# Aplicações para a Internet I JavaScript

#### JavaScript – Arrays e Objetos

- Tal como em outras linguagens, JavaScript permite usar Arrays para armazenar múltiplos valores numa única variável.
- Criação de um Array:

```
var cars = ["Saab", "Volvo", "BMW"]; var cars = new Array("Saab", "Volvo", "BMW");
```

Obter um elemento de um Array:

```
var name = cars[0];
```

• Alterar um elemento de um Array:

```
cars[0] = "Opel";
```

#### JavaScript – Arrays e Objetos

- Um Array em JavaScript é um tipo especial de um Objeto:
  - Um Array utiliza o índice para aceder aos valores;
  - Um outro objeto utiliza nomes para obter os valores do Objeto.
- Declaração do Objeto:

```
var person = {firstName:"John", lastName:"Doe", age:46};
```

• Obtenção de valores:

```
person.firstName
```

• Nota: Podemos ter Objetos dentro de um Array e vice-versa.

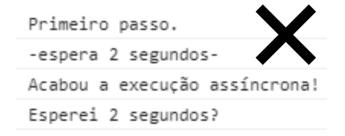
#### JavaScript – Arrays e Objetos

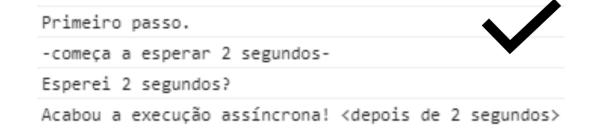
- Métodos e propriedades dos Arrays:
  - Obter tamanho do Array: array.length
  - Ordenar o Array: array.sort();
  - Ordenar de forma inversa o Array: array.reverse();
  - Obter elementos em String: array.toString();
  - Adicionar elemento a Array: array.push(elemento);
  - Remover o último elemento de um Array: array.pop();
  - Remover o primeiro elemento de um Array: array.shift();
  - Adicionar elemento ao início do Array: array.unshift(elemento);

- Diferença entre pedidos síncronos e assíncronos:
  - Síncronos: Um pedido síncrono para a execução do restante código até o pedido concluir. Nesses casos, o motor JavaScript do browser está bloqueado.
  - Assíncronos: Um pedido assíncrono não bloqueia o cliente JavaScript. Nesse momento, o utilizador pode continuar a executar outras operações.

- A função setTimeout() é um exemplo de uma chamada assíncrona.
- Vamos ver como funciona:

```
// primeiro passo
console.log("Primeiro passo.");
// execução assíncrona
setTimeout(function() {
  console.log("Acabou a execução assíncrona!");
}, 2000);
// e isto vai sair quando?
console.log("Esperei 2 segundos?");
```





- AJAX = Asynchronous JavaScript And XML. Não é uma linguagem de programação
- Faz uso do objeto do browser "XMLHttpRequest".
- Ajax permite a troca de informação com o servidor de forma assíncrona.
- AJAX é muito útil no desenvolvimento porque permite:
  - Obter dados de um servidor mesmo após a página ser carregada;
  - Atualizar página ou partes da página sem a recarregar;
  - Enviar dados ao servidor em background e de forma assíncrona.
- Os pedidos devem ser feitos ao mesmo servidor que contém a página, porque os browsers atuais não permitem Access Across Domains.

Criar um pedido AJAX:

var xhttp = new XMLHttpRequest();

Métodos do XMLHttpRequest:

new XMLHttpRequest()	Creates a new XMLHttpRequest object
abort()	Cancels the current request
getAllResponseHeaders()	Returns header information
getResponseHeader()	Returns specific header information
open(method, url, async, user, psw)	Specifies the request  method: the request type GET or POST  url: the file location  async: true (asynchronous) or false (synchronous)  user: optional user name  psw: optional password
send()	Sends the request to the server Used for GET requests
send(string)	Sends the request to the server. Used for POST requests
setRequestHeader()	Adds a label/value pair to the header to be sent

#### • Propriedades do XMLHttpRequest:

onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest.  0: request not initialized  1: server connection established  2: request received  3: processing request  4: request finished and response is ready
responseText	Returns the response data as a string
responseXML	Returns the response data as XML data
status	Returns the status-number of a request 200: "OK" 403: "Forbidden" 404: "Not Found" For a complete list go to the <u>Http Messages Reference</u>
statusText	Returns the status-text (e.g. "OK" or "Not Found")

• Efetuar um pedido ao servidor:

```
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

• O 3º parâmetro define se o pedido é síncrono (false) ou assíncrono (true).

#### GET vs. POST:

- GET é mais simples e mais rápido que o POST e pode ser utilizado quase sempre;
- POST deve ser utilizado quando:
  - Queremos enviar um ficheiro para o servidor;
  - Estamos a enviar uma grande quantidade de dados para o servidor;
  - Enviamos dados introduzidos por um utilizador POST é mais robusto que GET, no controlo de valores.

```
xhttp.open("GET", "demo_get2.asp?fname=Henry&lname=Ford", true);
xhttp.send();

xhttp.open("POST", "ajax_test.asp", true);
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhttp.send("fname=Henry&lname=Ford");
```

#### JavaScript - Callbacks

- As funções de Callback permitem definir código a ser executado quando o pedido efetuado é concluído.
  - No caso de setTimeout, temos: setTimeout(<callback>, <delay>);
  - Conjugando com um pedido Ajax, temos:

```
var xhttp;
xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
    callbackFunction(this);
}
};
xhttp.open("GET", url, true);
xhttp.send();
```

 Uma função de Callback, terá de ser passada como parâmetro para outras funções assíncronas (que o permi tam).

#### JavaScript- Promises

- Promises em JavaScript são objetos que encapsulam uma operação assíncrona e notifica quando a execução é concluída.
- Pode ser utilizado de uma forma semelhante às Callbacks para execução de código após a conclusão da execução, no entanto, providencia dois métodos, um para quando a execução foi concluída com sucesso (then...) e outra para quando ocorreu um erro (catch...).

```
someAsyncOperation(someParams)
.then(function(result) {
      // Do something with the result
})
.catch(function(error) {
      // Handle error
});
```

#### JavaScript - Promises

 Não confundir com Try, catch. Até porque não vai considerar erro se for tratado anteriormente com o Promise.

#### JavaScript - Promises

console.log('Some error has occured');

• Criar Promise:

catch (function () {

}).

});

#### JavaScript - Promises

```
var promise = new Promise(function(resolve, reject) {
    reject('Promise Rejected')
})

promise
    .then(function(successMessage) {
        console.log(successMessage);
    })
    .catch(function(errorMessage) {
        //error handler function is invoked
        console.log(errorMessage);
    });
```

#### JavaScript – EventListener

```
element.addEventListener(event, function, useCapture);
element.addEventListener("click", function() { alert("Hello World!"); });
```

#### JavaScript – EventListener

```
element.addEventListener("click", myFunction);
function myFunction() {
  alert ("Hello World!");
}
```

#### JavaScript – EventListener

```
element.addEventListener("mouseover", myFunction);
element.addEventListener("click", mySecondFunction);
element.addEventListener("mouseout", myThirdFunction);
```

#### JavaScript – Manipulação da DOM

- DOM = **D**ocument **O**bject **M**odel. É a árvore de elementos de uma página.
- Podemos utilizar JavaScript para:
  - Alterar qualquer elemento ou atributo HTML na página;
  - Alterar qualquer estilo CSS da página;
  - Remover ou criar qualquer elemento ou atributo HTML na página;
  - Reagir a qualquer evento HTML da página;
  - Criar qualquer evento HTML na página.

#### JavaScript – Manipulação da DOM

```
<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>
document.getElementById(id)
                                                       Find an element by element id
document.getElementsByTagName(name)
                                                       Find elements by tag name
document.getElementsByClassName(name)
                                                       Find elements by class name
element.innerHTML = new html content
                                                  Change the inner HTML of an element
element.attribute = new value
                                                  Change the attribute value of an HTML element
element.style.property = new style
                                                  Change the style of an HTML element
element.setAttribute(attribute, value)
                                                  Change the attribute value of an HTML element
```

Ver mais <a href="https://www.w3schools.com/js/js">https://www.w3schools.com/js/js</a> <a href="https://www.w3schools.com/js/js">httmldom</a> document.asp

#### JavaScript – Validação de dados

- JavaScript pode ser utilizado também para efetuar a validação de formulários e dados introduzidos pelo utilizador.
- Estas validações têm a particularidade de permitir verificar os dados introduzidos pelo utilizador sem existir a necessidade de um pedido ao servidor.

```
function validateForm() {
  var x = document.forms["myForm"]["fname"].value;
  if (x == "") {
    alert("Name must be filled out");
    return false;
  }
}
```

#### JavaScript – Validação de dados

- Podemos utilizer JavaScript para validar os nossos dados em termos de:
  - Preenchimento;
  - Comprimento dos dados;
  - Tipo de dados;
  - Formato;
  - Valores repetidos;
  - Etc...

## JavaScript – Validação de dados

- Adicionalmente, HTML5 fornece-nos um novo conceito chamado "constraint validation", que nos permite validar por:
  - Atributos do input HTML
  - Pseudo-selectors de CSS
  - Métodos e Propriedades da DOM

```
<form>
    <label for="choose">Would you prefer a banana or a cherry?</label>
    <input id="choose" name="i_like" required pattern="banana|cherry">
        <button>Submit</button>
</form>
```

Ver mais <a href="https://www.w3schools.com/js/js">https://www.w3schools.com/js/js</a> validation.asp

```
input:invalid {
  border: 2px dashed red;
}
input:valid {
  border: 2px solid black;
}
```

```
var email = document.getElementById("mail");

email.addEventListener("input", function (event) {
   if (email.validity.typeMismatch) {
     email.setCustomValidity("I expect an e-mail, darling!");
   } else {
     email.setCustomValidity("");
   }
});
```

 Podemos utilizar JavaScript para armazenar dados em armazenamento local (no browser). Podemos por exemplo armazenar os dados como:

```
    Cookies; document.cookie = "username=John Doe";
    LocalStorage; localStorage.setItem("lastname", "Smith");
    SessionStorage.setItem("lastname", "Smith");
```

- Cookies Armazenamento de dados no computador local em ficheiros de texto pequenos;
- LocalStorage Armazena pares chave-valor, sem data de expiração no browser;
- SessionStorage Armazena pares chave-valor no browser durante a sessão do utilizador.

- Cookies
  - Definimos uma cookie em texto:

```
document.cookie = "cookiename=cookievalue"
```

• Podemos definir uma data de expiração:

```
document.cookie = "cookiename=cookievalue; expires= Thu, 21 Aug 2014 20:00:00 UTC"
```

• E até um caminho para organizar em pastas:

```
document.cookie = "cookiename=cookievalue; expires= Thu, 21 Aug 2014 20:00:00 UTC; path=/ "
```

• Obter uma Cookie:

```
var x = document.cookie
```

 Eliminar uma Cookie – Devemos definir o conteúdo como vazio e a data de expiração anterior à data atual:

```
document.cookie = "cookiename= ; expires = Thu, 01 Jan 1970 00:00:00 GMT"
```

- LocalStorage
  - Definir um elemento chave-valor:

```
window.localStorage.setItem('melhor Disciplina', 'Programação e Serviços Web');
```

• Obter um elemento:

```
window.localStorage.getItem("melhor Disciplina");
```

Remover um elemento:

```
window.localStorage.removeItem('melhor Disciplina');
```

• Limpar o armazenamento:

```
window.localStorage.clear()
```

• Obter valor da chave por index:

```
window.localStorage.key(0)
```

- LocalStorage Características
  - Não é seguro usar localStorage para guardar dados sensíveis;
  - Em qualquer browser, o localStorage é limitado a guardar apenas 5Mb de dados. (Ainda assim, no caso das Cookies é 4Kb);
  - O uso do localStorage é síncrono;
  - LocalStorage só armazena String, se quisermos armazenar objetos terão de ser manipulados (JSON string por exemplo);
  - Não pode ser usado por web workers.

#### SessionStorage

- Igual à LocalStorage, mas os dados apenas são guardados durante uma sessão.
- Uma sessão dura enquanto o browser estiver aberto e mantem-se durante atualizações de página.
- Uma sessão é criada sempre que abrimos uma nova janela/tab no browser, e é eliminada (e limpar o SessionStorage) sempre que fecharmos uma janela/tab.
- A sintaxe é igual à Local Storage.