Por Morgana Maria



Exam 70-480: Programming in HTML5 with JavaScript and CSS3

**Guia de estudo para certificação**

**PLANO DE ESTUDO**

**Capítulo 1 – Implementar e manipular estruturas e objetos do documento**

**Objetivo 1.1: Criar a estrutura do documento usando HTML**

* + Estruturar a UI usando marcação semântica, incluindo marcações para mecanismos de pesquisa e leitores de tela, como Seção, Artigo, Nav, Cabeçalho, Rodapé e A parte; criar um contêiner de layout em HTML.
  + Criar a estrutura do documento;
  + Usar a semântica de mark-up do HTML5;
  + Otimização para buscadores;
  + Otimização para leitores;
  + Sumário do objetivo;
  + Revisão do objetivo.

**Objetivo 1.2: Escrever código que interaja com controles de UI**

* + Adicionar e modificar elementos HTML programaticamente; implementar controles de mídia; implementar tela HTML5 e elemento gráfico SVG.
  + Adicionar ou modificar elementos HTML;
  + Implementar controles de mídia;
  + Implementar gráficos com HTML5 <canvas> e SVG;
  + Sumário do objetivo;
  + Revisão do objetivo.

**Objetivo 1.3: Aplicar estilo a elementos HTML programaticamente**

* + Alterar a localização de um elemento; aplicar uma transformação; exibir e ocultar elementos.
  + Alterar a localização de um elemento;
  + Aplicar transformações;
  + Exibindo e escondendo elementos;
  + Sumário do objetivo;
  + Revisão do objetivo.

**Objetivo 1.4: Implementar APIs do HTML5**

* + Implementar APIs de armazenamento e API de Geolocation
  + Usar a Storage API;
  + Usar o AppCache API;
  + Usar o Geolocation API;
  + Sumário do objetivo;
  + Revisão do objetivo.

**Objetivo 1.5: Estabelecer o escopo de objetos e variáveis**

* + Definir variáveis do tempo de vida; manter objetos fora o namespace global; usar a palavra-chave “este” para fazer referência a um objeto que disparou um evento; estabelecer escopo de variáveis localmente e globalmente
  + Estabelecer o ciclo de vida de variáveis e escopos de variáveis;
  + Evitar usar namespace global;
  + Utilizando “this”;
  + Sumário do objetivo;
  + Revisão do objetivo.

**Objetivo 1.6: Criar e implementar objetos e métodos**

* + Implementar objetos nativos; criar objetos personalizados e propriedades personalizadas para objetos nativos usando protótipos e funções; herdar de um objeto; implementar métodos nativos e métodos personalizados
  + Implementar objetos nativos;
  + Criar objetos customizados e propriedades customizadas usando protótipos;
  + Implementar herança;
  + Implementar métodos nativos e criar métodos customizados;
  + Sumário do objetivo;
  + Revisão do objetivo.

**Capítulo 2 - Implementar fluxo de programa**

**Objetivo 2.1: Implementar fluxo de programa**

* + Iterar em coleções e itens da matriz; gerenciar decisões do programa usando instruções switch, if/then e operadores; avaliar expressões
  + Analisar expressões;
  + Trabalhando com arrays;
  + Implementando tipos especiais de array;
  + Usando métodos avançados de array;
  + Implementação de controle de fluxo iterativo;
  + Sumário do objetivo;
  + Revisão do objetivo.

**Objetivo 2.2: Criar e gerir um evento**

* + Manipular eventos comuns expostos por DOM (OnBlur, OnFocus, OnClick); declarar e manipular eventos com bolhas; manipular um evento usando uma função anônima
  + Usar eventos, incluindo gerenciar o evento por uso de funções anônimas;
  + Gerenciar eventos DOM (Documento Object Model), ex: onBlur, onFocus, onClick e sua convenção de nomenclatura “on\_” + Nome do evento;
  + Criar eventos customizados;
  + Sumário do objetivo;
  + Revisão do objetivo.

**Objetivo 2.3: Implementar manipulação de exceções**

* + Definir e responder aos códigos de erro; lançar uma exceção; solicitar verificações de nulos; implementar blocos try-catch-finally
  + Implementar try-catch-finally, incluindo setar e responder a códigos e lançar exceções;
  + Checar valores nulos;
  + Sumário do objetivo;
  + Revisão do objetivo.

**Objetivo 2.4: Implementar callback**

* + Receber mensagens do API WebSocket do HTML5; usar o jQuery para fazer uma chamada AJAX; ligar um evento; implementar um retorno de chamada usando funções anônimas; manipular o ponteiro “this”
  + Implementar comunicação bi-direcional com WebSocket API;
  + Tornar páginas dinâmicas com JQuery e Ajax;
  + “Ligar” um evento com jQuery;
  + Implementar call-back com funções anônimas;
  + Usar o ponteiro this;
  + Sumário do objetivo;
  + Revisão do objetivo.

**Objetivo 2.5: Criar um processo de trabalho da Web**

* + Iniciar e parar um web worker; passar dados para um web worker; configurar tempos limite e intervalos em um web worker; registrar um ouvinte de eventos para o web worker; limitações de um web worker
  + Iniciando com WebWorker;
  + Criar um processo WebWorker com o WebWorker API;
  + Usar um WebWorker;
  + Entendendo as limitações do WebWorker;
  + Configurar timeouts e intervalos;
  + Sumário do objetivo;
  + Revisão do objetivo.

**Capítulo 3 - Acessar e proteger dados**

**Objetivo 3.1: Validar entrada de usuário usando elementos do HTML5**

* + Escolher os controles apropriados com base nos requisitos; implementar tipos de entrada e atributos de conteúdo do HTML para coletar entrada do usuário
  + Escolher os controles HTML5 de entrada de dados de usuário;
  + Implementar atributos contente;
  + Sumário do objetivo;
  + Revisão do objetivo.

**Objetivo 3.2: Validar entrada de usuário usando JavaScript**

* + Avaliar uma expressão regular para validar o formato de entrada; validar que você está obtendo o tipo certo de tipo de dados usando funções internas; evitar injeção de código.
  + Avaliando expressões regulares;
  + Avaliando expressões regulares em JS;
  + Validando dados com funções nativas;
  + Prevenindo code injection;
  + Sumário do objetivo;
  + Revisão do objetivo.

**Objetivo 3.3: Dados de consumo**

* + Consumir dados JSON e XML; recuperar dados usando serviços Web; carregar dados ou obter dados de outras fontes usando XMLHTTPRequest
  + Consumir JSON e XML usando webservices;
  + Usar o objeto XMLHTTPRequest;
  + Sumário do objetivo;
  + Revisão do objetivo.

**Objetivo 3.4: Serializar, desserializar e transmitir dados**

* + Manusear dados binários; manusear dados de texto como JSON e XML; implementar o método de serialização jQuery, usar o Form.Submit; analisar dados; enviar dados usando XMLHTTPRequest; limpar a entrada usando URI/codificação de formulário
  + Enviar dados usando XMLHttpRequest;
  + Serializar e deserializar dados JSON;
  + Serializar e deserializar dados binários;
  + Sumário do objetivo;
  + Revisão do objetivo.

**Capítulo 4 - Usar CSS3 em aplicativos**

**Objetivo 4.1: Definir estilo de propriedade de texto de HTML**

* + Aplicar estilos para a aparência do texto; aplicar estilos para a fonte do texto, incluindo WOFF, @font-face, tamanho e fontes coadjuvantes; aplicar estilos no alinhamento de texto, espaçamento e recuo; aplicar estilos para a hifenação de texto; aplicar estilos para uma sombra de texto.
  + Aplicar estilos à aparência do texto;
  + Aplicar estilos à fonte do texto;
  + Aplicando estilos ao alinhamento, espaçamento e recuo do texto;
  + Aplicando estilos à hifenização de texto;
  + Aplicando estilos drop shadow no texto;
  + Sumário do objetivo;
  + Revisão do objetivo.

**Objetivo 4.2: Definir estilo de propriedade de caixa de HTML**

* + Aplicar estilos para alterar atributos de aparência, incluindo tamanho, bordas, arredondamento dos cantos das bordas, contorno, preenchimento e margem; aplicar estilos para alterar efeitos gráficos, incluindo transparência, opacidade, imagem em segundo plano, gradientes, sombra e recorte; aplicar estilos para estabelecer e alterar a posição de um elemento.
  + Aplicando estilos para alterar atributos de aparência;
  + Aplicando estilos para alterar efeitos gráficos;
  + Aplique estilos para estabelecer e alterar a posição de um elemento;
  + Sumário do objetivo;
  + Revisão do objetivo.

**Objetivo 4.3: Criar um layout de conteúdo flexível**

* + Implementar um layout usando um modelo de caixa flexível; implementar um layout de colunas múltiplas usando várias colunas; implementar um layout usando flutuação e exclusões de posição; implementar um layout usando alinhamento de grade; implementar um layout usando regiões, agrupamento e aninhamento.
  + Implementar um layout usando um modelo de box model flexivel;
  + Implementar um layout usando multi-column;
  + Implementar um layout usando position, floating, and exclusions;
  + Implementar um layout usando grid alignment;
  + Implementar um layout usando regions, grouping, and nesting;
  + Sumário do objetivo;
  + Revisão do objetivo.

**Objetivo 4.4: Criar uma UI animada e adaptável**

* + Animar objetos aplicando transições CSS; aplicar transformações 2-D e 3-D; ajustar UI com base em consultas de mídia, incluindo adaptações de dispositivos para formatos de saída, visores e representações; ocultar ou desabilitar os controles.
  + Animando objetos aplicando transições CSS;
  + Aplicando transformações 3D e 2D;
  + Ajustando a interface do usuário com base em consultas de mídia;
  + Ocultando ou desativando controles;
  + Sumário do objetivo;
  + Revisão do objetivo.

**Objetivo 4.5: Localizar elementos usando seletores de CSS e JQuery**

* + Escolher o seletor correto para referenciar um elemento; definir seletores de elemento, estilo e atributos; encontrar elementos usando pseudoelementos e pseudoclasses.
  + Definindo seletores de elemento, estilo e atributo;
  + Escolhendo o seletor correto para referenciar um elemento;
  + Localizando Elementos Usando Pseudoelementos e pseudo-classes;
  + Sumário do objetivo;
  + Revisão do objetivo.

**Objetivo 4.6: Estruturar um arquivo CSS usando seletores CSS**

* + Referenciar elementos corretamente; implementar herança; substituir herança usando !important; estilo de um elemento baseado em pseudoelementos e pseudoclasses.
  + Implementando herança;
  + Substituindo herança usando !important;
  + Estilizando um elemento baseado em pseudo-elementos e pseudo-classes;
  + Sumário do objetivo;
  + Revisão do objetivo.

**ANOTAÇÕES**

**Capítulo 1 – Implementar e manipular estruturas e objetos do documento**

**Objetivo 1.1: Criar a estrutura do documento usando HTML**

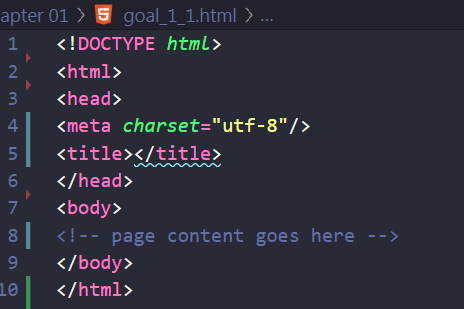
No HTML5 existe uma marcação semântica e não semântica é de suma importância para a interpretação dos buscadores, técnicas de SEO e otimização de leitores. O objetivo da estrutura de um documento é informar ao navegador como o conteúdo deve ser exibido. E é exatamente isso o que esse objetivo 1.1 ensina.

Nessa tabela mostro os elementos semânticos do HTML5 definido pela especificação. Certamente só deve ser usado os elementos necessários de acordo com o conteúdo proposto do documento.

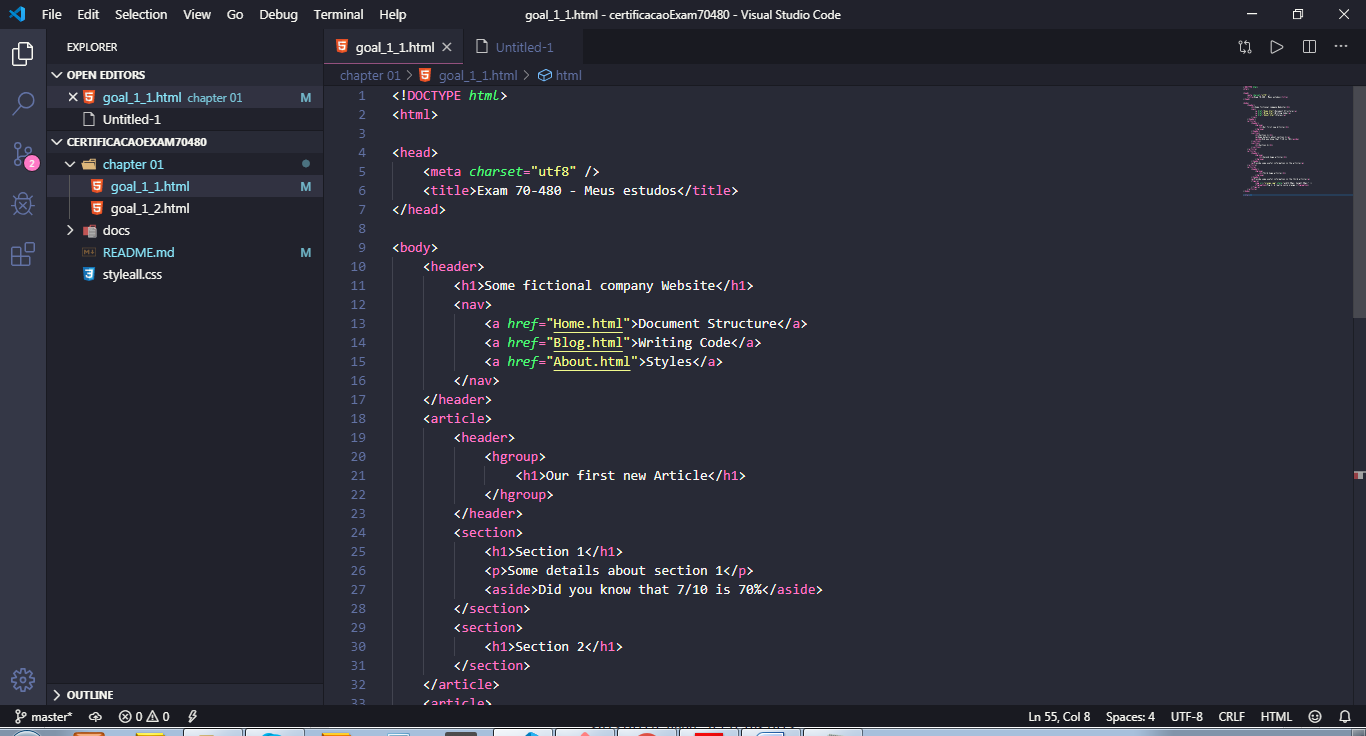
Elementos não semânticos são todas as tags que não representa nada, como por exemplo, <div> e <span>.

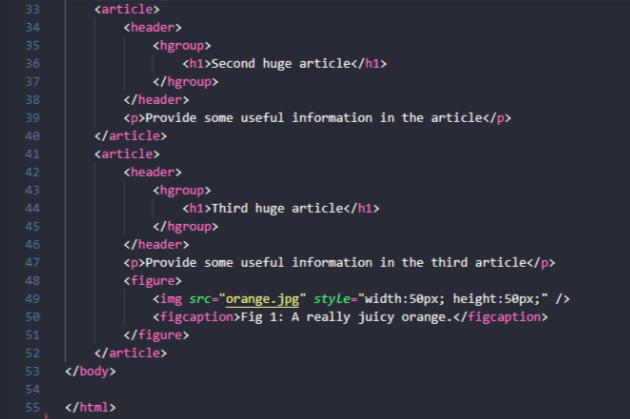
|  |  |
| --- | --- |
| **HTML5 element Semantic** | **Description** |
| <article> | Define áreas na página |
| <aside> | Define áreas de conteúdos menores fora do fluxo da página |
| <details> | Define detalhes adicionais que o usuário pode exibir ou ocultar |
| <figcaption> | Define o título do elemento de figura |
| <figure> | Define um conteúdo que contém uma figura, imagem, gráfico |
| <footer> | Define o rodapé de uma seção ou página |
| <header> | Define o cabeçalho de uma **seção** ou **página** |
| <main> | Define o conteúdo principal de um documento |
| <mark> | Define que um texto precisa ser destacado |
| <nav> | Define a navegação para outras páginas no site |
| <section> | Define conteúdo distinto de um documento ou área |
| <summary> | Define um cabeçalho visível para o elemento <details> assim como o <figcaption> funciona para o elemento <figure> |
| <time> | Define a data/hora |
| <hgroup> | Define um grupo de headings (de h1 até h6) |
| <progress> | Define o progresso de uma tarefa |

Uma estrutura básica HTML5:



Abaixo um documento em HTML5 utilizando boas práticas de semântica.

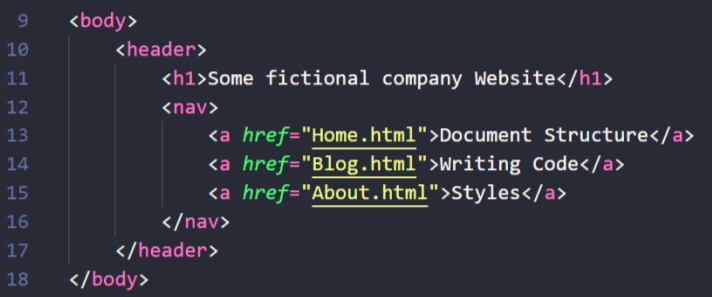




**O elemento <header>** não se limita apenas ao início da página. Ele fornece uma maneira semântica de declarar o cabeçalho para qualquer área da página. Podemos usar o <header> como um cabeçalho para um elemento <section> ou para um <article>. Dentro do <header> usamos qualquer marcação que atenda às necessidades para a área especifica. Como por exemplo <hgroup>, <h1> - <h6> e/ou <nav>.



**O elemento <nav>** define um conjunto de links de navegação. Destina-se apenas ao bloco principal de links de navegação.



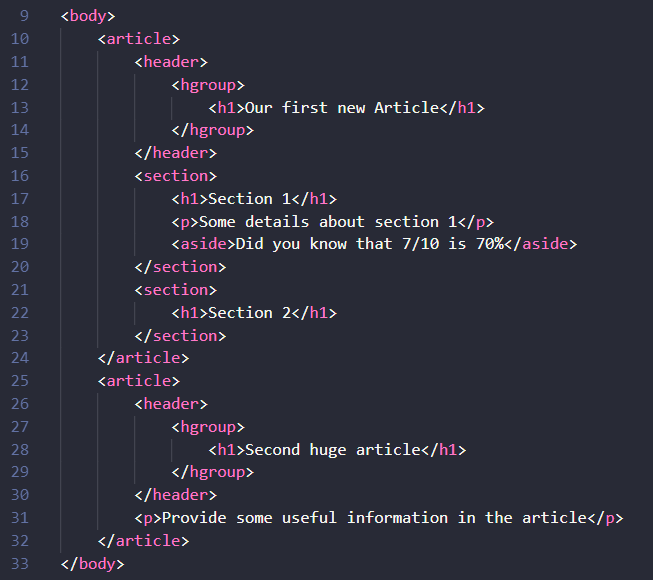
**O elemento <hgroup>** é um método semântico que organiza cabeçalhos e subtítulos. Esse elemento contém os elementos <h1> a <h6>. O <hgroup> pode ser utilizado em toda página desde que atenda esse proposito. A forma de renderização no navegador do <h1> dentro do <hgroup> é diferente quando o <h1> está fora do <hgroup>.



**O elemento <article>** representa um conteúdo completo e independente. Por exemplo, se no conteúdo da página vai ter um artigo, fórum, notícia ou uma postagem de blog, utiliza-se o <article>, por ser um elemento independente e que pode ser redistribuído e não perder o significado. Cada artigo ou postagem deve começar com a tag <article>.

Cada artigo adicionado ao documento neste exemplo abaixo representa uma parte independente do documento que pode estar totalmente contida.

Normalmente dentro do <article> terá um cabeçalho <header> e dentro do <header> um grupo de títulos. Logo em seguida inicia com a <section>.



**O elemento <section>** define seções no documento, geralmente o primeiro elemento dentro de uma <section> é um cabeçalho ou um grupo de cabeçalhos. De acordo com a imagem a cima.

**O elemento <aside>** define qualquer conteúdo que não se enquadre no fluxo principal ou no conteúdo principal da página atual - por exemplo, uma barra lateral, uma nota, um alerta ou um anúncio. O elemento <aside> não se coloca automaticamente em nenhum lado específico da página; serve apenas como uma maneira de definir semanticamente uma seção de texto ou gráfico como um aparte.

Na saída do navegador o <aside> não é tratado de forma especial, o elemento utilizado serve como garantia de otimização de leitores e buscadores.



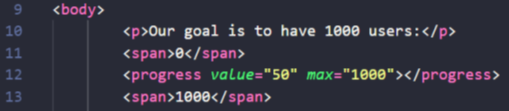
**O elemento <figure>** especifica conteúdo independente, como gráficos e figuras. Geralmente esta dentro de um <article> ou <section>. Sua posição é independente do fluxo principal da página e se removido, não deve afetar o fluxo do documento. Já o <figcaption> define uma legenda para o elemento <figure>, neste caso sendo utilizado só quando uma imagem precisar de uma legenda. O não uso dele na sequência do <figure> não afeta a renderização do navegador.

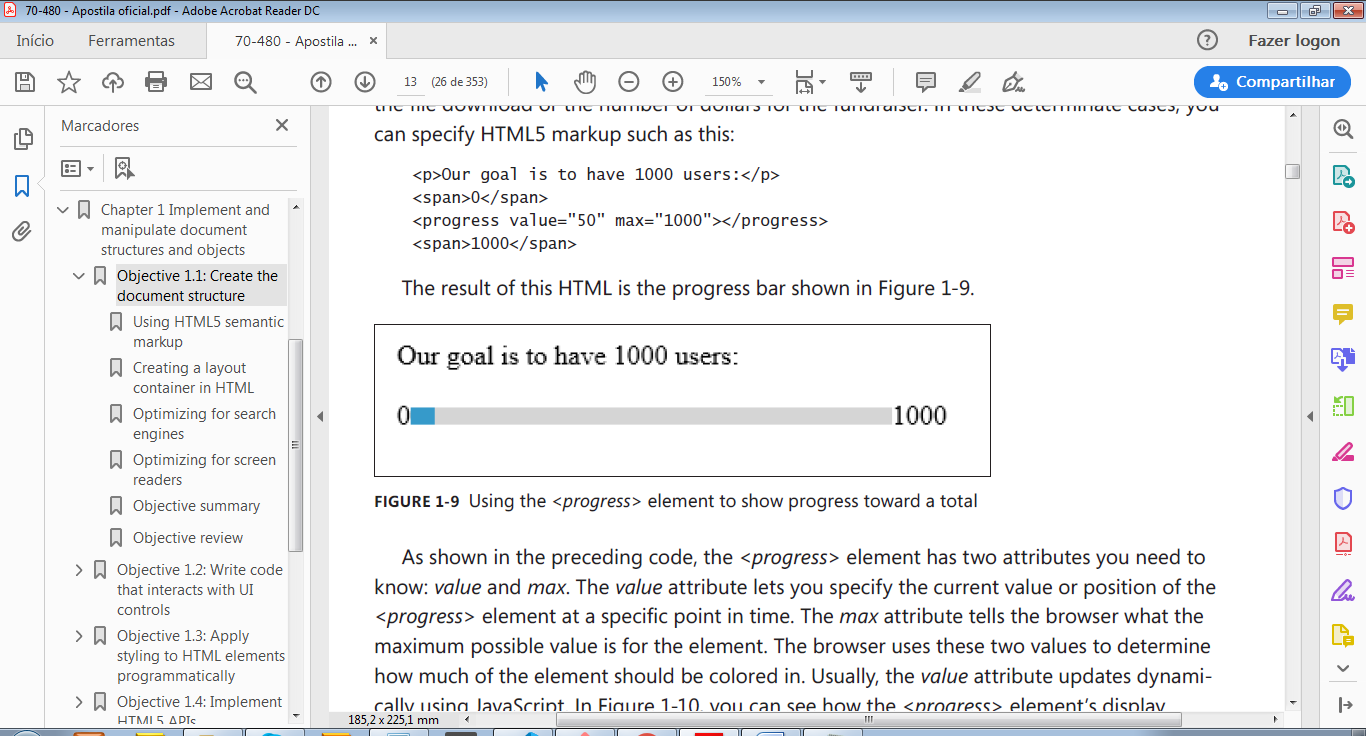


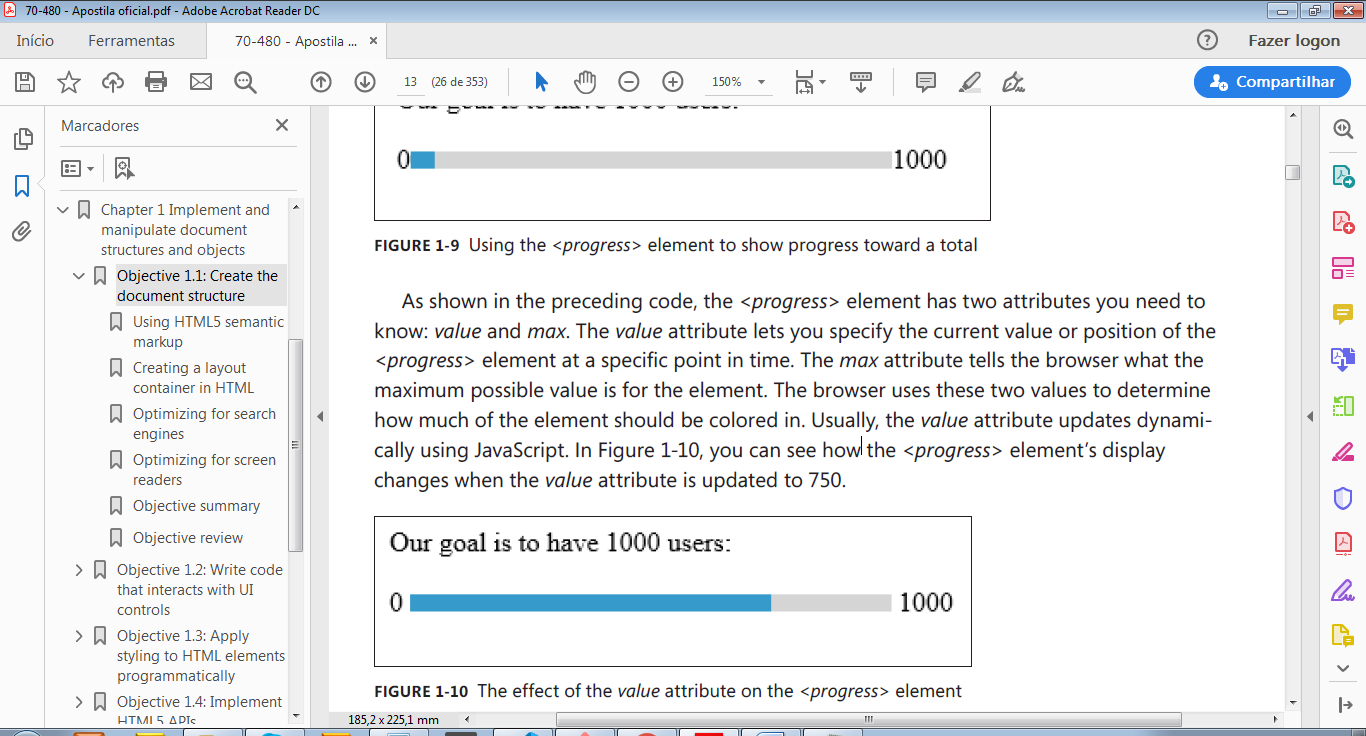
**O elemento <progress>** define o progresso de uma tarefa. Os dois tipos suportados de tarefas de progresso são determinados e indeterminados.

Usa-se uma tarefa de progresso determinado quando já souber os valores inicial e final. Para isso utilizo os atributos: value e max. O value permite especificar o valor atual ou a posição inicial do elemento <progress>. O atributo max informa o valor máximo para o elemento. Geralmente o atributo value é atualizado de forma dinâmica usando JavaScript.

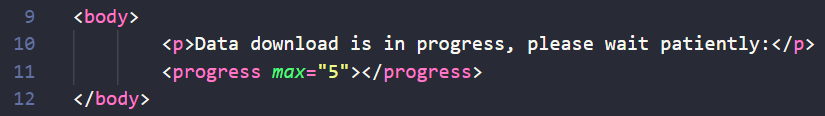
Sabendo os valores usa o código abaixo:

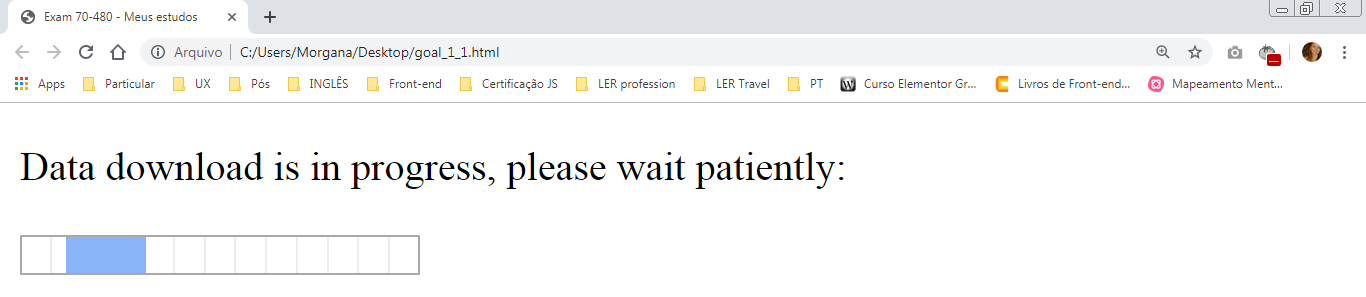


E que o resultado ficará assim: 

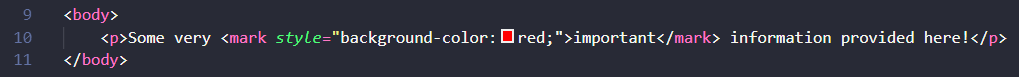
Se mudar o value de 50 para 750 vai ficaria assim:

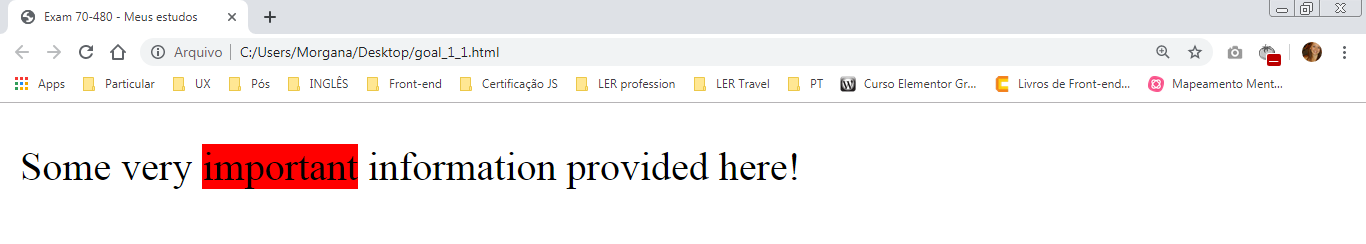
Usa-se uma tarefa indeterminada quando não sabemos os valores da tarefa ou quanto tempo levará para ser concluída. Por exemplo, quando você faz alguma ação no site e espera o resultado aparece um loading enquanto o processo é feito. Para isso, utilizamos o <progress> mas remove o atributo value. Quando não especificamos o atributo value o navegador vai deduzir que é uma tarefa indeterminada.





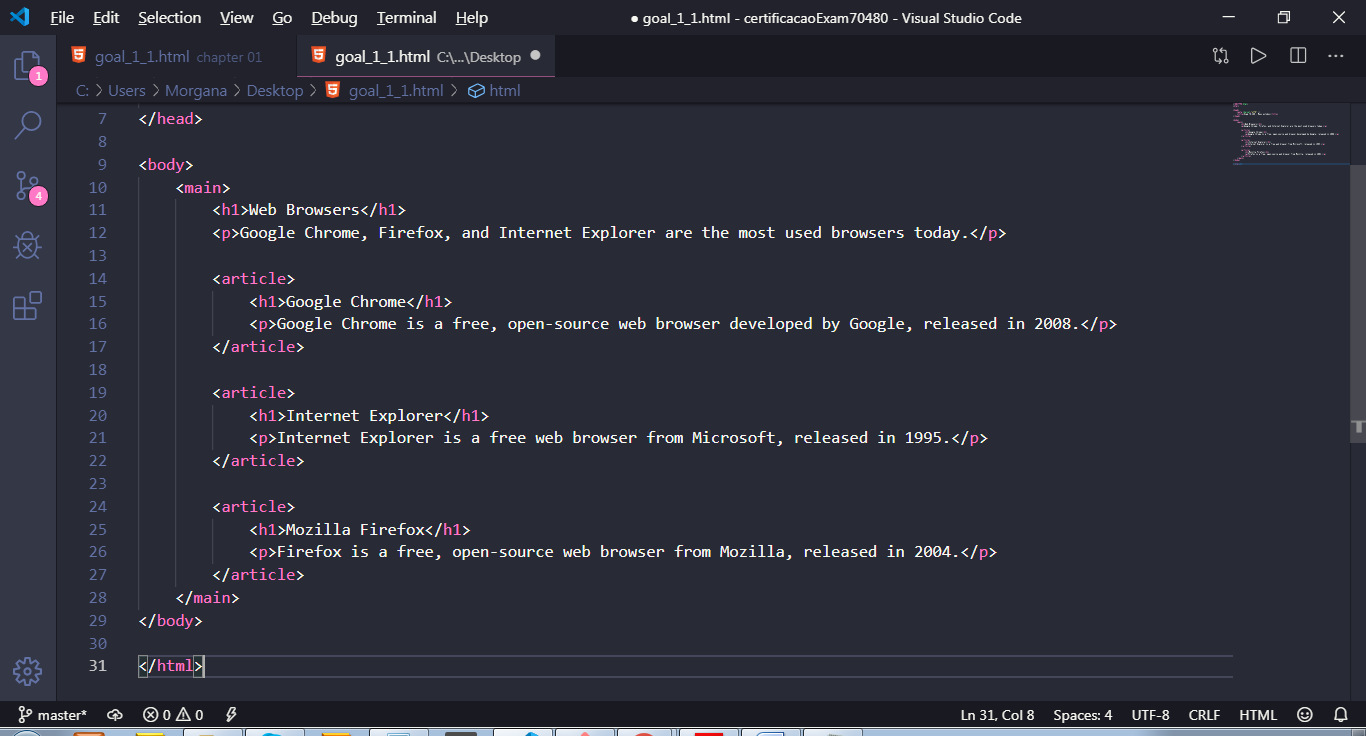
**O elemento <mark>** basicamente funciona como um marca-texto. Ao envolver um texto em um elemento <mark> e fornecer um atributo de cor de plano de fundo ao elemento de estilo vai obter o destaque desejado.



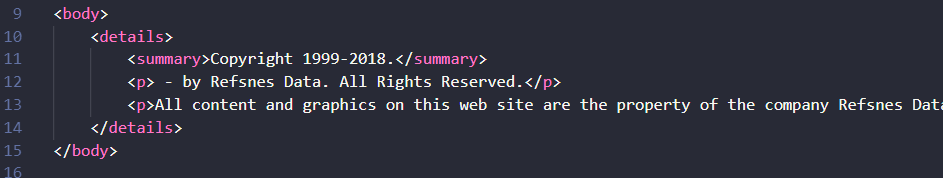


**O elemento <main>** define o conteúdo principal de um documento. O conteúdo dentro do elemento <main> deve ser exclusivo para o documento. Ele não deve conter nenhum conteúdo repetido em documentos como barras laterais, links de navegação, informações de direitos autorais, logotipos do site e formulários de pesquisa.

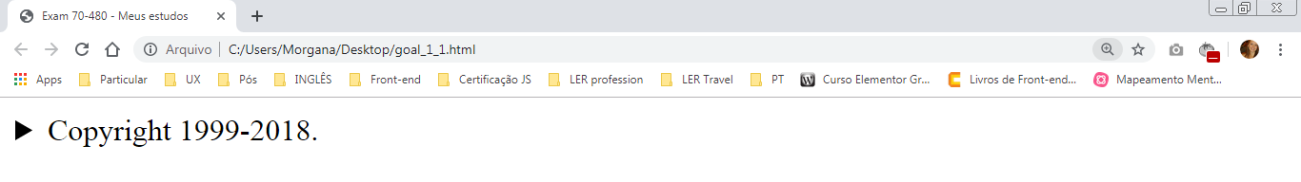
Não deve haver mais de um elemento <main> em um documento. O elemento <main> NÃO deve ser um descendente de um elemento <article>, <aside>, <footer>, <header> ou <nav>.

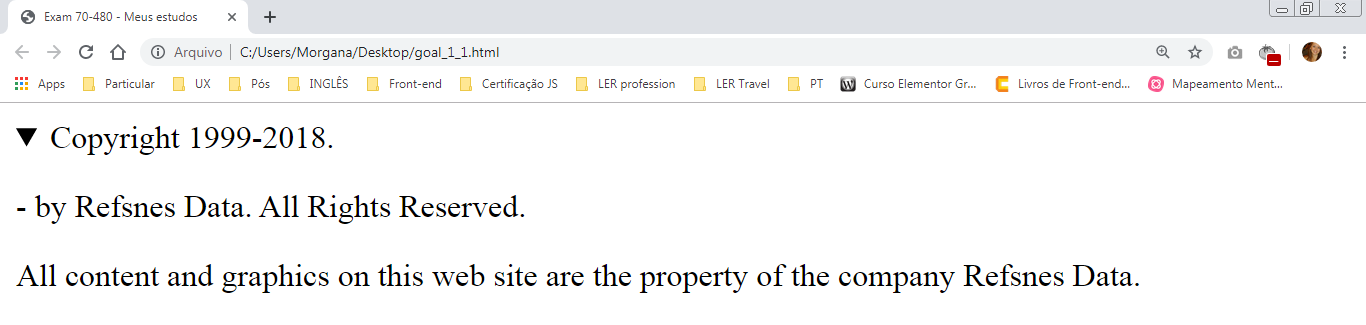


|  |  |
| --- | --- |
| **O elemento <details>** e **<sumary>:** O <details> especifica detalhes adicionais que o usuário pode exibir ou ocultar. O conteúdo do <details> não fica visível, a menos que seja definido. Podemos observar que o <sumary> é que define o cabeçalho para o elemento <details>, quando clica na setinha, o texto expande. |  |

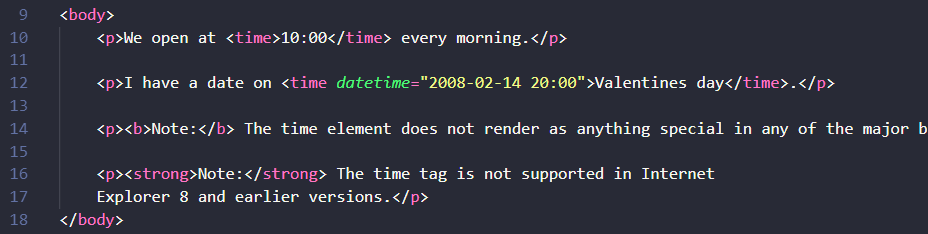


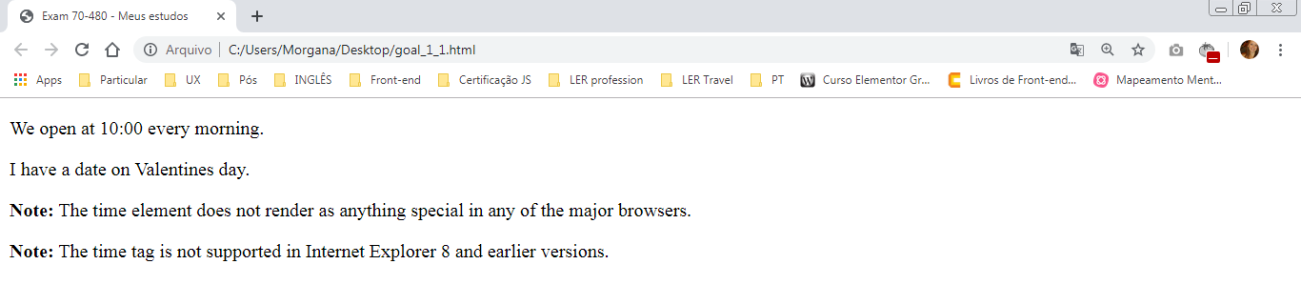
O <details> fechado.



O <details> aberto.

**O elemento <time>:** Define uma data/hora. Serve também para codificar o conteúdo de forma legível por máquinas e mecanismos de pesquisa possam produzir resultados de pesquisa mais inteligentes.





**Criando um container de layout em HTML**

Os dois métodos mais comuns de criação de um layout em HTML envolvem o uso de elementos <div> e <table>. Em ambos os casos, é mais provável que você ainda use CSS para ajudar no posicionamento e dimensionamento. Porém o principal problema com o uso de elementos <div> para estruturar o documento é sua incapacidade de transmitir significado semântico padrão para cada seção. O elemento <table> é muito flexível. Elementos adicionais como <thead> e <tfoot> fornecem uma abordagem mais semântica para rotular as células da tabela. A preocupação em usar a abordagem do elemento <table> é a natureza estática da estrutura. Para alterar a estrutura geral de um site que usa tabelas para layout, você precisa acessar todas as páginas e fazer as alterações.

**Otimizando para mecanismos de pesquisa (SEO)**

A otimização de mecanismo de busca “Search engine optimization” (SEO) é uma técnica usada para tornar os elementos do site facilmente detectáveis ​​e indexados pelos mecanismos de pesquisa. Eles usam os dados indexados para permitir que os usuários pesquisem basicamente qualquer coisa na Internet e recebam resultados relevantes. Com os elementos semânticos de HTML5, lembre-se de algumas coisas adicionais com relação aos mecanismos de pesquisa lógicos usados ​​para descobrir o que está nos sites.

O elemento <div> não fornece contexto sobre o que deve conter. Por isso os novos elementos semânticos do HTML5, você pode trazer um maior contexto do segmento da página, viabilizando um melhor desempenho aos mecanismos de pesquisa.

Os elementos <article> e <section> são os principais usados ​​pelo algoritmo SEO. Sabe-se que esses elementos contêm o corpo principal da página.

Dentro de cada elemento <article>, o mecanismo procura elementos como <hgroup> ou <h1> para obter o tópico principal do elemento <article> para indexação relevante.

**Otimizando para leitores de tela**

Os leitores de tela confiam no esboço do documento para analisar a estrutura e apresentar as informações ao usuário. Os programas de leitura de tela podem ler o texto da página e convertê-lo em áudio por meio de um algoritmo de conversão de texto em fala. Isso é útil para usuários que podem ter dificuldade para visualizar a página da web.

O HTML5 introduziu elementos semânticos para criar novas seções. Isso significa que todos os elementos <section>, <article>, <nav> e <aside> definem novas seções. A introdução dos elementos semânticos altera como o contorno do documento é criado.

Por exemplo você tem um conjunto de títulos:

|  |  |
| --- | --- |
| Os elementos de cabeçalho criam seções e subseções implícitas no documento. Isso ainda é válido no HTML5. No entanto, você não deve deixar o conjunto de seções da página em seção implícita, conforme apresentado pelos elementos do cabeçalho; em vez disso, você deve definir explicitamente as seções usando a semântica apropriada. Também é recomendado que os elementos <h1> sejam utilizados apenas em todo o  um documento HTML5. | Este HTML produz a mesma saída, como mostrado anteriormente na Figura 1-13. A diferença é que agora cada elemento <seção> cria uma nova seção da página em vez de confiar nos elementos do cabeçalho para criar as seções. Os leitores de tela podem analisar os elementos semânticos para criar o esboço do documento e, eventualmente, fornecer uma experiência ao usuário muito mais rica, devido à forma como o HTML5 permite que os designers de páginas da web organizem as páginas. |

**Resumo do objetivo**

* O HTML5 introduziu novos elementos semânticos para definir mais claramente as seções de uma página HTML. Esses elementos incluem <section>, <article>, <nav>, <header>, <footer>, <aside>, <progress>, <mark>, <figure> e <figcaption>.
* Os elementos em uma página HTML podem ter seu layout controlado quando incluídos estruturas internas, como elementos <div> e/ou tables HTML.
* Os elementos semânticos do HTML5 fornecem os mecanismos necessários para estruturar a página mais facilmente para acessibilidade via leitores de tela.
* Os mecanismos de pesquisa tiram vantagem da semântica HTML5, aproveitando o <article> elemento para determinar a finalidade da página.

**Objetivo 1.2: Escrever código que interaja com controles de UI**

Nesse objetivo vamos ver como adicionar e modificar elementos HTML via javaScript. Manipular o DOM com métodos. Implementar controles de mídia e elementos gráficos com SVG.

O DOM (Document Object Model) é uma representação da estrutura da sua página HTML com a qual você pode interagir programaticamente. O navegador produz um esboço com base na hierarquia HTML apresentada e exibe isso no navegador para o usuário. E por dentro o navegador constrói um DOM. A interface de programação de aplicativos (API) do DOM é exposta como objetos com propriedades e métodos, permitindo escrever código JavaScript para interagir com o HTML. Para permitir alterações na página, ao carregar o HTML, os navegadores carregam em memória uma estrutura de dados que representa cada uma das nossas tags no JavaScript.



**Selecionando itens no DOM**

O DOM é essencialmente uma coleção de nós organizados em uma árvore. Todos os nós estão relacionados entre si. O DOM tem uma hierarquia e é importante entender como você manipula o DOM através do código. Você pode acessar elementos DOM por meio de um objeto global fornecido pelo navegador, chamado document, ou pelos próprios elementos após obter uma referência.

Métodos disponíveis para selecionar elementos DOM

|  |  |
| --- | --- |
| **Métodos** | **Descrição de uso** |
| getElementById | Obtém um elemento individual na página por seu valor de atributo de identificação exclusivo |
| getElementsByClassName | Obtém todos os elementos que têm a classe CSS especificada aplicada a eles |
| getElementsByTagName | Obtém todos os elementos da página que possuem o nome de marca ou nome de elemento especificado. Por exemplo a tag <p>. |
| querySelector | Obtém o primeiro elemento filho encontrado que corresponde aos critérios de seletor CSS fornecidos |
| querySelectorAll | Obtém todos os elementos filhos que correspondem aos critérios do seletor CSS fornecidos |

**O método getElementById** retorna o elemento que estiver contendo o nome do ID passado. Como os IDs devem ser únicos, é um método muito útil para pegar apenas o elemento desejado.

|  |  |
| --- | --- |
| Retorna nulo se nenhum elemento da página tiver o ID especificado. Por exemplo também se no lugar do (“outerDiv”) não tivesse nenhuma informação o retorno seria *null*. | Quando executamos a página, vai abrir um alert e mostrar todo conteúdo HTML da <div> selecionado no DOM.  Retorna isto: |

**O getElementsByTagName** É um método você pode fazer algo com todos os elementos de um tipo específico - por exemplo, todos os elementos de parágrafo.

|  |  |
| --- | --- |
| O método getElementsByTagName() retorna uma coleção de todos os elementos que contém no documento com a tag especificada. | Esse é retorno do alerta JavaScript. Que exibe quantos itens foram retornados na lista quando o parâmetro NodeList é chamado através do objeto (x.length). Que retorna a quantidade de itens e a capacidade de acessar cada item individual. |

Da mesma maneira que você pode usar o método getElementsByTagName para obter todos os elementos do mesmo tipo, você pode usar o método **getElementsByClassName** para obter todos os elementos da mesma classe CSS. Isso é útil quando você possui muitos elementos com o mesmo estilo, mas talvez queira modificá-los em tempo de execução. Este método também retorna um NodeList.

|  |  |
| --- | --- |
|  |  |

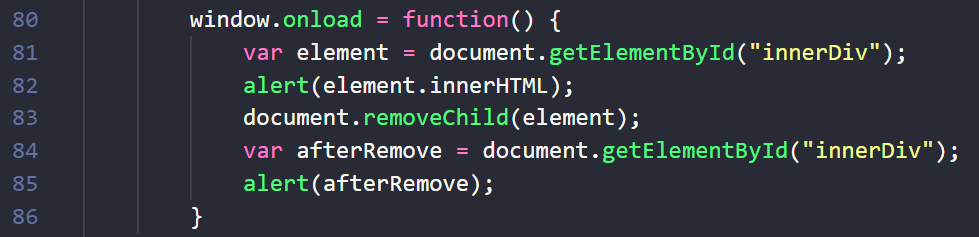
**O método querySelector()** retorna *o primeiro* elemento que corresponde ao seletor especificado. O **querySelectorAll()** retorna *todos* os elementos que correspondem aos critérios do seletor.

|  |  |
| --- | --- |
| //Para encontrar todas os elementos <p>  //de uma página, podemos usar desta forma          document.querySelectorAll("p"); | //Para encontrar um elemento único com ID,  //podemos usar desta forma          document.querySelector("#outerDiv"); |

**Alterando o DOM**

O objetivo de recuperar elementos do DOM é poder fazer algo com eles. Como manipular o DOM usando o código JavaScript para adicionar e remover itens.

Depois de ter uma referência a um elemento de contêiner, você pode dinamicamente: adicionar elementos filhos, remover elementos ou simplesmente ocultar elementos. Quando você remove um elemento do DOM, ele desaparece. Portanto, se você deseja tornar algo invisível para o usuário, mas poder usá-lo novamente mais tarde, basta ocultá-lo *usando o CSS* apropriado em vez de removê-lo. Por exemplo:



O código a cima funciona assim:

**1º** O primeiro alert mostra corretamente a propriedade innerHTML do innerDiv, mas o código nunca atinge o segundo alerta.

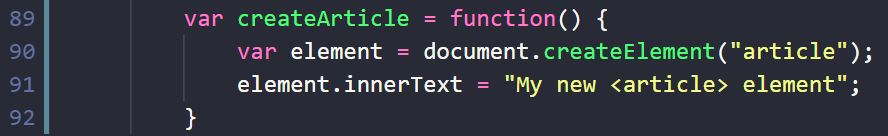
**2º** O método getElementById gera um erro porque o ID do elemento especificado não existe mais no documento.

**Adicionar elementos e removê-los do DOM.**

**Usando o método** document.createElement

Para criar um novo elemento HTML, usamos o document.createElement. O método recebe um único parâmetro - o nome do elemento que você deseja criar.

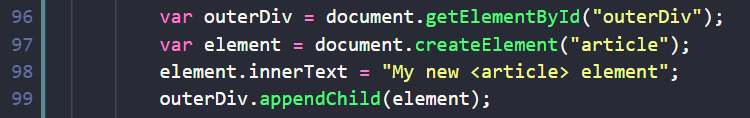
O código a seguir cria um novo elemento <article> mas não fica visível para ninguém neste momento; ele simplesmente existe no DOM para uso em sua página.

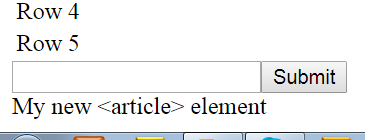


**Usando o método** appendChild

Você usa esse método para adicionar um novo elemento HTML à coleção de elementos filho pertencentes ao contêiner de chamada. O nó é adicionado ao final da lista de filhos que o nó pai já contém. O método appendChild existe no objeto de documento e em outros elementos de contêiner HTML. Retorna uma referência ao nó recém-adicionado.

Este exemplo anexa um novo elemento <article> ao outerDiv:





O método appendChild retorna uma referência ao novo elemento anexado aos elementos filho. Essa é uma boa maneira de garantir que você sempre tenha uma referência a um elemento para uso futuro, especialmente ao excluir elementos. Também permite simplificar ou reestruturar o código.

  // Esse código abaixo funciona da mesma forma que o de cima, porém de forma reestruturada.

        var element = document.getElementById("outerDiv").appendChild(document.createElement("article"));

        element.innerText = "My new <article> element restructure";

**Usando o método** insertBefore

Para inserir o novo elemento <article> em algum lugar mais preciso, o método insertBefore pode ser mais adequado. Este método usa dois parâmetros: o novo elemento em si e o nó antes do qual você deseja anexar o novo elemento. Por exemplo, para inserir seu novo <article> antes do elemento innerDiv, você usa o método getElementById para obter uma referência ao nó antes do qual você deseja inserior o <article> no DOM. Por exemplo:

 //Esse código insere seu novo artigo antes do elemento innerDiv

        var element = document.getElementById("outerDiv").insertBefore(

            document.createElement("article"),

            document.getElementById("innerDiv"));

            element.innerText = "My new <article> element";

As propriedades listadas na tabela a seguir obtêm referências aos nós mais comuns ao trabalhar com o DOM.

|  |  |
| --- | --- |
| **Propriedade** | **Descrição de uso** |
| childNodes | Uma coleção de todos os nós filhos do elemento pai. |
| firstChild | Uma referência ao primeiro nó filho na lista de nós filhos do nó pai. |
| lastChild | Uma referência ao último nó filho na lista dos nós filhos do nó pai. |
| hasChildNodes | Uma propriedade útil que retorna true se o elemento pai tiver algum nó filho. Uma boa prática é verificar essa propriedade antes de acessar outras propriedades, como firstChild ou lastChild. |

Para um exemplo dessas propriedades, inserir um elemento <article> como o primeiro elemento no elemento innerDiv:

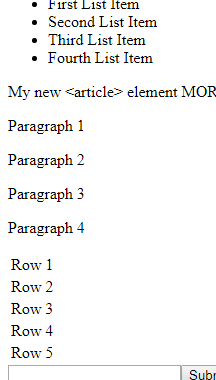
// A propriedade firstChild, indica que eu vou inserir o elemento <article>

// antes da div com id="innerDiv".

      var inner = document.getElementById("innerDiv");

        var element = inner.insertBefore(document.createElement("article"), inner.firstChild);

        element.innerText = "My new <article> element MORGANA";



Todo elemento que pode ter elementos filhos suporta toda essa funcionalidade; no entanto, se você tentar inserir elementos em um nó que não oferece suporte a nós filhos, como um <img>, por exemplo, o intérprete gera um erro em tempo de execução.

**Usando o método** removeChild

O método removeChild remove um nó filho do contêiner de chamada. O método removeChild retorna uma referência ao nó removido. Isso é especialmente útil se você planeja retornar esse nó ao DOM. Lembrando que, se não mantiver a referência retornada ao nó removido, não há como adicionar o elemento novamente sem recriá-lo completamente.

O exemplo abaixo remove o primeiro elemento <p> do seu elemento innerDiv.

O primeiro elemento <p> foi removido. No entanto capturei o elemento removido na variável p (var p = ...), poderá usá-lo mais tarde se desejar colocar o elemento <p> em outro lugar.

var innerDiv = document.getElementById("innerDiv");

var p = innerDiv.removeChild(document.getElementById("P1"));

**Usando o método** removeNode

O removeNode, utiliza um parâmetro booleano. Definir o parâmetro como true informa ao método para fazer uma remoção profunda, o que significa que todos os filhos também são removidos.

O código abaixo mostra isso:

var innerDiv = document.getElementById("innerDiv");

      innerDiv.removeNode(true);

**Usando o método** replaceNode e replaceChild.

replaceNode e replaceChild funcionam da mesma maneira que removeNode e removeChild em termos dos parâmetros que eles usam e quais elementos eles afetam. A diferença, no entanto, é que você pode substituir o elemento de destino por um elemento completamente novo.

**Implementando controles de mídias**

Novos elementos de mídia

|  |  |
| --- | --- |
| **Tag** | **Descrição** |
| <audio> | Define o conteúdo do som |
| <embed> | Define um container para uma aplicação externa (não HTML) |
| <source> | Define vários recursos de mídia para elementos de mídia (<video> e <audio>) |
| <track> | Define faixas de texto para elementos de mídia (<video> e <audio>) |
| <vídeo> | Define vídeo ou filme |

Os elementos <video> e <audio> funcionam com multimídia nativamente no navegador e com JavaScript.

É possível usar atributos e eventos para controlar o vídeo de forma declarativa, através de HTML estático ou dinamicamente, usando JavaScript. É necessário considerar o suporte ao navegador disponibilizando vários formatos de vídeo.

**Usando o elemento** <vídeo>

Principais atributos disponíveis para uso no elemento <video>.

|  |  |
| --- | --- |
| **Atributo** | **Descrição** |
| src | Este atributo especifica o vídeo a ser reproduzido. Pode ser um recurso local em seu próprio site ou algo exposto por meio de uma URL pública na Internet. |
| autoreplay | Este atributo diz ao navegador para começar a reproduzir o vídeo assim que ele for carregado. Se esse atributo for omitido, o vídeo será reproduzido apenas quando solicitado pelos controles do player ou JavaScript. |
| controls | Este atributo diz ao navegador para incluir seus controles de vídeo internos, como reproduzir e pausar. Se isso for omitido, o usuário não terá uma maneira visível de reproduzir o conteúdo. Você usaria a reprodução automática ou forneceria outro mecanismo através do JavaScript para reproduzir o vídeo. |
| height/width | Esses atributos controlam a quantidade de espaço que o vídeo ocupará na página. Omitir isso faz com que o vídeo seja exibido em seu tamanho nativo. |
| loop | Este atributo informa ao navegador para reproduzir continuamente o vídeo após a conclusão. Se esse atributo for omitido, o vídeo será interrompido após a reprodução completa. |
| poster | Este atributo especifica uma imagem a ser exibida no local alocado para o vídeo até que o usuário comece a reproduzir o vídeo. Use isso quando não estiver usando a reprodução automática. É muito útil para fornecer uma imagem ou arte profissional para representar o vídeo. Se for omitido, o pôster aparecerá no primeiro quadro do vídeo. |

Então, com esses atributos a cima ficaria mais ou menos assim:

    <video *src*="samplevideo.mp4" *controls* *poster*="picture.jpg" *height*="400" *width*="600">

    </video>

Com esse código a cima, estou dizendo que o navegador vai entender o código da seguinte forma:

- Deve exibir inicialmente uma imagem de pôster;

- Define que os controles padrão devem estar disponíveis;

- Define os parâmetros de altura e largura.

Como o loop não foi utilizado, isso que dizer que o vídeo não vai repetir automaticamente. Assim como o atributo autoreplay não foi especificado então o navegador não vai reproduzir de forma automática e o usuário que deve operar a reprodução com os controles ou se preferir invocar uma operação de reprodução com o JavaScript.

Os controles padrão fornecem:

* Um botão play / pause
* Um timer mostra a posição atual do vídeo
* O botão de controle de áudio
* Um controle que permite o vídeo em tamanho de tela cheia

É preciso garantir que o vídeo seja reproduzido em outros navegadores, disponibilizando outras opções de formato de vídeo para que o navegador escolha qual formato de vídeo reproduzir. Mesmo assim se não possuir todos os formatos de vídeo suportado é indicado que forneça uma alternativa ou informações que o navegador do usuário não suporta este vídeo.

**E para isso é feito o código abaixo.**

- Remove o atributo src do elemento <video> e adicionou elementos filhos <source>. O elemento <video> suporta vários elementos <source>, para que você possa incluir um para cada tipo de vídeo. Um navegador percorre os elementos <source> de cima para baixo e reproduz o primeiro que ele suportar.

- O elemento <object> serve para cobrir a possibilidade do navegador não ter suporte para o elemento <video>. Os navegadores que não suportam o elemento <video> ignoram o elemento completamente, mas mostram o elemento <object> que eles entendem.

- Por fim, o elemento <p> é o último recurso para fornecer pelo menos algumas informações aos usuários que um vídeo deveria estar sendo reproduzido aqui, mas que o navegador não o suporta.

    <video *controls* *height*="400" *width*="600" *poster*="picture.jpg">

        <source *src*="samplevideo.ogv" *type*="video/ogg"/>

        <source *src*="samplevideo.mp4" *type*="audio/mp4"/>

        <object>

         <p>Video is not supported by this browser.</p>

        </object>

    </video>

Métodos e propriedades no objeto <vídeo>

|  |  |
| --- | --- |
| **Atributo** | **Descrição** |
| play() | Reproduz o vídeo de sua posição atual. |
| pause() | Pausa o vídeo na sua posição atual. |
| volume | Permite ao usuário controlar o volume do vídeo. |
| currentTime | Representa a posição atual do vídeo. Aumente ou diminua esse valor para avançar ou retroceder no vídeo. |

    <script>

        // Inclusão de barra de controle de vídeo, via javaScript

        var video;

        window.onload = function() {

            video = document.getElementById("sampleVideo");

        }

        function play() {

            video.play();

        }

        function pause() {

            video.pause();

        }

        function back() {

            video.currentTime -= 10;

        }

    </script>

**Usando o elemento** <audio>

O elemento <audio> é essencialmente idêntico ao elemento <video>. Tem todos os mesmos atributos e os mesmos métodos. A única diferença real é como ela é exibida no navegador.

Como nenhum vídeo está disponível para exibição, o elemento <audio> não ocupa espaço na tela. No entanto, você pode mostrar os controles padrão - ou pode optar por não mostrar os controles padrão e criar seu próprio mecanismo para controlar o áudio, através de elementos personalizados da interface do usuário ou nos bastidores em JavaScript.

Aqui está um exemplo de como uma declaração <audio> se parece na sua página da web:

<audio *controls*>

<source *src*="sample.mp3" *type*="audio/mp3"/>

<source *src*="sample.ogg" *type*="audio/ogg"/>

<p>Your browser does not support HTML5 audio.</p>

</audio>

No exemplo à cima optei por usar os controles padrão que fornecem:

* Um botão play / pause
* Um contador inicial e tempo total do áudio
* Uma barra de progresso
* O botão de controle de áudio

**Implementando gráficos com HTML5 <canvas> e SVG**

A especificação HTML5 apresenta o elemento da página da web <canvas>, que fornece uma tela em branco na qual você pode desenhar dinamicamente. Você pode desenhar linhas, texto e imagens na tela e manipulá-los com JavaScript. O <canvas>elemento não possui habilidades de desenho próprias (é apenas um contêiner para gráficos).

Para adicionar uma tela à página, declara o <canvas> que é um elemento semelhante a <div> no entanto, é um contêiner para gráficos. Dentro do elemento deve conter os atributos id para ser referido no JavaScript e o width e height para determinar o tamanho do canvas. Se o navegador do usuário não suportar o elemento <canvas>, é só incluir o texto de fallback dentro do elemento <canvas>.

<canvas *id*="drawingSurface" *width*="600" *height*="400">

    Your browser does not support HTML5.

</canvas>

Para trabalhar com a *tela* através do código, você precisa obter uma referência a ela no seu JavaScript com um evento onload para encapsular seu código e fazer com que os gráficos sejam renderizados quando a página for carregada.

Só lembrando que uma página HTML pode ter vários elementos <canvas>.

window.onload = function() {

var drawingSurface = document.getElementById("drawingSurface");

var ctxt = drawingSurface.getContext("2d");

}

No código anterior, obtive uma referência ao elemento canvas seguido por uma referência a um contexto "2d". O método getContext() retorna um objeto que fornece os métodos e propriedades de API que você usa para desenhar na tela. Esta referência cobrirá as propriedades e métodos do getContext("2d").

O sistema de coordenadas que a tela usa, corresponde ao sistema de coordenadas da janela do navegador, com (0,0) no canto superior esquerdo. As coordenadas para desenhar na tela são sempre baseadas nas coordenadas da própria tela, onde o pixel superior esquerdo é (0,0).

No entanto, a posição da tela na janela do navegador é irrelevante para os métodos de desenho que você usa para desenhar na tela.

A tela fornece um sistema de coordenadas fixo (x, y) no qual o canto superior esquerdo da tela é (0,0). Nesse caso, o canto inferior esquerdo da tela é (0,400), o canto superior direito é (600,0) e o canto inferior direito é (600,400).

**Desenhando linhas** <canvas>

Você pode desenhar linhas na tela com o objeto getContext 2D que você está referenciando. O objeto de contexto fornece os seguintes métodos para desenhar linhas:

|  |  |
| --- | --- |
| **Método** | **Descrição** |
| beginPath | Redefine/Inicia um novo caminho de desenho |
| moveTo | Move o contexto para o ponto definido no método beginPath |
| lineTo | Define o ponto final de destino para a linha |
| Stroke | Traça a linha, tornando-a visível |

Para desenhar algum objeto na tela, precisa de algumas propriedades. Lembrando que cada propriedade abaixo tem seus valores. Por exemplo:

|  |  |
| --- | --- |
| **ESTILOS DE LINHAS** | |
| **Propriedades** | **Descrição** |
| lineCap | Define ou retorna o estilo nas extremidades de uma linha. Por exemplo apresentar uma linha com o limite arredondado usando o round. O limite é adicionado ao final da linha e seu comprimento corresponde ao que você definiu para a largura da linha. Neste exemplo, a linha tem uma largura de 5. Com um limite redondo definido na linha, o comprimento total da linha seria estendido em 5. |
| lineJoin | Define ou retorna o tipo de canto criado, quando duas linhas se encontram |
| lineWidth | Define ou retorna a largura da linha atual |
| miterLimit | Define ou retorna o comprimento máximo da mitra |

Exemplo de valores:

lineCap = “round”;

lineJoin = "miter";

|  |  |
| --- | --- |
| **CORES, ESTILOS E SOMBRAS** | |
| **Método** | **Descrição** |
| fillStyle | Define ou retorna a cor, gradiente ou padrão usado para preencher o desenho |
| strokeStyle | Define ou retorna a cor, o gradiente ou o padrão usado para os traçados |
| shadowColor | Define ou retorna a cor a ser usada nas sombras |
| shadowBlur | Define ou retorna o nível de desfoque para sombras |
| shadowOffsetX | Define ou retorna a distância horizontal da sombra da forma |
| shadowOffsetY | Define ou retorna a distância vertical da sombra da forma |

A propriedade **strokeStyle** permite alterar a cor da linha. Esta propriedade aceita todos os formatos de estilo comuns para especificar cores em HTML, incluindo valores hexadecimais ou cores nomeadas.

|  |  |
| --- | --- |
| **MÉTODOS** | |
| **Método** | **Descrição** |
| createLinearGradient() | Cria um gradiente linear (para usar no conteúdo da tela) |
| createPattern() | Repete um elemento especificado na direção especificada |
| createRadialGradient() | Cria um gradiente radial / circular (para usar no conteúdo da tela) |
| addColorStop() | Especifica as cores e as posições de parada em um objeto gradiente |

|  |  |
| --- | --- |
| **ESTILOS DE LINHAS** | |
| **Propriedades** | **Descrição** |
| lineCap | Define ou retorna o estilo das tampas nas extremidades de uma linha |
| lineJoin | Define ou retorna o tipo de canto criado, quando duas linhas se encontram |
| lineWidth | Define ou retorna a largura da linha atual |
| miterLimit | Define ou retorna o comprimento máximo da mitra |

|  |  |
| --- | --- |
| **RETANGULOS** | |
| **Método** | **Descrição** |
| rect() | Cria um retângulo |
| fillRect() | Desenha um retângulo "preenchido" |
| strokeRect() | Desenha um retângulo (sem preenchimento) |
| clearRect() | Limpa os pixels especificados em um determinado retângulo |

|  |  |
| --- | --- |
| **TRANSFORMAÇÕES** | |
| **Método** | **Descrição** |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **TEXTO** | |
| **Método** | **Descrição** |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **DESENHO DE IMAGEM** | |
| **Método** | **Descrição** |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |
| --- | --- |
| **MANIPULAÇÃO DE PIXEL** | |
| **Método** | **Descrição** |
|  |  |
|  |  |
|  |  |
|  |  |