

Valderi Leithardt, Dr.

IPW (Class notes for 3 weeks)

Summary

- Review of previous class *
- The Secret Life of Objects
- Method, Prototypes, Builders
- Node.js

* Asynchronous Programming

- Exercises

Objects (*Review previous class)

- Diferente de uma variável, um objeto pode conter diversos valores e/de tipos diferentes armazenados nele **(atributos)** e também possuir funções que operem sobre esses valores **(métodos)**. Tanto os atributos, quanto os métodos, são chamados de propriedades do objeto.

→ Criando objetos usando a sintaxe literal de objeto

```
const pessoa = {  
  nome: 'testeNome',  
  apelido: 'testeApelido'  
};
```

→ Usando a palavra-chave 'new' com a função construtora Object integrada

```
const pessoa = new Object();
```

Functions (*Review previous class)

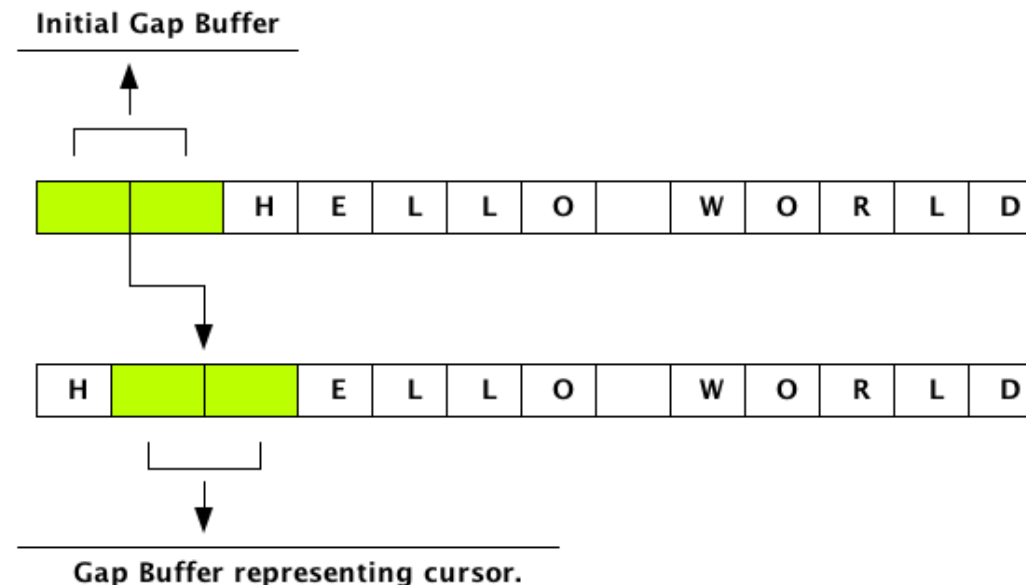
Existem cinco tipos de definições de funções:

- Functions declaration (Função de declaração)
- Functions expression (Função de expressão)
- Arrow Functions (Função de flecha)
- Functions constructor (Função construtora)
- Generator Functions (Função gerador)

--» As definições mais comuns e similares: declaration e expression.

Arrays (*Review previous class)

- Array é uma estrutura de dados, todos do mesmo tipo. Vetores (1D) e Matrizes (2D, 3D, ...) são exemplos de arrays.
- Arrays são geralmente descritas como "lista de objetos"; elas são basicamente objetos que contem múltiplos valores armazenados em uma lista.
- Um objeto array pode ser armazenado em variáveis e ser tratado de forma muito similar a qualquer outro tipo de valor, a diferença está em podermos acessar cada valor dentro da lista individualmente.



higher order function (*Review previous class)

Funções de alta ordem

Funções que funcionam em conjunto com outras funções, seja ela apenas devolvem argumentos, são chamadas de funções de ordem superior.

[* Conforme descrito em: eloquent-javascript/chapters/05](https://eloquent-javascript.com/chapters/05)

O termo vem da matemática onde a distinção entre funções e outros valores é levado mais a sério.

Funções de ordem superior nos permitem abstrair as ações. Elas podem ser de diversas formas. Por exemplo: pode-se ter funções que geram novas funções.

Exemplo de função que recebe função como parâmetro, executando a função 3 vezes, cada vez passando um número:

```
function executaDe1a3(funcao) {  
    funcao(1);  
    funcao(2);  
    funcao(3);  
}  
  
executaDe1a3(function (x) { console.log('IPW turma '  
+ x); });
```

The Secret Life of Objects

A maioria das histórias de programação, começa com um problema de complexidade. A teoria é de que a complexidade pode ser administrada separando-a em pequenos compartimentos isolados um do outro. Esses compartimentos acabaram ganhando o nome de objetos.

De acordo com https://eloquentjavascript.net/06_object.html

Um objeto é um escudo duro que esconde a complexidade grudenta dentro dele e nos apresenta pequenos conectores (como métodos) que apresentam uma interface para utilizarmos o objeto. A ideia é que a interface seja relativamente simples e toda as coisas complexas que vão dentro do objeto possam ser ignoradas enquanto se trabalha com ele.

No ECMAScript 2015 foi introduzida uma sintaxe reduzida para definição de métodos em inicializadores de objetos. É uma abreviação para uma função atribuída ao nome do método. [Material adaptado com base em MDN](#)

Métodos são propriedades simples que comportam valores de funções. Isso é um método simples:

```
var coelho = {};  
coelho.diz = function(linha) {  
  console.log("O coelho diz '" + linha + "'");  
};  
  
coelho.diz("Estou vivo.");  
// → O coelho diz 'Estou vivo.'
```

Definição de Método

No ECMAScript 2015 foi introduzida uma sintaxe reduzida para definição de métodos em inicializadores de objetos. É uma abreviação para uma função atribuída ao nome do método.

10 métodos da classe array do javascript muito utilizados na construção de aplicações.

1. map()

Este método cria um novo array e executa uma função sobre cada um dos items do array fonte.

```
const arr = [2,4,6,8];  
const newArr = arr.map(num => num * 2);  
console.log(newArr) // saída: [4,8,12,16]
```

2. reduce()

Este método cria um acumulador ou reduz o array a um valor único utilizando uma função que cria um valor único.

```
const arr = [2,4,6,8];  
const soma = arr.reduce((total, value) => total + value, 0);  
console.log(soma); // saída: 20
```


Definição de Método

3. some()

Este método testa se pelo menos um dos elementos **do** array atende a condição passada, se for verdade devolve **true** se for falso devolve **false**

```
const arr = [2,4,6,8];  
const rest = arr.some(num => num > 4);  
console.log(rest); // saída: true
```

4. every()

Este método testa se todos os elementos **do** array atendem a condição passada, se for verdade devolve **true** se for falso devolve **false**

```
const arr = [2,4,6,8];  
const maiorQue = arr.every(num => num > 4);  
console.log(maiorQue); // Saída: false
```

Definição de Método

5. forEach()

Este método faz um loop sobre cada um dos itens **do** array.

```
const arr = [2,4,6,8];  
arr.forEach(num => {  
  console.log(num); saída: 2 4 6 8  
});
```

6. includes()

Este método checa se o array inclui em seus itens o valor passado como parametro.

```
const arr = [2,4,6,8];  
console.log(arr.includes(6)); // saída: true  
console.log(arr.includes(9)); // saída: false
```

Definição de Método

7. filter()

O método filter cria um novo array com os elementos que passaram na condição da função que foi data como parametro para o método.

```
const words = ['spray', 'limit', 'elite', 'exuberant', 'destruction', 'present'];
```

```
const result = words.filter((word) => word.length > 6);
```

```
console.log(result);
```

```
// Expected output: Array ["exuberant", "destruction", "present"]
```

8. sort()

Este método é utilizado para arranjar ou ordenar os itens de um array de maneira ascendente ou descendente.

```
const arr = [2,4,6,8];
```

```
const alpha = ['d', 'c', 'b', 'a'];
```

```
const descendente = arr.sort((a, b) => a > b ? -1 : 1);
```

```
console.log(descendente);
```

```
const ascendente = alpha.sort((a, b) => a > b ? 1 : -1);
```

```
console.log(ascendente);
```

Definição de Método

9. Array.from()

Gera um array a partir de uma constante ou variável não array, como por exemplo uma string.

```
const title = 'Valderi';  
const novo = Array.from(title);  
console.log(novo); // saída: ['V', 'a', 'l', 'd', 'e', 'r', 'i']
```

10. Array.of()

Este método gera um array a partir dos parâmetros passados para o método.

```
const nums = Array.of(2,4,6,8);  
console.log(nums); // saída: [2,4,6,8]
```

Definição de Protótipo

- Em JavaScript um objeto é uma coleção de propriedades, sendo cada propriedade definida como uma sintaxe de par chave : valor. A chave pode ser uma string e o valor pode ser qualquer informação.
- Cada objeto em **JavaScript** tem uma referência interna a outro objeto, chamado de “**protótipo**”. Rever conceitos sobre objetos, [tutorial JS*](#)
- Quando uma propriedade ou método é acessado em um objeto, o **JavaScript** primeiro verifica se essa propriedade ou método existe no próprio objeto.
- Se não for encontrado, o **JavaScript** procurará no **protótipo** do objeto.
- Protótipos são os mecanismos pelo qual objetos JavaScript herdam recursos uns dos outros.

Protótipo

Exemplo utilizando protótipo
Calculadora.prototype para definir
os métodos adicionar, subtrair e
obterResultado. Esses métodos
podem ser utilizados por qualquer
objeto criado a partir do
construtor Calculadora.

Outros exemplos:

<https://www.javascripttutorial.net/javascript-prototype/>

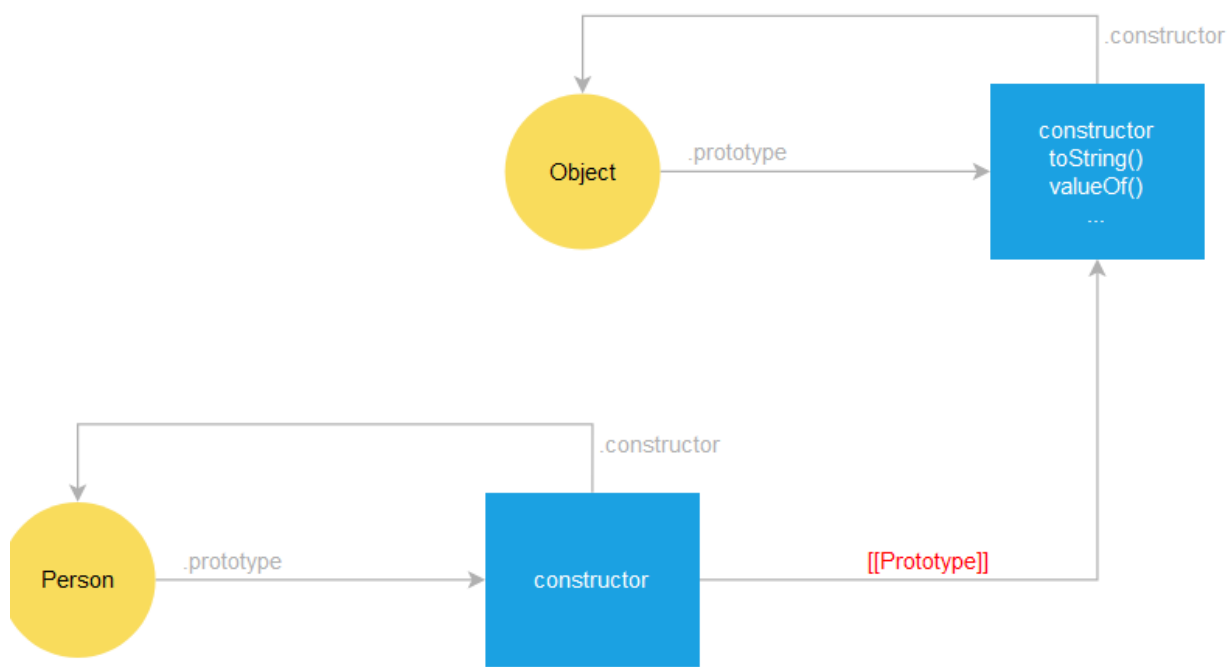
https://www.w3schools.com/js/js_object_prototypes.asp

```
function Calculadora() {  
    this.valor = 0;  
}  
  
Calculadora.prototype.adicionar = function(valor) {  
    this.valor += valor;  
};  
  
Calculadora.prototype.subtrair = function(valor) {  
    this.valor -= valor;  
};  
  
Calculadora.prototype.obterResultado = function() {  
    return this.valor;  
};  
  
const minhaCalculadora = new Calculadora();  
minhaCalculadora.adicionar(5);  
minhaCalculadora.subtrair(3);  
console.log(minhaCalculadora.obterResultado()); // 2
```

Construtores

- Em JavaScript, chamar uma função precedida pela palavra-chave **new** vai fazer com que ela seja tratada como um construtor.
- O construtor terá sua variável **this** relacionada a um novo objeto, e a menos que explicitamente o retorno do valor de outro objeto, esse novo objeto será retornado a partir da chamada.
- Um objeto criado com **new** é chamado de instância do construtor.

*Material adaptado conforme descrito em [livro PT](#)



```
function Coelho(tipo) {  
  this.tipo = tipo;  
}
```

```
var coelhoAssassino = new Coelho("assassino");  
var coelhoPreto = new Coelho("preto");  
console.log(coelhoPreto.tipo);  
// → preto
```

**Outros exemplos em [tutoriais JS](#)

Construtores

Classes

- Em JavaScript são introduzidas no ECMAScript 2015 e tratadas na sintaxe [moderna ECMA](#), são simplificações da linguagem para as heranças baseadas nos protótipos.
 - A sintaxe para classes não introduz um novo modelo de herança de orientação a objetos em JavaScript.
- Classes em JavaScript provêm uma maneira mais simples e clara de criar objetos e lidar com herança.

Herança

- É quando uma nova classe é criada e herda as propriedades da classe já existente.
- Possui suporte ao conceito de reutilização de código e reduz o comprimento do código na poo.

Polimorfismo

- Polimorfismo é aquele em que podemos realizar uma tarefa de várias formas ou maneiras e é aplicado às funções ou métodos.
- O polimorfismo permite que o objeto decida qual forma da função implementar em tempo de compilação e também em tempo de execução.

Os tipos de polimorfismo são:

- ✓ Polimorfismo de tempo de compilação (sobrecarga de método)
- ✓ Polimorfismo de tempo de execução (substituição de método)

Material adaptado conforme [MDN](#) e [Livro PT](#) e [W3S](#)

Node.js

- O Node.js é uma ferramenta que nos permite executar códigos escritos na linguagem JavaScript no servidor.
- O JavaScript é interpretado por uma engine que a princípio só era encontrado nos navegadores de páginas web. É através desse motor que o navegador web vai interpretar o código escrito na linguagem JavaScript. Por este motivo o JavaScript era uma linguagem web apenas.
- O Node.js utiliza o motor V8 do Google Chrome, ou seja a mesma ferramenta utilizada pelo navegador do Google para executar códigos JavaScript. Pelo fato do Node.js conseguir interpretar códigos JavaScript fora do navegador, o JavaScript deixou de ser apenas uma linguagem web e passou a ser também uma linguagem back-end (servidor) e mobile.

Node.js

- ✓ Interpreta códigos JavaScript fora do navegador.
- ✓ Nos permite criar códigos Back-end utilizando JavaScript.
- ✓ Nos permite criar códigos para dispositivos móveis utilizando JavaScript.

Node.js

Walmart 

ebay

PayPal®

DOW JONES


intuit.

NETFLIX

Linked 

The New York Times

 Microsoft

 U B E R

YAHOO!

King  sher

Fonte: <http://www.roweb.ro/blog/5-great-use-cases-for-node-js/>

Node.js



Fonte: <https://strongloop.com/node-js/why-node/>

Node.JS

Tutorial NODE.JS para iniciantes

<https://learn.microsoft.com/pt-br/windows/dev-environment/javascript/nodejs-beginners-tutorial>

Tutorial e documentação

<https://www.tutorialspoint.com/nodejs/index.htm>

https://www.tutorialspoint.com/nodejs/nodejs_tutorial.pdf

Tutorial W3S

<https://www.w3schools.com/nodejs/>

JSON

JavaScript Object Notation

JSON (JavaScript Object Notation)

É um modelo para armazenamento e transmissão de informações no formato texto.

Apesar de muito simples, tem sido **muito utilizado por aplicações Web** devido a sua capacidade de estruturar informações de uma forma bem mais compacta do que a conseguida pelo modelo XML, tornando mais rápido o parsing dessas informações.

Isto explica o fato de o JSON ter sido adotado por empresas como Google e Yahoo, cujas aplicações precisam transmitir grandes volumes de dados.

JSON

JavaScript Object Notation

JSON x XML

Podemos entender o JSON como uma espécie de “concorrente” da XML na área de troca de informações. Nesta seção, apresentamos algumas das principais semelhanças e diferenças entre os dois modelos para a representação de informações.

Semelhanças:

Os dois modelos representam informações no formato texto.

Ambos possuem natureza auto-descritiva (ou seja, basta “bater o olho” em um arquivo JSON ou em um arquivo XML para entender o seu significado).

Ambos são capazes de representar informação complexa, difícil de representar no formato tabular.

Alguns exemplos: objetos compostos (objetos dentro de objetos), relações de hierarquia, atributos multivalorados, arrays, dados ausentes, etc.

Ambos podem ser utilizados para transportar informações em aplicações AJAX.

Ambos podem ser considerados padrões para representação de dados. XML é um padrão W3C, enquanto JSON foi formalizado na RFC 4627.

Ambos são independentes de linguagem. Dados representados em XML e JSON podem ser acessados por qualquer linguagem de programação, através de API's específicas.

JSON

JavaScript Object Notation

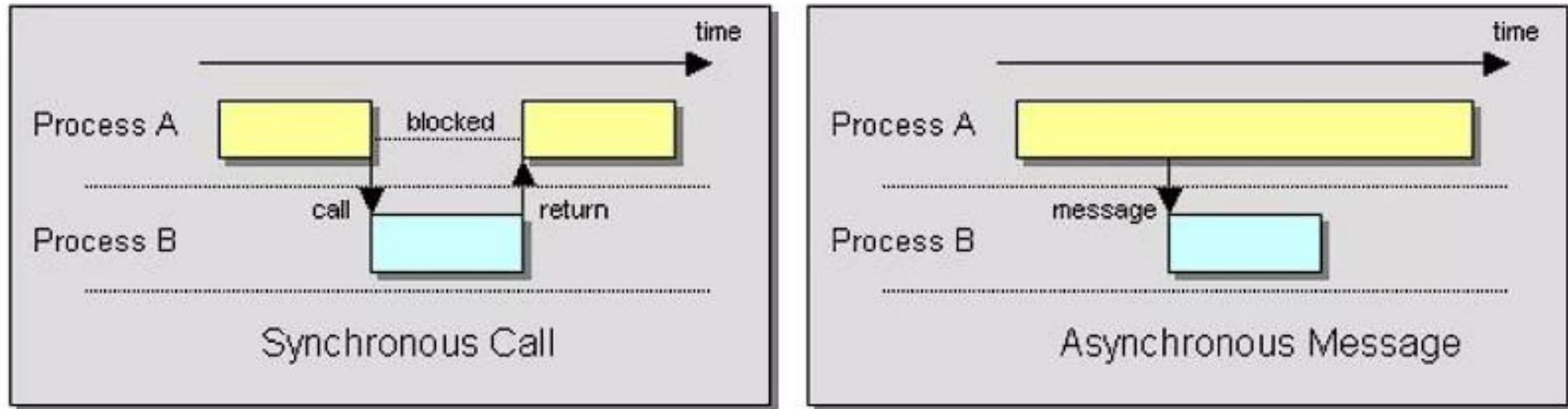
Diferenças:

- JSON não é uma linguagem de marcação. Não possui tag de abertura e muito menos de fechamento!
- JSON representa as informações de forma mais compacta.
- JSON não permite a execução de instruções de processamento, algo possível em XML.
- JSON é tipicamente destinado para a troca de informações, enquanto XML possui mais aplicações.
- Por exemplo: nos dias atuais existem bancos de dados inteiros armazenados em XML e estruturados em SGBD's XML nativo.

Programação assíncrona (cont)...

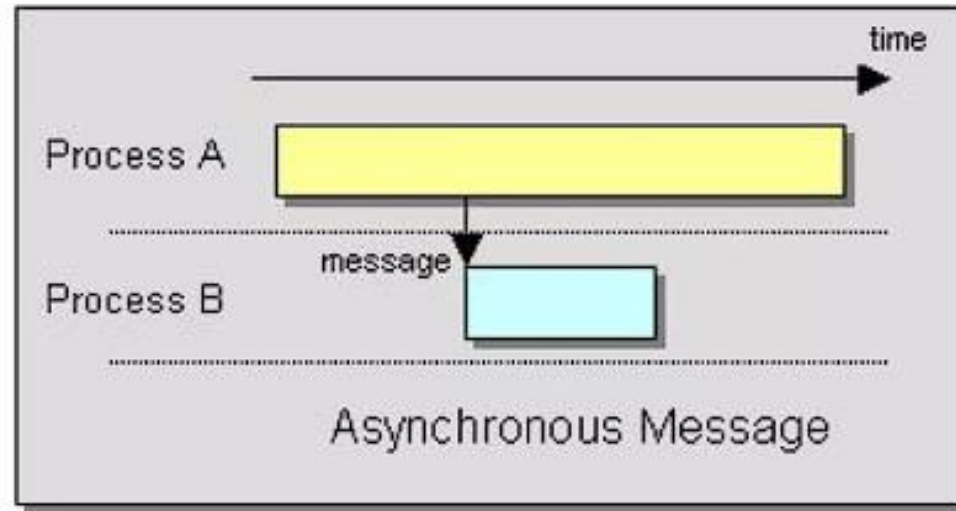
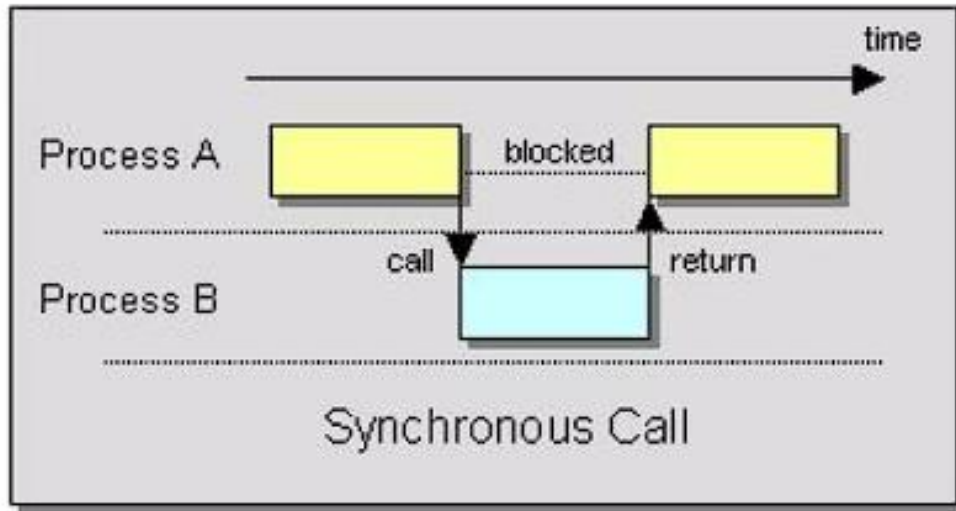
É um conceito essencial em JavaScript que permite que seu código seja executado em segundo plano sem bloquear a execução de outro código. Os desenvolvedores podem criar aplicativos mais eficientes e responsivos usando recursos como retornos de chamada, `async/await` e `promises`.

Material adaptado disponível em: * google 2023



Em códigos síncronos, todas as funções e requisições trabalham em sincronia, em um contato direto, do início ao fim da comunicação. Dessa maneira, esse código só permite uma requisição por vez.

Programação assíncrona



Por exemplo: se fizermos uma requisição para uma API, precisamos esperar a sua resposta. Mas, imagine que tenhamos mais de uma requisição para fazer. Com um código síncrono quaisquer outras requisições além da principal são bloqueadas até que a primeira termine. Em outras palavras, precisamos esperar a resposta da primeira requisição para só então ir para a próxima.

Uma requisição assíncrona é diferente. Ela não trabalha em sincronia. Dessa forma, nós podemos realizar várias requisições ao mesmo tempo, e uma requisição não irá afetar a outra. Em poucas palavras, com um código assíncrono, a aplicação é mais dinâmica.

Programação assíncrona

Um método de callback é uma rotina que é passada como parâmetro para outro método. É esperado então que o método execute o código do argumento em algum momento. A invocação do trecho pode ser imediata, como em um (callback síncrono), ou em outro momento (callback assíncrono). Conforme descrito em: [\[1\]](#)

Os meios em que os callbacks são suportados em diferentes linguagens de programação diferem, porém eles são normalmente implementados com sub-rotinas, expressões lambda, blocos de código ou ponteiros de funções.

Uma função que recebe outra função como argumento é considerada uma função de ordem maior, em contraste à uma função de primeira ordem. Há de se notar no entanto que existem outros tipos de funções de ordem maior, como as que retornam uma nova função sem receber uma como argumento.

Dois exemplos clássicos de funções de callback são as **funções `setTimeout()` e `setInterval()`**, que são nativas do JavaScript. Ambas esperam uma função de retorno.

Exercises

- * Concluir exercícios da aula anterior.
- https://eloquentjavascript.net/06_object.html Exercícios final do capítulo
- Ficha de exercícios

References

- <https://eloquentjavascript.net/>
- <https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference>
- <https://www.freecodecamp.org/news/author/freecodecamp/>
- <https://www.devmedia.com.br/javascript-arrays/4079>
- <https://github.com/braziljs/eloquente-javascript/tree/master>
- <https://www.javascripttutorial.net/>
- <https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/>
- <https://www.w3schools.com/js/default.asp>
- Também foram realizadas adaptações e modificações com base no material disponibilizado pelo Professor Luís Falcão, acesso online em: <https://github.com/isel-leic-ipw/>

Valderi Leithardt, Dr.

Professor IPW

valderi.leithardt@isel.pt