Co-funded by the
Erasmus+ Programme
of the European Union

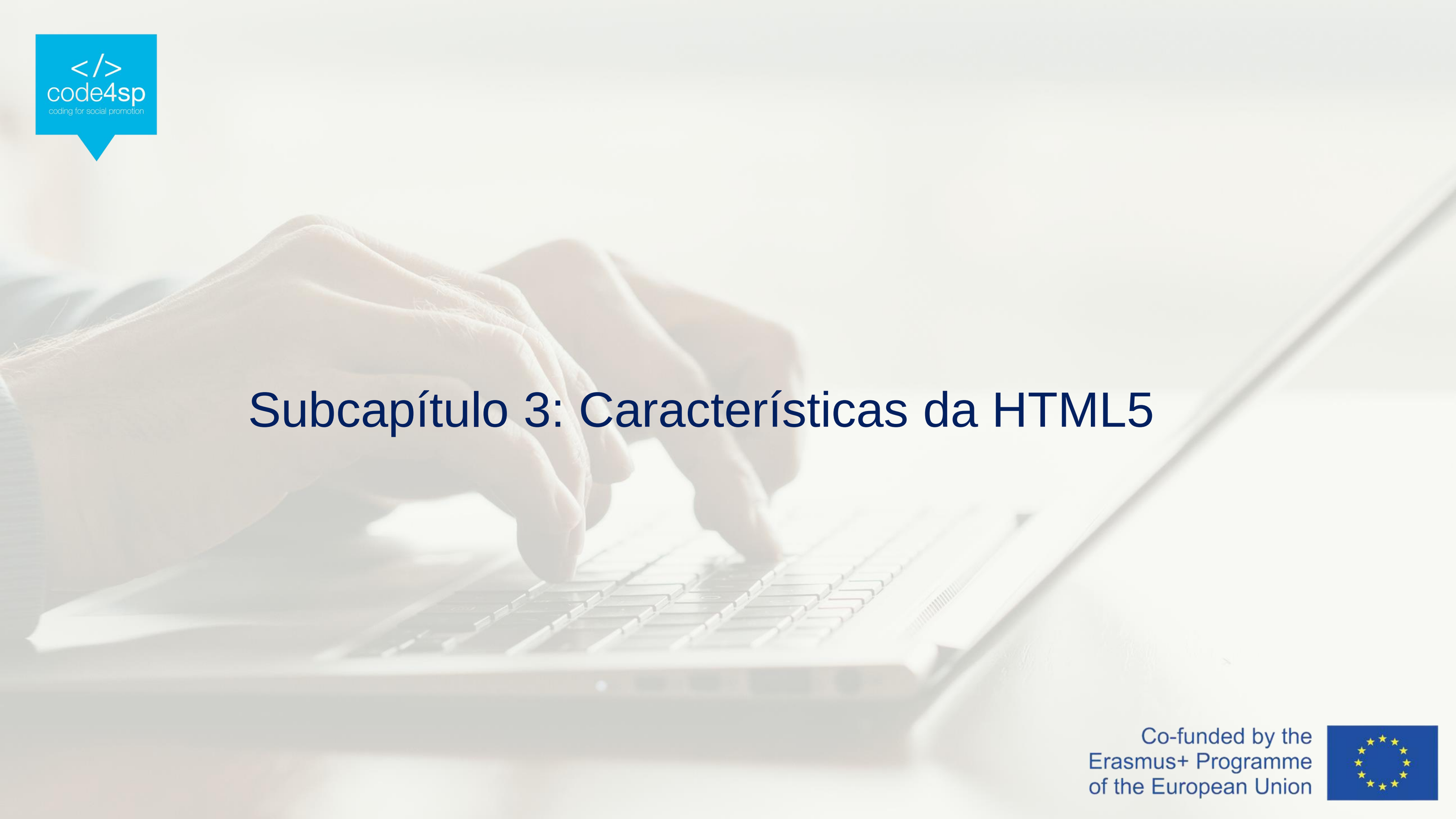
code4sp
coding for social promotion

Materiais do Formador de HTML
Subcapítulo 3 – Características da HTML5

WP3: Materiais de Formação do Code4SP

Apresentação elaborada por

Projeto financiado com o apoio da Comissão Europeia. A informação contida nesta publicação (comunicação) vincula exclusivamente o autor, não sendo a Comissão responsável pela utilização que dela possa ser feita.
(Número do projeto: 621417-EPP-1-2020-1-PT-EPPKA3-IPI-SOC-IN)



Subcapítulo 3: Características da HTML5

Co-funded by the
Erasmus+ Programme
of the European Union

Características da HTML5

Neste subcapítulo, serão explicadas as principais características da quinta, e última, versão principal da HTML.

Novos tipos de *input* HTML5

- A HTML5 introduz vários novos tipos de <input>, como por exemplo: *email, date, time, color, range*, entre outros, com o objetivo de melhorar a experiência do utilizador e tornar os formulários mais interativos.
- No entanto, se um navegador não reconhecer estes novos *input*, considerá-los-á uma caixa de texto normal.

Novos tipos de *input* HTML5

Alguns exemplos de novos *input* são:

- *color*
- *date*
- *datetime-local*
- *email*
- *month*
- *number*
- *range*
- *search*
- *tel*
- *time*
- *url*
- *week*

Novos tipos de *input* HTML5

O *input* ‘Color’

Este permite selecionar a cor a partir de uma paleta de cores e fornece o código da cor em formato hexadecimal (ex., #000000, que corresponde à cor preto, a cor predefinida caso o utilizador não especifique um valor), tal como podemos confirmar na figura do próximo diapositivo. É de salientar que o *input* ‘color’ é suportado pelos principais navegadores (*Firefox*, *Chrome*, *Opera*, *Safari* (12.1+), *Edge* (14+)), mas não é suportado pelo *Microsoft Internet Explorer* nem por versões mais antigas dos navegadores *Apple Safari*.

Novos tipos de *input* HTML5

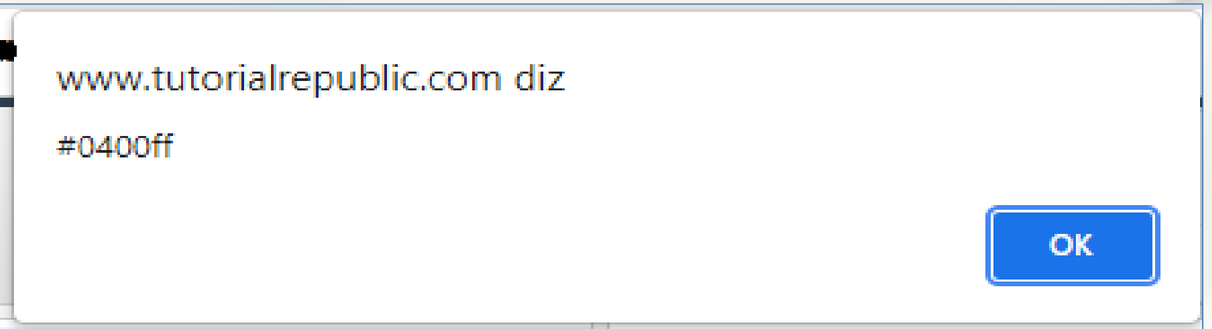
O *input* 'Color'

Apps

codeLAB

Show Output

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  <title>HTML5 Color Input Type</title>
5  <script>
6      function getValue() {
7          var color = document.getElementById("mycolor").value;
8          alert(color);
9      }
10 </script>
11 </head>
12 <body>
13     <form>
14         <label for="mycolor">Select Color:</label>
15         <input type="color" value="#00ff00" id="mycolor">
16         <button type="button" onclick="getValue();">Get Value</button>
17     </form>
18 </body>
19 </html>
```



Select Color: Get Value

Novos tipos de *input* HTML5

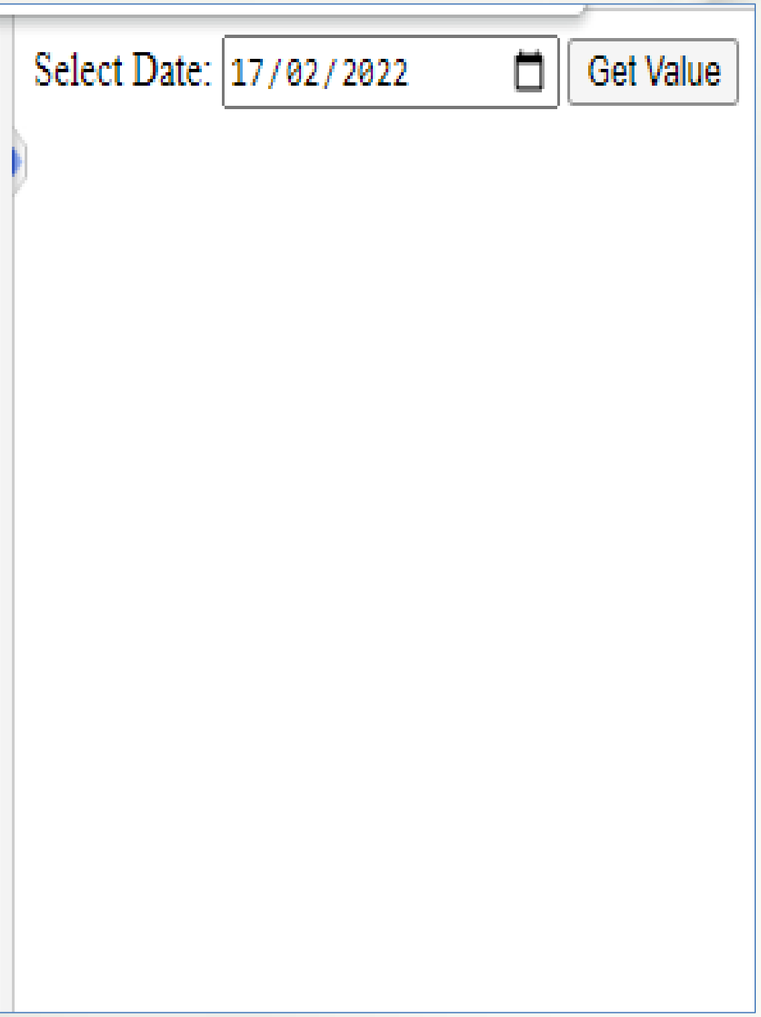
O *input* ‘Date’

Este permite ao utilizador selecionar a data a partir de um calendário *drop-down*, no qual ele(a) pode selecionar o ano, mês e dia (mas não a hora). Esta função é também suportada por grande parte dos navegadores, exceto o *Internet Explorer* e o *Safari*.

Novos tipos de *input* HTML5

O *input* 'Date'

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Date Input Type</title>
<script>
    function getValue() {
        var date = document.getElementById("mydate").value;
        alert(date);
    }
</script>
</head>
<body>
    <form>
        <label for="mydate">Select Date:</label>
        <input type="date" value="2019-04-15" id="mydate">
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
</body>
</html>
```



Novos tipos de *input* HTML5

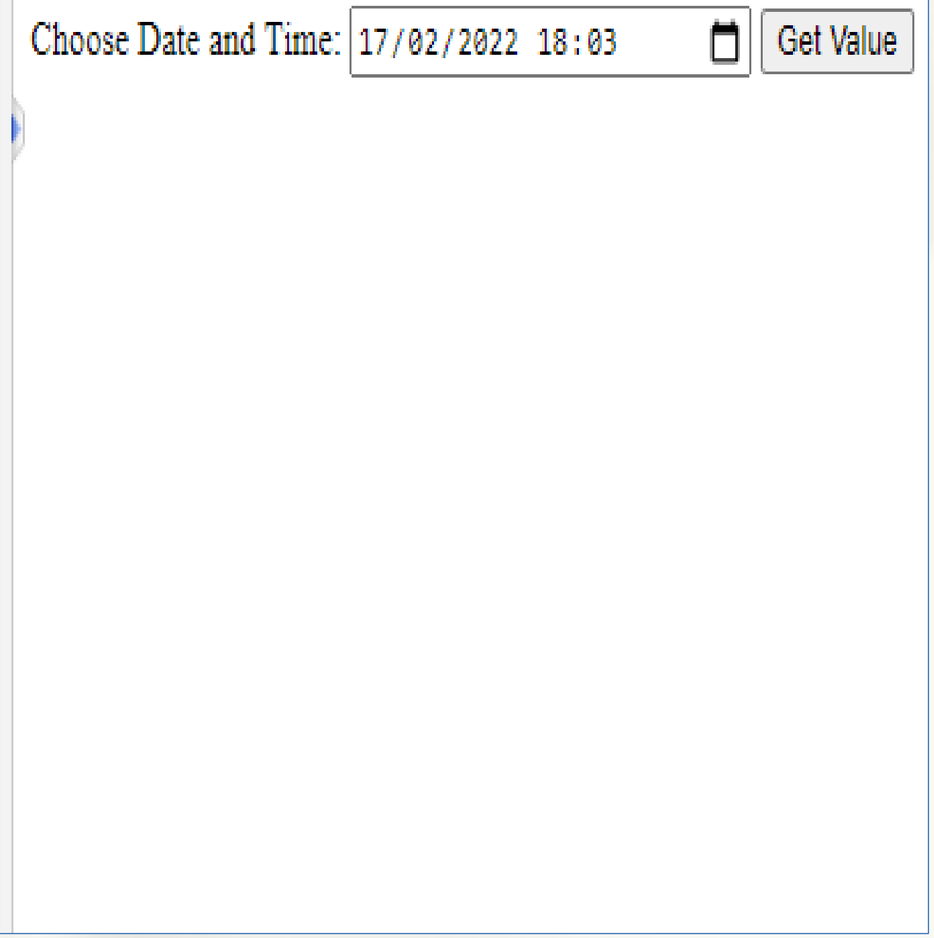
O *input* ‘datetime-local’

Este *input* permite ao utilizador selecionar a hora e data locais, incluindo ano, mês, dia e a hora em horas e minutos. É suportado pelo *Safari*, *Firefox* e *IE*, mas não pelo *Chrome*, *Edge* nem *Opera*.

Novos tipos de *input* HTML5

O *input* ‘datetime-local’

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Datetime-local Input Type</title>
<script>
    function getValue() {
        var datetime = document.getElementById("mydatetime").value;
        alert(datetime);
    }
</script>
</head>
<body>
    <form>
        <label for="mydatetime">Choose Date and Time:</label>
        <input type="datetime-local" id="mydatetime">
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
</body>
</html>
```



Novos tipos de *input* HTML5

O *input* ‘email’

Permite ao utilizador inserir o seu endereço de e-mail, sendo *email* o tipo de *input* preferencial. É muito semelhante a um *input* de texto comum, mas quando combinado com o atributo *required*, os navegadores podem procurar padrões para garantir que é inserido um endereço de e-mail no formato adequado. O *input* ‘email’ pode ser editado para ter diferentes estados de validação, quando inseridos os seguintes valores: *:valid, :invalid or :required*. Este tipo de *input* é suportado pelos principais navegadores.

Novos tipos de *input* HTML5

O *input* 'email'

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Email Input Type</title>
<style>
    input[type="email"]:valid{
        outline: 2px solid green;
    }
    input[type="email"]:invalid{
        outline: 2px solid red;
    }
</style>
<script>
    function getValue() {
        var email = document.getElementById("myemail").value;
        alert(email);
    }
</script>
</head>
<body>
    <form>
        <label for="myemail">Enter Email Address:</label>
        <input type="email" id="myemail" required>
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
</body>
</html>
```

Novos tipos de *input* HTML5

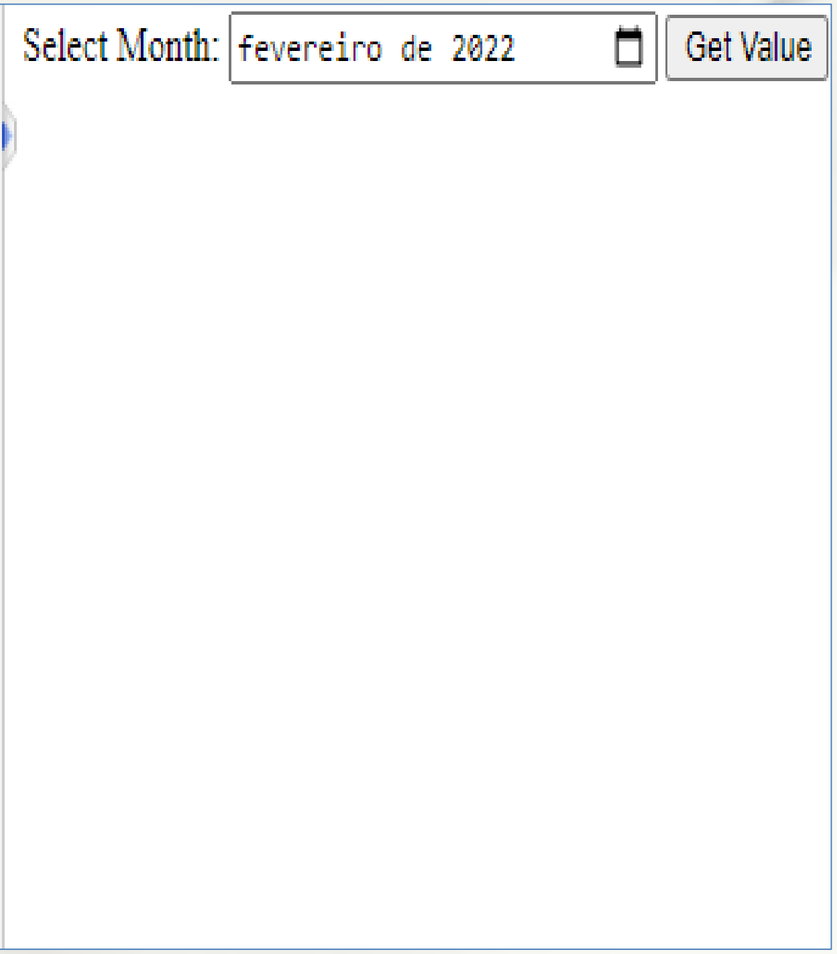
O *input* ‘month’

Este *input* é muito semelhante aos referidos anteriormente, já que permite selecionar o mês (**month**) e o ano a partir de um calendário *drop-down* (SENDO ‘YYYY’ o ano e ‘MM’ o mês). Não é suportado pelo *Firefox*, *Safari* nem *Internet Explorer*. É apenas suportado pelo *Chrome*, *Edge* e *Opera*.

Novos tipos de *input* HTML5

O *input* ‘month’

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Month Input Type</title>
<script>
    function getValue() {
        var month = document.getElementById("mymonth").value;
        alert(month);
    }
</script>
</head>
<body>
    <form>
        <label for="mymonth">Select Month:</label>
        <input type="month" id="mymonth">
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
</body>
</html>
```



Novos tipos de *input* HTML5

O *input* ‘number’

Este *input* é utilizado para inserir um código numérico.

O *web designer* pode ainda limitar os valores a ser inseridos pelo utilizador, para tal, devem ser usados os atributos *min*, *max*, e *step*. É suportado pelos principais navegadores.

Novos tipos de *input* HTML5

O *input* ‘number’

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Number Input Type</title>
<style>
    input[type="number"]:valid{
        outline: 2px solid green;
    }
    input[type="number"]:invalid{
        outline: 2px solid red;
    }
</style>
<script>
    function getValue() {
        var number = document.getElementById("mynumber").value;
        alert(number);
    }
</script>
</head>
<body>
    <form>
        <label for="mynumber">Enter a Number:</label>
        <input type="number" min="1" max="10" step="0.5" id="mynumber">
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
    <p><strong>Note</strong>: If you try to enter the number out of the range (1-10) or
text character it will show error.</p>
</body>
</html>
```

Enter a Number: 1.5 Get Value

Note: If you try to enter the number out of the range (1-10) or text character it will show error.

Novos tipos de *input* HTML5

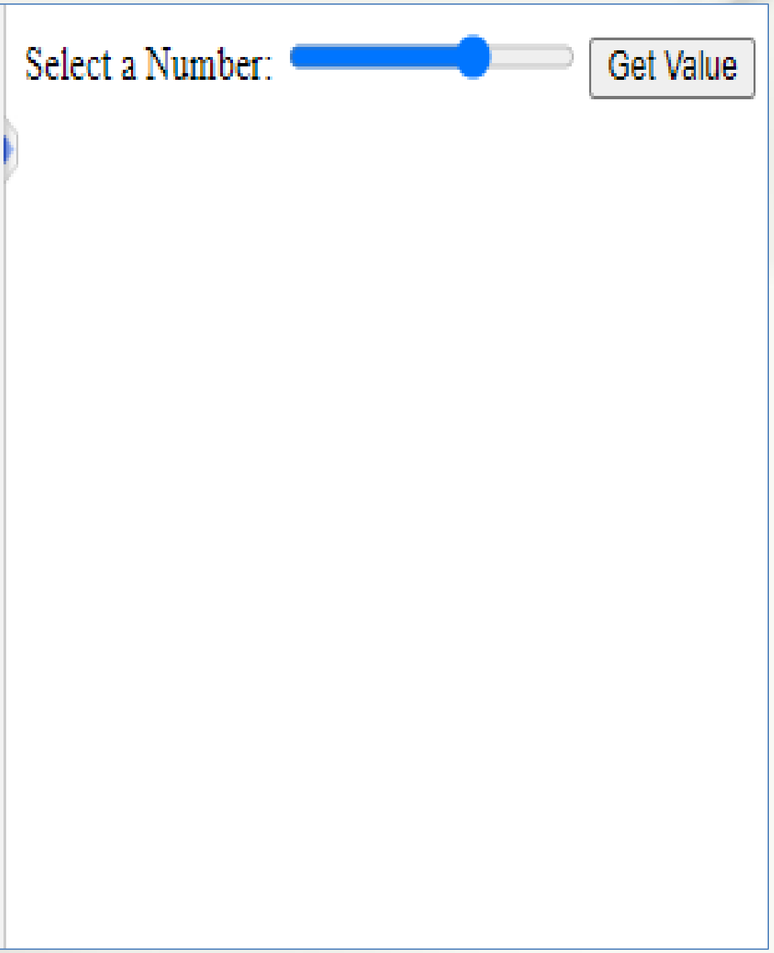
O *input* ‘range’

Pode ser usado para registar um valor numérico num determinado intervalo. É muito semelhante ao *input number* mencionado anteriomente, contudo, é mais fácil inserir um número. É suportado por todos os navegadores.

Novos tipos de *input* HTML5

O *input* ‘range’

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Range Input Type</title>
<script>
    function getValue() {
        var number = document.getElementById("mynumber").value;
        alert(number);
    }
</script>
</head>
<body>
    <form>
        <label for="mynumber">Select a Number:</label>
        <input type="range" min="1" max="10" step="0.5" id="mynumber">
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
</body>
</html>
```



Novos tipos de *input* HTML5

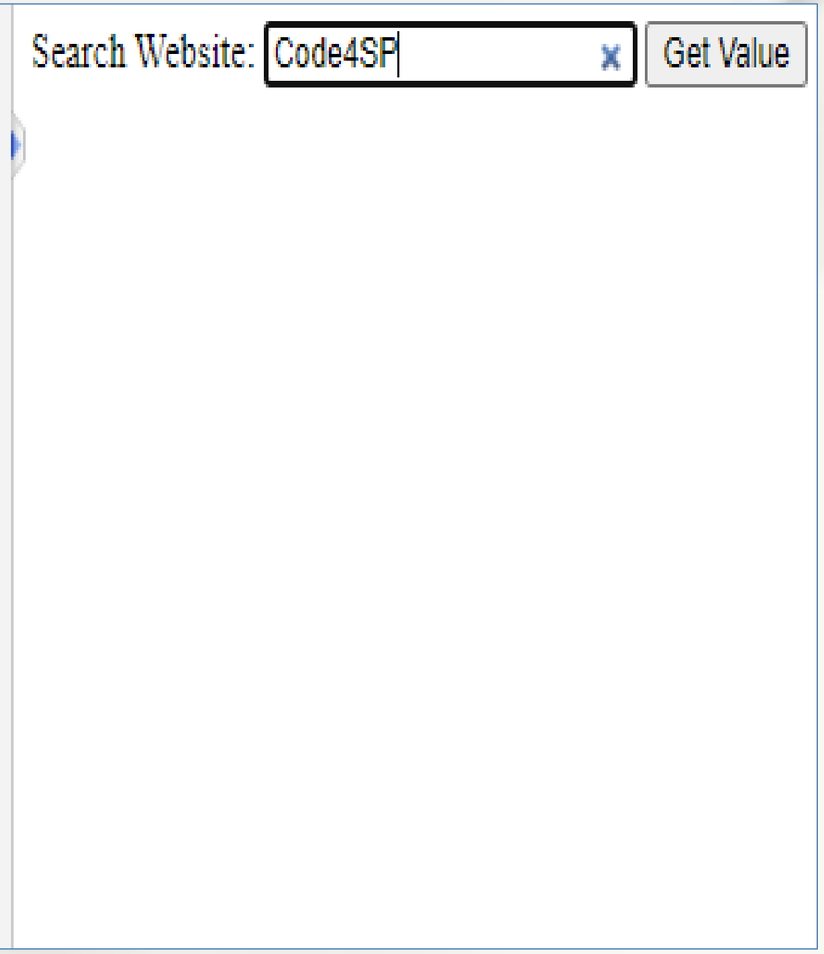
O *input* ‘search’

É adequado para criar *input* de campos de pesquisa. Em alguns navegadores (ex., *Chrome* e *Safari*), assim que o utilizador começa a escrever na caixa de pesquisa, aparece uma pequena cruz do lado direito, que pode ser útil para apagar todo o campo de pesquisa. É suportado pelos principais navegadores.

Novos tipos de *input* HTML5

O *input* 'search'

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Search Input Type</title>
<script>
    function getValue() {
        var term = document.getElementById("mysearch").value;
        alert(term);
    }
</script>
</head>
<body>
    <form>
        <label for="mysearch">Search Website:</label>
        <input type="search" id="mysearch" placeholder="Type something...">
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
</body>
</html>
```



Novos tipos de *input* HTML5

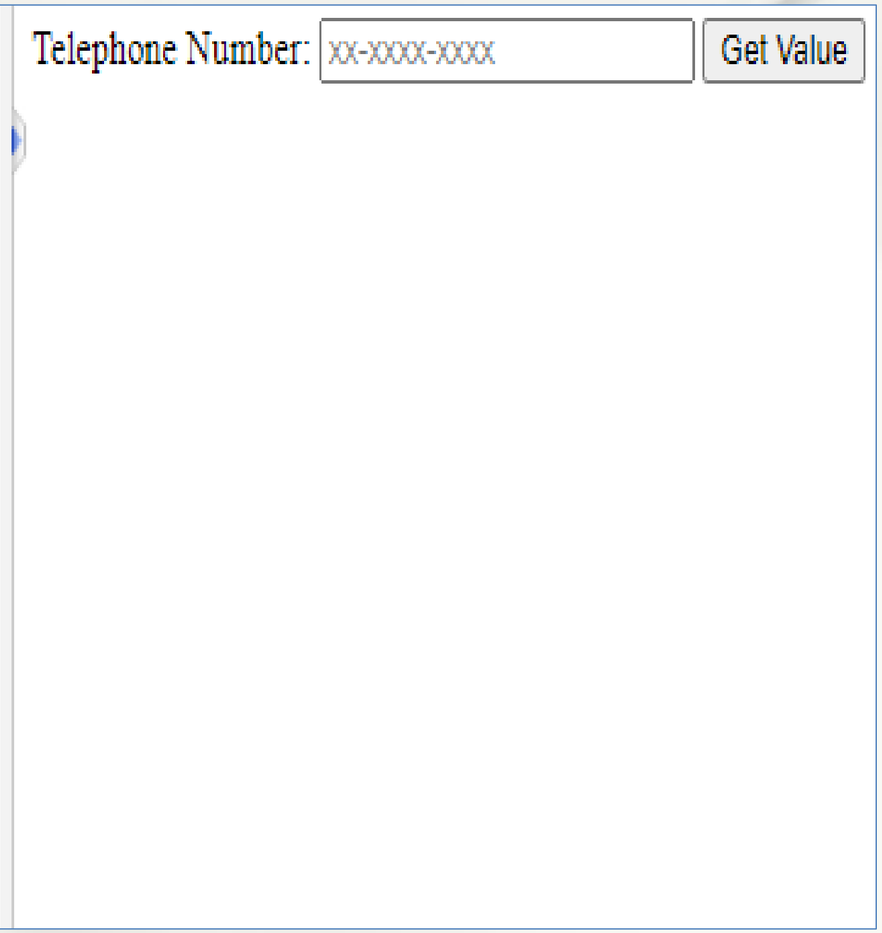
O *input* ‘tel’

É especialmente útil para a inserção de um número de telefone. Tendo em conta que os navegadores não suportam o *input* ‘tel’ por defeito, o atributo *placeholder* pode ser usado para ajudar os utilizadores a inserir o formato correto do seu número de telefone ou indicar uma expressão comum para validar o *input* do utilizador ao usar o atributo *pattern*. Não é suportado por nenhum navegador, já que os números de telefone variam de país para país.

Novos tipos de *input* HTML5

O *input* 'tel'

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Tel Input Type</title>
<script>
    function getValue() {
        var phone = document.getElementById("myphone").value;
        alert(phone);
    }
</script>
</head>
<body>
    <form>
        <label for="myphone">Telephone Number:</label>
        <input type="tel" id="myphone" placeholder="xx-xxxx-xxxx" required>
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
</body>
</html>
```



Novos tipos de *input* HTML5

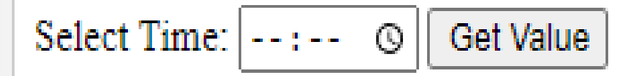
O *input* ‘time’

Este tipo de *input* pode ser usado para inserir qualquer hora (hora e minutos) e o navegador pode usar os dois formatos (12 ou 24 horas) para inserir a hora, dependendo da região. Não é suportado pelo *IE* nem pelo *Safari*.

Novos tipos de *input* HTML5

O *input* 'time'

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Time Input Type</title>
<script>
    function getValue() {
        var time = document.getElementById("mytime").value;
        alert(time);
    }
</script>
</head>
<body>
    <form>
        <label for="mytime">Select Time:</label>
        <input type="time" id="mytime">
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
</body>
</html>
```



Novos tipos de *input* HTML5

O *input* ‘url’

Este pode ser usado para inserir diferentes URL ou endereços *web.* O atributo *multiple* pode ser usado para que se possa inserir mais do que um URL. Se o atributo *required* for restrito, o navegador executará validação de imediato para assegurar que é inserido na caixa *input* apenas texto em formato URL. Os principais navegadores suportam este tipo de *input.*

Novos tipos de *input* HTML5

O *input* 'url'

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 URL Input Type</title>
<style>
    input[type="url"]:valid{
        outline: 2px solid green;
    }
    input[type="url"]:invalid{
        outline: 2px solid red;
    }
</style>
<script>
    function getValue() {
        var url = document.getElementById("myurl").value;
        alert(url);
    }
</script>
</head>
<body>
    <form>
        <label for="myurl">Enter Website URL:</label>
        <input type="url" id="myurl" required>
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
    <p><strong>Note</strong>: Enter URL in the form like https://www.google.com</p>
</body>
</html>
```

Enter Website URL: Get Value

Note: Enter URL in the form like https://www.google.com

O elemento Canvas da HTML5

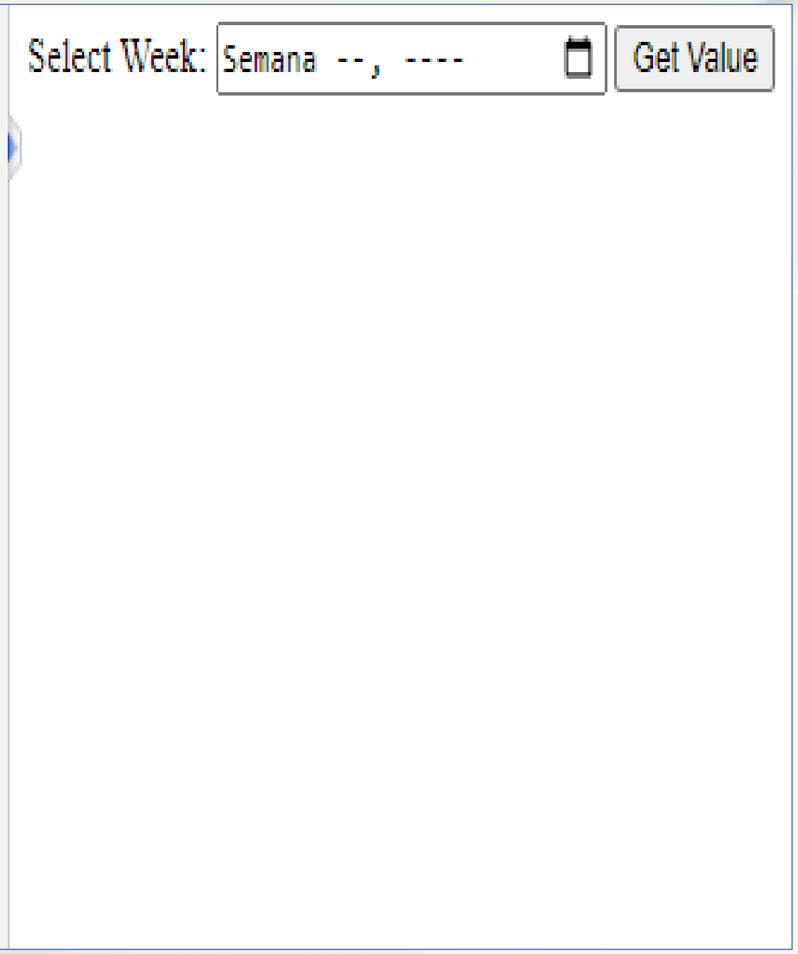
O *input* ‘week’

Finalmente, o *input week* permite ao utilizador escolher a semana e o ano a partir de um calendário *drop-down.* Não é suportado pelo *Firefox*, *Safari* nem *IE*, mas é suportado pelo *Chrome*, *Edge* e *Opera*.

Novos tipos de input HTML5

O *input* 'week'

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>HTML5 Week Input Type</title>
<script>
    function getValue() {
        var week = document.getElementById("myweek").value;
        alert(week);
    }
</script>
</head>
<body>
    <form>
        <label for="myweek">Select Week:</label>
        <input type="week" id="myweek">
        <button type="button" onclick="getValue();">Get Value</button>
    </form>
</body>
</html>
```



O elemento *Canvas* da HTML5

- Esta secção focar-se-á no uso do elemento *canvas* da HTML5. Este elemento foi originalmente criado pela *Apple* para os *widgets* do *dashboard* do *Mac OS* e para renderizar os gráficos no *Safari*. Foi ainda adotado por outros navegadores como o *Firefox*, *Google Chrome* e *Opera*.

- Por defeito, o elemento <canvas> tem 300px de largura e 150px de altura sem quaisquer delimitações ou conteúdo. Contudo, a largura e a altura podem ser personalizadas através do uso das funções *CSS height* e *CSS width*.

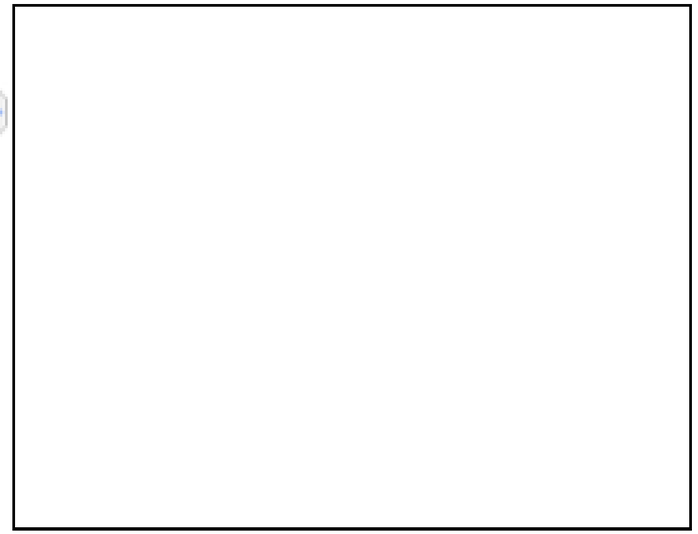
- O *canvas* pode ser criado em contexto bidimensional (2D) no qual tem quatro lados. As coordenadas do canto superior esquerdo são (0, 0) e identificam-se como o ponto de origem, as coordenadas do canto inferior direito são (*canvas width, canvas height*), tal como demonstrado aqui.

- Para desenhar linhas e formas básicas utilizando o elemento *canvas* da HTML5 e JavaScript, poderá ser útil ver alguns exemplos primeiro.

O elemento *Canvas* da HTML5

Primeiro, o modelo de como desenhar linhas e formas no contexto 2D:

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Embedding Canvas Into HTML Pages</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        // draw stuff here
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```



O elemento *Canvas* da HTML5

Todas as linhas, exceto da 7 à 11, são simples e fáceis de interpretar. A função *anonymous* afixada ao *window.onload* será executada quando a página carregar. Assim que a página carregar, é possível aceder ao elemento *canvas* usando o método *document.getElementById()*.

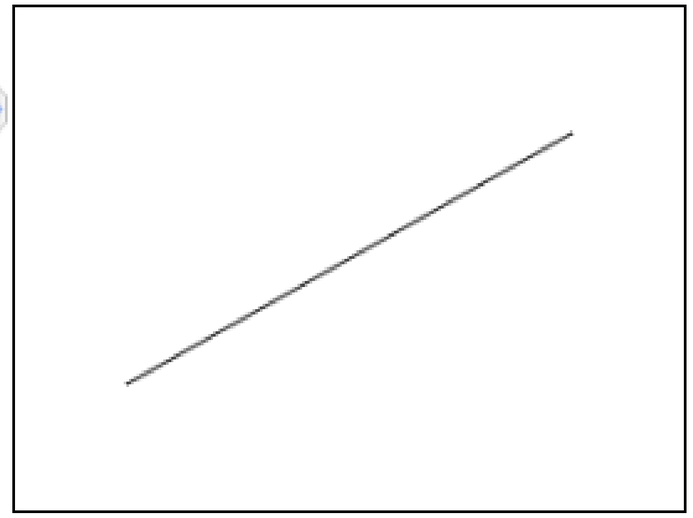
Depois, é definido o contexto 2D ao inserir ‘2d’ no método *getContext().*

O elemento *Canvas* da HTML5

O primeiro passo para desenhar no *canvas* é fazer uma **linha reta**. Os passos mais importantes para este propósito são *moveTo()*, *lineTo()* e *stroke()*. O método *moveTo()* identifica a posição do cursor de desenho no *canvas*, ao passo que o método *lineTo()* define as coordenadas do final da reta, e o método *stroke()* torna a linha visível:

O elemento *Canvas* da HTML5

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Drawing a Line on the Canvas</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.moveTo(50, 150);
        context.lineTo(250, 50);
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```



O elemento *Canvas* da HTML5

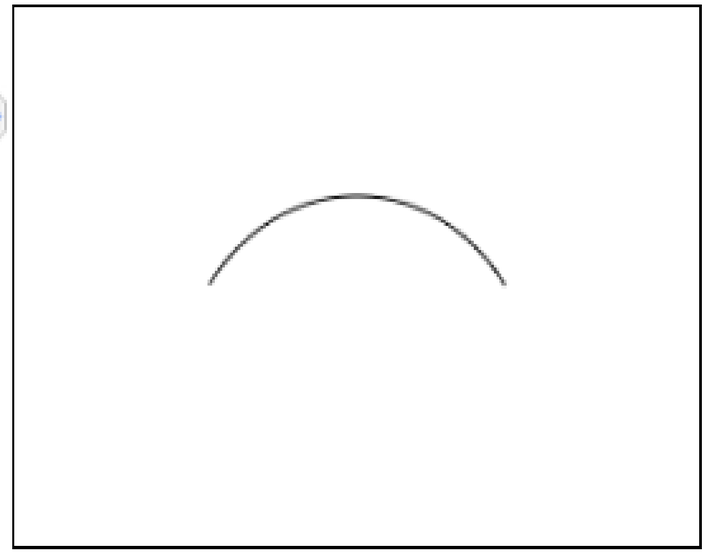
Como podemos **desenhar um arco**? Isto pode ser feito com o método, *arc()* cuja sintaxe é:

```
context.arc(centerX, centerY, radius, startingAngle,
            endingAngle, counterclockwise);
```

No próximo diapositivo está um exemplo de um arco desenhado no *canvas* através da inserção de um código *JavaScript*:

O elemento *Canvas* da HTML5

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Drawing an Arc on the Canvas</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.arc(150, 150, 80, 1.2 * Math.PI, 1.8 * Math.PI, false);
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```



O elemento *Canvas* da HTML5

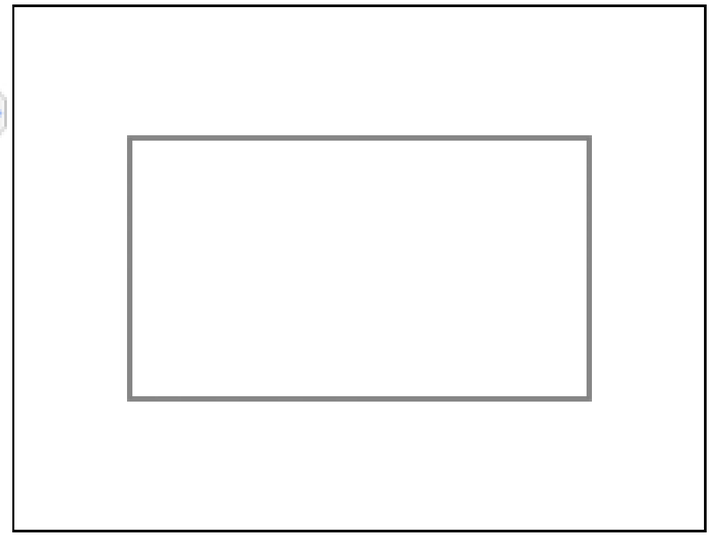
Para desenhar um **retângulo** e um **quadrado** devemos recorrer ao método *rect()*. Este tem quatro parâmetros: as coordenadas do retângulo dadas por x, y, a sua largura e a altura. A sintaxe do método *rect()* é a seguinte:

```
context.rect(x, y, width, height);
```

Segue um exemplo de como desenhar estas formas usando um código JS:

O elemento *Canvas* da HTML5

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Drawing a Rectangle on the Canvas</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.rect(50, 50, 200, 100);
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```



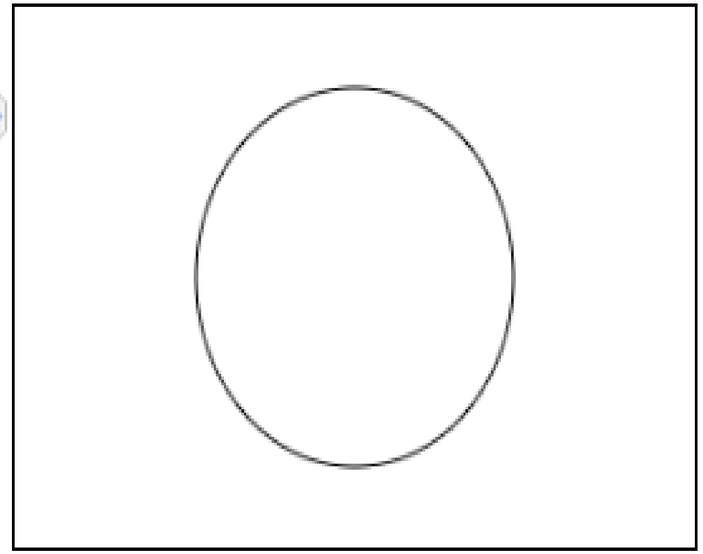
O elemento *Canvas* da HTML5

Contrariamente ao método *rect()*, não existe um procedimento específico para desenhar um **círculo**. Não obstante, esta forma pode ser desenhada criando um arco totalmente fechado através do método *arc()*. A sintaxe para desenhar um círculo através do método *arc()* é a seguinte:

```
context.arc(centerX, centerY, radius, 0, 2 * Math.PI,
                                                false);
```

O elemento *Canvas* da HTML5

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Drawing a Circle on the Canvas</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.arc(150, 100, 70, 0, 2 * Math.PI, false);
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

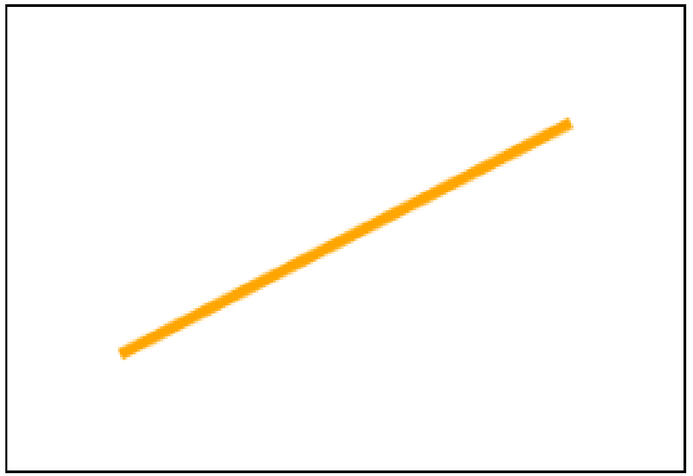


O elemento *Canvas* da HTML5

No que diz respeito aos **estilos e cores dos traços**, a cor predefinida é preto e a espessura é de 1 píxel. Contudo, estes atributos podem ser alterados através das propriedades *strokeStyle* e *lineWidth*, tal como exemplificado no próximo diapositivo:

O elemento *Canvas* da HTML5

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Setting Stroke Color and Width on the Canvas</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.lineWidth = 5;
        context.strokeStyle = "orange";
        context.moveTo(50, 150);
        context.lineTo(250, 50);
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```



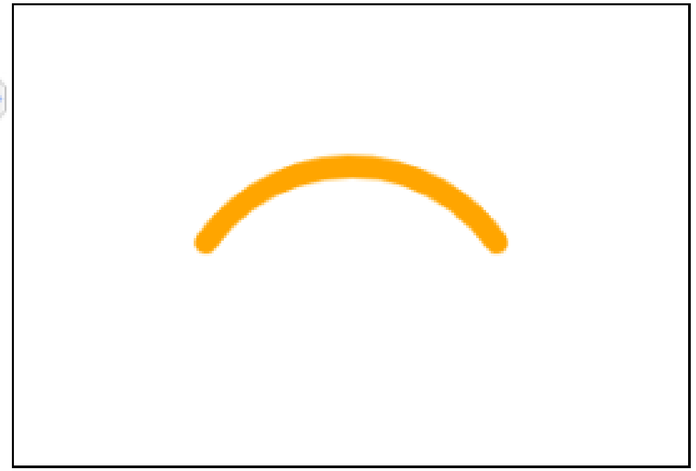
O elemento *Canvas* da HTML5

Também é possível definir o estilo dos fins das linhas através da propriedade *lineCap*, que tem três estilos: *butt*, *round*, e *square*.

Serve de exemplo o próximo diapositivo:

O elemento *Canvas* da HTML5

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Setting Stroke Cap Style on the Canvas</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.lineWidth = 10;
        context.strokeStyle = "orange";
        context.lineCap = "round";
        context.arc(150, 150, 80, 1.2 * Math.PI, 1.8 * Math.PI, false);
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```

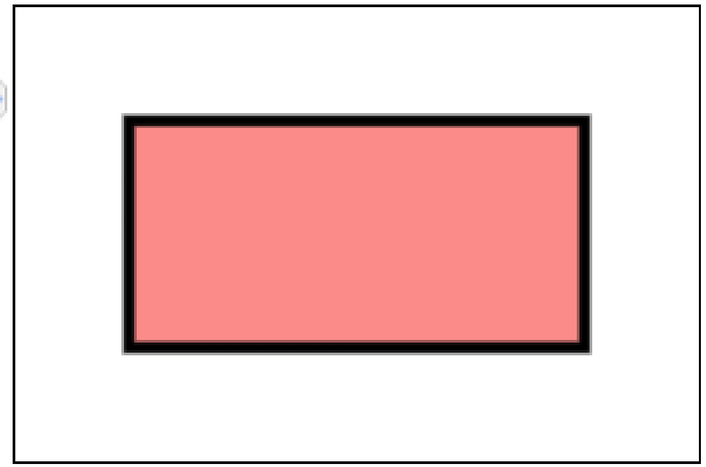


O elemento *Canvas* da HTML5

É também possível preencher as formas com cor usando o método *fillStyle()*. A imagem do próximo diapositivo mostra como preencher uma forma retangular com uma cor uniforme. De forma a renderizar o traço corretamente, aconselha-se que o método *fill()* seja inserido antes de *stroke()*.

O elemento *Canvas* da HTML5

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Filling Color inside a Rectangle on the Canvas</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.rect(50, 50, 200, 100);
        context.fillStyle = "#FB8B89";
        context.fill();
        context.lineWidth = 5;
        context.strokeStyle = "black";
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```



O elemento *Canvas* da HTML5

Há ainda a opção de preencher as formas com um degradê de cores.

Existem dois tipos de degradê: linear e radial.

A sintaxe para criar um **degradê linear** é:

```
var grd = context.createLinearGradient(startX, startY, endX,
                                       endY);
```

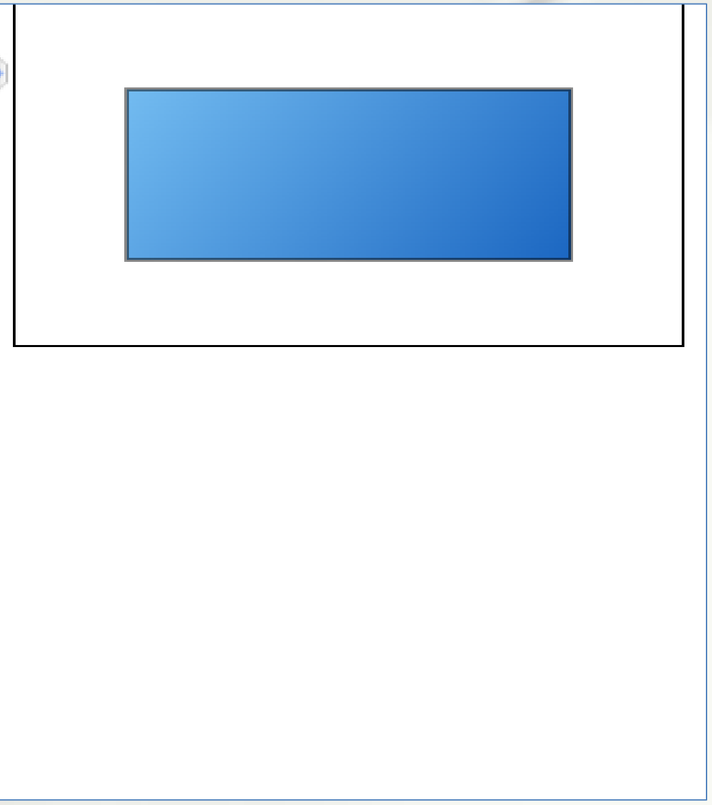
A sintaxe para criar um **degradê radial** é:

```
var grd = context.createRadialGradient(startX, startY,
                  startRadius, endX, endY, endRadius);
```

O elemento *Canvas* da HTML5

A imagem em baixo mostra como preencher um retângulo com um degradê linear através do método *createLinearGradient()*:

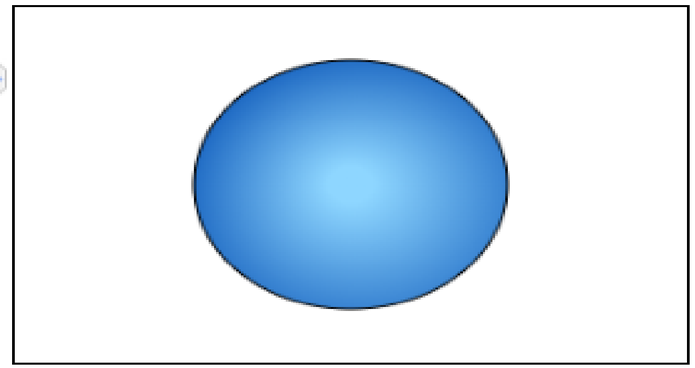
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Filling Linear Gradient inside Canvas Shapes</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.rect(50, 50, 200, 100);
        var grd = context.createLinearGradient(0, 0, canvas.width, canvas.height);
        grd.addColorStop(0, '#8ED6FF');
        grd.addColorStop(1, '#004CB3');
        context.fillStyle = grd;
        context.fill();
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```



O elemento *Canvas* da HTML5

A imagem em baixo mostra como preencher um círculo com um degradê radial através do método *createRadialGradient()* :

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Filling Radial Gradient inside Canvas Shapes</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.arc(150, 100, 70, 0, 2 * Math.PI, false);
        var grd = context.createRadialGradient(150, 100, 10, 160, 110, 100);
        grd.addColorStop(0, '#8ED6FF');
        grd.addColorStop(1, '#004CB3');
        context.fillStyle = grd;
        context.fill();
        context.stroke();
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```



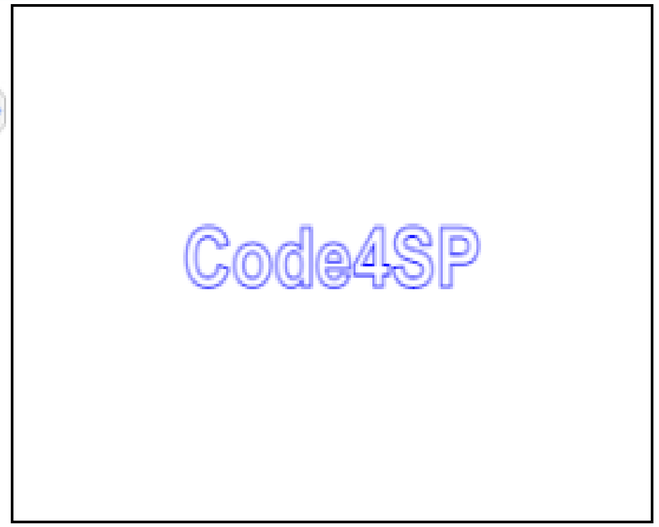
O elemento *Canvas* da HTML5

É ainda possível **desenhar texto** no *canvas* (apenas com carateres *Unicode*), adicionar **cor** e **alinhamento** e **desenhar texto com linhas e traços** usando o método *strokeText()*, que permite colorir os contornos do texto em vez de o preencher.

Contudo, se o *web designer* preferir pode também preencher o texto com cor através dos métodos *fillText()* e *strokeText()*. Aconselha-se que o método *fillText()* anteceda o *strokeText()* .

O elemento *Canvas* da HTML5

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Adding Stroke to Canvas Text</title>
<style>
    canvas {
        border: 1px solid #000;
    }
</style>
<script>
    window.onload = function() {
        var canvas = document.getElementById("myCanvas");
        var context = canvas.getContext("2d");
        context.font = "bold 32px Arial";
        context.textAlign = "center";
        context.textBaseline = "middle";
        context.strokeStyle = "blue";
        context.strokeText("Code4SP", 150, 100);
    };
</script>
</head>
<body>
    <canvas id="myCanvas" width="300" height="200"></canvas>
</body>
</html>
```



O elemento SVG da HTML5

- Aprenderemos agora como usar o elemento SVG para desenhar gráficos vetoriais numa página *web*. Para tal, comecemos por definir **SVG**:
- **SVG** significa *Scalable Vector Graphics*, que pode ser literalmente traduzido como Gráficos Vetoriais Escaláveis, este consiste numa imagem em vetor que tem por base o formato XML e é usado para criar gráficos vetoriais bidimensionais para a *web*. Neste formato, o tamanho pode ser escalado sem distorcer por se tratar de uma imagem em vetor, não afetando assim a qualidade da imagem. As imagens vetoriais são compostas por uma série de formas definidas por matemática, enquanto que imagens *raster* são formadas por pontos fixos (pixéis). Esta é a principal característica que distingue este formato da imagem *raster* (e.g. .jpg, .gif, .png, e outros formatos bidimensionais).
- **SVG** é também uma *W3C Recommendation*

O elemento SVG da HTML5

Uma imagem SVG é feita a partir de uma sequência de demonstrações de acordo com o esquema XML, assim, as **imagens SVG** podem ser criadas e editadas com um editor de texto como o *Notepad.* São várias as vantagens de usar imagens SVG em vez de outros formatos:

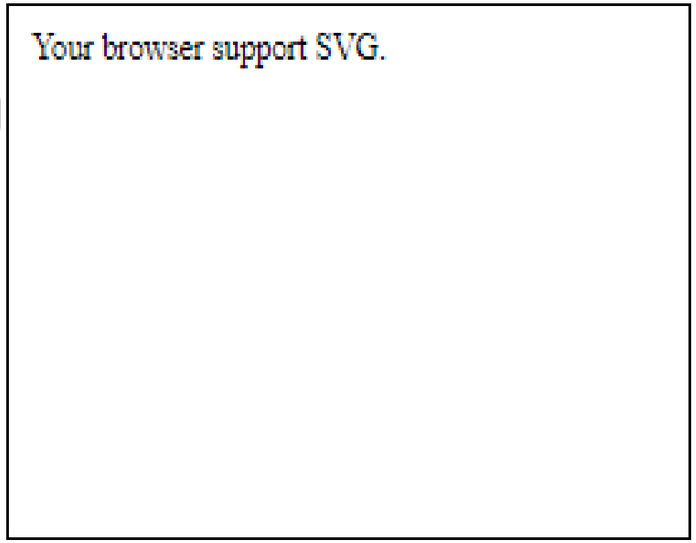
- Podem ser procuradas, indexadas, programadas e comprimidas;
- Podem ser criadas e modificadas em tempo real usando o JavaScript;
- Podem ser impressas com alta qualidade em qualquer resolução;
- Podem ser animadas através de elementos de integração de animação;
- Podem conter *hyperlinks* para outros documentos.

Os principais navegadores são compatíveis com a renderização de SVG (*Chrome*, *Firefox*, *Safari*, and *Opera*), bem como o *Internet Explorer* 9 e atualizações seguintes.

O elemento SVG da HTML5

Os gráficos SVG podem ser diretamente integrados num documento usando o elemento HTML5 <svg>, como exemplificado em baixo:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Embedding SVG Into HTML Pages</title>
<style>
    svg {
        border: 1px solid black;
    }
</style>
</head>
<body>
    <svg width="300" height="200">
        <text x="10" y="20" style="font-size:14px;">
            Your browser support SVG.
        </text>
        Sorry, your browser does not support SVG.
    </svg>
</body>
</html>
```

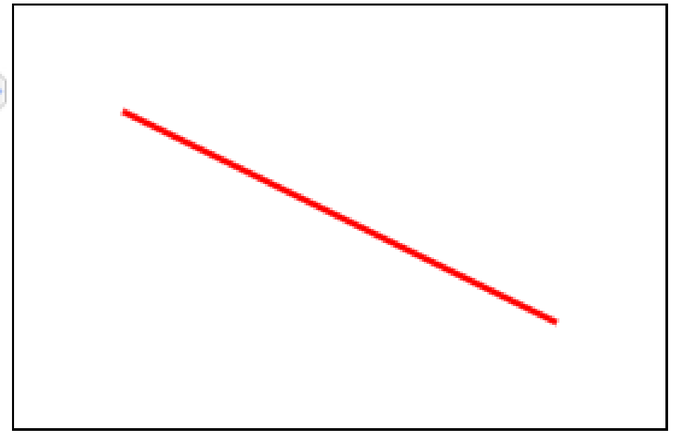


O elemento SVG da HTML5

Podem ser desenhadas linhas e formas vetoriais simples nas páginas *web* utilizando o elemento HTML5 <svg>.

A forma mais simples de trabalhar com o SVG é **desenhar uma linha reta**. Para tal, deve ser usado o elemento SVG <line>. Como demonstrado na figura, os atributos ‘x1’, ‘x2’, ‘y1’ e ‘y2’ do elemento <line> desenham uma linha de (x1,y1) a (x2,y2).

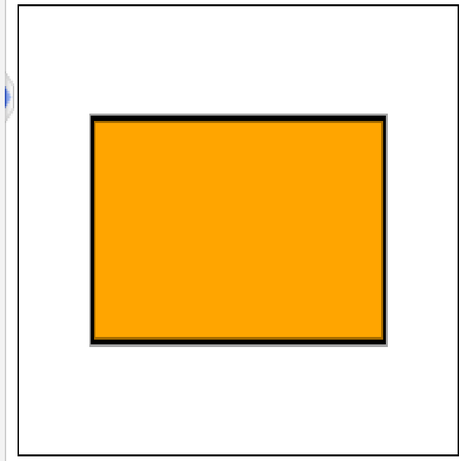
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Create a Line with HTML5 SVG</title>
<style>
    svg {
        border: 1px solid black;
    }
</style>
</head>
<body>
    <svg width="300" height="200">
        <line x1="50" y1="50" x2="250" y2="150" style="stroke:red; stroke-width:3;" />
    </svg>
</body>
</html>
```



O elemento SVG da HTML5

A forma mais apropriada para desenhar **retângulos** e **quadrados** é com o elemento SVG <rect>. Os atributos ‘x’ e ‘y’ do elemento <rect> especificam as coordenadas do canto superior esquerdo do retângulo. Os atributos *width* e *height* especifiam a largura e a altura.

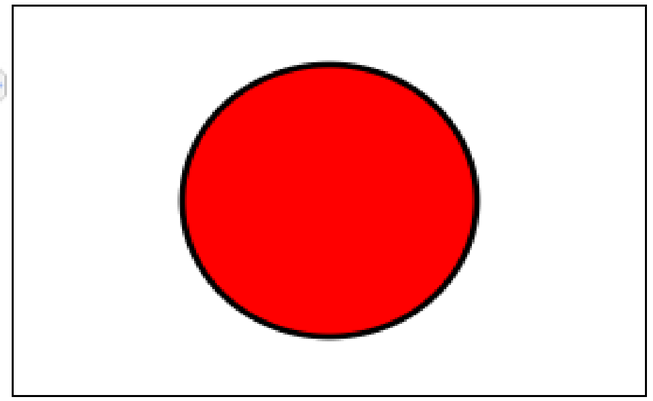
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Create a Rectangle with HTML5 SVG</title>
<style>
    svg {
        border: 1px solid black;
    }
</style>
</head>
<body>
    <svg width="300" height="200">
        <rect x="50" y="50" width="200" height="100" style="fill:orange; stroke:black;
stroke-width:3;" />
    </svg>
</body>
</html>
```



O elemento SVG da HTML5

Para desenhar um **círculo** devemos usar o elemento SVG <circle>. Os atributos ‘cx’ e ‘cy’ do elemento <circle> definem as coordenadas do centro do círculo e o atributo ‘r’ identifica o raio. Se os atributos ‘cx’ e ‘cy’ estiverem ausentes ou não forem especificados, o centro do círculo será (0,0).

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Create a Circle with HTML5 SVG</title>
<style>
    svg {
        border: 1px solid black;
    }
</style>
</head>
<body>
    <svg width="300" height="200">
        <circle cx="150" cy="100" r="70" style="fill:red; stroke:black; stroke-width:3;"
/>
    </svg>
</body>
</html>
```



O elemento SVG da HTML5

É ainda possível **desenhar texto** com o SVG. O texto em SVG é renderizado como um gráfico para que o *web designer* possa editá-lo graficamente com as ferramentas possíveis, mas é exibido como texto, podendo ser selecionado e copiado como texto pelo utilizador. Os atributos ‘x’ e ‘y’ do elemento identificam a localização do canto superior esquerdo em termos absolutos, apesar dos atributos ‘dx’ e ‘dy’ indicarem a localização relativa.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Render Text with HTML5 SVG</title>
<style>
    svg {
        border: 1px solid black;
    }
</style>
</head>
<body>
    <svg width="300" height="200">
        <text x="20" y="30" style="fill:purple; font-size:22px;">
            Welcome to Our Website!
        </text>
        <text x="20" y="30" dx="0" dy="20" style="fill:navy; font-size:14px;">
            Here you will find lots of useful information.
        </text>
    </svg>
</body>
</html>
```

O elemento SVG da HTML5

Em alternativa, os *web designers* podem usar o elemento <tspan> para redimensionar e mover o corpo do texto. O texto é inserido em *tspans* separados, mas o texto que está no elemento pode ser selecionado na sua totalidade de uma vez só – basta clicar e arrastar para selecionar o texto. Contudo, o texto em elementos separados não pode ser selecionado de só uma vez.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Rotate and Render Text with HTML5 SVG</title>
<style>
    svg {
        border: 1px solid black;
    }
</style>
</head>
<body>
    <svg width="300" height="200">
        <text x="30" y="15" style="fill:purple; font-size:22px; transform:rotate(30deg);">
            <tspan style="fill:purple; font-size:22px;">
                Welcome to Our Website!
            </tspan>
            <tspan dx="-230" dy="20" style="fill:navy; font-size:14px;">
                Here you will find lots of useful information.
            </tspan>
        </text>
    </svg>
</body>
</html>
```

O elemento SVG da HTML5

Ainda que ambos os elementos gráficos tenham sido introduzidos para criar gráficos de alta qualidade para a web, estes elementos diferem.

Para ajudar os formandos a compreender melhor como usar estes elementos de forma apropriada, seguem as principais diferenças entre SVG e *Canvas*:

SVG	Canvas
Baseado em vetores (composto por formas)	**Baseado em *raster* (composto por pixéis)**
Múltiplos elementos gráficos, que se tornam parte da DOM da página	**Elemento único com comportamento semelhante a <img>. O diagrama do *Canvas* pode ser guardado em formato PNG ou JPG**
Modificado através de *script* e CSS	**Modificado apenas através de *script***
Boas capacidades de renderização de texto	**Fracas capacidades de renderização de texto**
Melhor desempenho com menor número de elementos ou superfícies maiores, ou ambos	**Melhor desempenho com maior número de elementos ou superfícies mais pequenas, ou ambos**
Melhor capacidade para redimensionar. Pode ser impresso com alta qualidade em qualquer resolução. Não ocorre desfoque/pixelização	**Fraca capacidade para redimensionar. Não adequado para impressões em grande resolução. Pode ocorrer desfoque/pixelização**

Áudio em HTML5

- Esta secção tenciona explicar como integrar áudio num documento HTML.

- Sendo que os navegadores da *web* não tinham um padrão uniforme para a integração de ficheiros áudio, esta era uma tarefa difícil no passado. Contudo, existem várias formas de integrar som numa página *web*, desde usar um simples *link*, a usar o elemento HTML5 <audio>. Este elemento faculta uma forma padrão de inserir áudio em páginas *web*. Considerando que o elemento *Audio* é relativamente recente, este é suportado por quase todos os navegadores.

- Há diferentes formas de inserir áudio num documento HTML5. Uma delas é usando os controlos predefinidos do navegador com uma fonte definida pelo atributo *src* , como pode ser verificado neste código.

Áudio em HTML5

- Outra maneira de o fazer pode ser usando o elemento <object>, que é utilizado para integrar diferentes tipos de ficheiros multimédia. Este exemplo integra um ficheiro áudio numa página *web* através do método mencionado anteriormente. Note-se que o elemento não é amplamente suportado e depende do tipo de elemento que está a ser implementado. Em muitos casos, outros métodos como o elemento HTML5 <audio> ou leitores de áudio de terceiros, podem ser uma melhor opção.

- Finalmente, o elemento <embed> pode ser uma outra forma de integrar multimédia num documento HTML, seguindo este exemplo. Apesar do elemento <embed> ser suportado de forma excelente pelos navegadores atuais e estar definido como o padrão em HTML5, o áudio inserido pode não ser reproduzido devido à incapacidade do navegador de suportar o áudio ou inacessibilidade de *plugins*.

Vídeo em HTML5

- Chegou a hora de aprender a inserir ficheiros de vídeo num documento HTML.

- Em semelhança aos ficheiros áudio, os ficheiros de vídeo eram igualmente difíceis de inserir numa página *web* pelas mesmas razões (os navegadores *web* não tinham um padrão uniforme para definir a integração de ficheiros multimédia como o vídeo). Nos próximos diapositivos, serão explicadas diferentes formas de inserir este tipo de conteúdo.

Vídeo em HTML5

O novo elemento <video> funciona em grande parte dos navegadores atuais. Este exemplo explica como inserir um vídeo num documento HTML, usando os controlos predefinidos do navegador com uma fonte definida pelo atributo *src*.

O elemento <object> também é usado para integrar diferentes ficheiros multimédia. Seguindo este exemplo, é possível entender como inserir um vídeo *Flash* numa página *web* (o ficheiro será apenas reproduzido por navegadores/aplicações capazes de suportar *Flash*). Note-se que o elemento <object> não é amplamente suportado e depende muito do tipo de elemento integrado. Em muitos casos, outros métodos poderão ser melhores opções, por exemplo, os dispositivos iPad e iPhone não são capazes de reproduzir vídeos *Flash*.

Vídeo em HTML5

Como inserir **vídeos do YouTube**? Esta é a forma mais simples e comum de inserir vídeos em páginas *web* hoje em dia. O *web designer* só necessita de fazer o *upload* do vídeo para o YouTube e inserir o código HTML para exibir o vídeo na sua página. Fica aqui apresentado um pequeno guia, passo a passo:

1º passo – Fazer o *upload* do vídeo para o YouTube.

2º passo – Depois de fazer o *upload* do vídeo, o *web designer* deve procurar a opção 'Share', que normalmente está por baixo do vídeo, como demonstrado em baixo:

Vídeo em HTML5

Ao clicar em ‘Share’, aparecerá um painel onde aparecerão algumas opções. Agora, deverá clicar na opção ‘**Embed**’, e será gerado um código HTML para inserir diretamente o vídeo na página. Para tal, o *web designer* deverá copiar e colar o código no documento HTML.

Vídeo em HTML5

Este código pode ser editado, para tal o *web designer* necessita apenas de selecionar a opção de edição dada por baixo da caixa onde é inserido o código.

A integração do vídeo de YouTube na página está explicada neste exemplo.

Armazenamento *Web* em HTML5

Alguma vez se perguntou como usar o armazenamento *Web* em HTML5 para armazenar dados no navegador do utilizador? Os próximos diapositivos irão explicar como o fazer.

Antes de mais, é importante perceber as implicações de ‘web storage’ (armazenamento *Web*).

Armazenamento *Web* em HTML5

Através do armazenamento *web*, as aplicações *web* podem armazenar dados localmente no navegador do utilizador. Antes do HTML5, os dados tinham de ser guardados em *cookies*, incorporados em todos os pedidos do servidor. O armazenamento *web* é mais seguro e uma quantidade substancial de dados pode ser armazenada localmente, sem afetar o desempenho da página *web*.

A informação contida neste tipo de armazenamento não é enviada para o servidor, ao contrário do que acontece com os *cookies*, em que os dados são apresentados ao servidor com todos os pedidos. Em acréscimo, os *cookies* permitem apenas armazenar uma pequena quantidade de dados (aproximadamente 4KB), enquanto o armazenamento *web* permite armazenar até 5MB.

Armazenamento *Web* em HTML5

Existem dois tipos de armazenamento *web*:

Local storage — usa o objeto *localStorage* para armazenar dados referentes a toda a página *web* de forma permanente. Disto isto, os dados armazenados localmente estarão disponíveis no dia, semana ou ano seguintes, a não ser que sejam removidos.

Session storage — usa o objeto *sessionStorage* para armazenar dados referentes a um único separador ou janela do navegador temporariamente. Os dados são eliminados quando terminada a sessão, por exemplo, quando o utilizador fecha a janela ou separador.

Armazenamento *Web* em HTML5

No que diz respeito à ***local storage***, cada parcela de dados é recolhida num par *key/value*. *Key* identifica o nome da informação (ex., ‘nome próprio’), e *value* é a informação inserida relacionada com a mesma *key* (ex., ‘Pedro’). Este código JS expressa o seguinte:

- localStorage.setItem (*key*, *value*) armazena o *value* associado a um *key*
- localStorage.getItem (*key*) guarda o *value* associado a um *key*

Também é possível remover um item específico do armazenamento ao passar o nome do *key* para o método *removeItem()*, ex. localStorage.removeItem("first_name").

Armazenamento *Web* em HTML5

Contudo, se for intenção do *web designer* remover todo os dados armazenados, ele(a) deve usar o método *clear(),* ex. *localStorage.clear().* Este método elimina de uma só vez todos os pares *key/value* do *localStorage*, por isso **deve ser usado com cuidado**. Os dados do armazenamento *web* não poderão ser acedidos em navegadores diferentes.

Finalmente, o objeto *sessionStorage* é muito semelhante ao *localStorage*, com a diferença de que armazena os dados para apenas uma sessão. Este exemplo serve de explicação.

Application Cache em HTML5

Esta secção será sobre a criação de aplicações *offline* com o elemento ***caching*** da HTML5.

É sabido que a maioria das aplicações *web* não funcionam se o *web designer* estiver *offline*. No entanto, a HTML5 trouxe um mecanismo de *Application Cache* que permite ao navegador guardar automaticamente o ficheiro HTML, e outros recursos, para os exibir adequadamente na máquina local. Assim, o navegador pode aceder à página *web* e aos seus recursos sem estabelecer ligação à *Internet*.

É suportado pelos principais navegadores (*Firefox*, *Chrome*, *Opera*, *Safari*, e *Internet Explorer 10* e posteriores.

Application Cache em HTML5

Há várias vantagens em usar esta função:

- **Navegação *Offline*** — Os utilizadores podem usar a aplicação mesmo não estando *online* ou quando ocorrem interrupções inesperadas da ligação à rede;
- **Melhor desempenho** — Os recursos de *Cache* carregam diretamente da máquina do utilizador do servidor remoto para que a página carregue mais rápido e tenha melhor desempenho;
- **Reduzir solicitação HTTP e carga do servidor** — O navegador terá apenas de fazer o *download* de recursos atualizados/alterados do servidor remoto para reduzir os pedidos HTTP e poupar banda larga, assim como diminuir a carga do servidor *web*.

Application Cache em HTML5

Os passos para armazenar ficheiros em *cache* de forma a usá-los em modo *offline* são os seguintes:

1º Passo – Criar um arquivo *Cache Manifest*. Este é um arquivo especial de texto que informa os navegadores que ficheiro devem armazenar (e não armazenar), e que ficheiros substituir. Inicia-se sempre com as palavras CACHE MANIFEST (sempre em maiúsculas).

code4sp
coding for social promotion

Application Cache em HTML5

Example

Download

```
CACHE MANIFEST
# v1.0 : 10-08-2014

CACHE:
# pages
index.html

# styles & scripts
css/theme.css
js/jquery.min.js
js/default.js

# images
/favicon.ico
images/logo.png

NETWORK:
login.php

FALLBACK:
/ /offline.html
```

Application Cache em HTML5

Vamos agora explicar a codificação da imagem do último diapositivo.

- Primeiro, é importante entender que os arquivos *Manifest* podem ter três secções diferentes: **CACHE, NETWORK**, e **FALLBACK**.
- Os ficheiros listados na secção CACHE: (ou logo depois da linha *CACHE MANIFEST*) são armazenados em *cache* depois de ter sido feito o seu *download* pela primeira vez;
- Os ficheiros listado na secção NETWORK: são recursos *white-listed* que nunca são armazenados em *cache* e estão apenas disponíveis *online*. O que significa que os utilizadores nunca poderão aceder à página *login.php offline*;
- A secção FALLBACK: especifica páginas alternativas que o navegador deve usar caso a ligação ao servidor não possa ser estabelecida. Cada entrada nesta secção lista dois URLs — o primeiro é o recurso primário, o segundo é o recurso secundário (*fallback*). Por exemplo, no caso da figura 98, a página *offline.html* será exibida se o utilizador estiver *offline*. Além disso, ambos os URLs devem ter a mesma origem que o arquivo *Manifest*
- Note-se que as linhas que começam por '#' (asterisco) são comentários.

Application Cache em HTML5

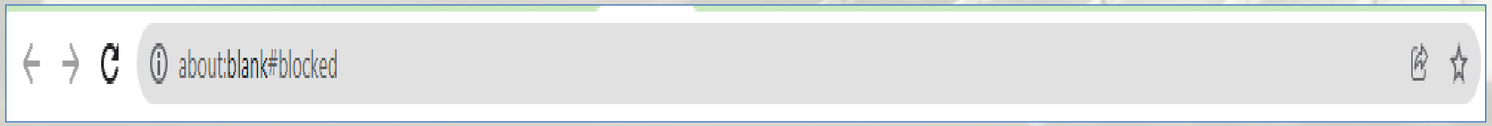
2º Passo – Usar o arquivo *Cache Manifest*. Depois de o criar, o *web designer* deve fazer o *upload* do arquivo *Cache Manifest* para o servidor *web* – certificando-se de que o servidor está configurado para servir os arquivos *Manifest* com o tipo MIME text/cache-manifest.

Para fazer o arquivo funcionar, o *web designer* irá precisar de permitir o seu uso nas páginas *web*, para tal basta acrescentar o atributo *manifest* ao elemento *root* <html>, como demonstrado na imagem do diapositivo seguinte.

Application Cache em HTML5

```html
<!DOCTYPE html>
<html lang="en" manifest="example.appcache">
<head>
    <title>Using the Application Cache</title>
</head>
<body>
    <!--The document content will be inserted here-->
</body>
</html>
```

Se o utilizador estiver *online*, o resultado para este código será o seguinte:



Web Workers em HTML5

Esta secção será sobre tópicos JS, já que ensinará como usar HTML5 *Web Workers* para executar Código JS em segundo plano. Se experienciarem alguma dificuldade, os formandos devem expor a situação.

Se tentarmos executar de forma intensiva cálculos pesados e morosos que exijam tarefas com JavaScript, provavelmente, as páginas web irão congelar impossibilitando os utilizadores de fazer o que quer que seja até o trabalho estar terminado. Porquê? Porque o Código JS é sempre executado em segundo plano. Contudo, a HTML5 tem uma nova tecnologia (‘web worker’) criada para desempenhar tarefas em segundo plano para além de outros *user-interface scripts*, sem afetar o desempenho da página. Ao contrário do que acontece com as operações JS normais, o *web worker* não interrompe o utilizador e a página mantém-se responsiva porque as tarefas estão a ser executadas em segundo plano.

Web Workers em HTML5

Os *web workers* são especialmente úteis para executar tarefas morosas. Assim, no primeiro exemplo, uma simples tarefa JS que conta de 0 a 100 000 será criada (o nome do ficheiro deve ser worker.js) da seguinte forma:

```
1  var i = 0;
2  function countNumbers() {
3      if(i < 100000) {
4          i = i + 1;
5          postMessage(i);
6      }
7
8      // Wait for sometime before running this script again
9      setTimeout("countNumbers()", 500);
10 }
11 countNumbers();
```

Web Workers em HTML5

Agora que o arquivo *web worker* foi criado, é altura de colocar o *web worker* de um documento HTML a executar o código dentro do ficheiro com o nome "worker.js" em segundo plano, os resultados aparecerão gradualmente na página. Note-se que o número à direita crescerá sempre até alcançar 100,000.

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Using HTML5 Web Workers</title>
<script>
    if(window.Worker) {
        // Create a new web worker
        var worker = new Worker("/examples/js/worker.js");

        // Fire onMessage event handler
        worker.onmessage = function(event) {
            document.getElementById("result").innerHTML = event.data;
        };
    } else {
        alert("Sorry, your browser do not support web worker.");
    }
</script>
</head>
<body>
    <div id="result">
        <!--Received messages will be inserted here-->
    </div>
</body>
</html>
```

26

Web Workers em HTML5

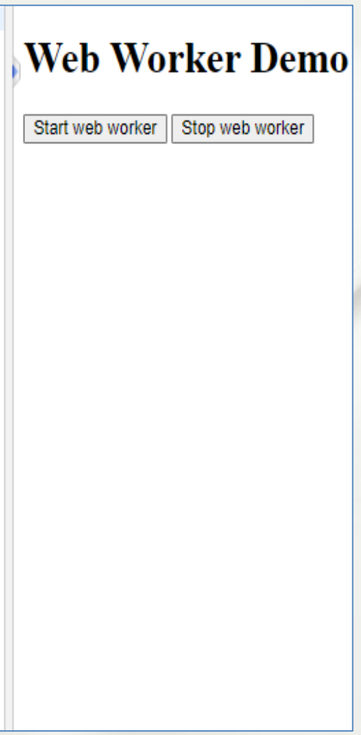
- Explicando agora o exemplo anterior, a declaração ***var worker = new Worker("worker.js");*** cria um novo objeto *web worker*. Quando o *worker* publica uma mensagem, desencadeia a resposta ***onmessage*** (linha 14), que permite ao código receber mensagens do *web worker*.

- O elemento ***event.data*** constitui a mensagem enviada pelo *web worker*. Oficialmente, o código executado pelo *worker* é sempre armazenado num arquivo JAvaScript à parte para prevenir o programador da *web* de escrever código *web worker* que tenta usar variantes globais ou abrir os elementos na página *web* diretamente.

Web Workers em HTML5

Também é possível parar um ***worker* em execução** a meio de uma operação, seguindo este exemplo:

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Start/Stop Web Worker in HTML5</title>
<script>
    // Set up global variable
    var worker;

    function startWorker() {
        // Initialize web worker
        worker = new Worker("/examples/js/worker.js");

        // Run update function, when we get a message from worker
        worker.onmessage = update;

        // Tell worker to get started
        worker.postMessage("start");
    }

    function update(event) {
        // Update the page with current message from worker
        document.getElementById("result").innerHTML = event.data;
    }

    function stopWorker() {
        // Stop the worker
        worker.terminate();
    }
</script>
</head>
```



A função *server-sent events* da HTML5

Esta secção focar-se-á no uso da função *server-sent events* da HTML5 para estabelecer ligações permanentes e unidirecionais entre uma página *web* e o servidor.

A função *server-sent events* da HTML5

- Esta característica é uma forma inovadora das páginas *web* comunicarem com o servidor. Contudo, existem algumas circunstâncias nas quais as páginas necessitam de uma ligação de longa duração ao servidor, por exemplo, cotações de bolsa em páginas de finanças em que os preços se atualizam automaticamente ou game tickers(?) em execução em várias páginas de desporto

- Tais tarefas são alcançáveis com a função *server-sent events da HTML5*, já que esta permite a uma página manter uma ligação aberta a um servidor, de forma a que este possa enviar uma nova resposta mecânica a qualquer momento. Nesta fase, não é necessário restabelecer a ligação e executar novamente o mesmo *script*.

- Para um melhor entendimento do que foi mencionado em cima, crie um ficheiro PHP denominado "server_time.php" e escreva o seguinte *script*. Este ficheiro irá meramente reportar a hora atual do relógio do servidor em intervalos regulares:

A função *server-sent events* da HTML5

```php
<?php
header("Content-Type: text/event-stream");
header("Cache-Control: no-cache");

// Get the current time on server
$currentTime = date("h:i:s", time());

// Send it in a message
echo "data: " . $currentTime . "\n\n";
flush();
?>
```

NOTA: Um ficheiro PHP é um ficheiro de texto simples que contém código escrito em linguagem de programação PHP. Como a PHP é uma linguagem *server-side* (*back-end*), o código escrito é executado no servidor. De facto, o ficheiro PHP pode conter texto simples, etiquetas HTML, ou código de acordo com a sintaxe PHP. A PHP é normalmente usada para desenvolver aplicações *web* que são processadas por um mecanismo PHP no servidor.

A função *server-sent events* da HTML5

- As duas primeiras linhas do *script* de PHP estabelecem dois cabeçalhos cruciais. A primeira é o tipo de MIME de *text/event-stream*, que é necessário para um *server-side event* padrão. A segunda linha diz ao servidor para desligar o *caching* de forma a evitar que o resultado do *script* seja armazenado em *cache*.

- Todas as mensagens enviadas através de *server-sent events* da HTML5 devem começar por *data:* seguido da mensagem de texto e da sequência da nova linha de carateres *(\n\n).*

- Por fim, a função *flush()* da PHP foi usada para assegurar que os dados são enviados imediatamente em vez de serem enviados só quando o código PHP está completo.

- No que diz respeito ao **processamento de mensagens numa página *web***, o objeto *EventSource* é usado para receber mensagens *server-sent events*. No exemplo em baixo, os formandos verão como um documento HTML recebe a hora atual reportada pelo servidor e a exibe aos visitantes da página. Para um melhor entendimento, será criado um documento HTML denominado por “demo_sse.htlm”, que será colocado na mesma lista de projeto onde está o “server_time.php”.

A função *server-sent events* da HTML5

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>Using Server-Sent Events</title>
<script>
    window.onload = function() {
        var source = new EventSource("server_time.php");
        source.onmessage = function(event) {
            document.getElementById("result").innerHTML += "New time received
from web server: " + event.data + "<br>";
        };
    };
</script>
</head>
<body>
    <div id="result">
        <!--Server response will be inserted here-->
    </div>
</body>
</html>
```

A função *Geolocation da* HTML5

- Neste subtópico, os formandos terão uma perceção de como usar a função *Geolocation* da HTML5 para detetar a localização do utilizador. Esta função permite ao programador descobrir as coordenadas geográficas (latitude e longitude) da localização atual do visitante da página. É especialmente útil para facultar uma melhor experiência de navegação para o visitante, já que é capaz de mostrar os resultados de pesquisa próximos da localização física do utilizador.

- Receber informação sobre a localização do visitante usando a *API Geolocation* da HTML5 não é difícil. Esta explora os três métodos contidos no objeto *navigator.geolocation* — *getCurrentPosition()*, *watchPosition()* e *clearWatch()*.

- Depois de o utilizador autorizar partilhar a sua localização com o servidor (o navegador não partilhará a localização do utilizador a não ser que este autorize), a função *Geolocation* deve ocorrer como exemplificado na imagem que se segue.

A função *Geolocation da* HTML5

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Get Visitor's Location Using HTML5 Geolocation</title>
<script>
    function showPosition() {
        if(navigator.geolocation) {
            navigator.geolocation.getCurrentPosition(function(position) {
                var positionInfo = "Your current position is (" + "Latitude: " +
position.coords.latitude + ", " + "Longitude: " + position.coords.longitude + ")";
                document.getElementById("result").innerHTML = positionInfo;
            });
        } else {
            alert("Sorry, your browser does not support HTML5 geolocation.");
        }
    }
</script>
</head>
<body>
    <div id="result">
        <!--Position information will be inserted here-->
    </div>
    <button type="button" onclick="showPosition();">Show Position</button>
</body>
</html>
```

Your current position is (Latitude: 41.0079509, Longitude: -8.6270736)

Show Position

A função *Geolocation da* HTML5

- Caso o utilizador não queira partilhar a sua localização com o navegador, o programador pode colmatar com duas funções ao iniciar a função *getCurrentLocation()*. A primeira é usada caso a tentativa de usar a *Geolocation* seja bem sucedida, ao passo que a segunda é usada caso a tentativa falhe.

code4sp
coding for social promotion

A função *Geolocation da* HTML5

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Handling the Geolocation Errors and Rejections</title>
<script>
    // Set up global variable
    var result;

    function showPosition() {
        // Store the element where the page displays the result
        result = document.getElementById("result");

        // If geolocation is available, try to get the visitor's position
        if(navigator.geolocation) {
            navigator.geolocation.getCurrentPosition(successCallback, errorCallback);
            result.innerHTML = "Getting the position information...";
        } else {
            alert("Sorry, your browser does not support HTML5 geolocation.");
        }
    };

    // Define callback function for successful attempt
    function successCallback(position) {
        result.innerHTML = "Your current position is (" + "Latitude: " + position.coords.latitude + ", " + "Longitude: " + position.coords.longitude + ")";
    }

    // Define callback function for failed attempt
    function errorCallback(error) {
        if(error.code == 1) {
            result.innerHTML = "You've decided not to share your position, but it's OK. We won't ask you again.";
        } else if(error.code == 2) {
            result.innerHTML = "The network is down or the positioning service can't be reached.";
        } else if(error.code == 3) {
            result.innerHTML = "The attempt timed out before it could get the location data.";
        } else {
            result.innerHTML = "Geolocation failed due to unknown error.";
        }
    }
</script>
</head>
<body>
    <div id="result">
        <!--Position information will be inserted here-->
    </div>
    <button type="button" onclick="showPosition();">Show Position</button>
</body>
</html>
```

A função *Geolocation da* HTML5

- Há várias funções interessantes a explorar com os dados fornecidos pela *Geolocation*, como por exemplo, mostrar a localização do utilizador no *Google Maps*. Este exemplo mostra a localização atual do leitor obtida através da latitude e longitude recolhidas pela função *Geolocation* da HTML5. Já este exemplo mostra uma imagem estática do *Google Maps,* com a possibilidade de aproximar, afastar e arrastar, que demonstra a localização do utilizador.

- Os exemplos anteriormente mencionados baseiam-se no método *getCurrentPosition().* No entanto, a função *Geolocation* tem outra técnica – a *watchPosition(),* que permite seguir o movimento do utilizador ao devolver a localização atualizada à medida que esta vai mudando. A *watchPosition()* tem os mesmos parâmetros de *input* que *getCurrentPosition().* Ainda assim, a *watchPosition()* pode ativar a função *success* múltiplas vezes — quando recebe informação sobre a localização pela primeira vez e quando recebe atualizações de cada vez que deteta uma nova localização, como sugere o exemplo.

Drag e *Drop* em HTML5

- **Arrastar e largar** (*drag and drop*) um elemento para outra localização numa página *web* é um procedimento comum *online.*

- A função ***drag and drop*** da HTML5 permite fazer a ação descrita em cima com qualquer elemento.

- Este é um simples exemplo que familiarizará os formandos com o conceito. Apesar do código aparentar ser difícil de compreender, é bastante simples e lógico:

Drag e *Drop* em HTML5

- Para capacitar um elemento de ser arrastado, o atributo *draggable* deve ser ‘true’:

```
<img draggable="true">
```

- Em seguida, deve ser especificado o que é suposto acontecer assim que o elemento for arrastado. No exemplo em cima, o atributo *ondragstart* solicita a função (drag (event)), que especifíca que dados serão movidos. O processo *dataTransfer.setData()* estabelece o tipo de informação e valores dos dados movidos:

```
Functiondrag(ev){ev.dataTransfer.setData("text",ev.target.id);}
```

- Neste exemplo, o tipo de informação é “text”, e o valor a *id* do elemento a arrastar, (“drag1”).

Drag e *Drop* em HTML5

- O evento *ondragover* estipula onde pode ser largado o elemento movido. Por defeito, *data/elements* não podem ser largados em outros elementos. De forma a permitir largar o elemento, o *web designer* deve prevenir o processamento predefinido ao solicitor a técnica *event.preventDefault()* para o evento *ondragover* :

```
event.preventDefault()
```

- Assim que o elemento é largado, acontece um evento *drop. No exemplo anterior, o atributo ondrop* solicita a função *drop(event):*

```
function drop(ev) {
  ev.preventDefault();
  var data=ev.dataTransfer.getData("text");
  ev.target.appendChild(document.getElementById(data));
}
```

Drag e *Drop* em HTML5

- ✓ A função preventDefault() é usada para impedir o processamento dos dados predefinido do navegador (a predefinição está aberta como um *link* ao largar);
- ✓ O programador obtém os dados movidos com a técnica *dataTransfer.getData()*. Esta técnica devolve qualquer informação que tenha sido definida para o mesmo tipo na técnica *setData()*;
- ✓ A informação arrastada é a *id* do element movido ("drag1")
- ✓ O programador também deve anexar o elemento arrastado ao elemento que é largado.

Vamos praticar!

Aceda ao link em baixo para praticar as competências adquiridas:

- https://www.etutorialspoint.com/index.php/exercise/html-practice-exercises-with-solutions

code4sp
coding for social promotion

Obrigado(a)!

Co-funded by the
Erasmus+ Programme
of the European Union