EAD UNISANTA

FRONT END (HTML5 E CSS3)

Dr. Fábio Mossini

GUIA DA
DISCIPLINA

1. WEB

Objetivos

- Iniciar os estudos sobre WEB
- Apresentar as definições de WEB e HTML

Introdução

Bem-vindo, elaborei o texto a seguir como uma referência e roteiro para o seu estudo, não pretendo exaurir nenhum dos temas aqui apresentados. Na verdade, recomendo que busque e amplie os tópicos apresentados.

Vamos juntos!

1.1. Bases fundamentais da WEB.

A Web é um conjunto de documentos acessíveis por meio da de uma mídia digital (Internet). Esses documentos (ou páginas Web) estão baseados no modelo de hipertexto, que permite navegação e consulta a outros documentos através de *links*, ligações entre as páginas.

A volume de dados, documentos e informações que está disponível no universo da Internet é gigantesco e está em constante expansão, podem ser encontrados os mais variados conteúdos. Ao consideraremos as dimensões da internet é evidente e necessária a condição de organização. Podemos descrever a World Wide Web como um sistema de informação em hipertexto, gráfico, distribuído, independente de plataforma, dinâmico, interativo e global, acessível por mídia digital, o grande atributo do hipertexto é sua habilidade de navegação, leitura e interconexão.

A Web em seu desenvolvimento tem ampliado a oferta de recursos visuais e de animação, permitindo imaginar e construir um Metaverso (um novo universo, paralelo ao atual). Outras características da WEB são: sua independência de plataforma para acesso: computadores, celulares, tabletes etc. Todo o conteúdo está distribuído em vários sites e servidores e ordenados por meio da URLS (endereços). Diferente de outras mídias a Internet oferece uma imersão, reagindo as interações e ações dos usuários.

Uma página Web é um elemento específico de uma apresentação da Web que está contida e organizada em uma estrutura. A primeira página de uma apresentação é chamada de *home page (index.htm)*.

Uma apresentação da Web é um conjunto de páginas Web estruturadas sobre um determinado conteúdo, este conteúdo é aquilo que você elaborou e disponibilizou em seu site, por exemplo. Informações, texto de ficção, imagens, ilustrações, programas, textos humorísticos, diagramas, jogos etc. Tudo isso é conteúdo. Esse conteúdo vai precisar seguir certos padrões em sua produção como linguagens e os padrões da Internet para constituir um site e ser visível e acessível.

1.1.1. Como se cria uma página web?

Uma página Web é composta de textos e comandos especiais (*tags*) de HTML, um acrônimo de *Hypertext Markup Language*. Essa linguagem é pouco complexa e tem como finalidade formatar o texto que será exibido e estruturar suas ligações entre as outras páginas Web, originando um hipertexto. Por uma questão de concepção e desenvolvimento, as instruções e conteúdos, dependem de um navegador (*Browser*), software capaz de interpretar as *tags* e exibir o conteúdo e oferecer as interações planejadas. Essa ligação entre o desenvolvimento do HTML e o navegador é fundamental e deve ser usado para testar e avaliar o resultado esperado de sua codificação. Dessa forma, uma das etapas de desenvolvimento de sua codificação HTML, pode ser feita em *offline*, com o navegador de seu computador e um editor de HTML com o Sublime (sublimetext.com).

1.1.2. Servidor WEB

Para exibir páginas na Web e navegar por elas, você precisará apenas de um navegador da Web. Para divulgar páginas na Web, você precisará, na maioria dos casos, de um servidor Web. Servidor Web é um serviço disponibilizado por empresas especializadas que hospedarão seu conteúdo e disponibilizarão quando solicitado, pelos acessos dos usuários e seus navegadores, quando conectados à internet.

1.1.3. Protocolos internet

Para que os computadores se comuniquem eles precisam seguir conjuntos de regras chamados protocolos, que organizam e distribuem o acesso aos conteúdos da WEB.

Internet Protocol (IP)

Este é um dos protocolos fundamentais. O IP é o sistema que define o "local", ou endereço IP, das redes que compõem a Internet. E um certo sentido, o IP forma o "mapa" da Internet e cada rede pode ser contatada em um ponto localizado nesse mapa.

Transmission Control Protocol (TCP)

Este é o protocolo que define a estrutura dos dados transmitidos.

File Transfer Protocol (FTP)

Protocolo desenvolvido para a transmissão de mensagens longas (grandes volumes) entre duas pontas.

Hypertext Markup Language e Hypertext Transfer Protocol (HTML e HTTP)

Juntos eles coordenam a World Wide Web. O HTML define um método de incluir formatação em arquivos de texto para que, quando exibidos com um browser. O HTTP define a maneira como os arquivos HTML devem ser enviados e recebidos.

1.2. Passos para criar meu projeto

1.2.1. O será divulgado? Produto, serviço, pessoas, empresas

Qual o objetivo do seu projeto? O que seu cliente deseja? Que tipo de conteúdo você deseja apresentar na Web? Considere os seguintes exemplos:

Hobbies ou interesses especiais: filmes, motocicletas etc.

Publicações: como jornais, revistas.

Perfis de empresa: o que uma empresa faz ou vende etc.

Documentação On-line: desde manuais, guias de treinamento, dicionários etc.

Catálogos de compras: comercialização de artigos.

Lojas on-line, *marketplace*

Pesquisas: a interatividade com o usuário através de formulários.

Educação: universidades, escolas e empresas particulares oferecem o ensino através da Web.

Dessa forma o limite para a Web é sua capacidade de imaginar e criar. Dessa forma não desista de sua ideia, nessa apresenta lista estão algumas e apenas as mais comuns.

1.2.2. Estabeleça seus objetivos!

Considere que antes de começar a codificar, você deve pensar o que quero colocar no meu projeto? Como será estrutura e imersão? Quem é o público-alvo do projeto? Entenda que um site é apenas uma parte do processo de comunicação e identidade do seu cliente / empresa / produto.

Compreender os objetivos do seu cliente e construir um projeto coerente é uma necessidade, para elaborar seu projeto será necessário dedicação e estudo para organizar e codificar as páginas que vão compor seu projeto e atender os objetivos do demandante. Caso vá desenvolver uma apresentação Web para uma empresa ou pessoas é importante que você colha junto ao seu cliente seus objetivos, ideias, a forma que imagina sua página etc. Assim, ficará bem mais fácil de começar seu trabalho.

1.2.3. Divida seu conteúdo em tópicos

Crie uma lista com os principais tópicos, a princípio não importa a ordem. Esta é uma forma de começar a se organizar. Sua lista poderá ter quantos tópicos desejar, mas se perca listando uma quantidade enorme de tópicos (seu leitor poderá se cansar e se perder em meio a tantas opções). Considere fazer um mapa mental para perceber como os assuntos, tópicos ou itens se relacionam e podem ser estruturados / apresentados.

1.2.4. Organização e navegação

A estrutura e organização do seu projeto é um dos ingredientes para realizar um site funcional, intuitivo e que causará impacto desejado. Os elementos a seguir devem ser considerados na elaboração:

Hierarquias

A maneira mais fácil e mais lógica de estruturar os seus documentos é utilizar uma ordenação. Por exemplo as hierarquias, dependências e os menus adaptam-se especialmente bem aos documentos on-line e de hipertexto, facilitando a navegação.

Home Page

Em uma organização hierárquica, é fácil para os leitores descobrir a posição em que se encontram na estrutura. Nas hierarquias, a *home page* fornece uma visão geral do conteúdo que está subordinado a ela e ainda define os principais vínculos às páginas dos níveis inferiores da hierarquia. Este tipo de estrutura oferece um risco mínimo de ficar perdido, além de ser uma forma mais fácil de localizar informações, organize e divida de forma simples e objetiva, use a hierarquia para melhorar a visibilidade e coerência de seu conteúdo.

Linear

Muito semelhante à forma como são organizados documentos impressos. Neste tipo de estrutura a *home page* é o título, ou introdução, e todas as outras páginas acompanham-na em sequência com vínculos que levam de uma página a outra, normalmente com opções de avançar e retroceder, como a experiencia de folhear um livro.

Uma organização linear é rígida e limita a ordem de consulta / acesso dos leitores as informações. As estruturas lineares são ideais para apresentar, no ambiente on-line, um material que já tenha esse tipo de estrutura no ambiente off-line. Como por exemplo: instruções passo-a-passo.

Estrutura Linear com Alternativas

Você pode tornar a estrutura linear menos rígida permitindo que o leitor se desvie do caminho principal. Pode ter, por exemplo, uma estrutura linear com ramificações alternativas que partam de um único tronco. As ramificações podem se reunir ao tronco principal em algum ponto mais adiante, em um nível mais baixo da estrutura, ou continuar se ramificando em níveis inferiores seguindo caminhos próprios até chegar a um "fim". Dependendo do volume de conteúdo você pode considerar um modelo estação de metrô, com ramais principais e secundários, interligados por coerência e estratégias de comunicação / divulgação.

Modelar e apresentar sua ideia

1.2.5. O STORYBOARD

Para continuar seu planejamento e oferecer uma visibilidade para as suas ideias é recomendável utilizar uma *storyboard,* ferramenta que consiste em definir o conteúdo que

será apresentado em cada uma das páginas e criar alguns vínculos simples que possibilitem a navegação por essas páginas (folhas de papel). O *StoryBoard* de uma é um conceito emprestado do cinema, no qual cada cena e cada tomada de câmera é esboçada / desenhada na ordem em que ocorre no filme. O *storyboard* fornece uma ordem e um plano globais para o projeto / filme. Um *storyboard* é um mapa do esforço e das ligações entre as partes e o objetivo do projeto.

Dicas para seu Storyboard

1. Cada tópico em uma página (com coerência entre volume de itens e conteúdo).
2. Defina a forma de navegação entre as páginas. Se houver formas alternativas, torne a navegação para os leitores a mais intuitiva possível.
3. A *home page* é seu cartão de visita, escolha com clareza o que será apresentado.
4. Tenha sempre em mente os objetivos de seu projeto.

1.2.6. Diagramação

É a referência ao consumo do espaço visível, você precisa imaginar que existem arranjos mais interessantes, atraentes e adequados ao conteúdo que será veiculado, dessa forma a disposição de imagens, textos, vídeos devem ser estudados. Fazer uma diagramação é estudar a disposição de elementos que comporão uma página. Deve ser observado o tamanho das fontes, disposição das imagens, forma como o texto será apresentado etc. Uma boa diagramação também garante o retorno do internauta.

1.2.7. Web design e visão do projeto

A maioria das pessoas associam Design unicamente a aparência e visual ou ao projeto gráfico de um WebSite, isso é incorreto! Ao descrever um Web Design estamos observando todos os aspectos da construção de um site, desde o desenho de sua estrutura de navegação e forma de disponibilização da informação até o desenvolvimento do projeto gráfico e a interação do usuário. A construção de um Website deve, inicialmente, ser entendida como um projeto, composto de fases e desenvolvido por pessoas de experiencias diferentes.

1.2.8. Conteúdo e forma

Um site é o resultado da junção de conteúdo e forma, por isso é essencial considerar:

a. Por que acessamos um site?
b. O nos faz retornar ao site?

Estas perguntas devem sempre estar na cabeça durante o desenvolvimento, porque elas focam o público-alvo e a relevância do site / conteúdo / interação. Por isso: Quais são as prioridades e interesse do público? E sobre o conteúdo temos três pontos: 1º Elemento central - conteúdo; 2º Design do site (estrutura de navegação e projeto gráfico) e.3º Atualização e relevância do site.

1.2.9. EQUIPE PARA CONSTRUÇÃO DE UM WEBSITE

Ao considerar um grande projeto ou cliente podemos considerar que a construção de um Website deverá exigir uma equipe com *backgrounds* diferentes. A equipe de profissionais adequada seria composta:

a) Analistas de sistemas (com experiência em gerência de Informação), responsável pela organização da informação;
b) Design Gráfico, responsável pelo projeto gráfico;
c) Programadores, que, dependendo do tipo de site, podem ser programadores HTML, Java, JavaScript, CGI, etc;
d) Assistentes, que auxiliam o trabalho do Analista de sistemas ou do Design Gráfico (ilustradores, digitadores, etc.).
e) Coordenador, responsável em fazer o trabalho fluir, dentro de um cronograma aproveitando o máximo de cada profissional.
f) WebMaster, que será responsável pela administração do site.

1.2.10. FASES NA CONSTRUÇÃO DE UM WEBSITE

1ª Organização da Informação

A primeira etapa é a organização da informação que irá compor o projeto. Deverá ser feita uma coleta formal de toda informação, para a etapa seguinte, onde se iniciará o design do site.

2ª Conceituação do site e definição da estrutura.

Desenhar fluxogramas de navegação, mostrando como as páginas estão conectadas gerando uma visão integral do site. No futuro, isso facilitará também o processo de modificar, acrescentar ou retirar páginas.

3ª Montagem do Site

Uma vez definida a estrutura de navegação, os membros da equipe poderão então trabalhar cada um em sua área: o Design Gráfico e seus Assistentes desenvolverão o projeto gráfico, enquanto o Analista de Sistemas e programadores montarão a estrutura de navegação (seja em HTML, Java, JavaScript ou no que tiver sido escolhido como mais adequado para a construção do site).

4ª Testes

Antes de ser disponibilizado para o público em geral é importante testar a navegação de todo o site, checando os links de texto e imagens. É importante entrar no site a partir de várias plataformas e conexões diferentes, para testar a velocidade de carga e consertar falhas de diagramação, que pode mudar, de acordo com o tamanho de monitor e resolução usada, isto é, usar como visitante.

2. HTML – Parte 1

Objetivos

- Descobrir o HTML
- Compreender as funções básicas
- Realizar os exemplos

Introdução

A seguir os primeiros passos para entender e usar o HTML, sua origem e funções básicas para ser consideradas na elaboração de um projeto. Segundo a *World Wide Web Comitee,* W3C, a internet tem por fundamento: Esquemas de nomes para localização de fontes de informação – URI, protocolo de acesso as fontes – HTTP e a linguagem de hipertexto – HTML.

Vamos juntos!

2.1. HTML

A HTML é uma linguagem de marcação, desenvolvida para criar documentos, os quais conterão *tags* especiais no início e no final de determinadas palavras ou parágrafos. Essas *tags* indicam as diversas partes da página e produzem diferentes efeitos ao serem renderizadas no navegador. As *tags* normalmente são especificadas em pares, delimitando um texto que sofrerá algum tipo de formatação. As *tags* são identificadas por estarem entre os sinais < > e < / >. Entre os sinais < > são especificados os comandos propriamente ditos. No caso de *tags* que necessitam envolver um texto, sua finalização deve ser utilizada a barra de divisão /, indicando que a *tag* está finalizando a marcação de um texto. O formato genérico de uma *tag* é: <nome da *tag*> texto</nome da *tag*>

Mas algumas *tags* não possuem finalização. Vejamos agora alguns comandos e o tipo de efeito que estes causam ao texto quando interpretados pelo browser. Assim como outras linguagens, a HTML possui uma estrutura básica para seus programas. Para que um browser intérprete corretamente o programa, ele deve possuir alguns comandos básicos que sempre presentes. Alguns browsers até dispensam seu uso, porém é melhor assumir como parte fundamental do programa tais comandos. Um arquivo em HTML possui três

partes básicas, a estrutura principal, o cabeçalho e o corpo do arquivo. Todo programa deve iniciar com o comando, *tag*, <html> e ser encerrado com o comando </html>. Esse par de comandos é essencial. A área de cabeçalho é delimitada pelo par de comandos <head> e </head>. Estes comandos para cabeçalho são usados para especificar alguns poucos comandos da linguagem. Eles são opcionais, ou seja, um programa HTML pode funcionar sem eles. Mas é conveniente usá-los, pois o título da página é acrescentado através deles. E ainda temos as *tag* <Title> e </Title>, estes comandos delimitam o texto que irá aparecer na barra de título do browser. A maioria dos comandos será especificado no corpo do programa que é delimitado pelas *tags* <body> e </body>. É um comando obrigatório.

<HTML> </HTML> Esta *tag* marca o início do arquivo HTML

<HEAD> </HEAD> Esta *tag* marca início e fim de cabeçalho

<TITLE> </TITLE> Esta *tag* delimita o texto que irá ser visualizado na barra de título do browser.

<BODY> </BODY> Esta *tag* delimita o corpo do arquivo, aonde a maior parte do comando serão marcados.

2.1.1. OS PRINCIPAIS ELEMENTOS DE UMA PÁGINA HTML

Uma página HTML é composta basicamente de títulos, textos, parágrafos, imagens e links, responsáveis pela chamada de outras páginas para a tela. Todos esses elementos são posicionados na página por meio de comandos específicos da linguagem.

Título

É o texto que aparece na barra de título do browser.

Imagem

São figuras, desenhos e fotos usados para ilustrar a página.

Texto

É a informação mais comum dentro da página. Pode ser formatado através de vários comandos.

Link

É um trecho que aparece destacado do restante do texto, normalmente sublinhado e com outra cor. Ao clicar no link, o browser acessa outra região da página atual ou uma página localizada em qualquer lugar da Internet.

Título

O título de uma página web indica qual o assunto abordado e irá aparecer na barra de título do browser. Para atribuir um título a página você deverá utilizar a *tag*. Esta *tag* sempre será incluída no cabeçalho (entre as *tags* e descrevem a mesma). Considerações sobre o título: 1. Você poderá ter apenas um título, 2. Texto deverá ser simples e curto e não poderá ter outras *tags* e 3. Escolha um texto curto e que descreva o conteúdo da página.

Cabeçalhos

Os cabeçalhos são usados para dividir seções do texto, como capítulos de um livro. A HTML divide seis tamanhos de cabeçalhos, de H1 a H6, que aplicam um tamanho de fonte diferenciado no texto que envolvem, uma linha antes e depois além disso dão um efeito de negrito. O maior tamanho é o H1 e o menor é o H6. <H1> Texto </H1>

Experimente o seguinte código:

```
</html>
<head><title>Cabeçalhos</title></head>
<body>
<h1>C</h1>
<h2>A</h2>
<h3>B</h3>
<h4>E</h4>
<h5>ç</h5>
<h6>A</h6>
<h5>L</h5>
<h4>H</h4>
<h3>O</h3>
<h2>S</h2>
</body>
</html>
```

Vejamos um exemplo de outro código em HTML:

```
<html>
<head><b> Meu site é lindo</b></HEAD>
<body>
<p>
```

Em nosso exemplo introduza também as seguintes *tags*. Os comandos:

<B> </B> Aplica o estilo negrito ao texto

<I> </I> Aplica o estilo itálico ao texto.

<P> </P> Inicia um novo parágrafo.

 Faz uma quebra de linha

Parágrafos

Para definir o início de um novo parágrafo, ou seja, avançar uma linha em branco e iniciar o texto na Segunda linha após o final do parágrafo anterior, deve ser usado o comando . Esse comando pode aparecer individualmente ou em par: . Quando um comando é inserido, o browser irá sempre avançar uma linha em branco, posiciona-se na linha seguinte ao comando .

Quebra de linha

O comando faz uma quebra de linha. Este comando inseri uma linha em branco no seu local ou na linha seguinte à qual ele foi inserido, caso esteja no meio de uma linha e não em seu final. A função deste comando é avançar para a linha imediatamente após aquela em que ele ocorre.

Assim como em um editor de texto, em HTML você poderá utilizar efeitos em seu texto. Como já percebeu os comandos, *tags*, trabalham em contêiner. Alguns comandos de estilo:

TAG	SINTAXE	FUNÇÃO
STRONG	texto	Similar ao negrito
TYPERWRITER	<tt>texto</tt>	Deixa o texto com espaçamento regular
BIG	<big>texto</big>	Aumenta a fonte e aplica negrito
SMALL	texto	Reduz e altera a fonte
SOBRESCRITO	<sup>texto>/sup>	Eleva o texto e diminui seu corpo
SUBESCRITO	_{texto}	Rebaixa o texto e diminui seu corpo
BLINK	<blink>texto</blink>	Faz com que o texto pisque
NEGRITO	texto	Aplica o estilo negrito
ITÁLICO	<I>texto</I>	Aplica o estilo itálico

SUBLINHADO	<u>texto</u>	Aplica um sublinhado (em alguns browsers está *tag* não funciona)

Estes são os mais utilizados, porém há outros que merecem atenção. Para alterar a formatação de um texto em HTML existem dois tipos de estilos que devem ser observados: o estilo lógico e o estilo físico.

Estilo lógico

As *tags* deste tipo indicam como o texto destacado deve ser utilizado e não como será apresentado. Este estilo não indica como o texto será formatado e sim como será utilizado no documento. Não é possível garantir que um texto destacado que utilize *tags* deste tipo sempre será apresentado em negrito ou itálico, por exemplo. Dependerá do browser. Algumas *tags* de estilo lógico utilizadas em HTML padrão:

<EM>

Indica que os caracteres deverão ser enfatizados de alguma forma. De forma diferente do restante do texto. Geralmente em itálico

<STRONG>

Esta *tag* enfatiza ainda mais que a anterior. Em negrito.

<CODE>

Esta *tag* indica um código de exemplo a ser exposto.

<SAMP>

Esta *tag* indica texto de exemplo. Quando você quer dar um exemplo de endereço para a Internet sem criar

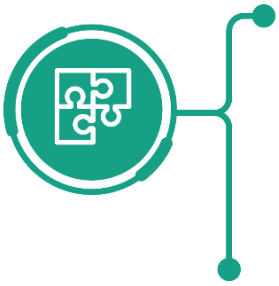
Estilos físicos

Este estilo de *tag* realmente altera a formatação do texto, no estilo anterior você não possui garantia que o texto irá ser visualizado da forma que planejou. Neste estilo ele será visualizado da forma que atribuir as *tags* a ele. Algumas *tags* de estilo físico para HTML padrão:

<B> </B>	coloca o texto em negrito
<I> </I>	coloca o texto em itálico
<TT> </TT>	fonte de máquina de escrever com espaçamento uniforme.
<u> </u>	coloca o texto sublinhado

<S> </S>	coloca o texto tachado
<SMALL> </SMALL>	o texto irá aparecer menor que o restante ao redor
<BIG> </BIG>	o texto irá aparecer maior que o restante ao redor
	coloca o texto subscrito
	coloca o texto sobrescrito

Optando por utilizar *tags* do estilo físico, caso o navegador não reconheça uma das *tags* ele a irá substituir por outra equivalente ou ignorará a formatação.



Entendendo

O HTML foi criado por Tim Berners-Lee, como uma ferramenta que permitia acrescentar alguns elementos nos textos usados por ele e um grupo de pesquisadores, na publicação dos seus trabalhos em uma época em que a Internet ainda dava seus primeiros passos.

2.2. Visão Geral do Hypertexto

HTML é uma abreviação de Hypertext Markup Language - Linguagem de Marcação de Hypertexto. Resumindo em uma frase: o HTML é uma linguagem para publicação de conteúdo (texto, imagem, vídeo, áudio etc.) na Web.

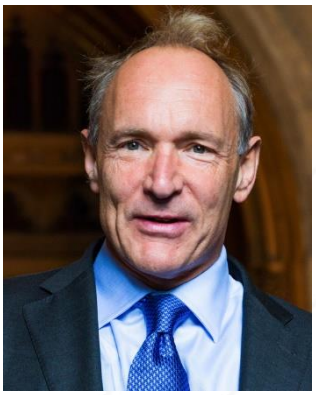
O HTML é baseado no conceito de Hipertexto. Hipertexto são conjuntos de elementos – ou nós – ligados por conexões. Estes elementos podem ser palavras, imagens, vídeos, áudio, documentos etc. Estes elementos conectados formam uma grande rede de informação. Eles não estão conectados linearmente como se fossem textos de um livro, onde um assunto é ligado ao outro seguidamente. A conexão feita em um hipertexto é algo imprevisto que permite a comunicação de dados, organizando conhecimentos e guardando informações relacionadas.

Para distribuir informação de uma maneira global, é necessário haver uma linguagem que seja entendida universalmente por diversos meios de acesso. O HTML se propõe a ser esta linguagem.

Desenvolvido originalmente por Tim Berners-Lee o HTML ganhou popularidade quando o Mosaic - browser desenvolvido por Marc Andreessen na década de 1990 -

ganhou força. A partir daí, desenvolvedores e fabricantes de browsers utilizaram o HTML como base, compartilhando as mesmas convenções.

Saiba mais



Por fim, além de criticar sua criação, Tim Berners-Lee afirmou que já chegou a hora dos dispositivos móveis, como celulares e tablets, começarem a operar em função dos seus donos, em vez de servirem aos seus fabricantes ou aos desenvolvedores de seus sistemas operacionais.

Tim Berners-Lee (Pai da Internet)

https://www.hardware.com.br/noticias/2022-01/tim-berners-lee-criador-do-www-nao-esta-feliz-com-o-que-a-web-se-tornou.html

2.2.1. Interoperabilidade – novo desafio

Entre 1993 e 1995, o HTML ganhou as versões HTML+, HTML2.0 e HTML3.0, onde foram propostas diversas mudanças para enriquecer as possibilidades da linguagem. Contudo, até aqui o HTML ainda não era tratado como um padrão. Apenas em 1997, o grupo de trabalho do W3C (Word Wide Web Comitee) responsável por manter o padrão do código, trabalhou na versão 3.2 da linguagem, fazendo com que ela fosse tratada como prática comum. Você pode ver: http://www.w3.org/TR/html401/appendix/changes.html.

Desde o começo o HTML foi criado para ser uma linguagem independente de plataformas, browsers e outros meios de acesso. Interoperabilidade significa menos custo de desenvolvimento, licenciamento e uso. Em sua concepção um arquivo HTML criado terá apenas um código e este código pode ser lido em vários meios, ao invés de versões diferentes para cada dispositivos. Dessa forma, evitou-se que a Web fosse desenvolvida em uma base proprietária, com formatos incompatíveis ou proprietários.

Por isso o HTML foi desenvolvido para que essa barreira fosse ultrapassada, fazendo com que a informação publicada por meio deste código fosse acessível por dispositivos e outros meios com características diferentes, não importando o tamanho da tela, resolução, variação de cor. Dispositivos próprios para deficientes visuais e auditivos

ou dispositivos móveis e portáteis. O HTML deve ser entendido universalmente, dando a possibilidade para a reutilização dessa informação de acordo com as limitações de cada meio de acesso. Essa premissa torna o HTML uma linguagem universal, padronizada e independente de plataforma.

2.2.2. WHAT Working Group

O grupo de desenvolvimento Web Hypertext Application Technology Working Group ou WHATWG trabalhava em uma versão do HTML que trazia mais flexibilidade para a produção de websites e sistemas baseados na web. O trabalho do WHATWG (http://www.whatwg.org/) foi estabelecido por desenvolvedores de empresas como Mozilla, Apple e Opera, esse esforço tinha por objetivo propor uma alternativa ao desenvolvimento XHTML (futura versão do HTML). O trabalho da WHATWG é a origem do HTML5, atualmente utilizado. Caso você deseje contribuir com o desenvolvimento das próximas verões, no site acima existe um canal de comunicação.

Em 2006, Tim Berners-Lee anunciou que trabalharia juntamente com o WHATWG na produção do HTML5 em detrimento do XHTML 2, sendo descontinuado em 2009.

2.2.3. O HTML5 e suas mudanças

Quando o HTML4 foi lançado, o W3C alertou os desenvolvedores sobre algumas boas práticas que deveriam ser seguidas ao produzir códigos *client-side*. As recomendações eram: separação da estrutura do código com a formatação e princípios de acessibilidade. Embora o HTML4, ainda não apresentasse diferencial para a semântica do código, o HTML4 também não facilitava a manipulação dos elementos via Javascript ou CSS.

Advento do HTML5?

O HTML5 é a nova versão do HTML4. Enquanto o WHATWG define as regras de marcação que usaremos no HTML5 e no XHTML, eles também definem APIs que formarão a base da arquitetura web. Essas APIs são conhecidas como DOM Level 0.

Um dos principais objetivos do HTML5 é facilitar a manipulação do elemento possibilitando ao desenvolvedor modificar as características dos objetos de forma não

intrusiva e de maneira que seja transparente para o usuário final e independente do browser.

A partir do HTML5, as ferramentas de CSS e Javascript estão disponíveis e apresentam os melhores resultados, outra novidade está nas Application Programming Interfaces (API) que permitem a manipulação das características dos elementos, de forma que o website ou a aplicação continue leve e funcional.

Saiba mais

Application Programming Interfaces (API) são construções disponíveis nas linguagens de programação que permitem a desenvolvedores criar funcionalidades complexas.

Tim Berners-Lee (Pai da Internet)

https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Client-side_web_APIs/Introduction

O HTML5 também cria *tags* e modifica a função de outras. As versões antigas do HTML não continham um padrão universal para a criação de seções comuns e específicas como rodapé, cabeçalho, *sidebar*, menus etc. E estavam ausentes um padrão de nomenclatura de IDs, Classes ou *tags*, e método de capturar de maneira automática as informações localizadas nos rodapés.

Uma premissa do HTML5 está na forma de escrever o código e organizamos a informação, essa alteração semântica, permite maior interatividade, dispensa plugins e evita a redução da performance, dessa forma o código é interoperável, pronto para futuros dispositivos. Porém para respeitar o legado e os sites já desenvolvidos é mantida uma retro compatibilidade com as versões antigas do HTML.

2.2.4. O desenvolvimento modular

A concepção e desenvolvimento das versões anteriores do HTML / CSS forma liberadas integralmente após testes e revisões gerais, entretanto a partir do HTML 5 foram estabelecidos grupos de trabalho para temas específicos, o que permite o lançamento de

novidades de forma independente de cada grupo. Eliminando a necessidade de completo desenvolvimento e divulgação de novidades como um único pacote/produto.

As propriedades do CSS3, foram divididas em pequenos grupos. Há um grupo cuidando da propriedade *Background*, outro da propriedade *Border*, outro das propriedades de Texto etc. Cada um destes grupos são independentes e podem lançar suas novidades a qualquer momento. Logo, o desenvolvimento para web ficou mais dinâmico, com novidades mais constantes, que vão ampliar as funcionalidades após a absorção pelos desenvolvedores dos browsers.

2.2.5. Motores de Renderização

Há uma grande diversidade de dispositivos que acessam a internet. Entre eles, há uma série de tablets, smartphones, computadores etc. Cada um destes acessa a partir de um determinado browser, entretanto são os motores de renderização contidos nos browsers que fazem a leitura e interpretação dos códigos. Abaixo, segue uma lista dos principais browsers e seus motores:

MOTOR	BROWSER
WEBKIT	Safari, Google Chrome
GECKO	Firefox, Mozilla, Camino
TRIDENT	Internet Explorer
PRESTO	Opera 7 ao 10
CHROMIUM	EDGE Chrome e GOOGLE Chrome

É interessante que você faça código compatível com estes motores. Focando a compatibilidade nos motores de renderização você atingirá uma amplitude maior de browsers.

2.2.6. Compatibilidade com HTML5

Atualmente o Webkit é o motor mais compatível com os Padrões do HTML5. Como a Apple tem interesse que seus dispositivos sejam ultra compatíveis com os Padrões, ela tem feito um belo trabalho de atualização e avanço da compatibilidade deste motor.

Saiba mais

WebKit é o motor do Safari, Mail, App Store e outros apps da Apple.

https://webkit.org/

Contudo o Firefox e o Opera já estão compatíveis com grande parte da especificação do HTML5 e CSS3 e a cada upgrade eles trazem mais novidades e atualização dos padrões.

A Microsoft abandonou seu motor de renderização e adotou o padrão do consórcio Chromium, torando a mais nova versão do EDGE dependente desse motor. O Chromium é um projeto de navegador web de código aberto desenvolvido pela Google, no qual o Google Chrome baseia o seu código-fonte.

Saiba mais

O Chromium é um web browser de código aberto, com objetivo de ser mais leve estável e seguro, para todos na internet.

https://www.chromium.org/Home/

2.2.7. Técnicas de detecção

Pode ser que o usuário não utilize um browser que suporta HTML5. Neste caso, você pode redirecioná-lo para uma versão do site mais simples, ou talvez apenas mostrar uma mensagem alertando o usuário sobre a importância da atualização do browser.

2.3. Estrutura básica, doctype e charsets

A estrutura básica do HTML5 continua sendo a mesma das versões anteriores da linguagem, há apenas uma exceção na escrita do *Doctype*. Segue abaixo como a estrutura básica pode ser seguida:

```
1  <!DOCTYPE HTML>
2  <html lang="pt-br">
3  <head>
4  <meta charset="UTF-8">
5  <link rel="stylesheet" type="text/css" href="estilo.css">
6  <title></title>
7  </head>
8  <body>
9
10 </body>
11 </html>
```

2.3.1. O Doctype

O *Doctype* deve ser a primeira linha de código do documento antes da *tag* HTML.

```
<!DOCTYPE html!>
```

O *Doctype* não é uma *tag* do HTML, mas uma instrução para que o browser tenha informações sobre qual versão de código a marcação foi escrita.

2.3.2. O elemento HTML

O código HTML é uma árvore de elementos onde alguns são filhos de outros e assim por diante. O elemento principal dessa grande árvore é sempre a *tag* HTML.

```
<html lang="pt-br">
```

O atributo LANG é necessário para que os *user-agents* saibam qual a linguagem principal do documento e pode ser pode ser utilizado em qualquer outro elemento para indicar o idioma do texto.

Saiba mais

Para encontrar a listagem de códigos das linguagens, acesse:

http://www.w3.org/International/questions/qa-choosing-language-tags.

2.3.3. HEAD

A *Tag* HEAD é onde fica toda a parte inteligente da página. No HEAD ficam os metadados, que contém informações sobre a página e o conteúdo ali publicado.

Metatag Charset

No nosso exemplo há uma *metatag* responsável por chavear qual tabela de caracteres a página está utilizando.

```
<meta charset="utf-8">
```

A Web foi imaginada para ser ampla e irrestrita por isso foi definida a tabela de Unicode que contém mais de um milhão de caracteres, uma tabela padrão e que permitirá a visualização em qualquer idioma. O que o Unicode faz é fornecer um único número para cada caractere, não importa a plataforma, nem o programa, nem a língua.

2.4. Modelos de conteúdo

Considere as regras básicas do HTML desde o início, estas regras definem onde os elementos podem ou não estar, são filhos ou pais de outros elementos e quais os seus comportamentos. Destacamos os elementos de linha e bloco.

Os elementos de linha marcam, na sua maioria das vezes, texto. Alguns exemplos:

```
a, strong, em, img, input, abbr, span.
```

Os elementos de blocos são como caixas, que dividem o conteúdo nas seções do layout.

Abaixo segue algumas premissas que você precisa saber:

- Os elementos de linha podem conter outros elementos de linha, dependendo da categoria.
- Os elementos de linha nunca podem conter elementos de bloco.
- Elementos de bloco sempre podem conter elementos de linha.
- Elementos de bloco podem conter elementos de bloco, dependendo da categoria que ele se encontra. Por exemplo, um parágrafo não pode conter um DIV. Mas o contrário é possível.

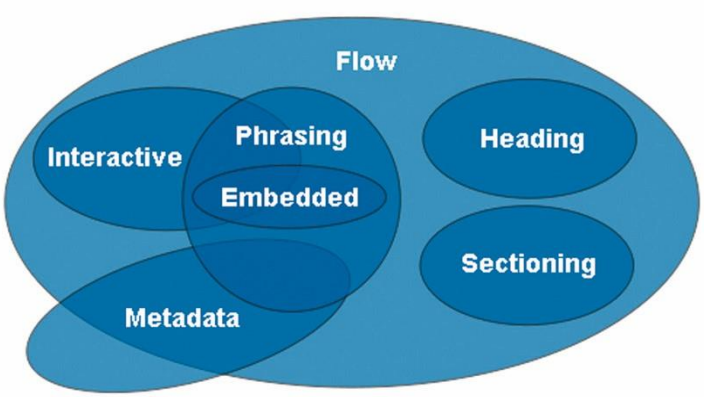
Estes dois grandes grupos podem ser divididos em categorias. Estas categorias dizem qual modelo de conteúdo o elemento trabalha e como pode ser seu comportamento.

2.4.1. Categorias

Cada elemento no HTML pode ou não fazer parte de um grupo de elementos com características similares. As categorias estão a seguir. Manteremos os nomes das categorias em inglês para que haja um melhor entendimento:

a) Metadata content
b) Flow content
c) Sectioning content
d) Heading content
e) Phrasing content
f) Embedded content
g) Interactive contente

Abaixo segue como as categorias estão relacionadas de acordo com o WHATWG:



Fonte: whatwg.org

Metadata content

Os elementos que compõe a categoria Metadata são:

a) base
b) command
c) link
d) meta
e) noscript
f) script

g) style
h) title

Este conteúdo vem antes da apresentação, formando uma relação com o documento e seu conteúdo com outros documentos que distribuem informação.

Flow content

A maioria dos elementos utilizados no body e aplicações são categorizados como Flow Content.

São eles:

- a
- abbr
- address
- area (se for um descendente de um elemento de mapa)
- article
- aside
- audio
- b
- bdo
- blockquote
- br
- button
- canvas
- cite
- code
- command
- datalist
- del
- details
- dfn
- div • dl
- em
- embed
- fieldset
- figure
- footer
- form
- h1
- h2
- h3
- h4
- h5

- h6
- header
- hgroup
- hr
- i
- iframe
- img
- input
- ins
- kbd
- keygen
- label
- link (Se o atributo `itemprop` for utilizado)
- map
- mark
- math
- menu
- meta (Se o atributo `itemprop` for utilizado)
- meter
- nav
- noscript
- object
- ol
- output
- pre
- progress
- q
- ruby
- samp
- script
- section
- select
- small
- span
- strong
- style (Se o atributo `scoped` for utilizado)
- sub
- sup
- svg
- table
- textarea
- time
- ul
- var

- video
- wbr
- Text

Geralmente, elementos que seu modelo de conteúdo permite inserir qualquer elemento que se encaixa no *Flow Content*, devem ter pelo menos um descendente de texto ou um elemento descendente que faça parte da categoria `embedded`.

2.4.2. *Sectioning content*

Estes elementos definem um grupo de cabeçalhos e rodapés.

- article
- aside
- nav
- section

Basicamente são elementos que juntam grupos de textos no documento.

2.4.3. *Heading content*

Os elementos da categoria *Heading* definem uma seção de cabeçalhos, que podem estar contidos em um elemento na categoria *Sectioning*.

- h1
- h2
- h3
- h4
- h5
- h6
- hgroup

2.4.4. *Phrasing content*

Fazem parte desta categoria elementos que marcam o texto do documento, bem como os elementos que marcam este texto dentro do elemento de parágrafo.

- a
- abbr
- area (se ele for descendente de um elemento de mapa)
- audio
- b
- bdo
- br
- button
- canvas

- cite
- code
- command
- datalist
- del (se ele contiver um elemento da categoria de Phrasing)
- dfn
- em
- embed
- i
- iframe
- img
- input
- ins (se ele contiver um elemento da categoria de Phrasing)
- kbd
- keygen
- label
- link (se o atributo `itemprop` for utilizado)
- map (se apenas ele contiver um elemento da categoria de Phrasing)
- mark
- math
- meta (se o atributo `itemprop` for utilizado) meter
- noscript
- object
- output
- progress
- q
- ruby
- samp
- script
- select
- small
- span
- strong
- sub
- sup
- svg
- textarea
- time
- var
- video
- wbr
- Text

2.4.5. *Embedded content*

Na categoria *Embedded*, há elementos que importam outra fonte de informação para o documento.

- audio
- canvas
- embed
- iframe
- img
- math
- object
- svg
- video

2.4.6. *Interactive content*

Interactive Content são elementos que fazem parte da interação de usuário.

- a
- audio (se o atributo `control` for utilizado)
- button
- details
- embed
- iframe
- img (se o atributo `usemap` for utilizado)
- input (se o atributo `type` não tiver o valor hidden)
- keygen
- label
- menu (se o atributo `type` tiver o valor `toolbar`)
- object (se o atributo `usemap` for utilizado)
- select
- textarea
- video (se o atributo `control` for utilizado)

Alguns elementos no HTML podem ser ativados por um comportamento. Isso significa que o usuário pode ativá-lo de alguma forma. O início da sequência de eventos depende do mecanismo de ativação e normalmente culminam em um evento: teclado, mouse, comando de voz etc.

3. HTML – Parte 2

Objetivos

- Ampliar o conhecimento sobre o HTML
- Compreender as outras funções
- Realizar os exemplos

Introdução

A seguir novas dimensões do HTML, funções que ampliarão o resultado do seu projeto.

Vamos juntos!

3.1. Novos elementos e atributos

A função do HTML é indicar que tipo de informação a página está exibindo. Quando lemos um livro, conseguimos entender e diferenciar um título de um parágrafo. Basta percebermos a quantidade de letra, tamanho da fonte, cor etc. No código isso é diferente, depende da *tag.* Robôs de busca e outros *user-agents* não conseguem diferenciar tais detalhes. Por isso, cabe ao desenvolvedor marcar a informação para que elas possam ser diferenciadas por diversos dispositivos.

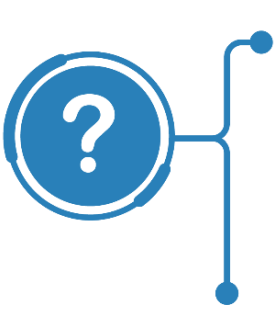
Um atributo do HTML5 são os elementos que nos ajudam a definir setores principais no documento HTML, por exemplo diferenciar diretamente pelo código HTML5 áreas importantes do site como *sidebar*, rodapé e cabeçalho, identificando exatamente é o texto do artigo.

Esta nova estrutura do HTML5 simplificam a operação dos buscadores ao vasculhar o código de maneira mais eficaz, coletando dados exatos de cada página.

Abaixo segue uma lista dos novos elementos e atributos incluídos no HTML5:

- **section** - A *tag* `section` define uma nova seção genérica no documento.
- **nav** O elemento `nav` representa uma seção da página que contém links para outras partes do website.

- **article** O elemento `article` representa uma parte da página que poderá ser distribuído e reutilizável.
- **aside** - O elemento `aside` representa um bloco de conteúdo que referência o conteúdo que envolta do elemento `aside`.
- **hgroup** - Este elemento consiste em um grupo de títulos.
- **header** - O elemento `header` representa um grupo de introdução ou elementos de navegação.
- **footer** - O elemento `footer` representa literalmente o rodapé da página.
- **Time** - Este elemento serve para marcar parte do texto que exibe um horário ou uma data precisa no calendário gregoriano.



Saiba mais

O W3C mantém um documento sobre as diferenças entre HTML 5 E HTML4, atualizado constantemente nesta página:

https://www.w3.org/TR/html5-diff/

3.2. Elementos modificados e ausentes

Existiam no HTML alguns elementos que traziam apenas características visuais e não semânticas para o conteúdo da página, esses elementos foram descontinuados porque sua função será realizada pelo CSS.

3.3. Novos tipos de campos

Novos valores para o atributo type

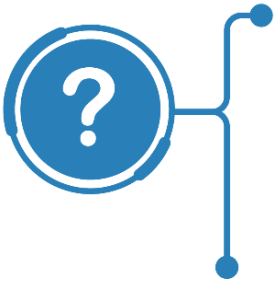
O elemento input aceita os seguintes novos valores para o atributo type:

tel

Telefone. Não há máscara de formatação ou validação, propositalmente, visto não haver no mundo um padrão bem definido para números de telefones.

search

Um campo de busca. A aparência e comportamento do campo pode mudar ligeiramente dependendo do agente de usuário, para parecer com os demais campos de busca do sistema.



Saiba mais

Um agente de usuário é uma “string” - isto é, uma linha de texto - identificando o navegador e sistema operacional para o servidor web. Isso parece simples, mas os agentes do usuário se tornaram uma bagunça ao longo do tempo.

https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Headers/User-Agent

email

E-mail, com formatação e validação. O agente de usuário pode inclusive promover a integração com sua agenda de contatos.

url

Um endereço web, também com formatação e validação.

Datas e horas

O campo de formulário pode conter qualquer um desses valores no atributo *type*:

- datetime
- date
- month
- week
- time
- datetime-local

Todos devem ser validados e formatados pelo agente de usuário, que pode inclusive mostrar um calendário, um seletor de horário ou outro auxílio ao preenchimento que estiver disponível no sistema do usuário.

number

Veja um exemplo do tipo *number* com seus atributos opcionais:

```
<!DOCTYPE html>
<html lang="en-US">
<head>
<meta charset="UTF-8" />
<title>Number type</title>
</head>

<body>

<input name="valuex" type="number"
value="12.4" step="0.2"
min="0" max="20" />

</body>

</html>

```

range

Vamos modificar, no exemplo acima, apenas o valor de *type*, mudando de "number" para "*range*":

```
<!DOCTYPE html>
<html lang="en-US">
<head>
<meta charset="UTF-8" />
<title>Range type</title>
</head>

<body>

<input name="valuex" type="range"
value="12.4" step="0.2"
min="0" max="20" />

</body>

</html>

```

color

O campo com `type="color"` é um seletor de cor. O agente de usuário pode mostrar um controle de seleção de cor ou outro auxílio que estiver disponível. O valor será uma cor no formato

`#ff6600`.

3.4. TIPOS DE DADOS E VALIDADORES

3.4.1. Formulários vitaminados

Conforme você deve ter percebido o HTML5 avançou bastante nos recursos de formulários, facilitando muito a vida de quem precisa desenvolver aplicações web baseadas em formulários.

autofocus

Ao incluir em um campo de formulário o atributo *autofocus*, assim:

```
<input name="login" autofocus >
```

O foco será colocado neste campo automaticamente ao carregar a página.

Placeholder text

Você já deve ter visto um "*placeholder*". Tradicionalmente, vínhamos fazendo isso:

```
1   <!DOCTYPE html>
2   <html lang="en-US">
3   <head>
4   <meta charset="UTF-8" />
5   <title>Placeholder, the old style</title>
6   </head>
7
8   <body>
9   <input name="q" value="Search here"
10  onfocus="if(this.value=='Search here')this.value=''">
11  </body>
12
13  </html>
```

HTML5 nos permite fazer isso de maneira muito mais elegante:

```
1   <!DOCTYPE html>
2   <html lang="en-US">
3   <head>
4   <meta charset="UTF-8" />
5   <title>Placeholder, HTML5 way</title>
6   </head>
7
8   <body>
9   <input name="q" placeholder="Search here">
10  </body>
11
12  </html>
```

required

Para tornar um campo de formulário obrigatório (seu valor precisa ser preenchido) basta, em HTML5, incluir o atributo `required`:

```
<input name="login" required>
```

maxlength

Você já conhecia o atributo *maxlength*, que limita a quantidade de caracteres em um campo de formulário e também no elemento *textarea*.

3.4.2. Validação de formulários

No HTML5 você pode tornar seus campos pré-definidos, estabelecendo valores para o atributo *type*, que já incluem validação para datas, emails, URLs e números.

pattern

O atributo `pattern` nos permite definir expressões regulares de validação, sem Javascript. Veja um exemplo de como validar CEP:

```
1 <!DOCTYPE html>
2 <html lang="pt-BR">
3 <head>
4 <meta charset="UTF-8" />
5 <title>O atributo pattern</title>
6 </head>
7
8 <body>
9
10 <form>
11 <label for="CEP">CEP:
12 <input name="CEP" id="CEP" required pattern="\d{5}-?\d{3}" />
13 </label>
14 <input type="submit" value="Enviar" />
15 </form>
16
17 </body>
18
19 </html>
20
```

novalidate e formnovalidate

Existem situações em que você precisa que um formulário não seja validado. Nestes casos, basta incluir no elemento `form` o atributo `novalidate`.

Outra situação comum é querer que o formulário não seja validado dependendo da ação de *submit*.

Nesse caso, você pode usar no botão de *submit* o atributo `formnovalidate`. Veja um exemplo:

```
<!DOCTYPE html>
  <html lang="pt-BR">
  <head>
  <meta charset="UTF-8" />
  <title>Salvando rascunho</title>
  <style>
  label{display:block;}
  </style>
  </head>

<body>

<form>
<label>nome: <input name="nome" required></label>
<label>email: <input name="email" type="email" required></label>
<label>mensagem: <textarea name="mensagem" required></textarea></label>
<input type="submit" name="action" value="Salvar rascunho" formnovalidate>
<input type="submit" name="action" value="Enviar">
</form>

</body>

</html>

```

Custom validators

É claro que as validações padrão, embora atendam a maioria dos casos, não são suficientes para todas as situações. Muitas vezes você vai querer escrever sua própria função de validação Javascript.

Há alguns detalhes na especificação do HTML5 que vão ajudá-lo com isso:

1. **O novo evento oninput** é disparado quando algo é modificado no valor de um campo de formulário. Diferente de *onchange*, que é disparado ao final da edição, *oninput* é disparado ao editar. É diferente também de *onkeyup* e *onkeypress*, porque vai capturar qualquer modificação no valor do campo, feita com mouse, teclado ou outra interface qualquer.
2. O método **setCustomValidity** pode ser invocado por você. Ele recebe uma *string*. Se a *string* for vazia, o campo será marcado como válido. Caso contrário, será marcado como inválido.

Com isso, você pode inserir suas validações no campo de formulário e deixar o navegador fazer o resto. Não é mais preciso capturar o evento *submit* e tratá-lo. Veja, por exemplo, este formulário com validação de CPF:

```
<!DOCTYPE html>
    <html lang="pt-BR">
    <head>
    <meta charset="UTF-8" />
    <title>Custom validator</title>
    <!-- O arquivo cpf.js contém a função validaCPF, que
    recebe uma string e retorna true ou false. -->
    <script src="cpf.js"></script>
    <script>
    function vCPF(i){
    i.setCustomValidity(validaCPF(i.value)?'':'CPF inválido!')
    }
    </script>
    </head>

<body>
<form>
<label>CPF: <input name="cpf" oninput="vCPF(this)" /></label>
<input type="submit" value="Enviar" />
</form>
</body>

</html>

```

3.5. Detalhes e conteúdo editável.

Ainda mais formulários

Vejamos mais duas coisas que foram simplificadas pelo HTML5.

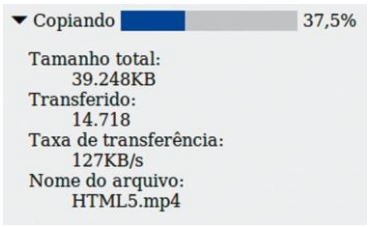
Detalhes e sumário

Veja um exemplo de uso dos novos elementos *details* e *summary*:

```
      <details>
  <summary>Copiando <progress max="39248" value="14718"> 37,5%</
summary>   <dl>
          <dt>Tamanho total:</dt>
          <dd>39.248KB</dd>
          <dt>Transferido:</dt>
          <dd>14.718</dd>
          <dt>Taxa de transferência:</dt>
          <dd>127KB/s</dd>
          <dt>Nome do arquivo:</dt>
          <dd>HTML5.mp4</dd>
        </dl>
      </details>
```

Veja como um agente de usuário poderia renderizar isso:

E ao clicar:

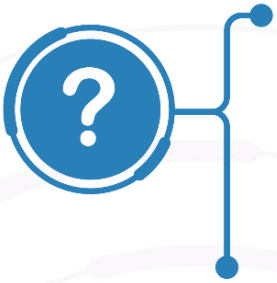


Conteúdo editável

Para tornar um elemento do HTML editável, basta incluir nele o atributo *contenteditable*, assim:

```
      <div contenteditable=”true”>
Edite-me...
      </div>
```

Você pode ler e manipular os elementos editáveis normalmente usando os métodos do DOM. Isso permite, com facilidade, construir uma área de edição de HTML.



Saiba mais

O Modelo de Objeto de Documento (DOM) é uma interface de programação para documentos HTML, XML e SVG . Ele fornece uma representação estruturada do documento como uma árvore. O DOM define métodos que permitem acesso à árvore, para que eles possam alterar a estrutura, estilo e conteúdo do documento.

https://developer.mozilla.org/pt-BR/docs/Web/API/Document_Object_Model

3.6. DRAG-N-DROP E CORREÇÃO ORTOGRÁFICA

3.6.1. Drag and Drop

A API de Drag and Drop é relativamente simples. Basicamente, inserir o atributo `draggable="true"` num elemento o torna arrastável. E há uma série de eventos que você pode tratar. Os eventos do objeto sendo arrastado são:

dragstart

O objeto começou a ser arrastado. O evento que a função recebe tem um atributo target, que contém o objeto sendo arrastado.

drag

O objeto está sendo arrastado

dragend

A ação de arrastar terminou

O objeto sobre o qual outro é arrastado sofre os seguintes eventos:

dragenter

O objeto sendo arrastado entrou no objeto target

dragleave

O objeto sendo arrastado deixou o objeto target

dragover

O objeto sendo arrastado se move sobre o objeto target

drop

O objeto sendo arrastado foi solto sobre o objeto target

Detalhes importantes:

A ação padrão do evento *dragover* é cancelar a ação de d*ragging* atual. Assim, nos objetos que devem receber *drop*, é preciso *setar* uma ação de *dragover* com, no mínimo, `return false`.

Seleções de texto são automaticamente arrastáveis, não precisam do atributo `draggable`. E se você quiser criar uma área para onde seleções de texto possam ser arrastadas, basta tratar esses mesmos eventos.

Por fim, todas funções de tratamento de evento de drag recebem um objeto de evento que contém uma propriedade `dataTransfer`, um *dataset* comum a todos os eventos durante essa operação de drag.

```
<!DOCTYPE HTML>
<html>
<head>
<meta content="text/html; charset=UTF-8" http-equiv="content-type"/>
<title>HTML5 Drag and drop demonstration</title>
<style type="text/css">
#boxA, #boxB {
float:left; width:100px; height:200px; padding:10px; margin:10px; font-size:70%;
}
#boxA { background-color: blue; }
#boxB { background-color: green; }

#drag, #drag2 {
width:50px; padding:5px; margin:5px; border:3px black solid; line-height:50px;
}
#drag { background-color: red;}
#drag2 { background-color: orange;}
</style>
<script type="text/javascript">

// Quando o usuário inicia um drag, guardamos no dataset do evento
    // o id do objeto sendo arrastado
    function dragStart(ev) {
    ev.dataTransfer.setData("ID", ev.target.getAttribute('id'));
    }

// Quando o usuário arrasta sobre um dos painéis, retornamos
// false para que o evento não se propague para o navegador, o
// que faria com que o conteúdo fosse selecionado.
function dragOver(ev) { return false; }


// Quando soltamos o elemento sobre um painel, movemos o
// elemento, lendo seu id do dataset do evento
    function dragDrop(ev) {
    var idelt = ev.dataTransfer.getData("ID");
    ev.target.appendChild(document.getElementById(idelt));
    }

</script>
</head>
<body>
<!-- Painel 1 -->
<div id="boxA"
ondrop="return dragDrop(event)"
ondragover="return dragOver(event)">
<!-- Draggable 1 -->
<div id="drag" draggable="true"
ondragstart="return dragStart(event)">drag me</div>
```

```
49 <!-- Draggable 2 -->
50 <div id="drag2" draggable="true"
51  ondragstart="return dragStart(event)">drag me</div>
52 </div>
53
54 <!-- Painel 2 -->
55 <div id="boxB"
56 ondrop="return dragDrop(event)"
57 ondragover="return dragOver(event)">
58 </div>
59
60 </body>
61 </html>
```

3.6.2. Revisão ortográfica e gramatical

Os agentes de usuário podem oferecer recursos de revisão ortográfica e gramatical, dependendo do que houver disponível em cada plataforma. Utilize o atributo `spellcheck`. Inserir `spellcheck="true"` num elemento faz com que a revisão esteja habilitada para ele, ou `spellcheck="false"`, desabilita.

4. HTML – Parte 3

Objetivos

- Apresentar algumas funções avançadas do HTML
- Realizar os exemplos

Introdução

A seguir novas dimensões do HTML, funções que permitirão novas interações do usuário com seu projeto.

Vamos juntos!

4.1. Elementos audio e video, e codecs

Áudio

Para inserir áudio em uma página web, basta usar o elemento `audio`:

```
<audio src=”mus.oga” controls=”true” autoplay=”true” />
```

O valor de *controls* define se um controle de áudio, com botões de play, pause, volume, barra de progresso, contador de tempo etc. será exibido na tela. O valor de *autoplay* define se o áudio vai começar a tocar assim que a página carregar.

Origens alternativas de áudio

Todo agente de usuário deveria suportar o codec livre OggVorbis, mas, infelizmente, pode acontecer de seu arquivo oga não tocar no computador ou celular de alguém. Quem sabe do seu chefe ou seu cliente. Então é preciso saber como oferecer um formato alternativo de áudio. Segue uma sugestão:

```
<audio controls=”true” autoplay=”true”>
  <source src=”mus.oga” />
  <source src=”mus.mp3” />
  <source src=”mus.wma” />
</audio>
```

Claro, o agente de usuário pode ainda não saber tocar nenhum desses formatos, ou sequer ter suporte a áudio. Para esses casos, ofereça um conteúdo alternativo:

```
<audio controls=”true” autoplay=”true”>
  <source src=”mus.oga” />
```

```
    <source src="mus.mp3" />
    <source src="mus.wma" />
    <p>Faça o <a href="mus.mp3">download da música</a>.</p>
</audio>
```

Vídeo

O uso de vídeo é muito semelhante ao de áudio:

```
<video src="u.ogv" width="400" height="300" />
```

E com vários elementos source:

```
<video controls="true" autoplay="true" width="400"
height="300">
    <source src="u.ogv" />
    <source src="u.mp4" />
    <source src="u.wmv" />
    <p>Faça o <a href="u.mp4">download do vídeo</a>.</p>
</video>
```

Codecs

É muito importante que você inclua, nos seus elementos *source* de áudio e vídeo, informação a respeito do container e *codecs* utilizados. Isso evita que o navegador faça download, e descubra que não consegue tocá-lo. É importante lembrar que a extensão do arquivo não é relevante.

Para indicar ao navegador o container e *codecs* de determinado arquivo, usa-se o atributo `type`, no formato:

```
type='MIME-type do container; codecs="codec de vídeo, codec de áudio"'
```

Por exemplo, um vídeo em Ogg, usando os codecs Theora e Vorbis, terá seu source assim:

```
<source src='video.ogv' type='video/ogg; codecs="theora, vorbis"'>
```

Com MPEG-4 a coisa é um pouco mais complicada, porque é preciso indicar ao navegador também o profile do codec de vídeo utilizado. Veja um exemplo:

```
<source src='video.mp4' type='video/mp4; codecs="mp4v.20.240, mp4a.40.2"'>
```

4.2. MATHML E SVG

4.2.1. MathML

O HTML5 incorpora o padrão MathML. Trata-se de uma linguagem de marcação, baseada em XML, para representação de fórmulas matemáticas. Você pode ler mais sobre MathML em https://html.spec.whatwg.org/multipage/embedded-content-other.html#mathml Para incorporar código MathML em seu documento HTML5, não preciso fazer declarações especiais. Basta escrever normalmente o código, iniciando com um elemento math. Veja este exemplo:

```
1<!DOCTYPE html>
 2 <html>
 3 <head>
 4 <meta charset=”UTF-8” />
 5 <title>MathML</title>
 6 </head>
 7 <body>
       8
 9  <math>
 10 <mrow>
 11 <mi>x</mi>
 12 <mo>=</mo>
 13 <mfrac>
 14 <mrow>
 15 <mo form=”prefix”>&minus;</mo>
 16 <mi>b</mi>
 17 <mo>&PlusMinus;</mo>
 18 <msqrt>
 19 <msup>
 20 <mi>b</mi>
 21 <mn>2</mn>
 22 </msup>
 23 <mo>&minus;</mo>
 24 <mn>4</mn>
 25 <mo>&InvisibleTimes;</mo>
 26 <mi>a</mi>
 27 <mo>&InvisibleTimes;</mo>
 28 <mi>c</mi>
 29 </msqrt>
 30 </mrow>
 31 <mrow>
 32 <mn>2</mn>
 33 <mo>&InvisibleTimes;</mo>
 34 <mi>a</mi>
 35 </mrow>
 36 </mfrac>
 37 </mrow>
 38 </math>
       39
 40 </body>
 41 </html>
```

Veja como esse exemplo é renderizado no navegador:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Mesmo que você nunca tenha visto MathML, e este código pareça um pouco assustador, dê uma olhada com calma no código, comparando com a imagem do resultado, e você vai perceber sua simplicidade. Talvez algo que possa deixá-lo confuso é a entidade `&InvisibleTimes;`, que aparece algumas vezes no código. Ela está lá para separar os fatores 4ac, por exemplo. Esses valores são multiplicados, é o que a fórmula representa, mas não queremos colocar um operador de multiplicação entre eles, porque por convenção se simplesmente escrevemos 4ac qualquer leitor saberá que isso é uma multiplicação.

Por que então se preocupar em inserir `&InvisibleTimes;`? Você vai notar que se remover a entidade e a *tag* mo correspondente o resultado visual será o mesmo. Colocamos `&InvisibleTimes;` porque MathML não é só visual, é semântica. Um outro agente de usuário pode ter recursos de importar essa fórmula para uma ferramenta de cálculo, por exemplo.

4.2.2. SVG

O SVG é uma outra linguagem XML que pode ser incorporada com facilidade em HTML5, está é uma linguagem de marcação para gráficos vetoriais. Consulte: https://html.spec.whatwg.org/multipage/embedded-content-other.html#svg-0

Vejamos um exemplo bem simples:

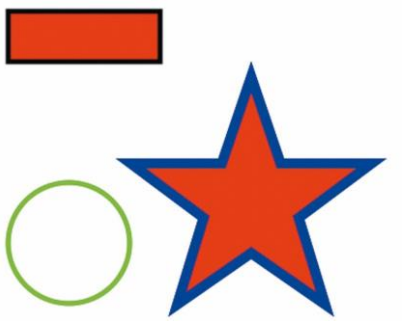
```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8" />
5 <title>SVG</title>
6 </head>
7 <body>
8
9 <svg width="400" height="400">
10
11 <!-- Um retângulo: -->
12 <rect x="10" y="10" width="150" height="50" stroke="#000000" stroke-
   width="5" fill="#FF0000" />
13
```

```
<!-- Um polígono: -->
<polygon fill="red" stroke="blue" stroke-width="10"
points="250,75 279,161 369,161 297,215
323,301 250,250 177,301 203,215
131,161 221,161" /> 19
<!-- Um círculo -->
<circle cx="70" cy="240" r="60" stroke="#00FF00" stroke-width="5"
fill="#FFFFFF" />

</svg>

</body>
</html>
```

E veja como isso é renderizado no navegador:



É possível fazer muito mais com SVG. A maioria dos editores de gráficos vetoriais hoje exporta e importa automaticamente SVG, permitindo a um designer construir um gráfico em seu editor vetorial predileto e exportá-lo diretamente.

4.3. Menus e toolbars

O elemento menu

O elemento `menu` é usado para definir menus e barras de ferramenta. Dentro do menu, você pode inserir submenus ou comandos. Para inserir submenus, basta inserir outros elementos `menu`. Para definir comandos, você pode inserir:

1. Um link, um elemento a com atributo `href`;
2. Um botão, um elemento `button`;
3. Um botão, um elemento `input` com o atributo `type` contendo button, submit, reset ou image;
4. Um *radiobutton,* um elemento `input` com o atributo `type` contendo radio;
5. Um *checkbox,* um elemento `input` com o atributo `type` contendo *checkbox*;
6. Um elemento `select`, contendo um ou mais *options*, define um grupo de comandos
7. Um elemento qualquer com o atributo *`accesskey`*
8. Um elemento *`command`*

Tipos de comando

Há três tipos de comando:

command

Uma ação comum;

checkbox

Uma ação que pode estar no status de ligada ou desligada, e alterna entre esses dois stauts quando clicada;

radio

Uma ação que pode estar no status de ligada ou desligada, e quando clicada vai para o status de ligada, deligando todas as ações com o mesmo valor no atributo `radiogroup`;

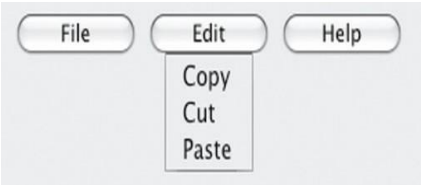
Da lista de elementos possíveis para definir comandos, os três primeiros: *link*, *button* e input *button*, definem comandos do tipo *command*. O quarto elemento, *radiobutton*, define um comando do tipo *radio*. O quinto, *checkbox*, define um comando do tipo *checkbox*. O sexto elemento, o *select*, vai definir um grupo de comandos. Se o *select* tiver o atributo *multiple*, definirá uma lista de comandos do tipo *checkbox*. Caso contrário, os comandos serão do tipo radio, tendo o mesmo *radiogroup*. No sétimo caso, um elemento qualquer com tecla de acesso, o tipo de comando vai depender do tipo de elemento que recebeu *accesskey*. Por fim, temos o oitavo método, o elemento `command`. Neste caso o tipo de comando dependerá do valor do atributo `type`. Veja um exemplo de como usá-lo:

```
<command type=”command” label=”Salvar” onclick=”salvar()” >
```

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset=”UTF-8” />
5 <title>Menus</title>
6 </head>
7
8 <body>
9
10 <menu type=”toolbar”>
11 <li>
12 <menu label=”File”>
13 <button type=”button” onclick=”fnew()”>New...</button>
```

```
14 <button type="button" onclick="fopen()">Open...</button>
15 <button type="button" onclick="fsave()">Save</button>
16 <button type="button" onclick="fsaveas()">Save as...</button>
17 </menu>
18 </li>
19 <li>
20 <menu label="Edit">
21 <button type="button" onclick="ecopy()">Copy</button>
22 <button type="button" onclick="ecut()">Cut</button>
23 <button type="button" onclick="epaste()">Paste</button>
24 </menu>
25 </li>
26 <li>
27 <menu label="Help">
28 <li><a href="help.html">Help</a></li>
29 <li><a href="about.html">About</a></li>
30 </menu>
31 </li>
32 </menu>
33
34 </body>
35
36 </html>
37
```

O agente de usuário deveria renderizar algo como:



4.4. Tipos de links

Links

A habilidade de ligar documentos é o que torna a Web o que ela é. Existem duas maneiras principais de linkar documentos, os elementos `a` e `link`. O elemento **a** cria um *link* no conteúdo da página. Você conhece sua sintaxe:

```
<a href="http://visie.com.br">Visie</a>
```

O elemento `link`, por sua vez, cria um metadado, que não é mostrado no conteúdo, mas o agente de usuário usa de outras maneiras. O uso mais comum é vincular um documento a uma folha de estilos:

```
<link rel="stylesheet" href="estilo.css" />
```

Note o atributo `rel="stylesheet"`. O atributo `rel` pode estar presente nos elementos `a` e `link`, e ter uma série de valores:

Metadados de navegação

- **archives** - os arquivos do site
- **author** - a página do autor do documento ual
- **bookmark** - o *permalink* da seção a que este documento pertence
- **first** - o primeiro documento da série a qual este pertence
- **help** - ajuda para esta página
- **index** - o índice ou sumário que inclui o link para esta página
- l**ast** - o último documento da série a qual este pertence
- **license** - a licença que cobre este documento
- **next** - o próximo documento da série a qual este pertence
- **prefetch** - o agente de usuário deve fazer cache desse link em segundo plano tão logo o documento atual tenha sido carregado. O autor do documento indica que este link é o provável próximo destino do usuário.
- **prev** - o documento anterior da série a qual este pertence
- **search** - a busca deste site
- **up** - O documento um nível acima deste

Metadados da página

- **alternate** - um formato alternativo para o conteúdo atual. Precisa estar acompanhado do atributo type, contendo o tipo MIME do formato. Por exemplo, para indicar o RSS da página atual usamos:

```
<link rel="alternate" type="application/rss+xml" href="rss.xml"
/>
```

- **icon** - o ícone que representa esta página
- **pingback** - a URL de pingback desta página. Através desse endereço um sistema de blogging ou gerenciador de conteúdo pode avisar automaticamente quando um link para esta página for inserido. **stylesheet** a folha de estilo linkada deve ser vinculada a este documento para exibição.

Comportamento dos links na página

- **external -** indica um link externo ao domínio do documento atual
- **nofollow** - indica que o autor do documento atual não endossa o conteúdo desse link. Os robôs de indexação para motores de busca podem, por exemplo, não seguir este link ou levar em conta o *nofollow* em seu algoritmo de ranking.
- **noreferrer** - o agente de usuário não deve enviar o header HTTP *Referer* se o usuário acessar esse link
- **sidebar** - o link deve ser aberto numa *sidebar* do navegador, se este recurso estiver disponível.

4.5. MICRODATA

Semântica adicional

Dê uma olhada no seguinte código:

```
1<!DOCTYPE html>
2<html>
3<head>
4<meta charset="UTF-8" />
5<title>Microdata 1</title>
6</head>
7<body>
8
 9  <h1>Resultados do trimestre</h1>
 10 <ol>
 11 <li>
 12 <dl>
 13 <dt>nome</dt> <dd>Joaquim</dd>
 14 <dt>total</dt> <dd>10.764</dd>
 15 </dl>
 16 </li>
 17 <li>
 18 <dl>
 19 <dt>nome</dt> <dd>Manoel</dd>
 20 <dt>total</dt> <dd>12.449</dd>
 21 </dl>
 22 </li>
 23 <li>
 24 <dl>
 25 <dt>nome</dt> <dd>Antonio</dd>
 26 <dt>total</dt> <dd>9.202</dd>
 27 </dl>
 28 </li>
 29 <li>
 30 <dl>
 31 <dt>nome</dt> <dd>Pedro</dd>
 32 <dt>total</dt> <dd>17.337</dd>
 33 </dl>
 34 </li>
```

```
35 </ol>
36
37 </body>
38 </html>
```

A Microdata API nos permite tornar esta estrutura semântica um pouco mais específica, definindo o que é o conteúdo de cada elemento. Veja este exemplo:

```
1<!DOCTYPE html>
2<html>
3<head>
4<meta charset="UTF-8" />
5<title>Microdata 2</title>
6</head>
7<body>
8
9  <h1>Resultados do trimestre</h1>
10 <ol>
11 <li>
12 <dl itemscope>
13 <dt>nome</dt> <dd itemprop="nome">Joaquim</dd>
14 <dt>total</dt> <dd itemprop="total">10.764</dd>
15 </dl>
16 </li>
17 <li>
18 <dl itemscope>
19 <dt>nome</dt> <dd itemprop="nome">Manoel</dd>
20 <dt>total</dt> <dd itemprop="total">12.449</dd>
21 </dl>
22 </li>
23 <li>
24 <dl itemscope>
25 <dt>nome</dt> <dd itemprop="nome">Antonio</dd>
26 <dt>total</dt> <dd itemprop="total">9.202</dd>
27 </dl>
28 </li>
29 <li>
30 <dl itemscope>
31 <dt>nome</dt> <dd itemprop="nome">Pedro</dd>
32 <dt>total</dt> <dd itemprop="total">17.337</dd>
33 </dl>
34 </li>
35 </ol>
36
37 </body>
38 </html>
```

Adicionamos atributos especiais, *itemscope* e *itemprop*. Cada elemento *itemscope* define um item de dados. Cada *itemprop* define o nome de uma propriedade. O valor da propriedade é o conteúdo da *tag* HTML. A Microdata API nos fornece acesso especial a esses dados. Veja como acessar esses dados:

```
        resultados=document.getItems() for(var
i=0;i<resultados.length;i++){
alert(resultados[i].properties.nome[0].content+": R$ "+
resultados[i].properties.total[0].content)
        }
```

Diferentes tipos de dados

No exemplo acima, temos uma listagem de pessoas. Agora imagine que você precise ter, no mesmo documento, uma listagem de pessoas e carros. Poderia escrever assim:

```
1<!DOCTYPE html>
2<html>
3<head>
4<meta charset="UTF-8" />
5<title>Microdata 3</title>
6</head>
7<body>
8
 9  <h1>Resultados do trimestre</h1>
 10 <ol>
 11 <li>
 12 <dl itemscope>
 13 <dt>nome</dt> <dd itemprop="nome">Joaquim</dd>
 14 <dt>total</dt> <dd itemprop="total">10.764</dd>
 15 </dl>
 16 </li>
 17 <li>
 18 <dl itemscope>
 19 <dt>nome</dt> <dd itemprop="nome">Manoel</dd>
 20 <dt>total</dt> <dd itemprop="total">12.449</dd>
 21 </dl>
 22 </li>
 23 <li>
 24 <dl itemscope>
 25 <dt>nome</dt> <dd itemprop="nome">Antonio</dd>
 26 <dt>total</dt> <dd itemprop="total">9.202</dd>
 27 </dl>
 28 </li>
 29 <li>
 30 <dl itemscope>
 31 <dt>nome</dt> <dd itemprop="nome">Pedro</dd>
 32 <dt>total</dt> <dd itemprop="total">17.337</dd>
 33 </dl>
 34 </li>
 35 </ol>
36
 37 <h2>Carros mais vendidos</h2>
 38 <ol>
 39 <li>
 40 <dl itemscope>
 41 <dt>nome</dt> <dd itemprop="nome">Fusca</dd>
```

```
42 <dt>total</dt> <dd itemprop="total">382</dd>
43 </dl>
44 </li>
45 <li>
46 <dl itemscope>
47 <dt>nome</dt> <dd itemprop="nome">Brasília</dd>
48 <dt>total</dt> <dd itemprop="total">298</dd>
49 </dl>
50 </li>
51 <li>
52 <dl itemscope>
53 <dt>nome</dt> <dd itemprop="nome">Corcel</dd>
54 <dt>total</dt> <dd itemprop="total">102</dd>
55 </dl>
56 </li>
57 </ol>
58
59 </body>
60 </html>
```

Note que pessoas e carros têm propriedades em comum, nome e total. Quando você executar `document.getItems()` vai obter uma lista de todos os elementos com `itemscope`. Como obter uma lista apenas de pessoas ou de carros? Você pode adicionar a cada item um atributo `itemtype`, que diz de que tipo de entidade são aqueles dados:

```
1<!DOCTYPE html>
2<html>
3<head>
4<meta charset="UTF-8" />
5<title>Microdata 4</title>
6</head>
7<body>
8
9  <h1>Resultados do trimestre</h1>
10 <ol>
11 <li>
12 <dl itemscope itemtype="pessoa">
13 <dt>nome</dt> <dd itemprop="nome">Joaquim</dd>
14 <dt>total</dt> <dd itemprop="total">10.764</dd>
15 </dl>
16 </li>
17 <li>
18 <dl itemscope itemtype="pessoa">
19 <dt>nome</dt> <dd itemprop="nome">Manoel</dd>
20 <dt>total</dt> <dd itemprop="total">12.449</dd>
21 </dl>
22 </li>
23 <li>
24 <dl itemscope itemtype="pessoa">
25 <dt>nome</dt> <dd itemprop="nome">Antonio</dd>
26 <dt>total</dt> <dd itemprop="total">9.202</dd>
27 </dl>
```

```
28 </li>
29 <li>
30 <dl itemscope itemtype=”pessoa”>
31 <dt>nome</dt> <dd itemprop=”nome”>Pedro</dd>
32 <dt>total</dt> <dd itemprop=”total”>17.337</dd>
33 </dl>
34 </li>
35 </ol>
36
37 <h2>Carros mais vendidos</h2>
38 <ol>
39 <li>
40 <dl itemscope itemtype=”carro”>
41 <dt>nome</dt> <dd itemprop=”nome”>Fusca</dd>
42 <dt>total</dt> <dd itemprop=”total”>382</dd>
43 </dl>
44 </li>
45 <li>
46 <dl itemscope itemtype=”carro”>
47 <dt>nome</dt> <dd itemprop=”nome”>Brasília</dd>
48 <dt>total</dt> <dd itemprop=”total”>298</dd>
49 </dl>
50 </li>
51 <li>
52 <dl itemscope itemtype=”carro”>
53 <dt>nome</dt> <dd itemprop=”nome”>Corcel</dd>
54 <dt>total</dt> <dd itemprop=”total”>102</dd>
55 </dl>
56 </li>
57 </ol>
58
59 </body>
60 </html>
```

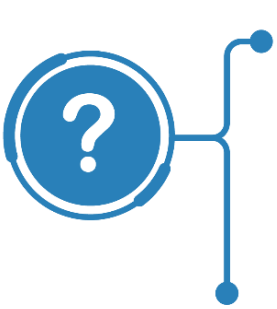
Agora você pode executar: `document.getItems('carro')` para obter só os carros, por exemplo.

Falando um idioma comum

Você deve ter notado que pode definir seus próprios padrões de metadados com microdata. Recomendo que, antes de criar seu próprio formato, procure por soluções no site **www.data-vocabulary.org**. Por exemplo, use o formato definido em **http://www.data-vocabulary.org/Organization.** Para o valor de `itemtype` deve ser a própria URL que documenta o formato. Veja como fica:

```
1     <!DOCTYPE html>
2     <html>
3     <head>
4     <meta charset=”UTF-8” />
5     <title>Visie Padrões Web</title>
```

```
6       </head>
7       <body>
8
9   <address itemscope itemtype="http://data-vocabulary.org/Organization">
10  <h1 itemprop="name">Visie Padrões Web</h1>
11  <div itemprop="address" itemscope itemtype="http://data-vocabulary.org/
        Address">
12  <p itemprop="street-address">Alameda dos Ubiatans, 257 - Planalto
        Paulista</p>
13  <p>
14  <span itemprop="locality">São Paulo</span> -
15  <span itemprop="region">SP</span> -
16  <span itemprop="country-name">Brasil</span>
17  </p>
18  <p itemprop="postal-code">04070-030</p>
19  </div>
20  <div itemprop="tel">+55.11.3477-3347</div>
21  </address>
22
23  </body>
24  </html>
```



Saiba mais

Os sistemas de busca, e outros sistemas que acessem seu site, podem entender e tratar esses dados. O Google já faz isso, veja neste endereço:

http://www.google.com/support/webmasters/bin/answer.py?hl=en&answer=146861

4.6. HISTÓRICO DE SESSÃO E API STORAGE

Histórico de Sessão

Você provavelmente conhece o objeto *history* do navegador e seus métodos `go`, `back` e `forward`. Ele nos permite, via javascript, um controle básico do histórico de navegação.

O objeto *history* no HTML5 recebeu dois novos métodos:

1. **pushState(data,title[,url])**: acrescenta uma entrada na lista de histórico.

2. **replaceState(data,title[,url])**: modifica a entrada atual na lista de histórico.

Com isso, você pode acrescentar itens à lista de histórico, associando dados ou mesmo uma URL a eles. Por exemplo, digamos que você tenha três elementos de

conteúdo em sua página e um script que exiba um por vez de acordo com os cliques do usuário no menu:

```
function showContent(n){

  // Escondemos todos os elementos de
conteúdo   for(var i=1;i<4;i++)
    document.getElementById('cont'+i).style.display='none'

  // Exibimos o elemento escolhido
 document.getElementById('cont'+n).style.display='block' }
```

Vamos fazer com que nosso script acrescente uma linha de histórico ao selecionar um elemento:

```
function showPage(n){

  // Escondemos todos os elementos de
conteúdo   for(var i=1;i<4;i++)
    document.getElementById('cont'+i).style.display='none'

  // Exibimos o elemento escolhido
document.getElementById('cont'+n).style.display='block' }

function showContent(n){
  // Mostramos o conteúdo escolhido
showPage(n)
  // Salvamos a página atual no
histórico
history.pushState({page:n},'Conteúdo '+n) }
```

Fazendo isso, cada vez que o usuário escolher um item no menu, o elemento será exibido e uma linha será acrescentada no histórico. O usuário poderá acessar normalmente esses itens de histórico usando o botão de voltar do navegador. Cada vez que ele usar o histórico, será disparado um evento *popstate*. Assim, para completar esse script, devemos tratar o evento:

```
function showPage(n){

  // Escondemos todos os elementos de
conteúdo   for(var i=1;i<4;i++)
    document.getElementById('cont'+i).style.display='none'

  // Exibimos o elemento escolhido


document.getElementById('cont'+n).style.display='block' }
```

```
        function showContent(n){
          // Mostramos o conteúdo escolhido
showPage(n)
          // Salvamos a página atual no
histórico
history.pushState({page:n},'Conteúdo '+n) }

        // Quando o usuário navegar no histórico, mostramos a
página relacionada: window.onpopstate=function(e){
if(e.state)      showPage(e.page) }
```

localStorage e sessionStorage

Para o HTML5 temos uma nova maneira de armazenar dados no client, a API Storage. Um objeto Storage possui os métodos:

1. **getItem(key)**: obtém um valor armazenado no Storage
2. **setItem(key,value)** guarda um valor no Storage
3. **removeItem(key)** exclui um valor do Storage
4. **clear()** limpa o Storage.

Estão disponíveis dois objetos no escopo global (window): localStorage e sessionStorage. O objeto localStorage armazena os dados no client sem expiração definida. Ou seja, se o usuário fechar o navegador e voltar ao site semanas depois, os dados estarão lá. O sessionStorage armazena os dados durante a sessão atual de navegação.

O código para armazenar um valor na Storage:

```
        localStorage.
setItem('userChoice',33)
```

E quando você precisar desse valor, em outra página:

```
        localStorage.getItem('userChoice')
```

Essa interface já é muito mais simples que a de Cookies. Mas pode ficar melhor. Você pode usar o Storage como um array. Por exemplo:

```
        if(!sessionStorage['theme']){

sessionStorage['theme']='oldfurniture'; }
```

Não há como isso ser mais simples! Além disso, o espaço de armazenamento sugerido pela documentação é de 5MB para cada domínio, resolvendo, acredito que por mais uma década, o problema de espaço de armazenamento local.

4.7. APLICAÇÕES OFFLINE

Caching

No HTML5 temos uma maneira de indicar ao navegador que elementos são necessários e devem ser postos em cache para que uma aplicação funcione offline.

```
1     <!DOCTYPE HTML>
2     <html>
3     <head>
4     <title>Clock</title>
5     <script src="clock.js"></script>
6     <link rel="stylesheet" href="clock.css">
7     </head>
8     <body>
9     <p>The time is: <output id="clock"></output></p>
10    </body>
11    </htmlFAZ>
```

Trata-se de um widget de relógio. Para funcionar, este HTML depende dos arquivos "clock.js" e "clock.css". Para permitir que o usuário acesse esta página *offile*, precisamos escrever um arquivo de manifesto, indicando que URLs devem ser postas em cache. Vamos preparar uma nova versão do widget, contendo o manifesto, que é um arquivo com a extensão .manifest e que deve ser do tipo MIME `text/cache-manifest`. Em nosso caso, o arquivo vai se chamar clock.manifest e terá o seguinte conteúdo:

```
      CACHE
MANIFEST
clock1.html
clock.css clock.js
```

```
1     <!DOCTYPE HTML>
2     <html manifest="clock.manifest">
3     <head>
4     <title>Clock</title>
5     <script src="clock.js"></script>
6     <link rel="stylesheet" href="clock.css">
7     </head>
8     <body>
9     <p>The time is: <output id="clock"></output></p>
10    </body>
11    </html>
```

Note que é recomendado que você insira o próprio HTML principal na lista de URLs do arquivo de manifesto, embora não seja necessário. Ao encontrar uma página com um arquivo de manifesto vinculado, o navegador fará cache das URLs listadas no manifesto e da própria página.

Note também que não é necessário que todas as URLs para cache estejam importadas no documento atual. O arquivo de manifesto pode contar todas as páginas de sua aplicação que forem necessárias para permitir o funcionamento offline, inclusive a navegação entre páginas.

O objeto ApplicationCache

O objeto ApplicationCache controla o status e operações de *caching* da página. Ele pode ser acessado via javascript, assim:

```
window.applicationCache
```

Seu método mais interessante é o *update(),* que faz com que o agente de usuário recarregue o cache da aplicação. Além disso, ele possui a propriedade status, cujo valor numérico pode ser um dos seguintes:

0 - UNCACHED
Não há um arquivo de manifesto nesta página ou apontando para ela

1 - IDLE
O objeto *ApplicationCache* está ocioso. O cache está atualizado.

2 - CHECKING
O arquivo de manifesto está sendo baixado e conferido.

3 - DOWNLOADING
As URLs vinculadas no manifesto estão sendo baixadas.

4 - UPDATEREADY
O cache é antigo, mas ainda não foi marcado como obsoleto.

5 - *OBSOLETE*
O cache foi marcado como obsoleto e precisa ser atualizado.

O objeto *ApplicationCache* também possui os seguintes eventos, relacionados a sua mudança de status:

- *onchecking*

- *onerror*
- *onnoupdate*
- *ondownloading*
- *onprogress*
- *onupdateready*
- *oncached*
- *onobsolete*

Como você pode ver, além de *onerror,* temos um evento para cada um dos status da lista acima.

Controle de status da aplicação

No exemplo do relógio acima não há formulários ou submissões Ajax. O agente de usuários não troca dados com o servidor. Assim é muito fácil fazer sua aplicação rodar offline, mas essa não é a realidade da maioria das aplicações. Vimos no capítulo anterior como fazer armazenamento local de dados. Com isso, você pode armazenar os dados que o navegador deveria enviar para o servidor enquanto a aplicação estiver offline e, tão logo ela esteja online, enviar tudo.

Para saber se a aplicação está online, basta acessar a propriedade onLine do objeto window. navigator:

```
      function salva(dados){
if(window.navigator.onLine){
enviaAjax(dados)
        }else{
          salvaLocal(dados)
        }
      }
```

E para disparar o envio quando a aplicação estiver online e avisar o usuário quando ela estiver offline, usamos os eventos ononline e onoffline do objeto window:

```
      window.ononline=function(){
enviaAjax(obtemLocal())
document.getElementById('warning').innerHTML='' }

      window.onoffline=function(){
        document.getElementById('warning').innerHTML='Aplicação
offline.'
      }
```

4.8. SCROLL IN TO VIEW E HIDDEN

Scrolling into view

Um truque simples, mas muito útil. Você pode fazer:

```
document.getElementById('aviso').scrollIntoView()
```

Isso vai rolar a página até que o elemento com o id "aviso" esteja visível no topo do *viewport*. Você pode passar um parâmetro opcional top:

```
document.getElementById('aviso').scrollIntoView(false)
```

O valor default é *true*. Se você passar false, a rolagem vai deixar o objeto visível na base do *viewport*.

hidden

Ocultar e exibir elementos é uma das tarefas mais comuns em Javascript. Em HTML5 existe um atributo específico para isso, o atributo *hidden*. Ao inseri-lo em um elemento assim:

```
<div hidden>Xi, se esconde!</div>
```

Ou assim:

```
<div hidden="true">Xi, se esconde!</div>
```

O elemento estará oculto.

hidden e Javascript

Acessar o atributo hidden em Javascript é muito conveniente:

```
function switchElement(elm){
  if(elm.hidden)
elm.hidden=false   else

elm.hidden=true }
```

Claro, você pode fazer:

```
function
switchElement(elm){
elm.hidden=!elm.hidden }
```

Considere usar o atributo *hidden*. Descobrir se o elemento está oculto lendo as propriedades *display* e *visibility* do CSS, além de dar mais trabalho, pode gerar confusão.

4.9. GEOLOCATION API

Métodos de Geolocalização

Há três populares maneiras de um agente de usuário descobrir sua posição no globo:

Geolocalização IP

É o método usado pela maioria dos navegadores web em computadores. Através de consultas *whois* e serviços de localização de IP, vai determinar a cidade ou região em que você está.

Triangulação GPRS

Dispositivos conectados a uma rede de celulares e sem um GPS, ou com o GPS desligado, podem determinar sua posição pela triangulação das antenas GPRS próximas. É bem mais preciso que o método baseado em IP, e a margem de erros é de metros.

GPS

É o método mais preciso. Em condições ideais, a margem de erro é de apenas 5 metros.

Embora essas sejam as três maneiras mais populares de se resolver o problema, podem não ser as únicas. Alguns agentes de usuário podem usar uma combinação desses métodos, ou mesmo um novo método que venha a ser inventado.

Para obter a posição do usuário, basta executar o script:

```
navigator.geolocation.getCurrentPosition(showpos)
```

Onde `showpos` é uma função *callback*, que vai receber um objeto de posicionamento. Veja um exemplo:

```
function showpos(position){
lat=position.coords.latitude
```

```
lon=position.coords.longitude    alert('Your
position: '+lat+','+lon)
      }
```

Claro, você pode fazer o que quiser, abrir um mapa, submeter a posição via Ajax, enviar os dados para um webservice etc.

O método *getCurrentPosition* recebe dois outros parâmetros. O primeiro é uma função para tratamento de erro. O segundo, um objeto de configuração.

Tratando erros

O usuário pode então escolher se deseja ou não compartilhar sua posição com o site. Além de o usuário poder dizer não, muita coisa pode dar errado na hora de obter a geolocalização. Para tratar isso, você pode passar o segundo parâmetro a getCurrentPosition:

```
navigator.geolocation.getCurrentPosition(showpos,erropos)
```

Caso algo dê errado, a função *erropos* vai receber um objeto *PositionError*, que tem o atributo *code*, que pode ter um dos seguintes valores:

1. - **Permissão negada**
   O usuário clicou em "não compartilhar".

2. - **Posição indisponível**
   O agente de usuário está desconectado, os satélites de GPS não puderam ser alcançados ou algum erro semelhante.

3. - **Timeout**
   Tempo esgotado ao obter uma posição. Você pode definir o tempo máximo ao chamar getCurrentPosition.

0. - **Erro desconhecido**
   Alguma outra coisa impediu o agente de usuário de obter uma posição.

Não trate a resposta do usuário como um erro

Em sua função de tratamento de erro, se obtiver o código de erro 1, não incomode o usuário com um novo pedido ou mensagem de erro, aceite a opção do usuário.

O objeto de configuração

O terceiro parâmetro de getCurrentPosition é um objeto de configuração, que pode ter as seguintes propriedades:

enableHighAccuracy

Se true, liga o modo de alta precisão. Num celular isso pode instruir o navegador, por exemplo, a usar o GPS ao invés da triangulação GPRS

timeout

O tempo em milissegundos que o agente do usuário vai esperar pela posição antes de disparar um erro tipo 3.

maximumAge

O tempo, em milissegundos, que o navegador pode cachear a posição.

watchPosition

Se o que você deseja é rastrear a posição do usuário continuamente, pode usar, ao invés de getCurrentPosition, o método watchPosition. Ele tem a mesma assinatura de getCurrentPosition:

```
w=navigator.geolocation.watchPosition(showpos,erropos)
```

A diferença é que a função *showpos* será chamada toda vez que a posição do usuário mudar. O valor de retorno é um número, que pode ser usado posteriormente para cancelar o *watcher*:

```
navigator.geolocation.clearWatch(w)
```

4.10. UNDO

e-mail de uma pasta para outra.

O objeto UndoManager possui os seguintes métodos e propriedades:

length o número de entradas no histórico

position o número da entrada atual no histórico

add(data,title) adiciona uma entrada específica no histórico. `data` pode ser um objeto literal com dados arbitrários. `title` é como essa entrada vai aparecer descrita na lista do histórico

remove(index) remove uma entrada específica do histórico

clearUndo() remove todas as entradas antes da atual no histórico

clearRedo() remove todas as entradas após a atual no histórico

Além disso, os itens no histórico podem ser acessados com `window.undoManager[index]`.

Respondendo às ações de undo e redo

Cada vez que o usuário disparar uma ação de *undo* ou *redo*, e o item do histórico for um objeto *undo*, será disparado o evento correspondente, *window.onundo* ou *window.onredo*. As funções associadas a estes eventos receberão como parâmetro um objeto *event,* contendo uma propriedade data, cujo valor é o objeto *undo* que você inseriu no histórico.

Veja o exemplo:

```
window.onundo=function(e){
  alert('Refazer a alteração:
'+e.data) }
```

Disparando as ações de undo e redo

Se você quiser oferecer em sua aplicação botões para undo e redo, basta que eles executem:

`document.execCommand('undo')` Ou:

```
document.execCommand('redo')
```

5. CSS – Parte 1

Objetivos

- Descobrir o CSS
- Compreender as funções básicas
- Realizar os exemplos

Introdução

A seguir os primeiros passos para entender e usar o CSS, sua origem e funções básicas para ser consideradas na elaboração de um projeto e aprimoramento visual.

Vamos juntos!

5.1. O que é CSS?

O CSS é a linguagem capaz de realizar a formatação das informações entregues pelo HTML. Essas informações podem ser: imagens, textos, vídeos, áudios ou qualquer outro elemento. Lembre-se! CSS = formatação de informação. Essa formatação na maioria das vezes é visual, mas não exclusivamente. Por exemplo no CSS Aural, é possível manipular o áudio entregue ao visitante pelo sistema de leitura de tela.

Com o CSS podemos formatar algumas características básicas: cores, background, características de *font, margins, paddings*, posição e controlamos como em trabalhos manuais. Entretanto para um desempenho mais elaborado não esqueça, pode ser necessário um software de edição de imagens, Javascript para alguns tratamentos no HTML e gestão sobre o SEO do seu site.

Entendendo

Guia de otimização de mecanismo de pesquisa (SEO)

https://developers.google.com/search/docs/beginner/seo-starter-guide?hl=pt-br/

Com as atualizações do CSS3 e com os browsers atualizando o suporte ao CSS, o conjunto HTML5 + CSS3 está em um novo patamar de funcionalidades e resultados. Algumas novidades:

1) selecionar primeiro e último elemento;
2) selecionar elementos pares ou ímpares;
3) selecionarmos elementos específicos de um determinado grupo de elementos;
4) gradiente em textos e elementos;
5) bordas arredondadas;
6) sombras em texto e elementos;
7) manipulação de opacidade;
8) controle de rotação;
9) controle de perspectiva;
10) animação;
11) estruturação independente da posição no código HTML;

Vamos ao CSS!

5.2. SELETORES COMPLEXOS

A sintaxe do CSS é simples: seletor { propriedade: valor; }

A propriedade é a característica que você deseja modificar no elemento.
O valor é referente a esta característica.
Se você quer modificar a cor do texto, o valor é um Hexadecimal, RGBA ou até mesmo o nome da cor por extenso.

Propriedades são criadas todos os dias pelos grupos de desenvolvimento do CSS, não é necessário saber todas, entretanto os seletores são fundamentais e você deve dominá-los. É através dos seletores que você irá escolher um determinado elemento dentro todos os outros elementos do site para formatá-lo. A compreensão e o uso dos seletores é que realizam a mágica do CSS.

O que é um seletor?

O seletor representa uma estrutura e essa estrutura é usada como uma condição para determinar quais elementos de um grupo de elementos serão formatados. Os seletores encadeados e seletores agrupados são a base do CSS. Veja o exemplo:

Exemplo de seletor encadeado:

```
div p strong a {  color: red;}
```

Este seletor formata o link (a), que está dentro de um strong, que está dentro de P e que por sua vez está dentro de um DIV.

Exemplo de seletor agrupado:

```
strong, em, span {    color: red; }
```

Você agrupa elementos separados por vírgula para que herdem a mesma formatação.

Estes seletores são para cobrir suas necessidades básicas de formatação de elementos. Os seletores complexos selecionam elementos que talvez você precisaria fazer algum *script* em Javascript para poder marcá-lo com uma CLASS ou um ID para então você formatá-lo. Com os seletores complexos você consegue formatar elementos que antes eram inalcançáveis.

Exemplo de funcionamento - Imagine que você tenha um título (h1) seguido de um parágrafo (p). Você precisa selecionar todos os parágrafos que vem depois de um título h1. Com os seletores complexos você fará assim:

```
h1 + p {
color:red;
}
```

Esses seletores é um dos mais simples e mais úteis.

Abaixo, veja uma lista de seletores complexos e quais as suas funções. Não é possível apresentar todas, além de existirem atualizações.

Saiba mais

Lista de seletores elaborada e atualizada pelo W3C.

https://www.w3.org/TR/selectors-3/#selectors

5.3. GRADIENTE

Uma das *features* mais interessantes é a criação de gradientes apenas utilizando CSS. Todos os browsers mais novos como Safari, Opera, Firefox, EDGE e Chrome já aceitam essa *feature* e você pode utilizá-la hoje.

Veja abaixo um exemplo de código, juntamente com o fallback de imagem:

```
      div {      width:200px;
height:200px;
background-color: #FFF;

          /* imagem caso o browser não aceite a feature */
background-image: url(images/gradiente.png);

          /* Firefox 3.6+ */
          background-image: -moz-linear-gradient(green, red);

          /* Safari 5.1+, Chrome 10+ */
          background-image: -webkit-linear-gradient(green, red);

          /* Opera 11.10+ */
          background-image: -o-linear-gradient(green,
      red); }
```

Atenção: Até que os browsers implementem de vez essa *feature*, iremos utilizar seus prefixos.

Como ficou:

5.3.1. *"Stops" ou definindo o tamanho do seu gradiente*

O padrão é que o gradiente ocupe 100% do elemento como vimos no exemplo anterior, mas muitas vezes queremos fazer apenas um detalhe.

Nesse caso nós temos que definir um STOP, para que o browser saiba onde uma cor deve terminar para começar a outra. Perceba o 20% ao lado da cor vermelha. O browser calcula quanto é 20% da altura (ou largura dependendo do caso) do elemento, e começa o gradiente da cor exatamente ali.

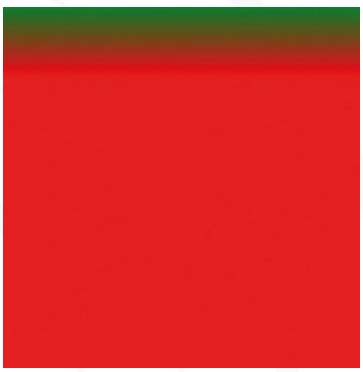
O código de exemplo segue abaixo:

```
/* Firefox 3.6+ */
 background-image: -moz-linear-gradient(green, red 20%);

 /* Safari 5.1+, Chrome 10+ */
 background-image: -webkit-linear-gradient(green, red
20%);

 /* Opera 11.10+ */     background-image: -o-
linear-gradient(green, red 20%);
```

Veja o resultado:

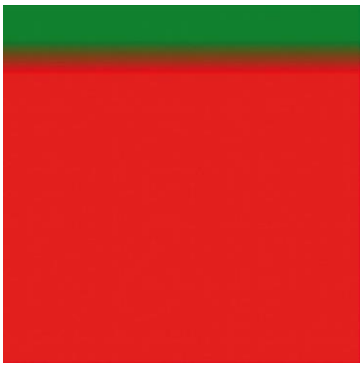


Se colocarmos um valor para o verde, nós iremos conseguir efeitos que antes só conseguiríamos no Illustrator ou no Photoshop. Segue o código e o resultado logo após:

```
        /* Firefox 3.6+ */
         background-image: -moz-linear-gradient(green 10%, red
20%);

         /* Safari 5.1+, Chrome 10+ */
         background-image: -webkit-linear-gradient(green 10%, red
20%);

         /* Opera 11.10+ */     background-image: -o-linear-
gradient(green 10%, red 20%);
```



Perceba que o tamanho da transição vai ficando menor à medida que vamos aumentando as porcen*tag*ens.

5.4. COLUMNS

Com o controle de colunas no CSS, podemos definir colunas de texto de forma automática. Até hoje não havia maneira de fazer isso de maneira inteligente com CSS e o grupo de propriedades columns pode fazer isso de maneira livre de gambiarras.

column-count

A propriedade column-count define a quantidade de colunas terá o bloco de textos.

```
      /* Define a quantidade de colunas, a largura é definida
uniformemente. */
     -moz-column-count: 2;
     -webkit-column-count: 2;
```

column-width

Com a propriedade column-width definimos a largura destas colunas.

```
     /* Define qual a largura mínima para as colunas. Se as
colunas forem espremidas, fazendo com que a largura delas fique
menor que este valor, elas se transformam em 1 coluna
automaticamente */
     -moz-column-width: 400px;
     -webkit-column-width: 400px;
```

O column-width define a largura mínima das colunas. Na documentação do W3C é a seguinte: imagine que você tenha uma área disponível para as colunas de 100px. Ou seja, você tem um div com 100px de largura (width). E você define que as larguras destas colunas (column-width) sejam de 45px. Logo, haverá 10px de sobra, e as colunas irão automaticamente ter 50px de largura, preenchendo este espaço disponível.

column-gap

A propriedade column-gap cria um espaço entre as colunas, um gap.

```
 /* Define o espaço entre as colunas. */
-moz-column-gap: 50px;
-webkit-column-gap: 50px;
```

Utilizamos aqui os prefixos -moz- e -webkit-, estas propriedades não funcionam oficialmente em nenhum browser. Mas já podem ser usados em browsers como Firefox e Safari.

5.5. Transform 2d

A propriedade t*ransform* manipula a forma com que o elemento aparecerá na tela. Você poderá manipular sua perspectiva, escala e ângulos. Uma transformação é especificada utilizando a propriedade *transform*. Os browsers que suportam essa propriedade, a suportam com o prefixo especificado.

Os valores possíveis até agora estão especificados abaixo:

scale

O valor scale modificará a dimensão do elemento. Ele aumentará proporcionalmente o tamanho do elemento levando em consideração o tamanho original do elemento.

skew

Skew modificará os ângulos dos elementos. Você pode modificar os ângulos individualmente dos eixos X e Y como no código abaixo:

```
-webkit-transform: skewY(30deg);
-webkit-transform: skewX(30deg);
```

translation

O translation moverá o elemento no eixo X e Y. O interessante é que você não precisa se preocupar com floats, positions, margins etc. Se você aplica o translation, ele moverá o objeto e pronto.

rotate

O rotate rotaciona o elemento levando em consideração seu ângulo, especialmente quando o ângulo é personalizado com o transform-origin.

CSS Transform na prática

Veja o código abaixo e seu respectivo resultado:

```
img {
    -webkit-transform: skew(30deg); /* para webkit */
    -moz-transform: skew(30deg); /* para gecko */
    -o-transform: skew(30deg); /* para opera */
    transform: skew(30deg); /* para browsers sem
prefixo */ }
```

O código acima determina que o ângulo da imagem seja de 30deg. Colocamos um exemplo para cada prefixo de browser. Ficando assim:

Várias transformações em um único elemento

Para utilizarmos vários valores ao mesmo tempo em um mesmo elemento, basta definir vários valores separando-os com espaços em uma mesma propriedade transform:

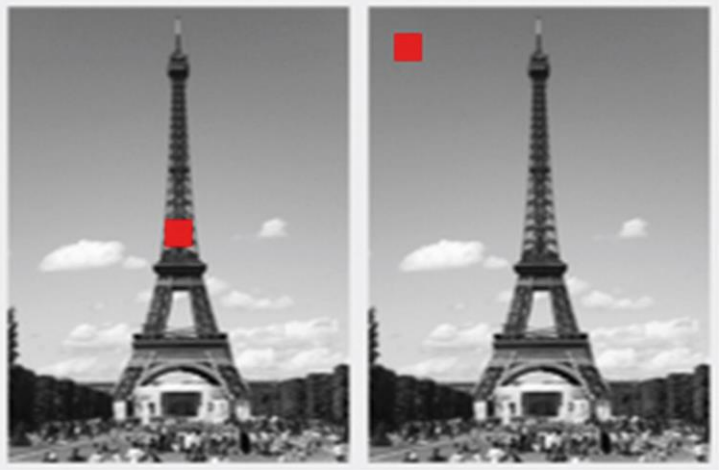
```
img {
    -webkit-transform: scale(1.5) skew(30deg); /* para
webkit */
    -moz-transform: scale(1.5) skew(30deg); /* para gecko */
    -o-transform: scale(1.5) skew(30deg); /* para opera */
transform: scale(1.5) skew(30deg); /* para browsers sem prefixo */
}
```

transform-origin

A propriedade transform-origin define qual o ponto do elemento a transformação terá origem. A sintaxe é idêntica ao background-position. Observe o código abaixo:

```
        img {
            -webkit-transform-origin: 10px 10px; /* para webkit */
            -moz-transform-origin: 10px 10px; /* para webkit */
            -o-transform-origin: 10px 10px; /* para
webkit */     transform-origin: 10px 10px; /* para
webkit */
        }
```

Como padrão as transições sempre acontecem tendo como ponto de âncora o centro do objeto. Há algumas situações que pode ser que você queira modificar essa âncora, fazendo com que por exemplo, a rotação aconteça em algum dos cantos do elemento. O código de exemplo acima fará com que a transformação aconteça a partir dos 10px do topo no canto esquerdo. Veja a comparação entre o padrão e o resultado do código acima.



A propriedade *transform* fica mais interessante quando a utilizamos com a propriedade *transition,* onde podemos executar algumas animações básicas manipulando os valores de transformação.

5.6. Transições e animações

A propriedade *transition*, a regra *keyframe* e outras propriedades vieram acrescentar funções ao CSS permitindo produzir animações e transições. O fato é que as animações via CSS colocam a experiência do usuário para outro patamar.

transition

A propriedade *transition* altera as cores como a seguir:

```
a {      color:
white;
background: gray;
}
```

No código definimos que o link terá sua cor de texto igual a preta e seu background será cinza. O resultado esperado é que ao passar o mouse no link a cor do texto seja modificada, mudando do branco para o preto e que a cor de background mude de cinza para vermelho. O código abaixo faz exatamente isso:

```
a {      color:
white;
background: gray;
}

a:hover {
color: black;
background: red;
}
```

O problema é que a transição é muito brusca. O browser apenas modifica as características entre os dois blocos e pronto. Não há nenhuma transição suave entre os dois estados. Podemos fazer a mudança de um estado para outro utilizando a propriedade transition. Suponha que ao passar o mouse, as mudanças acontecessem em um intervalo de meio segundo. Bastaria colocar a propriedade transition no a:hover e pronto. Ao passar o mouse, o browser modificaria as características do link com uma pequena transição de meio segundo. O código seria como se segue abaixo:

```
a:hover {
color: black;
background: red;
        -webkit-transition: 0.5s;
}
```

Dessa forma a transição apenas acontece quando o *hover* é ativado. O problema é que ao tirar o mouse, o browser volta bruscamente para as características iniciais. Para modificar isso basta inserir também a propriedade *transition* no estado inicial.

```
a {     color:
white;
background: gray;
        -webkit-transition: 0.5s;
}
a:hover {
color: black;
background: red;
        -webkit-transition: 0.5s;
}
```

O que a propriedade *transition* faz é comparar os valores das propriedades em comum entre os dois estados do link ou de qualquer outro elemento, assim ela modifica suavemente os valores quando há a ativação. Esta é uma técnica simples e que serve para manipularmos transições básicas como cor, tamanho, posição etc. Agora suponha que em um bloco há uma determinada propriedade que no outro bloco não há, como no código abaixo:

```
a {   border:1px solid
orange;     color: white;
background: gray;
        -webkit-transition: 0.5s;
}
a:hover {
color: black;
background: red;
        -webkit-transition: 0.5s;
}
```

Nesse caso o browser detecta que há uma propriedade no primeiro estado, mas não no segundo, por isso ele não faz a transição desta propriedade, apenas das propriedades em comuns. Abaixo veja o código. copie em um arquivo HTML e veja o efeito:

```
<!DOCTYPE html>
<html lang=”pt-br”>
<head>
        <meta charset=”utf-8”>
        <title>CSS Transition</title>
        <style type=”text/css” media=”screen”>
        a {
        color:white;
    background:gray;
        -webkit-transition: 0.5s linear;
    }
a:hover {
color:black;
        background:red;
        -webkit-transition: 0.5s linear;
    }
    </style>
</head>
<body> <a href=”#”>Link! Hello World!</a>
</body>
</html>
```

Propriedade animation e regra keyframe

A propriedade *trasition* trabalha de forma muito simples e inflexível. Você praticamente diz para o browser qual o valor inicial e o valor final para que ele aplique a transição automaticamente, controlamos praticamente apenas o tempo de execução. Para termos mais controle sobre a animação temos a propriedade *animation* que trabalha juntamente com a *rule keyframe*.

Basicamente você consegue controlar as características do objeto e suas diversas transformações definindo fases da animação. Observe o código abaixo e veja seu funcionamento:

```
@-webkit-keyframes rodar {
```

```
        from {
            -webkit-transform:rotate(0deg);
        }
        to {
            -webkit-transform:rotate(360deg);
        }
    }
```

O código acima define um valor inicial e um valor final. Agora vamos aplicar esse código a um elemento. Minha ideia é fazer um DIV girar. ;-) O código HTML até agora é este. Fiz apenas um *div* e defini este *keyframe*:

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <title></title>
    <meta charset="utf-8">
    <style>
    @-webkit-keyframes rodaroda {
        from {
            -webkit-transform:rotate(0deg);
        }
        to {
            -webkit-transform:rotate(360deg);
        }
    }
    </style>
</head>
<body>
<div></div>
</body>
</html>
```

Primeiro você define a função de animação, no caso é o nosso código que define um valor inicial de 0 graus e um valor final de 360 graus. Agora vamos definir as características deste DIV.

```
div {
        width:50px;
        height:50px;
        margin:30% auto 0;
        background:black;
}
```

Nas primeiras linhas defini qual será o estilo do *div*. Ele terá uma largura e uma altura de 50px. A *margin* de 30% do topo garantirá um espaço entre o objeto e o topo do documento, e background preto.

A propriedade *animation* tem uma série de propriedades que podem ser resumidas em um *shortcode* bem simples. Veja a tabela logo a seguir para entender o que cada propriedade significa:

Propriedade	**Definição**
animation-name	Especificamos o nome da função de animação
animationduration	Define a duração da animação. O valor é declarado em segundos.
animation-timingfunction	Descreve qual será a progressão da animação a cada ciclo de duração. Existem uma série de valores possíveis e que pode ser que o seu navegador ainda não suporte, mas são eles: ease, linear, ease-in, ease-out, ease-in-out, cubic-bezier(<number>, <number>, <number>, <number>) [, ease, linear, ease-in, ease-out, ease-in-out, cubic-bezier(<number>, <number>, <number>, <number>)]*
animationinteration-count	O valor padrão é *ease*. Define o número de vezes que o ciclo deve acontecer. O padrão é um, ou seja, a animação acontece uma vez e

	para. Pode ser também infinito definindo o valor *infinite* no valor.
animationdirection	Define se a animação irá acontecer ou não no sentido inverso em ciclos alternados. Ou seja, se a animação está acontecendo no sentido horário, ao acabar a animação, o browser faz a mesma animação no elemento, mas no sentido anti-horário. Os valores são *alternate* ou *normal*.
animation-playstate	Define se a animação está acontecendo ou se está pausada. Você poderá por exemplo, via script, pausar a animação se ela estiver acontecendo. Os valores são *running* ou *paused*.
animation-delay	Define quando a animação irá começar. Ou seja, você define um tempo para que a animação inicie. O valor 0, significa que a animação começará imediatamente.

Voltando para o nosso código, vamos aplicar algumas dessas propriedades:

```
div {
        width:50px;
        height:50px;
        margin:30% auto 0;
        background:black;
-webkit-animation-name: rodaroda;
-webkit-animation-duration: 0.5s;
-webkit-animation-timing-function: linear;
-webkit-animation-iteration-count: infinite;
-webkit-animation-direction: alternate;
}
```

Veja que na propriedade *animation-name* chamamos o mesmo nome que demos na nossa função de *keyframe* logo no começo da explicação. Depois definimos uma duração de ciclo de meio segundo. Definimos que o comportamento da animação será linear, e com a propriedade *animationiteration-count* definimos que ele girará infinitamente. E por último definimos pelo *animation-direction* que a animação deverá ser alternada, ou seja, o DIV

girará para um lado, e quando alcançar o final da animação, o browser deverá alternar essa animação.

Podemos melhorar esse código utilizando a versão *shortcode*, que é mais recomendado. Veja a ordem que devemos escrever as propriedades *animation* em forma de *shortcode*:

animation: [<animation-name> || <animation-duration> || <animation-timing-function> || <animation-delay> || <animation-iteration-count> || <animation-direction>] [, [<animation-name> || <animation-duration> || <animation-timing-function> || <animation-delay> || <animation-iteration-count> || <animation-direction>]]*

Aplicando isso ao nosso código:

```
div {
        width:50px;
        height:50px;
        margin:30% auto 0;
        background:black;
        -webkit-animation: rodaroda 0.5s linear infinite alternate;
}
```

Pronto. Agora temos um elemento que gira sem parar, hora para direita hora para esquerda.

Definindo ciclos

Nós definimos no *keyframe* do nosso último exemplo apenas um início e um fim. Mas e se quiséssemos que ao chegar na metade da animação o nosso elemento ficasse com o background vermelho? Ou que ele mudasse de tamanho, posição etc.? É aí onde podemos flexibilizar melhor nosso *keyframe* definindo as fases da animação. Por exemplo, podemos dizer para o elemento ter uma cor de background diferente quando a animação chegar aos 10% do ciclo, e assim por diante.

Veja o exemplo:

```
@-webkit-keyframes rodaroda {
        0% {
                -webkit-transform:rotate(0deg);
        }
        50% {
                background:red;
                -webkit-transform:rotate(180deg);
        }
        100% {
                -webkit-transform:rotate(360deg);
        }
}
```

Definimos acima que o início da animação o elemento começará na posição normal, 0 graus. Quando a animação chegar aos 50% do ciclo, o elemento deverá ter girado para 180 graus e o background dele deve ser vermelho. E quando a animação chegar a 100% o elemento deve ter girado ao todo 360 graus e o background, como não está sendo definido, volta ao valor padrão, no nosso caso black, que foi definido no CSS onde formatamos este DIV.

Logo nosso elemento girará para sempre e ficará alternando a cor de fundo de preto para vermelho. Fiz um exemplo bem simples modificando apenas o background, mas você pode muito bem definir um position e modificar os valores de left e top para fazer o elemento se movimentar.

No exemplo estão definidos apenas 3 estágios (0%, 50% e 100%) você pode definir um maior número de estágios: 5%, 10%, 12%, 16% e etc. Adequando as fases da animação às suas necessidades.

Há exemplos muito interessantes na internet onde podemos ver todo o poder das animações feitas pela CSS.

Referências

Mdn Web docs – Animation (CSS)

https://developer.mozilla.org/pt-BR/docs/Web/CSS/animation

5.7. Bordas

Definir imagem para as bordas é uma daquelas propriedades da CSS que você se pergunta como vivíamos antes de conhecê-la. É muito mais fácil entender testando na prática, por isso sugiro que se você estiver perto de um computador, faça testes enquanto lê este texto. A explicação pode não ser suficiente em algumas partes, mas a prática irá ajudá-lo a entender.

Esta propriedade ainda está em fase de testes pelos browsers, por isso utilizaremos os prefixos para ver os resultados. Utilizarei apenas o prefixo do Safari, mas o Firefox já entende essa propriedade muito bem.

Dividindo a imagem

Para posicionar a imagem devidamente em seu objeto o browser divide a imagem em 9 seções: Quando a imagem é colocada no elemento, o browser posiciona os cantos da imagem juntamente com os cantos correspondentes do elemento. Ou seja, o bloco 1 da imagem é colocado no canto superior esquerdo, o 3 no canto superior direito e assim por diante. Se você fizer o teste, a imagem aparecerá no elemento como se estivesse desproporcional. Isso é normal porque a imagem deve seguir as proporções do elemento e não as suas próprias.

Referências

w3schools.com!

https://www.w3schools.com/cssref/tryit.asp?filename=trycss3_border-image

Comportamento da imagem

O comportamento da imagem é a parte mais importante porque define como o centro da imagem (no caso do nosso exemplo a seção de número 5), irá ser tratada. Há vários valores, mas que é mais simples de se entender é a stretch, que estica e escala a imagem para o tamanho do elemento aplicado. Há outros valores como round e repeat. Mas hoje alguns browsers não têm tanto suporte e acabam ou ignorando esses valores ou como no caso do Safari e o Chrome, interpretam o round como o repeat. Vamos nos concentrar com o stretch e você entenderá como funciona a propriedade.

Aplicação

Vamos utilizar a imagem abaixo para aplicar a propriedade. Iremos fazer um botão ao estilo iPhone. A coisa é simples e sugiro que você faça testes individualmente brincando com os valores das bordas e com diversas outras imagens para ver como funciona o recurso.

Aplicar o estilo em um link. O código segue abaixo:

```
a {
        display:block;
        width:100px;
        text-align:center;
        font:bold 13px verdana, arial, tahoma, sans-serif;
        text-decoration:none;
        color:black;
}
```

Para inserir a imagem, colocamos as duas linhas abaixo:

```
        border-width:10px;
        -webkit-border-image: url(button.png) 10 stretch;
```

Os efeitos precisam ser testados e avaliados, por isso, teste na prática essa propriedade para que não haja problemas a implementar em seus projetos.

5.8. Múltiplos backgrounds

A sintaxe para múltiplos backgrounds é praticamente idêntica a sintaxe para definir um background. Segue abaixo um exemplo:

```
div {
        width:600px;
        height:500px;
        background:
        url(img1.png) top left repeat-X,
        url(img2.png) bottom left repeat-Y;
```

```
}
```

Definimos apenas uma propriedade background, o valor dessa propriedade em vez de conter apenas um valor como normalmente fazemos, poderá haver vários, com suas respectivas definições de posição, *repeat e attachment (fixed*).

6. CSS – Parte 2

Objetivos

- Aprofundar a compreensão do CSS
- Compreender outras funções
- Realizar os exemplos

Introdução

A seguir novas funções para aplicar o CSS.

Vamos juntos!

6.1. MÓDULO TEMPLATE LAYOUT

Considere as propriedades *flexbox, grid e float,* para realizar a organização de seu conteúdo, inicialmente considere a propriedade *float* cuida da diagramação do site, desde os elementos que definirão as áreas mestres do site até os pequenos detalhes de imagens e ícones, porém seus efeitos sobre os elementos próximos, pode ser indesejado. Tendo em vista estes e outros problemas o W3C criou o *Template Layout*. Esse módulo consiste em uma forma para criar e organizar os elementos e informações do layout com mais flexibilidade.

Podemos dividir a construção do layout em duas grandes partes: 1) Definição dos elementos mestres e grid a ser seguido. 2) Formatação de *font*, cores, *background*, bordas etc.

As propriedades nesta especificação trabalham associando uma política de layout de um elemento, denominada de *template-based positioning*, a qual dotará ao elemento uma grid invisível para alinhar seus elementos descendentes.

Porque o layout deve se adaptar em diferentes janelas e tamanhos de papéis, as colunas e linhas do grid devem ser fixas ou flexíveis dependendo do tamanho.

Template-based positioning são uma alternativa para a propriedade position com o valor *absolute.* A recomendação é usar o *position:absolute* para posicionar o elemento nas coordenadas exatas que você quer.

Sintaxe e funcionamento

O módulo *Template Layout* basicamente define *slots* de *layout* para que você encaixe e posicione seus elementos. O mapeamento dos *slots* é feito com duas propriedades que já conhecemos que este módulo adiciona mais alguns valores e funcionalidades, são as propriedades *position* e *display*. Dessa forma a propriedade display definirá a estrutura do *Grid* e a propriedade position irá posicionar seus elementos nestes *slots*.

Veja o código HTML abaixo:

```
<div class=”geral”>
  <nav class=”menu”>...</nav>
  <aside class=”menulateral”>...</aside>
  <aside class=”publicidade”>...</aside>
  <article class=”post”>...</article>
  <footer>...</footer>
</div>
```

Agora iremos definir a posição destes elementos. O código CSS seria assim:

```
.geral {
     display: “aaa”
                 “bcd”
                 “eee”;
}
nav.menu {position:a;}
aside.menulateral {position:b;}
aside.publicidade {position:d;}
article.post {position:c;}
footer {position:e;}
```

O funcionamento da propriedade display

A propriedade *display* define a organização dos *slots*. Um *slot* é um local onde o seu elemento ficará. Cada elemento fica em um slot. Uma forma é utilizar o elemento display trabalha como um *table,* onde seu conteúdo será localizado por colunas e linhas. A única diferença é que o número de linhas e colunas não dependem do conteúdo é fixa pelo valor da propriedade. E a outra principal diferença é que a ordem dos descendentes no código não importa. Cada letra diferente é um slot de conteúdo diferente, o @ define um sinal para um slot padrão. E o “.” (ponto) define um espaço em branco. Caso de erro, a declaração é ignorada pelo browser.

Pra definir a altura da linha (*row-height*) podemos utilizar o valor padrão “auto”, que define altura que a altura, para a largura da coluna (*col-width*) é definida com valores fixos, como o *row-height*. Procure os outros comandos para melhora o ajuste de sua tabela.

Referências

w3schools.com!
CSS TEMPLATES

https://www.w3schools.com/css/css_templates.asp

Definindo a largura e altura dos slots

Para definir a altura dos slots, utilizamos uma sintaxe diretamente na propriedade display. Veja abaixo um exemplo de como podemos fazer isso:

```
display: “a a a” /150px
         “b c d”
         “e e e” / 150px
         100px 400px 100px;
```

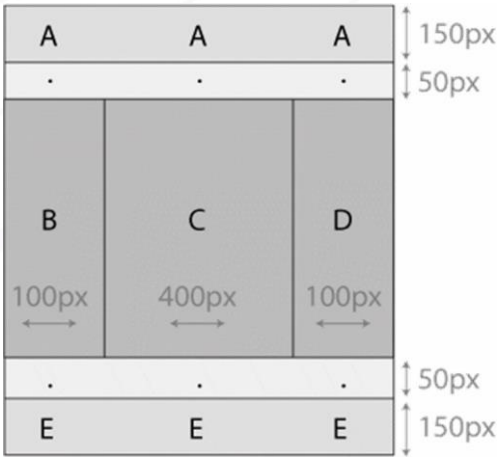
Formatando de uma maneira que você entenda, fica assim:

```
display: “a a a” /150px
         “b c d”
         “e e e” /150px
         100px  400px  100px;
```

Ou seja, a primeira coluna do grid terá 100px de largura, a segunda 400px e a terceira 100px.

O espaço entre as colunas é definido com “.” (ponto). Veja o exemplo abaixo:

```
display: “a a a” /150px
        “ . . .” /50px
         “b c d”
         “. . .” /50px
         “e e e” /150px
         100px  400px  100px;
```



No exemplo apresenta duas colunas no código compostos por pontos em vez de fazer com letras. Isso quer dizer que entre as colunas do grid haverá um espaço em branco de 50px de altura.

O funcionamento da propriedade position

O valor da propriedade position especifica qual linha e coluna o elemento será colocado no *template*. Você escreve apenas uma letra por elemento. Vários elementos podem ser colocados em um mesmo slot. Logo estes elementos terão o mesmo valor de position.

Abaixo veja uma imagem que pegamos diretamente do exemplo do W3C. O layout é muito simples:

• navigation	**Weather** There will be weather	**Football** People like football. **Chess** There was a brawl at the chess tournament	**Your Horoscope** You're going to die (eventually).
		Copyright some folks	

Este layout é representado pelo código abaixo. Primeiro o HTML:

```
<!DOCTYPE html>
<html>
<head>
```

```
<style>
p {color: black;}
h3 {color: blue;}

p.ex1 {display: none;}
p.ex2 {display: inline;}
p.ex3 {display: block;}
p.ex4 {display: inline-block;}
</style>
</head>
<body>
<ul id=”nav”>
     <li>navigation</li>
</ul>
<div id=”content”>
    <div class=”module news”>
         <h3>Weather</h3>
         <p>There will be weather</p>
    </div>
    <div class=”module sports”>
         <h3>Football</h3>
         <p>People like football.</p>
    </div>
    <div class=”module sports”>
         <h3>Chess</h3>
         <p>There was a brawl at the chess tournament</p>
    </div>
    <div class=”module personal”>
         <h3>Your Horoscope</h3>
         <p>You’re going to die (eventually).</p>
    </div>
    <p id=”foot”>Copyright some folks</p>
</div>
</body>
</html>
```

Agora veja o CSS com toda a mágica:

```
body {
        display: “a     b”
                10em   *;
}
#nav {
        position: a;
}
#content {
```

```
        position: b;
        display: “c   .   d   .   e “
                 “.   .   .   .   .“  /1em
                 “.   .   f   .   .“
                 *  1em  *  1em  *;
    }
    .module.news {position: c;}
    .module.sports {position: d;}
    .module.personal {position: e;}
    #foot {position: f;}
```

Lembre-se que não importa a posição dos elementos no código. E essa é a mágica. Podemos organizar o código HTML de acordo com nossas necessidades e levando em consideração SEO, Acessibilidade e processo de manutenção. O HTML fica totalmente intacto separado de qualquer formatação.

Pseudo-elemento ::slot()

Você pode formatar um slot específico ao declará-lo no CSS. Suponha que você queira que um determinado slot tenha um fundo diferente, alinhamento etc. Essa formatação será aplicada diretamente no slot e não no elemento que você colocou lá. Fica mais simples de se formatar porque você não atrela um estilo ao elemento e sim ao slot.

```
body { display: “aaa”
               “bcd” }
body::slot(b) { background: #FF0 }
body::slot(c), body::slot(d) { vertical-align: bottom }
```

As propriedades aplicadas no pseudo elemento slot() seguem abaixo:

- Todos as propriedades background.
- vertical-align
- overflow
- box-shadow, block-flow e direction ainda estão sendo estudados pelo W3C se elas entrarão ou não.

6.2. CORES

RGBA

As cores na WEB são mais bem descritas pela forma hexadecimal, porém o padrão RGB também pode ser utilizado em qualquer propriedade que permita cor, exemplo: *background, border* etc.

```
p {
background:rgb(255,255,0);
padding:10px;
        font:13px verdana;
}
```

Este código RGB define que o background o elemento P será amarelo.

RGBA e a diferença da propriedade OPACITY

Até então nós só podíamos escrever cores sólidas, sem nem ao menos escolhermos a opacidade dessa cor, a partir do CSS3 incluiu a propriedade *opacity*, resolve essas situações.

Veja um exemplo de código aplicado abaixo:

```
p {
background:rgba(255,255,0, 0.5);
padding:10px;
        font:13px verdana;
}
```

O último valor é referente ao canal Alpha, onde 1 é totalmente visível e 0 é totalmente invisível. No exemplo acima está com uma opacidade de 50%.

currentColor

Imagine que o *currentColor* é uma variável cujo seu valor é definido pelo valor da propriedade color do seletor. Veja o código abaixo:

```
p {
background:red;
padding:10px;
font:13px verdana;
color: green;
        border:1px solid green;
}
```

Note que o valor da propriedade color é igual ao valor da cor da propriedade *border*. Há uma redundância de código, que nesse caso é irrelevante, mas quando falamos de dezenas de arquivos de CSS modulados, com centenas de linhas cada, essa redundância pode incomodar a produtividade. A função do *currentColor* é simples: ele cópia para outras propriedades do mesmo seletor o valor da propriedade color. Veja o código abaixo para entender melhor:

```
p {
background:red;
padding:10px;
font:13px verdana;
color: green;
border:1px solid currentColor;
}
```

Isso funciona em qualquer propriedade que faça utilização de cor como *background, border* etc. Obviamente não funciona para a propriedade color. O currentColor também não funciona em seletores separados, ou seja, você não consegue atrelar o valor da propriedade color ao *currentColor* de outro seletor.

6.3. Paged média

A especificação de *Paged Media* permite formatar as páginas, controlando suas medidas, tamanhos, margens, quebras de páginas etc.

@page

Definiremos com CSS3 um modelo de página que especifica como o documento será formatado em uma área retangular, chamada de *page box,* com larguras e alturas

limitadas. Nem sempre o *page box* tem referência correspondente para uma folha de papel física, como normalmente conhecemos em diversos formatos: folhas, transparências e etc. Esta especificação formata o *page box*, mas é o browser ou o meio de acesso que o usuário está utilizando que tem a responsabilidade de transferir o formato do *page box* para a folha de papel no momento da impressão.

O documento é transferido no modelo da página para um ou mais page boxes. O page box é uma caixa retangular que será sua área de impressão. Isso é como se fosse um viewport do browser. Como qualquer outro box, a page box tem margin, border, padding e outras áreas. O conteúdo e as áreas de margin do page box tem algumas funções especiais:

A área de conteúdo do page box é chamada de area da página ou page area. O conteúdo do documento flui na área de página. Os limites da área da página na primeira página estabelecem o retângulo inicial que contém o bloco do documento.

A área da margem da page box é dividido em 16 caixas de margem ou margin boxes. Você pode definir para cada caixa de margem sua própria borda, margem, padding e áreas de conteúdo. Normalmente essas caixas de margem são usadas para definir headers e footers do documento.

Confesso que o nome utilizado (caixas de margem) é meio estranho.

As propriedades do page box são determinadas pelas propriedades estabelecidas pelo page context, o qual é a regra de CSS @page. Para definir a largura e altura de uma page box não se usa as propriedades width e height. O tamanho da page box é especificada utilizando a propriedade size no page context.

6.3.1. Terminologia e Page Model (modelo de página)

O *page box* tem algumas áreas simples de se entender que facilitará a explicação.

Page box

O *page box* é onde tudo acontece. Tenha em mente que o page box é o viewport das medias impressas. É lá que conterá as áreas de margem, padding, border e onde o texto será consumido.

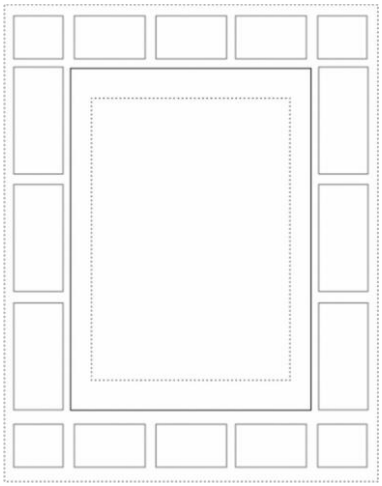
A largura e altura do page box é determinada pela propriedade size. Em um caso simples, o page box tem a largura e a altura de uma folha. Entretanto em casos complexos onde page box difere das folhas de papel em valores e orientações já que você pode personalizar de acordo com sua necessidade.

Page area

A page area é a area de conteúdo (content area) do page box.

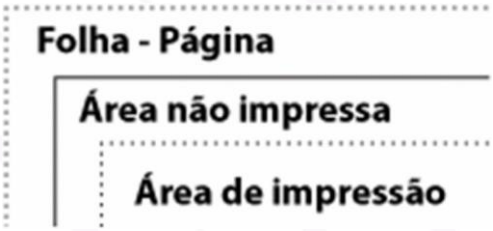
Margin box

Margin boxes contém boxes para header *e footer*. São conjuntos de 16 boxes onde você pode inserir conteúdo útil como número da página, título do livro, etc, etc, etc. Essas áreas ficam fora do *Page area*. Cada um tem suas *margins, paddings* e bordas individuais. Veja o diagrama abaixo para visualizar melhor.



Page sheet

A folha, a página, a superfície que será impresso o conteúdo. A ilustração abaixo mostra a representação de uma folha.



Non-printable area - Área não impressa

A área de não impressão é a área onde o dispositivo de impressão não é capaz de imprimir. Esta área depende do dispositivo que você está utilizando. O page box fica dentro da área de impressão.

Área de impressão

A área impressa é onde o dispositivo de impressão é capaz de imprimir. A área de impressão é o tamanho da page sheet menos a área de não impressão. Como a área de não impressão, a área útil de impressão depende muito do dispositivo. O dispositivo pode ajustar o conteúdo para que seja impresso sem problemas nessa área. Cada dispositivo tem seu meio de ajuste.

6.3.2. Propriedade size

A propriedade size especifica o tamanho e a orientação da área do de conteúdo, o page box. O tamanho do page box pode ser definida com valores absolutos (px) ou relativos (%). Você pode usar três valores para definir a largura e a orientação do page box:

auto

O page box irá ter o tamanho e orientação do page sheet escolhido pelo usuário.

landscape

Define que a página será impressa em modo paisagem. O page box neste caso tem o mesmo tamanho da página, mas o lado maior é o horizontal. Se o tamanho da página não for definido, o tamanho é especificado pelo usuário e seu dispositivo.

portrait

Define que a página será impressa em modo retrato. O page box neste caso tem o mesmo tamanho da página, mas o lado maior é o vertical. Se o tamanho da página não for definido, o tamanho é especificado pelo usuário e seu dispositivo.

Veja um exemplo abaixo:

```
@page {
   size: auto; /* auto is the valor padrão */
   margin: 10%; /* margem */
 }
```

Como nesse caso a margem é variável, ela está sendo relativa às dimensões da página. Logo se a página uma A4, com as dimensões: 210mm x 297mm, as margens serão 21mm e 29.7mm.

Outro exemplo:

```
@page {
      size: 210mm 297mm; /* definem o page-sheet para um tamanho
de A4 */
      }
```

Page-size

O *page-size* pode ser especificado utilizando um dos media names abaixo. Isso é o equivalente a utilizar os valores escritos diretamente na propriedade size. Contudo é muito melhor utilizar o nome de um formato de formato de papel.

Formato	Descrição
A5	A página deve ser definida para o tamanho ISO A5: 148mm x 210mm.
A4	A página deve ser definida para o tamanho ISO A4: 210 mm x 297 mm.
A3	A página deve ser definida para o tamanho ISO A3: 297mm x 420mm.
B5	A página deve ser definida para o tamanho ISO B3 media: 176mm x 250mm.
B4	A página deve ser definida para o tamanho ISO B4: 250mm x 353mm.
Letter	A página deve ser definida para o tamanho papel carta: 8.5 pol x 11 pol

Abaixo veja um exemplo de aplicação:

```
@page {
  size: A4 landscape;
}
```

Entendendo

O W3C tem uma especificação que pode ser encontrada aqui:

http://www.w3.org/TR/css3-page/

6.4. @FONT-FACE

A regra @font-face é utilizada para que você utilize fontes fora do padrão do sistema em seus sites. Para que isso funcione, nós disponibilizamos as fontes necessárias em seu servidor e linkamos estas fontes no arquivo CSS. A sintaxe é bem simples e tem suporte a todos os navegadores, com algumas ressalvas.

```
@font-face {
        font-family: helveticaneue;
        src: url(helveticaNeueLTStd-UltLt.otf);
}
```

Na primeira linha você customiza o nome da font que você usará durante todo o projeto. Na segunda linha definimos a URL onde o browser procurará o arquivo da font para baixar e aplicar no site.

Você aplica a fonte como abaixo:

```
p {font:36px helveticaneue, Arial, Tahoma, Sans-serif;}
```

Suponha que o usuário tenha a font instalada, logo ele não precisa baixar a font, assim podemos indicar para que o browser possa utilizar o arquivo da própria máquina do usuário. Menos requisições, mais velocidade. Veja o código abaixo:

```
@font-face {
      font-family: helveticaneue;
      src: local(HelveticaNeueLTStd-UltLt.otf),
      url(HelveticaNeueLTStd-UltLt.otf);
}
```

Abaixo segue uma série de formatos que podem ser usados e que os browsers podem adotar:

String	Font Format	Common Extensions
truetype	TrueType	.ttf
opentype	OpenType	.ttf, .otf
truetupe-aat	TrueType with Apple Advanced Typography extensions	.ttf
embedded-opentype	Embedded Open Type	.eot
woff	WOFF (Web Open Font Format)	.woff
Svg	SVG Font	.svg, .svgz

6.5. PRESENTATION-LEVELS

A informação na web é reutilizada de diversas maneiras. Toda informação publicada é reutilizada por diversos meios de acesso, seja o seu browser, leitor de tela ou robôs de busca. O HTML proporciona essa liberdade para a informação. Por ser uma linguagem muito simples, podemos reutilizar a informação marcada com HTML em diversos meios de acesso. Mas o HTML não cuida da forma com que o usuário vai visualizar a informação em seu dispositivo. O HTML apenas exibe a informação. A maneira que o usuário consome essa informação é diferente em cada um dos meios de acesso e dispositivos. É aí que entra todo o poder do CSS. O CSS formata a informação para que ela possa ser acessível em diversos usar agents (meios de acesso). Se você acessa o site do seu banco pelo monitor de 22'' da sua casa ou pelo seu celular, a informação tem que aparecer bem-organizada em ambos os cenários. É o CSS que organiza visualmente essas informações.

Além disso podemos apresentar a informação de diversas formas em um mesmo dispositivo. Por exemplo: você pode ver uma galeria de imagens da maneira convencional, clicando nas thumbs das fotos ou ver em forma de slideshow. Podemos levar essas experiências para websites de conteúdo textual também. A especificação de presentation-levels é uma das especificações que levam o usuário a terem conteúdo mostrados de uma outra forma da qual estamos acostumados. É muito útil para apresentações de slides, com efeitos, transições e etc ou qualquer documento que seria mais bem apresentado no formato de apresentação, como uma proposta, documentos técnicos e etc.

6.5.1. Como funciona o modelo

O modelo por trás da especificação é simples. Cada elemento no documento é definido como um "elemento de apresentação" ou no formato original "element's presentation level" - EPL.

O EPL pode ser explicito em uma folha de estilo ou calculado automaticamente baseado na posição do elemento pela estrutura do documento. É assim que o browser calcula para mostrar os elementos progressivamente, como se faz normalmente em programas de apresentação.

O elemento fica em um dos três seguintes níveis que também são representadas por classes: *belowlevel, at-level e above-level*. Dependendo da pontuação de EPL que o

browser dá, o elemento fica em um determinado nível. Essas pseudo-classes podem e devem ser modificadas via CSS.

6.5.2. A propriedade presentation-level

A propriedade *presentation-level* define como os valores de apresentação (EPL) de um determinado objeto devem ser calculados. São três valores possíveis: números inteiros, *increment e same*.

Quando definimos um valor inteiro, o elemento tem aquele valor fixo.

Quando colocamos *increment*, o valor do objeto aumenta um ponto em relação ao objeto anterior. Suponha que há duas LI em uma UL. A primeira LI tem o valor de 1, a segunda tem valor de 2 e assim por diante.

Quando definimos o valor *same*, o *browser* computa o mesmo valor do objeto anterior.

Isso tudo vai ficar mais esclarecido com os exemplos a seguir.

Utilizando o mesmo exemplo da especificação do W3C, temos o código abaixo:

```
<!DOCTYPE html>
<html>
<body>
<h1>strategies</h1>
<h2>our strategy</h2>
<ul>
  <li>divide</li>
  <li>conquer
    <p>(in that order)</p>
  </li>
</ul>
<h2>their strategy</h2>
<ul>
  <li>obfuscate</li>
  <li>propagate</li>
</ul>

</body>
</html>
```

Vamos definir o CSS de *presentation-levels* para esse HTML adicionado o código CSS:

```
@media projection {
```

```
        h1 { page-break-before: always }
li { presentation-level: increment }
        :below-level { color: black }
        :at-level { color: red }
        :above-level { color: silver }
      }
```

Definimos que o H1 irá sempre iniciar em uma nova página.

Mas o mais importante é a propriedade presentation-level que definimos para a LI. Isso quer dizer que a cada LI o browser contará mais um ponto.

As três pseudo-classes que falamos no começo do texto: below-level, at-level, above-level, que formata os elementos que foram mostrados anteriores, o que elemento que está sendo mostrado e o próximo elemento.

Sendo assim, o browser calcula a pontuação de cada um dos elementos utilizados no exemplo como mostra abaixo:

HTML	Valor de EPL
strategies	0
our strategy	0
	0
divide	1
conquer	2
	0
their strategy	0
	0
obfuscate	1
propagate	2
	0

Temos um outro exemplo, segue abaixo o HTML e logo depois a tabela com os valores de EPL:

```
<!DOCTYPE html>
<html>
```

```
        <style>
          @media projection {   h1 {
presentation-level: 0; }   h2 {
presentation-level: 1; }   h3 {
presentation-level: 2; }   body * {
presentation-level: 3; }   :above-
level { display: none; }
          }
        </style>
        <body>
        <h1>strategies</h1>
        <h2>our strategy</h2>
        <ul>
          <li>divide</li>
          <li>conquer</li>
        </ul>
        <h2>their strategy</h2>
        <ul>
          <li>obfuscate</li>
          <li>propagate</li>
        </ul>

        </body>
        </html>
```

Perceba que agora definimos no CSS que tudo dentro de body tem o valor de 3. Logo o H1 que foi definido como 0 pela propriedade *presentation-level* tem o valor de 3.

Definimos também *display:none*; para os próximos elementos. Agora veja a pontuação aplicada:

HTML	Valor de EPL
strategies	3
our strategy	2
	0
divide	0
conquer	0
	0
their strategy	2
	0
obfuscate	0
propagate	0
	0

Entendendo

O W3C tem uma especificação completa do presentation-levels:

http://www.w3.org/TR/css3-preslev/

Concluindo

A oportunidade é perdida pela maioria das pessoas porque ela vem vestida de macacões e se parece com trabalho.

" De fato, não fracassei ao tentar, cerca de 10.000 vezes, desenvolver um acumulador. Simplesmente, encontrei 10.000 maneiras que não funcionam."

"Jamais fiz alguma coisa que valesse a pena por acidente, nem qualquer de minhas invenções aconteceu por acidente; elas aconteceram pelo trabalho."

Thomas Alva Edison (2332 Patentes)