**Organize seu código Javascript de maneira fácil!**

Postado dia 10/03/2014 por Leonardo Souza em [Front-end](https://blog.caelum.com.br/category/front-end/) [49](https://blog.caelum.com.br/organize-seu-codigo-javascript-de-maneira-facil/#comments)

[](https://i2.wp.com/blog.caelum.com.br/wp-content/uploads/2014/02/js.jpg?ssl=1)

Ao longo desses anos nesta indústria vital já participei de inúmeros projetos… e foram tantos que os dedos das mãos seriam insuficientes para contá-los. Alguns deles me orgulho, outros nem tanto (afinal como você, eu também já produzi muita coisa de qualidade duvidosa). Ainda assim acredito na máxima que as pessoas sempre evoluem, uma vez que estas buscam sempre o melhor dentro da missão de cada uma.

Organizar código Javascript para quem está iniciando nem sempre é fácil, pois na ótica do programador novato a linguagem é o mesmo que um monte de blocos de código (leia funções) que este foi criando ou até mesmo copiando para resolver os problemas do dia-a-dia. Esse pensamento, somado a inúmeras más práticas vão tornando o código feito com Javascript um tanto que caótico.

Entre as principais (más práticas), eis algumas que a meu ver atrapalham e irritam grande parte dos membros de um time de desenvolvimento (ah e faz a mãe do programador, ser lembrada):

* falta de indentação
* falta de comentários
* variáveis espalhadas
* variáveis globais
* funções e variáveis criadas sem propósito

Mas o objetivo deste artigo não é só citar as partes ruins do desenvolvimento, e sim propor uma forma elegante de resolver isso. Então vamos a uma solução simples para organização dos seus códigos feitos com JavaScript.

**Funções nomeadas x funções anônimas**

Todo mundo aqui, deve saber o que são funções **nomeadas** e **anônimas** em JavaScript. Se não sabe, eis alguns exemplos:

|  |
| --- |
| // função nomeada  function common() {      return true;  } |

Acima vemos uma **função nomeada** e, a seguir, uma **função anônima**:

|  |
| --- |
| // função anônima  var anonymous = function() {   return true;  } |

Observando os dois códigos, vê-se pouca diferença, não é mesmo? Os dois quando executados retornam a mesma coisa (true), porém são padrões sintáticos bem distintos.

No primeiro, temos uma função definida com o nome **common**, e é através desse nome que será possível invocar essa função, desse jeito:

|  |
| --- |
| common();  // => true |

No segundo, note que a função não possui um nome, então ela precisou ser atribuída a variável **anonymous** para que pudesse ser invocada. Neste caso a chamada seria idêntica ao exemplo anterior:

|  |
| --- |
| anonymous();  // => true |

Entendeu o conceito? Em linhas gerais, desculpe a ambiguidade, a **função nomeada** chama-se assim porque **tem um nome**. A **função anônima**, por sua vez **não possui um nome**, então a única forma de utilizá-la é atribuindo-a algo (uma variável, uma propriedade em um objeto, etc). Perdi esse tempo, explicando isso, pois acho que vai valer a pena para você entender o que vamos fazer a seguir.

**Começando a organização**

Vamos iniciar os trabalhos, **criando uma função anônima**:

|  |
| --- |
| function() {} |

Se você salvar esse código e tentar executá-lo em sua página, ou mesmo rodá-lo no console do seu navegador tomará um ***SyntaxError***. E isso tem um motivo óbvio, pois se a função não tem nome como eu vou chamá-la? Então teoricamente eu seria obrigado a atribui-la a uma variável ou algo do tipo. Neste caso, vamos fazer diferente:

|  |
| --- |
| (function() {}) |

Colocando esse código entre parênteses, o ***SyntaxError*** que estava rolando até então sumirá. Mágica? Magia negra? Não! Simplesmente os parênteses agruparam a nossa função, formando uma espécie de expressão. Esse código servirá de embrulho (**wrapper**) para tudo que vamos fazer daqui pra frente. Farei um pequeno teste agora, colocando um simpático console.log dentro da nossa função e, se tudo der certo, o resultado será a exibição do texto **“less is more!”**:

|  |
| --- |
| (function() {   console.log("less is more!");  }) |

Se você executou esse código na sua página, e deu uma olhadinha no console, vai perceber que não houve retorno algum. Ué, o que que houve? Ocorre que apesar de termos uma função que implementa uma única linha de código, ele só está definida, porém não foi executada. Para fazer isso, basta adicionar parênteses no final da expressão, desta forma:

|  |
| --- |
| (function() {   console.log("less is more!");  })(); |

Bingo! A partir de agora, toda vez que a página for carregada, esse função será executada “automagicamente”! Então temos a base necessária para organização de nossos códigos.

**Organizando seu código**

Você já deve ter ouvido falar, que toda variável precisa ser declarada usando a palavra chave **var**. Não usar o **var**, custa caro, pois uma variável definida sem ele vai parar no **escopo global**, então para seguir a régua de boas práticas vamos sempre usá-lo.

|  |
| --- |
| (function() {      console.log("less is more!");        var box = {};    })(); |

No exemplo acima, defini uma variável **box** que armazena um **objeto vazio**. A grande vantagem dessa abordagem é que **box** não está disponível em nenhum outro escopo a não ser o definido pela função **wrapper** que fizemos. Então se algum espertinho tentar ir no console e digitar:

|  |
| --- |
| console.log(box);  // => ReferenceError: box is not defined |

Vai tomar um belo de um ***ReferenceError*** no meio da cara! Bom né? Não sujamos o escopo global e ainda ganhamos privacidade!

Uma vez que temos o objeto **box**, podemos adicionar propriedades e métodos a este objeto. Imagine agora que precisamos criar um controle bem simples, para organizar uma fila com nomes de automóveis. A ideia é começar com a fila vazia, ir adicionando os veículos e retornar a lista no final. Para isso, vamos usar uma propriedade chamada **queue** e dois métodos, que vamos chamar de **addItem** e **getQueue**:

|  |
| --- |
| (function() {      console.log("less is more!");        // criando o objeto (vazio) box      var box = {};        // adicionando a propriedade queue (fila)      box.queue = [];        // adicionando o métodos addItem (adicionar item)      box.addItem = function() {        };        // adicionando o métodos getQueue (recuperar fila)      box.getQueue = function() {        };  })(); |

Como podemos perceber, a propriedade **queue** é um array vazio. Então nosso método **addItem** precisa receber um veículo como parâmetro e adicioná-lo nesta fila. Por sua vez, o método **getQueue** apenas retorna uma string com os veículos que temos na fila, separados por um hífen.

|  |
| --- |
| (function() {      console.log("less is more!");        // criando o objeto (vazio) box      var box = {};        // adicionando a propriedade queue (fila)      box.queue = [];        // adicionando o métodos addItem (adicionar item)      box.addItem = function(car) {          return box.queue.push(car);      };        // adicionando o métodos getQueue (recuperar fila)      box.getQueue = function() {          return box.queue.join(" - ");      };  })(); |

Feito isso, se você rodar esse código, deve estar se perguntando: como diabos vou invocar os métodos **addItem** e **getQueue** do objeto **box** que acabei de implementar, visto que ele não está acessível no escopo global? A resposta é simples, basta fazer com que nossa função **wrapper** retorne o objeto **box** (forma simples) ou podemos configurar exatamente o que vai ser retornado. Vamos ao primeiro exemplo:

|  |
| --- |
| var GLOBALCAR = (function() {      console.log("less is more!");        // criando o objeto (vazio) box      var box = {};        // adicionando a propriedade queue (fila)      box.queue = [];        // adicionando o métodos addItem (adicionar item)      box.addItem = function(car) {          return box.queue.push(car);      };        // adicionando o métodos getQueue (recuperar fila)      box.getQueue = function() {          return box.queue.join(" - ");      };        return box;  })(); |

Como você deve ter percebido, além de retornar o objeto **box**, eu atribui o nosso **wrapper** a uma variável global chamada **GLOBALCAR** (mesmo com a palavra chave var, como **GLOBALCAR** não está dentro de uma função, ela fica visível no escopo mais amplo do Javascript, podendo ser acessada de qualquer local). Dessa forma, **GLOBALCAR** tornou-se um objeto. Para testar nossa implementação podemos executar (eu incentivo você neste momento a testar via console):

|  |
| --- |
| GLOBALCAR;  // => Object {queue: Array[0], addItem: function, getQueue: function} |

Viu isso? **GLOBALCAR** é um objeto contendo **queue**, **addItem** e **getQueue**. Então podemos executar o método **addItem** para adicionar os carros a nossa fila:

|  |
| --- |
| GLOBALCAR.addItem("Gol");  // => 1  GLOBALCAR.addItem("Palio");  // => 2  GLOBALCAR.addItem("Corsa");  // => 3 |

E o método **getQueue** para retornar os itens que estão na fila:

|  |
| --- |
| GLOBALCAR.getQueue();  // => "Gol - Palio - Corsa" |

Legal né? Mas ai se você souber um pouquinho mais de JavaScript vai se perguntar porque é possível acessar a propriedade **queue** diretamente!? Isso tem tudo a ver com a forma que retornamos o objeto **box**. Se você quiser esconder essa propriedade, deixando disponível apenas os métodos você pode fazer isso:

|  |
| --- |
| var GLOBALCAR = (function() {      console.log("less is more!");        // criando o objeto (vazio) box      var box = {};        // adicionando a propriedade queue (fila)      box.queue = [];        // adicionando o métodos addItem (adicionar item)      box.addItem = function(car) {          return box.queue.push(car);      };        // adicionando o métodos getQueue (recuperar fila)      box.getQueue = function() {          return box.queue.join(" - ");      };        // retornando um objeto personalizado (só com o necessário)      return {          add: box.addItem,          get: box.getQueue      };  })(); |

Agora um novo teste no console revelará:

|  |
| --- |
| GLOBALCAR;  // => Object {add: function, get: function} |

Note que ninguém, a não ser você saberá da existência da propriedade **queue**. O mesmo vale para outros métodos que você queira implementar, mais que não fazem sentido serem expostos.

Com base na nova implementação, o método **addItem** virou apenas **add**, e o **getQueue** virou um simples **get**. Para adicionar itens a nossa fila, usaremos então:

|  |
| --- |
| GLOBALCAR.add("Gol");  // => 1  GLOBALCAR.add("Palio");  // => 2  GLOBALCAR.add("Corsa");  // => 3 |

E para retornar a fila:

|  |
| --- |
| GLOBALCAR.get();  // => "Gol - Palio - Corsa" |

Elegante não? Agora você já tem uma base sólida para começar a organizar os seus projetos em JavaScript. O mais bacana, é que você pode entender a timeline do processo para chegarmos nesse objetivo. Aproveite o momento de satisfação, e divulge para a galera que agora você aprendeu a usar um **pattern** para organização do seu código, conhecido por aí como “**Module Pattern**“. Chique não?

No nosso [curso de Programação JavaScript da Caelum](http://www.caelum.com.br/curso-javascript-jquery/), vemos ainda mais boas práticas de organização de JavaScript.