Objeto II

Objetos Nativos

Os objetos nativos são objetos que vem da própria linguagem.

A exemplo temos alguns bem conhecidos como:

```
Math
String
Boolean
Number
```

Um exemplo de utilização desses objetos é quando criamos variáveis. Geralmente utilizamos da notação literal, como demonstrado a baixo:

```
var nome = 'João';
```

Mas também podemos utilizar da maneira formal, utilizando o nosso objeto String

```
var nome = new String('João');
```

Ambos terão o mesmo valor, mas um é um tipo primitivo e outro é objeto.

A diferença que o objeto irá possuir atributos e métodos que pode ser utilizado para inúmeras situações.

Como exemplo o atributo length que retorna a quantidade de carácter em uma string.

```
var nome = new String('João')
nome.length //Retornará 4
```

Objeto do ambiente de hospedagem

Esse objeto diz respeito a onde o código está rodando, isso significa que alguns objetos podem variar dependendo do local onde o código está em execução.
Um exemplo é a “função” alert.

```
alert('Olá, mundo!').
```

Essa é uma “função” do navegador, e um servidor não faz sentido termos um alert.

Mas o alert não é especificamente uma função, ela é uma método do objeto window, que é o objeto do browser.

```
window.alert('Olá, mundo!');
```

Construtores

Um construtor é utilizado para criar uma instância de um objeto.

Por exemplo, quando utilizamos o objeto date, passamos um operador `new`, esse operador é responsável por criar uma nova instância do objeto construtor date, exemplo:

```
var date = new Date();
```

Valores e referência

Quando atribuímos o valor de uma variável a outra o que é passado é o valor, conforme exemplo abaixo:

```
var x = 0;
var y = x;
```

Já se com objetos o comportamento é um pouco diferente. Por exemplo:

```
var x = [1];

var y = x;

y.push(2);
```

No exemplo acima temos um array sendo atribuído ao x. O array no javascript é um tipo de objeto, quando passamos o valor de um objeto para uma variável, na verdade, não estamos passando o valor e sim uma referencia ao objeto original.

Sendo assim, nesse exemplo o y refere-se ao objeto de array presente no x, logo se o y sofrer alteração, essa alteração será aplicada para o objeto original também.

Função construtora

Uma função construtora é responsável por definir um modelo de métodos e propriedade que poderá ser instanciado, exemplo:

```
function Caneta(){
  this.cor = 'azul';
  this.preco = 3;
  this.mudarCor = function(cor){
      this.cor = cor;
    }
}
```

Quando criamos uma função construtora uma boa prática é deixar a primeira letra em maiúsculo.

Agora se seguindo o exemplo criamos uma variável e essa variável criar uma instancia dessa função ela herdará todas as propriedades e métodos.

```
var objetoCaneta = new Caneta();
```

Muito cuidado ao criar uma instância de um objeto. Caso o operador new seja ocultado todo o this dentro da função construtora irá representar o objeto window, logo todas as propriedades e métodos passaram a fazer parte do objeto window, o que gera um grande problema.

Mas é possível previnir esse tipo de situação, utilizando a função auto-invocável e o ‘use strict’.

```
(function(){
  'use strict'
```

```
  function Caneta(cor, preco){
    this.cor = cor;
    this.preco = preco;
  }

  var canetaVermelha = new Caneta('vermelha', 2);
})()
```

No exemplo acima caso seja omito o new, o this passa a ser undefined. Como não é possível definir propriedades e métodos para um valor undefined, um erro será retornado.

Agora geramos um outro problema. Caso queira acessar o objeto canetaVermelha verá que ele não vai está mais disponível, já que ele existia apenas de forma local na função auto invocável.

Uma forma de previnir esse comportamento é passando o objeto window para dentro da função da seguinte forma:

```
(function(window){
  'use strict'

  window.Caneta = function (cor, preco){
    this.cor = cor;
    this.preco = preco;
  }

  var canetaVermelha = new Caneta('vermelha', 2);
})(window)
```

Exercício 01

Temos três objetos, cada objeto tem duas propriedades: nome e sobrenome. Precisamos mostrar no console o nome completo de cada um deles.

Solução:

```
(function(){
        'use strict';

        var pessoa01 = new Pessoa('Vinicius', 'Sousa');
```

```
        var pessoa02 = new Pessoa('Maria', 'Silva');
        var pessoa03 = new Pessoa('Godofredo', 'Alves');

        pessoa01.exibirNome();
        pessoa02.exibirNome();
        pessoa03.exibirNome();

        console.log(pessoa01);


        function Pessoa(nome, sobrenome){
            this.nome = nome;
            this.sobrenome = sobrenome;
            this.exibirNome = function(){
                console.log(this.nome + ' ' + this.sobrenome);
            }
        }
    })()
```

Exercício 02

Refatorar o código anterior de maneira que os três objetos fiquem em um array.

Resolução:

```
        'use strict';

        var pessoas = [];

        gerarPessoas(pessoas, 'Vinicius', 'Sousa');
        gerarPessoas(pessoas, 'Joãozinho', 'Silva');
        gerarPessoas(pessoas, 'Maria', 'Dorvalina');

        exibirPessoas(pessoas);

       function exibirPessoas(arr){
            for(var i = 0; i < arr.length; i++){
                console.log(arr[i].nome + ' ' + arr[i].sobrenome);
            }
       }

        function gerarPessoas(arr, nome, sobrenome){
            var pessoa = new Pessoa(nome, sobrenome);
            arr.push(pessoa);
        }

        function Pessoa(nome, sobrenome){
            this.nome = nome;
            this.sobrenome = sobrenome;
```

```
        }
    })()
```