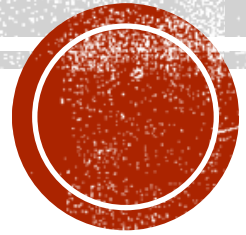
CAPÍTULO 3 - FUNÇÕES

Apresentação de um capítulo do livro "Eloquent JavaScript" escrito por Marijn Haverbeke.

Esta Apresentação foi feita por João Manuel Freitas Ribeiro

Número de Aluno: 27926



IPCA
INSTITUTO POLITÉCNICO
DO CÁVADO E DO AVE

O QUE É UMA FUNÇÃO ?

- Uma função é o conceito de comprimir uma parte do programa em apenas um valor, permitindo assim estruturar aplicações de larga escala reduzindo as repetições de Código, assossiando nomes a subprogramas e isolando esses subprogramas uns dos outros, o que promove uma melhor organização geral do código, algo fundamental para o melhor entendimento do mesmo.

COMO DECLARAR UMA FUNÇÃO ?

- Para declarar uma função precisamos usar a palavra-chave ***function*** e em seguida precisamos de lhe dar um ***nome***, entre os parênteses definimos os parâmetros, seguido do corpo da função. O corpo da função tem sempre de estar cercado por chavetas **{}**.

```
function saudacao(nome) {
    console.log("Olá, " + nome + "!");
}

saudacao("Mario");
// output-> Olá, Mario!
```

Este exemplo mostra uma função que diz olá e o nome inserido.

PARÂMETROS DE UMA FUNÇÃO

- Ao declarar uma função, podemos acrescentar também parâmetros. Ao chamar a função os parâmetros declarados têm de ser preenchidos, isso claro se forem solicitados, pois uma função não necessariamente precisa de ter parâmetros ou até mesmo um método ***return***.

```
const mostrarTexto = function () {
  console.log("Texto");
};

mostrarTexto();
// output-> Texto
```

Função que imprime a palavra "Texto".

```
const calc_Potencia = function (numero, expoente) {
  var resultado = 1;
  for (var i = 0; i < expoente; i++) {
    resultado *= numero;
  }
  return resultado;
};

console.log(calc_Potencia(4, 6));
// output-> 4096
```

Função que faz a potencia usando a Base e o Expoente inseridos.

VARIÁVEIS GLOBAIS E LOCAIS

- Ao declarar uma variável, a mesma variável pode ser declarada dentro da função, essa variável será uma variável Local e apenas poderá ser usada dentro da função em que foi declarada, enquanto que se a mesma variável for chamada em qualquer outra parte do código, tomará o valor que lhe foi atribuído inicialmente, no caso diz-se Globalmente.

```
let variable = 10;

const func = function() {
  let variable = 20;
  console.log(variable);
};

func();
// output-> 20

console.log(variable);
// output-> 10
```

Exemplo que demonstra a diferença entre variável Local e Global.

FUNÇÕES DENTRO DE FUNÇÕES

- Funções também podem ser declaradas localmente dentro de outras funções, para além de auxiliar na organização do código, permite também usar variáveis locais da função na qual foi declarada.

```
function createPlayer(name, health, strength) {
    return {
        name: name,
        health: health,
        strength: strength,
        attack: function (target) {
            console.log(" >> " + name + " attack " + target.name + " with " + strength + " damage.");
        }
    }
}

const player_mario = createPlayer("Mario", 100, 30);

player_mario.attack(player_luigi);
```

Este exemplo mostra uma função responsável criar um "personagem" com parâmetros específicos e dentro contém uma função responsável por "atacar", e usa esses parâmetros como base.

FUNÇÕES COMO VALORES

- Funções também podem ser usadas como valores e modificadas, permitindo assim que a mesma função tenha diferentes comportamentos em situações diferentes.

```
let cumprimentar = function(nome) {
   console.log("Bom Dia " + nome + "!");
};
if (horaAtual == "noite") {
    cumprimentar = function(nome) {
        console.log("Boa Noite " + nome + "!");
    }
}

cumprimentar("Mario");
```

Função que cumprimenta uma pessoa e alterna o cumprimento consoante a hora do dia.

FUNÇÕES DECLARADAS POR SETAS

- Existe também outra forma de declarar funções sem precisar da palavra chave ***function***. Usando uma seta (**=>**) que expressa algo como *"esta entrada (os parâmetros) produz este resultado (o corpo)"*. A função também pode ser organizada de forma diferente, podendo omitir os parenteses **()** e as chavetas **{}** em casos onde apenas há um parâmetro ou uma expressão, permitindo a simplificação do código.

```
const duplicar = (n) => {
  n= n * 2
  return n;
};
```

```
const quadrado1 = (x) => { return x * x; };

const quadrado2 = x => x * x;

const printText = () => console.log("text");
```

Exemplos de funções declaradas por setas.

ORDEM DE DECLARAÇÃO

- Ao declarar uma função de forma tradicional, usando a palavra-chave ***function***, podemos chamar a função mesmo antes da função ser declarada, algo que já não acontece caso a função seja declarada como valor. Isso acontece porque as funções não obedecem ao fluxo regular de controle, que funciona de cima a baixo, elas são movidas para o topo do código para que possam ser usadas livremente, já as variáveis obedecem a esse fluxo.

```
console.log("Só sei", frase());

function frase() {
  return "que nada sei.";
}
// output-> Só sei que nada sei.
```

```
console.log("Só sei", frase());

let printFrase = function frase() {
  return "que nada sei.";
}
// output-> ReferenceError: frase is not defined
```

Funções semelhantes, onde apenas difere a forma como a função é declarada, que imprimem uma frase, onde a função é chamada antes de ser declarada.

ARGUMENTOS OPCIONAIS

- Ao chamar uma função pode-se inserir argumentos a mais ou a menos, se forem inseridos argumentos a mais, o JavaScript vai ignora-los, caso sejam inseridos a menos, o JavaScript vai assumi-los como ***undefined***. Ou também dar um valor a um parâmetro fará com que caso não lhe seja dado um argumento o parâmetro irá assumir aquele valor. Isso gera uma certa flexibilidade para criar derivados tipos de funções.

```
function subtração(a, b) {
  if (b === undefined) return -a;
  else return a - b;
}

console.log(subtração(10));
//output -> -10
console.log(subtração(10, 5));
//output -> 5
```

Função caso seja dado um número o apresenta de forma negativa, porém caso sejam dados dois números a função irá fazer uma subtração.

```
function subtração(n, n2 = 1) {
  let result = n - n2;
  return result;
}

console.log(subtração(4.5));
//output -> 3.5
console.log(subtração(4.5, 2));
//output -> 2.5
```

Função que subtrai os números dados, porém caso não lhe seja dado um segundo número a função irá subtrai o valor 1.

CLOSURE

- Normalmente, quando uma função termina, as variáveis que foram declaradas localmente naquela função desaparecem, no entanto casa seja criado um **Closure** pudemos possibilitar que essas variáveis sejam lembradas pelo programa e chama-las posteriormente.

```
function wrapValue(n) {
  let local = n;
  return () => local;
}

let wrap1 = wrapValue(1);
let wrap2 = wrapValue(2);

console.log(wrap1());
//Output -> 1
console.log(wrap2());
//Output -> 2
```

Função que cria um Closure que guarda um número.

```
function multiplier(factor) {
  return number => number * factor;
}

let twice = multiplier(2);
console.log(twice(5));
//Output -> 10

let three = multiplier(3);
console.log(three(5));
//Output -> 15
```

Função que cria um Closure que guarda um multiplicador, e que chamamos o multiplicador e colocamos um número ele devolve a multiplicação.

RECURSÃO

- Recursão é quando uma função se chama a si própria, isso pode gerar um *loop*. Em muitas situações é ineficiente usar a Recursão ao invés de um *loop*, porém há alguns casos onde a recursão pode tornar o código mais simples e claro, por exemplo em ***Estruturas Hierárquicas*** onde usar a recursão ao invés de *loops* torna o código mais legível e intuitivo, ou também em problemas matemáticos recursivos como por exemplo a factoração.

```
function factorial(n) {
  if (n === 0) return 1;
  return n * factorial(n - 1);
}

console.log(factorial(5));
//output -> 120
```

```
function factorial(n) {
  let result = 1;
  for (let i = 1; i <= n; i++) {
    result *= i;
  }
  return result;
}

console.log(factorial(5));
// Output -> 120
```

Funções que fazem o calculo fatorial de um número, a primeira usa recursão e a segunda usa um loop *for*.