Introdução ao JAVASCRIPT

A lingaguem JAVASCRIPT

Também chamada de JS, é a linguagem de criação de scripts para a Web que permite:

– Adicionar funcionalidades;

– Verificar formulários;

– Comunicar com servidores;

– Progarmação Front-end e Back-end.

Sua evolução:

– Originalmente criada na Netscape por Brendan Eich em 1994;

– Disputa: Netscape vs Microsoft (VB Script);

– Java da Sun surgia como potencial;

– No início o JavaScript não permitia a criação de applets nem de aplicativos. Apenas residia dentro de documentos HTML para promover diferentes níveis de interatividades não são suportados pelo HTML sozinho;

– Atualmente, o maior mantedor da linguagem é a Fundação Mozilla, com documentação disponível em https://developer.mozilla.org/en/docs/JavaScript

– Com o tempo, muitas funcionalidades foram criadas em forma de Script para os browsers e foram "incorporadas" ao JavaScript;

– Os principais padrões são:

- **ECMAScript** (Versão 7, de Junho de 2016): A Linguagem Núcleo. Mantido pela ECMA International (Associação Internacional de padronização de tecnologias da Informação e Comunicação);
- **DOM**: (Document Object Model): Define a Interface da Linguagem com o Browser. Mantido por W3C;

Comandos e funções JS

Iniciando um script JS (para o browser)

Para inserir códigos JavaScript é necessário iniciar com uma Tag HTML apropriada

```
<script> </script>
```

Da mesma forma que nos arquivos CSS, pode-se deixar funções e comandos JavaScript em arquivos externos. Estes arquivos devem ter a extensão .JS

- Para importar o arquivo externo deve-se utilizar o seguinte comando:

```
<script src= "[nomedoarquivo].js"></script>
```

Saída de dados na página (document.write)

O JavaScript é orientado a objetos e utiliza a classe: **document**

Exemplo de Hello World:

```
<script>
  document.write("Hello World!");
</script>
```

Usando formações HTML:

```
<script>
  document.write("<h2> Hello World! </h2>");
</script>
```

Comentários

- Comentários de única linha;
- Comentários de múltiplas linhas.

```
// Este comentário ocupa uma única linha

/* Já este comentário é mais longo e utiliza
várias linhas */
```

Declaração de Variáveis e Constantes

Variáveis

- var
- let

```
Usando a declaração por VAR
  for(var x = 0; x < 5; x++) {
     // x existe aqui
  };
  // x ainda está visível aqui
```

```
Usando a declaração por LET
  for(let x = 0; x < 5; x++ ) {
     // x existe aqui
  };
  // x não está visível aqui
```

Constantes

- const

```
const pi = 3.141564;
```

As variáveis possuem tipos de dado dinâmico:

- **var x;** // x é indefinido
- **x = 5;** // x é um número
- **x = "John";** // x é uma string
- **x = true;** // x é um valor lógico
- **x = null;** // x é indefinido

Arrays

- Variáveis do tipo Array

  Arrays são criados através de um construtor e possuem tamanho dinâmico:

```
var nomes = new Array();
//var nomes = [];
nomes[0] = "Fulano de Tal";
nomes[1] = "Beutrano";
nomes.push("Ciclano");
```

- Variáveis do tipo Structure (objetos)

  Structures definidos por são várias composições de atributos: valores dentro de uma única variáveis e são muito utilizadas com passagens de dados via SOAP e REST:

```
var mail = {
   from: "Fulano <suaconta@gmail.com>",
   to: "destinatario@gmail.com",
   subject: "Envio de email usando Node.js",
   message: "<b>Olá mundo!</b>"
}
```

  Para usar os atributos:

```
mail.message = "mensagem importante";  //adicionar valor
document.write(mail.subject);   //exibir valor
```

Operadores Aritméticos:

+	Efetuar soma de números ou Concatenação de strings
-	Efetuar subtração de números
*	Efetuar multiplicação de números
/	Efetuar divisão de números (Sempre divisão real)
%	Resto da divisão
++	Incremento
--	Decremento

Operadores Lógicosa:

==	Valor igual. (5 == “5”) retorna true
===	Valor e tipo iguais. (5 === “5”) retorna false
!=	Valor diferente. (5 != “5”) retorna false
!==	Valor e tipos diferentes. (5 !== “5”) retorna true
>	Maior
<	Menor
>=	Maior ou Igual
<=	Menor ou Igual
&&	E (and)
||	OU (or)
!	NÃO (not)

Carregando conteúdos de arquivos externos (bibliotecas)

- require
- import

O **require** existe só em CommonJS (a maneira que o Node.js criou para importar e exportar módulos dentro de uma aplicação)

O **import** é ES6, ou seja, uma nova ferramenta que ambos JavaScript do browser e JavaScript do servidor (Node.js) podem usar.

```
import Library from 'some-library';
const Library = require('some-library');
```

Funções

```
function nomeDaFuncao( /*parâmetros*/ ) {
 /* código que será executado */

 return/*Valor retornado*/;
}
```

Hierarquia do objeto

```
//Construtor
function Exemplo() {
   this.propriedade = 'Isso é uma propriedade.',
   this.metodo = function() {
         return 'Isso é um metódo';
   }
}
```

Instanciando objetos

```
var objeto = new Exemplo(); //Instância do construtor "Exemplo"

//Alerta os respectivos textos na tela
alert(objeto.propriedade),
alert(objeto.metodo());
```

Estruturas Condicionais

if else - funciona igual em C/Java:

```
if (condição) {
   código para quando retornar true
}
else {
   código para quando retornar false
}
```

Estruturas de Repetição

for, while e **do while** - funcionam da mesma forma que em C/Java;

– Incluindo os comandos continue e break;

```
for (x=0;x<10;x++) { }

while (x < 10) { }

do { } while (x < 10);
```

Tratamento de Exceções

try e **catch** - funcionam da mesma forma que Java e C#;

```
// Delimitar área que será verificada:
try {
  //Código passivo de erro
}

// Capturar um eventual erro:
catch (erro) {
  //Tratamento para eventual erro capturado
}
```

Entrada de Dados

Ao desenvolver páginas para internet, a entrada de dados sempre serão informadas pelo usuário por meio de formulário HTML utilizando as seguintes tags:

<form>

<input> Para caixas de textos, caixas de seleção, caixas de opções únicas ou múltiplas;

<textarea> Para textos longos e múltiplas linhas;

<button> Para botões sem ações predefinidas;

<submit> Para ações de enviar os dados informados;

</form>

É possível, antes de enviar estes dados ao servidor, validar/verificar se eles possuem coerência em relação ao que é solicitado.

Exemplo:

– O usuário esqueceu campos em branco?

– O e-mail digitado é válido?

– A data digitada é válida?

– Num campo numérico, foi digitado um número?

Trabalhando com JS em servidor (Node)

Node.JS é um interpretador de javascript que roda do lado do servidor. Pensado para trabalhar como um servidor web, ele foi criado como uma plataforma sobre o runtime do interpretador V8 do Google Chrome. Isso é bom porque o V8 transforma o javascript em código nativo de máquina, o que poderia ser traduzido como performance. A idéia central é escrever aplicações em rede rápidas e escaláveis através de chamadas assíncronas.

Executando por console:

Na linha de comando do seu computador digite:

```
node
```

Exemplo de comando por console:

Saida de dados:

```
> console.log('Hello World');
```

Para encerrar a execução:

```
> CTRL ^ C (2 vezes)  ou  .exit
```

Executando com um arquivo JS existente:

Crie e salve o arquivo contendo o script JS.
Exemplo de um arquivo para entrada de dados (pergunta.js):

```
var readline = require('readline');
var resp = "";
var leitor = readline.createInterface({
   input: process.stdin,
   output: process.stdout
});
leitor.question("Qual o valor do dólar hoje?\n", function(answer) {
   var resp = answer;
   console.log("\nSua resposta '" + resp + "' foi recebida com sucesso ");
   leitor.close();
});
```

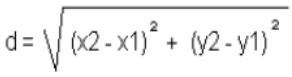
Na linha de comando do seu computador digite:

```
Node pergunta.js
```

Neste caso não precisa forçar a execução pois o node encerra automaticamente por não haver mais comando dentro do arquivo.

Exercício de prática:

1) Construa um programa em JS que, tendo como dados de entrada os valores de x1, x2, y1 e y2 quaisquer, escreva a distância entre eles em um plano cartesiano. A fórmula que efetua tal cálculo é:



Sugestão: utilize como valores para x1 e y1 o dia e mês de seu nascimento (respectivamente) e x2 e y2 o dia e mês na data atual (respectivamente).

Exiba o valor resultante em d.

2) Fazer um código em JS que receba o valor referente a capacidade de carga de um elevador e o peso de 5 pessoas. Informar se o elevador está liberado para subir ou se excedeu a carga máxima.

Criando uma aplicação WEB com NodeJS usando o servidor HTTP

```
var http = require("http");
```

Nota: Existem outros módulos que criam servidores como é caso de "net", "tcp" e "tls".

A partir deste momento temos uma variável http que na realidade é um objeto, sobre o que podemos invocar métodos que estavam no módulo requerido. Por exemplo, uma das tarefas implementadas no módulo HTTP é a de criar um servidor, que se faz com o módulo "createServer()". Este método receberá um *callback* que será executado cada vez que o servidor receba uma solicitação.

```
var server = http.createServer(function (pedido, resposta){
  resposta.end("Ola mundo");
});
```

A função callback que enviamos a createServer() recebe dois parâmetros que são a solicitação e a resposta. Não usamos a solicitação por agora, mas contém dados da solicitação realizada. Usaremos a resposta para enviar dados ao cliente que fez a solicitação. De modo que "resposta.end()" serve para terminar a solicitação e enviar os dados ao cliente. Agora vou dizer ao servidor que se ponha em funcionamento porque até o momento só criamos o servidor e escrevemos o código a ser executado quando se produza uma solicitação, mas não o iniciamos.

```
server.listen(3000, function(){
  console.log("seu servidor está pronto em " + this.address().port);
});
```

Com isto dizemos ao servidor que escute no porto 3000, embora pudéssemos ter posto qualquer outro porto que gostássemos. Ademais "listen()" recebe também um função callback que realmente não seria necessária, mas que nos serve para fazer coisas quando o servidor tenha sido iniciado e esteja pronto. Simplesmente, nessa função callback indico que estou pronto e escutando no porto configurado.

Código completo de servidor HTTP em node.JS
Como você pode ver, em muitas poucas linhas de código geramos um servidor web que está escutando em um porto dado. O código completo é o seguinte:

```
var http = require("http");
var server = http.createServer(function (request, response){
  response.end("Ola CriarWeb.com");
});
server.listen(3000, function(){
```

```
  console.log("seu servidor está pronto em " + this.address().port);
});
```

Colocar em execução o arquivo com Node.JS para iniciar o servidor
Agora podemos executar com Node o arquivo que criamos. Vamos da linha de comandos à pasta onde salvamos o arquivo servidor.js e executamos o comando "node" seguido do nome do arquivo que pretendemos executar:

node servidor.js

Então no console de comandos nos deve aparecer a mensagem que informa que nosso servidor está escutando no porto 3000.

O modo de comprovar se realmente o servidor está escutando a solicitações de clientes no tal porto é acessar com um navegador. Deixamos ativa essa janela de linha de comandos e abrimos o navegador. Acessamos a:

http://localhost:3000