Aula 01 – Conhecendo JavaScript

Antes que o Javascript se tornasse popular, para que a linguagem evoluísse obedecendo a determinados padrões e normativas, os criadores do Javascript se associaram ao ECMA (European Computer Manufactures Association) em 1996. Como o nome Javascript já havia sido patenteado pela Sun Microsystems (atual Oracle), optou-se por se definir um novo nome à linguagem utilizando a junção das palavras ECMA e Javascript, surgindo então o ECMAScript.

Após a adesão ao ECMA, o ECMAScript passou por diversas versões:

ECMAScript 1: lançada oficialmente em 1997, representa a primeira versão da linguagem criada por Brandon, mas com padrões e normativas definidos após a adesão ao ECMA.
ECMAScript 2: criada em agosto de 1997 para se adequar à ISO/IEC 16262.
ECMAScript 3: criada em dezembro de 1999, esta versão possui melhorias importantes para a comunidade, permitindo implementações de laços de repetição do-while, tratamento de exceções, dentre outros recursos.
ECMAScript 4: concluída em 2008, esta versão teve seu desenvolvimento baseado em ML (Meta Language), uma linguagem de programação funcional mais utilizada em ambientes de pesquisa acadêmica. Devido à grande quantidade de mudanças em relação ao ECMAScript 3, o que tornaria esta versão totalmente disruptiva em relação às anteriores, a mesma foi abandonada pelo comitê técnico da ECMA-262, optando-se por se dar continuidade evolutiva à versão anterior.
ECMAScript 5: após o impasse gerado com a versão 4, foi lançada em 2012 oficialmente a versão 5 do ECMAscript possuindo recursos valiosos como o suporte a JSON, métodos mais avançados de manipulação de arrays, getters e setters, dentre outros.
ECMAScript 6: lançada em 2015, atribui recursos avançados à linguagem como reflection, collections, binary data, dentre outros.
ECMAScript 7: também conhecida como ECMAScript 2016 (ano da sua conclusão), possui como operadores exponenciais, dentre outros.

Fonte: https://medium.com/trainingcenter/afinal-javascript-e-ecmascript-s%C3%A3o-a-mesma-coisa-498374abbc47

A linguagem

JavaScript utiliza dois tipos: primitivos e de referência. Os tipos primitivos são armazenados como tipo de dados simples. Os tipos de referência são armazenados como objetos e, na realidade, são apenas referências a posições de memória.
Enquanto outras linguagens de programação distinguem tipos primitivos de tipos de referência ao armazenar tipos primitivos na pilha e os tipos de referência no heap, o Javascript não utiliza esse tipo de conceito. Valores primitivos são armazenados diretamente no objeto variável.

Tipos primitivos

Boolean	true ou false
Number	Qualquer valor numérico inteiro ou ponto flutuante
String	Um caractere ou uma sequencia de caracteres
Null	Um tipo primitivo que tem apenas um valor: null
Undefined	Um tipo primitivo que tem apenas um valor: undefinded

```
var nome = “Orlando”
var count = 25
var custo = 1.5
var encontrado = true
var objeto = null
var flag = undefined
var ref // undefined é atribuido automaticamente

typeof(nome)
```

```
“5” == 5  // true
“5” === 5 // false
```

Quando a igualdade dupla é utilizada, a string ‘5’ e o número 5 são considerados iguais porque a igualdade dupla converte a string em um número antes de fazer a comparação. A igualdade tripla faz a comparação sem converter a variável de um tipo para outro.

Métodos Primitivos

Os tipos null e undefined não contêm métodos. Os demais tipos primitivos sim.

```
var nome = “Orlando”;
var lowecaseNome = nome.toLowerCase();
var primeiraLetra = nome.charAt(0);
var letras = nome.substring(2, 5);

var count = 10;
var fixedCount = count.toFixed(2);
var hexCount = count.toString(16); // converte para a

var flag = true;
var stringFlag = flag.toString();
```

O Javascript faz com que eles pareçam objetos para oferecer uma experiência consistente na linguagem.

Tipos de referência

Os tipos de referência representam objetos em JavaScript e são o recurso encontrado na linguagem que mais se assemelha às classes. Valores de referência são instâncias de tipos de referência e são sinônimos de objetos. Um objeto é uma lista não ordenada de propriedades constituídas de um nome (sempre uma string) e um valor. Quando o valor de uma propriedade for uma função, ela será chamada de método.

```
var obj1 = new Object();
var obj2 = obj1 ;
obj2.minhaprop = “Incrivel”;
obj1.minhaprop;
obj2.minhaprop;
obj1 = null;
```

Instanciando tipos próprios

O tipo Object é somente um entre uma grande variedade de tipos de referência do Javascript. Outros tipos próprios:

Array: uma lista ordenada de valores indexados numericamente;
Date: uma data e uma hora;
Error: um erro de execução (há vários subtipos mais específicos de erro)
Function: uma função;
Object: um objeto genérico;
RegExp: uma expressão regular;

```
var items = new Array();
var now = new Date();
var erro = new Error("Deu ruim");
var func= new Function("console.log('oi');");
var objeto = new Object();
var re = new RegExp("\\d+");
```

Podemos criar objetos com a sintaxe de objeto literal.

```
var livro = {
   nome: "Introducao a Orientação a Objetos ",
  ano: 2017
};
```

Ou usar string literais para nome de propriedades

```
var livro = {
   "nome": "Introducao a Orientação a Objetos ",
  "ano" : 2017
};
```

```
var livro = new Object();
livro.nome = "Introducao a Orientação a Objetos ";
livro.ano = 2017
```

O resultado final de cada um dos três é o mesmo: um objeto com duas propriedades.

Arrays

Você pode definir um literal de array

```
var cores = ['vermelho','azul','verde'];
cores[0];
```

```
var cores = new Array("vermelho","azul","verde");
cores[0];
```

Literais de função

Você pode definir um literal de função

```
function reflect(value) {
    return value;
}
```

O efeito é o mesmo se:

```
var reflect = new Function("value","return value;");
```

Normalmente o uso do construtor Function normalmente é desaconselhável por ser difícil de manter, ler e depurar.

Literais de expressões regulares

É possível criar uma expressão regular sem o uso de RegExp

```
var numbers = /\d+/g;
```

O efeito é o mesmo se:

```
var number = new RegExp("\\d+","g");

str = '10';
str.search(number);

str2 = 'abc';
str2.search(number);
```

Identificando os tipos de referência.

```
var number = new RegExp("\\d+","g");

numbers instanceof Function;

numbers instanceof RegExp;
```

Tipos wrapper primitivos

Há três tipos wrapper primitivos: String, Number e Boolean. Esses tipos de referência especiais existem para fazer com que trabalhar com valores primitivos seja tão simples quanto trabalhar com objetos.

```
var nome = "Orlando Saraiva";
var primeiraLetra = nome.charAt(0);
primeiraLetra;
```

Internamente, ocorre isso:

```
var nome = “Orlando Saraiva”;
var temp = new String(nome);
var primeiraLetra = temp.charAt(0);
temp = null;
primeiraLetra;
```

Resumo

Embora não tenha classes, o Javascript tem tipos. Cada variável ou porção de dado é associado a um tipo primitivo ou de referência específico. Os cinco tipos primitivos (string, number, Booleans, null e undefined) representam valores simples armazenados diretamente no objeto variável em um determinado contexto.