JavaScript - Funções

Cristiano Pires Martins

Fonte: JavaScript - Guia do Programador Maujor

Definições

- Função é um poderoso objeto destinado a executar uma ação;
- É um bloco de código capaz de realizar ações;
- Função é um exemplo de reutilização inteligente de código;
- Tem a finalidade de dar maior legibilidade ao programa e facilitar a manutenção.

Criando Funções

- Declaração de função:

```
function minhaFuncao(){
        // aqui bloco de código
};
```

- Expressão de função:

```
var minhaFuncao = function(){
      // aqui bloco de código
};
```

- Com o uso do objeto construtor precedido da palavra-chave *new:*

```
var minhaFuncao = new Function (/*aqui bloco de código*/);
```

DECLARAÇÕES X EXPRESSÕES

- Declaração de Funções:

```
var result = add(5,5);
function add(num1, num2){
        return num1 + num2;
}
```

- Expressão de Funções:

```
var result = add(5,5);   //ERRO!!!!
var add = function(num1, num2){
        return num1 + num2;
}
```

Parâmetros

- É possível passar qualquer quantidade de parâmetros para qualquer função sem causar erros;
- Os parâmetros são armazenados em uma estrutura **semelhante** a arrays chamada ***arguments***;
- *arguments* pode receber qualquer quantidade de valores.

Exemplo 1 - parâmetros

```
function reflect(value){
    return value;
}
console.log(reflect(“Hi!”));     //“Hi!"
console.log(reflect(“Hi!”,25)); //“Hi!”
console.log(reflect.lenght);     // 1
```

Exemplo 2 - parâmetros

```
reflect = function(){
    return arguments[0];
}
console.log(reflect(“Hi!”));     //“Hi!"
console.log(reflect(“Hi!”,25)); //“Hi!”
console.log(reflect.lenght);     // 0
```

Exemplo 3 - parâmetros

```
function soma(){
    var result = 0, i = 0;
    var len = arguments.length;
    while(i < len){
      result += arguments[i];
      i++;
    }
    return result;
}
console.log(soma(1,2));          //3
console.log(soma(3,4,5,6));      //18
console.log(soma(50));           //50
console.log(soma());             //0
```

Sobrecarga de funções

```
function soma(){
    var result = 0, i = 0;
    var len = arguments.length;
    while(i < len){
      result += arguments[i];
      i++;
    }
    return result;
}
console.log(soma(1,2));          //3
console.log(soma(3,4,5,6));      //18
console.log(soma(50));           //50
console.log(soma());             //0
```

Sobrecarga de funções

- É a possibilidade de uma função ter diversas assinaturas;
- Assinatura é composta do nome da função e da quantidade e dos tipos de parâmetros esperados pela função;
- **JavaScript não possui Sobrecarga de funções.**

Resolvendo Sobrecarga em Funções

```
function mensagem(msg){
  if(arguments.length === 0)
    msg = “default”;
  console.log(msg);
}
mensagem(“Olá”);  //exibe “Olá"
mensagem();  //exibe “default”;
```

Declaração Function

```
<head>
...
<script type=”text/javascript”>
   function ola(){
      alert(“Bem-vindo ao meu site”);
   };
</script>
</head>
<body>
...
   <button type=”button” onclick=”ola();”>
   Executar função</button>
```

Declaração Function

```
<head>
...
<script type=”text/javascript”>
   function calculaRetangulo(b,h){
      var area = (b*h);
      var perimetro = (b+h)*2;
      alert(“Área: ” + area + “\nPerímetro: ” + perimetro);
   };
</script>
</head>
<body>
...
   <button type=”button” onclick=”calculaRetangulo(5,10);”>
   Executar função</button>
...
```

Function( )

```
<head>
...
<script type=”text/javascript”>
      var ola = new Function(“alert(‘Bem-vindo!’);”);
</script>
</head>
<body>
...
   <button type=”button” onclick=”ola( );”>
   Executar função</button>
...
```

Function( )

```
<head>
...
<script type=”text/javascript”>
        var calculaAreaRetangulo = new
        Function(“b”,”h”,“return b*h;”);
</script>
</head>
<body>
...
    <button type=”button”
    onclick=”alert(calculaAreaRetangulo(5,10));”>
    Executar função</button>
...
```

Sintaxe Literal

```
<head>
...
<script type=”text/javascript”>
        var calculaAreaRetangulo = function(b,h){
            return b*h;
        };
</script>
</head>
<body>
...
    <button type=”button”
    onclick=”alert(calculaAreaRetangulo(5,10));”>
    Executar função</button>
...
```

Retornando Objetos

```
<head>
...
<script type=”text/javascript”>
    function calculaRetangulo(b,h){
        var area = (b*h);
        var perimetro = (b+h)*2;
        return alert(“Área: ” + area + “\nPerímetro: ” + perimetro);
    };
</script>
</head>
<body>
...
    <button type=”button” onclick=”calculaRetangulo(5,10);”>
    Executar função</button>
...
```

Retornando Objetos

```
<head>
...
<script type=”text/javascript”>
    function calculaRetangulo(b,h){
        var area = (b*h);
        var perimetro = (b+h)*2;
        return {
            area: area,
            perimetro: perimetro
        };
    };
</script>
</head>
<body>
...
    <button type=”button” onclick=”var resultados = calculaRetangulo(8,4);
    alert(`Área:’ + resultados.area); alert(‘Perímetro: ‘ +
    resultados.perimetro);”>
    Executar função</button>
...
```

Retornando Array

```
<head>
...
<script type=”text/javascript”>
    function calculaRetangulo(b,h){
        var area = (b*h);
        var perimetro = (b+h)*2;
        return [area, perimetro];
    };
</script>
</head>
<body>
...
    <button type=”button” onclick=”var resultados = calculaRetangulo(8,4);
    alert(`Área:’ + resultados[0]); alert(‘Perímetro: ‘ + resultados[1]);”>
    Executar função</button>
...
```

Sintaxe(function f(){...})()

```
<head>
...
<script type=”text/javascript”>
    function calculaArea(b,h){
        var area = (b*h);
        return area;
    };
    alert(calculaArea(3,7));
    alert(calculaArea);
</script>
</head>
<body>
...
```

O segundo alert() mostra a função em si.

Escopo da Função

- O corpo de uma função cria um escopo local para variáveis nele declaradas com o uso da palavra-chave var.
- Os argumentos de uma função também pertencem ao escopo local.

Escopo de uma Função

```
<script type=”text/javascript”>
    function testeEscopo( ){
        var soma = 2 + 6;
        alert(“A soma é: “ + soma);       //A soma é 8
    };
    testeEscopo( );
    try{
        alert(“O dobro da soma é: ” + 2*soma);  //Resulta em soma undefined
    }
    catch(e){
        alert(e.message);         //Mostra a mensagem de erro
    }
</script>
```

Escopo de uma Função

```
<script type=”text/javascript”>
    function testeEscopo( ){
        soma = 2 + 6;
        alert(“A soma é: “ + soma);       //A soma é 8
    };
    testeEscopo( );
    try{
        alert(“O dobro da soma é: ” + 2*soma);  //Resulta em 16
    }
    catch(e){
        alert(e.message);         //Não há mensagem de erro
    }
</script>
```

Closures

- A ideia central de uma closure é exatamente a de confinamento de uma função dentro da outra.

Closures

```
<script type=”text/javascript”>
    function funcaoExterna( ){
        alert(“Função externa”);
        function funcaoInterna( ){
            alert(“Função interna”);
        };
    };
</script>
<body>
    <button type=”button” onclick=”funcaoExterna( )”>Executar função
    externa</button><br />
    <button type=”button” onclick=”funcaoInterna( )”>Executar função
    interna</button>
</body>
```

A função externa executa normalmente, mas a interna não é executada, pois foi chamada fora da função externa

Closures

```
<script type=”text/javascript”>
    function funcaoExterna( ){
        alert(“Função externa”);
        function funcaoInterna( ){
            alert(“Função interna”);
        };
        funcaoInterna();
    };

</script>
<body>
    <button type=”button” onclick=”funcaoExterna( )”>Executar função
    externa</button><br />
</body>
```

Como a função interna foi chamada dentro da função externa, é executada normalmente

Closures

```
<script type=”text/javascript”>
    function funcaoExterna( ){
        alert(“Função externa”);
        function funcaoInterna( ){
            alert(“Função interna”);
        };
        variavelGlobal = funcaoInterna;
    };

</script>
<body>
    <button type=”button” onclick=”funcaoExterna( )”>Executar função
    externa</button><br />
    <button type=”button” onclick=”variavelGlobal( )”>Executar função
    interna</button>
</body>
```

As duas funções são executadas normalmente

Funções Globais

- Funções que não estão associadas a um objeto particular da linguagem.
  - eval(código): executar um script inserido no argumento código (cuidado ao usar, pois proporciona meios de servir código malicioso);
  - isFinite(valor): testa um valor passado como argumento da função. True se for número ou false caso contrário;
  - isNaN(valor): testa um valor passado como argumento da função. True se não for um número e falso caso contrário.

Funções Globais

- Number(valor): converte em um número o valor passado como argumento da função.
- parseFloat(string [,base]): converte em um número o valor passado como argumento da função. Admite um argumento opcional que é a base na qual o número deverá ser retornado.

Exercícios

- 1. Crie uma função que retorna um Array com os meses do ano. Mostre o array retornado usando for;
- 2. Faça uma função que retorne um **objeto** com o cardápio relacionado aos dias da semana. Para cada dia da semana existe um prato diferente. Quando a função for chamada, retornará um objeto que será mostrado usando o FOR IN.