PUC Minas

Tecnologias Web

Linguagem JavaScript
Noções Básicas

Prof. Rommel Vieira Carneiro

JavaScript – Tópicos

- Polyfills
- Case e Comentários
- Variáveis e constantes
- Tipos e Estruturas de Dados
- Controle de Fluxos
- Funções
- Classes
- Módulos
- Destructuring

Polyfills

O termo Polyfill refere-se a porções de código utilizados para compatibilizar funções modernas com browsers antigos. Famoso "*tapa-buracos*".

Exemplos

- SessionStorage no IE7 → weblink
- HTML5Shiv – Tags Semânticas no IE 6, 7 e 8 → weblink
- Respond.js – Media Queries (Min/Max-width) no IE 6, 7 e 8 → weblink

Recursos Interessantes

- Polyfill.io – Serviço de Polyfill ajustável ao browser
- Babel Polyfill – Transpilador de Javascript que gera o código ES5

Case e Comentários

- JavaScript é ***case sensitive*** e utiliza conjunto de caracteres **Unicode**

```
Nome é diferente de nome que é diferente de NOME
```

- Comentários

```
// uma linha de comentário

/*
   Comentário de
   várias linhas
*/
```

Variáveis e Constantes

- Declaração de variáveis, opcionalmente com atribuição de valor

```
var a = 0;
```

- Declaração de variáveis de escopo local, opcionalmente com atribuição de valor

```
let a = 0;
```

- Declaração de constantes

```
const A = 0;
```

  - Valores constantes não podem ser alterados
  - O escopo das constantes é semelhante ao das variáveis declaradas como let.

Escopo de variáveis

```
<script>
  var a = 1;
  {
    let c = 5;
  }
  console.log (a);
  console.log (c);
</script>
```

Saída

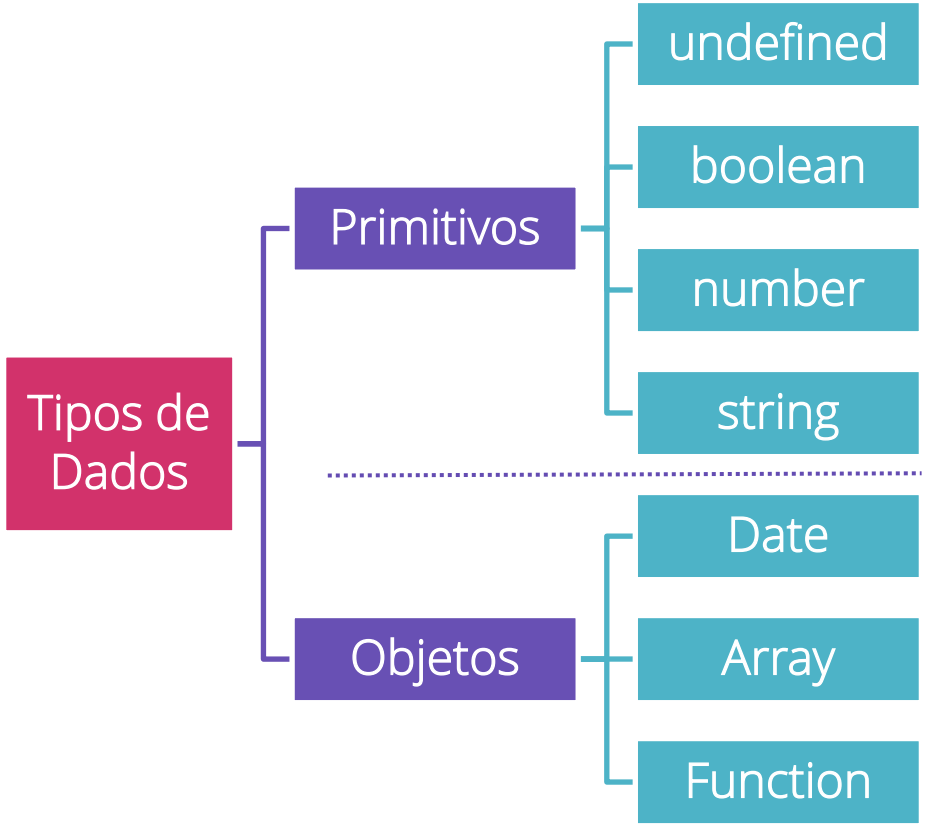
```
1
Uncaught ReferenceError: c is not defined
```

Variáveis e Constantes

- **var** define elementos de escopo global ou escopo de função, se for dentro de funções
- **let** e **const** definem elementos de escopo local.

```
var z = 'original Z';
{
    let x = 'original X';
    var y = 'original Y';
    console.log(`x: ${x} -- y: ${y} -- z: ${z} -- `);
    {
        // novo escopo ==> novo x
        const x = "novo---- X";
        console.log(`x: ${x} -- y: ${y} -- z: ${z} -- `);
    }
    // retorno ao escopo anterior ==> original x
    console.log(`x: ${x} -- y: ${y} -- z: ${z} -- `);
}
console.log(`z: ${z}`); // z é global
console.log(`y: ${y}`); // y é global
console.log(`x: ${x}`); // x é local e linha gera erro
```

Tipos e Estruturas de Dados



Tipagem Dinâmica

```
<script>
  var a = 1;
  console.log (a);
  var a = 2.5;
  console.log (a);
  var a = "abc";
  console.log (a);
  var a = true;
  console.log (a);
  var a = ["a", "e", "i", "o", "u"];
  console.log (a);
</script>
```

Saída

```
1
2.5
abc
true
["a", "e", "i", "o", "u"]
```

Tipos e Estruturas de Dados - Boolean

- Variáveis boolean podem ter valores **true** ou **false**
- Os operadores lógicos permitem comparações de expressões. São eles:

Operador	Descrição	Exemplo	Resultado
==	Igual a	`x == y`	`true` se x e y são iguais
===	Igual e mesmo tipo	`x === y`	`true` se x e y são iguais e do mesmo tipo
!==	Não igual a	`x !== y`	`true` se x e y são diferentes
>	Maior que	`x > y`	`true` se x é maior que y
<	Menor que	`x < y`	`true` se x é menor que y
>=	Maior ou igual a	`x >= y`	`true` se x é maior ou igual a y
<=	Menor ou igual a	`x <= y`	`true` se x é menor ou igual a y
!	Negativo	`!x`	`true` se x é `false`
&&	E exclusivo (ambos true)	`x && y`	`true` se x e y são `true`
||	Ou (pelo menos um true)	`x || y`	`true` se ou x ou y são `true`

Tipos e Estruturas de Dados – Number

- Todos os números são de ponto flutuante com tamanho de 64 bits
- O objeto Math contém funções matemáticas avançadas
- NaN é o valor retornado por qualquer operação que não resulte em um número válido
- Existem os valores infinito e menos infinito.
  - Number.POSITIVE_INFINITY e Number.NEGATIVE_INFINITY

Números

```
<script>
  var a = 123;
  console.log (a);
  var a = 1.23;
  console.log (a);
  var a = Math.PI;
  console.log (a);
  var a = 5.8;
  console.log (Math.round (a),
    Math.trunc(a), 0/0);
</script>
```

Saída

```
123
1.23
3.141592653589793
6 5 NaN
```

Tipos e Estruturas de Dados – String

- Strings podem ser declaradas com aspas ("abc") ou apostrofes ('abc') ou crases (`abc`)
- São compostas por sequências de caracteres Unicode
- O operador (+) pode ser utilizado para concatenar strings
- Embora sejam primitivos, é possível utilizar métodos e propriedades a partir de variáveis do tipo string
  - length → tamanho da string
  - charAt() → Caracter específico
  - toUpperCase() → String maiúscula
  - toLowerCase() → String minúscula

Strings

```
<script>
  var a = "abc";
  console.log (a);
  var a = 'abc';
  console.log (a);
  console.log ("Tamanho de a: " +
                        a.length);
  console.log (a.toUpperCase());
</script>
```

Saída

```
abc
abc
Tamanho de a: 3
ABC
```

Tipos e Estruturas de Dados – String

Caracteres Especiais

Para se representar caracteres especiais é necessário utilizar a notação de escape com a barra invertida (\).

Exemplo

```
<script>
  var a = "Exemplo\nOlá \'Mundo\'";
  console.log (a);
</script>
```

Saída

```
Exemplo
Olá 'Mundo'
```

Código	Saída
\0	Caracter NULL
\'	Aspas simples
\"	Aspas duplas
\\	Barra invertida
\n	Nova linha (ENTER)
\r	Retorno de Carro (CR)
\v	Tabulação vertical
\t	Tabulação
\b	Backspace
\f	form feed
\uXXXX	Caracter Unicode
\xXX	Caracter Latin-1

Tipos e Estruturas de Dados – String – Template Literals

- Similar a strings, com crases (`` `...` ``) no lugar de apostrofes (`'...'`) ou aspas (`"..."`)
- Permite textos em múltiplas linhas
- Permite a inserção de expressões por meio da construção `${ expr }`

```
// Basic literal string creation
let s1 = `In JavaScript '\n' is a line-feed.`

// Multiline strings
let s2 = `In JavaScript this is
not legal.`

// String interpolation
var name = "Bob", time = "today";
let s3 = `Hello ${name}, how are you ${time}?`

console.log(`${s1} \n${s2} \n${s3}`);
```

Tipos e Estruturas de Dados

Conversão de strings em números

- Função parseInt ()
- Função parseFloat ()

Conversão de números para strings

- A conversão é automática

Conversões

```
<script>
  var a = parseInt ("256");
  console.log (a);
  var a = parseInt ("2.56")
  console.log (a);
  var a = parseFloat ("2.56");
  console.log (a);
  var a = "valor: " + 2.56;
  console.log (a);
</script>
```

Saída

```
256
2
2.56
Valor: 2.56
```

Tipos e Estruturas de Dados – Objetos

Objetos

- Em JavaScript, os objetos são coleções de pares nome e valor
  - O nome é uma string em JavaScript
  - O valor pode ser de qualquer tipo, incluindo outros objetos
- Os objetos são dinâmicos e podem receber novas propriedades e métodos a qualquer momento
- JSON ou **JavaScript Object Notation** é a forma como se descrevem os objetos em JavaScript

Objetos

```
<script>
  var o = new Object();
  o.name = "João";
  o["age"] = 16;
  console.log (o.name+"-"+o.age);

  var email = {
    mensagem: "Olá Pamela",
    detalhes: {
      para: "Pamela",
      de: "João"
    }
  };
  console.log (email.mensagem);
</script>
```

Saída

```
João-16
Olá Pamela
```

Tipos e Estruturas de Dados – Date

- As variáveis Date são objetos que permite tratar Datas e Horas
- Existem diversos construtores para Datas:
  - Date ()
  - Date (milissegundos)
  - Date (string)
  - Date (Ano, Mês, Dia, Hora, Minuto, segundo, milissegundo)
- Ao referenciar o número dos meses, é importante saber que janeiro é 0 e dezembro é 11.

Objetos – Datas

```
<script>
  var a = new Date();
  console.log (a.toLocaleString());
  var a = new Date(2016, 0, 31);
  console.log (a.toLocaleString());
</script>
```

Saída

```
03/09/2016 15:07:58
31/01/2016 00:00:00
```

Tipos e Estruturas de Dados – Array

- Os arrays também são objetos
- A criação pode ser feita de duas formas
  - Função Array ()
  - Atribuição de literal
- Possuem a propriedade length que retorna o número de itens

Objetos – Vetores

```
<script>
  var a = new Array();
  a[0] = "vermelho";
  a[1] = "preto";
  console.log (a);

  var b = new Array("rosa", "azul");
  console.log (b);

  var c = ["amarelo", "verde"];
  console.log (c);
  console.log (c.length);
</script>
```

Saída

```
["vermelho", "preto"]
["rosa", "azul"]
["amarelo", "verde"]
2
```

Tipos e Estruturas de Dados – Array – Métodos

Método	Descrição
concat()	Junta dois ou mais vetores e retorna uma cópia dos vetores assoriados
fill()	Preencher os elementos em um vetor com um valor estático
find()	Retorna o valor do primeiro elemento em um vetor que atender ao filtro
findIndex()	Retorna o índice do primeiro elemento em um vetor que atender ao filtro
forEach()	Chama uma função para cada elemento do vetor
indexOf()	Busca um elemento no vetor e retorna a sua posição
isArray()	Verifica se um objeto é um vetor
join()	Junta todos os elementos de um vetor em uma string
lastIndexOf()	Pesquisar o vetor por um elemento, começando no final e retorna a sua posição
pop()	Remove o último elemento de um vetor e retorna o elemento
push()	Adiciona novos elementos para o final de um vetor e retorna o novo comprimento
reverse()	Inverte a ordem dos elementos em um vetor
slice()	Seleciona uma parte de um vetor e retorna o novo vetor
sort()	Classifica os elementos de um vetor
splice()	Adiciona/remove elementos de um vetor
toString()	Converte um vetor em uma string e retorna o resultado
valueOf()	Retorna o valor primitivo de um vetor

Fonte: Adaptado de Array Reference - http://www.w3schools.com/jsref/jsref_obj_array.asp

Controle de Fluxo – Seleção – if

Estrutura

```
// Seleção simples
if (expressão) {
   comandos
}

// Seleção dupla
if (expressão) {
   comandos_true
}
else {
   comandos_false
}
```

Exemplo

```
if (nota > 60) {
    alert('O aluno foi aprovado.');
} else {
    alert('O aluno foi reprovado.');
}
```

Ao avaliar a expressão, que chamamos de expressão booleana, o programa chegará a um resultado do tipo **boolean** que pode ser verdadeiro (**true**) ou falso (**false**). De acordo com este resultado, o programa escolhe qual parte do programa será executado.

Controle de Fluxo – Seleção Múltipla – switch

Estrutura

```
switch (expressão) {
  case valor_1:
    comandos_1
    [break;]

  case valor_2:
    comandos_2
    [break;]

  default:
    comandos_default
}
```

Exemplo

```
switch (fruta) {
    case 'abacaxi':
        alert('Abacaxi');
        break;
    case 'banana':
        alert('Banana');
        break;
    default:
        alert('Outras frutas');
        break;
}
```

Na declaração de seleção múltipla, o programa escolha entre diversas partes do código a partir do valor de uma expressão. O comando **break** é utilizado para que, caso um valor seja identificado, o programa finaliza e vá para a próxima linha depois da estrutura de seleção múltipla.

Controle de Fluxo – Repetição – while

Estrutura

```
while (expressão) {
    comandos
}
```

Exemplo

```
// variável para servir de contador
let i = 3;

// Contagem regressiva avaliando contador
while (i > 0) {
    alert ('Contagem: ' + i );

    // Diminui o contador em 1
    i--;
}
```

É importante que nos comandos haja a alteração dos termos envolvidos na expressão avaliada para evitar que o programa entre em **loop infinito** e não consiga parar de repetir os comandos.

Controle de Fluxo – Repetição – for

Estrutura

```
for (inicialização; expressão; finalização)
{
   comandos
}
```

Exemplo

```
let aFrutas = ['banana', 'pera', 'uva'];

// Loop para processar array
for (let i = 0; i < aFrutas.length; i++) {
    const element = aFrutas[i];
    console.log (element);
}
```

As três partes que compõe a definição da repetição:

- **Inicialização**: parte que inicializa termos geralmente utilizados no controle da repetição como um índice das iterações;
- **Expressão**: condição que valida a cada iteração se o programa deve continuar repetindo os comandos;
- **Finalização**: parte que é executada ao final de cada iteração e geralmente atualiza os termos utilizados na expressão que valida a continuidade da repetição.

Funções

- As funções são objetos JavaScript
- Podem ter qualquer número de parâmetros
- Os parâmetros não são obrigatórios
- Mais parâmetros podem ser passados do que especificados pela função
- Podem retornar um valor explícito ou undefined

```
function soma( x, y ) {
    var total = x + y;
    return total;
}
soma()       // retorna NaN
soma(2,3)    // retorna 5
```

Funções – arguments

É possível acessar um objeto **arguments** dentro de uma função para obter os valores recebidos pela função como parâmetros.

O objeto **arguments** é uma variável local acessível dentro de funções que se baseia em um array.

```
function media() {
    var soma = 0;
    for( var i=0, j=arguments.length; i<j; i++ ) {
        soma += arguments[i];
    }
    return soma/arguments.length;
}

media( 2, 3, 4, 5 ) retorna 3.5
media.apply( null, [2, 3, 4, 5] ) retorna 3.5
```

Funções Anônimas – Arrow Functions

Arrow Functions são uma sintaxe simplificada para a definição de funções anônimas.

```
// forma tradicional de declaração
soma = function (a, b) { return a + b }

// Declaração com arrow functions
soma = (a, b) => { return a + b }

// ou sem as chaves
soma = (a, b) => a + b
```

Funções Anônimas – Arrow Functions

É importante ressaltar que o operador **this** em uma Arrow Function faz referência ao bloco que contém a função, diferentemente de funções normais.

```
// Lexical this
var bob = {
    _name: "Bob",
    _friends: [],

    printFriends() {
        this._friends.forEach(f =>
            console.log(this._name + " knows " + f));
    }
}
```

Funções Anônimas – Arrow Functions – Exemplos

```
// Expression bodies
evens = [0, 2, 4, 6, 8, 10, 12, 14, 16];
var odds = evens.map(v => v + 1);
var nums = evens.map((v, i) => v + i);
var pairs = evens.map(v => ({ even: v, odd: v + 1 }));

// Statement bodies
fives = [];
nums.forEach(v => {
    if (v % 5 === 0)
        fives.push(v);
});
```

Funções – Parâmetros default

Funções agora suportam definição de valores padrão para parâmetros não enviados

```
function foo(index = 0, testing = true) {
    console.log(`index: ${index} | testing: ${testing} `);
};

foo(); // Imprime ==> index: 0 | testing: true
foo(5); // Imprime ==> index: 5 | testing: true
foo(8, false); // Imprime ==> index: 8 | testing: false
```

Classes

- As classes em JavaScript são funções que inicializam novos objetos (como construtores)
- **this** é uma referência ao objeto atual
- **new** é semelhante a C++, uma chamada a funções construtoras

```
function Pessoa( primeiro, ultimo ) {
    this.primeiro = primeiro;
    this.ultimo = ultimo;
    this.nomeCompleto = function() {
        return this.primeiro + '   ' + this.ultimo;
    }
    this.nomeCompletoInvertido = function() {
        return this.ultimo + ',  ' + this.primeiro;
    }
}

var chefe = new Pessoa( "João", "Paiva" );
```

Classes

- A classe anterior duplica os métodos para cada instância
- Uma forma melhor seria:

```
function nomeCompletoDePessoa() {
    return this.primeiro + '    ' + this.ultimo;
}
function nomeInvertidoDePessoa() {
    return this.ultimo + ',  ' + this.primeiro;
}
function Pessoa( primeiro, ultimo ) {
    this.primeiro = primeiro;
    this.ultimo = ultimo;
    this.nomeCompleto = nomeCompletoDePessoa;
    this.nomeCompletoInvertido = nomeiInvertidoDePessoa;
}
```

Módulos

Suporte para módulos de componentes na Linguagem JavaScript

```
// lib/math.js
export function sum(x, y) {
    return x + y;
}
export const pi = 3.141593;
```

```
// app.js
import * as math from "lib/math";
alert("2π = " + math.sum(math.pi, math.pi));
```

```
// otherApp.js
import { sum, pi } from "lib/math";
alert("2π = " + sum(pi, pi));
```

Módulos

Implementações anteriores

- CommonJS Modules
  - Carregamento síncrono
  - Foco em servidores

- Asynchronous Module Definition (AMD)
  - Carregamento assíncrono
  - Foco nos browsers

Módulos

Tipos de exportação: Named Exports | Default Export

Permite exportação de vários componentes independentes por módulo

```
//------ lib.js ------
export const sqrt = Math.sqrt;
export function square(x) {
    return x * x;
}
export function diag(x, y) {
    return sqrt(square(x) + square(y));
}

//------ main.js ------
import { square, diag } from 'lib';
console.log(square(11)); // 121
console.log(diag(4, 3)); // 5
```

Fonte: ECMAScript 6 modules: the final syntax - http://2ality.com/2014/09/es6-modules-final.html

Módulos

Tipos de exportação: Named Exports | Default Export

Permitem um único componente a ser exportado pelo módulo como default

```
//------ myFunc.js ------
export default function () { ... };

//------ main1.js ------
import myFunc from 'myFunc';
myFunc();
```

```
//------ MyClass.js ------
export default class { ... };

//------ main2.js ------
import MyClass from 'MyClass';
let inst = new MyClass();
```

Fonte: ECMAScript 6 modules: the final syntax - http://2ality.com/2014/09/es6-modules-final.html

Módulos

Formas de Importação de Componentes

```
// Default exports and named exports
import theDefault, { named1, named2 } from 'src/mylib';
import theDefault from 'src/mylib';
import { named1, named2 } from 'src/mylib';

// Renaming: import named1 as myNamed1
import { named1 as myNamed1, named2 } from 'src/mylib';

// Importing the module as an object
// (with one property per named export)
import * as mylib from 'src/mylib';

// Only load the module, don’t import anything
import 'src/mylib';
```

Fonte: ECMAScript 6 modules: the final syntax - http://2ality.com/2014/09/es6-modules-final.html

Destructuring

Permite a associação de elementos utilizando casamento de padrões

```
// list matching
var [a, b, c] = [1, 'orange', true];
console.log(`a: ${a} | b: ${b} | c: ${c}`)
// Imprime ==> a: 1 | b: orange | c: true

// Can be used in parameter position
function g({ name }) {
    console.log(`name: ${name}`); // Imprime ==> name: 5
}
g({ name: 5 })

// Fail-soft destructuring
var [a] = [];
console.log(`a: ${a}`); // Imprime ==> a: undefined

// Fail-soft destructuring with defaults
var [a, b = 1] = [2];
console.log(`a: ${a} | b: ${b}`); // Imprime ==> a: 2 | b: 1
```

Referências

- **ECMAScript 2015 Language Specification**
  http://www.ecma-international.org/ecma-262/6.0/
- **The ES6 Guide**
  https://flaviocopes.com/es6/
- **ECMAScript 6 New Features: Overview & Comparison**
  http://es6-features.org
- **Exploring ES6**
  http://exploringjs.com/es6/
- **ECMAScript 6 - Luke Hoban ➔ Fonte de parte dos códigos de exemplo deste material**
  https://github.com/lukehoban/es6features
- **Top 10 ES6 features by example**
  https://blog.pragmatists.com/top-10-es6-features-by-example-80ac878794bb
- **Top 10 ES6 Features Every Busy JavaScript Developer Must Know**
  https://webapplog.com/es6/

Obrigado!

Prof. Rommel Vieira Carneiro