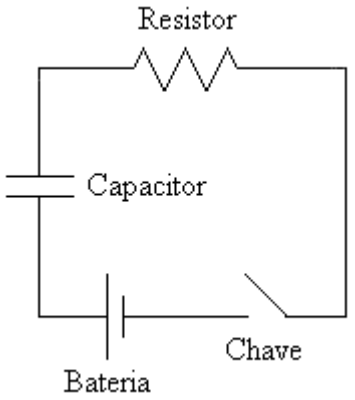
21 Circuito RC

Um circuito *RC* típico contém uma bateria que mantém um potencial constante ε, um resistor *R*, um capacitor *C* e uma chave para controlar o circuito. Quando a chave é fechada, uma corrente aparece no circuito e o capacitor é progressivamente carregado. A corrente inicial é dada pela força eletromotriz da bateria dividida pela resistência e decai exponencialmente com uma constante de tempo *RC*:



$$I(t) = (\varepsilon/R)\, e^{-t/RC}$$

Para um sistema com uma bateria de 3 V (duas pilhas comuns) e resistência de 10 kΩ, a corrente como função do tempo é a apresentada na tabela abaixo:

***t* (ms)**	0	100	200	300	400	500	600	700	800	900	1000
***I* (mA)**	300	246	201	165	135	110	90	74	61	50	41

O script abaixo produz um gráfico em modo texto dentro de uma área de texo (`textares`). Para tanto, utiliza os valores de pontos (*x*,*y*) digitados pelo usuário num quadro à esquerda da área do gráfico. O script desenha o gráfico quando o usuário aperta o botão OK, assumindo que há apenas um ponto por linha e que os valores de *x* e *y* estão separados por apenas um espaço em branco.

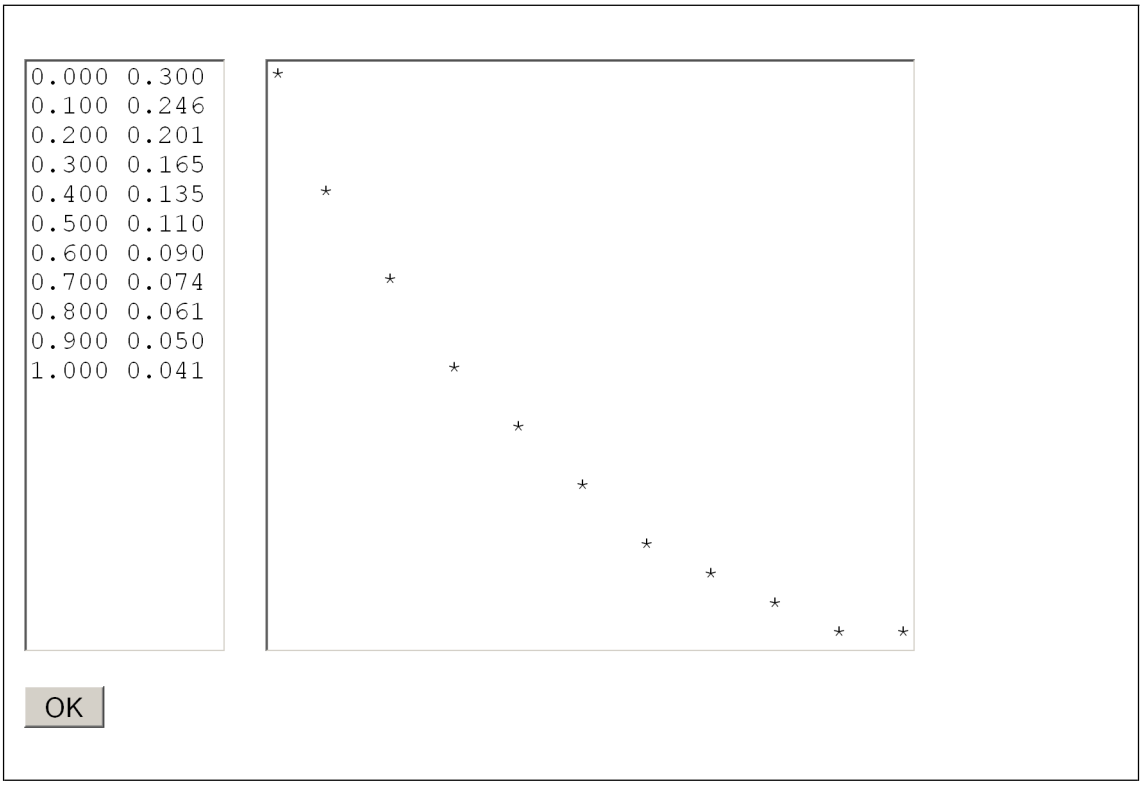
`exemplo-21-1.html`

```
<p align="center">
<textarea cols=12 rows=20 id="inputRC"></textarea>

<textarea cols=40 rows=20 id="outputRC"></textarea>
</p>

<p><input type="button" value="OK" onClick="fazGrafico()"></p>
<script>
```

```
function TransCoordRC(cLrg,cAlt,mxi,myi,mxf,myf) {
   this.Bx = cLrg/(mxf-mxi);
   this.Ax = - this.Bx * mxi;
   this.By = cAlt/(myf-myi);
   this.Ay = - this.By * myi;
}
TransCoordRC.prototype.cx = function(mx) {
   return this.Ax + this.Bx * mx;
}
TransCoordRC.prototype.cy = function(my) {
   return this.Ay + this.By * my;
}

function fazGrafico() {

   var inStr = document.getElementById("inputRC").value;
   var xy = inStr.split(/\n/g);
   var x = new Array();
   var y = new Array();
   for (var i in xy) {
      var pto = xy[i].split(" ");
      x[i] = parseFloat(pto[0]);
      y[i] = parseFloat(pto[1]);
   }

   var xmin = +1*Number.MAX_VALUE;
   var ymin = +1*Number.MAX_VALUE;
   var xmax = -1*Number.MAX_VALUE;
   var ymax = -1*Number.MAX_VALUE;

   for (var i in x) {
      if (x[i]<xmin) xmin = x[i];
      if (y[i]<ymin) ymin = y[i];
      if (x[i]>xmax) xmax = x[i];
      if (y[i]>ymax) ymax = y[i];
   }

   xmin = xmin - 0.01 * Math.abs(xmin);
   xmax = xmax + 0.01 * Math.abs(xmax);
   ymin = ymin - 0.01 * Math.abs(ymin);
   ymax = ymax + 0.01 * Math.abs(ymax);

   var cxmax = parseInt(document.getElementById("outputRC").cols);
   var cymax = parseInt(document.getElementById("outputRC").rows);

   var tc = new TransCoordRC(cxmax,cymax,xmin,ymin,xmax,ymax);

   var mat = Array(cymax);
   for (var i=0;i<cymax;i++)
      mat[i] = new Array(cxmax);

   for (var i=0;i<cymax;i++)
      for (var j=0;j<cxmax;j++)
         mat[i][j] = " ";

   for (var i in x) {
      var cx = Math.floor(tc.cx(x[i]));
      var cy = Math.floor(tc.cy(y[i]));
      mat[cy][cx] = "*";
   }

   var str = "";
   for (var i=(cymax-1);i>=0;i--) {
      for (var j=0;j<cxmax;j++) {
```

```
            str += mat[i][j];
        }
        str += "\n";
    }

    document.getElementById("outputRC").value = str;

}

var E = 3;
var R = 1e4   ;
var C = 50e-6;
var str = "";
for (var i=0;i<=10;i++) {
    var t = i/10;
    var I = 1000 * E/R * Math.exp(-t/(R*C));
    str += t.toFixed(3) + " " + I.toFixed(3);
    if (i!=10) str += "\n";
}
document.getElementById("inputRC").value = str;

</script>
```

Resultado:



A estrutura visível do documento `HTML` contém apenas 3 elementos: dois `textarea` e um `input` do tipo `button`. O primeiro `textarea` permite que o usuário digite os valores *x*,*y* dos pontos que deseja ver no gráfico. Os valores de *x* e *y* devem estar separados por um espaço em branco e a linha terminada por um `ENTER`. O elemento pode receber dados digitados em outro aplicativo (uma planilha ou um editor de textos) usando o tradicional esquema "copiar & colar".

O segundo elemento `textarea` mostra o gráfico dos dados no modo texto, isto é, utilizando

caracteres comuns para representar os pontos em uma matriz de espaços em branco. O script contém uma transformação de coordenadas que mapeia o domínio dos dados digitados para o número de linhas e colunas disponíveis neste elemento.

A função `fazGrafico()` inicia interpretanto o texto contido na primeira `textarea`. Primeiro busca no documento o seu conteúdo (`value`) armazenando-o na variável `inStr`. Essa variável passa a conter toda a sequência de caracteres que estava no `textarea`. Essa sequência, além de conter os valores de *x*, *y* e os espaços em branco que os separam, contém alguns caracteres invisíveis que controlam o fluxo do texto.

Na linha seguinte, é utilizado o método `split` do objeto `String` para separar o texto em blocos que contenham os pontos (*x*,*y*). O método retorna uma matriz (`Array`) de elementos do tipo `String`, contendo por exemplo o valor `"0.100 0.246"`. Para fazer isso, recebe como argumento a *expressão regular* `"/\n/g"`. Expressões regulares são compostas de códigos que o método interpreta para identificar caracteres ou cadeias de caracteres de interesse e fica a cargo do leitor decidir se quer ou não ampliar o seu conhecimento a respeito delas em outras fontes (a Internet é riquíssima em informações sobre expressões regulares/*regular expressions)*.

A seguir, o script cria dois objetos do tipo `Array` para armazenar os valores de *x* e *y* separadamente. Um laço `for` é montado para percorrer os elementos de `inStr`. A cada passo, o método `split()` é chamado para os sucessivos elementos de `xy`, tendo como argumento um espaço em branco (`" "`), que é o que separa os valores de *x* e *y* armazenados como `String`. O resultado desta chamada é guardado na variável temporária `pto`. Note que não é preciso declarar a variável `pto` como `Array`; o interpretador faz isso automaticamente uma vez que o resultado da chamada a `split()` retorna este tipo de dado. Nas linhas seguintes, os valores de `pto[0]` e `pto[1]` são convertidos para números reais pelo `parseFloat()` e guardados nas variáveis definitivas.

O próximo passo é buscar os valores mínimo e máximo para *x* e *y*, para definir a transformação de coordenadas que fará com que os dados "caibam" num quadro desenhado na página. Primeiro, as variáveis `xmin` e `ymin`, que guardarão os menores valores encontrados, são criadas e inicializadas com o maior valor possível que o JavaScript suporta; as variáveis `xmax` e `ymax`, que guardarão os maiores valores encontrados, são criadas e inicializadas com o menor valor suportado, que é o negativo do maior. A propriedade `MAX_VALUE` do objeto `Number` corresponde a 1.79769313486232e+308.

A seguir, é feita a busca propriamente dita: um laço `for` varre as matrizes `x[i]` e `y[i]` e pergunta se os valores contidos em seus elementos são maiores do que os valores correntes de `xmax` e `ymax` e menores do que os valores correntes de `xmin` e `ymin`; caso isto aconteça, estes valores passam a ser os novos valores correntes.

As linhas seguintes fazem uma pequena (1%) ampliação dos intervalos em *x* e *y*, o que pode ser interessante para evitar que pontos sejam desenhados exatamente em cima da moldura do quadro. Este procedimento pode não ter efeito algum em um gráfico de caracteres como este, mas pode ser interessante quando utilizarmos procedimentos gráficos mais refinados.

Para definir a transformação de coordenadas, são necessárias, além dos valores mínimos e máximos para *x* e *y* (os limites do sistema de coordenadas do usuário) as dimensões da "tela" onde será apresentado o gráfico. Estas dimensões, armazenadas em `cxmax` e `cymax`, são obtidas da propriedade `cols` (colunas) e `rows` (linhas) do elemento buscado pelo método

`getElementById()`, que recebeu como argumento o `id` do elemento `output`, a `textarea` onde será desenhado o gráfico.

A seguir, é criado um objeto `Array` com `cymax` linhas e `cxmax` colunas, o que neste exemplo resulta em uma matriz com 20 × 40 = 800 elementos que representa a área do gráfico. Todos os elementos são em seguida inicializados com um espaço em branco (`" "`), "limpando" a área do gráfico.

O bloco seguinte utiliza os métodos `cx()` e `cy()` da transformação de coordenadas `tc` para produzir, a partir dos valores de *x* e *y*, índices para os elementos da matriz que serão preenchidos com um asterísco (`"*"`), indicando um ponto no gráfico. Note o uso do método `floor()` do objeto `Math`, que retorna o primeiro número inteiro *abaixo* do número passado como argumento.

O próximo bloco tranforma o mapa de espaços em branco e asteriscos que está na memória em um objeto do tipo `String` que pode ser colocado na área de texto. O primeiro laço `for` refere-se às linhas, que devem ser colocadas de cima para baixo, por isso o início em `cymax-1` e o decremento (`i--`) até 0. O segundo laço `for` refere-se às colunas, que seguem da esquerda para a direita. Note o uso do operador `"+="` na contração de `str = str + mat[i][j]` em `str += mat[i][j]`. Note também que no final de cada linha (do laço em `j`) é anexado à string de saída um caracter de quebra de linha (`"\n"`). Finalmente, a string é colocada no atributo `value` do elemento identificado por `output`, o que a faz aparecer na respectiva `textarea` do documento.

Exercícios

1. Modifique o script acima de modo que imprima asteriscos entre o eixo horizontal e os pontos do gráfico (isto é, transforme o gráfico de pontos em um gráfico de barras feitas de asteriscos).

2. Faça um script que leia uma coluna de dados da área de `input` e imprima um histograma (gráfico de barras) a partir dela. Os dados de entrada podem ter um número arbitrário de linhas e seu algoritmo deve comprimi-las para caber no número de colunas da `textarea`.