FONTE

Definição de alto nível

JavaScript é uma linguagem de programação que permite implementar itens complexos em páginas web, toda vez que uma página da web faz mais do que simplesmente mostrar a você informação estática, mostrando conteúdo que se atualiza em um intervalo de tempo, mapas interativos ou gráficos 2D/3D animados, etc; O JavaScript provavelmente está envolvido.

O que ele realmente faz?

APIs (Application Programming Interfaces)

APIs são um conjuntos prontos de blocos de construção de código que permitem que um dev implemente programas que sejam difíceis ou impossíveis de implementar.

Elas geralmente se dividem em duas categorias

APIs de navegadores: já vem implementadas no navegador, e são capazes de expor dados do ambiente do computador, ou fazer coisas complexas e úteis. Por exemplo:

A **API DOM(Document Object Model)** permite você manipular HTML e CSS, removendo e mudando HTML, aplicando dinamicamente novos estilos para a sua página, etc. Toda vez que uma janela pop-up aparece em uma página, ou vê algum novo conteúdo sendo exibido, isso é o DOM.

A **API de Geolocalização** recupera informações geográficas. É assim que o Maps consegue encontrar sua localização e colocar em um mapa.

As **APIs Canvas e WebGL** permitem a você criar gráficos 2D e 3D animados. Há pessoas fazendo algumas coisas fantásticas usando essas tecnologias web — veja Chrome Experiments e webglsamples.

APIs de áudio e vídeo como HTMLMediaElement e WebRTC permitem a você fazer coisas realmente interessantes com multimídia, tanto tocar música e vídeo em uma página da web, como capturar vídeos com a sua câmera e exibir no computador de outra pessoa

APIs de terceiros não estão implementados no navegador automaticamente, e você geralmente tem que pegar seu código e informações em algum lugar da Web. Por exemplo:

A **API do Twitter** permite a você fazer coisas como exibir seus últimos tweets no seu website.

A **API do Google Maps** permite a você inserir mapas customizados no seu site e outras diversas funcionalidades.

O que JavaScript está fazendo na página web?

Quando você carrega uma página web em um navegador. Quando você carrega uma página web no seu navegador, você está executando seu código (na ordem em que foi escrito, de cima para baixo) dentro de um ambiente de execução (a guia do navegador).

O JS pode ser usado para modificar o HTML e por meio da API DOM, atualizar uma interface do usuário, funciona como um css no JS (var.style.backgroundColor = “green”).

Segurança do navegador

Cada guia do navegador tem seu próprio espaço (ambientes de execução em termos técnicos) para executar código, isso significa que na maioria dos casos o código em cada guia está sendo executado separadamente, e o código em uma guia não pode afetar diretamente o código de outra guia ou de outro website.

Código interpretado x compilado

JS é uma linguagem interpretada, o código é executado de cima para baixo e o resultado da execução do código é imediatamente retornado (compilação just-in-time). Não é preciso transformar(compilar) o código em algo diferente para ser executado, como é o caso das linguagens C/C++. Essas linguagens precisam ser compiladas em linguagem Assembly, e depois são executadas pelo computador.

Lado do servidor x Lado do cliente (JS pode ser usado em ambos os lados)

Códigos do lado do cliente são executados no computador do usuário, quando uma página web é visualizada, o código do lado do cliente é baixado, executado e exibido pelo navegador.
Códigos do lado do servidor, por outro lado, são executados no servidor e o resultado da execução é baixado e exibido no navegador. Exemplos de linguagens do lado do servidor populares incluem PHP, Python, Ruby, e ASP.NET. E.

Código dinâmico x estático

Dinâmico é usado para descrever tanto o JS client-side como o server-side, se refere a quando a exibição de uma página web/app mostra coisas diferentes em circunstâncias diferentes, gerando novo conteúdo. Código server-side dinâmico gera um novo conteúdo no servidor, puxando dados de um banco de dados, enquanto que JavaScript client-side dinâmico gera novo conteúdo dentro do navegador do cliente, como criar uma nova tabela HTML com dados recebidos do servidor e mostrar a tabela em uma página web exibida para o usuário. Uma página web **estática** só mostra o mesmo conteúdo o tempo todo.

Como adicionar JavaScript na sua página?

Há duas maneiras de adicionar JS em um página web, de maneira interna ou externa.

Ex JS interno: <script> O JavaScript fica aqui </script>

Ex JS externo: <script src="script.js"></script>

Estratégias para o carregamento de scripts

Há um considerável número de problemas envolvendo o carregamento de scripts na ordem correta. Um problema comum é que todo o HTML de uma página é carregado na ordem em que ele aparece. Se você estiver usando Javascript para manipular alguns elementos da página (DOM), o código não irá funcionar caso o JavaScript for carregado e executado antes mesmo dos elementos HTML estarem disponíveis.

Meios de resolução

No JS **interno**: document.addEventListener("DOMContentLoaded", function() { … });
Isso é um *event listener*, que ouve e aguarda o disparo do evento "DOMContentLoaded" vindo do *browser*, que significa que o corpo do HTML está completamente carregado e pronto. O código JavaScript que estiver dentro desse bloco não será executado até que o evento seja disparado, o erro será evitado.

No JS **externo**: é usado um recurso moderno do JavaScript para resolver esse problema: Trata-se do atributo **defer**, que informa ao *browser* para continuar renderizando o conteúdo HTML uma vez que a tag **<script>** foi atingida.
Scripts que são carregados utilizando o atributo defer (veja abaixo) irão rodar exatamente na ordem em que aparecem na página e serão executados assim que o script e o conteúdo for baixado.

Neste caso, ambos script e HTML irão carregar de forma simultânea e o código irá funcionar.

Nota: No caso externo, nós não precisamos utilizar o evento **DOMContentLoaded** porque o atributo **defer** resolve o nosso problema. Nós não utilizamos **defer** como solução para os exemplos internos pois **defer** funciona apenas com scripts externos.

Além do defer há um outro recurso que podemos usar para evitar o problema com o bloqueio de scripts, o async.

Os scripts que são carregados usando o atributo async irão baixar o script sem bloquear a renderização da página e irão executar imediatamente após o script terminar de ser disponibilizado. Nesse modo não há garantia nenhuma que os scripts carregados irão rodar em uma ordem específica, mas saberá que dessa forma eles não irão impedir o carregamento do restante da página. O melhor uso para o async é quando os scripts de uma página rodam de forma independente entre si e também não dependem de nenhum outro script. async deve ser usado quando houver muitos scripts rodando no *background*, e você precisa que estejam disponíveis o mais rápido possível.
Ex: <script async src="js/vendor/jquery.js"></script> / <script async src="js/script2.js"> /

</script><script async src="js/script3.js"></script>

Não dá pra garantir que o script. jquery.js carregará antes ou depois do script2.js e script3.js . Nesse caso, se alguma função desses scripts dependerem de algo vindo do jquery, ela produzirá um erro pois o jquery ainda não foi definido/carregado quando os scripts executarem essa função.