) {  
2.   console.log(value);  
3. };  
4.  
5. var print = value => console.log(value);
```

Arrow Functions

```
1. var createBook = function (title, pages) {  
2.   return {title, pages};  
3. };  
4.  
5. var createBook = (title, pages) => ({title, pages});
```

Arrow Functions