JavaScript: É uma linguagem de programação para execução do lado do cliente. Depende da capacidade e permissões junto do browser. A linguagem é dinâmica, ou seja, não requer uma compilação. É uma linguagem com tipagem fraca (não é necessário informar o tipo das varáveis).
Qual o propósito do JavaScript? Permite adicionar uma camada de interatividade a um documento Web, verificar campos e estado de submissão de um form, mostrar ou esconder conteúdo com base num clique.

```
<script>
  … JavaScript code goes here
</script>
```

Modo Embebido: inserido no conteúdo HTML e delimitado pelas tags do elemento script.
Modo externo: Recurso externo ao documento HTML, referido no atributo src do elemento script, como apontador para um ficheiro autónomo .js.

```
<script src="my_script.js"></script>
```

O elemento script pode estar em qualquer ponto do documento, mas é comum estar dentro do elemento head (especialmente se for necessário executar algo antes do carregamento total do corpo do documento) ou antes da etiqueta </body>

Conceitos fundamentais:
Cada variável tem um nome e um valor. Com a variável é possível referir várias vezes o mesmo valor, ao longo do programa.
Tipos de dados:
var foo; variável undefined. Está declarada, mas ainda não recebeu valor.
var foo = null; Valor nulo. Palavra reservada para este significado especial.
var foo = 5; Number. Sempre que se atribui um número à variável.
var foo = “five”. String
var foo = true; Boolean.
var foo = [5, “five”, “5”]. Array.
var person = {firstName: “John”, age:20}. Objeto.
Uma function() é um tipo complexo que representa uma função. Para avaliar qual o tipo de uma expressão fazer “typeof 3.14”.

Janelas de Diálogo: Alert box: alert(“Exemplo”); Confirm Box: window.confirm(“sometext”); Prompt Box: window.prompt(“xx”,”tt”);

Operadores para comparação fora do comum: === é idêntico a “ser igual e possuir o mesmo tipo”.

Ciclos for e iteração em arrays: for(x of arr) x itera sobre os elementos do array.

Funções built-in: Funções previamente disponíveis no JavaScript. Ex: alert().
Funções definidas pelo programador: São escritas no código fonte, com a palavra reservada function. Exemplo function name().

Variável de âmbito local é uma variável que é usada apenas dentro do código de uma função, ou bloco.
Variável de âmbito global é uma variável acessível desde qualquer script ou zona de código na página.
Let declara uma variável, normalmente para uso localizado numa expressão ou bloco curto. O let não permite redeclarar variáveis no mesmo âmbito.

Tratamento de Eventos:
Um evento é uma ação que pode ser detetada com JS e desencadear a execução de scripts. Os eventos são tratados por event handlers, ou controladores de eventos. Exemplo: onload, onclicl, onmouseover, onerror.
Os controladores de eventos podem ser aplicados de três maneiras:

1. Atributo de um elemento html.
2. Propriedade onclick do objeto.
3. Função addEventListener().

```
<body onclick="myFunction();">
/* myFunction runs when the user clicks anything within 'body' */
window.onclick = myFunction;
/* myFunction will run when the user clicks anything within the browser window */
window.addEventListener("click", myFunction);
```

Note-se a ausência do prefixo “on” na sintaxe desta abordagem.

DOM (Document Object Model) é uma interface para consultar e manipular o conteúdo do documento web. Estrutura de objetos com o mapeamento do documento, com os vários elementos e um conjunto de métodos para interagir com eles.
Abaixo, a variável foo recebe o conteúdo HTML do elemento com id=”begginer”:
var foo = document.getElementById("beginner").innerHTML;
O objeto document representa a página/documento web atual.
getElementById é um método que devolve o elemento com id “beginner”
.innerHTML representa o conteúdo HTML dentro daquele elemento.

querySelectorAll()
Devolve os nós com que correspondem a um selector de CSS
Exemplo: `document.querySelectorAll(".sidebar p");`

getAttribute()
Devolve o valor de um determinado atributo do nó atual (sobre o qual este método é invocado)

getElementsByTagName()
Devolve o conjunto (**array**) com todos os elementos com a tag
Exemplo: `document.getElementsByTagName("p");`

getElementById()
Devolve o elemento cujo atributo `id` tem aquele valor
Exemplo: `document.getElementById("special");`

getElementsByClassName()
Devolve os elementos que estão associados aquela classe, num objeto HTMLCollection, cujo comprimento está na propriedade `length`

getElementsByName()
Devolve todos os elementos com o atributo name e em que o valor desse atributo é o argumento do método

Manipulação de Nós

Métodos e propriedades built-in para modificar o conteúdo dos nós:

setAttribute()

Define o valor para um determinado atributo:

```
bigImage.setAttribute("src", "newimage.jpg");
```

innerHTML

Define o conteúdo interior ao elemento (possivelmente com etiquetas HTML com outros elementos):

```
introDiv.innerHTML = "<p>This is the intro text.</p>"
```

style

O estilo de formatação com propriedades CSS:

```
document.getElementById("intro").style.backgroundColor = "#000;"
```

Adicionar e remover elementos

Através da API DOM é possível alterar a estrutura do documento Web, adicionando ou removendo nós:

- **createElement()**
- **createTextNode()**
- **appendChild()**
- **insertBefore()**
- **replaceChild()**
- **removeChild()**

Biblioteca JavaScript é uma coleção de funções e métodos previamente escritos e que podemos usar nos nossos scripts para simplificar a execução de tarefas.

Biblioteca jQuery: Como usar? Incluir conteúdo da biblioteca com um script de fonte externa em head. A sintaxe do código jQuery é diferente do JS base. Exemplo da sintaxe compacta com jQuery:

Alterar o conteúdo de um elemento (id="p2") quando clicar no botão com id="btn1":

```
$("#btn1").click(function(){
  $("#p2").html("Novo texto <b>após</b> clique!");
});
```

Sincrono vs. Assincrono. Comportamento síncrono, ao efetuar um pedido, a execução fica suspensa até a resposta chegar. Múltiplos pedidos serão enviados sequencialmente. Comportamento assíncrono, ao efetuar um pedido, a execução pode continuar de imediato, e a resposta será tratada quando (e se) chegar. O tratamento da resposta pode levar à atualização de um elemento específico, e não do documento inteiro. Podem esta múltiplos pedidos em execução ao mesmo tempo.

XMLHttpRequest é o objeto usado em JS na comunicação com servidores, nomeadamente para obter dados. Bastante flexível, pois permite atualizar parte da página sem carregar tudo, solicitar dados ao servidor (já após o carregamento da página), receber dados do servidor (já após o carregamento da página), enviar dados ao servidor em background (sem interromper demais a execução).

XMLHttpRequest

callback function, event handler

```
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
       // Typical action to be performed when the document is ready:
       document.getElementById("demo").innerHTML = xhttp.responseText;
    }
};
xhttp.open("GET", "filename", true);
     // open(method, url[, async[, user[, password]]])
xhttp.send();
     // sends the request to the server (asynchronous by default)
     // send() argumento opcional, em POST
```


AJAX (Asynchronous JavaScript and XML). Abordagem que combina HTML, CSS, JavaScript, DOM, XML, XSTL e XMLHttpRequest para troca de dados e atualização parcial e incremental da interface, para resultados mais rápidos e responsivos. Na abordagem em AJAX, um pedido, a página não tem de mudar, ou pode atualizar-se apenas um elemento em particular, poupando tempo e melhorando a experiência de utilização.

AJAX com XMLHttpRequest

```
<div id="demo">
<button type="button" onclick="loadXMLDoc()">Change Content</button>
</div>

<script>
function loadXMLDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML =
      this.responseText;
    }
  };
  xhttp.open("GET", "xmlhttp_info.txt", true);
  xhttp.send();
}
</script>
```

A biblioteca jQuery tem métodos para agilizar operações com AJAX. " load() " - loads data from a server and puts the returned data into the selected element.

" $.get() " – ler dados do servidor através de um pedido HTTP GET.

" $.post() " – ler dados do servidor através de um pedido HTTP POST.

Promise object: Representa o estado de execução consumada, ou de falha, de uma operação assíncrona. Permitem o encadeamento (chaining) de callbacks e o tratamento de erros/exceções. Por outras palavras, uma promise representa a eventual conclusão ou falha de uma operação assíncrona. É útil quando se está lidando com operações que levam algum tempo para serem concluídas. Uma promise pode estar em um dos três estados: pending, fulfilled (resolvida) ou rejected. A principal vantagem das Promises é que elas tornam o código mais legível e fácil de entender, especialmente quando se lida com várias operações assíncronas encadeadas.

```
const minhaPromise = new Promise((resolve, reject) => {
    // Código assíncrono aqui
    const operacaoBemSucedida = true;

    if (operacaoBemSucedida) {
        resolve("Operação bem-sucedida!");
    } else {
        reject("Houve um erro na operação.");
    }
});
```

Web Worker: Permite execução em Thread separada. Util para operações demoradas, libertando a thread principal de controlo da página web. Por outras palavrás, são scripts executados em segundo plano, fora do thread principal do navegador. Eles permitem a execução de operações intensivas sem afetar a responsividade da interface do utilizador.

Push Notifications: São comunicações Servidor → Browser. Mesmo sem interação do utilizador com o browser, há um canal onde o servidor pode escrever, no momento em que for oportuno, e esses dados chegam ao browser e permitem atualizar automaticamente uma interface. Muito úteis para transmissão direta da aplicação para o browser, mesmo quando o utilizador está parado. Exemplo: num chat, enviar aviso de uma nova mensagem a todos os utilizadores.