INSTITUTO FEDERAL
Catarinense
Campus Camboriú

Desenvolvimento Web I

Professor: Euclides Paim

euclides.paim@ifc.edu.br

INSTITUTO FEDERAL
Catarinense
Campus Camboriú

Funções
JavaScript

Professor: Euclides Paim

euclides.paim@ifc.edu.br

INSTITUTO FEDERAL
Catarinense
Campus Camboriú

Programas *JavaScript*

Sumário

Funções JavaScript:

- Definição
- Sintaxe
- Chamada de Função
- *Return* da Função
- Funções Atribuídas para Variáveis
- Funções Auto Invocadoras
- *Arrow Functions*
- Passagem de Parâmetros

Programas *JavaScript*

Funções JavaScript

- Uma função JavaScript é um bloco de código projetado para executar uma tarefa específica e é executada quando "algo" a invoca (chama).
- Uma função JavaScript é definida com a palavra-chave `function`, seguida pelo **`nome`** da função, seguido de parênteses **`()`** .
- Os nomes das funções podem conter letras, dígitos, sublinhados e cifrões (mesmas regras que as variáveis).
- Os parênteses podem incluir nomes de parâmetros separados por vírgulas: (***parâmetro1, parâmetro2, ...*** **)**
- O código a ser executado, pela função, é colocado entre colchetes: **{}**

Programas *JavaScript*

Sintaxe Funções JavaScript

```
function nome(parametro1, parametro2, parametro3) {
  // código a ser executado
}
```

- Os **parâmetros da** função estão listados entre parênteses **()** na declaração da função.
- Os **argumentos da** função são os **valores** recebidos pela função quando ela é chamada.
- Dentro da função, os argumentos (os parâmetros) se comportam como variáveis locais.

Obs.: Uma função é praticamente igual a um procedimento ou uma sub-rotina, em outras linguagens de programação.

Programas *JavaScript*

Chamadas de Função

O código dentro da função será executada quando "algo" **invoca** (chama) a função:

- Quando um **evento** ocorre (Ex. quando um usuário clica em um botão)
- Quando é **invocada** (chamada) pelo código JavaScript
- Automaticamente (auto invocada)

Obs.: Veremos mais sobre a chamada de funções posteriormente.

Programas *JavaScript*

Por que Funções?

Uma função pode ser vista como um **módulo** que contém um trecho de código que poderá ser acionado em vários pontos do programa. Os módulos servem para **organizar** o código além de facilitar a **leitura** e a **manutenção** do programa. Você pode usar o mesmo código várias vezes com argumentos diferentes, para produzir resultados diferentes. Exemplo, converter *Fahrenheit* em *Celsius*:

```
function toCelsius(fahrenheit) {
  return (5/9) * (fahrenheit-32);
}
document.getElementById("demo").innerHTML = toCelsius(88);
```

Programas *JavaScript*

Return da Função

Quando o JavaScript chega a uma instrução **`return`**, a função para de executar. Contudo, nossas funções se tornam mais úteis se retornarem um valor ao programa que a chamou. Para que isso seja possível utiliza-se a o comando **`return`** seguido do conteúdo a ser retornado.

```
var x = myFunction(4, 3);    /* Função é chamada, o valor retornado será
armazenando em x */

function myFunction(a, b) {
  return a * b;              // Função retorna o produto de a e b
}
```

O resultado será 12

INSTITUTO FEDERAL
Catarinense
Campus Camboriú

Programas *JavaScript*

Funções Atribuídas para Variáveis

A linguagem JavaScript permite que uma função seja atribuída a uma variável. Ou seja, no lugar de se atribuir uma expressão para uma variável, realizamos a atribuição de uma função. Esse formato também é conhecido como ***expressão de função***. Para chamar a função, devemos informar o nome da varável e, como nos outros formatos é possível declarar parâmetros a serem manipulados pela `function`.

```
var dobro = function(a) {
  return a * 2;
}

var num = Number(prompt("Digite um número: "));

alert("O dobro é: " + dobro(num));
```

Programas *JavaScript*

Funções Atribuídas para Variáveis

Depois que uma **expressão de função** é armazenada em uma variável, a variável pode ser usada como uma função:

```
var x = function (a, b) {return a * b};
var z = x(4, 3);
```

A função acima é na verdade uma **função anônima** (uma função sem nome). As funções armazenadas nas variáveis não precisam de nomes de funções. Eles são sempre invocadas (chamadas) usando o nome da variável.

Programas *JavaScript*

Funções Auto Invocadoras

As expressões de função podem ser "**auto invocadoras**". Uma expressão auto invocadora e iniciada automaticamente, sem ser chamada. As expressões de função serão executadas automaticamente se a expressão for seguida por **()**. Você não pode auto-invocar uma declaração de função. Você precisa adicionar parênteses ao redor da função para indicar que é uma expressão de função. Exemplo:

```
(function () {
  var x = "Hello!!";  // Vou chamar a mim mesma!!
})();
```

A função acima é na verdade uma **função auto invocadora anônima** (função sem nome).

Programas *JavaScript*

Arrow Functions

As *Arrow Functions* ou **Funções de Seta** permitem uma sintaxe curta para escrever expressões de função. Você não precisa da palavra-chave *function*, da palavra-chave *return* e dos **colchetes**.

```
// ECMAScript 5 ou ES5
var x = function(x, y) {
  return x * y;
}

// ECMAScript 6 ou ES6
const x = (x, y) => x * y;
```

Programas *JavaScript*

Arrow Functions

As funções de seta não têm suas próprias `this`. Elas não são adequadas para definir **métodos de objetos.** As funções de seta não são içadas (*hoisting*). Eles devem ser definidas **antes de** serem utilizadas.

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

Obs1.: Usar `const` é mais seguro do que usar `var`, porque uma expressão de função sempre é um valor constante, (boas práticas).

Obs2.: Você só pode omitir a palavra-chave *return* e os *colchetes* se a função tiver uma única instrução. Por isso, pode ser um bom hábito mantê-los sempre, (boas práticas).

Mais sobre a palavra-chave `this` https://www.w3schools.com/js/js_this.asp
Mais sobre *hoisting*: https://www.w3schools.com/js/js_hoisting.asp
Mais sobre *Arrow Functions:* https://www.w3schools.com/js/js_arrow_function.asp

Programas *JavaScript*

Passagem de Parâmetros

Nós já utilizamos passagem de parâmentros em exemplos anteriores. Na função/método `alert()`, deve-se passar por parâmetro a mensagem a ser exibida. **Parâmetros de** função são os **nomes** listados na declaração da função. Os **argumentos da** função são os **valores** reais passados (e recebidos pela) pela função.

```
function nome(parametro1, parametro2, parametro3) {
  // código a ser executado
}
```

Programas *JavaScript*

Passagem de Parâmetros

Regras e Padrões:

- As definições de função JavaScript não especificam tipos de dados para parâmetros.
- As funções JavaScript não executam a verificação de tipo nos argumentos passados.
- As funções JavaScript não verificam o número de argumentos recebidos.
- Se uma função é invocada com **argumentos ausentes** (menor que declarado), os valores ausentes são configurados para: **undefined.** As vezes isso é aceitável, mas às vezes é melhor atribuir um valor padrão ao parâmetro.

Exemplo:

```
// ECMAScript 6 ou ES6
function (a=1, b=1) {
  // código da função
}
```

Programas *JavaScript*

O objeto `Arguments`

As funções JavaScript possuem um objeto interno chamado objeto **`arguments`**. O objeto **`arguments`** contém uma matriz dos argumentos usados quando a função foi invocada. Dessa forma, você pode simplesmente usar uma função para encontrar (por exemplo) o valor mais alto em uma lista de números. Exemplo:

```
x = findMax(1, 123, 500, 115, 44, 88);

function findMax() {
  var i;
  var max = -Infinity;
  for (i = 0; i < arguments.length; i++) {
    if (arguments[i] > max) {
      max = arguments[i];
    }
  }
  return max;
}
```

Programas *JavaScript*

O Parâmetro Rest

A sintaxe de parâmetro **`rest`** permite representar um número indefinido de argumentos como um *array*. No exemplo, usamos parâmetros *`rest`* para coletar argumentos do segundo argumento ao último. Então os multiplicamos pelo primeiro argumento. Neste exemplo é usado uma *arrow function*, que será estudada anteriormente.

```
function multiplicar(multiplicador, ...args) {
    return args.map(x => multiplicador * x);
}
var arr = multiplicar(2, 1, 2, 3);
console.log(arr); // [2, 4, 6]
```

O método `map()`: https://www.w3schools.com/jsref/jsref_map.asp

Programas *JavaScript*

Passagem de Parâmetros

Argumentos são passados por **valor.**

Os parâmetros, em uma chamada de função, são os argumentos da função. Os argumentos JavaScript são passados por **valor**: a função apenas conhece os valores, não os locais do argumento. Se uma função altera o valor de um argumento, não altera o valor original do parâmetro.

Alterações nos argumentos não são visíveis (refletidas) fora da função.

Programas *JavaScript*

Passagem de Parâmetros

Objetos são passados por **referência.**

Em JavaScript, referências a objetos são valores. Por esse motivo, os objetos se comportarão como se fossem passados por **referência:** Se uma função altera uma propriedade do objeto, ela altera o valor original.

Alterações nas propriedades do objeto são visíveis (refletidas) fora da função.

INSTITUTO FEDERAL
Catarinense
Campus Camboriú

Sintaxe *JavaScript*

Wrap-up

- Definição
  => são modulares, facilitam a leitura e a manutenção do código...
  => `function` **`nome(parâmetro1...){}`**
- Sintaxe
  => dentro da função as variáveis se comportam como variáveis locais..
- Chamada de Função
  => código executado quando algo invoca a função (evento, chamada, auto invocadora)..
- *Return* da Função
  => função para a execução e retorna o valor definido
- Funções Atribuídas para Variáveis
  => conhecidas como expressão de função, chamar a função através do nome da variável, é uma função anônima...
- Funções Auto Invocadoras
  => executam automaticamente, devem estar entre parêntesis, e serem seguidas por **()**
- *Arrow Functions*
  => sintaxe curta, suprime as palavras-chave function entre outras, não hoisting...
- Passagem de Parâmetros
  => seguem regras e adotam padrões, argumentos são passados por valor e objetos por referência... vimos ainda o objeto ***arguments*** *e o padrão* ***rest***

INSTITUTO FEDERAL
Catarinense
Campus Camboriú

Referências

Referências Básicas

Livro NIEDERAUER, Juliano. Desenvolvendo websites com PHP: aprenda a criar websites dinâmicos e interativos com PHP e bancos de dados. 2. ed. rev. e atual. São Paulo: Novatec, 2011.

Livro SILVA, Maurício Samy. HTML 5: a linguagem de marcação que revolucionou a web. São Paulo: Novatec, 2011.

Livro SILVA, Maurício Samy. CSS3: desenvolva aplicações web profissionais com uso dos poderosos recursos de estilização das CSS3. São Paulo: Novatec, 2012.

Referências Complementares

Livro DEITEL, Paul J. Ajax,. Rich Internet applications e desenvolvimento Web para programadores. . Pearson Prentice Hall. 2009

Livro DALL'OGLIO, Pablo. PHP: programando com orientação a objetos. . Novatec. 2009

Livro SOARES, Walace. PHP 5: conceitos, programação e integração com banco de dados. . Érica. 2010

Livro SILVA, Maurício Samy. Criando sites com HTML: sites de alta qualidade com HTML e CSS. . Novatec. 2010

Livro FLANAGAN, David. o guia definitivo. . O Really. 2012

Referências na Internet

https://www.w3schools.com

https://developer.mozilla.org/pt-BR/docs/Web

https://illustrated.dev/advancedjs