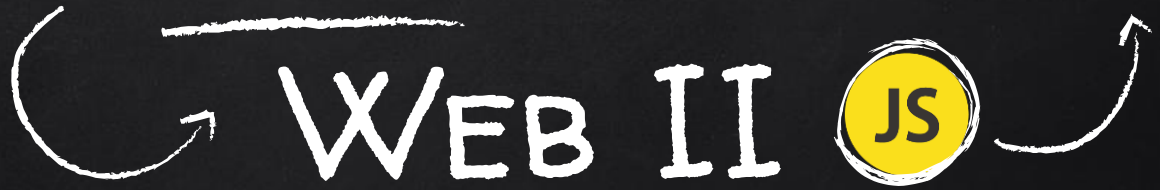
Instituto Federal de São Paulo

Análise e Desenvolvimento de Sistema

EDUCAÇÃO PÚBLICA 100% GRATUITA

Desenvolvimento Web II



Professor Johnata Santicioli

Sumários

- ✗ Case Styles
- ✗ Naming Convention
- ✗ Coding Conventions
- ✗ Boas e más práticas de desenvolvimento em JS
- ✗ Linters
- ✗ Code Smells

Case Style

- ✗ Camel Case
  - camelCase
  - nomeAluno
- ✗ Pascal Case
  - PascalCase
  - NomeAluno
- ✗ Snake Case
  - snake_case
  - nome_aluno
- ✗ Screaming Snake Case
  - SCMG_SNAKE_CASE
  - NOME_ALUNO
- ✗ Kebab Case
  - kebab-case
  - nome-aluno
- ✗ Screaming Kebab Case
  - SCMG-KEBAB-CASE
  - NOME-ALUNO
- ✗ Lowercase
  - lowercase
  - nomealuno
- ✗ Upercase
  - UPERCASE
  - NOMEALUNO
- ✗ Hungarian
  - strFirstName
  - arrUserNames

Naming Convention

✗ Camel Case

```
//variáveis e constantes
var nomeAluno
const pi

//métodos e funções
getElementById()
function calculaIdade()
```

✗ Pascal Case

```
//classes e data types
RegExp
DOMObject
class FolhaDePagamento
```

Coding Conventions

✗ Sintaxe expandida

```
if(dayOfWeek===7&&weather==='sunny'){
  goOnTrip('beach','car',['ice cream','bucket and spade','beach towel']);
}
```

```
if(dayOfWeek === 7 && weather === 'sunny') {
  goOnTrip('beach', 'car', ['ice cream', 'bucket and spade', 'beach towel']);
}
```

Coding Conventions

✗ Operador ternário

```
let status = (age >= 18)
  ? 'adult'
  : 'minor';
```

```
let status = (age >= 18) ? 'adult' : 'minor';
```

Coding Conventions

- ✗ Sempre comente seu código
- ✗ Definição de variáveis
  - usar let e const (evitar usar var por causa do hoisting)
  - usar principalmente const (conceito de imutabilidade)
  - usar vetor e objeto com as variáveis

Coding Conventions

✗ Use sempre o operador estrito

```
name == 'Chris';
age != 25;
```

```
name === 'Chris';
age !== 25;
```

✗ Utilize template string literals

```
let myName = 'Chris';
console.log('Hi! I\'m' + myName + '!');
```

```
let myName = 'Chris';
console.log(`Hi! I'm ${myName}!`);
```

Boas e más práticas

✗ Sempre utilize de chaves

```
if(someVariableExists)
 x = false
 anotherFunctionCall();
```

```
if(someVariableExists) {
 x = false;
 anotherFunctionCall();
}
```

```
if(someVariableExists) {
 x = false;
 anotherFunctionCall();
}
```

```
if(2 + 2 === 4) return 'nicely done';
```

Boas e más práticas

✗ Adicione os scripts na parte final da sua página

```
<p>E, agora, você sabe os meus tipos favoritos de milho. </p>
<script type="text/javascript" src="path/to/file.js"></script>
<script type="text/javascript" src="path/to/anotherFile.js"></script>
</body>
</html>
```

Boas e más práticas

✗ Declare variáveis fora de comandos

```
for(var i = 0; i < someArray.length; i++) {
 var container = document.getElementById('container');
 container.innerHtml += 'my number: ' + i;
 console.log(i);
}
```

```
var container = document.getElementById('container');
for(var i = 0, len = someArray.length; i < len;  i++) {
 container.innerHtml += 'my number: ' + i;
 console.log(i);
}
```

Boas e más práticas

✗ Usar métodos nativos independente do que esteja acontecendo por trás da camada de abstração, é, geralmente, muito mais rápido que os métodos alternativos, não nativos.

```
1  var arr = ['item 1', 'item 2', 'item 3', ...];
2  var list = '<ul><li>' + arr.join('</li><li>') + '</li></ul>';
```

Boas e más práticas

✗ Ao reduzir a quantidade de variáveis globais a um único nome, você reduz, significantemente, a chance de más interações com outras aplicações, widgets ou biblioteca.

```
1  var name = 'Jeffrey';
2  var lastName = 'Way';
3
4  function doSomething() {...}
```

```
1  var DudeNameSpace = {
2   name : 'Jeffrey',
3   lastName : 'Way',
4   doSomething : function() {...}
5  }
```

Boas e más práticas

✗ Não passe uma cadeia de caracteres como parâmetros para as funções setInterval ou setTimeOut, utilize funções nomeadas.

```
setInterval(
"document.getElementById('container').innerHTML += 'My new number: ' + i", 3000
);
```

```
setInterval(someFunction, 3000);
```

Boas e más práticas

- ✗ Utilize chaves e colchetes ({ } e [ ]) ao invés de Object() e Array() para criar objetos e vetores.

```
var o = new Object();
o.name = 'Jeffrey';
o.lastName = 'Way';
o.someFunction = function() {
 console.log(this.name);
}
```

```
var o = {
 name: 'Jeffrey',
 lastName = 'Way',
 someFunction : function() {
    console.log(this.name);
 }
};
```

- ✗ Um erro comum em programas JavaScript é usar um objeto no momento que um vetor é requerido, ou o contrário. A regra é simples: quando os nomes das propriedades são pequenos números inteiros sequenciais, você deve usar um vetor. Caso contrário, use um objeto.

Boas e más práticas

✗ Lista longa de variáveis, separe-as por vírgula.

```
1  var someItem = 'some string';
2  var anotherItem = 'another string';
3  var oneMoreItem = 'one more string';
```

```
1  var someItem = 'some string',
2    anotherItem = 'another string',
3    oneMoreItem = 'one more string';
```

Boas e más práticas

- ✗ Otimize seu código para que seja executado de forma mais rápida, utilize a função time para testar.

```
1  function TimeTracker(){
2   console.time("MyTimer");
3   for(x=5000; x > 0; x--){}
4   console.timeEnd("MyTimer");
5  }
```

https://developer.mozilla.org/en-US/docs/MDN/Guidelines/Code_guidelines/JavaScript

Boas e más práticas

- ✗ Linters
  - Lint é um substantivo que podemos traduzir como “Fiapo”
  - O termo "lint" surgiu em programação da necessidade de implementar algum tipo de checagem automática pra previnir e/ou solucionar erros enquanto escrevemos códigos.
  - Linters JavaScript: ESLint, JSLint, JSHint e Lynt (node e npm)

Boas e más práticas

✗ Code Smells

- Termo criado por Kent Beck quando ele ajudava Martin Fowler a escrever seu best seller Refatoração – Aperfeiçoando o Projeto de Código Existente (Refactoring)
- O termo smell, que pode ser traduzido livremente para cheiro, foi escolhido porque um odor é algo perceptível
- De igual forma, possíveis problemas no código de um programa podem ser percebidos com facilidade.

Boas e más práticas

- ✗ Code Smells
  - Código muito grande (Bloaters)
  - Violação a orientação a objetos (Object-Orientation Abusers)
  - Inibidores de modificação (Change Preventers)
    - Reescrever para tornar o código modular e de possível evolução
  - Código dispensável (Dispensables)
    - Comentário excessivos
  - Acopladores (Couplers)
    - Classes que dependem de outras classes

Referências

Você poderá estudar mais sobre o tema nas seguintes páginas:

https://developer.mozilla.org/en-US/docs/MDN/Guidelines/Code_guidelines/JavaScript

https://www.youtube.com/watch?v=HBPz6h1ck9g

https://devopedia.org/naming-conventions

https://www.devmedia.com.br/code-smells-conheca-antes-que-seja-tarde/39636

Refatoração – Aperfeiçoando o Projeto de Código Existente (Bookman, 2004)

THANKS!

Até a próxima aula!!!

www.ifsp.edu.br
johnata.santicioli@ifsp.edu.br