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.

**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.

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

* + Alterar a localização de um elemento; aplicar uma transformação; exibir e ocultar elementos.

**Objetivo 1.4: Implementar APIs do HTML5**

* + Implementar APIs de armazenamento e API de Geolocation

**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 “this” para fazer referência a um objeto que disparou um evento; estabelecer escopo de variáveis localmente e globalmente

**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

**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

**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

**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”

**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

**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

**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.

**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

**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

**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.

**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.

**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.

**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.

**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.

**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.

**ANOTAÇÕES**

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

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

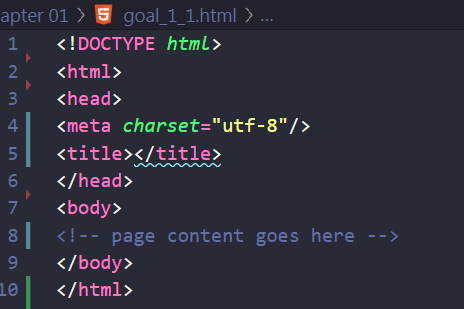
No HTML5 existe uma marcação semântica e não semântica é de suma importância para a interpretação dos buscadores, técnicas de SEO e otimização de leitores. O objetivo da estrutura de um documento é informar ao navegador como o conteúdo deve ser exibido. E é exatamente isso o que esse objetivo 1.1 ensina.

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.

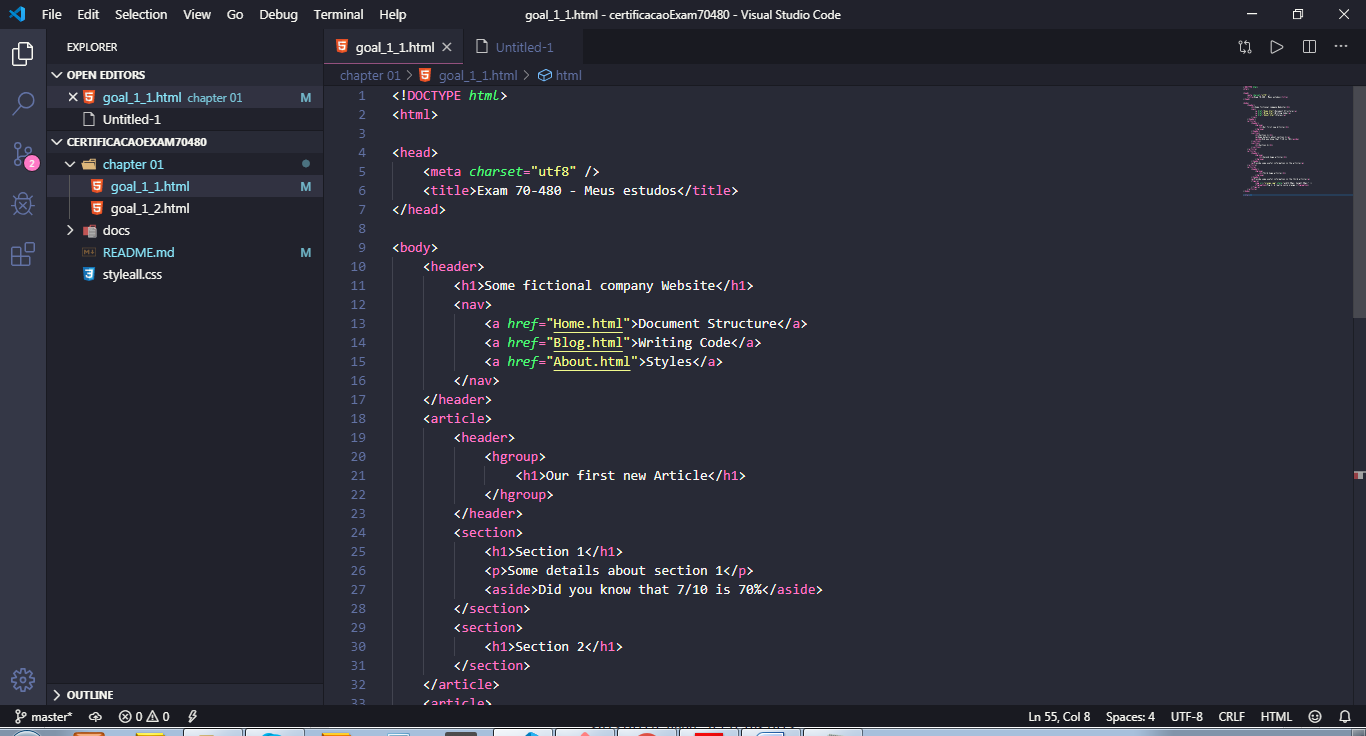
Elementos não semânticos são todas as tags que não representa nada, como por exemplo, <div> e <span>.

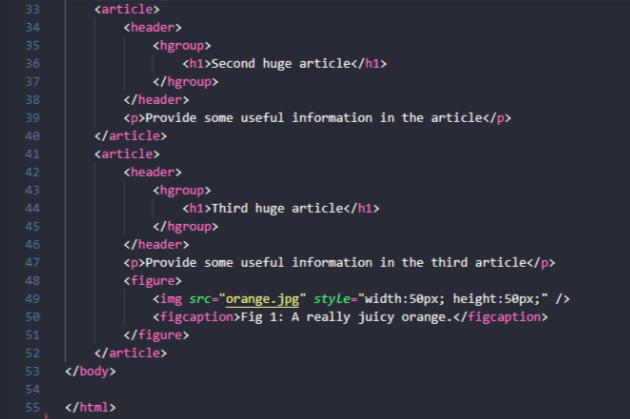
|  |  |
| --- | --- |
| **HTML5 element Semantic** | **Description** |
| <article> | Define áreas na página |
| <aside> | Define áreas de conteúdos menores fora do fluxo da página |
| <details> | Define detalhes adicionais que o usuário pode exibir ou ocultar |
| <figcaption> | Define o título do elemento 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.

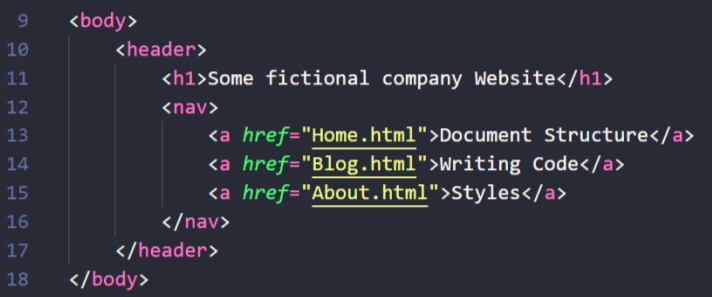




**O elemento <header>** não se limita apenas ao início da página. Ele fornece uma maneira semântica de declarar o cabeçalho para qualquer área da página. Podemos usar o <header> como um cabeçalho para um elemento <section> ou para um <article>. Dentro do <header> usamos qualquer marcação que atenda às 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.



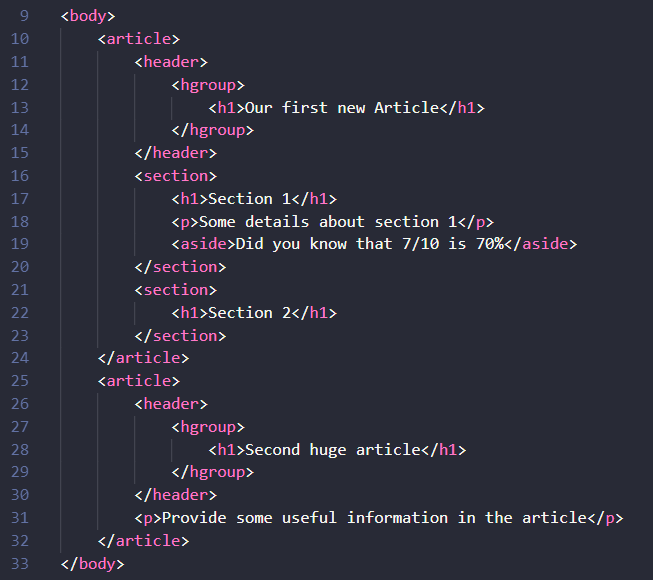
**O elemento <hgroup>** é um método semântico que organiza cabeçalhos e subtítulos. Esse elemento contém os elementos <h1> a <h6>. O <hgroup> pode ser utilizado em toda página desde que atenda esse proposito. A forma de renderização no navegador do <h1> dentro do <hgroup> é diferente quando o <h1> está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



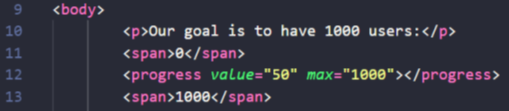
**O elemento <figure>** especifica conteúdo independente, como gráficos e figuras. Geralmente esta dentro de um <article> ou <section>. Sua posição é independente do fluxo principal da página e se removido, não deve afetar o fluxo do documento. Já o <figcaption> define uma legenda para o elemento <figure>, 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.

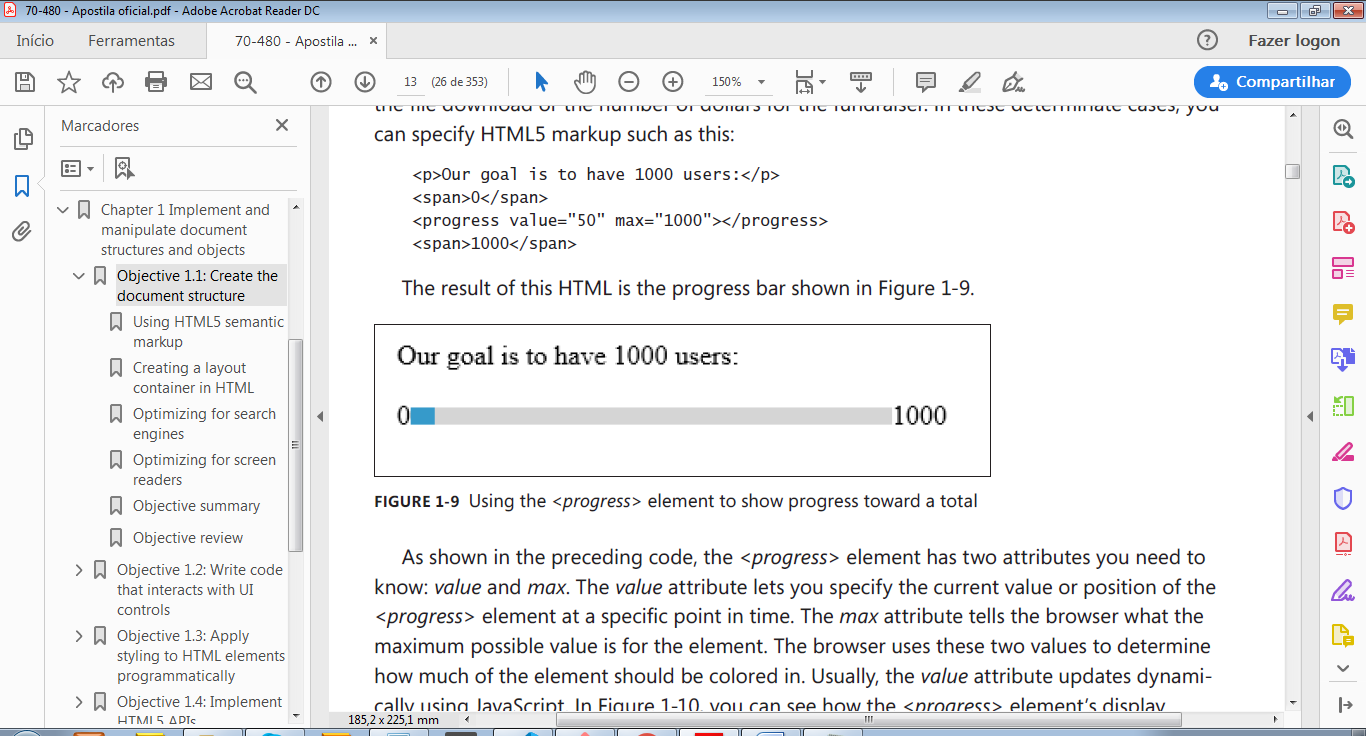


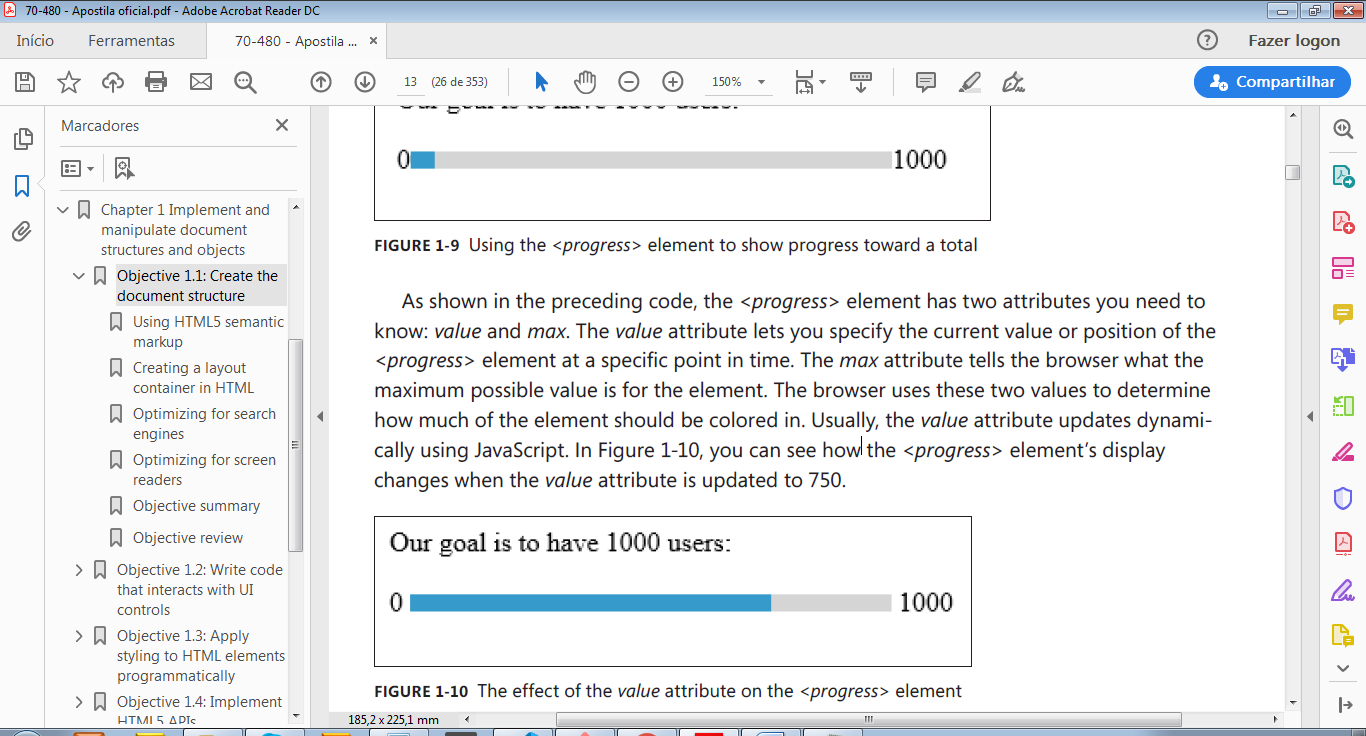
**O elemento <progress>** define o progresso de uma tarefa. Os dois tipos suportados de tarefas de progresso são determinados e indeterminados.

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.

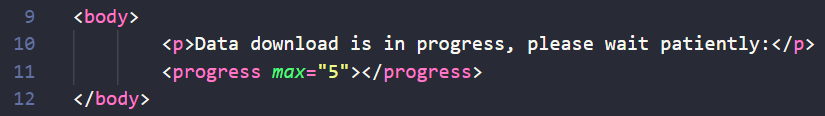
Sabendo os valores usa o código abaixo:

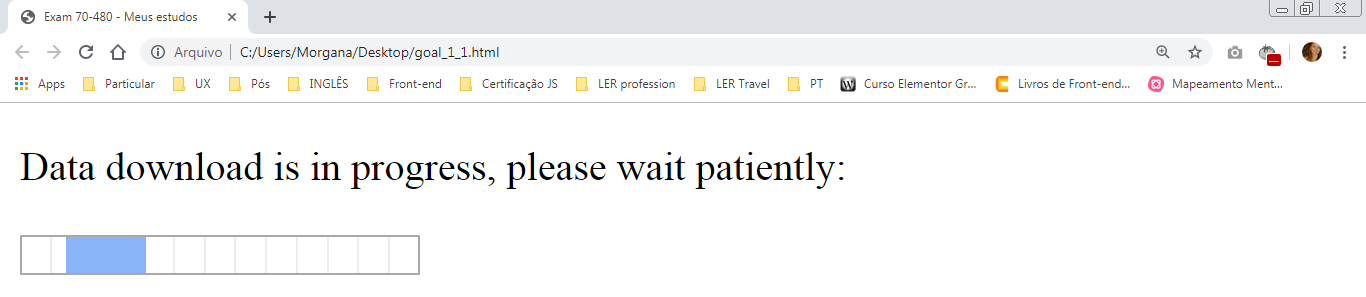


E que o resultado ficará assim: 

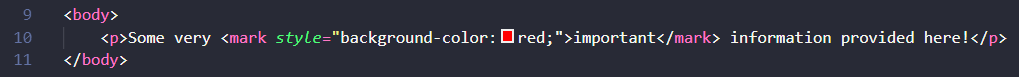
Se mudar o value de 50 para 750 vai ficaria assim:

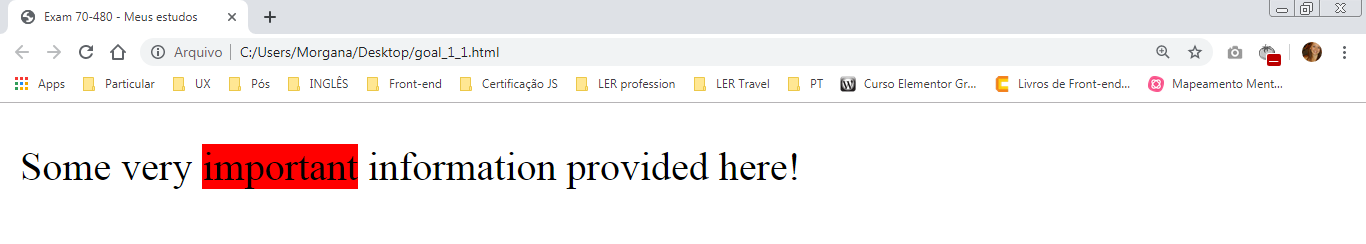
Usa-se uma tarefa indeterminada quando não sabemos os valores da tarefa ou quanto tempo levará para ser concluída. Por exemplo, quando você faz alguma ação no site e espera o resultado aparece um loading enquanto o processo é feito. Para isso, utilizamos o <progress> mas remove o atributo value. Quando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





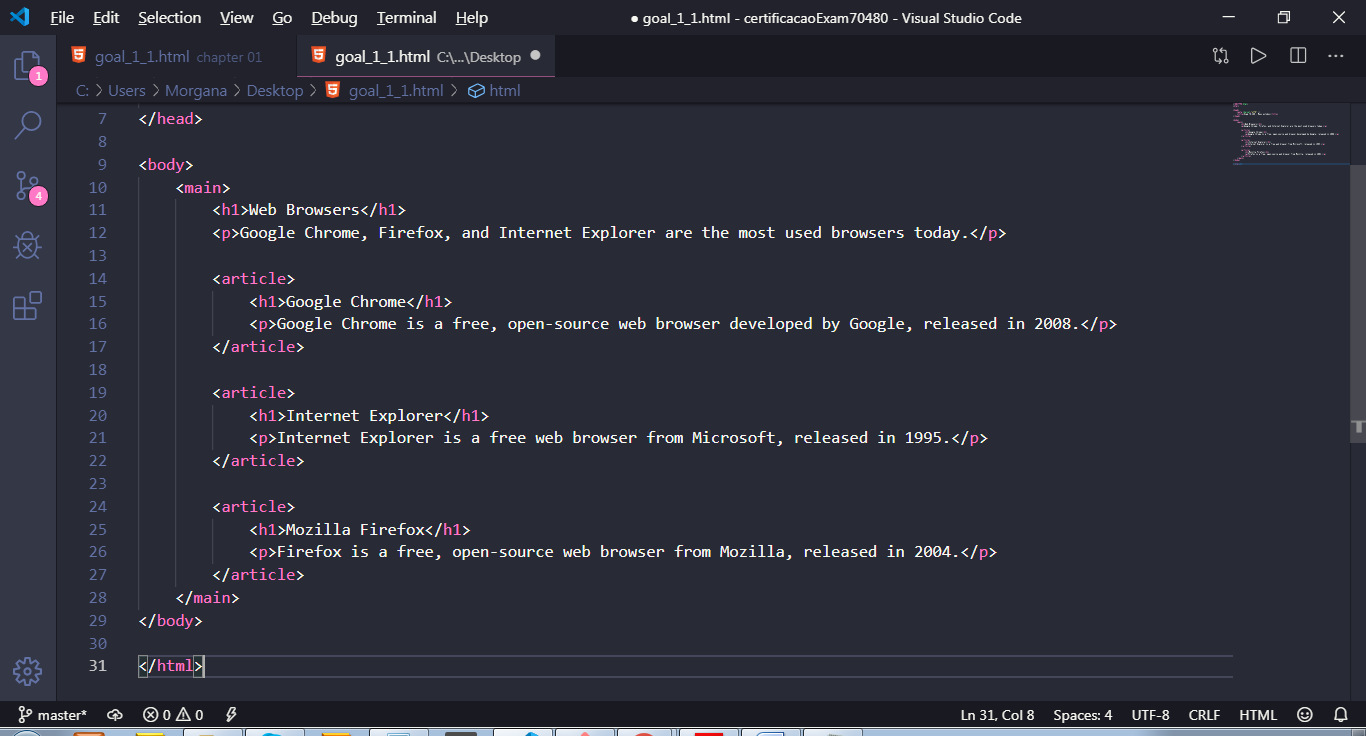
**O elemento <mark>** basicamente funciona como um marca-texto. Ao envolver um texto em um elemento <mark> e fornecer um atributo de cor de plano de fundo ao elemento de estilo vai obter o destaque desejado.



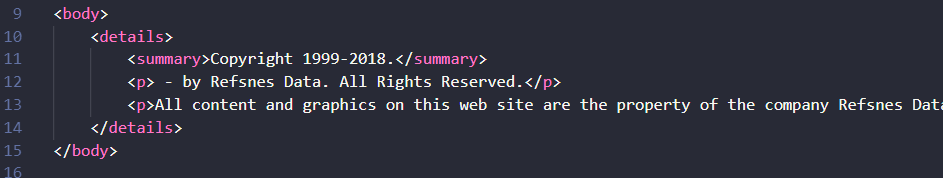


**O elemento <main>** define o conteúdo principal de um documento. O conteúdo dentro do elemento <main> deve ser exclusivo para o documento. Ele não deve conter nenhum conteúdo repetido em documentos como barras laterais, links de navegação, informações de direitos autorais, logotipos do site e formulários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

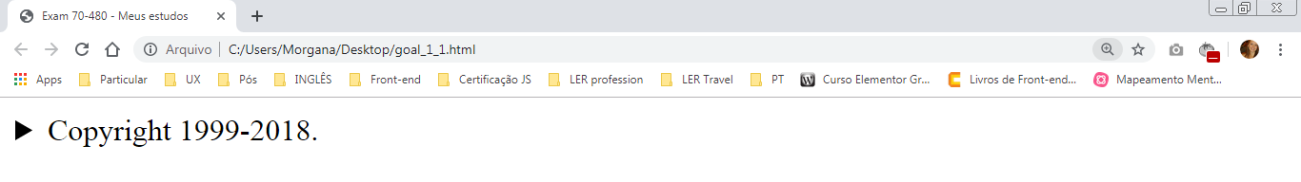
Não deve haver mais de um elemento <main> em um documento. O elemento <main> NÃO deve ser um descendente de um elemento <article>, <aside>, <footer>, <header> ou <nav>.

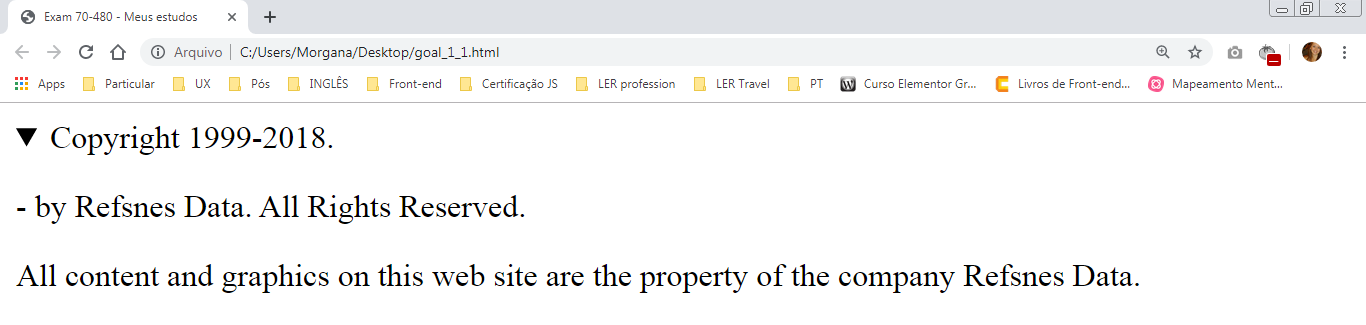


|  |  |
| --- | --- |
| **O elemento <details>** e **<sumary>:** O <details> especifica detalhes adicionais que o usuário pode exibir ou ocultar. O conteúdo do <details> não fica visível, a menos que seja definido. Podemos observar que o <sumary> é que define o cabeçalho para o elemento <details>, quando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

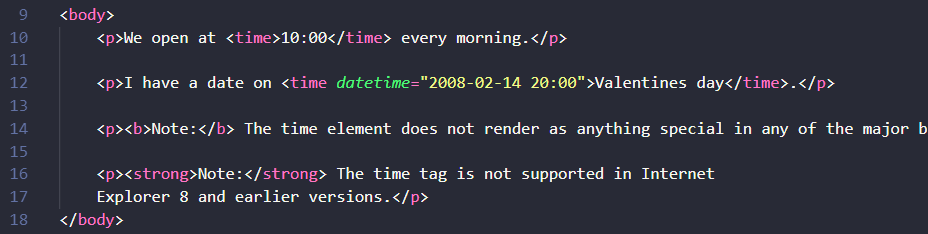


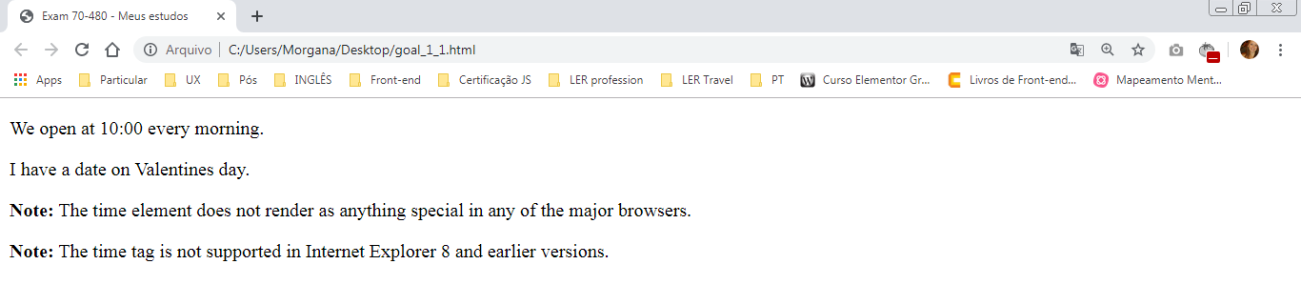
O <details> fechado.



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.

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

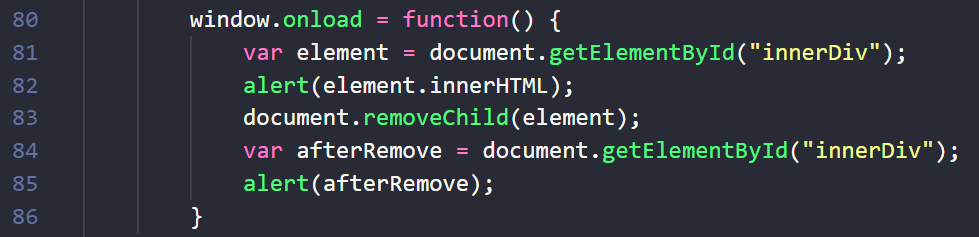
**O método querySelector()** retorna *o primeiro* elemento que corresponde ao seletor especificado. O **querySelectorAll()** retorna *todos* os elementos que correspondem aos critérios do seletor.

|  |  |
| --- | --- |
| //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.

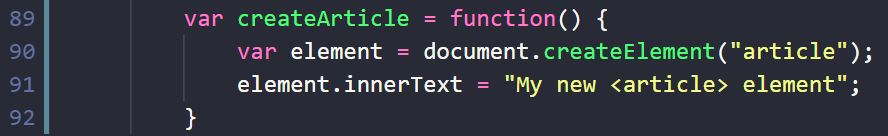
**2º** O método getElementById gera um erro porque o ID do elemento especificado não existe mais no documento.

**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.

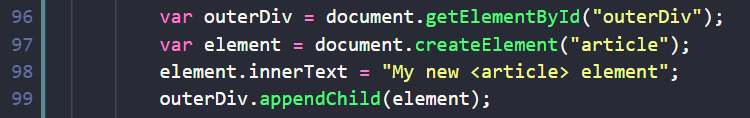
O código a seguir cria um novo elemento <article> mas não fica visível para ninguém neste momento; ele simplesmente existe no DOM para uso em sua página.

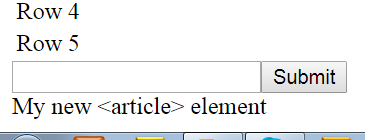


**Usando o método** appendChild

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:

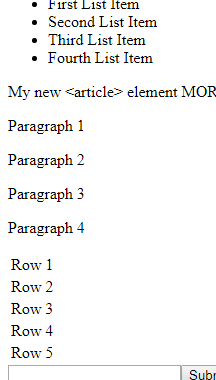
// A propriedade firstChild, indica que eu vou inserir o elemento <article>

// 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> não possui habilidades de desenho próprias.

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 fixo (x, y) da janela do navegador, com (0,0) no canto superior esquerdo para o centro. No entanto, a posição da tela na janela do navegador é irrelevante para os métodos de desenho que você usa para desenhar na tela.

Nesse caso abaixo, o canto inferior esquerdo da tela é (0,400), o canto superior direito é (600,0) e o canto inferior direito é (600,400).

Por exemplo as coordenadas do código são assim:

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

    Your browser does not support HTML5.

</canvas>

(0, 0)  Eixo X (600, 0)

(0,400) (600,400)

O Eixo X vai começar em 0 e vai até a largura do monitor.

E o Eixo Y vai de 0 até a altura do monitor.

**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.

**Desenhando curvas** <canvas>

Para desenhar curvas vamos usar alguns parâmetros. A tabela a seguir lista os métodos usados ​​ao trabalhar com curvas:

|  |  |
| --- | --- |
| **Método** | **Descrição** |
| arc | Um arco padrão baseado em um ângulo inicial e final e um raio definido |
| *quadradicCurveTo* | Um arco mais complexo que permite controlar a inclinação da curva |
| bezierCurveTo | Outro arco complexo que você pode inclinar |

Você pode controlar as propriedades lineWidth, strokeStyle e lineCap para alterar a forma como suas curvas são exibidas.

**Método** arc.

|  |  |
| --- | --- |
| **Método** | **Descrição** |
| X, y | Os dois primeiros parâmetros são as coordenadas X e Y do centro do círculo. |
| *radius* | O terceiro parâmetro é o raio. Este é o comprimento da distância do ponto central do círculo à curva. |
| startAngle, endAngle | O quarto e o quinto parâmetros especificam os ângulos inicial e final do arco a ser desenhado. Isso é medido em radianos, não em graus. |
| counterclockwise | O parâmetro final especifica a direção do desenho do arco. |

    ctx.beginPath();

            ctx.arc(150, 100, 75, 0, 2 \* Math.PI, false);

            ctx.lineWidth = 25;

            ctx.strokeStyle = '#0f0';

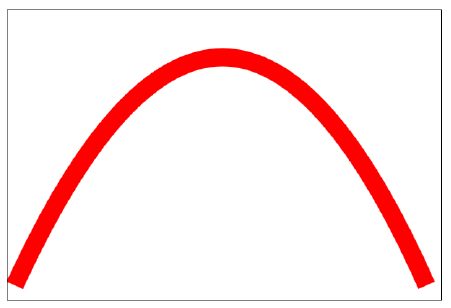
            ctx.stroke();

**Método** quadraticCurveTo

O método quadraticCurveTo permite especificar alguns parâmetros adicionais para alterar a “inclinação” da curva - em outras palavras, para alterar a distância do ponto central ao longo da curva. Desenhar uma curva quadrática é como desenhar uma linha reta, mas depois beliscar no meio e afastá-la para criar uma curva em que os pontos inicial e final da linha permaneçam fixos.

Você primeiro precisa usar o método moveTo para informar o contexto em que deseja que sua curva inicie. Em seguida, você passa os quatro parâmetros descritos para o método quadraticCurveTo.

À medida que o ponto de controle se afasta da linha formada pelos pontos inicial e final, o código a seguir cria uma curva mais acentuada. Este exemplo usou um número negativo para indicar que o ponto de controle deve estar acima da parte superior da tela para esticar a curva.



    ctx.beginPath();

            ctx.moveTo(10,380);

ctx.quadraticCurveTo(300,-250,580,380);

            ctx.lineWidth = 25;

            ctx.strokeStyle = '#0f0';

            ctx.stroke();

Os parâmetros para o método quadraticCurveTo

|  |  |
| --- | --- |
| **Parâmetro** | **Descrição** |
| controlX, controlY | Esses parâmetros definem o ponto de controle, relativo ao canto superior esquerdo da tela, que é usado para "esticar" a curva para longe da linha formada pelos pontos inicial e final. |
| endX, endY | Este é o ponto em que a curva deve terminar. |

**Método** beizerCurveTo

Uma curva de Bezier é semelhante à curva quadrática, exceto que ela possui dois pontos de controle em vez de apenas um. Ter dois pontos permite que a curva de Bezier crie curvas mais complexas.

    ctx.beginPath();

            ctx.moveTo(125, 20);

ctx.quadraticCurveTo(0, 200, 300, 300, 50, 400);

            ctx.lineWidth = 5;

            ctx.strokeStyle = '#0f0';

            ctx.stroke();

É preciso passar três conjuntos de coordenadas para o método bezierCurveTo, conforme listado na Tabela abaixo:

|  |  |
| --- | --- |
| **Parâmetro** | **Descrição** |
| controlX, controlY | Os dois primeiros parâmetros especificam o primeiro ponto de controle usado para esticar a curva. |
| Control2X, control2Y | Os segundos dois parâmetros especificam o segundo ponto de controle usado para esticar a curva. |
| endX, endY | Os dois parâmetros finais especificam o ponto final da curva. |



**Método** path

Quando usamos um objeto contexto para desenhar, vamos precisar de um ponto inicial e um ponto final. O ponto final de uma linha pode se tornar o ponto inicial da próxima linha. Para fazer isso chamamos o método beginPath no objeto do contexto e desenha todas as linhas antes de chamar o método closePath(para terminar a linha) ou o método beginPath(que inicia uma nova linha).

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

var ctxP = metPath.getContext("2d");

ctxP.beginPath();

ctxP.arc(300, 200, 75, 1.75 \* Math.PI, 1.25 \* Math.PI, false);

ctxP.lineTo(150, 125);

ctxP.quadraticCurveTo(300, 0, 450, 125);

ctxP.lineTo(353, 144);

ctxP.strokeStyle = "blue";

ctxP.lineCap = "round";

ctxP.lineWidth = 10;

ctxP.stroke();

**Método** Rect

O objeto de contexto abaixo possui um método chamado rect. E é preciso de alguns parâmetros.

|  |  |
| --- | --- |
| **Parâmetro** | **Descrição** |
| X, Y | As coordenadas x e y definem a posição inicial do retângulo. Este é o canto superior esquerdo do retângulo. |
| width | Isso define a largura do retângulo. |
| height | Isso define a altura do retângulo. |

var metPath = document.getElementById("metRect");

var ctxR = metPath.getContext("2d");

ctxR.beginPath();

      ctxR.rect(50, 50, 150, 75);

      ctxR.stroke();

Para centralizar o objeto retângulo, é necessário fazer um pouco de matemática para calcular o centro com base no tamanho da tela, bem como no tamanho do retângulo desejado.

Abaixo alguns métodos com cores.

|  |  |
| --- | --- |
| **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 |

**Método** createLinearGradient

Para criar um gradiente usa um novo objeto CanvasGradient . Primeiro chama o método createLinearGradient disponível no objeto de contexto para obter um objeto CanvasGradient.

Nesse objeto CanvasGradient , define as paradas de cores que deseja mesclar para criar o efeito de gradiente. Em seguida, atribua seu objeto CanvasGradient à propriedade fillStyle do contexto.

O código a seguir cria e preenche um retângulo com um gradiente linear:

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

      ctxt.lineWidth = 3;

      ctxt.rect(150, 150, 200, 125);

      var gradient = ctxt.createLinearGradient(150, 150, 200, 125);

      gradient.addColorStop(0, "Black");

      gradient.addColorStop(0.5, "Gray");

      gradient.addColorStop(1,"White");

      ctxt.fillStyle = gradient;

      ctxt.fill();

      ctxt.stroke();

Esse código cria o objeto CanvasGradient passando os pontos inicial e final de uma linha de gradiente.

O método addColorStop usa dois parâmetros:

* + 1. Um valor de 0 a 1, em que 0 é o ponto inicial da linha de gradiente e 1 é o ponto final.
    2. A cor para começar a encher nessa parada.

Este exemplo tem três paradas, então o gradiente transita por três cores.



**Método** createRadialGradient

Este método utiliza seis parâmetros, que especificam o ponto central e o raio de dois círculos e a cor transita pelas paradas ao longo do cone formado pelos dois círculos.

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

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

    ctxt.lineWidth = 3;

    ctxt.rect(120, 120, 220, 145);

    var gradient = ctxt.createRadialGradient(200, 200, 5, 250, 250, 100);

    gradient.addColorStop(0, "Red");

    gradient.addColorStop(.5, "Orange");

    gradient.addColorStop(1, "Blue");

    ctxt.fillStyle = gradient;

    ctxt.fill();

    ctxt.stroke();



**Método** createPattern

Nesse caso, uma imagem externa é aplicada como padrão em toda a forma. Você pode usar uma imagem que você criou como plano de fundo para sua tela usando o seguinte código:

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

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

    ctxt.lineWidth = 3;

    ctxt.rect(150, 150, 200, 125);

    var img = **new** *Image*();

    img.src = "image.jpg";

    img.onload = function () {

        var pat = ctxt.createPattern(img, "repeat");

        ctxt.fillStyle = pat;

        ctxt.fill();

        ctxt.stroke();

    }

O código anterior chama o método createPattern e passa uma referência a um objeto Image e um padrão de repetição. O padrão de repetição pode ser sem repetição , repetição x ou repetição y , mas o padrão é repetir se você não especificar nada.



**Desenhando** imagens

Para desenhar uma imagem em uma tela, use o drawImagemétodo do objeto de contexto. Esse método utiliza um objeto Image e algumas coordenadas (x, y) para definir onde a imagem deve ser desenhada.

O canto superior esquerdo da imagem é desenhado no especificado (x, y). O tamanho padrão da imagem é o tamanho real da imagem, mas você também pode redimensioná-la.

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

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

        var img = **new** *Image*();

        img.src = "image.jpg";

        img.onload = function () {

            ctxt.drawImage(img, 0, 0);

            ctxt.stroke();

        }



Se você deseja redimensionar a imagem, substitua a chamada do método drawImage pela seguinte linha:

ctxt.drawImage(img, 0,0,img.width \* .5, img.height \* .5);

**Desenhando** texto

A chamada strokeText desenha o texto especificado nas coordenadas especificadas na tela. Os parâmetros especificam qual texto desenhar e as coordenadas (x, y) especificam onde o desenho deve começar.

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

var ctxtText1 = drawingText.getContext("2d");

      ctxtText1.strokeText("1. Text with default font", 100, 100);

O método strokeText desenha no estilo de fonte padrão. Você pode alterar facilmente a propriedade da fonte do objeto de contexto para melhorar a aparência do seu texto. Por exemplo, a execução do código a seguir altera o tamanho da fonte para 24 e a família de fontes para Arial:

ctxtText2.font = "24px arial";

      ctxtText2.strokeText("2. Text with altered font", 100, 125);

E para colorir seu texto, é só adicionar este código:

      ctxtText3.strokeStyle = "Red";

Para desenhar um texto com cor(preenchimento),é só usar a propriedade fillStyle e chama o método fillText em vez dos métodos strokeStyle e StrokeText:

      ctxtText4.fillStyle = "Red";

      ctxtText4.fillText("4. Text with altered colored font", 100, 185);

Para centralizar o texto usa a propriedade abaixo. Ao definir a propriedade textAlign para o centro de valor, você está instruindo o contexto a considerar a coordenada especificada (x, y) como o ponto central da string em vez do ponto inicial da string. Assim, você divide a largura e a altura da tela por dois para obter o ponto central da tela e obtém uma corda centralizada na horizontal e na vertical.

      ctxtText5.textAlign = "center";

ctxtText5.fillText("5. Text with altered colored font Centered.",

drawingSurface.width / 2,

drawingSurface.height / 2);

O texto complete de tudo dito a cima esta aqui embaixo.

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

var ctxtText1 = drawingText.getContext("2d");

      var ctxtText2 = drawingText.getContext("2d");

      var ctxtText3 = drawingText.getContext("2d");

      var ctxtText4 = drawingText.getContext("2d");

      var ctxtText5 = drawingText.getContext("2d");

      ctxtText1.strokeText("1. Text with default font", 100, 100);

      ctxtText2.font = "24px arial";

      ctxtText2.strokeText("2. Text with altered font", 100, 125);

      ctxtText3.font = "24px arial";

      ctxtText3.strokeStyle = "Red";

      ctxtText3.strokeText("3. Text with altered font", 100, 160);

      ctxtText4.font = "24px arial";

      ctxtText4.fillStyle = "Red";

      ctxtText4.fillText("4. Text with altered colored font", 100, 185);

      ctxtText5.font = "24px arial";

      ctxtText5.textAlign = "center";

      ctxtText5.fillStyle = "Red";

      ctxtText5.fillText("5. Text with altered colored font Centered.", drawingSurface.width / 2,

drawingSurface.height / 2);

Com esse resultado na tela:



**Implementando gráficos com HTML5 - Scalable Vector Graphics <svg>** (Gráficos vetoriais escaláveis)

Scalable Vector Graphics (SVG) é uma linguagem baseada em XML para a criação de gráficos bidimensionais. É implementado usando tags definidas pelo namespace XML SVG e incorporadas nos documentos HTML5 dentro dos elementos <svg> de abertura e fechamento.

Os objetos SVG não perdem qualidade à medida que os usuários aumentam ou diminuem o zoom. Você pode acessar objetos SVG por meio do DOM e, semelhante aos elementos HTML, os elementos SVG suportam atributos, estilos e manipuladores de eventos. O elemento <svg> fornece um contêiner no qual renderiza gráficos;

<body>

    <svg *style*="padding:45px;">

       <circle *id*="Circle" *cx*="50" *cy*="50" *r*="50" *fill*="green" *onmouseover*="Red(evt)"

*onmouseout*="Green(evt)" />

    </svg>

    <script>

        function Red(*evt*) {

            var circle = evt.target;

            circle.setAttribute("style", "fill: red");

        }

        function Green(*evt*) {

            var circle = evt.target;

            circle.setAttribute("style", "fill: green");

        }

    </script>

</body>

Toda a funcionalidade de desenho de forma e desenho de linha que você viu na discussão do elemento <canvas> também existe para o SVG, embora a sintaxe seja diferente, é claro.

**Desenhando formas <SVG> usando funções**

O SVG também suporta as mesmas funções básicas de desenho de forma que o contexto da tela. O seguinte segmento de código mostra o uso da polyline, polygon, linha e ellipse:

<svg *style*="padding:15px;">

<polygon *points*="10,15 30,35 10,85 100,85, 70,35,100,15" *fill*="purple" />

<polyline *points*="10,150 30,170 50,132 62,196 78,165 96,170" *style*="stroke:orange; fill:none; stroke-width:5;" />

<line *x1*="150" *y1*="100" *x2*="150" *y2*="150" *style*="stroke:blue;stroke-width:3" />

<ellipse *cx*="250" *cy*="150" *rx*="30" *ry*="55" *fill*="green" />

<text *x*="10" *y*="10" *style*="stroke: black;stroke-width:1;">

Examples of SVG Shapes and Text

</text>

</svg>

**Desenhando imagens <SVG>**

O SVG também suporta a renderização de gráficos existentes na forma de arquivos de imagem externos, conforme mostrado no código abaixo:

<svg *id*="mySVG">

       <image *href*="image.jpg" *width*="250" *height*="100" />

</svg>

Em alguns casos, o uso de gráficos SVG é mais simples do que o elemento <canvas>. Como os exemplos mostraram, você pode criar imagens SVG declarativamente diretamente no próprio HTML. No entanto, à medida que você aumenta o número de objetos em uma renderização SVG, o desempenho pode se tornar uma preocupação. Nos casos em que o desempenho é considerado, o uso do elemento <canvas> é uma abordagem preferível.

**Resumo objetivo**

* JavaScript é uma ferramenta poderosa que permite que os desenvolvedores manipulem o DOM programaticamente no navegador.
* O HTML5 suporta controles rich media para incorporar vídeo usando o elemento <video> e áudio usando o elemento <audio>.
* O elemento <video> suporta vários formatos de mídia usando o elemento <source>.
* Os elementos <canvas> e <svg> do HTML5 oferecem suporte a uma API avançada para criar gráficos simples e complexos no navegador.
* Os mecanismos gráficos <canvas> e <svg> podem desenhar texto, linhas, formas, fontes, preenchimentos e gradientes.
* O elemento <canvas> é desenhado via JavaScript, obtendo uma referência ao contexto.
* O elemento <svg> renderiza gráficos usando uma sintaxe declarativa.

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

A seção aborda a aplicação de estilos aos elementos HTML na página dinamicamente, usando JavaScript. Quando você recupera referências de elementos usando métodos como getElementById, é possível manipular esses elementos, incluindo seus estilos. Mas antes falo minuciosamente sobre os cinco tipos de posicionamento: **Absolute, relative, static, fixed e sticky.** Ao usar o posicionamento absoluto ou relativo, as configurações padrão de borda ou margem são ignoradas porque o objeto é posicionado onde os atributos de posicionamento direcionam o elemento.

Este objetivo cobre como:

■ Alterar a localização de um elemento

■ Aplicar uma transformação

■ Mostrar e ocultar elementos

**Posicionamento absolute**

O posicionamento absolute tem como pai a página HTML, na prática ele é filho da página e não de um elemento qualquer no HTML. Com isso é possível posicionar ele em qualquer lugar que não vai afetar o layout. Portanto utilizando position: absolute o elemento será colocado no local exato especificado, em relação às bordas do container.

Como não faz parte da hierarquia da página o position: abolute, pode comprometer o layout em versões mobile, tablet.

Esse código abaixo mostra as imagens sendo manipuladas via javascript exatamente o que é abordado na apostila da Microsoft.

  var top = document.getElementById("topText");

var left = document.getElementById("leftText");

      var pos = document.getElementById("positioning");

       document.getElementById("btnPosition").onclick = function() {

                var img = document.getElementById("img2");

                img.style.position = pos.value;

                img.style.left = left.value + "px";

                img.style.top = top.value + "px";

            }

**Posicionamento relativo**

O posicionamento relative especifica uma posição relativa do elemento em relação ao elemento pai. Sua posição vai ser definida com as propriedade: top, right, bottom e left que vai indicar a distância do elemento tomando como referência o elemento pai.

Um exemplo extraído do site devmedia.

01 *.relative-parent* {

02    background: #156236;

03 }

04

05 *.relative* {

06    background: #af8542;

07    height: 100px;

08    width: 300px;

09    position: relative;

10    top: 20px;

11    left: 45px;

12 }

Uma explicação do código a cima:

Linha 01: foi declarado a class relative-parent. Que no HTML essa class é o elemento pai de uma div que tem o posicionamento relative. No elemento pai não foi declarado nenhum posicionamento, neste caso, por padrão entendo que tem o posicionamento “static”.

Linha 05: foi declarado um position: relative. Isso quer dize que o elemento filho terá o posicionamento determinado de acordo com o posicionamento do elemento pai.

Linhas 10 e 11: foi definido a distância com relação ao elemento pai.

Por exemplo: Essa caixa maior é o elemento PAI (.relative-parent)

E essa caixa abaixo azul é o elemento filho (.relative) Quando definimos o top ele vai se mover da borda do elemento pai para o centro. E quando definimos o left será da borda esquerda par o centro. Isso serve para as outras propriedades right e bottom, sempre visando o centro do elemento pai.

TOP

LEFT 

**Posicionamento estático**

**Posicionamento fixed**

Essa definição é bem lógica ao nome que é dado, position: fixed, o elemento ficará fixo na mesma posição definida. E não tem relação com nenhum outro elemento Pai. Ele é único.

**Posicionamento sticky**

Um elemento com position: sticky; é posicionado com base na posição de rolagem do usuário.

Um elemento sticky alterna entre relative e fixed, dependendo da posição de rolagem. É posicionado em relação até que uma determinada posição de deslocamento seja alcançada na janela de visualização - então "gruda" no lugar (como posição: fixa).

**Alterar a localização de um elemento**

Usando os métodos para recuperar um elemento do DOM em JavaScript, você pode aplicar estilos dinamicamente por meio de código que pode alterar a posição do elemento na página.

Por padrão, todos os elementos HTML fluem estaticamente da esquerda para a direita na mesma ordem em que são declarados na página HTML. No entanto, o CSS fornece um mecanismo para especificar algumas opções avançadas na posição do elemento. Você pode posicionar elementos usando posicionamento absoluto ou posicionamento relativo.

Você pode aplicar quatro propriedades individualmente ou em combinação para controlar a posição de um elemento: top,right, bottom e left. Essas propriedades só vão funcionar se o position for definido primeiro.

Cada propriedade usa um parâmetro de distância que especifica a distância relativa do objeto a partir de um ponto de referência com base no atributo de posicionamento especificado.

**Aplicando transform**

Existe algumas propriedades para o transform, mas a apostila só cita algumas, abaixo segue todas disponíveis W3S são elas:

* translate()
* rotate()
* scaleX()
* scaleY()
* scale()
* skewX()
* skewY()
* skew()
* matrix()

Transform permite alterar a aparência de um elemento. Você pode aumentar ou diminuir um elemento, girá-lo.

Existem muitos métodos de transformação disponíveis. Para adicionar uma transformação a um elemento, você a declara no CSS do elemento, adicionando a propriedade transform.

*.rota* {transform: rotate(90deg);}

**Usando método** rotate

O método de transformação de rotação permite que você gire um objeto por um número especificado de graus. O método aceita um único parâmetro que especifica o número de graus. Podemos usar valores positivos e vai girar no sentido horário ou valores negativos e vai girar no sentido anti-horário.

O transform também suporta rotateX e rotateY métodos, que aceitam um único parâmetro em graus para especificar um ângulo em torno do eixo X ou do eixo Y na qual girar.

Você pode, por exemplo, usar esses métodos para inverter um elemento vertical ou horizontalmente, especificando 180deg como o parâmetro. Nesse caso, o elemento gira 180 graus ao longo do eixo especificado - o que basicamente resulta na inversão ou espelhamento da imagem ao longo desse eixo.

**Usando o método** translate

O método de conversão permite mover um elemento HTML alterando sua posição X e Y relativa na página.

            transform: translate(50px, 0px);

O método de conversão move o elemento HTML ao qual é aplicado por 50 pixels na direção X e 0 pixels na direção Y em relação ao local onde agora reside.

Novamente, os métodos translateX e translateY estão disponíveis se o efeito desejado for mover o objeto ao redor do eixo x ou do eixo y.

**Usando o método** skew

Você pode inclinar um elemento HTML usando o método skew da propriedade transform. A inclinação inclina o objeto para que não fique paralelo ao eixo vertical ou horizontal.

            transform: skew(10deg, 10deg);

**Usando o método** scale

O método scale redimensiona os elementos por uma proporção especificada. O método de escala usa um parâmetro: um valor decimal que representa a porcentagem a ser dimensionada. O método scale usa como parâmetro: um valor decimal que representa a porcentagem a ser dimensionada.

Por exemplo, quando o valor é:

* Maior que 1 aumenta o objeto
* Menor que 1, mas maior que 0 diminui o objeto

            transform: scale (1,5);

**Mostrando e escondendo elementos**

Você pode mostrar e ocultar elementos declarativamente na marcação HTML ou programaticamente, modificando as propriedades CSS do objeto por meio de JavaScript.

Você pode criar as propriedades CSS que mostram ou ocultam um elemento diretamente na propriedade de estilo de um objeto ou em um estilo CSS e são adicionadas à coleção de estilos do elemento.

**Propriedade Display**

A propriedade display aceita dois valores possíveis. Um valor de INLINE diz ao navegador para mostrar o item, enquanto um valor de NONE significa que o navegador deve ocultar o item.

Quando você usa a propriedade CSS de exibição e a define como valor zero, o elemento HTML fica oculto. Mas ocultar o elemento dessa maneira também o remove do layout.

**Propriedade Visibility**

A segunda propriedade disponível para controlar a visibilidade do elemento é chamada visibility. Esta propriedade aceita quatro valores possíveis, conforme descrito na Tabela abaixo:

|  |  |
| --- | --- |
| **Value** | **Efeito** |
| visible | Define a propriedade como visível para mostrar o elemento |
| hidden | Oculta o elemento |
| collapse | Recolhe o elemento onde aplicável, como em uma linha da tabela |
| inherit | Herda o valor da propriedade de visibilidade do pai |

Alguns desses valores têm comportamentos interessantes. Quando você usa a propriedade CSS de exibição e a define como valor zero, o elemento HTML fica oculto. Mas ocultar o elemento dessa maneira também o remove do layout. Todos os elementos circundantes se realinham como se o elemento não estivesse lá. Quando a exibição é definida como embutida, o elemento é mostrado novamente e todos os elementos ao redor se afastam, voltando ao local em que estavam originalmente.

Alguns desses valores têm comportamentos interessantes. Quando você usa a propriedade CSS de exibição e a define como valor zero, o elemento HTML fica oculto. Mas ocultar o elemento dessa maneira também o remove do layout. Todos os elementos circundantes se realinham como se o elemento não estivesse lá. Quando a exibição é definida como embutida, o elemento é mostrado novamente e todos os elementos ao redor se afastam, voltando ao local em que estavam originalmente.

DICA DO EXAME

Se você precisar preservar o layout da página ao alterar VISIBILITY, use a propriedade VISIBILITY com o value hidden. Se você não precisar preservar o layout, poderá definir a propriedade de exibição como NONE ou VISIBILITY como COLLAPSE.

**Resumo objetivo**

* Você pode usar CSS para definir efeitos de transformação.
* Você pode aplicar transformações via JavaScript para manipular o DOM com efeitos como girar, inclinar, dimensionar e traduzir.
* A propriedade de visibilidade fornece opções para controlar a visibilidade de um elemento dentro da página.

**Objetivo 1.4: Implementar APIs HTML5**

As APIs JavaScript forneceram algumas novas funcionalidades poderosas, como a capacidade de armazenar mais dados localmente e disponibilizá-los para a página da Web por meio da API de armazenamento na Web e que está disponível como um objeto global.

A API do AppCache permite colocar aplicativos da Web offline. A API de geolocalização fornece métodos para trabalhar com o posicionamento global no aplicativo.

**Usando a API de armazenamento**

O WebA Storage é uma nova API para armazenar dados de páginas da web localmente. Isso permite que você leia, adicione, modifique e exclua itens de dados armazenados. O Armazenamento na Web, substitui o conceito de cookies.

Existem duas formas de armazenamento na Web:

* **Armazenamento local.** O armazenamento local é persistente; os dados armazenados estão disponíveis para a página da web, mesmo que o usuário feche o navegador e depois o abra novamente no site. Para acessar o armazenamento local, use o objeto localStorage.

**Sintaxe**

window.localStorage

**Sintaxe para salvar dados no localStorage:**

localStorage.setItem("key", "value");

**Sintaxe para LER dados de localStorage:**

var lastname = localStorage.getItem("key");

**Sintaxe para REMOVER dados do localStorage:**

localStorage.removeItem("key");

* **Armazenamento de sessão.** É volátil e está disponível apenas durante a sessão atual; portanto, se o usuário fechar o navegador, o armazenamento da sessão será limpo automaticamente e não estará mais disponível. Para acessar o armazenamento da sessão, use o objeto sessionStorage.

**Sintaxe**

window.sessionStorage

**Sintaxe para salvar dados em sessionStorage:**

sessionStorage.setItem("key", "value");

**Sintaxe para ler dados de sessionStorage:**

var lastname = sessionStorage.getItem("key");

**Sintaxe para REMOVER dados salvos da sessionStorage:**

sessionStorage.removeItem("key");

**Sintaxe para REMOVER TODOS os dados salvos do sessionStorage:**

sessionStorage.clear();

DICA

Os objetos localStorage e sessionStorage fornecem exatamente a mesma API. A única diferença é a vida útil do armazenamento. Lembre-se de que sessionStorage é limpo quando a sessão é fechada, enquanto localStorage ainda pode ser acessado depois que uma sessão é fechada e uma nova sessão é aberta.

O Armazenamento na Web é implementado como pares de valor de nome e armazenado como sequências de caracteres. Quaisquer dados que você possa colocar em um formato de sequência podem ser armazenados no Armazenamento na Web.

|  |  |
| --- | --- |
| **Method** | **Description** |
| setItem | Adiciona um key/value no armazenamento. Se nenhum item com a chave especificada existir, o item será adicionado; se essa chave existir, seu valor será atualizado. |
| getItem | Recupera dados do armazenamento com base em um valor ou índice de chave especificado. |
| clear | Limpa todo o armazenamento que foi salvo. Use este método para limpar o armazenamento conforme necessário. |
| key | Recupera a chave em um índice especificado. Você pode usar a chave resultante para passar como um parâmetro para um dos outros métodos que aceitam uma chave. |
| removeItem | Remove o key/value especificado no armazenamento. |

Além dos métodos descritos a cima, os objetos de armazenamento expõem uma propriedade length que retorna o número de pares de key/ value no armazenamento.

O código abaixo cria caixas de texto para aceitar uma chave e um valor, respectivamente. Os botões permitem **adicionar** itens ao armazenamento, **remover** um item ou **limpar** completamente o armazenamento.

Cada recurso é implementado por sua vez. Para exibir o conteúdo do armazenamento, a página contém uma div que mostra o conteúdo do armazenamento anexado a ele. O método **LoadFromStorage** é chamado para cada operação para atualizar a página com os dados disponíveis no armazenamento. Todos os exemplos a seguir usam armazenamento local, mas, novamente, eles funcionariam da mesma maneira com o armazenamento da sessão.

Se você deseja testar esses exemplos usando o armazenamento de sessão, basta substituir a referência localStorage por uma referência a sessionStorage.

Você primeiro precisa implementar o método **LoadFromStorage** para que, quando a página for carregada, você possa ver os itens que já foram colocados no armazenamento. Na parte superior foi adicionado uma chamada ao método LoadFromStorage, para que localStorage seja verificado após o carregamento da página.

window.onload = function() {

            LoadFromStorage();

O método **LoadFromStorage** pega qualquer elemento disponível no armazenamento local e os exibe em uma tabela HTML. Esse código aproveita a propriedade length para determinar se algum valor de armazenamento local precisa ser exibido. Caso contrário, a página exibirá uma mensagem sobre nenhum dado no armazenamento local.

Para adicionar eventos onclick nos botões é feito as funções abaixo e manipular o localStorage:

document.getElementById("btnAdd").onclick = function() {

    localStorage.setItem(document.getElementById("toStorageKey").value,

        document.getElementById("toStorageValue").value);

    LoadFromStorage();

}

document.getElementById("btnRemove").onclick = function() {

  localStorage.removeItem(document.getElementById("toStorageKey").value);

    LoadFromStorage();

}

document.getElementById("btnClear").onclick = function() {

    localStorage.clear();

    LoadFromStorage();

}

Um usuário agora pode adicionar itens ao armazenamento local e ver o que está armazenado. O usuário pode continuar adicionando ao armazenamento local (localStorage) neste aplicativo até que o armazenamento esteja cheio. **A disponibilidade do armazenamento local (localStorage) é limitada** **e o armazenamento disponível NÃO é consistente entre os navegadores.**

**Armazenamento na Web x Cookies**

O benefício de usar a API de armazenamento da Web em vez de cookies é que os dados residem localmente e permanecem locais. A alocação de espaço está disponível em um domínio por raiz. Por exemplo, domínios como localhost ou microsoft.com recebem seu próprio espaço de armazenamento na web seguro.

Conforme definido pela API, o armazenamento na Web permite o armazenamento apenas de pares de key /value em que a key e o componente de value são armazenados como uma sequência.

**Armazenando objetos**

Se você precisar armazenar objetos mais complexos no armazenamento na Web, poderá usar algumas técnicas. Por exemplo, o código abaixo:

document.getElementById("btnAddObject").onclick = function() {

                var customer = **new** *Object*();

                customer.firstName = "Morgana";

                customer.lastName = "Maria";

                customer.shirtSize = "P";

                localStorage.setItem("cart1", JSON.stringify(customer));

                LoadFromStorage();

}

Para armazenar o objeto customizado nos dados locais, é necessário um método para converter o objeto customizado em uma sequência que corresponda ao modelo de armazenamento local. É aqui que a JSON (JavaScript Object Notification) pode ser útil.

Você pode serializar o objeto em uma cadeia JSON, fornecer uma chave e armazená-lo no armazenamento da web. Quando você executa este aplicativo agora, ele mostra o objeto do cliente representado como uma string JSON.

A disponibilidade do armazenamento na Web local pode melhorar a experiência do usuário final e o desempenho de seus aplicativos da Web, salvando viagens de ida e volta ao servidor para recuperar ou armazenar dados temporários.

Mesmo quando você usa localStorage em vez de sessionStorage, pense no armazenamento como temporário e projete seus aplicativos para que eles possam voltar aos valores e comportamento padrão se o usuário limpar o armazenamento na Web. O armazenamento na Web fornece uma maneira de disponibilizar dados localmente e até persistir nas sessões do navegador.

**API do AppCache**

A API do AppCache disponibiliza conteúdo e páginas da Web, mesmo quando um aplicativo da Web está no modo offline. O AppCache armazena arquivos no cache do aplicativo no navegador. Assim como no armazenamento na Web, a quantidade de dados que o navegador pode armazenar localmente é limitada para uso offline. Dois componentes compõem a API do AppCache: o **ARQUIVO DE MANIFESTO** e uma **API JAVASCRIPT** para suportá-lo.

**Usando o manifesto do AppCache**

Para especificar que uma página deve estar disponível para uso offline basta adicionar um atributo ao elemento HTML na página. Exemplo abaixo:

<html *manifest*="webApp.appcache">

...conteúdo

</html>

O atributo *manifest* no elemento html informa ao navegador que esta página da web precisa estar disponível offline. ***O valor do atributo manifesto aponta para um arquivo manifesto*** com extensão **.*appcache***.

O arquivo de manifesto do cache do aplicativo deve listar todos os arquivos e recursos necessários para serem armazenados para uso offline. Quando o navegador analisa o atributo manifest do elemento html, ele baixa o manifesto e o armazena localmente. Ele também garante que faça o download de todos os arquivos listados no manifesto para que eles estejam disponíveis offline. O arquivo de manifesto contém três seções. De forma resumida é assim:

**- Cache:** Serve tudo off-line

**- Network:** Servir da rede

**- Fallback:** Primeiro tenta servir da rede se não funcionar ele busca local.

A *primeira linha* em um arquivo de manifesto deve ser **CACHE MANIFEST**. O arquivo de manifesto, como em qualquer arquivo de código, pode ter linhas de comentário adicionadas a ele para explicações adicionais, conforme indicado pelo símbolo **#.**

A seção **CACHE** lista todos os recursos que devem ser armazenados em cache offline. Isso deve incluir todos os arquivos CSS, JPG, arquivos de vídeo e áudio e qualquer outro recurso necessário para que a página funcione corretamente. Se você omitir um item do arquivo de manifesto, ele não será armazenado em cache, o que pode resultar em comportamento inesperado quando o aplicativo for executado offline.

A seção **NETWORK** declara quaisquer recursos que devem estar disponíveis na Internet. Esses itens *NÃO PODEM* ser armazenados em cache. Tudo o que a página exige da Internet, como elementos incorporados de terceiros, deve ser listado aqui. Se esse recurso não estiver listado aqui, o navegador não saberá procurar na Internet quando estiver no modo offline. Quando o navegador está no modo offline, ele não tenta acessar a Internet para nada, a menos que esteja listado na seção NETWORK.

A seção **FALLBACK** permite que você forneça instruções de fallback ao navegador, caso um item não esteja disponível no cache e o navegador esteja no modo offline. No arquivo de exemplo, se *login.html* não estiver disponível no cache, renderize fallback-login.html. Você pode usar atalhos na seção FALLBACK para fornecer redirecionamentos mais gerais, como o seguinte:

/resources /resource.jpg

Isso indica ao navegador que, se o navegador estiver offline e não puder acessar nada na pasta de recursos, ele deverá substituir as referências aos itens na pasta de recursos por resource.jpg. Observe que resource.jpg é armazenado em cache porque é especificado na seção FALLBACK. Você não precisa especificar também resource.jpg na seção CACHE. Nesse caso vai mostrar uma imagem padrão.

**Usando a API JavaScript AppCache**

Como no armazenamento na Web, o cache do aplicativo está disponível em JavaScript como um objeto global. O código a seguir obtém uma referência ao objeto global do AppCache:

var appCache = window.applicationCache;

Quando o cache do aplicativo disponibiliza as páginas off-line, você pode verificar seu status avaliando a propriedade status do objeto AppCache.

A propriedade status pode ser um dos valores listados:

|  |  |
| --- | --- |
| **Status** | **Description** |
| Uncached | O aplicativo da web não está associado a um manifesto do aplicativo. |
| Idle | A atividade de armazenamento em cache está ociosa e a cópia mais atualizada do cache está sendo usada. |
| Checking | O manifesto do aplicativo está sendo verificado quanto a atualizações. |
| Downloading | Os recursos no manifesto do aplicativo estão sendo baixados. |
| UpdateReady | Os recursos listados no manifesto foram baixados com sucesso. |
| Obsolete | O manifesto não pode mais ser baixado; portanto, o cache do aplicativo está sendo excluído. |

Dois métodos no objeto AppCache podem ser úteis.

|  |  |
| --- | --- |
| **Status** | **Description** |
| swapCache | Indica que o cache deve ser substituído por uma versão mais recente. |
| update | Diz ao navegador para atualizar o cache se uma atualização estiver disponível. |

Quando o método ***update*** é chamado, uma atualização para o cache é preparada. Quando estiver pronto para o download, o status do cache do aplicativo será alterado para ***UpdateReady***. Quando isso é definido, uma chamada para o método ***swapCache*** informa ao aplicativo para alternar para o cache mais recente.

DICA

A chamada para o método de atualização é assíncrona. Portanto, você deve manipular o evento onupdateready para determinar quando a atualização concluiu o processo de download.

Além das propriedades e métodos, o objeto AppCache pode gerar uma série de eventos que você pode manipular. O cache do aplicativo normalmente opera em segundo plano e você não precisa desses eventos. No entanto, em alguns casos, manipular alguns dos eventos e forçar uma atualização pode ser útil. OS eventos disponíveis são:

|  |  |
| --- | --- |
| **Event** | **Description** |
| onchecking | O navegador está procurando uma atualização no manifesto do aplicativo ou o aplicativo está sendo armazenado em cache pela primeira vez. |
| onnoupdate | O manifesto do aplicativo não possui atualização disponível. |
| ondownloading | O navegador está baixando o que foi solicitado a fazer pelo arquivo de manifesto. |
| onprogress | Os arquivos estão sendo baixados no cache offline. Este evento é acionado periodicamente para relatar o progresso. |
| oncached | O download do cache foi concluído. |
| onupdateready | Os recursos listados no manifesto foram novamente baixados e o método swapCache pode ser chamado. |
| onobsolete | Um arquivo de manifesto não está mais disponível. |
| onerror | Ocorreu um erro. Isso pode resultar de muitas coisas. Registro adequado é necessário para obter as informações e resolver. |

O cenário mais comum é manipular o método onupdateready e, em seguida, fazer uma chamada para o método swapCache .

    <script *type*="text/javascript">

        window.onload = function() {

            var appCache = window.applicationCache;

            appCache.oncached = function(*e*) {

                alert("cache successfully downloaded.");

            };

            appCache.onupdateready = function(*e*) {

                appCache.swapCache();

            };

        }

    </script>

O uso do cache do aplicativo *é mais sobre configuração* do que sobre codificação. No entanto, é importante que você saiba que a API está disponível para cenários avançados em que você precisa de mais controle sobre o processo ou quando precisa receber informações oportunas sobre o processo, como lidar com o evento *onprogress*.

**Usando a API de Geolocalização**

Os serviços de localização dependem do *Sistema de Posicionamento Global (GPS)*, *endereços IP* e *outras características do dispositivo*. Você pode tirar proveito da localização geográfica em aplicativos da Web, aproveitando os navegadores que suportam a API de localização geográfica.

Você pode obter uma referência à API de geolocalização na propriedade window.navigator, da seguinte maneira:

var geoLocator = window.navigator.geolocation;

Esse código salva uma referência à API de geolocalização em uma variável para fornecer acesso abreviado à API durante uso futuro.

A API de geolocalização suporta três métodos principais que você usa para interagir com ela:

* + - 1. **getCurrentPosition:** Posição atual do usuário.
      2. **watchPosition**
      3. **clearWatch**

**Usando o método getCurrentPosition**

Aqui está um exemplo de uso do método getCurrentPosition:

getCurrentPosition (positionCallback, [positionErrorCallback], [positionOptions])

Você usa *getCurrentPosition* para obter exatamente o que o nome indica - a posição atual do usuário. ***Este método utiliza um parâmetro necessário e dois parâmetros opcionais***.

* O primeiro parâmetro é um método de retorno de chamada que a API chama após a determinação da posição atual. positionCallback
* O segundo parâmetro é opcional, mas também é uma função de retorno de chamada(callback) quando ocorre um erro. [positionErrorCallback]

O método de retorno de chamada especificado aqui deve lidar com quaisquer erros que possam ocorrer ao tentar obter a posição atual. O *último parâmetro opcional* é um objeto especial chamado PositionOptions, que permite definir algumas opções especiais que controlam como o método getCurrentPosition se comporta.

A tabela a seguir lista as propriedades disponíveis no objeto PositionOptions:

|  |  |
| --- | --- |
| **Property** | **Description** |
| enableHighAccuracy | Isso faz com que o método consuma mais recursos se definido como true. O padrão é falso. Se verdadeiro, o método *getCurrentPosition* tenta chegar o mais próximo possível do local real. |
| timeout | Isso especifica um período de tempo limite para quanto tempo o método *getCurrentPosition* pode levar para ser concluído. Esse número é medido em milissegundos e o padrão é zero. Um valor zero representa infinito. |
| maximumAge | Se isso estiver definido, a API será instruída a usar um resultado em cache, se disponível, em vez de fazer uma nova chamada para obter a posição atual. O padrão é zero, portanto, sempre é feita uma nova chamada. Se *maximumAge* estiver definido como um valor e o cache não for mais antigo que a idade permitida, a cópia em cache será usada. Este valor é medido em milissegundos. |

O objeto de posição expõe duas propriedades:

* A propriedade *timestamp* indica a hora em que as cordas foram recebidas.
* A propriedade *coords* é ela própria um objeto de coordenadas que contém a latitude, longitude, altitude, rumo e velocidade da posição atual do dispositivo e / ou em relação à última posição adquirida.

O objeto positionError contém duas propriedades: uma para o código e outra para a mensagem.

**Usando o método watchPosition**

O método watchPosition fornece um mecanismo interno que pesquisa continuamente a posição atual. Aqui está um exemplo de como usar o método:

geoLocator.watchPosition (successCallBack, errorCallback, positionOptions)

O método watchPosition usa o mesmo conjunto de parâmetros que o método getCurrentPosition, mas retorna um objeto watchPosition:

var watcher = geoLocator.watchPosition ...

A variável watcher mantém uma referência à watchPosition . O método chama o método de retorno de chamada bem-sucedido toda vez que a API de localização geográfica detecta um novo local. A pesquisa continua para sempre, a menos que você pare.

Você pode cancelar a pesquisa chamando o método clearWatch . O código abaixo mostra a solução completa para o método watchPosition .

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

Um componente-chave de qualquer linguagem de programação é como ele usa variáveis, e o JavaScript não é exceção. Para usar variáveis efetivamente, você deve entender seu escopo e vida útil. Declarar variáveis e instanciar objetos consome recursos. O principal recurso do sistema usado para variáveis é a memória. Quanto mais memória um aplicativo usa, maior o potencial de o uso de outros recursos também aumentar - como a energia da bateria para suportar o uso adicional de memória. Quando os aplicativos esgotam a bateria, é menos provável que os usuários os usem.

OBS.: Quando uma variável é criada em JS, ela pode ser criada de forma global ou interna.

• Global: todo mundo da página pode acessar a variável ou o objeto.

• Interna: Se a variável for criada dentro de uma função, ela fica acessível só dentro da função.

**Estabelecer a vida útil das variáveis e escopo da variável**

As variáveis começam sua vida com uma declaração. O restante de sua vida útil no aplicativo depende de onde as variáveis são declaradas.

Para declarar uma variável em JavaScript, use a palavra-chave var.

        var myVariable;

Você pode declarar muitas variáveis simultaneamente. Por exemplo, o código a seguir declara três variáveis:

        var x, y, z;

Você também pode inicializar suas variáveis alinhadas com a declaração, fornecendo valores imediatos não padrão:

        var x = 0,0, y = 0,0, z = 0,0

Até que uma variável seja inicializada, ela não está realmente "viva" - ela tem um valor indefinido. Depois que uma variável está disponível para uso, ela é considerada "dentro do escopo". A duração pela qual a variável permanece no escopo depende de onde a variável é declarada. Uma variável com escopo global está disponível em toda a página da web. Uma variável com escopo local está disponível apenas dentro de um contexto especificado.

**Evite usar o espaço para nome global**

O espaço para nome global é o local onde vivem todas as bibliotecas JavaScript nativas. O espaço para nome global inclui espaços para nome aninhados, como Math, WebSocket e JSON.

O espaço para nome global está disponível para todo o código em uma sessão do aplicativo. Com o crescente número de bibliotecas de terceiros em uso, o potencial de conflitos de nomes aumenta.

Definindo um espaço para nome com o mesmo nome, o tempo de execução JavaScript não pode identificar qual espaço para nome eles pretendem usar. É por isso que é importante manter seus objetos fora do namespace global.

Uma estratégia para evitar colisões de nomes é criar seus próprios namespaces para suas bibliotecas JavaScript.

**Aproveitando a palavra-chave *this***

A palavra-chave this é um termo especial que permite que os desenvolvedores de JavaScript façam referência diretamente ao objeto que o contém.

Nesse snippet de código, a primeira referência está no espaço para nome global - portanto, fornece uma referência direta ao espaço para nome global. À medida que você avança no código, o contexto dessa palavra-chave muda. No evento onload da janela, isso se refere ao objeto da janela (sim, isso está no espaço de nomes global, mas continue lendo). Dentro da função onclick, a palavra-chave this refere-se ao elemento div retornado do método getElementById. A palavra-chave this sempre se refere ao objeto que contém o código em execução no momento. Esse é o seu contexto.

**Resumo objetivo**

* As variáveis são indefinidas até serem inicializadas.
* As variáveis têm escopo definido e acessíveis, dependendo de onde são declaradas. Se eles estiverem dentro de uma função, por exemplo, eles são locais para a função.
* Passar parâmetros é a única maneira de disponibilizar uma variável local em outra função.
* O espaço para nome global não deve ser usado porque é compartilhado por todos.
* Você deve aplicar um espaço para nome a objetos personalizados para evitar conflitos no espaço para nome global.
* A palavra-chave this fornece acesso direto ao objeto que gerou o evento.

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

JavaScript por ser uma linguagem de programação orientada a objetos, significa que para desenvolver aplicações em javaScript, é preciso entender como trabalhar com objetos.

Existem dois tipos de objetos em JavaScript:

* **Objetos JavaScript nativos**, fornecidos com o próprio JavaScript.
* **Objetos personalizados**, criados pelos desenvolvedores para representar suas próprias construções e comportamentos de dados.

Em alguns casos, a criação de um objeto totalmente novo não é necessária. Você pode basear objetos em outros objetos (se eles forem um subtipo desse objeto) usando a herança de objetos, na qual um objeto herda todos os atributos e comportamentos de outro objeto, mas também pode implementar aspectos adicionais exclusivos dele.

Os objetos encapsulam a funcionalidade e informam informações relevantes para eles. A funcionalidade é fornecida na forma de métodos, enquanto as informações de estado são fornecidas na forma de propriedades. Este objetivo examina o trabalho com objetos em JavaScript.

**Implementar objetos nativos**

Alguns objetos nativos estão disponíveis estaticamente, o que significa que você não precisa criar uma instância deles. Outros exigem que você crie uma instância. Você pode encontrar os dois tipos entre objetos no espaço para nome global. Um exemplo de um objeto estático é o Math, que está disponível no espaço de nomes global.

        var squareValue = Math.sqrt(144);

**Objeto Array**

Outros objetos, como Array, é preciso criar uma instância para trabalhar com eles.

   var listofPrimeNumbers = **new** *Array*(1, 2, 3, 5, 7, 11, 13, 17, 19, 23);

A palavra **new** é usada para instanciar um objeto e informa ao tempo de execução para alocar um novo objeto do tipo especificado. Nesse exemplo a cima, um novo objeto Array está sendo solicitado.

A lista após o tipo de objeto Array é chamada de construtor do objeto. Esta informação pode ser passada para o objeto como parâmetros para construir o estado inicial do objeto.

Alguns objetos têm muitos construtores para escolher, com diferentes conjuntos de parâmetros. A adição de vários construtores é chamada de *overloaded constructor.*.

**Objeto e métodos do wrapper**

Estes agrupam tipos nativos, por exemplo. Tipos nativos são definidos como número inteiro, string, caractere etc. Esses tipos não possuem métodos ou funcionalidades;

A sintaxe se aplica a objetos nativos e objetos personalizados. Objetos personalizados são criados pelo desenvolvedor, enquanto objetos nativos são fornecidos pelo JavaScript principal.

**Criar objetos personalizados e propriedades personalizadas para objetos nativos usando protótipos e funções**

**Herdar de um objeto**

**Implementar métodos nativos e métodos personalizados**