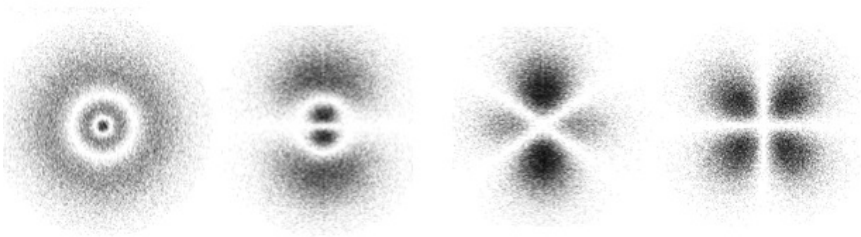
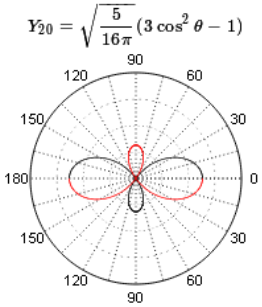
Universidade Federal de Santa Catarina
Centro de Ciências Físicas e Matemáticas
Departamento de Física

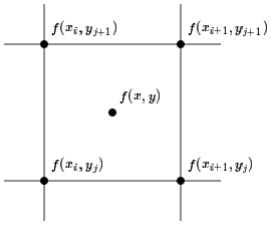
nelson.canzian@ufsc.br



Física



com



JavaScript

Prof. Nelson Canzian da Silva
Florianópolis, fevereiro de 2016

Sumário

Introdução

Este livro é a 2a. edição, revisada e atualizada, de parte de um texto que tenho utilizado nos últimos sete anos como apoio para disciplinas relacionadas à física computacional. Os exemplos e amplitude dos temas cobertos, sempre crescentes, têm permitido o seu uso em disciplinas optativas e obrigatórias, em nível introdutório e intermediário, da Licenciatura em Física, do Bacharelado em Física e do Mestrado Nacional Profissional em Ensino de Física oferecidas na Universidade Federal de Santa Catarina.

O material selecionado abrange aproximadamente o conteúdo desenvolvido em uma disciplina semestral (18 semanas com 4 horas-aula por semana) frequentada por alunos que, supõe-se, não têm nenhum conhecimento sobre linguagens de programação. O material complementar, não incluído no livro mas disponível em meu site na internet (canzian.fsc.ufsc.br), está sempre crescendo. Contém essencialmente alguns projetos ou esboços de projetos de maior fôlego, tanto do ponto de vista de programação quanto de física e matemática, selecionados ao longo desses anos.

A linguagem de programação escolhida para materializar os algoritmos foi o JavaScript. Costumo dizer que fiz a minha Iniciação Científica em Assembler, o Mestrado em Pascal, o Doutorado em Fortran e o Pós-Doutorado em C++. Recentemente conheci um pouco do Python e sua sintaxe enxuta. Mas há cerca de sete anos, buscando dar interatividade às páginas de material didático das minhas disciplinas e dos meus projetos de pesquisa e extensão, vim a conhecer o JavaScript e aprofundar meus conhecimentos de HTML e CSS. Hoje faço virtualmente tudo com essas tecnologias, de textos didáticos e artigos científicos a desenhos técnicos e modelagem de sistemas físicos.

HTML/CSS/JavaScript são as tecnologias de base da internet hoje. Se o seu dispositivo (computador, tablet, smartphone) possui um navegador (e qual não tem?) você está automaticamente equipado para consumir e produzir conteúdo com essas tecnologias. Para isso não é preciso baixar nada, comprar nada, instalar nada, compilar nada, link-editar nada. Basta abrir um editor de textos qualquer, digitar seu código, salvar e carregar no navegador como uma página qualquer.

A combinação HTML/CSS/JavaScript dá a um documento características dinâmicas tanto quando é carregado pelo navegador quanto posteriormente, através de ações do usuário ou de temporizadores programáveis. Com HTML/CSS/JavaScript você faz botões funcionarem, informações em caixas de texto serem preenchidas, desenhos, animações e até complexos cálculos de física e engenharia! E mais: são milhões de desenvolvedores em todo o mundo produzindo códigos intrinsecamente abertos e reaproveitáveis, e bilhões de usuários ajudando a melhorá-los, muitas vezes sem sequer aperceberem-se disso. Essa imensa base de

usuários e desenvolvedores leva a crer que nenhuma outra combinação de tecnologias irá substituir o HTML/CSS/JavaScript nas próximas décadas. Ao contrário, um modelo tão bem sucedido só tende a ter suas aplicações expandidas para os mais diferentes campos.

Este livro, por exemplo, foi originalmente escrito em HTML/CSS/JavaScript. Todos os botões, desenhos, gráficos e animações que estão descritas no livro funcionam quando o arquivo HTML original é carregado pelo navegador. Alguns recursos incorporados permitem fazer a numeração automática de capítulos, dos exemplos de programas e exercícios. Se eu quiser incluir ou remover um capítulo ou exemplo de programa do meio do livro, esses recursos se encarregam de renumerar tudo automaticamente. A formatação de cabeçalhos, quadros, títulos e tudo o mais pode ser alterada no livro todo, simultaneamente, com a mudança de algumas poucas linhas nos arquivos de script (.js) ou de estilo (.css).

Em vista disso trata-se, a meu ver, da melhor plataforma hoje disponível para o ensino de técnicas modernas e versáteis de organização de informação, produção de documentos e, obviamente, de programação. O JavaScript é uma linguagem orientada a objetos com elementos para o gerenciamento de ricas interfaces com o usuário e amplas capacidades para lidar com desenhos e animações. Sua sintaxe é essencialmente a mesma do C/C++/Java, o que facilita a migração para uma linguagem compilada quando necessário.

Para não dizer que tudo é perfeito, é preciso informar ao leitor de algumas características que são eventualmente vistas como "problemas" por usuários com demandas específicas. Uma dessas características é que o JavaScript é uma linguagem interpretada e não compilada. Para quem não sabe a diferença, talvez seja suficiente dizer que uma linguagem interpretada é dezenas de vezes mais lenta que uma linguagem compilada. Na vasta maioria das aplicações isso sequer é percebido, mas para quem quer fazer zilhões de cálculos (meteorologistas, geneticistas, engenheiros aeronáuticos e afins), talvez não seja a melhor opção (mas os supercomputadores, com suas linguagens especializadas, estão aí para isso).

Outro potencial problema é o fato de que os navegadores desenvolvidos por diferentes fabricantes (Apple, Google, MicroSoft, Mozilla etc.) às vezes respondem de maneira um pouco diversa a uma mesma sequência de instruções HTML/CSS/JavaScript. Isso está cada vez mais raro, mas ainda faz com que algumas aplicações tenham que incluir adaptações específicas para os diferentes navegadores.

Por fim, há o grupo que reclama que o JavaScript não tem comandos para ler e escrever arquivos em disco da mesma maneira que as linguagens "tradicionais", sem a intervenção do usuário. Eu mesmo já pensei assim, mas meu modo de pensar foi modificado pelo uso cotidiano e hoje raramente enfrento um problema em que isso não possa ser elegantemente contornado por outras características da linguagem, em geral com vantagens.

Neste livro procuro apresentar as estruturas básicas das linguagens de programação utilizando a sintaxe do JavaScript. Simultaneamente, são introduzidos conceitos do HTML/CSS, sempre com exemplos que remetem a problemas de física e matemática básicas. Não é uma apresentação exaustiva e não está organizada como um manual de referência. Por serem as tecnologias da internet, há um sem número de excelentes tutoriais e manuais de referência disponíveis sobre HTML/CSS/JavaScript. Na dúvida, busque "*informação* + javascript" no seu buscador predileto e uma enxurrada de sites virá em seu auxílio.

O livro contém 11 capítulos, que não necessariamente precisam ser lidos na ordem proposta, nos quais apresento os elementos básicos do HTML/CSS/JavaScript e, quase subliminarmente, lógica de programação. Um aviso importante: por mais simples que possam parecer os exercícios propostos no meio do texto (e em geral são muito simples), não deixe de fazê-los. Eles **não** são desafios cognitivos, mas estão ali para facilitar a sua interação inicial

com a linguagem e suas idiossincrasias e acostumar os seus olhos com um editor de textos sem formatação. Se você não está convencido disso, compare as muitas horas que mesmo um milionário jogador de futebol passa, em seus treinos, repetindo ações simples como embaixadinhas, chutes a gol e corridas em zigue-zague, com as horas em que efetivamente joga uma decisiva partida de futebol.

Além dos 11 capítulos, o livro contém uma bibliografia comentada (que, dadas as facilidades da internet hoje, pode nem lhe ser muito útil) e quatro apêndices. O primeiro deles avança um pouco além das técnicas básicas de programação e mostra como utilizar *plug-ins* para obter equações e gráficos com boa qualidade editorial. Os dois seguintes foram pensados como "exemplos integradores" de conteúdos, diferentemente dos exemplos curtos e pontuais dos capítulos anteriores. O quarto apêndice traz uma tabela com a maneira de especificar alguns caracteres especiais (letras gregas, símbolos matemáticos etc.) que de vez em quando precisamos utilizar em textos técnicos e científicos.

Finalmente, gostaria de agradecer a todos os alunos de graduação e pós-graduação que cursaram as disciplinas de Física com JavaScript, Introdução à Física Computacional, Física Computacional e Atividades Computacionais para o Ensino Fundamental e Médio e que, de uma maneira ou de outra, ao utilizarem e criticarem a edição anterior deste livro nos últimos seis anos, contribuíram para o seu aperfeiçoamento. Gostaria de agradecer também ao prof. Nilton da Silva Branco, do Departamento de Física da UFSC, pela sua criteriosa leitura desta edição e pelos valiosos comentários e infinita paciência para verificar se os exemplos apresentados funcionavam como esperado.

Bom estudo!

Nelson Canzian da Silva
Florianópolis, fevereiro de 2016

1 Operadores e funções matemáticas

Neste capítulo você verá como transformar expressões matemáticas que utilizam operadores aritméticos (+,−,×,÷) e funções matemáticas (seno, cosseno, logaritmo etc.) em linhas de programa executáveis pelo computador. Verá também como escrever na tela o resultado de cálculos com a precisão ou o número de algarismos significativos desejado e como colocar comentários (textos ignorados pelo interpretador) no seu programa.

Abra o seu editor de textos predileto, escreva as seguintes linhas e salve o arquivo com uma extensão `.html`:

Listagem:

```
<script>
document.write("Meu primeiro script");
</script>
```

Para utilizar HTML/CSS/JavaScript, assim como qualquer outra linguagem de programação, você precisa de um editor de textos que salve o documento sem qualquer formatação adicional. O Bloco de Notas (Windows), o Gedit (Linux) e vários outros disponíveis gratuitamente na internet (EditPad, TextPad etc.) são exemplos de editores que fazem isso automaticamente. Se você insistir em utilizar algo como o Word ou o LibreOffice, ao salvar escolha "texto sem formatação" para o tipo de arquivo e coloque explicitamente a extensão HTML no nome do arquivo.

Clique no arquivo para visualizá-lo no navegador. Você deverá ver abrir uma janela do seu navegador padrão com a frase "Meu primeiro script" escrita nela. Pronto, você fez o seu primeiro script em JavaScript!

Os scripts são sempre antecedidos por `<script>` e sucedidos por `</script>`, que podem aparecer múltiplas vezes e em vários pontos de um mesmo documento HTML.

Vamos incrementá-lo um pouco, introduzindo algumas variáveis. Experimente digitar as seguintes linhas, salve o arquivo e visualize-o no navegador:

Listagem:

```
<script>
var a = 2;
var b = 3;
var c = a + b;
document.write("A soma de " + a + " com " + b + " resulta em " + c);
</script>
```

Resultado:

A soma de 2 com 3 resulta em 5

O comando `var` é utilizado para declarar variáveis em JavaScript e o ponto-e-vírgula é utilizado para indicar que o comando foi terminado. O JavaScript não vai reclamar se você esquecer de usar o `var` nem o ponto-e-vírgula, mas é altamente recomendável que você sempre os utilize, pois em algumas situações específica sua ausência pode levar a erros difíceis de identificar.

Diferentemente de muitas outras linguagens de programação (C, C++, Java etc.) em JavaScript não se especifica o tipo da variável (inteiro, ponto flutuante (real), booleano etc.). O JavaScript vai decidir e modificar o tipo de variável conforme o contexto em tempo de execução. Mas você deve ter clareza disso e de como usa suas variáveis pois, dependendo do problema, essa característica pode ser o céu ou o inferno, também levando a erros difíceis de identificar.

O método `write()` do objeto `document`, acessado com o operador ponto (`"."`), escreve algo no documento HTML quando ele está sendo carregado. A instrução `document.write()` recebe como argumento uma expressão literal, que pode ser formada por constantes, que são colocadas entre aspas simples ou duplas, ou por variáveis ou combinações de constantes e variáveis. No exemplo acima o sinal de mais (`"+"`) é utilizado para concatenar (juntar) constantes e variáveis em literais.

Sua vez... (1-1)

Modifique o exemplo anterior de modo que defina as variáveis `x0`, `v0`, `a` e `t`, atribuindo-lhes os valores 0 m, 0 m/s, 9,8 m/s^2 e 1 s, respectivamente, e calcule o valor de `x = x0 + v0*t + a*t*t/2`. Para ter certeza de que seu programa está correto, experimente outros valores e verifique se o resultado é o esperado.

Em JavaScript, assim como em virtualmente todas as outras linguagens de programação, o ponto (".") e não a vírgula (",") é utilizado como separador decimal. Certifique-se que no programa o valor atribuído à aceleração é `9.8` e não `9,8`.

O objeto `Math` do JavaScript dispõe várias constantes pré-definidas e operações matemáticas básicas. O script abaixo calcula o logaritmo natural (na base *e*) de π e divide-o pelo seno de *e*:

Listagem:

```
<script>
var a = Math.PI;
var b = Math.E;
var c = Math.log(a)/Math.sin(b);
document.write("log(" + a + ")/sin(" + b + ")= " + c);
</script>
```

Resultado:

log(3.141592653589793)/sin(2.718281828459045)= 2.7867137873975154

Uma grande fonte de confusão na formatação da informação a ser apresentada é a quantidade e localização das aspas (`"`) e sinais de concatenação (`+`) na formação da expressão final. Olhe atentamente os exemplos anteriores e certifique-se de que entendeu perfeitamente o uso desses sinais para delimitar e juntar as constantes e as variáveis.

No exemplo acima foi utilizado novamente o operador `"."` para dar acesso a propriedades (`Math.PI`, `Math.E`) e métodos (`Math.log()`, `Math.sin()`) do objeto `Math`. Esta é a forma geral de acessar propriedades e métodos dos objetos em JavaScript.

O JavaScript, como toda linguagem de programação, oferece todas as funções matemáticas fundamentais. Dentre as funções disponíveis no objeto `Math` estão:

`Math.abs(x)` - valor absoluto (módulo) do argumento.
`Math.acos(x)` - arco-cosseno do argumento.
`Math.asin(x)` - arco-seno do argumento.
`Math.atan(x)` - arco-tangente do argumento.
`Math.atan2(y,x)` - ângulo cuja tangente é *y*/*x*.
`Math.ceil(x)` - menor inteiro maior ou igual ao argumento.
`Math.exp(x)` - retorna *e* = 2.71... elevado a *x*.
`Math.floor(x)` - maior inteiro menor ou igual ao argumento.
`Math.max(x,y)` - maior dos dois argumentos.
`Math.min(x,y)` - menor dos dois argumentos.
`Math.pow(x,y)` - valor de *x* elevado a *y*.
`Math.random(x)` - retorna um número aleatório entre 0 e 1 inclusive.
`Math.round(x)` - valor arredondado do argumento.

O método `Math.log(x)` retorna o logaritmo natural de `x`, que, em matemática, é usualmente representado por ln(*x*), o que às vezes causa certa confusão. O logaritmo de um número na base 10, pode ser obtido utilizando-se o método `Math.log10(x)` ou a fórmula de mudança de base:

$\log(x) = \ln(x) / \ln(10)$

Em um programa, esse cálculo poderia ser implementado como algo do tipo:

```
var x = 2;
var y = Math.log10(x);
var z = Math.log(x)/Math.log(10);
```

Sua vez... (1-2)

Faça um script que calcule os valores do seno, o cosseno, tangente, raiz quadrada, logaritmo na base *e* e logaritmo na base 10 de $x = 2$. Confira seus resultados com o que você obtém com uma calculadora de mão.

O JavaScript tem dois métodos para lidar com a maneira como são apresentados os dígitos de um número, os métodos `Number.toFixed(n)` e `Number.toPrecision(n)`. Os dois métodos transformam um objeto `Number`, que pode ser uma constante ou uma variável, em um literal. No caso do `toFixed(n)`, o número é transformado em um literal com *n* algarismos depois do ponto decimal; no caso do `toPrecision(n)`, em um literal com *n* algarismos significativos. O script a seguir ilustra a sua aplicação e efeitos.

Listagem:

```
<script>
var x = Math.sqrt(2);
document.write(x.toFixed(0) + "<br>");
document.write(x.toFixed(2) + "<br>");
document.write(x.toFixed(4) + "<br>");
document.write(x.toPrecision(1) + "<br>");
document.write(x.toPrecision(2) + "<br>");
document.write(x.toPrecision(4) + "<br>");
</script>
```

Resultado:

```
1
1.41
1.4142
1
1.4
1.414
```

Sua vez... (1-3)

Experimente ver e analisar o que acontece quando você altera a primeira linha do exemplo anterior para `var x = Math.sqrt(1000*2)` e para `var x = Math.sqrt(0.001*2)`.

Estes métodos podem ser utilizados com números especificados explicitamente:

Listagem:

```
<script>
document.write((100).toFixed(0) + "<br>");
document.write((100).toFixed(2) + "<br>");
document.write((100).toFixed(4) + "<br>");
document.write((100).toPrecision(1) + "<br>");
document.write((100).toPrecision(2) + "<br>");
document.write((100).toPrecision(4) + "<br>");
</script>
```

Resultado:

```
100
100.00
100.0000
1e+2
1.0e+2
100.0
```

Nos exemplos anteriores, você deve ter notado que no final de cada linha a literal "<br>" é adicionada. Trata-se de um marcador (*tag*) do HTML que faz com que o navegador quebre a linha (é uma abreviação de *break*). Experimente removê-lo e veja o que acontece. O documento onde o JavaScript opera e apresenta seus resultados é sempre um documento HTML e veremos aos poucos vários outros marcadores utilizados para sua estruturação e formatação.

Constantes, variáveis e funções podem ser combinadas em fórmulas complexas. O exemplo a seguir implementa a fórmula de Torricelli para o cálculo da velocidade no movimento retilíneo uniformemente variado e a fórmula para o cálculo da força elétrica entre um elétron e um próton separados de 1 nanometro (10^{-9} m).

Listagem:

```
<script>
// Equação de Torricelli
// utilizada para calcular a velocidade final
// a partir da velocidade inicial, da aceleração
// e da distância percorrida pelo objeto
var vo = 0;
var a  = 9.8;
var Dx = 10;
var v = Math.sqrt(vo*vo + 2*a*Dx);
document.write("<i>v</i> = " + v.toFixed(2) + " m/s <br>");

/*
Lei de Coulomb
utilizada para calcular a força
entre duas cargas elétricas q1 e q2
separadas de uma distância d
*/
```

```
var q1 = -1.6e-19;
var q2 = +1.6e-19;
var d  = 1e-9;
var F = 8.99e9 * q1 * q2 / Math.pow(d,2);
document.write("<i>F</i> = " + F.toPrecision(3) + " N <br>");
</script>
```

Números muito grandes ou muito pequenos são escritos em potências de 10, como a carga do elétron, que é 1.6×10^{-19} C. Em linguagens de programação potências de 10 são em geral indicadas por um `"e"` entre a mantissa e o expoente. Nesta notação a carga do elétron fica `1.6e-19` e a constante elétrica 8.99×10^9 fica `8.99e9`. Ao executar o script, você deverá obter os seguintes resultados:

v = 14.00 m/s
F = -2.30e-10 N

Note o uso de `//` para indicar um comentário. Qualquer texto colocado na linha (e apenas naquela linha) após as duas barras será ignorado pelo interpretador. Este tipo de comentário aplica-se linha a linha. Para comentar diversas linhas sucessivas utiliza-se `/* ... */`. Este esquema de incorporar comentários é o mesmo utilizado em C/C++/Java e outras linguagens delas derivadas.

As sequências `<i>...</i>` indicam o marcador HTML que coloca o texto entre eles em *itálico*, que é como normalmente aparecem variáveis escalares em textos técnicos e científicos. Os marcadores HTML sempre são identificados por estarem entre `"<"` e `">"` e, com raras exceções, andam aos pares, indicando o início e o fim do texto ou outro elemento marcado.

No caso da equação de Torricelli o quadrado da velocidade inicial foi calculado utilizando a função `Math.pow(x,y)` e no caso da lei de Coulomb o quadrado da distância foi calculado fazendo `(d*d)`. As duas formas são equivalentes. Não esqueça dos parênteses, pois o resultado seria desastroso.

Sua vez... (1-4)

Uma das maiores fontes de frustração de novatos em programação e marcação de informação é que os computadores são **absolutamente** intolerantes aos mais pequenos erros de digitação. Para acostumar seus olhos, dedos e cérebro a essa intolerância, um bom exercício é reescrever, "brincar", com um código que funciona e ver se ele continua funcionado.

Modifique o exemplo anterior de modo que:

- a maneira de fazer comentários seja invertida (isto é, use `/*...*/` para comentar a equação de Torricelli e `//` para comentar a lei de Coulomb)
- a equação de Torricelli e as variáveis associadas sejam trocadas pela equação horária do MRU e que a variável calculada seja o tempo. Isto é, dados x, x_0 e v_0 calcular t (`var t = (x-x0)/v0`).
- no exemplo da lei de Coulomb sejam dados os valores das cargas (`q1` e `q2`) e da força (`F`), e o programa calcule o valor da distância (`d`).

A expressão `Math.pow(vo,2) + 2*a*Dx` é passada como argumento para a função `Math.sqrt()`. Quando muitos aninhamentos são necessários, eventualmente fica mais claro dividir o cálculo de expressões como estas em várias linhas, evitando confusões no número de parênteses. Assim, a expressão:

```
var x = Math.pow(Math.sqrt(Math.sin(Math.atan(a*a+b*b))),1/2);
```

Pode ser reescrita equivalentemente como:

```
var tmp1 = a*a + b*b;
var tmp2 = Math.atan(tmp1);
var tmp3 = math.sin(tmp2);
var tmp4 = Math.sqrt(tmp3);
var x = Math.pow(tmp4,1/2)
```

Sua vez... (1-5)

Faça um programa que, dada a equação de 3o. grau:

x^3 + a_1 x^2 + a_2 x + a_3 = 0

calcule suas soluções:

\begin{eqnarray} x_1 & = & S + T + \frac{1}{3} a_1 \\ x_2 & = & -\frac{1}{2} (S + T) - \frac{1}{3} a_1 + \frac{1}{2} i \sqrt{3} (S-T) \\ x_3 & = & -\frac{1}{2} (S + T) - \frac{1}{3} a_1 - \frac{1}{2} i \sqrt{3} (S-T) \\ \end{eqnarray}

onde:

\begin{eqnarray} Q & = & \frac{3 a_2 - a_1^2}{9} \\ R & = & \frac{9 a_1 a_2 - 27 a_3 - 2 a_1^3}{54} \\ S & = & \left(R + \sqrt{Q^3 + R^2} \right)^{1/3} \\ T & = & \left(R - \sqrt{Q^3 + R^2} \right)^{1/3} \\ \end{eqnarray}

Finalmente, confira os resultados verificando se:

\begin{eqnarray} - a_1 & = & x_1 + x_2 + x_3 \\ a_2 & = & x_1 x_2 + x_2 x_3 + x_3 x_1 \\ - a_3 & = & x_1 x_2 x_3\\ \end{eqnarray}

Exercícios

Escreva scripts que calculem o valor das seguintes expressões com a precisão indicada na resposta:

1. Equação de movimento para o MRU:

   $x = x_o + v_o t + (1/2)at^2$

   para $x_o = 3.00$ m, $v_o = 5.00$ m/s, $a = -9.80$ m/s^2, $t = 2.00$ s. *Resposta:* -6.60.

2. Amplitude de uma onda transversal:

   $y = y_o \cos(2\pi(x/\lambda - ft))$

   para $y_o = 1.00 \times 10^{-2}$ m, $f = 1.00 \times 10^5$ Hz , $\lambda = 0.033$ m, $x = 2.00$ m , $t = 2.00$ s. *Resposta:* -0.0079.

3. Campo elétrico de uma carga em um ponto do espaço:

   $E_x = kqx/(x^2 + y^2 + z^2)^{3/2}$
   $E_y = kqy/(x^2 + y^2 + z^2)^{3/2}$
   $E_z = kqz/(x^2 + y^2 + z^2)^{3/2}$

   para $k = 8.99 \times 10^9$ N·m^2/C, $q = 1.602 \times 10^{-19}$ C e $x = 1.00$ nm, $y = 2.00$ nm e $z = 3.00$ nm (1 nm = 1×10^{-9} m). *Resposta:* $E_x = 2.75 \times 10^7$ N/C, $E_y = 5.50 \times 10^7$ N/C, $E_z = 8.25 \times 10^7$ N/C.

4. Distribuição normal ou gaussiana:

   $g = Ae^{-(x-x_o)^2/(2\sigma^2)}$

   para $A = 10$, $x_o = 5.0$, $\sigma = x_o/4$, $x = 2.5$. *Resposta:* 1.4.

5. Função de onda do estado fundamental de átomos com um elétron:

   $\Psi = (1/\sqrt{\pi})(Z/a_o)^{3/2}e^{-Zr/a_o}$

   para $Z = 2$, $a_o = 5 \times 10^{-11}$ m, $r = 2a_o$. *Resposta:* 8×10^{13} m$^{-3/2}$.

2 Caixas de diálogo

Neste capítulo você vai aprender como fazer entrada e saída de dados em seu programa utilizando as caixas de diálogo ***`window.alert()`***, ***`window.confirm()`*** *e* ***`window.prompt()`***. *Verá também como verificar se a entrada de dados numéricos foi corretamente realizada pelo usuário e terá uma breve introdução ao controle de fluxo (abordado em profundidade no próximo capítulo).*

É possível evocar 3 tipos de caixas de diálogo em JavaScript:

- `window.alert(`*`msg`*`)`, que mostra informações ao usuário;
- `window.confirm(`*`msg`*`)`, que mostra informações ao usuário e recebe respostas do tipo "sim/não", "verdadeiro/falso", "ok/cancelar";
- `window.prompt(`*`msg`*`,`*`default`*`)`, que mostra informações ao usuário e permite que este digite um texto como resposta.

O funcionamento destas caixas de diálogo e alguns comandos adicionais que contribuem para aprimorar o seu uso são discutidos a seguir.

`window.alert(`*`msg`*`)`

A caixa de diálogo do tipo `window.alert(msg)` abre uma janela com a mensagem *`msg`*. O script abaixo mostra três maneiras de utilizar essa caixa de diálogo:

Listagem:

```
<script>
window.alert("Clique OK para fechar");
window.alert(Math.cos(Math.PI));
window.alert("O cosseno de " + Math.PI + " é " + Math.cos(Math.PI));
</script>
```

A mensagem passadas para a caixa de alerta podem ser constantes, como o texto no primeiro caso, varíáveis ou resultados de cálculos, como no segundo caso, e combinações de constantes, variáveis e cálculos concatenados pelo operador `"+"`, como no terceiro caso.

> Essa caixa de diálogo, assim como as demais, é do tipo *modal*, o que significa que enquanto estiver aberta a execução do programa é interrompida e somente é retomada com o fechamento da caixa de diálogo.

Sua vez... (2-1)

Modifique o script acima de modo que:

a. defina uma variável `var angulo = (`*`ângulo em graus`*`) * Math.PI/180`;
b. defina uma variável `var tanAngulo = Math.tan(angulo)`;
c. componha e apresente em uma caixa tipo `window.alert` a mensagem abaixo utilizando o sinal de concatenação (+) para ligar textos às variáveis definidas nos itens (a.) e (b.):

"A tangente de (*ângulo em graus*) graus é (*tangente do ângulo*)"

`window.confirm(`*`msg`*`)`

A caixa de diálogo do tipo `window.confirm(msg)` mostra uma mensagem definida pelo usuário e dois botões, tipicamente contendo `OK` e `Cancelar`. Se o usuário pressionar o botão `OK`, a caixa de diálogo retorna `true` (verdadeiro); se pressionar o `Cancelar`, retorna `false` (falso). O script a seguir mostra uma possível forma de utilização dessa caixa de diálogo.

Listagem:

```
<script>
var opcao = window.confirm("3 + 5 = 8? Clique OK se você concorda!");
if (opcao==true) {
   var msg = "Que bom!";
   window.alert(msg);
   }
else {
   var msg = "Você já pensou em trocar de carreira?";
   window.alert(msg);
   }
</script>
```

Na primeira linha do script é definida a variável `opcao`, que receberá o valor (`true` ou `false`) retornado pela caixa de diálogo quando o usuário pressionar um dos botões. Na linha seguinte é utilizado o comando `if (condição) { instruções } else { instruções }`, que é uma comando para *controle de fluxo* (que será visto detalhadamente no próximo capítulo). A instrução `if` avalia a `condição` e executa o primeiro bloco de instruções se o resultado for `true` ou o segundo bloco de instruções se o resultado for `false` (um bloco de instruções é constituído de uma ou mais instruções terminadas por ponto-e-vírgula (`";"`) e delimitado por chaves (`"{"` e `"}"`).

> Note que o teste de igualdade em JavaScript (assim como C, C++ e Java) é feito utilizando-se o operador "==" (dois sinais de igual sucessivos). Isso porque um simples sinal de igual seria interpretado como o operador de atribuição, provocando um erro de lógica que atormenta até programadores experientes. Se no exemplo acima a condição fosse escrita como `(opcao=true)`, o valor `true` seria atribuído à variável `opcao` e o primeiro bloco de instruções *sempre* seria executado, independentemente da escolha feita pelo usuário. Experimente!

Sua vez... (2-2)

a. No exemplo anterior, troque a condição para `(opcao==false)` e faça as demais modificações necessárias para que o script continue funcionando corretamente.
b. Deixe o exemplo anterior mais "enxuto", eliminando a variável `msg` atribuindo seu conteúdo diretamente ao argumento da caixa de diálogo `window.alert()`.
c. Quando o bloco de instruções no comando `if` contém apenas uma instrução, as chaves passam a ser opcionais. Experimente remover as chaves da sua resposta para o item anterior e veja se seu script continua funcionando corretamente.

`window.prompt(msg,default)`

A caixa de diálogo do tipo `window.prompt(msg,default)` oferece ao usuário um campo no qual ele pode digitar algum texto em resposta a uma pergunta ou afirmação. O primeiro argumento (`msg`) é a pergunta feita ao usuário; o segundo argumento (`default`) é uma resposta padrão que será utilizada caso o usuário pressione OK sem modificar o campo. Caso não haja um segundo argumento, o valor retornado fica indefinido (`undefined`). O exemplo a seguir pergunta ao usuário qual a sua cor predileta e oferece "azul" como opção padrão. Esse texto ou o texto digitado pelo usuário será armazenado na variável `resposta`, apresentada em seguida por uma caixa de diálogo do tipo `window.alert()`:

Listagem:

```
<script>
var resposta = window.prompt("Cor predileta?","Azul");
window.alert(resposta);
</script>
```

> Quando utilizar a caixa de diálogo `window.prompt()`, é prudente incluir a expressão padrão (*default*), que em geral será automaticamente gerenciada por instruções subsequentes. Ao deixar que a caixa de diálogo retorno um valor indefinido (`undefined`), pode ser que seja necessário a adição de instruções adicionais para lidar com mais essa possibilidade.

Sua vez... (2-3)

a. Modifique o exemplo anterior de modo que utilize duas chamadas a caixas de diálogo do tipo `window.prompt()`, uma que pergunte o nome do usuário e outra a sua idade, armazenando as respostas em variáveis distintas. Finalmente, o script deve utilizar uma caixa de diálogo tipo `window.alert()` para apresentar a frase:

Seu nome é (*nome digitado*) e você tem (*idade digitada*) anos.

b. No exemplo acima, apesar de possível, não faz muito sentido oferecer valores padrão (*default*) para quando o usuário não digita a informação solicitada. Você saberia como empregar valores padrão e o comando `if` para fazer com que, caso o usuário não digite uma das informações solicitadas, o script apresente a seguinte frase em uma caixa de diálogo tipo `window.alert()`:

"Informações incompletas. Tente novamente!"

O exemplo a seguir mostra um exemplo de uso das três caixas de mensagem em um mesmo script:

Listagem:

```
<script>
var input = window.prompt("Ângulo?","30");
var angulo = parseFloat(input);
if (isNaN(angulo)) {
 window.alert("Informação inválida. Tente novamente!");
}
else {
 var opcao = window.confirm(angulo + " graus. Tem certeza?");
 if (opcao==true) {
  var txt = "O seno de " + angulo + " é ";
  txt = txt + Math.sin(angulo*Math.PI/180).toFixed(2);
  window.alert(txt);
 }
 else {
  window.alert("Até a próxima!");
 }
}
</script>
```

Na primeira linha uma caixa de diálogo do tipo `window.prompt()` pede ao usuário um ângulo, oferecendo 30° como valor padrão. Na segunda linha a instrução `var angulo = parseFloat(input)` verifica se o valor fornecido é um número real (*float* é o jargão computacional para números reais, ou com *ponto flutuante*). Se a informação digitada puder ser convertida em um número real, este será armazenado na variável `angulo`. Caso contrário, a variável armazenará um `NaN`, que representa um não-número (*Not-a-Number*). A terceira linha contém uma instrução `if` que verifica se o retorno do método `isNaN(angulo)` é `true` ou `false`. Se o retorno for `true` (não é um número), pede ao usuário que tente novamente e encerra o script; caso contrário, executa o bloco de instruções associado ao `else`.

A primeira linha do bloco de instruções associado ao `else` armazena `true` ou `false` na variável `opcao` dependendo da resposta do usuário a uma nova caixa de diálogo tipo `window.confirm()`. A seguir, a instrução `if` pergunta se o usuário confirmou a opção (`opcao==true`). Caso afirmativo, acumula na variável `txt` o texto que em seguida é apresentado ao usuário por uma caixa de diálogo do tipo `window.alert()`. Caso contrário, uma caixa de diálogo do tipo `window.alert()` no bloco associado ao `else` despede-se com otimismo.

Sua vez... (2-4)

Para melhorar as suas habilidades de navegar entre todos os detalhes de um programa de computador, faça as seguintes alterações no exemplo anterior:

a. troque a solicitação inicial por "Ângulo em radianos?";
b. troque o valor padrão inicial para $\pi/6$, apresentando-o com duas casas decimais (0.52);
c. apresente a frase "Você não digitou um número válido!" quando a informação digitada não puder ser convertida em um número;
d. calcule a tangente do ângulo fornecido, apresentando-o com uma casa decimal;
e. despeça-se do usuário pedindo que ele "Volte mais vezes!"

Além do método `parseFloat()`, que converte a variável do tipo literal em uma do tipo numérico de ponto flutuante (o mais próximo de um número real que o computador consegue representar), existe também o método `parseInt()`, que converte uma variável literal em uma do tipo numérico inteiro. Se o número contiver casas decimais, é truncado e não arredondado. Caso o

método `parseInt()` não seja bem sucedido, também retorna um `NaN`.

Convém ressaltar que a caixa de diálogo do tipo `window.prompt()` *sempre* retorna um texto, que deve ser convertido em um número quando queremos que seja interpretado como tal. Por exemplo, considere o script a seguir:

```
<script>
var a = window.prompt("Entre com um número: ");
var b = window.prompt("Entre com outro número: ");
alert(a+b);
</script>
```

Se você digitar "2" e "3" como resposta às perguntas vai receber como resultado "23" ao invés do esperado "5". Isto porque o resultado do `prompt()` é sempre um texto, mesmo que seja um texto que representa um número. Para obter o resultado desejado é preciso utilizar o método `parseFloat()` (ou `parseInt()`):

```
<script>
var a = parseFloat(window.prompt("Entre com um número: "));
var b = parseFloat(window.prompt("Entre com outro número: "));
alert(a+b);
</script>
```

Se o usuário digitar alguma coisa que não seja um conteúdo válido, por exemplo `"zás"`, o método `parseFloat("zás")` retorna um `NaN`. Se o conteúdo for precedido por algo que pode ser convertido em um número de ponto flutuante, por exemplo `"3.87zás"`), o método `parseFloat("3.87zás")` retorna o número 3.87. Se o argumento, entretanto, já começar com uma informação inválida, por exemplo `"zás3.87"`, o `parseFloat("zás3.87")` também devolve um `NaN`.

Sua vez... (2-5)

Use caixas de diálogo tipo `window.prompt()` para pedir ao usuário que digite algo a ser armazenado na variável `arg` e use os métodos `parseFloat(arg)` e `parseInt(arg)` para converter os valores digitados em números e multiplicá-los por 2. Se você digitar "3.87", "3.87zás" e "zás3.87" deve obter os seguintes resultados:

Utilizando `parseFloat(arg)`: 7.74, 7.74, NaN

Utilizando `parseInt(arg)`: 6, 6, NaN

Os métodos `window.prompt()`, `window.alert()` e `window.confirm()` são métodos do objeto `window`, que é o objeto que está no topo da hierarquia de objetos de um documento interpretado pelos navegadores. Neste caso em particular, não é obrigatória a referência ao objeto `window` na chamada, isto é, podemos utilizar apenas `prompt()` ao invés de `window.prompt()`. Entretanto, utilizar a referência completa vai ajudar você a sempre lembrar a que objeto estão associadas.

Exercícios

1. Faça um script que solicite um ângulo em radianos e converta-o para graus.
2. Faça um script que pergunte ao usuário se deseja converter graus centígrados para kelvin ou vice-versa e, em função da resposta, apresente caixas de diálogo que permitam a entrada de dados e a apresentação dos resultados.
3. Faça um script que utilize caixas de diálogo para solicitar ao usuário as componentes x, y e z de dois vetores e calcule:
   a. o produto escalar $c = a_x b_x + a_y b_y + a_z b_z$
   b. o produto vetorial $c_x = a_y b_z - b_y a_z \quad c_y = a_z b_x - b_z a_x \quad c_z = a_x b_y - b_x a_y$

   apresentando os resultados em uma caixa de diálogo.

3 Controle de fluxo

Neste capítulo você vai aprender como controlar o fluxo de ações do seu programa utilizando perguntas (condições) ou combinações de perguntas que podem ter respostas lógicas do tipo "verdadeiro" ou "falso". Você vai ver como utilizar instruções para criar desvios, bifurcações e múltiplos caminhos alternativos de processamento. Também vai aprender como fazer com que certas ações sejam realizadas um número predefinido de vezes ou enquanto condições bem definidas forem satisfeitas ou não.

Existem essencialmente 5 estruturas utilizadas para controlar o fluxo de um programa:

1. `if (expressão) { instruções } else { instruções }`, conhecida resumidamente como *if/else*, que executa um ou outro grupo de instruções dependendo do valor de uma expressão, que pode ser `true` (verdadeira) ou `false` (falsa).
2. `switch (expressão) case { rótulo: { instruções }; rótulo: { instruções }; ... }`, conhecida resumidamente como *switch/case*", que possibilita a execução de um ou mais grupos de instruções quando uma expressão dada resulta idêntica a um rótulo.
3. `for (inicialização; teste; incremento) { instruções }`, denominada usualmente como *laço for*, que em geral é utilizada para executar um grupo de instruções um número bem definido de vezes.
4. `while (condição) { instruções }`, conhecida como *laço while*, que executa um grupo de instruções enquanto uma condição for verdadeira.
5. `do { instruções } while (condição)`, conhecida como *laço do*, que executa um grupo de instruções pelo menos uma vez e repete sua execução enquanto a condição for verdadeira.

A seguir, uma análise mais detalhada de cada uma delas.

`if (expressão) { instruções } else { instruções }`

O exemplo a seguir pede ao usuário que digite dois números que compõem um ponto (x,y) no plano cartesiano, calcula a sua distância à origem e diz ao usuário se o ponto está dentro de um círculo de raio unitário. Note o uso do `parseFloat()` para converter o valor digitado em um número real.

Listagem:

```
<script>
var x = parseFloat(prompt("Digite um número entre 0 e 1: "));
var y = parseFloat(prompt("Digite outro número entre 0 e 1: "));
var r = Math.sqrt(x*x+y*y);
if (r<1) {
   var msg = "O ponto (" + x + "," + y + ") está DENTRO do círculo";
   alert(msg);
   }
else {
   var msg = "O ponto (" + x + "," + y + ") está FORA do círculo";
   alert(msg);
   }
</script>
```

O exemplo acima assume que se o raio calculado a partir dos valores digitados para x e y for *exatamente* igual a 1, o ponto está *fora* do círculo. Troque o operador menor ("<") pelo operador menor ou igual ("<=") para fazer com que o script considere que, neste caso, o ponto esteja

dentro da circunferência.

O script abaixo faz exatamente a mesma coisa, mas simplifica o bloco de instruções dentro das chaves. Note que quando há apenas uma instrução em cada bloco, o uso das chaves passa a ser opcional.

Listagem:

```
<script>
var x = parseFloat(prompt("Digite um número entre 0 e 1: "));
var y = parseFloat(prompt("Digite outro número entre 0 e 1: "));
var r = Math.sqrt(x*x+y*y);
if (r<1)
   alert("O ponto (" + x + "," + y + ") está DENTRO do círculo");
else
   alert("O ponto (" + x + "," + y + ") está FORA do círculo");
</script>
```

Na estrutura `if/else`, o uso do `else { ... }` é opcional. O exemplo a seguir pede ao usuário que digite um número e verifica se este é negativo. Se for, avisa ao usuário que ele digitou um número negativo e prossegue calculando a raiz quadrada (`Math.sqrt(arg)`) de seu módulo (`Math.abs(arg)`). Note que a função que calcula a raiz e mostra o resultado é sempre executada, independentemente do resultado do teste.

Listagem:

```
<script>
var x = parseFloat(prompt("Entre com um número: " ));
if (x<0) alert("Você digitou um número negativo!");
alert(Math.sqrt(Math.abs(x)));
</script>
```

Sua vez... (3-1)

Modifique o exemplo anterior de modo que continue avisando que o usuário digitou um número negativo quando for o caso, mas que só calcule a raiz quadrada de números maiores ou iguais a zero.

O aninhamento de várias instruções `if ... else` pode ser utilizado para a tomada de decisões que dependem de decisões anteriores. O exemplo a seguir solicita que o usuário digite um par de coordenadas (x,y) em um dos quadrantes do plano cartesiano, sorteado aleatoriamente.

Listagem:

```
<script>
var num = Math.random();
if (num<0.25) quad = "PRIMEIRO";
else if (num<0.5) quad = "SEGUNDO";
else if (num<0.75) quad = "TERCEIRO";
else quad = "QUARTO";
var x = parseFloat(prompt("Digite x no " + quad + " quadrante: "));
var y = parseFloat(prompt("Digite y no " + quad + " quadrante: "));
</script>
```

Na primeira linha, a variável `num` recebe um número aleatório entre 0 e 1. Se o número sorteado estiver entre 0 e 0.25, a variável `quad` recebe o texto "PRIMEIRO"; se estiver entre 0.25 e 0.5, recebe o texto "SEGUNDO"; se estiver entre 0.5 e 0.75, recebe o texto "TERCEIRO"; e se estiver entre 0.75 e 1, recebe o texto "QUARTO". Definido o quadrante, o script pede ao usuário que digite valores para (x,y) no quadrante especificado na variável `quad`.

No exemplo, como cada bloco de instruções consiste sempre de apenas uma instrução foram omitidas todas as chaves delimitadoras e indentação opcionais. A versão abaixo faz exatamente a mesma coisa mas coloca chaves e indentações:

Listagem:

```
<script>
var num = Math.random();
if (num<0.25) {
   quad = "PRIMEIRO";
   }
```

```
else {
   if (num<0.5) {
      quad = "SEGUNDO";
      }
   else {
      if (num<0.75) {
         quad = "TERCEIRO";
      }
      else {
         quad = "QUARTO";
      }
   }
}
var x = parseFloat(prompt("Digite x no " + quad + " quadrante: "));
var y = parseFloat(prompt("Digite y no " + quad + " quadrante: "));
</script>
```

Sua vez.. (3-2)

Utilize `if`'s aninhados para fazer com que no exemplo 3.2 o script tenha uma terceira opção, alertando o usuário que o ponto está SOBRE a circunferência quando o raio calculado for exatamente igual a 1.

As condições são construídas utilizando-se operadores de comparação, sendo os mais comuns o maior (`>`), menor (`<`), maior ou igual (`>=`), menor ou igual (`<=`), igual (`==`) e diferente (`!=`).

> Lembre-se (como visto no capítulo anterior) que o *teste* de igualdade é sempre feito com o uso de *dois* sinais de igual ("==") sucessivos. O uso de apenas um sinal de igual constitui uma instrução válida (atribuição), porém logicamente imprópria nessa situação.

Estas condições podem ser combinadas para produzir condições mais complexas utilizando-se os operadores "E" (`&&`) e "OU" (`||`). O exemplo abaixo solicita dois números e utiliza o operador "E" para verificar em que quadrante se encontram:

Listagem:

```
<script>
var x = parseFloat(prompt("x = "));
var y = parseFloat(prompt("y = "));
if ((x>0)&&(y>0)) alert("1o. quadrante");
if ((x<0)&&(y>0)) alert("2o. quadrante");
if ((x<0)&&(y<0)) alert("3o. quadrante");
if ((x>0)&&(y<0)) alert("4o. quadrante");
</script>
```

O exemplo a seguir verifica se o valor da variável `angulo` é menor que $\pi/2$ ou maior que $3\pi/2$:

Listagem:

```
<script>
var angulo = parseFloat(prompt("ângulo = "));
if ((angulo<Math.PI/2)||(angulo>3/2*Math.PI))
   alert("hemisfério direito");
else
   alert("hemisfério esquerdo");
</script>
```

Muitos testes podem ser feitos na mesma expressão, desde que o resultado seja sempre algo que pode ser interpretado como `false` (falso) ou `true` (verdadeiro). No exemplo a seguir o usuário deve digitar valores para as coordenadas x, y e z que serão utilizadas para calcular a distância a uma carga elementar e o módulo do campo elétrico que ela produz naquele ponto. Se todos os valores forem nulos (`x = y = z = 0`), o valor não pode ser calculado pois haveria um zero no denominador:

Listagem:

```
<script>
var x = parseFloat(prompt("x = "));
var y = parseFloat(prompt("y = "));
var z = parseFloat(prompt("z = "));
if ((x==0)&&(y==0)&&(z==0))
   alert("x, y e z nulos! Operação inválida!");
```

```
else
   alert(9e9 * 1.6e-19 / (x*x+y*y+z*z));
</script>
```

> Coloque sempre parênteses entorno de cada teste e em torno da expressão final. A não delimitação apropriada de cada condição pode levar a erros lógicos muito difíceis de detetar.

`switch (expressão) case { rotulo: { instruções } ... }`

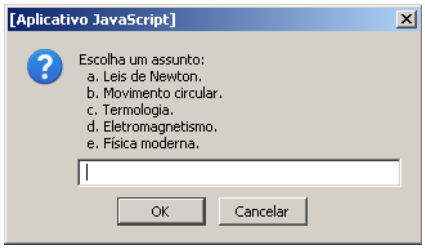
O `switch ... case` permite a execução de um grupo de instruções quando o valor de uma expressão é igual ao rótulo do grupo de instruções, e é geralmente empregado em situações em que um grande número `if`'s aninhados e complexos poderiam ser utilizados.

No exemplo a seguir uma questão é acumulada numa variável literal (`str`) que é utilizada para solicitar que o usuário faça uma escolha. Se a letra digitada estiver entre "a" e "e" frases específicas são apresentadas; para qualquer outra escolha será executado o `default` (padrão).

Listagem:

```
<script>
str = "";
str += "Escolha um assunto:\n";
str += "  a. Leis de Newton.\n";
str += "  b. Movimento circular.\n";
str += "  c. Termologia.\n"
str += "  d. Eletromagnetismo.\n"
str += "  e. Física moderna.\n"
var escolha = prompt(str,"");
switch (escolha) {
   case "a": alert("F = ma\n");
             break;
   case "b": alert("A cos(wt)\n");
             break;
   case "c": alert("Dilatação\n");
             break;
   case "d": alert("Lei de Faraday\n");
             break;
   case "e": alert("Efeito fotoelétrico\n");
             break;
   default : alert("Escolha inválida.\n");
}
</script>
```

Resultado:



> Note novamente o uso de um literal inicialmente vazio (`str = ""`) onde vai sendo acumulada a questão utilizando o operador "+=" (`x += y` é equivalente a `x = x + y`). Note também o uso do caracter de escape `\n` (*newline*), que significa uma quebra de linha dentro do texto para a caixa de alerta (o <br> funciona apenas dentro do documento HTML e a caixa de alerta não é um documento HTML).

Um elemento extremamente importante no `switch ... case` é a instrução `break`. Note que cada um dos blocos é terminado por ela pois aqui se deseja que uma letra defina apenas uma escolha. Se o `break` for removido das instruções, todos os blocos a partir do rótulo escolhido serão executados.

Sua vez... (3-3)

Como exercício, retire do script acima as instruções `break` e execute-o extensivamente, escolhendo todas as diferentes opções para observar os resultados. Experimente também colocar instruções `break` em um ou outro caso e observe o

resultado.

```
for (inicialização; teste; incremento) { instruções }
```

O laço `for` é muito útil quando desejamos realizar cálculos um número bem definido de vezes. O exemplo a seguir calcula a posição de um objeto em MRUV para t indo de 0 a 10 em intervalos de 1 segundo:

Listagem:

```
<script>
var xo = 0;
var vo = 0;
var a  = 10;
for (var t=0; t<=10; t=t+1) {
   x = xo + vo*t + a*t*t/2;
   document.write("<i>t</i> = " + t.toFixed(2) + " ");
   document.write("<i>x</i> = " + x.toFixed(2) + "<br>");
}
</script>
```

Resultado:

$t = 0.00$ $x = 0.00$
$t = 1.00$ $x = 5.00$
$t = 2.00$ $x = 20.00$
$t = 3.00$ $x = 45.00$
$t = 4.00$ $x = 80.00$
$t = 5.00$ $x = 125.00$
$t = 6.00$ $x = 180.00$
$t = 7.00$ $x = 245.00$
$t = 8.00$ $x = 320.00$
$t = 9.00$ $x = 405.00$
$t = 10.00$ $x = 500.00$

Sua vez... (3-4)

Modifique o exemplo anterior de modo que calcule e imprima a posição e a velocidade a cada 0.5 segundos e inclua as unidades das grandezas:

$t = 0.00$ s, $x = 0.00$ m, $v = 0.00$ m/s.
$t = 0.50$ s, $x = 1.25$ m, $v = 5.00$ m/s.
$t = 1.00$ s, $x = 5.00$ m, $v = 10.00$ m/s.
$t = 1.50$ s, $x = 11.25$ m, $v = 15.00$ m/s.
...

Assim como no `if`, no laço `for` as chaves não são obrigatórias caso o bloco de instruções contenha apenas uma instrução.

Laços `for` podem ser aninhados, como no exemplo abaixo, que coloca nos elementos de uma matriz $n \times m$ os valores $a_{i,j} = i + j$.

Listagem:

```
<script>
var n = 3;
var m = 4;
for (var i=1; i<=n; i++) {
   for (var j=1; j<=m; j++) {
      document.write(i+j + " ");
   }
   document.write("<br>");
}
</script>
```

Resultado:

```
2 3 4 5
3 4 5 6
```

```
4 5 6 7
```

Note o uso das contrações `i++` e `j++`, que são equivalentes a `i = i + 1` e `j = j + 1`, respectivamente. Note também o uso de um espaço em branco (`" "`) na instrução `document.write`, utilizado para separar os elementos da matriz.

Sua vez... (3-5)

Modifique o exemplo anterior de modo que passe a imprimir a matriz transposta:

```
2 3 4
3 4 5
4 5 6
5 6 7
```

Apesar de serem em geral utilizados para executar um conjunto de instruções um número definido de vezes, é possível "escapar" do laço em qualquer iteração utilizando o comando `break`. No script a seguir o exemplo do MRUV é modificado para interromper o laço caso a posição do objeto seja maior que 100:

Listagem:

```
<script>
var xo = 0;
var vo = 0;
var a  = 10;
for (var t=0; t<=10; t=t+0.5) {
   x = xo + vo*t + a*t*t/2;
   if (x>100) break;
   document.write("<i>t</i> = " + t.toFixed(2) + " ");
   document.write("<i>x</i> = " + x.toFixed(2) + "<br>");
}
</script>
```

Resultado:

$t = 0.00\ x = 0.00$
$t = 0.50\ x = 1.25$
$t = 1.00\ x = 5.00$
$t = 1.50\ x = 11.25$
$t = 2.00\ x = 20.00$
$t = 2.50\ x = 31.25$
$t = 3.00\ x = 45.00$
$t = 3.50\ x = 61.25$
$t = 4.00\ x = 80.00$

O uso deste tipo de recurso em um laço `for` não é recomendável pois implica em um teste adicional que, como veremos a seguir, pode ser eficientemente substituído por outras estruturas de controle de fluxo.

`while (condição) { instruções }`

O laço `while` é utilizado quando um grupo de instruções deve ser executado um número não pré-definido de vezes, mas enquanto uma condição for verdadeira. O script a seguir calcula as posições de uma partícula em queda livre a partir de uma posição especificada enquanto sua coordenada *y* for maior do que zero:

Listagem:

```
<script>
var yo = parseFloat(prompt("Posição inicial: ","2"));
var t = 0;
var y = yo;
var vo = 0;
var a = -9.8;
var dt = 0.1;
while (y>0) {
   y = yo + vo*t + a*t*t/2;
   document.write("<i>t</i> = " + t.toFixed(2) + " ");
   document.write("<i>y</i> = " + y.toFixed(2) + "<br>");
   t = t + dt;
```

```
   }
</script>
```

Resultado:

```
t = 0.00 y = 2.00
t = 0.10 y = 1.95
t = 0.20 y = 1.80
t = 0.30 y = 1.56
t = 0.40 y = 1.22
t = 0.50 y = 0.77
t = 0.60 y = 0.24
t = 0.70 y = -0.40
```

Note que, como a condição de fim é testada *antes* do cálculo da coordenada, o script imprime o último valor calculado, que é menor que zero, algo que pode ser indesejável em certas situações. Pode também acontecer que o bloco nunca venha a ser executado caso a condição resulte em falsa logo no primeiro teste.

Sua vez... (3-6)

Modifique o exemplo anterior de modo que a partícula saia sempre do solo (y = 0 m), com uma velocidade positiva ($v_0 > 0$) informada pelo usuário, e que interrompa a execução logo após a partícula atingir a altura máxima da sua trajetória. *Dica:* uma das características do máximo da trajetória é a mudança do sinal da velocidade. Assim, se além da posição você calcular também a velocidade, poderá testar se essa é positiva ou negativa. Quando passar a ser negativa, significa que a partícula passou pelo máximo.

do { instruções } while (condição);

Muito semelhante ao laço `while`, o laço `do` também executa o bloco de instruções enquanto a condição for verdadeira, mas o teste é feito *depois* da execução do bloco. Isto significa que o bloco *sempre* será executado pelo menos uma vez. O script a seguir faz o mesmo que o anterior, mas com o laço `do`. Note que se o usuário digitasse um número negativo no exemplo anterior, nenhum resultado seria mostrado, enquanto neste seria mostrada a posição inicial em t = 0.

Listagem:

```
<script>
var yo = parseFloat(prompt("Posição inicial: ","2"));
var t = 0;
var y = yo;
var vo = 0;
var a = -9.8;
var dt = 0.1;
do {
   y = yo + vo*t + a*t*t/2;
   document.write("<i>t</i> = " + t.toFixed(2) + " ");
   document.write("<i>y</i> = " + y.toFixed(2) + "<br>");
   t = t + dt;
   } while (y>0);
</script>
```

Exercícios

1. Utilize laços `while`, `do ... while` e `for` para fazer scripts que calculem e mostrem os números inteiros entre 0 e 10, inclusive.

2. A equação de um oscilador harmônico amortecido com constante elástica k, massa m e constante de amortecimento b é:

$$x(t) = A \cos(2\pi f t)\, e^{-(b/2m)t}$$

onde A é a amplitude e $f = [k/m - (b/2m)^2]^{1/2}$ a frequência. Utilize laços `while`, `do ... while` e `for` para fazer scripts que calculem a posição do oscilador em função do tempo para 20 pontos entre 0 e 2 segundos, em intervalos de 0.1 segundos, para A = 0.10 m, f = 2

Hz e $b/2m = 2$.

3. Utilize laços `while` e `do ... while` para fazer um script que solicite ao usuário a altura em que um objeto é liberado em queda livre, a partir do repouso, e calcule posições sucessivas em intervalos de 0.1 s até que atinja o solo.

4. Faça um script que solicite ao usuário que escolha uma função dentre uma lista (seno, cosseno, tangente, raiz quadrada, exponencial, por exemplo) e a seguir um número que servirá de argumento. Utilize a estrutura `switch ... case` para realizar e apresentar o valor do cálculo da função para o argumento dado.

5. Refaça o exercício anterior utilizando `if ... else` aninhados.

4 Matrizes Unidimensionais

Neste capítulo você vai aprender como utilizar matrizes, também conhecidas como vetores, que são estruturas apropriadas para armazenar coleções de dados do mesmo tipo que são em geral processados da mesma maneira. Você também vai aprender como utilizar alguns métodos específicos para a manipulação de matrizes, utilizados para selecionar subgrupos de dados, unir informações e ordenar seus elementos, entre outras.

Matrizes são estruturas de dados que agregam muitas informações sob um mesmo identificador. As diferentes informações podem ser acessadas através de um índice que as seleciona. O exemplo abaixo cria duas matrizes unidimensionais, guarda nelas as posições x e y de 4 partículas em um plano e imprime os pontos no documento:

Listagem:

```
<script>
var x = [1,3,5,7];
var y = [-3,-1,1,3];
document.write(x[0]+ "," + y[0] + "<br>");
document.write(x[1]+ "," + y[1] + "<br>");
document.write(x[2]+ "," + y[2] + "<br>");
document.write(x[3]+ "," + y[3] + "<br>");
```

Resultado:

1,-3
3,-1
5,1
7,3

Em JavaScript, assim como em C, C++ e Java, o primeiro elemento de uma matriz tem índice 0, diferentemente do que acontece em FORTRAN (linguagem ainda muito utilizada em alguns campos de pesquisa em física), cujo primeiro elemento de uma matriz tem índice 1.

O mesmo resultado seria obtido primeiro antecipando que as variáveis `x` e `y` são matrizes (de tamanho indefinido) e depois atribuindo valores a cada um dos seus elementos individualmente:

Listagem:

```
<script>
var x = [];
var y = [];
x[0] = 1; y[0] = -3;
x[1] = 3; y[1] = -1;
x[2] = 5; y[2] =  1;
x[3] = 7; y[3] =  3;
for (var i=0; i<x.length; i++) {
   document.write(x[i] + "," + y[i] + "<br>");
}
</script>
```

No exemplo acima, a propriedade `length` (comprimento) da matriz foi utilizada. No caso, `x.length` vale 4, e o laço `for` é executado 4 vezes, para `i` valendo 0, 1, 2, e 3. Esta propriedade torna-se particularmente importante quando queremos colocar no script matrizes de tamanhos

variáveis, pois não precisamos nos preocupar em tomar conta disto.

`var x = new Array(1,3,5,7)` também é uma maneira de criar e inicializar uma matriz, sendo equivalente a `var x = [1,3,5,7]`. Quando apenas um argumento numérico `n` é utilizado, por exemplo em `var x = new Array(n)`, é criada uma matriz com `n` elementos, todos indefinidos. Quando nenhum argumento é utilizado, por exemplo em `var x = new Array()`, é criada uma matriz de tamanho indefinido, sendo equivalente a `var x = []`.

Sua vez... (4-1)

Modifique o exemplo anterior de modo que inclua uma terceira matriz `z`, correspondente à terceira coordenada espacial, inicializada com 2, 4, 6, 8. O script também deve modificar a forma de imprimir os resultados, agrupando as coordenadas em parênteses:

```
(1,-3,2)
(3,-1,4)
(5,1,6)
(7,3,8)
```

O exemplo a seguir utiliza matrizes para calcular a posição do centro de massa de um sistema unidimensional com 5 partículas posicionadas em diferentes pontos ao longo do eixo x. Para sistemas discretos, a coordenada x_{CM} do centro de massa é dada por:

x_{CM} = \frac{x_1 m_1 + x_2 m_2 + \dots + x_N m_N} {m_1 + m_2 + \dots + m_N} = \frac{\sum_{i=1}^{N} x_i m_i}{\sum_{i=1}^{N} m_i}

onde `x_i` e `m_i` são a coordenada e a massa de cada partícula.

Listagem:

```
<script>
var m = [3,2,1,4,5];
var x = [1,3,2,4,7];
var smx = 0;
var sm = 0;
for (i=0;i<m.length;i++) {
   smx = smx + x[i]*m[i];
   sm += m[i];
}
var xcm = smx/sm;
document.write(xcm.toFixed(2));
</script>
```

Resultado:

```
4.13
```

O laço `for` acumula a soma do produto da massa pela posição de cada partícula e a soma de suas massas. Um erro muito comum é esquecer de inicializar com zero os acumuladores, como são chamadas variáveis como `smx` e `sx` definidas neste exemplo.

A instrução `sm += m[i]` é uma abreviação para a instrução `sm = sm + m[i]`. Estruturas semelhantes podem ser construídas com os outros operadores aritméticos, como por exemplo `a *= b` para representar `a = a * b`.

Sua vez... (4-2)

Modifique o exemplo anterior de modo que inclua valores para a coordenada `y` de cada partícula e calcule também a coordenada `ycm` do centro de massa.

O exemplo a seguir calcula o produto escalar **a** · **b** entre os vetores **a** e **b**, que é dado por:

$$c = a_1 b_1 + a_2 b_2 + a_3 b_3$$

onde os índices 1, 2 e 3 representam as coordenadas x, y e z de cada vetor.

Listagem:

```
<script>
var a = [3,2,1];
var b = [1,2,3];

var prEsc = 0;

for (i=0;i<3;i++) {
   prEsc = prEsc + a[i] * b[i];
}

document.write("<b>a</b> &middot; <b>b</b> = ");
document.write(prEsc.toFixed(2));
</script>
```

Resultado:

a · **b** = 10.00

Neste exemplo as novidades estão relacionadas à formatação HTML, que vamos explorar em detalhes mais adiante. Os marcadores `<b>...</b>` delimitam regiões do texto que aparecerão em **negrito**; já a sequência `&middot;` apresenta o símbolo "·", um dos muitos caracteres especiais disponíveis no HTML. Para adiantar, em matemática os mais comuns são `&times;`, que mostra "×", `&lt;`, que mostra "<", `&le;`, que mostra "≤", `&gt;`, que mostra ">" e `&ge;`, que mostra "≥".

Note que tanto no exemplo do cálculo de centro de massa, apesar de na fórmula terem sido utilizados índices de 1 a N, nos scripts foram utilizados índices de 0 a $N-1$.

O objeto `Array` dispõe de alguns métodos que podem ser úteis.

concat

O método `concat` retorna uma matriz que é a combinação de duas matrizes, "pendurando" (ou concatenando) uma à outra:

Listagem:

```
<script>
var quarks = ["up","down","charm","strange","bottom","top"];
var leptons = ["electron","&tau;","muon",
               "eletron-neutrino","tau-neutrino","muon-neutrino"];
document.write(leptons.concat(quarks));
document.write("<br><br>");
var fermions = leptons.concat(quarks);
for (var i=0;i<fermions.length;i++) document.write(fermions[i] + " ");
</script>
```

Resultado:

electron,tau,muon,eletron-neutrino,tau-neutrino,muon-neutrino,up,down,charm,strange,bottom,top

electron tau muon eletron-neutrino tau-neutrino muon-neutrino up down charm strange bottom top

A instrução `leptons.concat(quarks)` apensa `quarks` a `leptons` e retorna uma terceira matriz que não é armazenada, mas apenas impressa em bloco, com os elementos automaticamente separados por vírgulas. A instrução `var fermions = leptons.concat(quarks)` apensa a matriz `leptons` à matriz `quarks`, guardando o resultado em uma terceira matriz `fermions`. A instrução seguinte imprime os elementos de `fermions` um a um, separando-os por espaços em branco.

Note o uso da instrução `document.write("<br><br>")`, que faz uso de dois elementos HTML <br> para "pular" duas linhas no texto. O uso de elementos CSS (*Cascade Style Sheets*), que serão vistos superficialmente neste texto, pode substituir de maneira mais prática e elegante esse tipo de recurso. A internet é rica em materiais sobre o tema: informe-se!

Sua vez... (4-3)

Modifique o exemplo anterior de modo que sejam criadas duas matrizes unidimensionais `aditivas` e `subtrativas`, a primeira inicializada com os valores "vermelho", "verde" e "azul" e a segunda inicializada com os valores "ciano", "amarelo" e "magenta", e crie e imprima a matriz `cores` que contém a concatenação das duas

anteriores.

---

join

O método `join` retorna um literal que é a junção de todos os elementos da matriz. O método pode receber como parâmetro um literal que separará os elementos no literal resultante. No exemplo a seguir este parâmetro é um sinal de adição com espaços em branco antes e depois:

Listagem:

```
<script>
var mat = [1,3,5,7,9];
var soma = 0;
for (var i=0;i<mat.length;i++) soma += mat[i];
var str = mat.join(" + ") + " = " + soma;
document.write(str);
</script>
```

Resultado:

```
1 + 3 + 5 + 7 + 9 = 25
```

Sua vez.. (4-4)

Modifique o exemplo anterior de modo que apresente uma expressão para o produto dos elementos da matriz (1 * 3 * 5 * 7 * 9 = 945).

---

reverse

O método `reverse` inverte os elementos de uma matriz:

Listagem:

```
<script>
var mat1 = [1,3,5,7,9];
var mat2 = mat1.reverse();
for (var i in mat2) document.write(mat2[i] + " ");
</script>
```

Resultado:

```
9 7 5 3 1
```

Note o uso da expressão `for (var i in mat2) { ... }` que é uma alternativa ao uso de `for (var i=0;i<mat2.length;i++) { ... }` para percorrer os elementos de uma matriz, particularmente quando nem todos estão definidos.

slice

O método `slice` retorna um pedaço de uma matriz. Recebe dois parâmetros, um obrigatório (o índice do elemento que inicia a sequência a ser extraída) e outro opcional (o índice do elemento que sucede o último elemento a ser extraído), ambos com a contagem iniciada em 0 (zero). Caso o elemento final não seja fornecido, a extração se dá até o final da matriz. No exemplo a seguir, os índices inicial e final são passados quando se deseja extrair as variáveis associadas às coordenadas retangulares, mas apenas o índice inicial é passado quando se deseja extrair as variáveis associadas às coordenadas esféricas:

Listagem:

```
<script>
var coords = ["x","y","z","r","&theta;","&phi;"];
var rets = coords.slice(0,2);
var esfs = coords.slice(3);
document.write(rets)
document.write("<br>");
document.write(esfs);
</script>
```

Resultado:

```
x,y
r,θ,φ
```

Note o uso das *entidades* HTML `&theta;` e `&phi;` para produzir as letras gregas θ e φ. Faça uma busca por "HTML entities" na internet para ver a enorme gama de símbolos que podem ser produzidos desta (e de outras) maneiras.

Se o segundo parâmetro for negativo, representa um deslocamento a partir do final da matriz. No exemplo a seguir, todos os elementos de uma matriz são copiados para outra, exceto os dois últimos:

Listagem:

```
<script>
var mat1 = [1,3,5,7,9];
var mat2 = mat1.slice(0,-2);
document.write(mat2);
</script>
```

Resultado:

```
1,3,5
```

Sua vez... (4-5)

Modifique o exemplo anterior de modo que extraia e imprima os elementos intermediários (3,5,7) da matriz.

sort

O método `sort` ordena uma matriz:

Listagem:

```
<script>
var mat1 = ["u","d","s","c","t","b"];
var mat2 = mat1.sort();
document.write(mat2);
</script>
```

Resultado:

```
b,c,d,s,t,u
```

O método `sort`, como alguns outros, não necessariamente precisa colocar o resultado em outra matriz, podendo ser utilizado para ordenar uma matriz e deixar o resultado nela mesma:

Listagem:

```
<script>
var mat1 = ["u","d","s","c","t","b"];
mat1.sort();
document.write(mat1);
</script>
```

Resultado:

```
b,c,d,s,t,u
```

Sua vez... (4-6)

Faça um script que defina uma matriz com os 5 primeiros números pares e outra

com os 5 primeiros números ímpares. A seguir, o script deve utilizar o método `concat()` para combinar as duas matrizes, o método `sort()` para ordená-las em ordem crescente e a instrução `document.write()` para imprimir a matriz resultante.

O método `sort()` sempre interpreta as informações armazenadas em uma matriz como texto e nunca como números, o que pode levar a resultados indesejáveis. Experimente criar e ordenar duas matrizes, por exemplo `var mat1 = [1,9,11,20,22,99])` e `var mat2 = ["1","9","11","20","22","99"]` e aplicar o método `sort()` em ambas e observe os resultados. Para mais detalhes e sugestões de usos, pesquise algo como "sort javascript" na internet.

Exercícios

1. Faça um script que defina uma matriz com a velocidade de um objeto em diferentes instantes, preencha seus elementos e compute e apresente a média entre eles:

   $$v_{med} = (1/N)\,\Sigma_i\, v_i$$

2. Modifique o script anterior para obter também o desvio quadrático médio da velocidade, dado por:

   $$\delta_{rms}(v,v_{med}) = (1/N)\,[\,\Sigma_i\,(v_i - v_{med})^2\,]^{1/2}$$

3. Dados dois vetores:

   $$\mathbf{a} = a_x\,\mathbf{i} + a_y\,\mathbf{j} + a_z\,\mathbf{k} \quad \text{e} \quad \mathbf{b} = b_x\,\mathbf{i} + b_y\,\mathbf{j} + b_z\,\mathbf{k}$$

   o vetor soma **c** é dado por:

   $$\mathbf{c} = (a_x + b_x)\,\mathbf{i} + (a_y + b_y)\,\mathbf{j} + (a_z + b_z)\,\mathbf{k}$$

   Faça um script que utilize matrizes para representar e somar dois vetores, armazenando o resultado da soma em uma terceira matriz.

4. O produto vetorial é dado por:

   $$c_x = a_y\,b_z - b_y\,a_z \qquad c_y = a_z\,b_x - b_z\,a_x \qquad c_z = a_x\,b_y - b_x\,a_y$$

   Faça um script que utilize matrizes para realizar o produto vetorial entre dois vetores, armazenando o resultado em uma terceira matriz.

5. Faça um script que percorra uma matriz com N números inteiros e copie para outra matriz apenas os números ímpares.

6. Faça um script que preencha uma matriz com dez números aleatórios e imprima-os em ordem crescente.

7. Faça um script que preencha uma matriz com uma lista de 20 palavras relacionadas a ciência e tecnologia, coloque-as em ordem alfabética e imprima somente aquelas cuja primeira letra está entre o j (jota) e o n (ene), inclusive.

5 Matrizes Multidimensionais

Neste capítulo você vai aprender como criar, preencher e utilizar matrizes com duas ou mais dimensões. Matrizes multidimensionais são ideais para armazenar e processar diferentes informações associadas a uma mesma entidade, facilitando a criação e visualização de combinações e correlações entre elas.

As linguagens de programação em geral permitem a criação de matrizes com duas ou mais dimensões, ou seja, estruturas cujos elementos possuem o mesmo nome mas são acessados através do uso de dois ou mais índices. Um jogo de batalha naval, o plano cartesiano e uma planilha de notas de um conjunto de alunos podem ser pensados como estruturas de dados bidimensionais; o potencial elétrico em uma região do espaço pode ser armazenado em uma estrutura tridimensional, onde cada dimensão corresponde a uma coordenada espacial.

Em JavaScript, matrizes multidimensionais são criadas em etapas. O código a seguir ilustra a criação de uma estrutura bidimensional com 3 elementos em uma dimensão e 4 na outra, a cada elemento sendo atribuído o valor *i+j*, para *i* variando de 0 a 2 e *j* variando de 0 a 3:

Listagem:

```
<script>
// primeiro bloco: cria e preenche a matriz 2d
var m2d = [];
for (var i=0;i<3;i++) {
    m2d[i] = [];
    for (var j=0;j<4;j++) {
        m2d[i][j] = i + j;
    }
}
// segundo bloco: imprime a matriz 2d
for (var i=0;i<3;i++) {
    for (var j=0;j<4;j++) {
        document.write(m2d[i][j] + " ");
    }
    document.write("<br>");
}
</script>
```

Resultado:

```
0 1 2 3
1 2 3 4
2 3 4 5
```

Na primeira linha do primeiro bloco do script é criada a matriz unidimensional `m2d`, sem tamanho definido. Dentro do primeiro laço `for`, em cada *i*-ésimo elemento da matriz unidimensional é criada uma nova matriz unidmensional (`m2d[i] = []`) e em um novo laço `for` são atribuídos os valores `i + j` aos elementos da matriz. Um segundo bloco é criado exclusivamente para imprimir os elementos da matriz em uma formatação apropriada. Note o uso de um espaço em branco após cada elemento e de uma quebra de linha (`"<br>"`) após cada ciclo da variável `j`, ou seja, após cada linha.

No exemplo anterior, a criação, preenchimento e impressão dos elementos da matriz poderiam ser feitos em um único conjuntos de laços `for` aninhados. Entretanto, para facilitar reaproveitamento de códigos, em geral é uma boa prática isolar as operações de entrada e saída de dados das operações lógicas e de cálculo (exceto quando a minimização do tempo de execução for uma prioridade).

Sua vez... (5-1)

Modifique o exemplo anterior de modo produza uma matriz 4×3 cujos elementos são definidos pelo produto dos índices *i* e *j*.

O exemplo a seguir mostra como utilizar matrizes para, por exemplo, calcular a média das notas de alguns alunos a partir de notas obtidas em avaliações parciais. Note, em particular, que todos os campos de cada linha são inicializados na mesma instrução e que, em JavaScript, uma matriz pode conter tipos diferentes de elementos, no caso literais (os nomes dos alunos) e números (as notas).

Listagem:

```
<script>

// define matriz para nomes e notas parciais
var a = [];

// inicializa matriz com nomes e notas
a[0] = ["Ana",   7.5, 5.0, 9.0];
a[1] = ["Júlio", 9.0, 7.0, 8.0];
a[2] = ["Paula", 8.0, 6.5, 7.0];
a[3] = ["Rita",  8.5, 6.0, 8.5];

// cria e inicializa com zeros a matriz de médias
var m = [0,0,0,0];

// calcula as médias
for (var i=0;i<4;i++) {
   for (var j=1; j<=3; j++) {
      m[i] = m[i] + a[i][j];
   }
   m[i] = m[i] / 3;
}

// imprime resultados em modo pre-formatado
document.write("<pre>");

for (var i=0;i<4;i++)
   document.write(a[i][0] + "\t" + m[i].toFixed(2) + "\n");

document.write("</pre>");

</script>
```

Resultado:

```
Ana 7.17
Júlio 8.00
Paula 7.17
Rita 7.67
```

As últimas linhas do script ilustram mais uma característica do HTML: a primeira linha do último bloco chama o método `write()` do objeto `document`, passando como argumento o elemento `<pre>` do HTML; a última linha passa como argumento o elemento `</pre>`. Esses elementos demarcam o início e o fim de uma região do documento onde o texto será impresso com fonte de tamanho fixo (em que o "i" ocupa o mesmo espaço de um "m"), onde os espaços em branco adicionais não serão ignorados e onde certas combinações de caracteres, tais como a tabulação (`\t`) e a nova linha (`\n`) serão interpretados como sequências de escape. Esta estratégia é interessante quando se quer tentar tabular informações sem de fato utilizar uma tabela HTML (que gera resultados de qualidade editorial muito superior, mas que pode ser trabalhoso para programar).

Sua vez... (5-2)

Para ter mais clareza do que isto significa, experimente comentar as duas linhas que imprimem o `<pre>` e o `</pre>` e veja o resultado. Experimente também aumentar o número de espaços em branco entre as aspas, o número de tabulações e de quebras de linhas com essas linhas comentadas ou não. Para finalizar, faça uma busca na internet sobre as propriedades do elemento `pre` do HTML.

Matrizes, no sentido computacional, são obviamente bastante apropriadas para a realização de operacões matriciais no sentido matemático. Cálculos com matrizes são frequentemente

necessários para a solução de problemas em física e engenharia (resolução de sistemas lineares, transformações de coordenadas etc.). O script a seguir realiza o produto de uma matriz **A** com m_A linhas e n_A colunas com uma matriz **B** com m_B linhas e n_B colunas fazendo a soma do produto de cada elemento das linhas da matriz **A** com cada elemento das colunas da matriz **B**, resultando na matriz **C** = **A** · **B**:

$$c_{ij} = a_{i1} b_{1j} + a_{i2} b_{2j} + ... + a_{ik} b_{kj} + ... + a_{i n_A} b_{n_A j}$$

Listagem:

```
<script>
// matriz A
var mA = 3; // número de linhas
var nA = 2; // número de colunas
var matA= [];
for (i=0;i<mA;i++) matA[i] = [];
matA[0][0] = 1;  matA[0][1] = 0;
matA[1][0] = 1;  matA[1][1] = 1;
matA[2][0] = 0;  matA[2][1] = 1;

// matriz B
var mB = 2; // número de linhas
var nB = 3; // número de colunas
var matB = [];
for (i=0;i<mB;i++) matB[i] = [];
matB[0][0] = -1;  matB[0][1] = 0;  matB[0][2] = -1;
matB[1][0] = -3;  matB[1][1] = 1;  matB[1][2] = -2;

// matriz C
var mC = mA; // número de linhas
var nC = nB; // número de colunas
var matC = [];
for (i=0;i<mC;i++) matC[i] = [];
for (i=0;i<mC;i++) for (j=0;j<nC;j++) matC[i][j] = 0;

// produto das matrizes
for (i=0;i<mA;i++)
   for (j=0;j<nB;j++)
      for (k=0;k<nA;k++)
          matC[i][j] = matC[i][j] + matA[i][k]*matB[k][j];

// imprime a matriz resultante no modo preformatado
document.write("<pre>");
for (i=0;i<mA;i++) {
    for (j=0;j<nB;j++) document.write(matC[i][j] + "    ");
    document.write("<br>");
    }
document.write("</pre>");
</script>
```

Resultado:

```
-1 0 -1
-4 1 -3
-3 1 -2
```

Todos os elementos da matriz resposta **C** foram inicializados com zero em uma instrução que tem dois laços `for` aninhados (um dentro do outro). Isto é necessário porque na estrutura seguinte, que faz o produto das matrizes, os elementos da matriz serão utilizados como acumuladores e, como tal, precisam ter um valor inicial.

No trecho do scritp que faz o produto das matrizes existem três laços `for` aninhados: um para percorrer as linhas da primeira matriz, outro para percorrer as colunas da segunda matriz e um terceiro que faz a soma dos produtos dos elementos das linhas pelos elementos das colunas. Note que, por conterem apenas uma instrução em cada bloco, as `{...}` (chaves) não são necessárias.

A impressão da matriz resultante é feita utilizando dois laços `for` aninhados: um para as linhas e outro para as colunas. O primeiro tem duas instruções no bloco, portanto as chaves são obrigatórias; o segundo só tem uma instrução, dispensando as chaves.

Essa implementação de um simples produto de duas matrizes é relativamente longa e intrincada. Algumas linguagens mais dedicadas a aplicações científicas e tecnológicas têm instruções nativas que simplificam enormemente esse tipo de operação. JavaScript não tem, mas é possível encontrar na internet muitas bibliotecas em código aberto que, incorporadas ao seu programa com a inclusão de apenas uma linha, fazem o mesmo serviço.

Sua vez... (5-3)

Modifique o exemplo anterior para apresentar o produto da matriz **B** pela matriz **A**, que resulta em uma matriz 2×2. Incremente o seu exercício fazendo com que os elementos das matrizes sejam inicializados com funções dos índices (tipo $i^2 + j^2$ ou qualquer outra que você quiser inventar.

Matrizes com mais de duas dimensões são criadas propagando o método apresentado. Para criar uma matriz de três dimensões, cria-se os elementos da primeira dimensão; a seguir cria-se neles os elementos da segunda dimensão e finalmente cria-se nesses os da terceira dimensão.

Considere, por exemplo, duas estantes, cada uma com duas prateleiras e cada prateleira com quatro livros. A localização de cada livro pode ser dada por um código do tipo "124", indicando a primeira estante, a segunda prateleira e o quarto livro. Uma maneira de criar uma matriz tridimensional para guardar os nomes dos livros associados à sua localização seria:

Listagem:

```
var nEst = 2;
var nPra = 2;
var nLiv = 4;
var Bibl = [];
for (var i=0; i<nEst; i++) {
   Bibl[i] = [];
   for (var j=0; j<nPra;j++) {
      Bibl[i][j] = [];
   }
}
```

Preencher a matriz com os nomes dos livros seria trabalhoso, mas nada difícil, utilizando agora 3 índices:

Listagem:

```
Bibl[0][0][0] = "Livro 1";   Bibl[1][0][0] = "Livro 9";
Bibl[0][0][1] = "Livro 2";   Bibl[1][0][1] = "Livro 10";
Bibl[0][0][2] = "Livro 3";   Bibl[1][0][2] = "Livro 11";
Bibl[0][0][3] = "Livro 4";   Bibl[1][0][3] = "Livro 12";
Bibl[0][1][0] = "Livro 5";   Bibl[1][1][0] = "Livro 13";
Bibl[0][1][1] = "Livro 6";   Bibl[1][1][1] = "Livro 14";
Bibl[0][1][2] = "Livro 7";   Bibl[1][1][2] = "Livro 15";
Bibl[0][1][3] = "Livro 8";   Bibl[1][1][3] = "Livro 16";
```

A recuperação da informação é feita da mesma maneira. Por exemplo, para imprimir os nomes dos livros introduzindo indicações sobre a estante e a prateleira em que se encontram:

Listagem:

```
for (var i=0; i<nEst; i++) {
   document.write("Estante " + (i+1) + ":<br>");
   for (var j=0; j<nPra; j++) {
      document.write("Prateleira " + (j+1) + ":");
      for (var k=0; k<nLiv; k++) {
         document.write(" " + Bibl[i][j][k]);
      }
      document.write("<br>");
   }
   document.write("<br><br>");
}
```

Resultado:

```
Estante 1:
Prateleira 1:   Livro 1   Livro 2   Livro 3   Livro 4
Prateleira 2:   Livro 5   Livro 6   Livro 7   Livro 8

Estante 2:
Prateleira 1:   Livro 9   Livro 10   Livro 11   Livro 12
Prateleira 2:   Livro 13   Livro 14   Livro 15   Livro 16
```

Sua vez.. (5-4)

Modifique o exemplo acima de modo que, ao invés de atribuir os nomes dos

livros elemento a elemento individualmente, faça-o a partir de uma lista armazenada em uma variável tipo:

```
var livros = ["Livro 1", "Livro 2", ... , "Livro N"]
```

utilizando laços `for` que controlem a sua distribuição com uma instrução do tipo:

```
Bibl[i][j][k] = livros[f(i,j,k)];
```

onde `f(i,j,k)` é uma combinação linear dos índices da distribuição `Bibl[i][j][k]` ($a \cdot$`i` $+ b \cdot$`j` $+ c \cdot$`k`) que gera um novo índice para recuperação da informação no vetor unidimensional `livros`.

É possível construir matrizes multidimensionais mais complexas. Por exemplo, considere uma estrutura de dados construída para armazenar informações sobre partículas emergindo de uma colisão num acelerador. Pode ser interessante armazenar, para cada partícula, a massa, a carga e as três componentes do momento linear. O exemplo a seguir mostra uma possível maneira de implementar esta estrutura para 5 partículas:

Listagem:

```
var nPart = 5;
var particula = [];

// para cada partícula...
for (var i=0; i<nPart; i++) {
   // massa, carga e vetor momento
   particula[i] = [];
   // para o 3o. elemento, nova matriz (px,py,pz)
   particula[i][2] = [];
}

// preenchendo a matriz:
particula[0][0] = ...; // massa
particula[0][1] = ...; // carga
particula[0][2][0] = ...; // px
particula[0][2][1] = ...; // py
particula[0][2][2] = ...; // pz
...
```

Note que neste exemplo a matriz tem 5 linhas (que indexam as partículas) e 3 colunas (massa, carga e momento), mas somente o terceiro elemento (momento) possui profundidade. A matriz é, portanto, parcialmente bidimensional e parcialmente tridimensional.

Sua vez... (5-5)

Modifique o exemplo acima de modo que use o método `Math.random()` para atribuir valores para a massa e a carga das partículas escolhidas aleatoriamente entre a massa do elétron e do próton (9.1×10^{-31} kg e 1.7×10^{-27} kg) e suas respectivas cargas (-1.6×10^{-19} C e $+1.6 \times 10^{-19}$ C). Para cada componente do momento, seu programa deve atribuir números aleatórios entre 0 e 1. Para imprimir os resultados, o seu programa deve utilizar uma estrutura do tipo:

```
for (var i=0; i<nPart; i++) {
    document.write(particula[i]);
    document.write("<br>");
}
```

que deve produzir uma listagem como a que segue:

```
9.1e-31,-1.6e-19,0.9,0.1,0.4
1.7e-27,1.6e-19,0.3,0.2,1.0
9.1e-31,-1.6e-19,0.5,0.2,0.4
9.1e-31,-1.6e-19,0.4,0.7,0.4
1.7e-27,1.6e-19,0.6,0.4,0.6
```

Exercícios

1. Faça um script que crie e defina uma matriz bidimensional $n \times m$ cujos elementos são definidos por $A_{i,j} = (i^2 + j^2)^{1/2}$ onde $i = 1, ..., n$ e $j = 1, ..., m$. Para $n = 2$ e $m = 3$, o resultado deve ser formatado como apresentado a seguir:

```
1.41  2.24  3.16
```

```
2.24  2.83  3.61
```

2. Faça um script que crie uma matriz quadrada $n \times n$ cujos elementos são definidos por $A_{i,j} = 2i + j$ onde $i,j = 1, ..., n$, e imprima o valor da soma dos elementos da diagonal da matriz. Para $n = 3$ uma possível formatação da resposta seria:

```
Dada a matriz:

3 4 5
5 6 7
7 8 9

A soma dos elementos da diagonal é 18.
```

3. Faça um script que reproduza o seguinte quadro inicial de uma "Batalha Naval", onde os "–" significam água os "X" significam navios:

```
X - - - - - - -
X - - - - - - X
X - X X X - - X
- - - - - - - X
- - - - - - - -
- - - X X X - X
- - - - - - - X
- X X X - - - X
```

4. Modifique o script que calcula a média dos alunos de modo que ele:

   a. atribua pesos diferentes para as avaliações.

   b. imprima as notas e a média dos alunos.

   c. imprima a média de cada aluno e, em seguida, a menção "Aprovado" se a média for maior ou igual a 6.00; caso contrário, imprima "Reprovado".

5. Modifique o script do produto de matrizes para que:

   a. faça a soma de duas matrizes.

   b. imprima e a matriz transposta de uma matriz dada.

   c. encontre o determinante de uma matriz dada.

6. Termine de desenvolver o último exemplo do texto de modo que sejam atribuídas aleatoriamente as massas, cargas e componentes do momento das partículas. Faça o script de tal modo que existam partículas de massa $m = 1$ e $m = 2$ com igual probabilidade mas que partículas com carga $q = 1$ sejam 5 vezes mais prováveis que partículas com carga $q = 2$. O script deve atribuir componentes p_x, p_y e p_z entre 0 e 1. O resultado final deve ser formatado de modo a produzir uma listagem que se assemelhe à que segue:

```
Partícula 0: 2, 1, 0.39, 0.20, 0.19
Partícula 1: 1, 1, 0.56, 0.67, 0.91
Partícula 2: 2, 1, 0.89, 0.13, 0.44
Partícula 3: 1, 1, 0.81, 0.27, 0.81
Partícula 4: 2, 1, 0.51, 0.02, 0.21
...
```

7. Modifique o script do problema anterior de modo que as partículas continuem tendo valores aleatórios entre 0 e 1 mas que tenham momento total sempre igual à unidade ($|\mathbf{p}| = 1$).

6 Funções

Neste capítulo você vai aprender a criar e utilizar suas próprias funções, que são subprogramas que executam tarefas específicas repetidas vezes ou em diferentes lugares do seu programa principal. Você vai aprender como passar parâmetros para as funções e como fazer para que elas devolvam um ou mais resultados de seus cálculos em variáveis simples ou estruturadas.

Como todas as linguagens de programação, o JavaScript permite ao usuário definir as suas próprias funções. Em geral as funções aglutinam sob um mesmo nome uma série de ações, que podem ser cálculos, movimentações de dados, comparações lógicas, desenhos etc. Funções podem ou não receber parâmetros (também chamados argumentos) e também podem ou não retornar resultados.

A definição de uma função é feita com a instrução `function`, seguida da lista de parâmetros entre parênteses e do bloco de instruções que deve executar, entre chaves. O exemplo a seguir define uma função que calcula o valor da força elétrica entre duas cargas e utiliza a função para construir uma tabela com os valores da força para cargas de diferentes valores separadas por diferentes distâncias.

Listagem:

```
<script>
function forcaEletrica(q1,q2,r)
/*
   Função que recebe como parâmetros as cargas
   de duas partículas, em unidades da carga elementar,
   e a distância entre elas, em nanometros,
   e retorna a força entre elas, em piconewtons
*/
{
   var k = 8.99e+9;  // constante elétrica, em N*m2/C2
   var e = 1.60e-19; // carga elétrica, em C
   var nm = 1e-9;    // 1 nanometro
   var f = (k*e*e/nm/nm) * (q1 * q2) / (r*r);
   return f * 1e12;  // força em piconewtons
}

var str = "<pre>";
var cga1 = 1;
str += "\t";
for (dist=1; dist<=10; dist=dist+2) str += "d=" + dist + "\t";
str += "\n";
for (cga2=1; cga2<10; cga2=cga2+2) {
   str += "q2=" + cga2 + "\t";
   for (dist=1; dist<=10; dist=dist+2) {
      str += forcaEletrica(cga1,cga2,dist).toFixed(0) + "\t";
   }
   str += "\n";
}
str += "</pre>";
document.write(str);
</script>
```

Resultado:

```
 d=1 d=3 d=5 d=7 d=9
q2=1 230 26 9 5 3
q2=3 690 77 28 14 9
q2=5 1151 128 46 23 14
q2=7 1611 179 64 33 20
q2=9 2071 230 83 42 26
```

A função implementa a Lei de Coulomb, que diz que a força entre duas cargas é proporcional ao produto das cargas e inversamente proporcional ao quadrado da distância entre elas:

F = k \frac{q_1 q_2}{r^2}

Na fórmula, $k = 8.99 \times 10^9$ N/C^2/m^2 é a constante da força elétrica, q_1 e q_2 as cargas elétricas em coulombs e r é a distância, medida em metros. Por conveniência, a função modifica um pouco as unidades tradicionais. Na escala atômica as cargas são muito pequenas comparadas com o coulomb e é mais conveniente referir-se a elas em termos da carga elementar, que é a carga do elétron ou do próton, cujo valor é $e = 1.60 \times 10^{-19}$ C. As distâncias também são muito pequenas comparadas ao metro e são usualmente especificadas em nanometros (1 nm = 10^{-9} m) que ainda assim é 10 a 20 vezes maior que o diâmetro atômico. As forças, consequentemente, são muito pequenas e podem ser expressas em piconewtons (1 pN = 10^{-12} N).

O script contém um comentário com várias linhas delimitado por `/* ... */` que explica o que faz a função, que parâmetros recebe, que valor retorna e as unidades que utiliza. O hábito de comentar o código simplifica enormemente o problema da manutenção a longo prazo e mesmo de sua compreensão pelo próprio autor pouco tempo depois de sua criação.

Seguindo o comentário encontram-se as declarações de três variáveis que na verdade são utilizadas como constantes: a constante elétrica, a carga elementar e o valor do nanometro. Estas constantes são utilizadas no cálculo da força. Caso a função seja executada muitas vezes e a otimização seja importante, as constantes podem ser substituídas por um único número na expressão que calcula a força, que já pode incluir a conversão do resultado para piconewtons feita na última linha. Suprimindo ainda o comentário, a função poderia ser escrita em uma única linha:

```
function forcaEletrica(q1,q2,r) { return 230.14*q1*q2/(r*r); }
```

As variáveis declaradas dentro da função (ou de qualquer outro par de chaves num script) são variáveis *locais*, isto é, existem apenas enquanto o bloco de instruções está sendo executado. É o caso das variáveis `k`, `e`, `nm` e `f`. Outras variáveis definidas com o mesmo nome em outros pontos do script ou em outras funções não são afetadas quando a função é executada.

Variáveis definidas fora de contextos delimitados por chaves são chamadas variáveis *globais*, como as variáveis `str` e `cga1`, utilizadas no trecho seguinte para o cálculo das forças. Variáveis globais podem ser acessadas e modificadas em qualquer ponto do scritp, dentro ou fora de funções.

A função retorna o valor calculado da força utilizando a instrução `return (expressão)`, onde `expressão` pode ser uma constante, uma variável, uma expressão mais complexa ou mesmo uma chamada a outra função. Existem funções que não retornam valores, como as que agregam um conjunto de instruções simplesmente para deixar o código mais claro (por exemplo para inicializar um grande número de variáveis globais) ou para desenhar alguma coisa na tela.

Seguindo a definição da função está o trecho que a utiliza para calcular a força em piconewtons para diferentes cargas, dadas em unidades da carga elementar na primeira coluna da tabela, e a diferentes distâncias, dada em nanometros na primeira linha.

Note o uso do modo preformatado `<pre>...</pre>` e das sequências de escape `\n` (quebra de linha) e `\t` (tabulação) em locais estratégicos para formatar a saída dos dados.

Sua vez.. (6-1)

A capacitância de um capacitor de placas paralelas é proporcional à razão entre a área das placas e a distância entre elas, $C = \varepsilon_0 A/d$, onde $\varepsilon_0 = 8{,}9 \times 10^{-12}$ C^2/N·m^2. Modifique o exemplo anterior de modo a criar uma nova função que receba como argumentos a área e a distância entre as placas de um capacitor e retorne a sua capacitância. Utilize a sua função para montar uma tabela semelhante à do exemplo, mas tal que na primeira linha estejam as áreas e na primeira coluna as distâncias fornecidas.

Funções podem ou não retornar informações, que podem ser de números a palavras a objetos bastante complexos. No exemplo abaixo são definidas duas funções, uma para criar matrizes bidimensionais e inicializar seus elementos com zeros e outra para imprimir matrizes bidimensionais. No primeiro caso, a função retorna um objeto, a matriz criada; no segundo caso, a função imprime a matriz, não retornando qualquer valor.

Listagem:

```
<script>

function criaMatriz(a,b) {
   var mat = [];
   for (i=0; i<a; i++) {
      mat[i] = [];
      for (var j=0; j<b; j++) mat[i][j] = 0;
   }
   return mat;
}

function imprimeMatriz(mat) {
   str = "<pre>";
   for (var i=0;i<mat.length;i++) {
      for (var j=0;j<mat[i].length;j++) str += mat[i][j] + " ";
      str += "\n";
   }
   str += "</pre>";
   document.write(str);
}

var matA = criaMatriz(3,4);
imprimeMatriz(matA);

</script>
```

Resultado:

```
0 0 0 0
0 0 0 0
0 0 0 0
```

Note que neste exemplo a função que cria a matriz recebe como parâmetros as suas dimensões. Já a função que imprime a matriz não precisa receber essa informação, pois pode obtê-las da própria matriz, neste caso utilizando utilizando a propriedade `length`.

Sua vez... (6-2)

Inclua no exemplo acima uma função `somaMatrizes(matA,matB)` que recebe como parâmetros duas matrizes `matA` e `matB` e retorna uma matriz contendo a soma de ambas. Para testar se sua função funciona, modifique a função `criaMatriz` de tal modo que seus elementos recebam valores não nulos (por exemplo, $i + j$ ou um número aleatório) e adicione ao corpo do script algo como:

```
var matA = criaMatriz(2,2);
var matB = criaMatriz(2,2);
var matC = somaMatriz(matA,matB);
imprimeMatriz(matC);
```

Frequentemente é conveniente criar funções que retornam valores booleanos (verdadeiro ou falso). No exemplo a seguir dois objetos estão em movimento retilíneo e uniforme, sendo que o primeiro sai de uma posição negativa e vai da esquerda para a direita e o segundo sai de uma posição positiva e vai da direita para a esquerda. O script imprime o tempo decorrido até ocorrer a colisão.

Listagem:

```
<script>

function bateu() {
   if (x1>x2) return true; else return false;
}

var x1 = -10;
var x2 =   5;
var v1 =   3;
var v2 =  -3;
var dt = 0.01;
var t = 0;

while (!bateu()) {
   x1 = x1 + v1 * dt;
   x2 = x2 + v2 * dt;
   t  = t + dt;
};

document.write("A colisão ocorreu em t = " + t.toFixed(2) + " s.");
```

```
</script>
```

Resultado:

```
A colisão ocorreu em t = 2.51 s.
```

A função `bateu()` compara as coordenadas dos objetos e retorna `true` (verdadeiro) quando a coordenada do primeiro for maior que a do segundo, o que significa que eles bateram, e `false` (falso) caso contrário. O corpo do script contem um laço `while` que chama a função `bateu()` antes de cada iteração. O laço pode ser lido como "enquanto não bateu faça ..." (note o uso do operador de negação `"!"` para inverter o resultado retornado pela função). Quando a condição for satisfeita, o script imprime o tempo decorrido entre a partida dos objetos e a colisão.

Sua vez... (6-3)

Acrescente ao script uma função `ultrapassou()` que retorne `true` quando o móvel 1 ultrapassar o móvel 2 e `false` caso contrário. Teste seu script com dois objetos partindo das mesmas posições $x_1 = -10$ m e $x_2 = 5$ m, mas com velocidades $v_1 = 6$ m/s e $v_2 = 3$ m/s. No final, o script deve imprimir o tempo que isso levou para acontecer.

Funções em JavaScript podem retornar múltiplos valores não somente na forma de matrizes, mas também de objetos. No exemplo a seguir a função `achaMaximo(vo,theta)` recebe como parâmetros o módulo da velocidade inicial e o ângulo de lançamento de um projétil e retorna as coordenadas x e y do máximo da trajetória e o instante em que o atingiu.

Listagem:

```
<script>

function achaMaximo(vo,theta) {
   var x1 = 0;
   var y1 = 0;
   theta = theta * Math.PI/180;
   var vx = vo * Math.cos(theta);
   var vy = vo * Math.sin(theta);
   var g  = -9.8;
   var dt = 0.01;
   var t  = 0;

   while (vy>0) {
      x1 = x1 + vx * dt;
      y1 = y1 + vy * dt + g/2*dt*dt;
      vy = vy + g * dt;
      t  = t + dt;
   };

   return { x:x1 , y:y1 , t:t };
}


var max = achaMaximo(10,45);
var str = "";
str += "Máximo em ";
str += "<i>x</i> = " + max.x.toFixed(2) + " m e ";
str += "<i>y</i> = " + max.y.toFixed(2) + " m para ";
str += "<i>t</i> = " + max.t.toFixed(2) + " s. ";
document.write(str);

</script>
```

Resultado:

```
Máximo em x = 5.16 m e y = 2.55 m para t = 0.73 s.
```

A função `achaMaximo(vo,theta)` recebe como parâmetros o módulo da velocidade e o ângulo de lançamento, em graus. A função assume que a posição inicial é sempre $x = 0$ e $y = 0$, converte o ângulo para radianos, calcula as componentes v_x e v_y das velocidades e inicializa algumas variáveis (a gravidade, o intervalo de tempo e o tempo total).

A seguir a função utiliza um laço `while` para propagar a posição e a velocidade (no caso apenas v_y) à medida que o tempo passa. A condição de saída do laço é a inversão do sinal da velocidade, que é o que acontece quando o objeto está no máximo da sua trajetória. Finalmente, a função retorna três valores: as coordenadas x e y do máximo e o tempo t decorrido até o objeto chegar lá.

Note a estrutura da instrução `return` neste caso: são dados os nomes que cada parâmetro retornado deverá ter seguido de dois pontos (`":"`) e do nome da variável local que contém o resultado; outras variáveis retornadas são separadas por vírgula e todo o conjunto está entre chaves.

No corpo do script, após a definição da função, é criada a variável `max` que é inicializada com a chamada à função, momento em que são criadas e preenchidas as propriedades `x`, `y` e `t` do objeto `max`. Para imprimir essas propriedades é utilizado o operador `"."`.

Neste exemplo foi utilizado um método rudimentar de integração numérica da trajetória que introduz um pequeno erro nos cálculos. A solução analítica para esse problema é conhecida e é possível calcular exatamente quanto tempo levaria para a particula atingir o máximo. Dentro da precisão considerada, esse tempo seria de 0,72 s, com pequenas consequências também para os valores das coordenadas *x* e *y* do máximo.

Exercícios

1. Faça um script que contenha uma função `volume(raio)` que receba como argumento o raio de uma esfera e retorne o seu volume.

2. Faça um script que contenha uma função `volume(objeto,dim)` que receba como argumentos o nome do objeto, que pode ser `"cubo"` ou `"esfera"` e a sua dimensão característica `dim`, que é a aresta ou o raio, dependendo do objeto, e devolva o seu volume.

3. Faça um script que contenha uma função `potencial(x,y,z)` que receba como argumentos as coordenadas de um ponto no espaço e retorne o valor do potencial elétrico provocado naquele ponto por uma carga elementar na origem. O potencial elétrico num ponto (x,y,z) é dado por $V(x,y,z) = kq/(x^2+y^2+z^2)^{1/2}$, onde $k = 8.99 \times 10^9$ N·m^2/C^2 e q a carga elétrica. A carga elementar é 1.6×10^{-19} C.

4. Faça um script que contenha uma função `campo(x,y,z)` que receba como argumentos as coordenadas de um ponto no espaço e retorne o valor das três componentes do campo elétrico provocado naquele ponto por uma carga elementar na origem. As componentes do campo elétrico num ponto (x,y,z) são dadas por $E_x(x,y,z) = kq\text{sen}\theta\cos\varphi/(x^2+y^2+z^2)$, $E_y(x,y,z) = kq\text{sen}\theta\text{sen}\varphi/(x^2+y^2+z^2)$ e $E_z(x,y,z) = kq\cos\theta/(x^2+y^2+z^2)$, onde $k = 8.99 \times 10^9$ N·m^2/C^2 e q a carga elétrica. A carga elementar é 1.6×10^{-19} C, $\cos\theta = z/(x^2+y^2+z^2)^{1/2}$ e $\tan\varphi = y/x$ (utilize a função `Math.atan2(y,x)` para obter o ângulo, pois ela leva em conta os sinais de *x* e *y* para determinar o quadrante).

5. Faça um script que contenha uma função `modVet(x,y,z)` que receba as três componentes de um vetor e retorne o valor de seu módulo $(x^2 + y^2 + z^2)^{1/2}$.

6. Faça um script que contenha uma função `angVet(v1,v2)` que receba como argumentos duas matrizes `v1` e `v2` que representem dois vetores $\mathbf{v_1} = (x_1, y_1, z_1)$ e $\mathbf{v_2} = (x_2, y_2, z_2)$ num espaço tridimensional e retorne o ângulo entre eles (o produto escalar é definido como $\mathbf{v_1}\cdot\mathbf{v_2} = |\mathbf{v_1}|\,|\mathbf{v_2}|\cos\theta. = x_1x_2 + y_1y_2 + z_1z_2$.

7. Faça um script que contenha uma função `prodVet(a,b)` que receba como argumentos duas matrizes `a` e `b` que representem dois vetores $\mathbf{a} = (a_x, a_y, a_z)$ e $\mathbf{b} = (b_x, b_y, b_z)$ num espaço tridimensional e retorne o vetor resultante do produto vetorial entre eles (as componentes do vetor resultante $\mathbf{c} = \mathbf{a} \times \mathbf{b}$ são dadas por $c_x = a_y\,b_z - b_y\,a_z$, $c_y = a_z\,b_x - b_z\,a_x$ e $c_z = a_x\,b_y - b_x\,a_y$).

8. Faça um script que contenha uma função `iguais(a,b)` que retorne `true` caso os argumentos `a` e `b` sejam iguais e falso caso contrário.

9. Faça um script que contenha uma função `iguais(a,b,c)` que retorne `true` caso os argumentos `a`, `b` e `c` sejam iguais e falso caso contrário.

10. Faça um script que contenha uma função `zeros(a,b,c)` que retorne `true` caso os argumentos `a`, `b` e `c` sejam iguais a zero e falso caso contrário.

11. Faça um script que contenha uma função `iguais(mat)` que retorne `true` caso todos os elementos da matriz unidimensional `mat` sejam iguais e falso caso contrário.

12. Faça um script que contenha uma função `iguais(mat)` que retorne `true` caso todos os elementos da matriz unidimensional `mat` sejam iguais a zero e falso caso contrário.

13. Modifique o primeiro script deste módulo (cálculo da força elétrica) para que calcule o valor da força para distâncias indo de 1 nm a 10000 nm em potências de 10 (1, 10, 100, etc.) e imprima na tabela os valores do logaritmo (base 10) da força e do logaritmo (base 10) da

distância.

7 Strings

Neste capítulo você vai aprender a criar e utilizar variáveis literais (string) que são sequências com um ou mais caracteres sempre interpretados como texto . O bom entendimento desse tipo de objeto e das propriedades e funções a ele associadas lhe trarão uma melhor compreensão da maneira como dados são representados em computadores e ferramentas para manipulá-los com criatividade.

O objeto `String` acomoda e permite a manipulação de sequências de caracteres que formam textos. Instâncias do objeto podem ser criadas de duas maneiras, implícita ou explicitamente. Strings são criadas implicitamente simplesmente atribuindo uma sequência de caracteres entre aspas a uma variável:

```
var str1 = "Inércia";
var str2 = "F = ma";
var str3 = "Ação e reação";
```

Strings também podem ser criadas explicitamente utilizando o operador `new`:

```
var str1 = new String("Inércia");
var str2 = new String("F = ma");
var str3 = new String("Ação e reação");
```

A diferença entre as duas formas é sutil. No primeiro caso os literais foram adicionados a um objeto *global* do JavaScript que contém todas as literais criadas implicitamente, todas compartilhando das mesmas propriedades e funções. No segundo caso cada literal é um objeto distinto, e a cada um podem ser atribuídas propriedades e funções distintas.

Em JavaScript é possível adicionar propriedades aos objetos. Por exemplo, podemos adicionar a propriedade `conteudo` às literais, atribuindo-lhe valores. Para literais implicitamente declaradas a propriedade é adicionada ao objeto global e as referências são feitas sempre a este objeto global:

Listagem:

```
<script>
var str1 = "Inércia";
var str2 = "F = ma";
var str3 = "Ação e reação";
String.conteudo = "Leis de Newton";
document.write(String.conteudo + ":");
document.write(str1 + ", " + str2 + ", " + str3 + ".");
</script>
```

Resultado:

Leis de Newton: Inércia, F = ma, Ação e reação.

Note que o valor da propriedade `conteudo` é o mesmo, não importando quantos e quais são os literais declarados implicitamente. Se isto for feito em literais explicitamente declaradas, a propriedade é adicionada exclusivamente àquele objeto:

Listagem:

```
<script>
var str1 = new String("Inércia");
var str2 = new String("F = ma");
var str3 = new String("Ação e reação");
str1.conteudo = "1a. lei de Newton: ";
str2.conteudo = "2a. lei de Newton: ";
str3.conteudo = "3a. lei de Newton: ";
document.write(str1.conteudo + str1 + ".<br>");
document.write(str2.conteudo + str2 + ".<br>");
document.write(str3.conteudo + str3 + ".<br>");
</script>
```

Resultado:

1a. lei de Newton: Inércia.
2a. lei de Newton: F = ma.
3a. lei de Newton: Ação e reação.

Sua vez... (7-1)

Modifique os exemplos anteriores de modo que substituam as leis de Newton pelas leis da termodinâmica.

Objetos `Strings` tem propriedades e métodos muito úteis para sua manipulação, alguns deles apresentados a seguir. Procure um manual ou site de referência em JavaScript para obter mais detalhes dos apresentados aqui e sobre outros disponíveis.

length

A propriedade `length` contem um inteiro que representa o número de caracteres no literal:

Listagem:

```
<script>
var str = "12345";
document.write("\"" + str + "\" tem " + str.length + " caracteres");
</script>
```

Resultado:

"12345" tem 5 caracteres

No exemplo acima, o texto final é impresso com aspas (") antes e depois da sequência de números. Como em um programa em JavaScript as aspas têm a função de delimitar um literal, simplesmente colocá-las no texto provocaria um erro. Para evitar isso, coloca-se uma barra invertida (\) na frente das aspas a serem impressas. No primeiro item (`"\""`) da instrução `document.write`, as primeiras aspas delimitam o início de um literal; em seguida há a barra invertida, que indica que as aspas que a sucedem devem ser impressas, e não interpretadas como o final do literal; finalmente, as últimas aspas indicam o fechamento do literal. No terceiro item (`"\" tem "`), o mesmo acontece, com a diferença de que a palavra "tem" entre espaços em branco foi colocada depois das aspas a serem impressas e antes das aspas que delimitam o fim do literal.

Caracteres antecedidos com uma barra invertida são conhecidos como *caracteres de escape*, sendo interpretados como simples caracteres e destituídos de eventuais propriedades adicionais de que disponham. Por exemplo, a letra "a" é sempre interpretada como a letra "a"; já as aspas podem ser interpretadas como o caracter " (aspas) ou como um delimitador de início ou término de uma sequência de caracteres. Como a própria barra invertida é um caracter especial, para imprimi-la é necessário antecedê-la por outra barra invertida (\\).

Sua vez... (7-2)

Faça um script que utilize a instrução `window.prompt(...)` para solicitar que o usuário digite uma frase e um `window.alert(...)` para retornar quantos caracteres tem a frase digitada.

---

charAt(ind)

O método `charAt(ind)` retorna o caracter posicionado em um determinado índice. O primeiro caracter de um literal tem o índice 0 e o último `length-1`:

Listagem:

```
<script>
str = "pV = nRT";
document.write("A literal '" + str + "' contem os caracteres: ");
for (var i=0;i<str.length;i++) {
   document.write("'" + str.charAt(i) + "'");
   if (i!=str.length-1) document.write(", ");
   else document.write(".");
}
</script>
```

Resultado:

A literal 'pV = nRT' contem os caracteres: 'p', 'V', ' ', '=', ' ', 'n', 'R', 'T'.

Neste exemplo utilizamos aspas simples e não aspas duplas para envolver os caracteres e portanto a sequência de escape com a barra invertida não é necessária. Note também que os espaços em branco antes e depois do sinal de igual são tratados como qualquer outro caracter.

Sua vez... (7-3)

Modifique a sua resposta para o exercício anterior de modo que utilize a instrução `document.write(...)` para imprimir em uma linha os caracteres nas posições pares (0, 2, 4, ..., 2*n*) e em outra os caracteres ímpares (1, 3, 5, ..., 2*n*+1).

fromCharCode(cod1, cod2, ..., codn)

O método `fromCharCode(cod1, cod2, ..., codn)` devolve um literal a partir de uma sequência de códigos Unicode. Por exemplo, o código Unicode da letra "A" é 65, da letra "B" é 66 e assim por diante. Para gerar uma sequência de três letras aleatórias pode-se fazer algo como:

Listagem:

```
<script>
var cod1 = 65 + parseInt(Math.random()*26);
var cod2 = 65 + parseInt(Math.random()*26);
var cod3 = 65 + parseInt(Math.random()*26);
var str = String.fromCharCode(cod1,cod2,cod3);
document.write(str);
</script>
```

Resultado:

TYP

No exemplo acima, tomar a parte inteira (`parseInt`) do produto de um número aleatório entre 0 e 1 (não incluídos) por 26 (o número de letras no alfabeto ocidental), retorna um número entre 65 (letra "A") e 65+25 (letra "Z").

Note que o método `fromCharCode` deve ser evocado a partir do objeto global `String`, pois não é uma operação sobre um objeto já existente. Note também que pode receber vários argumentos simultaneamente.

Sua vez... (7-4)

Escreva um programa que imprima todas as letras maiúsculas e minúsculas do alfabeto ocidental utilizando um laço `for` e a instrução `String.fromCharCode(...)`.

charCodeAt(ind)

O método `charCodeAt(ind)` retorna o código Unicode correspondente ao caracter na posição especificada do literal:

Listagem:

```
<script>
var str = "12345";
for (var i=0;i<str.length;i++)
   document.write(str.charCodeAt(i) + " ");
</script>
```

Resultado:

```
49 50 51 52 53
```

Sua vez... (7-5)

Modifique o exemplo anterior de modo que imprima o produto de cada número fornecido com a sua posição na matriz (por exemplo, `var m = [3,6,1,2,4]`; no caso, o programa deve imprimir 1*3, 2*6, etc.). Você consegue enxergar nisso uma forma elementar de criptografia de dados?

---

concat(str)

O método `concat(str)` retorna um literal com a junção (concatenação) de dois literais:

Listagem:

```
<script>
var str1 = "2a. Lei de Newton: ";
var str2 = "F = ma.";
document.write(str1.concat(str2));
</script>
```

Resultado:

```
2a. Lei de Newton: F = ma.
```

Note que o resultado do método `concat` é idêntico ao uso do operador `"+"` entre dois literais:

Listagem:

```
<script>
var str1 = "2a. Lei de Newton: ";
var str2 = "F = ma.";
document.write(str1 + str2);
</script>
```

Resultado:

```
2a. Lei de Newton: F = ma.
```

indexOf(str)

O método `indexOf(str)` devolve a posição da primeira ocorrência de uma sequência de caracteres dentro de um objeto String. Se a sequência não for encontrada, o método devolve −1. O exemplo a seguir apresentada a posição da sequência " e ", que inclui espaços em branco antes e depois da letra "e", e que começa na posição 12, pois o primeiro caracter está na posição 0:

Listagem:

```
<script>
var str="Eletricidade e Magnetismo";
document.write(str.indexOf(" e "));
</script>
```

Resultado:

```
12
```

O exemplo a seguir conta quantas vezes aparece a letra "e" na literal. Note o uso da instrução `break` para sair do que seria um laço infinito caso o método `indexOf(...)` não devolvesse um −1 quando não encontra mais a sequência procurada.

Listagem:

```
<script>
var str = "Eletricidade e Magnetismo";
var i = 0;
var cnt = 0;
do {
   i = str.indexOf("e",i) + 1;
   if (i>0) cnt++; else break;
} while (true);
document.write("Foram encontradas " + cnt + " letras \"e\".");
</script>
```

Resultado:

```
Foram encontradas 4 letras "e".
```

Note que o algoritmo diferencia o "E" maiúsculo do "e" minúsculo — são 4 "e" 's minúsculos e um "E" maiúsculo.

Sua vez... (7-6)

Modifique o exemplo anterior de modo que imprima o número de vogais na frase (no caso 11, considerando maiúsculas e minúsculas).

lastIndexOf(str)

O método `lastIndexOf(str)` devolve a última ocorrência de uma sequência de caracteres em um literal. É essencialmente idêntico ao método `indexOf`, mas percorre a literal da direita para a esquerda. O exemplo a seguir mostra em que posição a sequência "ACGT" é encontrada pela última vez na literal.

Listagem:

```
<script>
var str="AATCGCAACGTGGCTATGACGTGCCACTACCGCACGTCGAG";
var pos = str.lastIndexOf("ACGT");
document.write(pos);
</script>
```

Resultado:

```
33
```

split(str|regexp)

O método *`string`*`.split(str|regexp)` retorna uma matriz cujos elementos são os trechos de *`string`* separados pela sequência `str`. O exemplo abaixo define uma sequência de pares de números. Os elementos de um par são separados por vírgula e os pares são separados por ponto-e-vírgula (uma sequência de pontos no plano cartesiano, por exemplo). A seguir são construídas duas matrizes com o método `split` aplicado à sequência. No primeiro caso, o caracter utilizado como critério de separação é o ponto-e-vírgula. No segundo caso é passada uma *expressão regular* que informa que tanto a vírgula quanto o ponto-e-vírgula podem ser utilizados como critérios de separação.

Listagem:

```
<script>

var seq = "1,2;2,4;3,9;4,16";
var par = seq.split(";");
var num = seq.split(/,|;/);

var str = "<pre>";
for (var i in par) str += par[i] + "\t";
```

```
str += "\n";
for (var i in num) str += num[i] + "\t";
str += "</pre>";
document.write(str);

</script>
```

Resultado:

```
1,2 2,4 3,9 4,16
1 2 2 4 3 9 4 16
```

Note o uso das sequências de escape `\t` (tabulação) e `\n` (nova linha) para gerar uma formatação razoável no modo pré-formatado (`<pre>...</pre>`).

Expressões regulares (*regular expressions*) são poderosas construções utilizadas em procedimentos de manipulação de texto. O leitor interessado pode facilmente encontrar mais informações e exemplos de uso na internet.

Sua vez... (7-7)

Faça um script que atribua a uma variável o texto:

"Os quarks são seis: up, down, charm, strange, bottom, top. Os léptons são seis: elétron, múon, tau, elétron-neutrino, múon-neutrino, tau-neutrino."

e utilize a instrução `split(...)` produzir algo parecido com:

Os quarks são seis: up, down, charm, strange, bottom, top

Os léptons são seis: elétron, múon, tau, elétron-neutrino, múon-neutrino, tau-neutrino

e a seguir com:

Os quarks são seis
up, down, charm, strange, bottom, top

Os léptons são seis
elétron, múon, tau, elétron-neutrino, múon-neutrino, tau-neutrino

---

slice(inicio [,fim]), substring(inicio [,fim]), substr(inicio [,tamanho]),

Os três métodos fazem essencialmente a mesma coisa — retornam um pedaço de um literal — com diferenças sutis quanto ao significado do segundo parâmetro. Nos três casos o segundo parâmetro é opcional e, se não for fornecido, o literal retornado conterá todos os caracteres a partir do índice `inicio`, inclusive, até o final do literal original, e os três métodos têm o mesmo resultado.

O parâmetro `fim` nos métodos `slice(...)` e `substring(...)` indica até que caracter o literal original deve ser copiado, mas não o inclui, deixando os dois métodos com a mesma funcionalidade. Já o parâmetro `tamanho` passado ao método `substr(...)` indica o comprimento do trecho a ser copiado.

O exemplo a seguir utiliza uma das possíveis maneiras de codificar cores no padrão RGB (*red-green-blue*), em que valores entre hexadecimal 00 (decimal 0) e hexadecimal FF (decimal 255) são atribuídos a cada componente de cor, para imprimir uma expressão na cor especificada e com as componentes vermelha e azul invertidas.

Listagem:

```
<script>
var cor1 = "#0000FF";
var cor2 = "#"+cor1.slice(5,7)+cor1.slice(1,3)+cor1.slice(3,5);
document.write("<span style='color:"+cor1+"'>"+cor1+"</span>");
document.write(" para ");
document.write("<span style='color:"+cor2 +"'>"+cor2 +"</span>");
</script>
```

Resultado:

#0000FF para #FF0000

Note o uso do elemento `span` do HTML associado a uma declaração de estilo `style=...` para formatar um trecho de texto utilizando CSS (para aprender mais, faça uma busca por estes termos na internet!). Esta maneira de fazê-lo (no caso, atribuir cor) pode parecer a muitos absurdamente complexa e prolixa, mas é justamente esse elevado grau de formalização que está na base da versatilidade e interatividade de todas as aplicações para a internet hoje.

Sua vez... (7-8)

Modifique o exemplo anterior de modo a obter o mesmo resultado utilizando (a) o método `substr(...)` e (b) o método `substring(...)` em substituição ao método `slice(...)`.

---

toLowerCase(), toUpperCase()

Os métodos `toLowerCase` e `toUpperCase` transformam os objetos sobre os quais operam em sequências em que todos os caracteres ficam em caixa baixa (minúsculas) e caixa alta (maiúsculas), respectivamente. Estas funções não têm efeito em caracteres não-alfabéticos, tais como números e sinais de pontuação. Estas funções podem ser úteis quando se deseja processar ou armazenar informações digitadas pelo usuário sem se preocupar se foi digitada em maiúsculas ou minúsculas. O exemplo a seguir deve ser suficientemente autoexplicativo:

Listagem:

```
<script>
var str = "EsTa É uMa fRasE qUaLqUeR.";
document.write(str + "<br>");
document.write(str.toLowerCase() + "<br>");
document.write(str.toUpperCase() + "<br>");
document.write("e EsTa É oUtRa FrAsE".toLowerCase() + "<br>");
</script>
```

Resultado:

EsTa É uMa fRasE qUaLqUeR.
esta é uma frase qualquer.
ESTA É UMA FRASE QUALQUER.
e esta é outra frase

Note que os métodos `toLowerCase()` e `toUpperCase()` operam sobre variáveis, como em `str.toLowerCase()` e sobre constantes literais, como em `"e EsTa É oUtRa FrAsE".toLowerCase()`.

Sua vez... (7-9)

Faça um programa que utilize um laço `do ... while (...)`, a instrução `window.prompt(...)` e o método `toLowerCase()` para solicitar continuamente ao usuário que digite qualquer coisa enquanto o que for digitado for diferente da letra "q", seja maiúscula ou minúscula.

---

toString([base])

O método `toString` converte valores numéricos e outros objetos em literais. Ao converter números, pode receber um parâmetro que especifica a base utilizada na conversão. O exemplo a seguir ilustra como conveter números decimais em binários e hexadecimais.

Listagem:

```
<script>
var str = "<pre>";
str += "Dec \tBin \tHex \n";
for (var i=7;i<17;i++) {
 str += i + "\t";
```

```
 str += i.toString(2) + "\t";
 str += i.toString(16) + "\n";
}
str += "</pre>";
document.write(str);
</script>
```

Resultado:

```
Dec  Bin  Hex
7 111 7
8 1000 8
9 1001 9
10 1010 a
11 1011 b
12 1100 c
13 1101 d
14 1110 e
15 1111 f
16 10000 10
```

No exemplo acima, note mais uma vez o uso do elemento `<pre>...</pre>` e das sequências de escape `\t` (tabulação) e `\n` (nova linha) para produzir um texto minimamente formatado.

Sua vez.. (7-10)

Faça um programa que crie três variáveis, `var a = 30` , `var b = 209` e `var c = a + b` e imprima os valores das variáveis e o resultado da soma em base hexadecimal:

$$1e + d1 = ef$$

Exercícios

1. Faça um script que peça ao usuário uma longa sequência de números separados por vírgulas que representem pares coordenados $x_1, y_1, x_2, y_2, \ldots, x_N, y_N$ e imprima-os no formato:

   x_1 = ..., y_1 = ...
   x_2 = ..., y_2 = ...
   ...
   x_N = ..., y_N = ...

2. O CPF (Cadastro de Pessoa Física) de um contribuinte consiste de 9 números e dois dígitos verificadores. Os dígitos verificadores são assim chamados porque são determinados por operações realizadas nos números: se algum deles estiver errado, os dígitos verificadores não serão iguais aos que podem ser calculados. O algoritmo que os define funciona assim:

   Multiplique o primeiro número por 10, o segundo por 9, o terceiro por 8 e assim por diante até o nono número, que é multiplicado por 2. Some tudo e divida o total por 11. Se o resto da divisão for menor que 2, o primeiro dígito verificador é 0; caso contrário é a diferença entre 11 e o resto.

   Para gerar o segundo dígito verificador o procedimento é semelhante, mas inclui o primeiro dígito verificador: multiplique o primeiro número por 11, o segundo por 10, o terceiro por 9 e assim por diante até o nono, que é multiplicado por 3. Multiplique o primeiro dígito verificador por 2. Some tudo e divida por 11. Se o resto da divisão for menor que 2, o segundo dígito verificador é 0; caso contrário é a diferença entre 11 e o resto.

   Faça um algoritmo que peça ao usuário para digitar o seu CPF no formato 000.000.000-00 e confira se o número fornecido está correto.

3. Um sistema de criptografia simples para senhas de 6 caracteres pode ser pensado da seguinte maneira. Construa uma sequência de seis números entre 1 e 6, inclusive, sem repetição, por exemplo "352164". Esta sequência é compartilhada pelo emissor e pelo receptor em um contato inicial, seguro. Em comunicações subsequentes o emissor utilizará esta sequência, que só ele e o receptor conhecem, e vai "embaralhar" a sua senha com ela: o 1o. caracter digitado vai para a 3a. posição, o 2o. para a 5a., o 3o. para a 2a., o 4o. para a 1a., o 5o. para a 6a. e o 6o. para a 4a. A senha embaralhada é enviada por um canal não-seguro ao receptor, que a "desembaralha" para verificar a sua autenticidade. Faça dois scripts: um que pegue a senha digitada pelo usuário e a embaralhe gerando a senha

criptografada e outro que pegue a senha criptografada e a transforme novamente na senha original.

4. Faça um script que peça ao usuário que digite três números entre 0 e 255 (inclusive) separados por vírgulas, converta-os em hexadecimais h1, h2 e h3, construa uma sequencia literal no formato "#h1h2h3" que representa uma cor em HTML e imprima o resultado na cor gerada.

5. Faça um script que peça dois números (separados por vírgula) em um sistema quaternário (que tem apenas quatro símbolos e no qual os números naturais são representados por 0, 1, 2, 3, 10, 11, 12, 13, 20 etc.) e imprima o resultado da soma e do produto deles.

6. Faça um script que peça ao usuário uma frase e faça a distribuição do tamanho das palavras na frase. Por exemplo, a primeira frase deste exercício ("Faça um ... na frase") contém 1 palavra com 1 letra, 4 palavras com 2 letras, 3 palavras com 3 letras, 3 palavras com 4 letras, 2 palavras com 5 letras, 1 palavra com 6 letras, 2 palavras com 7 letras, 1 palavra com 8 letras, 0 palavras com 9 letras, 0 palavras com 10 letras, 0 palavras com 11 letras, 1 palavra com 12 letras, 0 palavras com 13 letras e 0 palavras com 14 letras. Um possível algoritmo para resolver o problema poderia ser algo como:
    1. Defina uma variável literal para armazenar a frase que deseja analisar obtida por um prompt (`var str = prompt(...)`).
    2. Defina uma matriz (`Array`) com, por exemplo, 15 posições para armazenar a contagem de palavras para cada tamanho (não esqueça de inicializa-la com zeros!).
    3. Utilize o método `split` para separar a frase em palavras isoladas em uma outra matriz.
    4. Percorra a matriz com as palavras e utilize a propriedade `length` para obter o comprimento de cada um dos literais em seus elementos, acumulando-os na matriz criada para armazenar a contagem de palavras.
    5. Imprima a matriz com a contagem de palavras.

7. Faça um script que concatene em um único literal uma sequência genética ("ACGTCGTAGGTC...") aleatória com 100 mil bases e depois busque nela quantas vezes aparecem as subsequências "AA", "AAA" e "AAAA". Use o método `concat` ou o operador `+` em um laço `for` para construir o literal.

Loading [MathJax]/extensions/tex2jax.js

8 HTML/CSS/JavaScript

Neste capítulo você vai entender como os programas (scripts) *em JavaScript encaixam-se e operam sobre uma estrutura muito maior e poderosa: o* documento *HTML. Vai estudar como utilizar comandos de marcação* (mark-up) *para identificar, caracterizar, acessar e modificar cada elemento de seu documento, como utilizar os elementos de estilo* (styles) *para produzir dos mais simples aos mais sofisticados efeitos de formatação e interação e como utilizar o JavaScript para manipular tudo isso dinamicamente, dando interatividade ao seu documento.*

Talvez o arquivo HTML mais simples que alguém pode criar seja algo como:

Listagem:

```
<html>
<body>
<p>Equações de Maxwell</p>
</body>
</html>
```

Se você salvar este arquivo como `teste.html` e em seguida clicar sobre ele, verá que o navegador vai mostrar o seguinte:

Resultado:

Equações de Maxwell

De tudo o que foi escrito no arquivo, somente o texto "Equações de Maxwell" foi apresentado. Todos os outros elementos são *marcadores*, (*tags*), que orientam o navegador sobre o que fazer com os pedaços de texto que você coloca aqui e ali.

Sua vez... (8-1)

O HTML tem predefinidos outros elementos com comportamento semelhante, os cabeçalhos `<h1>...</h1>`, `<h2>...</h2>` etc. Troque o elemento `<p>...</p>` por estes outros elementos e veja o efeito. Quantos tipos diferentes de cabeçalhos seu navegador consegue identificar?

Você deve ter notado os marcadores `<html>...</html>` no início e no fim do arquivo. Este "envelope" diz ao navegador para interpretar o arquivo como um documento HTML. Se você procurar por aí, vai descobrir que este marcador pode ter uma estrutura bastante complexa, passando informações adicionais ao navegador. Não é objetivo deste texto aprofundar este assunto, mas se você quiser utilizar todo o poder dessa tecnologia, não vai conseguir escapar disso.

Há um envelope de segundo nível definido pelos marcadores `<body>...</body>`, que delimitam o *corpo* do documento. Este segundo envelope pode parecer redundante, mas não é. Como veremos, podemos associar a ele instruções de formatação global do documento (tamanho da fonte, cor ou imagem de fundo etc.) e ações específicas que serão executadas quando o navegador carregar o documento. Além disto, existem outros elementos que podem aparecer entre os marcadores `html` mas fora dos marcadores `body`.

O exemplo a seguir dá à página um título, que aparecerá na barra principal da janela quando o documento é carregado. Além disso, usa um elemento de estilo para colocar um fundo cinza claro (*lightgray*) em todo o documento e abre uma caixa de alerta dando as boas vindas ao usuário:

Listagem:

```
<html>
<title>Eletromagnetismo</title>
<body style="background-color:lightgray"
      onLoad="window.alert('Seja bem vindo!')">
<p>Equações de Maxwell.</p>
</body>
</html>
```

O próximo fato a ser notado é que os marcadores, salvo raríssimas exceções, aparecem sempre aos pares: `<html>...</html>`, `<body>...</body>`, `<title>...</title>`, `<p>...</p>`, sendo que o segundo elemento consiste na repetição do nome do marcador precedida de uma barra (`"/"`). O elemento `<p>...</p>` representa um parágrafo no documento. Experimente ver as diferenças entre as representações pelo navegador das sequências a seguir:

Listagem:

```
<html>
<body>
<p>Equações de Maxwell.</p>
<p>A base do eletromagnetismo clássico!</p>
</body>
</html>
```

Listagem:

```
<html>
<body>
<p>
Equações de Maxwell.
A base do eletromagnetismo clássico!
</p>
</body>
</html>
```

Listagem:

```
<html>
<body>
<p>
Equações    de
Maxwell.

A   base  do
Eletromagnetismo    cássico
!
</p>
</body>
</html>
```

Se você experimentou deve ter notado que as diferenças entre as duas primeiras sequências é grande: na primeira, cada frase saiu em uma linha diferente e há um espaçamento razoável entre elas, pois constituem dois parágrafos distintos, como indicam os marcadores `<p>...</p>`. Já entre a segunda e a terceira sequências não existem diferenças nas representações. O navegador encara tudo como pertencente a um único parágrafo e ignora espaços em branco ou linhas adicionais.

Inúmeras outras instruções podem ser passadas ao navegador através dos marcadores. Já vimos, por exemplo, que temos que envolver com `<i>...</i>` um trecho que queremos que apareça em *itálico* e com `<b>...</b>` um trecho que queremos que apareça em **negrito** (*boldface*). Podemos fazer _{subscritos} com `<sub>...</sub>` e ^{sobrescritos} com `<sup>...</sup>`.

Sua vez... (8-2)

Escreva uma carta à sua namorada ou namorado explicando toda a beleza das equações de Maxwell. Utilize os elementos `<i>...</i>` e `<b>...</b>` para atribuir diferentes formatações a palavras no texto e os elementos de estilo `<p style="text-align:left">`, `<p style="text-align:right">` e `<p style="text-`

`align:justify">` para atribuir diferentes modos de alinhamento dos parágrafos:

Florianópolis, 20 de outubro de 2015

Querida:

Você não imagina o que aprendi essa semana: as **equações de Maxwell**! Estou pasmo! São apenas quatro, mas com elas podemos ...

Com saudades,

Fulano de Tal

Imagens

A inclusão de imagens no HTML é feita com o uso do marcador `<img ... />`, que é um dos poucos que não requer o par de fechamento. Se o arquivo com a imagem chama-se `densimetro.gif`, por exemplo, e está na mesma pasta que o documento HTML, a inclusão é feita como no exemplo a seguir.

Listagem:

```
<p>
Um densímetro é um dispositivo desenhado para medir a
densidade de líquidos...
</p>

<p style="text-align:center">
<img src="densimetro.gif" style="width:100px" />
</p>

<p>
Como se pode notar, contém um bulbo meticulosamente
calibrado e uma escala...
</p>
```

Resultado:

Um densímetro é um dispositivo desenhado para medir a densidade de líquidos...

Como se pode notar, contém um bulbo meticulosamente calibrado e uma escala...

O elemento `img` requer um valor para o parâmetro `src` (*source*), a especificação da URL (*Universal Resource Locator*) da imagem, ou seu "endereço", que pode estar na mesma pasta do arquivo que a carregou, em outra pasta no mesmo computador.

Note também a declaração de estilo, que impõe que sua largura (*width*), no documento, seja exatamente de 100 pixels, independentemente do seu tamanho original.

Listas ordenadas e não-ordenadas

Particularmene úteis para ajudar na organização de um texto são as listas ordenadas e não-ordenadas. Uma lista ordenada é indicada pelos marcadores `<ol> ... </ol>` (*ordered list*) e seus elementos pelos marcadores `<li> ... </li>` (*list item*):

Listagem:

```
<p>Unidades de energia:</p>
<ol>
<li>Erg</li>
<li>Joule</li>
<li>Elétron-volt</li>
</ol>
```

Resultado:

Unidades de energia:

1. Erg
2. Joule
3. Elétron-volt

Uma lista não-ordenada é indicada pelos marcadores `<ul> ... </ul>` (*unordered list*) e seus elementos também são indicados pelos marcadores `<li> ... </li>`:

Listagem:

```
<p>Unidades de energia:</p>
<ul>
<li>Erg</li>
<li>Joule</li>
<li>Elétron-volt</li>
</ul>
```

Resultado:

Unidades de energia:

- Erg
- Joule
- Elétron-volt

Estas listas podem ser aninhadas (colocadas umas dentro de outras) e seus marcadores escolhidos dentre algumas possibilidades utilizando a declaração de estilo `style="list-style-type:`*`seletor`*`"`. Para listas ordenadas, alguns seletores disponíveis são: `decimal` (padrão), `lower-alpha`, `upper-roman`, entre outros. Para listas não ordenadas, alguns seletores disponíveis são: `disc` (padrão), `circle` e `square`.

Listagem:

```
<ol style="list-style-type:upper-alpha">
<li>Unidades de energia:</li>
   <ul style="list-style-type:square">
   <li>Erg</li>
   <li>Joule</li>
   <li>Elétron-volt</li>
   </ul>
<li>Unidades de potência:</li>
   <ul style="list-style-type:circle">
   <li>Watt</li>
   <li>Cavalo-vapor</li>
   <li>BTU</li>
   </ul>
</ol>
```

Resultado:

A. Unidades de energia:
   - Erg
   - Joule
   - Elétron-volt

B. Unidades de potência:
   - Watt
   - Cavalo-vapor
   - BTU

É também possível colocar qualquer imagem no lugar do conteúdo padrão da lista não-ordenada. O exemplo abaixo assume a existência de um arquivo `checkbox.gif` na mesma pasta em que se encontra o arquivo HTML e utiliza a imagem nele contida em cada item da lista:

Listagem:

```
<p>Para compreender bem as rotações você deve estudar:</p>
<ul style="list-style-image:url(checkbox.gif)">
<li>Momento de inércia</li>
<li>Torque</li>
<li>Momento angular</li>
</ul>
```

Resultado:

Para compreender bem as rotações você deve estudar:

- Momento de inércia
- Torque
- Momento angular

Sua vez... (8-3)

Escreva um trecho de código HTML que explore os vários estilos de listas ordenadas e não ordenadas. Consulte a internet para conhecer todos os possíveis valores para o `list-style-type`:

1. Primeiro nível
   A. Segundo nível
      a. Terceiro nível
         I. Quarto nível
            i. Quinto nível

- Primeiro nível
  - Segundo nível
    - Terceiro nível
      - Quarto nível (produza sua própria imagem!)

JavaScript

Trechos de código JavaScript podem ser introduzidos em qualquer ponto de um arquivo HTML utilizando os marcadores `<script>...</script>`, inclusive no meio de uma simples frase. O exemplo a seguir mostra a função `fatorial(x)` definida no cabeçalho e evocada no corpo do documento.

Listagem:

```
<html>
<head>
<script>
function fatorial(num) {
   if (num==0) return 1;
   var res = 1;
   do {
      res = res * num;
      num--;
   } while (num>0);
   return res;
}
</script>
</head>
<body>
<script>
var num = window.prompt("Entre com um número inteiro positivo: ");
</script>
<p>O fatorial de <script>document.write(num)</script>
é <script>document.write(fatorial(num))</script>.</p>
</body>
</html>
```

É muito comum encontrar grandes quantidades de JavaScript na forma de funções nos cabeçalhos (*head*) dos arquivos HTML. Todas as funções e variáveis globais definidas neste contexto permanecem disponíveis a menos que sejam explicitamente apagadas.

O exemplo a seguir apresenta uma questão de prova em que duas velocidades são sorteados na hora em que a página é carregada. Os valores sorteados são armazenados e posteriormente utilizados para calcular automaticamente a resposta para o problema.

Listagem:

```
<script>
var vn = (1 + Math.random()).toFixed(1);
var vb = (15 + Math.random()).toFixed(1);
str = "<p>A distância entre a Ilha do Campeche e a \
```

```
       Praia do Campeche é de cerca de 1.8 km. Um nadador \
       sai da praia em direção à ilha nadando a uma \
       velocidade de " + vn + " km/h. Quanto tempo depois \
       um barco deve deixar a praia, navegando a " + vb + "km/h, \
       para chegar à ilha junto com o nadador?</p>";
// Resolução
// xn = vn * tn;
// xb = vb * tb;
var tn = 1.8/vn;
var tb = 1.8/vb;
var dt = (tn - tb);
str += "<p><i>Resposta:</i> " + dt.toFixed(1) + " horas.</p>"
document.write(str);
</script>
```

Resultado:

A distância entre a Ilha do Campeche e a Praia do Campeche é de cerca de 1.8 km. Um nadador sai da praia em direção à ilha nadando a uma velocidade de 2.0 km/h. Quanto tempo depois um barco deve deixar a praia, navegando a 16.0 km/h, para chegar à ilha junto com o nadador?

Resposta: 0.8 horas.

Note o uso de uma variável `str` inicializada com o enunciado da questão e que, depois dos cálculos, acumula (`+=`) a resposta e é impressa com a chamada final a `document.write(...)`. Note também que o enunciado da questão está dividido em várias linhas por conveniência editorial, terminadas por uma barra invertida (`\`) para que o navegador não ache que você simplesmente esqueceu de fechar as aspas.

Sua vez... (8-4)

Inspire-se no exemplo anterior e construa uma questão que sorteie quatro números a_1, a_2, b_1 e b_2 entre −1 e +1 que representem as componentes de dois vetores bidimensionais e calcule e imprima o resultado da sua soma (vetorial).

Tabelas

Tabelas são um importante recurso de organização da informação, particularmente quando esta vem em grandes quantidades. O exemplo a seguir ilustra como fazer uma tabela muito simples em HTML, com duas linhas e duas colunas:

Listagem:

```
<table border='1' cellspacing='5' cellpadding='5' width='300'>
   <tr>
      <td>linha 1, coluna 1</td>
      <td>linha 1, coluna 2</td>
   </tr>
   <tr>
      <td>linha 2, coluna 1</td>
      <td>linha 2, coluna 2</td>
   </tr>
</table>
```

Resultado:

linha 1, coluna 1	linha 1, coluna 2
linha 2, coluna 1	linha 2, coluna 2

Tabelas são construídas utilizando-se os marcadores `<table> ... </table>` (*tabela*), que sinalizam o início e o fim da tabela, `<tr> ... </tr>` (*table row*), que sinalizam o início e o fim de uma linha, e `<td> ... </td>` (*table data*), que sinalizam o início e o fim de uma célula de dados da tabela. Como em qualquer outra estrutura HTML, quebras de linha, espaços e tabulações no código fonte são ignorados e podem ser utilizados ao gosto do autor. A sequência abaixo levaria exatamente ao mesmo resultado:

```
<table border='1' cellspacing='5' cellpadding='5' width='300'>
<tr><td>linha 1, coluna 1</td><td>linha 1, coluna 2</td></tr>
<tr><td>linha 2, coluna 1</td><td>linha 2, coluna</td></tr>
</table>
```

O elemento `table` pode ter diversos atributos e receber definições de estilo (`style=...`), o que não é o caso desta tabela em particular. São utilizados os atributos `border`, que define a espessura da borda da tabela e das células, `cellspacing`, que define o espaçamento entre duas células, `cellpadding`, que especifica a distância entre a parede da célula e o texto dentro dela e `width`, que especifica a largura da tabela. A menos que explicitamente especificados de outra maneira, todos os valores são dados em pixels. Experimente mudar os valores e veja o que acontece.

> Os marcadores para linhas (`tr`) e células (`td`), como praticamente todos os outros elementos HTML, também podem ter atributos e declarações de estilo.

A associação de tabelas, declarações de estilo e JavaScript pode ser bastante produtiva. O exemplo a seguir imprime em uma tabela a tabuada de 1 a 10, gerada por um script:

Listagem:

```
<p>
<table style='width:500; margin:auto'
       cellpadding='3' cellspacing='0' border='1'>
<script>
// sinal de vezes no canto superior esquerdo da tabela
var str = "<tr><th style='background-color:gray'>&times;</th>";
// cabeçalhos das colunas de 1 a 10
for (var col=1; col<=10; col++)
   str += "<th style='background-color:gray'>" + col + "</th>";
// fecha a linha de cabeçalhos das colunas
str += "</tr>";
// escreve as 10 linhas seguintes
for (var lin=1; lin<=10; lin++) {
   // começa a linha
   str += "<tr>";
   // célula escurecida no início da linha
   str += "<th style='background-color:gray'>" + lin + "</th>";
   // produtos propriamente ditos
   for (var col=1; col<=10; col++)
      str += "<td style='text-align:center'>" + lin*col + "</td>";
   // termina a linha
   str += "</tr>";
   }
document.write(str);
</script>
</table>
</p>
```

Resultado:

×	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

Algumas novidades do exemplo acima:

1. O declaração de estilo `style='...'` foi utilizada no marcador `table` para definir sua largura e estabelecer margens automáticas, o que centraliza a tabela.
2. O marcador `<th> ... </th>` (*table header*) deixa o conteúdo da célula em negrito e centralizado automaticamente.

3. A declaração de estilo `style='background-color:gray'` atribui a cor cinza ao fundo da célula de dados.

4. A tabela a ser impressa foi "acumulada" em uma variável literal `str`, escrita no documento somente no final do processo.

Sua vez... (8-5)

Para navegar melhor entre tantos `tr`'s e `td`'s, modifique o exemplo acima de modo que (a) passe a imprimir a tabela de multiplicação apenas de 1 a 5, (b) a linha e a coluna com fundo cinza passe a ter um fundo amarelo (`yellow`) e as demais células passem a ter um fundo verde-claro (`lightgreen`).

×	1	2	3	4	5
1	1	2	3	4	5
2	2	4	6	8	10
3	3	6	9	12	15
4	4	8	12	16	20
5	5	10	15	20	25

É possível juntar (mesclar) células em linhas e colunas. Isso é feito utilizando-se os atributos `rowspan` (abrangência de linhas) e `colspan` (abrangência de colunas). No exemplo abaixo, que mostra os isótopos do magnésio, suas meias-vidas e modos de decaimento, a primeira linha define 4 células; a segunda define três, mas a célula central ocupa duas colunas e duas linhas; como 2 das 4 células da terceira linha já foram ocupadas pela extensão da célula da segunda linha, nela são definidas somente 2 células; finalmente, a quarta linha volta ao "normal", apresentando quatro células.

Listagem:

```
<table style='text-align:center; font-size: 10pt;'
       align='center' border='1' cellspacing='0' cellpadding='3'
       width='300'>
<tr>
   <td width='25%'>20<br>0.1 s<br>β<sup>+</sup>, p</td>
   <td width='25%'>21<br>122 ms<br>β<sup>+</sup>, p</td>
   <td width='25%'>22<br>3.86 s<br>β<sup>+</sup></td>
   <td width='25%'>23<br>11.32 s<br>β<sup>+</sup></td>
</tr>
<tr>
   <td>31<br>0.25 s<br>β<sup>-</sup>, n </td>
   <td colspan='2' rowspan='2'><font size="+4">Mg</font></td>
   <td>24<br>estável<br>(78.99%)</td>
</tr>
<tr>
   <td>30<br>0.33 s<br>β<sup>-</sup></td>
   <td>25<br>estável<br>(10.00%)</td>
</tr>
<tr>
   <td>29<br>1.3 s<br>β<sup>-</sup></td>
   <td>28<br>21 h<br>β<sup>-</sup></td>
   <td>27<br>9.45 min<br>β<sup>-</sup></td>
   <td>26<br>estável<br>(11.01%)</td>
</tr>
</table>
```

Resultado:

<table>
<tr><td>20
0.1 s
β^+, p</td><td>21
122 ms
β^+, p</td><td>22
3.86 s
β^+</td><td>23
11.32 s
β^+</td></tr>
<tr><td>31
0.25 s
β^-, n</td><td colspan="2" rowspan="2">Mg</td><td>24
estável
(78.99%)</td></tr>
<tr><td>30
0.33 s
β^-</td><td>25
estável
(10.00%)</td></tr>
<tr><td>29
1.3 s
β^-</td><td>28
21 h
β^-</td><td>27
9.45 min
β^-</td><td>26
estável
(11.01%)</td></tr>
</table>

O uso do `rowspan` e do `colspan` não é fácil, uma vez que faz com que a contagem de linhas e colunas em uma tabela não seja mais tão direta. Mesmo desenvolvedores experientes têm dificuldade em produzir tabelas com mesclagens complexas. Só a experiência (e persistência!) vai deixá-lo confortável com isso.

Sua vez.. (8-6)

Reproduza as tabelas a seguir, que mostram os possíveis resultados de combinações lógicas utilizando os operadores E (*and*) e OU (*or*) entre valores que podem ser verdadeiros (V) ou falsos (F). *Dica:* comece pensando nelas como tabelas originalmente com quatro linhas e quatro colunas; que a célula no canto superior esquerdo (que contém **E** ou **OU** abrange 2 × 2 células; que a célula que contém a condição **A** abrange 2 colunas; que a célula que contém a condição **B** abrange duas linhas.

E		**A**	
		V	**F**
B	**V**	V	F
	F	F	F

OU		**A**	
		V	**F**
B	**V**	V	V
	F	V	F

Às vezes é interessante aninhar tabelas dentro de tabelas. No exemplo a seguir é construída uma tabela com 1 linha e 3 colunas, sem bordas, para conter, em cada uma de suas células, novas tabelas, desta vez com duas colunas, um número variável de linhas e bordas aparentes.

Listagem:

```
<p>
<table style='text-align:center' align='center'
       width='500' cellpadding='10'>
<tr>
   <td valign='center'>
      <table style='text-align:center' align='center'
            width='150' cellpadding='5' cellspacing='0'
            border='1'>
      <tr><td><sup>1</sup>H</td><td>estável</td></tr>
      <tr><td><sup>2</sup>H</td><td>estável</td></tr>
      <tr><td><sup>3</sup>H</td><td>12.26 a</td></tr>
      </table>
   </td>
   <td valign='center'>
      <table style='text-align:center' align='center'
            width='150' cellpadding='5' cellspacing='0'
            border='1'>
      <tr><td><sup>3</sup>He</td><td>estável</td></tr>
      <tr><td><sup>4</sup>He</td><td>estável</td></tr>
      <tr><td><sup>6</sup>He</td><td>0.808 s</td></tr>
      <tr><td><sup>8</sup>He</td><td>0.122 s</td></tr>
      </table>
   </td>
   <td valign='center'>
      <table style='text-align:center' align='center'
            width='150' cellpadding='5' cellspacing='0'
            border='1'>
      <tr><td><sup>5</sup>Li</td><td><10<sup>-20</sup> s</td></tr>
      <tr><td><sup>6</sup>Li</td><td>estável</td></tr>
      <tr><td><sup>7</sup>Li</td><td>estável</td></tr>
      <tr><td><sup>8</sup>Li</td><td>0.844 s</td></tr>
      <tr><td><sup>9</sup>Li</td><td>0.178 s</td></tr>
      </table>
   </td>
</tr>
</table>
</p>
```

Resultado:

^{1}H	estável
^{2}H	estável
^{3}H	12.26 a

^{3}He	estável
^{4}He	estável
^{6}He	0.808 s
^{8}He	0.122 s

^{5}Li	$<10^{-20}$ s
^{6}Li	estável
^{7}Li	estável
^{8}Li	0.844 s
^{9}Li	0.178 s

Cores

Os elementos de um documento HTML podem ter cor, tanto no seu conteúdo quanto no seu fundo (*background*). As cores podem ser atribuídas utilizando declarações de estilo (`style`) com os atributos `color` e `background-color`. As cores propriamente ditas podem ser especificadas de três maneiras: através de seus nomes (em inglês), de uma função específica do atributo de estilo ou de sequências de caracteres hexadecimais.

As cores podem ser utilizadas em todos os elementos HTML, tais como parágrafos, linhas e células de tabelas, listas ordenadas e não-ordenadas e em seus itens individualmente etc. O exemplo abaixo ilustra alguns usos. Note em particular o uso do marcador `<span>...</span>` (abrangência) quando se deseja marcar um pedaço de texto dentro de algum outro elemento, como em um parágrafo, por exemplo.

Listagem:

```
<p style='color: white; background-color: black'>
Os atributos de estilo <code>color</code> e
<code>background-color</code> definem a cor do texto
e a cor do fundo, respectivamente. Já quando queremos
que um <span style='color: black; background-color: white'>
trecho qualquer </span> fique diferente utilizamos o
marcador <code>span</code>.
</p>

<ol style='background-color: silver'>
<li> Este item está na cor de fundo padrão da lista toda.
<li style='background-color: gray'> Este item teve a cor
alterada localmente.
<li> Este item está na cor de fundo padrão da lista.
<li> <span style='background-color: gray'> Este item teve
a cor alterada com o uso de <code>span</code>.</span>
<li> Este item está na cor de fundo padrão da lista.
</ol>
```

Resultado:

Os atributos de estilo `color` e `background-color` definem a cor do texto e a cor do fundo, respectivamente. Já quando queremos que um trecho qualquer fique diferente utilizamos o marcador `span`.

1. Este item está na cor de fundo padrão da lista toda.
2. Este item teve a cor alterada localmente.
3. Este item está na cor de fundo padrão da lista.
4. Este item teve a cor alterada com o uso de `span`.
5. Este item está na cor de fundo padrão da lista.

Cores podem ser especificadas em declarações de estilo por nomes predefinidos — "black", "blue", "red", "gray", "silver", "navy", "teal", "purple" e dezenas de outros nomes exóticos. Cores também pode ser definidas através da declaração `rgb(`*R*`,` *G*`,` *B*`)`, onde *R*, *G* e *B* são números entre 0 e 255 (em base 10), ou por sequências hexadecimais do tipo `"#rrggbb"` em que 255 valores (00 a FF) podem ser atribuídos às componentes rr (*red*), gg (*green*) e bb (*blue*) que compõem a cor (dois caracteres hexadecimais para cada uma). Dessa maneira, estão disponíveis cerca de 16 milhões de cores.

O exemplo a seguir especifica as cores com declarações de estilo que utilizam a função `rgb(...)` mas identifica as cores, na tabela, com seus códigos hexadecimais.

Listagem:

```
<table align='center' cellspacing='10'>
<tr>
<td>
<table align="center" border="1" cellspacing="0" cellpadding="0"
       style="text-align:center" width="250">
<tr>
<th> </th><th> R </th><th> G </th><th> B </th>
</tr>
<tr>
 <td style="background-color:rgb(0,0,0)">#000000</td>
 <td>  0</td><td>  0</td><td>  0</td>
</tr>
<tr>
 <td style="background-color:rgb(127,127,127)">#7F7F7F</td>
 <td>127</td><td>127</td><td>127</td>
</tr>
<tr>
 <td style="background-color:rgb(255,255,255)">#FFFFFF</td>
 <td>255</td><td>255</td><td>255</td>
</tr>
<tr>
 <td style="background-color:rgb(255,0,0)">#FF0000</td>
 <td>255</td><td>0</td><td>0</td>
</tr>
<tr>
 <td style="background-color:rgb(0,255,0)">#00FF00</td>
 <td>0</td><td>255</td><td>0</td>
</tr>
</table>
</td>
<td>
<table align="center" border="1" cellspacing="0" cellpadding="0"
       style="text-align:center" width="250">
<tr>
 <th> </th><th> R </th><th> G </th><th> B </th>
</tr>
<tr>
 <td style="background-color:rgb(0,0,255)">#0000FF</td>
 <td>0</td><td>0</td><td>255</td>
</tr>
<tr>
 <td style="background-color:rgb(255,255,0)">#FFFF00</td>
 <td>255</td><td>255</td><td>0</td>
</tr>
<tr>
 <td style="background-color:rgb(255,0,255)">#FF00FF</td>
 <td>255</td><td>0</td><td>255</td>
</tr>
<tr>
 <td style="background-color:rgb(0,255,255)">#00FFFF</td>
 <td>0</td><td>255</td><td>255</td>
</tr>
<tr>
 <td style="background-color:rgb(255,127,63)">#FF7F3F</td>
 <td>255</td><td>127</td><td>63</td>
</tr>
</table>
</td>
</tr>
</table>
```

Resultado:

	R	G	B
	0	0	0
#7F7F7F	127	127	127
#FFFFFF	255	255	255
#FF0000	255	0	0
#00FF00	0	255	0

	R	G	B
#0000FF	0	0	255
#FFFF00	255	255	0
#FF00FF	255	0	255
#00FFFF	0	255	255
#FF7F3F	255	127	63

Note em particular os tons de cinza, indo do preto (`rgb(0,0,0)` ou hexadecimal `#000000`) ao branco (`rgb(255,255,255)` ou `#FFFFFF`), que são gerados fazendo com que as contribuições das três cores sejam iguais.

Sua vez... (8-7)

O script a seguir define uma função que sorteia uma cor aleatória. A função é então utilizada para atribuir uma cor de fundo para um parágrafo que contém apenas um ` ` (*non breakable space*) para que não seja tratado como vazio, de modo que o navegador mostre uma faixa com a cor sorteada, como a que segue.

```
<script>
```

```
function geraCor() {
    var R = parseInt(255*Math.random());
    var G = parseInt(255*Math.random());
    var B = parseInt(255*Math.random());
    var cor = "rgb(" + R + "," + G + "," + B + ")";
    return cor;
}
var estilo = "style='background-color:" + geraCor() + "'";
var str = "<p " + estilo + "> </p>";
document.write(str);
</script>
```

Substitua as reticências (. . .) no script a seguir de modo que ele produza uma tabela com uma linha e quatro colunas, cada uma com uma cor aleatoriamente sorteada.

```
<script>
var str = "";
str += "<table style='width:100%'>";
str += "<tr>";
for (var i=0;i<4;i++) {
    var estilo = ...
    str += ...
}
str += "</tr>";
str += "</table>";
document.write(str);
</script>
```

Exercícios

1. Reproduza o seguinte texto HTML utilizando listas ordenadas e não-ordenadas:

   I. Mecânica Quântica: fundamentos
      A. Partículas e campos
         - Difração de partículas
         - Pacotes de ondas
      B. Princípio da incerteza
         - Relação de incerteza posição-momento
         - Relação de incerteza energia-tempo
   II. Mecânica Quântica: aplicações
      A. Equação de Schröedinger
      B. Potenciais
         - Potencial degrau
         - Poço quadrado infinito
         - Oscilador harmônico

2. Reproduza o seguinte texto HTML, no qual estão embutidos trechos de JavaScript que sorteiam o ângulo (entre 30 e 60 graus) e o módulo da velocidade de lançamento de um projétil (entre 5 m/s e 15 m/s) e calcula a sua altura máxima e o alcance em função dos valores sorteados:

   Em um lançamento oblíquo, um projétil lançado a um ângulo de 48 graus com uma velocidade inicial de 9 m/s vai atingir uma altura máxima de 2.3 m e terá um alcance de 8.2 m. O tempo de vôo é de 1.4 s.

3. Reproduza as equações a seguir utilizando elementos do HTML:

   a. $\text{sen}(A \pm B) = \text{sen}\, A \cos B \pm \cos A \,\text{sen}\, B$

   b. $\text{sen}\,\theta = (e^{i\theta} - e^{-i\theta}) / 2i$

   c. $\int a f(x)\, dx = a \int f(x)\, dx$

   d. $\nabla \times \mathbf{B} = \mu_0 \varepsilon_0 \, d\mathbf{E}/dt$

4. Reproduza as equações a seguir utilizando a sintaxe do LaTeX e o MathJax.js.

   a. \frac{d}{dt} (\frac{\partial L}{\partial \dot{x}_{i}}) - \frac{\partial L}{\partial x_{i}} = 0

   b. T = \frac{1}{2} m v^{2} + \frac{1}{2} I \dot{\theta}^{2}

c. \Psi_{21\pm1} = \frac{1}{8\sqrt{2\pi}} \left(\frac{Z}{a_{0}}\right)^{3/2} \left(\frac{Zr}{a_{0}}\right) e^{-Zr/2a_{0}} \sin\theta \; e^{\pm i\phi}

5. O trecho de script a seguir define uma matriz com 5 linhas e 5 colunas. Cada linha representa os dados de um aluno, a saber, matrícula, notas em três provas e a média final (inicialmente zerada):

```
var aluno = new Array(5);
aluno[0] = ["08234045", 10,  9, 10,  0];
aluno[1] = ["08134033",  8,  7,  3,  0];
aluno[2] = ["09147001",  9,  7,  5,  0];
aluno[3] = ["09147076",  9,  6,  9,  0];
aluno[4] = ["08247804",  7,  7,  7,  0];
```

Complete o script de tal modo que imprima uma tabela semelhante à que segue:

Matrícula	P1	P2	P3	Média
08234045	10.0	9.0	10.0	9.7
08134033	8.0	7.0	3.0	6.0
09147001	9.0	7.0	5.0	7.0
09147076	9.0	6.0	9.0	8.0
08247804	7.0	7.0	7.0	7.0

6. Reproduza a seguinte tabela:

≅	⇒	≈
⇑	→ ↓ ↑ ←	⇓
∂	⇐	∞

Dica: é uma tabela 4 × 4 com os atributos `colspan` e `rowspan` para as células estrategicamente utilizados. Procure *html entities* na internet para obter os códigos dos símbolos apresentados na tabela.

7. As figuras abaixo mostram, à esquerda, o esquema de composição de cores aditivo, que é o que ocorre quando somamos luz, como em um monitor de vídeo, e, à direita, o esquema subtrativo, que é o que ocorre quando somamos tintas, como em uma impressora. No esquema aditivo as cores primárias são o vermelho, o verde e o azul. O vermelho com o verde leva ao amarelo, o vermelho com o azul leva ao magenta, o verde com o azul leva ao ciano e a soma dos três leva ao branco. No esquema subtrativo as cores primárias são o ciano, o magenta e o amarelo. O ciano com o magenta leva ao azul, o ciano com o amarelo leva ao verde, o magenta e o amarelo leva ao vermelho e a soma dos três leva ao preto.

As figuras foram montadas como duas tabelas de 3 linhas com 5 colunas cada. Reproduza-as!

8. O script abaixo escreve uma mensagem com letras vermelhas em meio a um conjunto de letras escolhidas e coloridas aleatoriamente. Se você pegar um pedaço de papel celofane vermelho (ou qualquer outro material translúcido vermelho) verá que a mensagem fica realçada enquanto as letras de outras cores ficam esmaecidas.

```
<script>
var cor = new Array("blue", "yellow", "green");
var msg = "AS CORES QUE VOCÊ VÊ DEPENDEM DA LUZ QUE VOCÊ USA";
var txt = "<p style='font-family:cursive; font-weight:bold;'>";
for (var i=0; i<(msg.length*4); i++) {
   var icor = parseInt(Math.random()*3);
   var ltr = String.fromCharCode(65+parseInt(Math.random()*26));
   txt += "<font color='" + cor[icor];
   txt += "'>" + ltr + "</font>" + " ";
   if ((i%4)==0) {
      txt += "<font color='red'>";
      txt += msg.charAt(i/4);
```

```
            txt += "</font>" + " ";
        }
    }
    txt += "</p>";
    document.write(txt);
    </script>
```

O script cria e inicializa uma matriz cujos três elementos são as cores alternativas à da mensagem; cria e inicializa a mensagem a ser apresentada em vermelho; cria a variável que vai conter o texto final a ser impresso e inicializa-a com a formatação do parágrafo (fonte e peso das letras). Um laço `for` executado um número de vezes 4 vezes maior do que o tamanho da mensagem sorteia uma letra (um código entre 65 e 91) e em seguida um número entre 0 e 2 que vai indexar a cor da letra; adiciona a letra sorteada especificando a sua cor e finalmente, a cada quatro interações, coloca uma letra da mensagem. Findo o laço, encerra o parágrafo e imprime a mensagem.

Modifique o script de modo que imprima 3 mensagens diferentes, uma vermelha, uma amarela e outra verde e que inclua mais duas cores a serem sorteadas aleatoriamente.

9 Interação

Neste capítulo você vai aprender a utilizar elementos interativos tais como botões, caixas de textos, listas com opções de escolha e figuras interativas em seu documento. Vai aprender também como capturar e gerenciar informações de eventos, tais como movimentos e cliques do mouse e teclas digitadas.

`input type="button"`

É possível criar documentos interativos adicionando funções gerenciadoras de eventos aos elementos de um documento. No exemplo a seguir, o botão muda de cor quando o ponteiro do mouse está sobre e ele e a função `sorteia()` é acionada quando for clicado:

Listagem:

```
<script>
function sorteia() {
   var num = parseInt(1 + Math.random()*10);
   alert("Você deve resolver a questão número " + num);
}
</script>

<p style='text-align:center'>
Clique no botão abaixo para receber sua tarefa:<br>
<input
    type='button'
    value='BOA SORTE'
    style = "background-color:lightgray"
    onClick="sorteia()"
    onMouseOver="this.style.backgroundColor = 'lightgreen'"
    onMouseOut="this.style.backgroundColor = 'lightgray'"
    />
</p>
```

Resultado:

Clique no botão abaixo para receber sua tarefa:

BOA SORTE

O elemento utiliza três gerenciadores de eventos para controlar o que acontece quando o usuário clica sobre o botão (`onClick`), quando o mouse passa sobre o botão (`onMouseOver`) e quando o mouse deixa o botão (`onMouseOut`). No primeiro caso, ao gerenciador de eventos foi atribuída a tarefa de chamar a função `sorteia()`. Nos outros dois, uma instrução JavaScript foi diretamente atribuída aos gerenciadores de eventos. A palavra `this` (este) nessas instruções indica que a operação deve ser realizada sobre o elemento que gerou o evento, que é o próprio botão.

Note que no atributo `style` a propriedade que define a cor de fundo é escrita com um hífen entre `background` e `color`, e não importa se letras maiúsculas ou minúsculas são utilizadas. Já na instrução JavaScript, a propriedade é escrita sem o hífen e as maiúsculas e minúsculas devem ser estritamente respeitadas. Isso porque o `background-color` está no contexto do HTML, no qual um hífen é somente um caracter, enquanto o `backgroundColor` está no contexto do JavaScript, em que o hífen é, na realidade, um sinal de subtração.

Sua vez... (9-1)

Modifique o exemplo anterior de modo que não aconteça nada quando o botão for clicado, mas sim quando o mouse passar sobre ele e, uma vez sorteada a questão, o botão fique permanentemente vermelho.

São vários os gerenciadores de eventos disponíveis, e entre os mais usuais estão:

onClick	Clique com o mouse
onDblClick	Clique duplo com o mouse
onChange	Conteúdo de elemento modificado
onBlur	Elemento perde o foco
onFocus	Elemento recebe o foco
onMouseMove	O mouse anda
onMouseDown	Botão pressionado
onMouseUp	Botão levantado
onMouseOver	Mouse sobre o elemento
onMouseOut	Mouse sai do elemento
onLoad	Um documento ou imagem é carregada
onMouseOut	Mouse sai do elemento
onKeyDown	Uma tecla é pressionada
onKeyPress	Uma tecla é pressionada e mantida assim
onKeyUp	Uma tecla é liberada
onResize	Uma janela é redimensionada
onSelect	Um texto é selecionado

Em alguns casos, quando um destes gerenciadores de evento é chamado, é importante saber em que condições isso aconteceu, por exemplo, qual a posição do mouse, qual o botão acionado ou, no caso do teclado, que tecla adicional (`CTRL`, `ALT` etc.) estava simultaneamente pressionada.

altKey	Se a tecla "ALT" estava pressionada
ctrlKey	Se a tecla "CTRL" estava pressionada
shiftKey	Se a tecla "SHIFT" estava pressionada
button	Que botão do mouse foi pressionado
clientX	Coordenada horizontal do mouse
clientY	Coordenada vertical do mouse

O exemplo anterior pode ser modificado para de modo a apresentar uma mensagem diferente para cada tecla extra pressionada:

Listagem:

```
<script>
function questao(event) {
   var num = parseInt(1 + Math.random()*10);
   if (event.altKey) alert("ALT " + num );
   else if (event.ctrlKey) alert("CTRL " + (num+10) );
   else if (event.shiftKey) alert("SHIFT " + (num+20) );
   else alert(num);
}
</script>

<p>
Clique no botão abaixo para receber sua tarefa.
São três grupos de questões:
<ul>
<li>1-10: clique no botão com a tecla "ALT" pressionada
<li>11-20: clique no botão com a tecla "CTRL" pressionada
<li>21-30: clique no botão com a tecla "SHIFT" pressionada
</ul>
</p>

<p style='text-align:center'>
<input type='button' value='BOA SORTE' onClick="questao(event)">
</p>
```

Resultado:

Clique no botão abaixo para receber sua tarefa. São três grupos de questões:

- 1-10: clique no botão com a tecla "ALT" pressionada
- 11-20: clique no botão com a tecla "CTRL" pressionada
- 21-30: clique no botão com a tecla "SHIFT" pressionada

BOA SORTE

Sua vez... (9-2)

Modifique o exemplo anterior de modo a fazer com que o botão fique vermelho, verde ou azul quando as telcas ALT, CTRL e SHIFT estiverem pressionadas, respectivamente, quando o botão for clicado.

`input type="text"`

Outro importante elemento de interatividade em um documento HTML é o `input` do tipo `text`, conhecido como *caixa de texto*. No exemplo a seguir, é colocada no documento uma caixa de texto que solicita ao usuário que digite o seu nome e, a seguir, coloca-o todo em letras maiúsculas.

Listagem:

```
<script>
function valida() {
   var ref = document.getElementById('nome');
   ref.value = (ref.value).toUpperCase();
}
</script>


<p style='text-align:center'>
Digite o seu nome e pressione ENTER

<input type='text' size='20' id='nome'
      value='<digite seu nome>' onChange='valida()'>
</p>
```

Resultado:

Digite o seu nome e pressione ENTER

<digite seu nome>

Ao ser detetada uma mudança (`onChange`) no conteúdo da caixa de texto, a função `valida()` é chamada. Esta função usa o método `getElementById()` para buscar no documento um elemento com a identidade (`id`) `"nome"` e coloca uma referência a ele na variável `ref`. Esta referência é essencialmente o "endereço" da caixa de texto dentro do documento (cada elemento de um documento HTML tem um endereço único que pode ser referenciado). Na linha seguinte, o atributo `value`, que contém o conteúdo da caixa, é transformado para letras maiúsculas e o valor resultante da transformação atribuído novamente ao atributo `value` do elemento.

As instruções na função `valida()` fazem algo parecido com o que faz a instrução `x = x + 1`, que pega o valor da variável `x`, modifica-o e coloca o resultado na própria variável `x`).

Sua vez... (9-3)

Modifique o exemplo anterior de modo a inserir outra caixa de texto com outro `id` e fazer com que a função `valida()` transfira o texto digitado na primeira caixa para a outra.

Combinações de caixas de texto com botões permitem a construção de uma calculadora elementar:

Listagem:

```
<script>
function operacao(opt) {
   var op1 = parseFloat(document.getElementById('op1').value);
   var op2 = parseFloat(document.getElementById('op2').value);
   var res = document.getElementById('res');
   switch (opt) {
```

```
      case '+': res.value = (op1 + op2).toFixed(2);
                break;
      case '-': res.value = (op1 - op2).toFixed(2);
                break;
      case '*': res.value = (op1 * op2).toFixed(2);
                break;
      case '/': res.value = (op1 / op2).toFixed(2);
                break;
   }
}
</script>

<p style='text-align:center'>
<input type='text' size='3' id='op1' value=''>
<input type='button' value='+' onClick="operacao('+')">
<input type='button' value='-' onClick="operacao('-')">
<input type='button' value='*' onClick="operacao('*')">
<input type='button' value='/' onClick="operacao('/')">
<input type='text' size='3' id='op2' value=''> =
<input type='text' size='3' id='res' value=''>
</p>
```

Resultado:

Note que:

- Diferentemente do exemplo anterior, as variáveis `op1` e `op2` já guardam o valor contido na caixa de texto e não o seu endereço. Já a variável `res`, por conveniência, guarda o endereço da caixa onde será escrita a resposta.
- O método `parseFloat()` é utilizá-lo para garantir que os valores digitados nas caixas sejam transformados em números. Se isso não for feito, a soma de '2' com '3' dá '23' e não 5, como desejado.

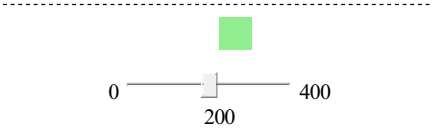
`input type="range"`

Elementos do tipo `range` correspondem a barras deslizantes normalmente utilizadas para varrer continuamente os possíveis valores de uma variável. O exemplo a seguir mostra como utilizar esse elemento para controlar a posição de uma divisão móvel dentro de uma divisão fixa.

Listagem:

```
<div id="rngDiv" style="width:400;margin:auto"></div>

<div align="center" style="margin-top:1em">
0 <input id="rngCtrl" type="range" min="0" max="10" value="5"
         onChange="atualizaRange()" /> 400
</div>

<div align="center" id="rngValue"></div>

<script>
function atualizaRange() {
    var x = parseInt(document.getElementById("rngCtrl").value)*40;
    document.getElementById("rngValue").innerHTML = x;
    var estilo = "position:relative;";
    estilo += "left:" + x + ";";
    estilo += "width:25;height:25;background-color:lightgreen";
    var str = "<div style=" + estilo + "></div>";
    document.getElementById("rngDiv").innerHTML = str;
}
atualizaRange();
</script>
```

Resultado:



Na seção HTML do exemplo são definidas três divisões, todas centralizadas. A primeira divisão, `rngDiv`, de coordenadas fixas no documento, foi criada vazia para receber dentro dela a divisão móvel (sem `id`) a ser construída dentro do script quando a posição da barra for modificada. A segunda divisão (também sem `id`) foi criada apenas para conter e centralizar a barra

deslizante e seus valores extremos (0 e 400). A terceira divisão, `rngValue` foi criada para receber dentro dela o valor da posição atual do cursor da barra deslizante.

> Note que são utilizadas duas maneiras distintas de centralizar uma divisão. Na primeira divisão isso é feito atribuindo-se à divisão um tamanho e impondo que as margens devem ser automaticamente escolhidas, o que leva à centralização do conteúdo. Na segunda e na terceira divisões isso é feito atribuindo o valor `center` ao parâmetro `align` do elemento `div`.

Na seção que contém o script é definida a função `atualizaRange()`, que responde ao evento `onChange` da barra deslizante. Após a definição, a função é chamada uma vez para que apareça no documento, ao ser carregado, a divisão móvel e sua respectiva posição na barra deslizante.

A função `atualizaRange()` começa atribuindo à variável `x` o valor corrente do controle deslizante, multiplicado por 40. Na instrução seguinte, atribui à div `rngValue` esse valor, para informar ao usuário a posição corrente do cursor. Em seguida, na variável `estilo` é construída uma declaração de estilo que (a) estabelece que as coordenadas que lhe serão atribuídas devem ser interpretadas como relativas à posição da divisão que a contém; (b) ajusta a largura e (c) a altura da divisão a ser inserida dinamicamente no documento para 25 pixels e finalmente (d) dá-lhe um verde claro como cor de fundo. A penúltima linha da função constrói a divisão móvel que será incluída na divisão fixa através da instrução na última linha da função.

> Note que os parâmetros `min`, `max` e `value` do `input type="range"` somente admitem valores inteiros, o que determina seus pontos de parada. No caso, como os extremos foram definidos entre 0 e 10, existem 11 paradas (0, 1, 2, ..., 10), que correspondem às posições 0, 40, 80, ..., 400 relativas ao início da divisão fixa.

Sua vez... (9-4)

Modifique o exemplo anterior de modo a atribuir uma altura de 400 pixels para a divisão fixa e incluir uma segunda barra deslizante que controle a posição vertical da divisão móvel.

`input type="checkbox"`

O elemento `checkbox` permite ao usuário escolher uma ou mais alternativas de um grupo, como no exemplo abaixo.

Listagem:

```
<script>
function checkBoxes() {
   var a1 = document.getElementById("cb1").checked;
   var a2 = document.getElementById("cb2").checked;
   var a3 = document.getElementById("cb3").checked;
   var a4 = document.getElementById("cb4").checked;
   if (a1&&a2&&a4&&!a3) alert("Parabéns!");
   else alert("Tente novamente!");
}
</script>

<p>Quais das seguintes unidades são unidades de energia?
<ol>
<input type='checkbox' id='cb1'> joule <br>
<input type='checkbox' id='cb2'> erg <br>
<input type='checkbox' id='cb3'> watt <br>
<input type='checkbox' id='cb4'> kwh <br>
</ul>
<input type='button' value=' OK ' onClick='checkBoxes()'>
```

Resultado:

Quais das seguintes unidades são unidades de energia?

a. ☐ joule
b. ☐ erg
c. ☐ watt
d. ☐ kwh

OK

Neste exemplo, são criados quatro elementos `input type="checkbox"`, cada um com o seu próprio `id`, através do qual pode ser inequivocamente identificado. Quando o usuário clica no botão OK, a função `checkBoxes()` é acionada. Essa função cria quatro variáveis que recebem os estados de cada um dos elementos. Caso o elemento tenha sido marcado pelo usuário, a chamada a `document.getElementById(id).checked` retorna `true`; caso contrário, retorna `false`. O `if` combina as quatro respostas com operadores "E" (`&&`) e negação (`!`) para verificar se o usuário marcou as respostas corretas e não marcou a resposta errada.

Sua vez... (9-5)

Modifique o exemplo anterior de modo que atribua uma nota de 0 a 3 para o usuário de duas maneiras: (a) sem prejuízo quando o usuário marcar a alternativa incorreta e (b) com prejuízo quando o usuário marcar a alternativa incorreta (isto é, se todas as alternativas forem marcadas a nota é 2).

Uma possível aplicação deste tipo de estrutura é contabilizar o número de escolhas que o usuário faz. No exemplo a seguir, não existem respostas certas ou erradas, apensas deseja-se contar o número de opções escolhidas.

Listagem:

```
<script>
function contaBoxes() {
   var cnt = 0;
   for (var i=1;i<=5;i++) {
      var str = "e" + i;
      var box = document.getElementById(str);
      if (box.checked) cnt++;
   }
   alert(cnt);
}
</script>

<p>Quais dos seguintes elementos químicos, em estado puro,
você já <b>viu</b> com seus próprios olhos?

<ol type='a'>
<li><input type='checkbox' id='e1'> hidrogênio <br>
<li><input type='checkbox' id='e2'> carbono <br>
<li><input type='checkbox' id='e3'> oxigênio <br>
<li><input type='checkbox' id='e4'> ferro <br>
<li><input type='checkbox' id='e5'> ouro <br>
</ol>

<input type='button' value=' OK ' onClick='contaBoxes()'>
```

Resultado:

Quais dos seguintes elementos químicos, em estado puro, você já **viu** com seus próprios olhos?

a. ☐ hidrogênio
b. ☐ carbono
c. ☐ oxigênio
d. ☐ ferro
e. ☐ ouro

OK

Sua vez... (9-6)

Modifique o exemplo anterior de modo a incluir nos elementos `input` o atributo `value`, atribuindo a eles os valores dos números atômicos dos elementos. Utilize o comando `document.getElementById(...).value` para coletar essa informação dos itens marcados e um `window.alert()` para apresentar os números atômicos dos elementos observados.

`input type="radio"`

O elemento `radio` é muito semelhante ao `checkbox`, mas permite fazer com que diversos

elementos formem um grupo de modo que quando algum deles é marcado, os outros do mesmo grupo ficam desmarcados. O pertencimento a um grupo é definido pelo uso do atributo `name`, como no exemplo a seguir.

Listagem:

```
<script>
function checkradio() {
   if (document.getElementById("r3").checked)
      alert("Parabéns!");
   else
      alert("Tente novamente!");
}
</script>

<p>Qual das seguintes unidades NÃO é uma unidade de energia?
<ol>
<input type='radio' name='grupo' id='r1'> joule <br>
<input type='radio' name='grupo' id='r2'> erg <br>
<input type='radio' name='grupo' id='r3'> watt <br>
<input type='radio' name='grupo' id='r4'> kwh <br>
</ul>
<input type='button' value=' OK ' onClick='checkradio()'>
```

Resultado:

Qual das seguintes unidades NÃO é uma unidade de energia?

- joule
- erg
- watt
- kwh

OK

Este elemento pode ser utilizado para avaliar, por exemplo, o número de questões que o usuário acertou ou algo semelhante. No exemplo abaixo, é simulada uma pergunta para a qual o usuário dá uma nota entre 1 e 3 a diferentes itens de uma pesquisa e obtém a pontuação total ao clicar no botão.

Listagem:

```
<script>
function pontua() {
   var total = 0;
   for (var i=1;i<=3;i++) {
      for (var j=1;j<=3;j++) {
         var str = "q" + i + j;
         var item = document.getElementById(str);
         if (item.checked) total = total + parseInt(item.value);
      }
   }
   alert(total);
}
</script>

<table style='text-align:center; width:400'>
<tr>
   <th>Item</th>
   <th>Ruim</th>
   <th>Regular</th>
   <th>Bom</th>
</tr>
<tr>
   <td>As instalações físicas são adequadas?</td>
   <td><input type='radio' name='q1' id='q11' value='1'></td>
   <td><input type='radio' name='q1' id='q12' value='2'></td>
   <td><input type='radio' name='q1' id='q13' value='3'></td>
</tr>
<tr>
   <td>O livro didático é de boa qualidade?
   <td><input type='radio' name='q2' id='q21' value='1'></td>
   <td><input type='radio' name='q2' id='q22' value='2'></td>
   <td><input type='radio' name='q2' id='q23' value='3'></td>
</tr>
<tr>
   <td>O professor está bem preparado?
   <td><input type='radio' name='q3' id='q31' value='1'></td>
   <td><input type='radio' name='q3' id='q32' value='2'></td>
   <td><input type='radio' name='q3' id='q33' value='3'></td>
</tr>
</table>
```

```
<p style='text-align:center'>
<input type='button' value=' OK ' onClick='pontua()'>
</p>
```

Resultado:

Item	Ruim	Regular	Bom
As instalações físicas são adequadas?	○	○	○
O livro didático é de boa qualidade?	○	○	○
O professor está bem preparado?	○	○	○

OK

Cada questão tem seu grupo de respostas formado pela atribuição do mesmo `name` a todos os seus itens e que os `id` são todos diferentes, uma vez que dois elementos do documento não podem ter o mesmo `id`. Na função `pontua()`, foram utilizados os índices `i` e `j` dos dois laços para a construção de um literal `str` que contém cada um desses `id`'s à medida que os laços progridem. Como duas propriedades dos elementos são utilizadas (`checked` e `value`), é conveniente armazenar a referência obtida pela chamada a `getElementById()` em uma variável `item`, que a seguir é utilizadas para a obtenção dos valores das duas propriedades.

A propriedade `value` é sempre interpretada como um literal e é convertida para um número inteiro utilizando `parseInt()`. Caso isso não fosse feito, o resultado de '2'+'3'+'1', por exemplo, seria '231' e não 6, como esperado.

Sua vez... (9-7)

(a) Invente e inclua mais duas perguntas no exemplo anterior e altere a função `pontua()` de modo que o novo script funcione apropriadamente. (b) Modifique os valores dos itens para −1, 0 e 1 e adicione instruções à função `pontua()` de modo que emita um aviso (`window.alert()`) com a mensagem "reprovado" caso a pontuação final seja negativa, "aprovado" caso seja positiva e "em análise" caso seja nula.

`input type="select"`

Outro elemento interativo é o `select`, que oferece um menu de opções a serem escolhidas. O exemplo a seguir mostra como estruturar as opções e mostrar a escolha do usuário.

Listagem:

```
<script>
function confirma() {
   var opt = document.getElementById("parts").value;
   if (opt=="eletron") window.alert("Elétron");
   else if (opt=="proton") window.alert("Próton");
}
</script>

<p style='text-align:center'>
<select id="parts" onChange="confirma()">
  <option value="escolha">-- escolha --</option>
  <option value="eletron">Elétron</option>
  <option value="proton">Próton</option>
</select>
</p>
```

Resultado:

-- escolha --

Note o uso de um `else if` para evitar que o programa evoque um `window.alert()` quando, ao fazer uma nova escolha, o usuário selecione `-- escolha --`.

Sua vez... (9-8)

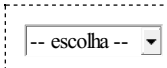
Modifique o exemplo acima de modo a incluir mais partículas (por exemplo, o nêutron) e fazer com que, quando o usuário voltar a escolher `-- escolha --`, o programa, ao invés de não fazer nada, emita um aviso do tipo "Você deve escolher uma partícula!"

O exemplo a seguir ilustra o uso do `select/option` com outro importante recurso do HTML/JavaScript, a propriedade `innerHTML` do objeto `document`. Esta propriedade pode ser utilizada para modificar o conteúdo dos elementos de um documento depois de ser carregado.

Listagem:

```
<script>
function processa() {
   var opt = document.getElementById("particulas").value;
   var str;
   switch (opt) {
      case "eletron":
         str  = "<i>m</i> = 9.11 × 10<sup>-31</sup> kg<br>";
         str += "<i>q</i> = -1.60 × 10<sup>-19</sup> C";
         break;
      case "proton":
         str  = "<i>m</i> = 1.67 × 10<sup>-27</sup> kg<br>";
         str += "<i>q</i> = 1.60 × 10<sup>-19</sup> C";
         break;
   }
document.getElementById("propriedades").innerHTML = str;
}
</script>

<table width="300">
<tr>

   <td style='vertical-align:top; width:150px; height:50px;'>
   <select id="particulas" onChange="processa()">
      <option value="escolha">-- escolha --</option>
      <option value="eletron">Elétron</option>
      <option value="proton">Próton</option>
   </select>
   </td>

   <td style='vertical-align:top; width:150px; height:50px;'>
   <div id="propriedades"></div>
   </td>

</tr>
</table>
```

Resultado:



O exemplo contém um bloco que define a função `processa()` e outro bloco que define uma tabela. A tabela contém apenas uma linha (delimitada por `<tr>...</tr>`) com duas células (delimitadas por `<td>...</td>`). A caixa de escolha `select/option` está dentro da primeira célula. A outra célula contém uma divisão (`<div>...</div>`) vazia, mas identificada (`id="propriedades"`). Esta divisão "reserva" um espaço que permite mostrar conteúdos gerados *depois* que o documento tenha sido carregado pelo navegador. A inserção de conteúdo nesta divisão é feita atribuindo o conteúdo desejado à propriedade `innerHTML` de objeto do documento.

A caixa de seleção tem três opções, sendo a primeira apenas um valor neutro que define o tamanho da caixa (é maior que todas as outras opções) e força o usuário a de fato fazer uma escolha. A função `processa()` usa o método `getElementById()` para obter que escolha fez o usuário, guardando-a na variável `opt`. Um `switch...case` constrói um literal (`str`) cujo conteúdo depende da escolha do usuário. Finalmente, este conteúdo é atribuído à propriedade `innerHTML`, que coloca o texto no objeto especificado em `getElementById()`, no caso a divisão `"propriedades"`.

Note a diferença entre as duas primeiras linhas dentro de cada `case`: a primeira tem somente um sinal de igual (=), que atribui a literal entre aspas à variável `str`; a segunda tem um sinal de mais seguido por um sinal de igual (+=), que atribui à variável `str` o seu conteúdo atual (definido na linha anterior) concatenado à litarel entre aspas (é a forma contraída de `str = str + "..."`).

Sua vez... (9-9)

Se, no exemplo anterior, você voltar a selecionar `-- escolha --` verá que na área destinada às informações sobre a partícula aparecerá o termo `undefined`. Modifique a função `processa()` de modo a introduzir mais um `case` que ao invés disso mostre algo como "escolha uma partícula".

`textarea`

O elemento `textarea` é um recurso para o processamento de textos não formatados com múltiplas linhas. O exemplo a seguir define duas caixas de texto para que o usuário digite o número de linhas e de colunas de uma tabela, uma área de texto para que ele entre com os conteúdos das células, separados por vírgulas e quebras de linha, e uma área de texto onde o script imprime a tabela resultante que pode ser recortada e colada em arquivo HTML.

Listagem:

```
<p>
Nro. de linhas: <input type='text' id='nlin' value='3' size='3'>
Nro. de colunas: <input type='text' id='ncol' value='3' size='3'>
</p>
<p>
<textarea rows='5' id='inarea' cols='60'>
11, 12, 13
21, 22, 23
31, 32, 33
</textarea>
</p>
<p><input type='button' value='FORMATA' onClick='formataTabela()'>
<p>
<textarea rows='7' id='outarea' cols='60'>
</textarea>
</p>

<script>
function formataTabela() {
    var nlin  = parseInt(document.getElementById("nlin").value);
    var ncol  = parseInt(document.getElementById("ncol").value);
    var intxt = document.getElementById("inarea").value;
    var linhas = (intxt.split("\n")).toString();
    var dados  = linhas.split(",");

    var str = "<table>\n";
    for (var i=0;i<nlin;i++) {
       str += "<tr>";
       for (var j=0;j<ncol;j++) {
       str += "<td>" + dados[i*ncol+j] + "</td>";
       }
       str += "</tr>\n";
    }
    str += "</table>";

    document.getElementById("outarea").value = str;
}
</script>
```

Resultado:

Nro. de linhas: 3 Nro. de colunas: 3

```
11, 12, 13
21, 22, 23
31, 32, 33
```

FORMATA

```
<table>
<tr><td>11</td><td> 12</td><td> 13</td></tr>
<tr><td>21</td><td> 22</td><td> 23</td></tr>
<tr><td>31</td><td> 32</td><td> 33</td></tr>
</table>
```

Os elementos `textarea` têm dois atributos fundamentais que são o número de linhas (`rows`) e o número de colunas (`col`). No exemplo, eles também têm identificadores únicos (`id`) que são utilizados pelo método `document.getElementById()` para ler ou escrever seu conteúdo.

Depois de buscar nos elementos o número de linhas, de colunas e o texto digitado na caixa de entrada, o script usa o método `split("\n")` do objeto `String` para remover as quebras de linha (`"\n"`). O método retorna uma matriz com três elementos (as três linhas do texto digitado). Na mesma instrução é aplicado a esta matriz o método `toString()`, que transforma a matriz em uma única literal, colocando vírgulas nas junções. O resultado disso é que a variável `linhas` fica com um literal formado pelos conteúdos das células, separados por vírgula. Finalmente é utilizado o método `split(",")` na variável `linhas`, separando seu conteúdo em uma matriz com 9 elementos denominada `dados`. O segundo bloco da função declara a variável `str` que irá receber a tabela formatada.

Note que, como a matriz `dados` é unidimensional, sua indexação é feita utilizando `[i*ncol+j]`, que tem o mesmo efeito que `[i][j]` teria no caso de uma matriz bidimensional.

Sua vez... (9-10)

Experimente utilizar o exemplo anterior colocando na área de entrada (a) todos os dados em uma única linha, separados por vírgula e (b) um dado em cada linha. O algoritmo funciona corretamente? Que modificações você faria para que funcionasse **exclusivamente** em um caso ou no outro?

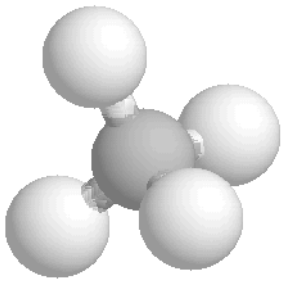
`map/usemap`

Mapas são uma maneira de dar interatividade a uma figura. O exemplo abaixo mostra uma figura que contém uma representação de uma molécula de metano (CH_4). Quando o usuário clica sobre um dos "átomos", uma janela de alerta (`window.alert()`) é apresentada com o nome do átomo:

Listagem:

```
<p style='text-align:center'>
<map name="mapa">
<area shape="circle" coords="130,140,20"
        onClick="window.alert('Carbono')"    >
<area shape="circle" coords="91,60,20"
        onClick="window.alert('Hidrogênio')" >
<area shape="circle" coords="200,120,20"
        onClick="window.alert('Hidrogênio')" >
<area shape="circle" coords="170,190,20"
        onClick="window.alert('Hidrogênio')" >
<area shape="circle" coords="60,190,20"
        onClick="window.alert('Hidrogênio')" >
</map>
<img src="js-interacao-metano.gif" border="0" usemap="#mapa">
</p>
```

Resultado:



O atributo `area` pode receber quatro valores: `default` (toda a figura), `rect` (retângulo), `circle` (círculo) e `poly` (polígono). Para um retângulo, o atributo `coords` deve receber `x1,y1,x2,y2` que são as coordenadas, em pixels, do canto superior esquerdo e do canto inferior direito da região mapeada; para um círculo, deve receber `xc,yc,r`, as coordenadas do centro do círculo e seu raio; para um polígono, deve receber `x1,y1,x2,y2,...,xn,yn`, as coordenadas

dos vértices do polígono. O mapa deve ter um nome (`name`) que será utilizado pelo `usemap` (com o caracter "#" na frente) no elemento que especifica a figura.

> As coordenadas das regiões de interesse da figura podem ser obtidas utilizando-se um editor de imagens tipo Paint, Gimp e similares.

Sua vez... (9-11)

Substitua a imagem do exemplo por uma foto de seu gosto e acrescente um mapa de elementos identificando pessoas, objetos, lugares etc.

`clientX, clientY`

A posição do mouse é um outro elemento de interação que pode ser útil. O script a seguir mostra como fazer para obter a posição do mouse no documento e mostrá-la em duas caixas de texto:

Listagem:

```
<script>
document.onmousemove = getMouseXY;

function getMouseXY(event) {
   document.getElementById("MouseX").value = event.clientX;
   document.getElementById("MouseY").value = event.clientY;
}
</script>

<p style="text-align:center">
<i>x</i>: <input type="text" id="MouseX" value="0" size="3">
<i>y</i>: <input type="text" id="MouseY" value="0" size="3">
</p>
```

Resultado:

x: 0 *y*: 0

O script começa adicionando ao método padrão `onmousemove` a função `getMouseXY`, definida pelo usuário. A função definida pelo usuário recebe como parâmetro uma referência ao objeto `event`, que traz informações sobre eventos detetados pelo navegador. O objeto `event` tem algumas propriedades, entre elas a posição *x*,*y* do mouse, armazenadas em `clientX` e `clientY`.

> As coordenadas fornecidas são relativas ao canto superior esquerdo da parte do documento que aparece na tela. Isto significa que se você mover o documento utilizando a barra de rolagem o referencial das coordenadas muda.

Sua vez... (9-12)

Modifique o exemplo anterior de modo que a função emita um aviso quando o mouse (a) estiver em uma região compreendendo aproximadamente o terço central da tela e (b) estiver fora dessa região (se sua tela tiver 800 pixels de largura e 600 de altura, o terço central será **aproximadamente** definido pelo retângulo cujos cantos superior esquerdo e inferior direito têm coordenadas (270,200) e (540,400), respectivamente.

No exemplo a seguir, os métodos `onmousedown` (pressionar um botão do mouse) e `onmouseup` (soltar um botão do mouse) são substituídos por funções especificadas pelo usuário. A função `onmousedown` guarda as coordenadas do ponto onde o botão do mouse foi abaixado e a função `onmouseup` guarda as coordenadas do ponto onde o botão foi liberado, calculando e apresentando a distância, em pixels, entre esses pontos.

Listagem:

```
<script>
document.onmousedown = mouseDown;
document.onmouseup = mouseUp;
```

```
var iniX, iniY, fimX, fimY;

function mouseDown(event) {
   iniX = event.clientX;
   iniY = event.clientY;
}

function mouseUp(event) {
   fimX = event.clientX;
   fimY = event.clientY;
   var dist = (fimX-iniX)*(fimX-iniX) + (fimY-iniY)*(fimY-iniY);
   dist = Math.sqrt(dist);
   window.alert(dist.toFixed(0) + " pixels");
}
</script>
```

Sua vez... (9-13)

Elimine do exemplo anterior as funções `mouseUp(event)` e `mouseDown(event)` substituindo-as por uma única função `mouseClick(event)` que deve ser atribuída ao gerenciador padrão de cliques do documento através da instrução `document.onclick = mouseClick;`. Essa função deve capturar as coordenadas de dois cliques distintos e mostrar a distância entre eles. *Dica:* crie uma variável global booleana cujo valor (`true` ou `false`) é verificado a cada clique: se for `true`, armazena as coordenadas do primeiro ponto e armazena a variável para `false`; se for `false`, armazena as coordenadas do segundo ponto e ajusta a variável para `true`.

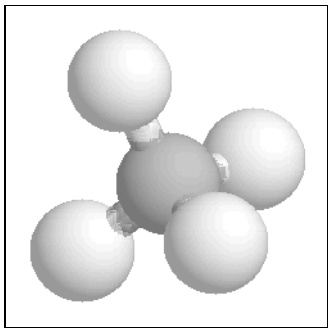
O exemplo a seguir ilustra como transformar as coordenadas do mouse, fornecidas pelo navegador, para o sistema de coordenadas de uma figura arbitrária.

Listagem:

```
<p>
<table align="center" cellpadding="10">
<tr>
<td>
<p style="text-align:center; border:1px solid;">
<img src="js-interacao-metano.gif">
</p>
</td>
<td>
<p>
Canto superior esquerdo:<br>
<i>x</i>: <input type="text" id="xCH41" value="0" size="3">
<i>y</i>: <input type="text" id="yCH41" value="0" size="3">
</p>
<p>
Canto inferior direito:<br>
<i>x</i>: <input type="text" id="xCH42" value="0" size="3">
<i>y</i>: <input type="text" id="yCH42" value="0" size="3">
</p>
<p style="text-align:center">
<input type="button" value="CALIBRAR" onClick="calibraCH4()">
</p>
<p>
Posição do mouse descalibrada:<br>
<i>x</i>: <input type="text" id="xCH4" value="0" size="3">
<i>y</i>: <input type="text" id="yCH4" value="0" size="3">
</p>
<p>
Posição do mouse calibrada:<br>
<i>x</i>: <input type="text" id="xCH4Cal" value="0" size="3">
<i>y</i>: <input type="text" id="yCH4Cal" value="0" size="3">
</p>
</td>
</tr>
</table>
</p>

<script>

var AxCH4 = 0;
var BxCH4 = 1;
var AyCH4 = 0;
var ByCH4 = 1;
document.onmousemove = getMouseCH4;

function getMouseCH4(event) {
   var docX = event.clientX;
   var docY = event.clientY;
   document.getElementById("xCH4").value = docX
   document.getElementById("yCH4").value = docY;
   var calX = AxCH4 + BxCH4 * docX;
   var calY = AyCH4 + ByCH4 * docY;
```

```
   document.getElementById("xCH4Cal").value = calX.toFixed(0);
   document.getElementById("yCH4Cal").value = calY.toFixed(0);
}

function calibraCH4() {
   var xMse1 = parseFloat(document.getElementById("xCH41").value);
   var yMse1 = parseFloat(document.getElementById("yCH41").value);
   var xMse2 = parseFloat(document.getElementById("xCH42").value);
   var yMse2 = parseFloat(document.getElementById("yCH42").value);
   var xFig1 = 0;
   var yFig1 = 0;
   var xFig2 = 250;
   var yFig2 = 250;
   BxCH4 = (xFig2-xFig1)/(xMse2-xMse1)
   AxCH4 = xFig1 - BxCH4 * xMse1;
   ByCH4 = (yFig2-yFig1)/(yMse1-yMse2);
   AyCH4 = yFig1 - ByCH4 * yMse2;
}

</script>
```

Resultado:



Canto superior esquerdo:
x: 0 *y*: 0

Canto inferior direito:
x: 0 *y*: 0

CALIBRAR

Posição do mouse descalibrada:
x: 0 *y*: 0

Posição do mouse calibrada:
x: 0 *y*: 0

Na primeira parte da listagem está o conteúdo HTML que define as caixas de texto utilizadas para introduzir e apresentar informações; na segunda parte está o script de calibração propriamente dito. O usuário deve posicionar o mouse nos cantos superior esquerdo e inferior direito da figura, obter as coordenadas com relação ao documento, digitá-las nos quadros correspondentes e pressionar o botão `CALIBRAR`. Feito isto, serão apresentadas tanto as coordenadas com relação ao canto superior esquerdo da parte visível do documento (a janela "cliente") quanto as coordenadas com relação ao canto inferior esquerdo da figura, em pixels.

Sua vez.. (9-14)

Modifique o exemplo anterior de modo que a calibração seja feita relativamente ao centro da molécula.

---

keyCode

É possível obter o código de uma tecla pressionada e realizar alguma ação em função do seu valor. O exemplo a seguir mostra o uso de `onkeydown` no marcador `<body>` do HTML para chamar a função `passoTecla(event)` toda vez que uma tecla estiver abaixada. Quando as telas "seta para a esquerda" (37) e "seta para a direita" (39) forem pressionadas, uma seta (← ou →) vai andar para a esquerda ou para a direita dentro de uma divisão; se qualquer outra tecla for pressionada, um × é mostrado parado na última posição.

Listagem:

```
<html>

<body onkeydown="mostraTecla(event)">

<script>
var passoTecla = 0;

function mostraTecla(evnt)
{
    if (window.event) tecla = evnt.keyCode;   // Internet Explorer
    else tecla = evnt.which;                  // Firefox, Chrome
```

```
    var txt = "&times;";
    if (tecla==37) {
       passoTecla += -10;
       txt = "&larr;";
    }
    if (tecla==39) {
       passoTecla += 10;
       txt = "&rarr;";
    }
    if (passoTecla>200) passoTecla = -200;
    if (passoTecla<-200) passoTecla = 200;
    var str = "<div style='position:relative; left:" + passoTecla + ";'>";
    str += txt;
    str += "</div>";
    document.getElementById("teclas").innerHTML = str;
}
</script>

<p>
<div id="teclas" style="border: 1px solid black; margin-left:60px;
  width: 420px; height: 40px; text-align:center; padding-top:10px;">
</div>
</p>
</div>

</body>
</html>
```

Resultado:

Neste exemplo foi novamente utilizado o atributo `innerHTML` para inserir conteúdo em uma divisão do documento. Neste caso, foi inserida uma nova divisão com os atributos de estilo `position` e `left`, sendo o valor desse último controlado ao pressionar-se a tecla de seta para a direita ou para a esquerda.

Note que é feita uma verificação da coordenada da divisão, fazendo com que a seta "circule" pela caixa caso o valor do atributo de estilo `left`, armazenado na variável `passoTecla`, atinja valores tais que a divisão seria posicionada além dos limites da caixa.

Sua vez... (9-15)

Modifique o exemplo anterior de modo a ampliar a altura da caixa e fazer com que o elemento flutuante responda também às teclas "seta para cima" e seta para baixo" (cujos códigos podem ser obtidos facilmente na internet).

Exercícios

1. Modifique o script da calculadora de modo a incluir botões para as seguintes funções: seno, cosseno, tangente, exponencial, logaritmo natural, logaritmo na base 10, raiz quadrada, raiz cúbica e função potência.

2. Refaça a sua calculadora utilizando somente as funções que recebem apenas um argumento. Ao invés de botões, utilize um grupo de controles `radio` para a escolha da função desejada. Utilize o `onChange` em cada item da sua lista de funções para que, quando a escolha for feita, o valor da função seja apresentado. Seu script dever gerar algo parecido com:

3. Faça um script que contenha quatro campos de texto e um botão. O usuário deve escrever ou colar em um dos campos de texto uma sequência de números inteiros arbitrários e, quando o usuário pressionar o botão, seu script deve escrever nos outros três campos o maior número, o menor número e a média dos números. *Dica 1:* Estude o método `split()` e utilize-o para transformar o texto digitado em uma matriz unidimensional de números inteiros. *Dica 2:* O script a seguir mostra uma possível estratégia para obter o maior número em uma matriz unidimensional.

```
<script>
```

```
var mat = new Array(5,3,2,7,4); // matriz 1D com os números
var maxNro = -Infinity;         // o menor número possível
for (var i=0; i<mat.length; i++) {
    // se o nro for maior que o max atual é o novo max
    if (maxNro < mat[i]) maxNro = mat[i];
}
window.alert(maxNro);
</script>
```

4. Elabore uma prova com cinco questões de múltipla escolha, utilizando elementos tipo `radio`. O usuário deve responder às questões e, ao clicar em um botão "OK", receber uma nota entre 0 e 10 proporcional ao número de acertos.

5. Elabore uma questão do tipo "marque verdadeiro ou falso" com 5 alternativas, utilizando elementos do tipo `checkbox`. O script deve imprimir a nota final obtida a partir das escolhas do usuário de modo que, para cada acerto (verdadeiro ou falso), o usuário ganhe 2 pontos e, para cada erro, perca dois pontos, sendo que a nota final não pode ser negativa.

6. A *impedância acústica* Z de um material é o produto da densidade ρ, medida em kg/m^3, com a velocidade do som v no meio, medida em m/s: $Z = \rho \times v$ (em kg/m^2/s). O *coeficiente de reflexão* de uma interface, para uma incidência normal (90 graus) do ultrassom, é:

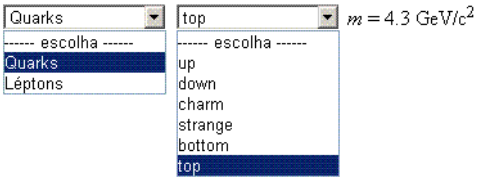
   R = \left(\frac{Z_1 - Z_2}{Z_1 + Z_2} \right)^2

   onde Z_1 e Z_2 são as impedâncias acústicas dos meios que definem a interface. Faça um script que apresente uma tabela em que o usuário possa escolher os tecidos que definem uma interface e calcular a fração de ultrassom refletida nela:

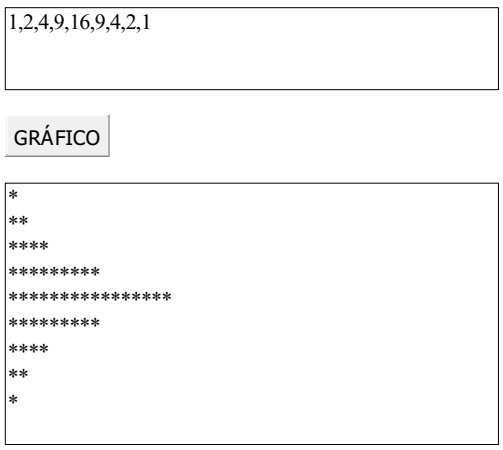
Material 1	Material 2	Impedância ($kg/m^2/s \times 10^6$)
◉ Ar	○ Ar	0,0004
○ Gordura	◉ Gordura	1,38
○ Água	○ Água	1,54
○ Músculos	○ Músculos	1,70
○ Ossos	○ Ossos	7,8

Calcula reflexão $R =$ [ ] %

7. Modifique o exemplo do `select/option` com elétrons e prótons de modo que as opções passem a ser "quarks" e "léptons". Ao escolher um tipo, deve aparecer uma nova caixa de escolhas com as opções "up", "down", "charm", "strange", "bottom" e "top" para os quarks ou com as opções "elétron", "múon", "tau", "eletron-neutrino", "muon-neutrino" e "tau-neutrino" para os léptons. Quando o usuário escolher um item desta segunda caixa, deve aparecer em um terceiro campo a massa das partículas (retorne ao capítulo sobre tabelas para obter os valores das massas dos quarks e dos léptons). Seu script deve gerar algo parecido com:



8. Uma maneira de fazermos um gráfico enquanto não aprendemos como lidar com elementos de desenho em HTML/JavaScript é fazê-lo com caracteres. Faça um script que contenha uma área de texto para entrada de dados, um botão e uma área de texto para saída de dados. Na área de dados, o usuário digita os valores da coordenada *y* do gráfico, separados por vírgula. Ao clicar no botão, o script deve imprimir na primeira linha da área de saída de dados um número de asteriscos igual ao valor do primeiro número digitado, na segunda linha um número de asteriscos igual ao valor do segundo número digitado e assim por diante. Seu script deve ter uma aparência aproximadamente igual a esta:



9. Modifique o exemplo sobre a calibração da posição do mouse sobre a figura de modo que os pontos de referência (canto superior esquerdo e canto inferior direito) não precisem ser digitados, sendo obtidos com cliques do mouse.

10 Tempo

Neste capítulo você vai aprender a acessar e controlar variáveis temporais, essenciais para a modelagem de quaisquer sistemas dinâmicos, da física à biologia à epidemiologia à economia. Além de aprender como formatar convenientemente e fazer cálculos com o tempo fornecido pelo sistema, vai descobrir como utilizar temporizadores para fazer com que ações sejam executadas em instantes pré-definidos ou a intervalos regulares.

`Date()`

Para obter a data e hora do sistema, emprega-se o objeto `Date`:

Listagem:

```
<script>
var agora = new Date()
document.write(agora);
</script>
```

Resultado:

Wed Sep 21 2016 19:48:23 GMT-0300 (Hora oficial do Brasil)

Note que a variável `agora` contém a data e hora do momento em que a instrução foi executada e **não** é constantemente atualizada.

O objeto `Date` tem vários métodos e aqui serão apresentados apenas os mais comuns. Alguns métodos retornam os diversos componentes de uma data separadamente:

Listagem:

```
<script>
var agora = new Date();
var dia  = agora.getDate();
var mes  = agora.getMonth();
var ano  = agora.getFullYear();
var hora = agora.getHours();
var min  = agora.getMinutes();
var sec  = agora.getSeconds();
var milisec = agora.getMilliseconds();
str  = "Esta frase foi impressa em às ";
str += hora + " h " + min  + " m " + sec + " s ";
str += milisec + " ms de ";
str += dia + "/" + (mes+1) + "/" + ano;
document.write(str);
</script>
```

Resultado:

Esta frase foi impressa às 19 h 48 m 23 s 528 ms de 21/9/2016

Note o uso do método `getFullYear()` ao invés de `getYear()`, que também existe. O problema deste último método é que em alguns sistemas operacionais ele devolve o ano com dois dígitos, contando a partir de 1900, o que faz com que "2009" fique "109". Note também que foi somada uma unidade ao mês no momento da impressão pois o método `getMonth()` retorna valores de 0 (janeiro) a 11 (dezembro).

O objeto `Date` também pode ser instanciado com uma data específica. Por exemplo, uma variável que contenha a representação das 12 horas e 10 minutos do dia 28 de outubro de 2015 pode ser criada utilizando a instrução `var umDiaQualquer = new Date(2015,9,28,12,10)`. Faça uma busca na internet (usando algo como "Date() JavaScript") para ver todos os tipos de parâmetros que podem ser passados para o construtor do objeto.

Sua vez.. (10-1)

Utilize os métodos do objeto `Date` para obter a data do sistema e determinar o tempo decorrido, em minutos, desde o início do mês até o momento em que seu programa é executado. Por exemplo, se seu programa fosse executado às 12h10m do dia 28 de outubro de 2015, deveria mostrar que passaram-se $dias \times 24 \times 60 + horas \times 60 + minutos = 41050$ minutos desde as 00h00m do primeiro dia do mês.

A informação de tempo no sistema é de fato guardada como o número de milisegundos decorridos desde as 0 horas do dia 1o. de janeiro de 1970 e o objeto é convertido nos vários formatos pelos diferentes métodos. O script a seguir mostra como obter este número com o método `getTime()` do objeto `Date` no momento em que a página é carregada.

Listagem:

```
<script>
var agora = new Date();
document.write(agora.getTime() + " milissegundos.");
</script>
```

Resultado:

1474498103529 milissegundos.

Utilizando este método é possível subtrair os valores de objetos `Date` para obter o tempo decorrido entre dois eventos. O exemplo a seguir ilustra como obter o tempo de execução de um trecho de código que soma 1 milhão de números aleatórios. O script utiliza duas instâncias do objeto `Date`, uma obtida no início dos cálculos e outra no final.

Listagem:

```
<script>
var soma = 0;
var N = 1000000;
var inicio = new Date();
for (var i=0;i<N;i++)
   soma = soma + Math.random();
document.write("Soma de " + N + " nros. aleatórios: ");
document.write(soma + "<br>");
var final  = new Date();
document.write("Tempo de execução: ");
document.write((final.getTime()-inicio.getTime()) + " ms");
</script>
```

Resultado:

Soma de 1000000 nros. aleatórios: 500280.8291543927
Tempo de execução: 45 ms

Note que o resultado da soma deve ser sempre próximo a 500 mil, como era de se esperar ao evocar 1 milhão de vezes um gerador de números aleatórios que produz números entre 0 e 1.

Sua vez.. (10-2)

Modifique o exemplo anterior para tentar estimar o tempo de execução de uma chamada à função (a) seno, (b) logaritmo e (c) raiz quadrada (que podem variar bastante de computador para computador).

Uma potencial aplicação do objeto `Date` e seus métodos é o cálculo da atividade de uma fonte radioativa em um determinado instante sabendo sua atividade em outro instante anterior. A lei do decaimento radioativo faz com que a atividade $A(t)$ de uma amostra radioativa em um determinado instante de tempo t seja dada por:

$$A(t) = A_0\, e^{-\lambda t}$$

onde A_0 é a atividade medida em um instante conhecido e λ é a constante de decaimento, relacionada à meia-vida $T_{1/2} = \ln(2)/\lambda$ do radioisótopo. O script a seguir assume que uma amostra de cobalto-60 ($T_{1/2}$ = 5,3 anos, $\lambda = 0.13$ ano^{-1}) tinha, em 15 de julho de 2005, uma atividade de 10 GBq, ou 10×10^9 decaimentos por segundo, e calcula qual a sua atividade em uma data especificada pelo usuário:

Listagem:

```
<script>
function calculaAtividade() {

   var ano = document.getElementById("ano").value;
   var mes = document.getElementById("mes").value;
   var dia = document.getElementById("dia").value;

   var dataUsr = new Date(ano,mes,dia);
   var dataRef = new Date(2005,7,15);

   var dif = (dataUsr.getTime() - dataRef.getTime());
   dif = dif/1000/60/60/24/365;
   var ativ = 1e10 * Math.exp(-0.13*dif);

   document.getElementById("atividade").value = ativ.toPrecision(3);

}
</script>
<p>
Cobalto-60 (<i>T</i><sub>1/2</sub> = 5,3 anos)<br>
Atividade em 15 de julho de 2005: 10 &times; 10<sup>9</sup> dec/s<br>
Atividade em:
<input type="text" size="1" value="" id="dia">/
<input type="text" size="1" value="" id="mes">/
<input type="text" size="1" value="" id="ano">
<input type="button" value=" OK " onClick="calculaAtividade()">
<input type="text" size="5" value="" id="atividade" disabled> dec/s
</p>
```

Resultado:

Cobalto-60 ($T_{1/2}$ = 5,3 anos)
Atividade em 15 de julho de 2005: 10×10^9 dec/s
Atividade em: [ ] / [ ] / [ ] OK [ ] dec/s

No exemplo, são criados dois objetos `Date` que recebem como parâmetros a data digitada pelo usuário e a data da medida de referência da atividade da fonte. Em seguida, é utilizado o método `getTime()` para obter ambas as datas em milissegundos desde 1o. de janeiro de 1970 e calcular a sua diferença, que na linha seguinte é convertida em anos. Com a diferença em anos, é possível utilizar a equação para a atividade com a constante de decaimento dada também em anos.

Note o atributo `disabled`, utilizado no elemento de texto que mostra o resultado dos cálculos. Este atributo faz com que o usuário não possa modificar o valor do campo.

Sua vez.. (10-3)

Modifique o exemplo anterior de modo que a data e a atividade de calibração também possam ser fornecidas pelo usuário.

`setTimeout`

O método `setTimeout(`*`expressão`*`,`*`ms`*`)` executa a `expressão` passado um certo tempo `ms`, especificado em milissegundos. O método devolve uma referência que pode ser utilizada para cancelar a tarefa com o método `clearTimeout(`*`ref`*`)`. O exemplo a seguir sorteia dois números e aguarda 10 segundos para que o usuário escreva o resultado de sua soma e aperte o botão "OK". Se o resultado correto for digitado antes disso, permanecerá na caixa de texto; caso contrário, será substituído por asteriscos.

Listagem:

```
<script>

function escreveSoma(soma) {
```

```
   document.getElementById("soma").value = "***";
}

function checaSoma() {
   var num = parseInt(document.getElementById("soma").value);
   if (num==soma) clearTimeout(timeout);
}
var timeout = setTimeout("escreveSoma(soma)",10000);

var num1 = parseInt(Math.random()*100);
var num2 = parseInt(Math.random()*100);
var soma = num1 + num2;
document.write(num1  + " + " + num2 + " = ");

</script>

<input type="text" size="5" value="" id="soma">
<input type="button" value=" OK " onClick="checaSoma()">
```

Resultado:

51 + 10 = [ ] OK

Note que a referência retornada pela chamada a `setTimeout` é guardada na variável `timeout`, que é passada como parâmetro para o método `clearTimeout()`, que desativará a chamada à função caso o usuário acerte a resposta.

`setInterval`

O método `setInterval(`*`expressão`*`,`*`ms`*`)` executa repetidamente a *`expressão`* a cada intervalo de tempo *`ms`*, dado em milissegundos (diferentemente do método `setTimeout()` que executa a expressão uma única vez). O exemplo a seguir utiliza o `setInterval()` para implementar um relógio no documento, atualizado uma vez por segundo.

Listagem:

```
<script>
function relogio() {
   var agora = new Date();
   var hora = agora.getHours();
   var min  = agora.getMinutes();
   var sec  = agora.getSeconds();
   var str  = hora + ":" + min + ":" + sec;
   document.getElementById("relogio").innerHTML = str;
}
setInterval("relogio()",1000);
</script>
<div id="relogio"></div>
```

Resultado:

Note o uso da propriedade `innerHTML` para atribuir a informação da hora atual à divisão inicialmente definida como vazia no documento.

Sua vez... (10-4)

Modifique o exemplo anterior de modo que mostre o número de segundos que decorreram desde que o documento foi carregado no navegador, atualizando a informação a cada segundo.

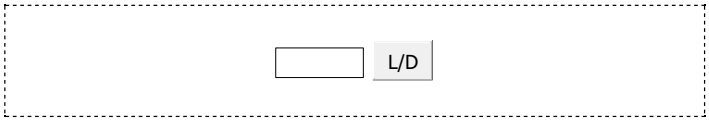
Assim como o método `clearTimeout()` cancela o método `setTimeout()`, o método `clearInterval()` cancela o `setInterval()`. No exemplo a seguir, ambos são utilizados para implementar um cronômetro simples.

Listagem:

```
<script>

var inicio;
var agora;
var crono = null;
```

```
function LigaDesliga() {
   if (crono) {
      clearInterval(crono);
      crono = null;
   }
   else {
      inicio = new Date();
      crono = setInterval("escreveTempo()",10);
   }
}

function escreveTempo() {
   agora = new Date();
   var tempo = (agora.getTime() - inicio.getTime())/1000;
   document.getElementById("tempo").value = tempo.toFixed(2);
}

</script>

<p style='text-align:center'>
<input type='text' value='' id='tempo' size='5'>
<input type='button' value=' L/D ' onClick='LigaDesliga()'>
</p>
```

Resultado:



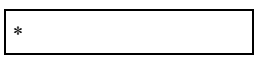
Note o uso da palavra reservada `null` para inicializar a variável global `crono` e para atribuir-lhe valor na instrução `if` no corpo da função `LigaDesliga()`. Na instrução `if`, `null` é equivalente a `false`.

O controle do tempo com o método `setInterval()` é essencial para a produção de animações. O exemplo a seguir mostra um asterisco em um movimento harmônico simples na coordenada horizontal enquanto permanece com a coordenada vertical constante.

Listagem:

```
<script>

function moveCaracter() {
   theta = theta + Math.PI/20;
   var x = parseInt(95 - 90 * Math.cos(theta));
   var y = 10;
   var str = "<div style='position:relative;";
   str += "left:" + x + ";top:" + y + "'>";
   str += "*";
   str += "</div>";
   document.getElementById("caixa").innerHTML = str;
   cnt++;
   if (cnt>100) clearInterval(intervalo);
}
var theta = 0;
var cnt = 0;
var intervalo = setInterval("moveCaracter()",100);

</script>

<p>
<div id="caixa"
     style="margin-left: 2em;border:1px solid black;
          width:200; height:35">
</div>
</p>
```

Resultado:



No exemplo, é definida uma divisão (`<div>...</div>`) identificada como `caixa` no documento. Essa divisão tem 200 pixels de largura, 35 pixels de altura e uma borda sólida preta com 1 pixel de espessura, compondo a "caixa" em que o asterisco oscila. O método `setInterval` faz com que a função `moveCaracter()` seja chamada a cada 100 milissegundos. A função `moveCaracter()` cria uma nova divisão que será colocada dentro da divisão existente no documento pelo uso do `innerHTML`. Nessa divisão, os atributos de estilo `position:relative`,

`left:x` e `top:y` são utilizados para fazer com que o asterisco seja impresso cada vez em uma posição diferente. O movimento cessa após 100 interações.

Sua vez... (10-5)

Modifique o exemplo anterior de modo a produzir uma caixa quadrada de largura e altura iguais a 200 pixels, substitua o asterisco pela letra "o" minúscula e faça com que ela ande em círculos, com o ângulo variando 60 graus por segundo (6 graus a cada 100 milissegundos).

Algumas modificações nesse exemplo permitem incluir controles para acelerar, desacelerar e parar o movimento, como mostrado a seguir.

Listagem:

```
<script>

var passo = 500;

function alteraPasso(opt) {
   switch (opt) {
      case  0:
         clearInterval(intervaloPasso);
         break;
      case -1:
         clearInterval(intervaloPasso);
         if (passo>=200) passo = passo - 100;
         intervaloPasso = setInterval("moveCaracterPasso()",passo);
         break;
      case +1:
         clearInterval(intervaloPasso);
         if (passo<=400) passo = passo + 100;
         intervaloPasso = setInterval("moveCaracterPasso()",passo);
         break;
   }
   document.getElementById("passo").value = "  " + passo;
}

var theta = 0;

function moveCaracterPasso() {
   theta = theta + Math.PI/20;
   var x = parseInt(95 - 90 * Math.cos(theta));
   var y = 10;
   var str = "<div style='position:relative;";
   str += "left:" + x + ";top:" + y + "'>";
   str += "*";
   str += "</div>";
   document.getElementById("caixaPasso").innerHTML = str;
}

</script>

<p>
<div id="caixaPasso"
     style="margin-left:2em;border:1px solid black;
            width:200; height:35;">
</div>

<div style="position: relative; top: -35; left:250">
<input type="button" value=" < " onClick="alteraPasso(-1);">
<input type="text" value="" id="passo" size="3" disabled>
<input type="button" value=" > " onClick="alteraPasso(+1);">
<input type="button" value=" PARA " onClick="alteraPasso(0);">
</div>
</p>

<script>
document.getElementById("passo").value = "  " + passo;
intervaloPasso = setInterval("moveCaracterPasso()",passo);
</script>
```

Resultado:

As alterações merecem alguns comentários adicionais:

- Qualquer que seja o botão pressionado, a função `alteraPasso()` sempre desliga o `intervaloPasso` para depois, se for o caso das opções −1 e +1, religá-lo. Se a variável

que contém sua referência receber um novo valor, a referência antiga será perdida e não será mais possível desligar o temporizador relacionado a ela.

- Para que ficassem ao lado da caixa onde o asterisco se move, os botões foram colocados dentro de uma divisão para a qual foram especificadas coordenadas relativas ao canto superior esquerdo do elemento anterior; por isso o valor negativo do atributo `top`.

- Há um trecho de script antes e outro depois das divisões com a caixa e os controles. O trecho que coloca o valor do passo na caixa de texto entre os controles utiliza o método `getElementById()`, que busca no documento o elemento solicitado. Caso esta instrução apareça antes da divisão, o método não encontra o elemento solicitado e um erro é gerado. Nada impede, no entanto, que todas as instruções sejam colocadas em um único bloco de script depois da definição da caixa.

- Foram colocados limitadores para que os valores do passo ficassem entre 100 e 500, impedindo que atinja valores nulos ou negativos, ou que o movimento fique lento demais.

Sua vez.. (10-6)

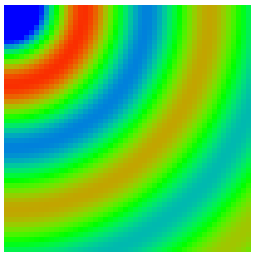
Modifique o exemplo anterior de modo a incluir mais um botão com a legenda `MOVE` e que, ao ser acionado, chame a função `alteraPasso()` com um outro parâmetro (por exemplo, `"m"`). Modifique a função `alteraPasso()` incluindo um novo `case` que responda a esse novo parâmetro, fazendo com que o asterisco passe a mover-se se estiver parado.

O exemplo a seguir modela uma onda bidimensional em movimento.

Listagem:

```
<div id="onda"></div>

<script>
function zCor(z) {
    var z1 = -100;
    var z2 = +100;
    var zm = (z1+z2)/2;
    var r = 0;
    var g = 0;
    var b = 0;
    if (z<=zm) r = parseInt(255*Math.cos((z-z1)/(zm-z1)*Math.PI/2));
    g = parseInt(255*Math.sin((z-z1)/(z2-z1)*Math.PI));
    if (z>=zm) b = parseInt(255*Math.sin((z-zm)/(z2-zm)*Math.PI/2));
    return "rgb(" + r + "," + g + "," + b + ")";
}

function tabela() {
   tempo = tempo + 0.01;
   var str = "";
   str += "<p>";
   str += "<table style='font-size: 1pt; line-height: 1pt'";
   str += "        border='0' cellpadding='1' cellspacing='0'";
   str += "        align='center'>";
   for (x=0;x<=1;x=x+0.02) {
      str += "<tr>";
      for (y=0;y<=1;y=y+0.02) {
         var r = Math.sqrt(x*x+y*y);
         // linha abaixo foi modificada
         var z = 50*Math.cos(4*Math.PI*(r - 2.5*tempo))/Math.sqrt(r);
         if (z>100) z = 100; else if (z<-100) z = -100;
         str += "<td style='background-color:" + zCor(z) + "'>";
         str += "  " + "</td>";
      }
      str += "</tr>";
   }
   str += "</table>";
   str += "</p>";
   document.getElementById("onda").innerHTML = str;
   if (tempo>0.5) clearInterval(tabInterval);
}

var tempo = 0;
var tabInterval = setInterval("tabela()",50);

</script>
```

Resultado:



A primeira providência foi criar uma `<div>` vazia para receber a tabela com a onda renovada a cada passo do temporizador, o que é feito quase no final do script atribuindo-se `str` (que contém a tabela) à propriedade `innerHTML` da divisão, acessada através da chamada a `document.getElementaById()`. O trecho que constrói a tabela (e que contém a física da onda) deve ser colocado dentro de uma função que pode ser passada como parâmetro para o método `setInterval()`.

> Usar uma tabela é uma maneira exótica e ineficiente de construir a onda e lhe dar movimento. No capítulo seguinte, sobre desenhos e animações, veremos uma maneira muito mais razoável e eficiente de fazer isso.

Exercícios

1. Adapte o script do decaimento radioativo para calcular a diferença aproximada (sem levar em conta anos bissextos) em dias entre a sua data de nascimento, digitada nas caixas de texto, e o dia de hoje.
2. Modifique o script do asterisco oscilante para incluir um segundo asterisco fazendo um movimento harmônico simples na direção vertical com controle de velocidade independente.
3. Faça um script que mostre um planeta girando em torno do Sol. Simule o planeta com uma letra "o" e o planeta com um ".".
4. Encontre figuras do Sol, da Terra e da Lua e modifique o script anterior para simular o movimento da Terra em torno do Sol e da Lua em torno da Terra.
5. Faça um script que simule um caça-níqueis com três elementos e cada elemento com 3 opções (três letras ou três figuras, por exemplo), tal que as opções sejam trocadas a cada 100 milissegundos e que o primeiro elemento pare 3 segundos após o acionamento, o segundo após 4 segundos e o terceiro após 5 segundos.
6. O lançamento oblíquo é um movimento bidimensional caracterizado por uma posição de lançamento (x_o, y_o), um ângulo de lançamento θ com a horizontal e uma velocidade de lançamento de módulo v_o. O movimento na direção horizontal (direção x) é retilíneo uniforme (MRU) e o movimento na direção vertical (direção y) é uniformemente variado (MRUV) devido à força da gravidade. A equação de movimento para o lançamento oblíquo pode ser escrita como:

   $$\mathbf{r}(t) = x(t)\,\mathbf{i} + y(t)\,\mathbf{j}$$

   onde

   $$x(t) = x_o + v_{xo}\,t$$
   $$y(t) = y_o + v_{yo}\,t + \tfrac{1}{2}\,a\,t^2$$

   onde x_o e y_o são as coordenadas do ponto de lançamento, $v_{xo} = v_o \cos\theta$, $v_{yo} = v_o \sin\theta$ as componentes da velocidade inicial e a a aceleração, no caso a da gravidade.

   A velocidade em qualquer instante é dada por:

   $$\mathbf{v}(t) = v_x(t)\,\mathbf{i} + v_y(t)\,\mathbf{j}$$

   onde

   $$v_x(t) = v_{xo}$$
   $$v_y(t) = v_{yo} + a\,t$$

   Faça um script que simule com uma letra ou uma figura um lançamento oblíquo em que o usuário escolhe o módulo da velocidade inicial e o ângulo de lançamento.

11 Canvas

Neste capítulo você vai ver uma das maneiras de utilizar o JavaScript para fazer desenhos no seu documento. Você vai ver que com uns poucos métodos para desenhar linhas, arcos e curvas é possível fazer praticamente tudo que você imagina. Vai entender como utilizar transformações de coordenadas para modelar com facilidade o sistema físico de interesse. Finalmente, vai ver como produzir animações, integrando temporizadores aos seus desenhos.

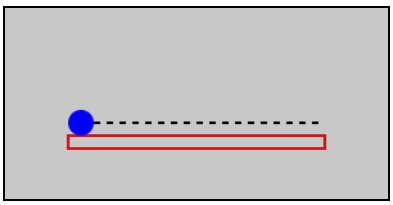
O canvas é o elemento onde podemos fazer desenhos. É colocado em um documento HTML com os marcadores `<canvas> ... </canvas>` que, além dos atributos gerais dos elementos HTML (`id`, `style`), tem dois atributos específicos: `width` e `height`, que definem, respectivamente, a largura e a altura do canvas, em pixels.

Uma vez no documento, a sua referência pode ser obtida com a instrução `document.getElementById()`. A partir dessa referência têm-se acesso a suas propriedades, em particular à sua largura (`.width`), altura (`.height`) e o seu contexto gráfico (`.getContext('2d')`). Vários métodos do contexto gráfico permitem o desenho de linhas, retângulos, círculos etc., com diferentes propriedades, tais como cor, espessura de linha, preenchimento, opacidade etc. O exemplo a seguir ilustra o esquema básico de utilização de um canvas para desenhar os principais elementos básicos.

Listagem:

```
<div style="text-align:center">
<canvas id="cnv" width="300" height="150"
        style="border: 1px solid black;
               background-color:rgb(200,200,200)">
</canvas>
</div>

<script>
var cnv = document.getElementById("cnv");
var ctx = cnv.getContext("2d");

// retângulo vermelho delineado
ctx.beginPath();
ctx.rect(50,100,200,10);
ctx.strokeStyle = "red";
ctx.lineWidth = 2;
ctx.stroke();

// linha preta horizontal
ctx.beginPath();
for (var i=0;i<19;i++) {
   ctx.moveTo(60 + 10*i,90);
   ctx.lineTo((60 + 10*i + 5),90);
}
ctx.lineWidth = 2;
ctx.strokeStyle = "black";
ctx.stroke();

// bola azul preenchida
ctx.beginPath();
ctx.arc(60,90,10,0,2*Math.PI,true);
ctx.fillStyle = "blue";
ctx.fill();

</script>
```

Resultado:



Em primeiro lugar, observe como é definido o canvas no documento HTML, com um `id` e os atributos de largura (`width`) e altura (`height`), além de especificações de estilo (no caso, relacionadas à borda e a um fundo cinza claro). O `canvas` está definido dentro de uma `div` com uma declaração de estilo que centraliza o seu conteúdo.

O script inicia buscando o elemento `canvas` desejado e a seguir atribui à variável `ctx` o seu contexto gráfico. A seguir, o script desenha três estruturas: o retângulo vermelho delineado, a linha preta horizontal tracejada e o círculo azul preenchido.

O retângulo vermelho delineado é desenhado com o método `rect(x,y,w,h)`, que recebe como parâmetros as coordenadas `x,y` do canto superior esquerdo do retângulo e a sua largura `w` e altura `h`. Para que seja desenhado sem preenchimento, é preciso executar o método `stroke()` ("pincelada").

A linha tracejada é desenhada como uma sucessão de pequenas linhas separadas por espaços em branco. O método `moveTo(x,y)` "levanta" o "pincel" e leva-o até o ponto `x,y` sem deixar marcas; o método `lineTo(x,y)` traça uma linha reta do ponto em que o pincel estiver até as coordenadas `x,y` que recebeu. No caso, as coordenadas `x` do início de cada traço são 60, 70, 80 etc. (`60 + 10*i`), as coordenadas do fim dos traços são 65, 75, 85 etc. (`60 + 10*i + 5`) e a coordenada `y`, fixa, é 90. A cor da pincelada é controlada pela propriedade `strokeStyle` (estilo da pincelada) e sua espessura pela propriedade `lineWidth` (largura da linha). No exemplo, o laço `for` desenha uma linha com comprimento de 5 pixels a cada 10 pixels.

O círculo azul é desenhado com o método `arc(x,y,r,ain,afn,ahor)`, onde `x,y` são as coordenadas do centro do arco, `r` o seu raio, `ain` e `afn` os ângulos inicial e final em radianos. O último parâmetro, `ahor`, é uma expressão booleana (`true` ou `false`, 0 ou 1) que, se verdadeira, indica que o arco deve ser desenhando no sentido anti-horário e, se falsa, no sentido horário. A cor de preenchimento é atribuída à propriedade `fillStyle` e o desenho, com preenchimento, é feito com uma chamada ao método `fill()`.

Sua vez.. (11-1)

1. Modifique o exemplo anterior de modo que (a) faça o retângulo ser desenhado preenchido e o círculo somente com contorno e (b) desenhe um arco de um quarto de círculo (0 a π/2), especificando `true` ou `false` para o último parâmetro.

2. Cada desenho começa com uma chamada ao método `beginPath()` do contexto gráfico, que indica o início de um conjunto de instruções relacionadas a uma determinada trajetória de desenho no documento. Comente algumas ou todas as linhas com `beginPath()` e veja o que acontece. Como você interpreta os resultados? O que conseguiu descobrir na internet a respeito disso?

Você encontrará facilmente na Internet tutoriais bastante abrangentes sobre o uso do canvas que podem lhe ensinar a fazer sofisticadas obras de arte animadas. Aqui vamos nos ater ao básico para visualizar informações e modelos elementares de sistemas físicos.

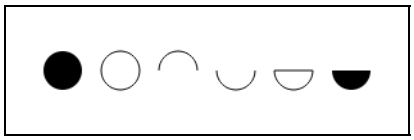
O exemplo a seguir foi pensado para deixar mais claras as diferenças entre o `stroke()` e o `fill()`, entre o uso de `true` ou `false` para a direção do arco e entre o uso ou não do método `closePath()` para fechar ou não o desenho (o método `closePath()` também pode ser utilizado para fechar um polígono construído a partir de linhas).

Listagem:

```
<p style="text-align:center">
<canvas id="cnv" width="315" height="100"
        style="border-width: 1px; border-style:solid">
</canvas>
</p>

<script>

var cnv = document.getElementById("cnv");
var ctx = cnv.getContext("2d");
```

```
// cores de linha e de preenchimento
ctx.strokeStyle = "black";
ctx.fillStyle = "black";
// círculo preenchido
ctx.beginPath();
ctx.arc(45,50,15,0,2*Math.PI,true);
ctx.fill();
// círculo delineado
ctx.beginPath();
ctx.arc(90,50,15,0,2*Math.PI,true);
ctx.stroke();
// semi-círculo para cima
ctx.beginPath();
ctx.arc(135,50,15,0,Math.PI,true);
ctx.stroke();
// semi-círculo para baixo
ctx.beginPath();
ctx.arc(180,50,15,0,Math.PI,false);
ctx.stroke();
// semi-círculo fechado
ctx.beginPath();
ctx.arc(225,50,15,0,Math.PI,false);
ctx.closePath();
ctx.stroke();
// semi-círculo preenchido
ctx.beginPath();
ctx.arc(270,50,15,0,Math.PI,false);
ctx.fill();

</script>
```

Resultado:



Sua vez... (11-2)

Modifique o exemplo anterior de modo a atribuir uma cor diferente a cada figura desenhada. Depois de ver o resultado comente alguns ou todos os `beginPath()` e analise os resultados.

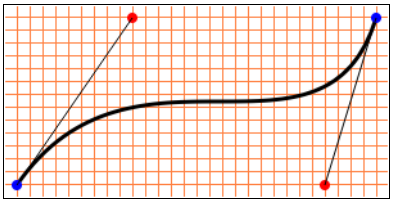
É possível fazer desenhos sobre uma imagem previamente carregada (uma foto, por exemplo). O exemplo a seguir ilustra como fazer isto, carregando uma imagem com o fundo quadriculado e desenhando algumas curvas de Bezier sobre ela.

Listagem:

```
<div style="text-align:center">
<canvas id="cnv" width="300" height="150"
        style="border:1px solid black">
</canvas>
</div>

<script>

var cnv = document.getElementById("cnv");
var ctx = cnv.getContext("2d");

var img = new Image();
img.src = "js-canvas-quadriculado.png";
img.onload = function() { desenhaTudo(); }

function desenhaTudo() {

    // imagem de fundo
    ctx.drawImage(img,0,0);

    var x1  = 10, y1  = 140, x2 = 290, y2 = 10;
    var xct1 = 100, yct1 = 10, xct2 = 250, yct2 = 140;

    // curva de Bézier
    ctx.beginPath();
    ctx.moveTo(x1,y1);
    ctx.bezierCurveTo(xct1, yct1, xct2, yct2, x2, y2)
    ctx.lineWidth = 3;
    ctx.strokeStyle = "black";
```

```
    ctx.stroke();

    // pto inicial
    ctx.beginPath();
    ctx.arc(x1,y1,4,0,2*Math.PI,true);
    ctx.fillStyle = "blue";
    ctx.fill();

    // 1o. pto de controle
    ctx.beginPath();
    ctx.arc(xct1,yct1,4,0,2*Math.PI,true);
    ctx.fillStyle = "red";
    ctx.fill();

    // linha ligando pto inicial a 1o. pto de controle
    ctx.beginPath();
    ctx.moveTo(x1,y1);
    ctx.lineTo(xct1,yct1);
    ctx.lineWidth = 1;
    ctx.strokeStyle = "black";
    ctx.stroke();

    // pto final
    ctx.beginPath();
    ctx.arc(x2,y2,4,0,2*Math.PI,true);
    ctx.fillStyle = "blue";
    ctx.fill();

    // 2o. pto de controle
    ctx.beginPath();
    ctx.arc(xct2,yct2,4,0,2*Math.PI,true);
    ctx.fillStyle = "red";
    ctx.fill();

    // linha ligando pto final a 2o. pto de controle
    ctx.beginPath();
    ctx.moveTo(x2,y2);
    ctx.lineTo(xct2,yct2);
    ctx.lineWidth = 1;
    ctx.strokeStyle = "black";
    ctx.stroke();

}
</script>
```

Resultado:



Depois da definição do canvas fora do script e da captura do contexto gráfico no início do script, é criado um objeto `Image()`, que tem várias propriedades e métodos. Em seguida, a propriedade `src` recebe o nome do arquivo com a imagem a ser carregada, que, como especificado, tem que estar na mesma pasta que o arquivo HTML.

A instrução seguinte, `img.onload = function() { desenhaTudo() }`, especifica que a função `desenhaTudo()` somente deve ser executada *depois* que o arquivo de imagem for carregado. Isso é necessário porque o tempo de acesso à imagem, armazenada em geral no disco rígido ou em algum outro local da internet, é muito maior do que o tempo de execução das instruções, o que pode levar a alguns resultados bizarros.

Sua vez.. (11-3)

Modifique o exemplo anterior de modo a substituir a instrução `img.onload = desenhaTudo()` por `desenhaTudo()`, isto é, uma chamada direta à função. Veja o que acontece quando você recarrega a página "do zero" (clicando sobre o arquivo na pasta ou utilizando `CTRL+F5`) ou simplesmente atualizando a página (`F5`).

Os resultados podem ser diferentes em diferentes navegadores (Chrome, Firefox, IE, Safari etc.) e diferentes sistemas operacionais (Androide, Linux, Windows etc.).

A instrução `ctx.drawImage(img,0,0)` coloca a imagem carregada no canvas, fazendo com que seu canto superior esquerdo seja colocado no canto superior esquerdo do canvas, que tem coordenadas (0,0). O método `drawImage()` pode receber dois parâmetros adicionais, que aplicam um fator de escala às direções horizontal e vertical da imagem.

Sua vez.. (11-4)

Modifique o exemplo anterior de modo a substituir a instrução `ctx.drawImage(img,0,0)` por (a) `ctx.drawImage(img,0,0,150,75)` e (b) `ctx.drawImage(img,75,37,150,75)` e observe os resultados.

Além de mostrar como carregar uma imagem e desenhar sobre ela, o exemplo mostra um uso do método `bezierCurveTo(xct1,yct1,xct2,yct2,x,y)`, que desenha curva do ponto onde o pincel encontra-se até o ponto `x,y` utilizando dois pontos de controle com coordenadas `xct1,yct1` e `xct2,yct2`. A figura sobre o fundo quadriculado foi construída para ilustrar o significado dos pontos de controle: a curva é desenhada com o método `bezierCurveTo()`, utilizando os pontos de controle indicados pelos círculos vermelhos.

Sua vez.. (11-5)

Modifique o exemplo anterior de modo a explorar o que acontece quando as coordenadas dos pontos de controle são modificadas. O que acontece quando os pontos de controle são os mesmos? Quando os valores de *x* ou *y* são os mesmos?

Transformações de coordenadas

Frequentemente o sistema de coordenadas próprio do canvas (com origem 0,0 no canto superior esquerdo e unidades contadas em pixels) é muito inconveniente para a modelagem de sistemas físicos e outras aplicações. Para contornar esse problema, o canvas dispõe de alguns métodos de transformação de coordenadas: `translate()`, `rotate()`, `scale()`, `transform()` e `setTransform()`, que serão explorados em detalhes a seguir.

`translate(x,y)`

O método `translate(x,y)` move a origem do canvas para as coordenadas `x,y`. Deste modo, para um canvas 300 × 300, a instrução `translate(150,150)` move a origem, anteriormente no canto superior esquerdo, para o centro do canvas, fazendo com que passe a mostrar elementos com coordenadas entre −150 e 150.

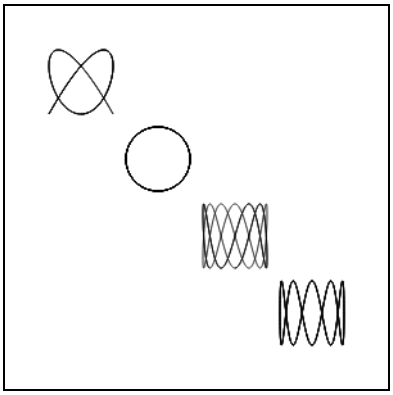
No exemplo a seguir, é definida uma função que gera uma figura de Lissajous (uma combinação de senos e cossenos nas direções *x* e *y*) centrada em uma origem (0,0). No laço `for` do script são desenhadas quatro figuras transladadas de 60 pixels nas direções *x* e *y*, em relação à posição prévia.

Listagem:

```
<p style="text-align:center">
<canvas id="cnv" width="300" height="300"
        style="border:1px solid black">
</canvas>
</p>

<script>

var cnv = document.getElementById("cnv");
var ctx = cnv.getContext("2d");

for (var i=0;i<4;i++) {
   ctx.translate(60,60);
   var Tx = 100*parseInt(1+Math.random()*7);
   var Ty = 100*parseInt(1+Math.random()*9);
   lissajous(ctx,Tx,Ty);
}

function lissajous(lctx,Tx,Ty) {
   var x = 0;
   var y = 25;
   lctx.beginPath();
   lctx.lineWidth = 1;
   lctx.strokeStyle = "black";
   lctx.moveTo(x,y);
   for (var t=0;t<2000;t++) {
      x = 25 * Math.sin(2*Math.PI*(t/Tx));
      y = 25 * Math.cos(2*Math.PI*(t/Ty));
      lctx.lineTo(x,y);
   }
```

```
   lctx.stroke();
}

</script>
```

Resultado:



Quando o canvas é inicializado, o pincel está na posição (0,0), que é o canto superior esquerdo do canvas. Na primeira interação do laço for, a origem do canvas é movida para a posição (60,60) e a figura de Lissajous é desenhada neste referencial. Na segunda interação, um deslocamento adicional de (60,60) é dado à origem e uma nova figura desenhada em torno desse ponto. O resultado final é que as figuras são desenhadas em torno das coordenadas (60,60), (120,120), (180,180) e (240,240).

Note que, neste exemplo, uma variável global `ctx` é utilizada para receber a referência do contexto gráfico obtido do canvas. Esta variável é passada como parâmetro para a função `lissajous()` que a referencia internamente como a variável `lctx`. Esta estratégia pode ser interessante para lidar com documentos com vários canvas, em que a confusão de nomes pode prejudicar a visualização.

Sua vez.. (11-6)

(a) Modifique o exemplo anterior de modo utilize dois laços `for` para desenhar 16 figuras de Lissajous sobre o canvas. (b) Remova a aleatoriedade dos períodos de modo que o seu "mapa" mostre como ficam as figuras com períodos indexados pela posição (isto é, 1,1 a 1,4 para a primeira linha, 2,1 a 2,4 para a segunda, 3,1 a 3,4 para a terceira e 4,1 a 4,4 para a quarta).

`rotate(ang)`

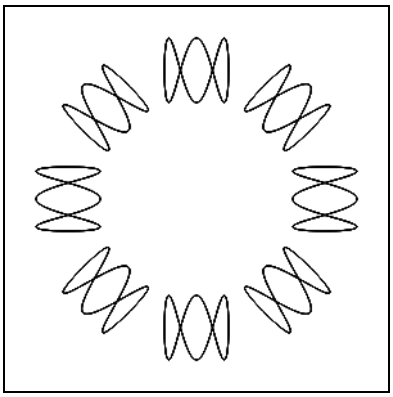
O sistema de coordenadas do canvas pode ser rodado de um ângulo `ang` (em radianos) utilizando-se o método `rotate(`*`ang`*`)`. O exemplo a seguir primeiro translada a origem do canvas para o centro e depois chama a função `lissajous()` definida no exemplo anterior dentro de um laço `for`, onde uma translação e uma rotação são utilizadas para posicionar cada nova figura (para que esse exemplo funcione, a função `lissajous()` deve ser incluída no script).

Listagem:

```
<p style="text-align:center">
<canvas id="cnv" width="300" height="300"
        style="border:1px solid black">
</canvas>
</p>

<script>

var ctx = document.getElementById("cnv").getContext("2d");

ctx.translate(150,150);
for (var i=0;i<8;i++) {
   ctx.save();
   ctx.translate(100,0);
   lissajous(ctx,100,300);
   ctx.restore();
   ctx.rotate(Math.PI/4);
}
```

```
</script>
```

Resultado:



O exemplo apresenta dois métodos extremamente úteis: `save()` (salva) e `restore()` (restaura). Como seus nomes indicam, esses métodos salvam e restauram a configuração do canvas no momento em que são chamados. Tudo o que for feito após o um `save()` (transformações de coordenadas, mudanças de espessuras de linhas, especificações de cores etc.) será "esquecido" após um `restore()`.

Quando o laço inicia, o pincel está em (0,0), no centro do canvas, e a configuração é salva. A seguir, é feita uma translação de 100 pixels para a direita, desenhada a figura e restaurada a configuração anterior, isto é, o pincel é recolocado em (0,0). A última instrução do laço gira o sistema de coordenadas em 45 graus, o que faz com que, na segunda interação, os 100 pixels da translação não sejam mais para a direita, mas a sudeste, e assim por diante.

> Neste exemplo em particular, seria mais econômico utilizar um `translate(-100,0)` após desenhar a figura, ao invés de chamar os métodos `save()` e `restore()`. Entretanto, a estratégia utilizada é mais genérica, o que pode ser mais conveniente quando uma série de tranformações são feitas e têm que ser revertidas.

`scale(sx,sy)`

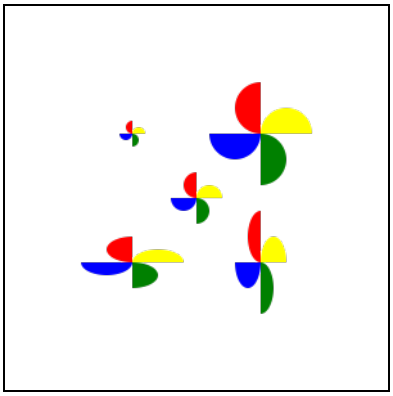
O exemplo a seguir ilustra o uso de mais uma transformação, a transformação de escala, bem como outro exemplo de uso dos métodos `save()` e `restore()`. No exemplo, a função `desenhaCatavento()` é chamada em cinco configurações diferentes do canvas. Na primeira vez, é desenhado o catavento no centro do canvas, em escala 1:1; na segunda, é desenhado o catavento no canto superior esquerdo, com metade do tamanho; na terceira, no canto superior direito, com o dobro do tamanho; e, nas últimas duas com cada uma das direções em escala 2:1.

Listagem:

```
<p style="text-align:center">
<canvas id="cnv" width="300" height="300"
        style="border:1px solid black">
</canvas>
</p>

<script>

var cnv = document.getElementById("cnv");
var ctx = cnv.getContext("2d");

ctx.save();
ctx.scale(1,1);
ctx.translate(150,150);
desenhaCatavento(ctx);
ctx.restore();

ctx.save();
ctx.scale(0.5,0.5);
ctx.translate(200,200);
desenhaCatavento(ctx);
ctx.restore();

ctx.save();
ctx.scale(2,2);
ctx.translate(100,50);
desenhaCatavento(ctx);
ctx.restore();
```

```
ctx.save();
ctx.scale(2,1);
ctx.translate(50,200);
desenhaCatavento(ctx);
ctx.restore();

ctx.save();
ctx.scale(1,2);
ctx.translate(200,100);
desenhaCatavento(ctx);
ctx.restore();

function desenhaCatavento(lctx) {
   lctx.beginPath();
   lctx.arc(+10,0,10,0,Math.PI,true);
   lctx.fillStyle = "yellow";
   lctx.fill();
   lctx.beginPath();
   lctx.arc(-10,0,10,0,Math.PI,false);
   lctx.fillStyle = "blue";
   lctx.fill();
   lctx.beginPath();
   lctx.arc(0,+10,10,Math.PI/2,Math.PI*3/2,true);
   lctx.fillStyle = "green";
   lctx.fill();
   lctx.beginPath();
   lctx.arc(0,-10,10,Math.PI/2,Math.PI*3/2,false);
   lctx.fillStyle = "red";
   lctx.fill();
}
</script>
```

Resultado:



Note que os argumentos para o método `translate()` que vêm após a chamada ao método `scale()` devem ser adaptados para a nova escala. Uma maneira de amarrar uma coisa com outra é definir fatores de escala `xfat` e `yfat`, que seriam utilizados tanto na chamada a `scale()` quanto na chamada a `translate()`:

```
ctx.scale(xfat,yfat);
ctx.translate(posx/xfat,posy/yfat);
```

Sua vez... (11-7)

Modifique o exemplo anterior de modo que os desenhos sejam feitos dentro de um único laço `for` com os argumentos da escala e da translação armazenados em objetos `Array()`.

`tranform()` e `setTranform()`

Esses métodos são uma síntese das outras transformações. Quando o canvas é inicializado, é atribuída a ele uma *matriz de transformação* **T** que, multiplicada pelo vetor de coordenadas **X**, leva um novo vetor de coordenadas **X'**. Em linguagem matricial, essa operação seria:

$$\mathbf{X'} = \mathbf{T} \cdot \mathbf{X}$$

$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} m_{11} & m_{21} & dx \\ m_{12} & m_{22} & dy \end{vmatrix} \cdot \begin{vmatrix} x \\ y \end{vmatrix}$$

$$\begin{vmatrix} 1 \end{vmatrix} \quad \begin{vmatrix} 0 & 0 & 1 \end{vmatrix} \quad \begin{vmatrix} 1 \end{vmatrix}$$

Essa equação matricial corresponde às equações:

$$x' = m_{11}\,x + m_{21}\,y + dx$$
$$y' = m_{12}\,x + m_{22}\,y + dy$$
$$1 = 1$$

Se (m_{11}, m_{12}, m_{21}, m_{22}) = (cosθ, sinθ, −sinθ, cosθ), essa operação será uma rotação de um ângulo θ, seguida de uma translação (*dx*, *dy*).

O método `transform(m11, m12, m21, m22, dx, dy)` multiplica a matriz de transformação corrente, que inicialmente é uma matriz identidade, pela matriz dada pelos coeficientes (m11, m12, m21, m22, dx, dy). O método `setTransform(m11, m12, m21, m22, dx, dy)` restaura a matriz de transformação inicial e a multiplica pela matriz composta pelos argumentos passados ao método.

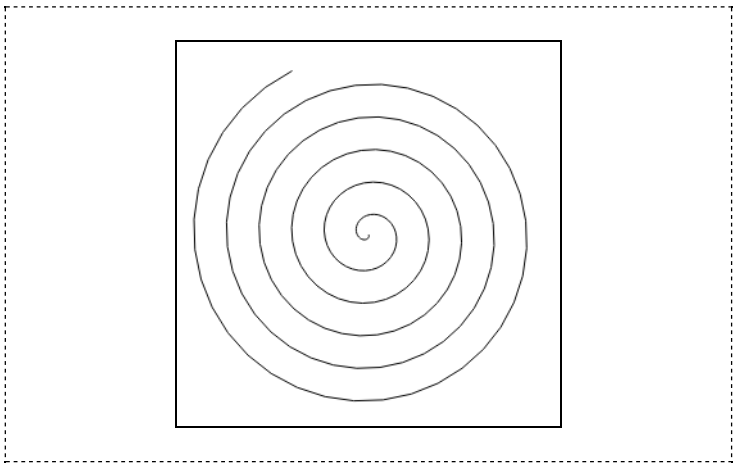
> Note que o método `transform()` multiplica sempre a matriz *corrente* pela matriz fornecida. Se, por exemplo, tratar-se de uma matriz que leva a uma rotação de 1 grau, 10 aplicações sucessivas levarão a uma rotação final de 10 graus, o que não aconteceria com a aplicação do método `setTransform(m11, m12, m21, m22, dx, dy)` que levaria sempre ao mesmo resultado.

O exemplo a seguir mostra como produzir uma espiral desenhando apenas linhas retas e aplicando uma matriz de transformação sucessivas vezes.

Listagem:

```
<p style="text-align:center">
<canvas id="cnv" width="300" height="300"
        style="border:1px solid black">
</canvas>
</p>

<script>

var cnv = document.getElementById("cnv");
var ctx = cnv.getContext("2d");

ctx.scale(0.1,0.1);
ctx.translate(1500,1500);
ctx.lineWidth = 10;

var xprev = 0;
var yprev = 0;
var xcurr, ycurr;

var sin = Math.sin(Math.PI/18);
var cos = Math.cos(Math.PI/18);

for (var i=0;i<200;i++) {
   xcurr = 5*i;
   ycurr = 5*i;
   ctx.beginPath();
   ctx.moveTo(xprev,yprev);
   ctx.transform(cos,sin,-sin,cos,0,0);
   ctx.lineTo(xcurr,ycurr);
   ctx.stroke();
   xprev = xcurr;
   yprev = ycurr;
}
</script>
```

Resultado:



O script inicialmente ajusta a escala, ampliando o canvas de um fator 10. A seguir, muda a origem para o centro e acerta o valor da espessura do traço para 10 unidades. Isto é feito porque agora uma unidade corresponde a um décimo de um pixel e, se a espessura não fosse aumentada, mal se veria a figura.

A seguir, como os valores dos senos e cossenos serão utilizados muitas vezes, o script os deixa pré-calculados nas variáveis `sin` e `cos`, para aumentar a eficiência do programa. Antes do laço, são declaradas e inicializadas as variáveis que armazenarão as coordenadas iniciais (`xprev`,`yprev`) e finais (`xcurr`,`ycurr`) utilizadas no desenho das retas. A cada interação do laço `for`, o pincel é movido para o ponto inicial, a rotação é realizada e a linha desenhada. Desenhada a linha, às coordenadas iniciais do ponto inicial da próxima linha são atribuídas as coordenadas do ponto final da linha recém-desenhada.

Sua vez.. (11-8)

Modifique o exemplo anterior de modo a colorir aleatoriamente as retas desenhadas (veja como fazer isso no exemplo sobre as curvas de Bezier) e verificar a variação de seus tamanhos. Aproveite também para ver como fica a figura quando é comentada a linha que faz a transformação de coordenadas.

Sistema do canvas × sistema do modelo

Talvez o uso mais importante das transformações do canvas seja a sua adaptação para um sistema de coordenadas do sistema que queremos modelar. Vamos supor, por exemplo, um oscilador harmônico amortecido cuja amplitude em função do tempo seja dada por:

$$x(t) = A_0 \cos(\omega t) \exp(-bt)$$

Queremos fazer um gráfico da posição em função do tempo para uma amplitude inicial A_0 = 1 cm, uma frequência angular $\omega = 2\pi$ rad/s (1 rotação por segundo), um coeficiente de amortecimento b = 0.5 s^{-1} para um intervalo de tempo entre 0 e 10 segundos. A direção x de nosso canvas terá que representar valores entre $x_{mín} = 0$ e $x_{máx} = 10$ e a direção y entre $y_{mín} = -1$ e $y_{máx} = 1$, sendo o −1 no extremo inferior do canvas e o 1 no extremo superior. Vamos supor um canvas com largura c_x = 300 pixels e altura c_y = 150 pixels.

Estes dois sistemas estão representados nas figuras a seguir. Note que a origem do eixo y no sistema de coordenadas do canvas está na borda *superior* do canvas e que os valores da coordenada y crescem para *baixo*, diferentemente do usualmente encontrado em matemática e física.

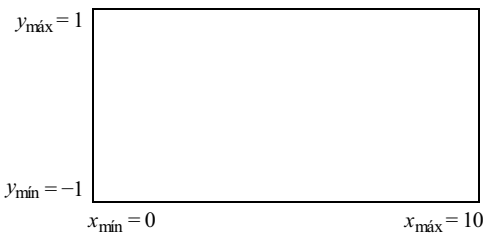


Figura 11.1 Sistema de coordenadas do modelo.

0

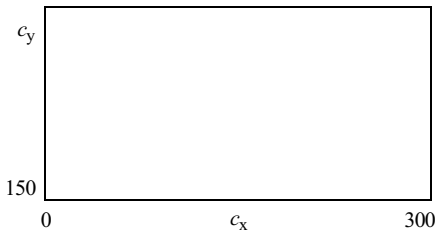


Figura 11.2 Sistema de coordenadas do canvas.

Chamando as coordenadas do modelo de x_m e y_m e as coordenadas do canvas de x_c e y_c, as transformações que levam de um sistema a outro são dadas por:

$$x_c = A_x + B_x \cdot x_m$$
$$y_c = A_y + B_y \cdot y_m$$

Colocando nestas equações os valores conhecidos das coordenadas nos dois sistemas, obtemos para a coordenada x:

$$0 = A_x + B_x \cdot x_{mín}$$
$$c_x = A_x + B_x \cdot x_{máx}$$

e para a coordenada y:

$$0 = A_y + B_y \cdot y_{máx}$$
$$c_y = A_y + B_y \cdot y_{mín}$$

Resolvendo para A e B obtemos, para a coordenada x:

$$B_x = c_x / (x_{máx} - x_{mín})$$
$$A_x = -B_x \cdot x_{mín} = -[c_x / (x_{máx} - x_{mín})] \cdot x_{mín}$$

e para a coordenada y:

$$B_y = c_y / (y_{mín} - y_{máx})$$
$$A_y = -B_y \cdot y_{máx} = -[c_y / (y_{mín} - y_{máx})] \cdot y_{máx}$$

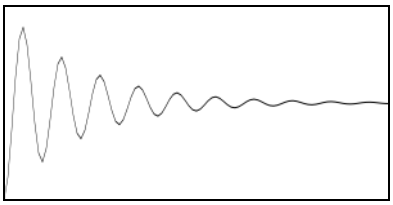
O exemplo a seguir implementa utiliza um conjunto de instruções para estabelecer os limites do canvas e do modelo e a instrução `setTranform()` para definir um sistema de coordenadas apropriado para o sistema físico em questão.

Listagem:

```
<p style="text-align:center">
<canvas id="cnv" width="300" height="150"
        style="border:1px solid black">
</canvas>
</p>

<script>
var cnv = document.getElementById("cnv");
var ctx = cnv.getContext("2d");
var cw  = cnv.width;
var ch  = cnv.height;
var xmin = 0;
var xmax = 10;
var ymin = -1;
var ymax = +1;
var m11 = cw/(xmax-xmin);
var m12 = 0;
var m21 = 0;
var m22 = - ch/(ymin-ymax);
var dx  = - cw*xmin/(xmax-xmin);
var dy  = - ch*ymax/(ymin-ymax);
ctx.setTransform(m11,m12,m21,m22,dx,dy);
ctx.lineWidth = 1/Math.max(m11,m22);

function OHSA(Ao,w,b,t) {
   return (Ao * Math.cos(w*t) * Math.exp(-b*t));
}

var t = 0;
var A = OHSA(1,2*Math.PI,0.5,0);
var dt = 0.1;
ctx.beginPath();
ctx.moveTo(t,A);
for (var i=0;i<100;i++) {
   t = t + dt;
   A = OHSA(1,2*Math.PI,0.5,t);
   ctx.lineTo(t,A);
}
```

```
ctx.stroke();

</script>
```

Resultado:



No início do script são declaradas todas as variáveis necessárias para a realização da transformação de coordenadas e são calculados os coeficientes da matriz de transformação segundo o esquema descrito anteriormente. O método `setTransform()` é chamado com os parâmetros calculados, e a partir desse ponto o novo sistema de coordenadas está vigente. Como a espessura da linha fica afetada pela transformação de coordenadas, esta também deve ser ajustada utilizando o maior fator de escala dentre os dois calculados.

Note que o script também utiliza, por conveniência, a função `OHSA(Ao,w,b,t)`, que recebe como argumentos os valores da amplitude inicial, a frequência angular, o coeficiente de amortecimento e um instante de tempo, e retorna o valor da amplitude nesse instante.

Sua vez... (11-9)

Modifique o exemplo anterior de modo que o fator de amortecimento seja 10 vezes menor que o especificado, e que a escala na direção horizontal seja modificada de modo a fazer com que o gráfico resultante também esteja próximo a zero no último terço do quadro.

Animações

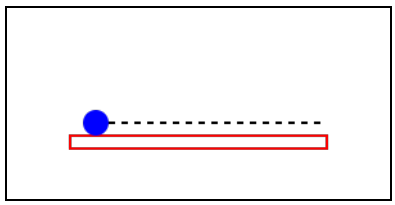
Animações no canvas são feitas de maneira semelhante às animações discutidas no capítulo sobre tempo, utilizando o método `setInterval()`. O exemplo a seguir retoma o exemplo da bola sobre a mesa do início deste capítulo, fazendo com que vá de um lado a outro.

Listagem:

```
<p style="text-align:center">
<canvas id="cnv" width="300" height="150"
        style="border:1px solid black">
</canvas>
</p>

<script>
var cnv = document.getElementById("cnv");
var ctx = cnv.getContext("2d");
var xbola = 60;
var timerBola = setInterval("passo()",100);

function passo() {

   ctx.clearRect(0,0,300,150);

   // retângulo vermelho delineado
   ctx.beginPath();
   ctx.rect(50,100,200,10);
   ctx.strokeStyle = "red";
   ctx.lineWidth = 2;
   ctx.stroke();

   // linha preta horizontal
   ctx.beginPath();
   for (var i=0;i<19;i++) {
      ctx.moveTo(60 + 10*i,90);
      ctx.lineTo((60 + 10*i + 5),90);
   }
   ctx.lineWidth = 2;
   ctx.strokeStyle = "black";
   ctx.stroke();

   // bola azul preenchida
```

```
    ctx.beginPath();
    xbola = xbola + 5;
    ctx.arc(xbola,90,10,0,2*Math.PI,true);
    ctx.fillStyle = "blue";
    ctx.fill();

    if (xbola>230) clearInterval(timerBola);

}
</script>
```

Resultado:



Compare o exemplo acima com o do início do capítulo e analise atentamente as diferenças. Em primeiro lugar, o procedimento de desenho de *tudo* o que desejamos que apareça no canvas foi colocado dentro de uma função `passo()`. Esta função é passada como argumento para o método `setInterval()` no começo do script e é ela que vai repetir-se (a cada 100 milissegundos) até que a coordenada *x* da bola ultrapasse o valor estipulado no `if` da última linha de código da função, quando é desligada.

A animação propriamente dita é feita adicionando 5 unidades à coordenada *x* da bola a cada disparo do temporizador. Note que, no início da função, o canvas é totalmente limpo com o método `clearRect(x,y,w,h)`, onde `x,y` são as coordenadas iniciais do retângulo a ser apagado e `w,h` sua largura e altura.

> Os desenhos no canvas, apesar de feitos um a um, resultam apenas em pixels coloridos na tela, e não em estruturas independentemente gerenciáveis. Assim, quando muitos elementos têm que ser apagados e redesenhados no canvas durante uma animação, ela pode ficar mais lenta do que o desejável. Tecnologias vetoriais, como o SVG (*scalable vector graphics*), também compatíveis com o HTML, são uma opção, mas não serão abordadas neste texto.

Sua vez... (11-10)

Modifique o exemplo anterior de modo que (a) o movimento horizontal seja substituído por um lançamento oblíquo a 45 graus com uma velocidade inicial tal que a bola atinja a outra extremidade da mesa e (b) ao invés de parar, o movimento fique sendo repetido indefinidamente (*dica:* ao invés de desligar o temporizador quando a condição de fim for atingida, pense em reinicializar as variáveis).

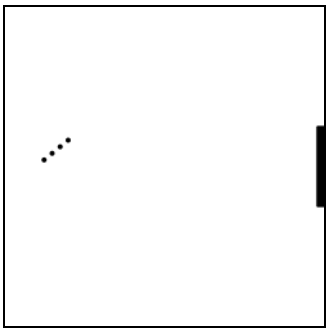
O exemplo a seguir modela uma bola movendo-se dentro de uma caixa bidimensional. A bola sai sempre do mesmo ponto, mas com uma velocidade (vetorial) aleatoriamente escolhida. No modelo, não há gravidade ou dissipação de energia nas colisões, e a bola sempre anda em linha reta e ficaria quicando indefinidamente caso sua trajetória não fosse interrompida ao atingir o "gol".

Listagem:

```
<p style="text-align:center">
<canvas id="cnv" width="250" height="250"
        style="border:1px solid black">
</canvas>
</p>

<script>
var cnv = document.getElementById("cnv");
var ctx = cnv.getContext("2d");
var cW = cnv.width, cH = cnv.height;
var xmin = -1, xmax = +1;
var ymin = -1, ymax = +1;
ctx.setTransform(cW/(xmax-xmin),0,0,-cH/(ymax-ymin),cW/2,cH/2);
var rpix = (xmax-xmin)/cW;
ctx.lineWidth = rpix;

var x = -0.8;
var y = 0;
var vx = 1;
```

```
var vy = -1 + 2 * Math.random();
var dt = 0.05;
var tictac = setInterval("passoBola()",50);

function passoBola() {
    ctx.clearRect(-1,-1,2,2);
    if ((x>0.95*xmax)&&(y<0.25*ymax)&&(y>0.25*ymin))
     clearInterval(tictac);
    if ((x>xmax)||(x<xmin))
        vx = -vx;
    if ((y>ymax)||(y<ymin))
        vy = -vy;
    x = x + vx * dt;
    y = y + vy * dt;
    ctx.beginPath();
    ctx.rect(0.95*xmax,0.25*ymin,0.05,0.5);
    ctx.fill();
    ctx.beginPath();
    ctx.arc(x,y,2*rpix,0,2*Math.PI,0);
    ctx.fill();
}
</script>
```

Resultado:



No primeiro bloco do script são definidas várias variáveis que auxiliam na construção da transformação de coordenadas. As variáveis `xmin`, `xmax`, `ymin` e `ymax` contêm os limites do espaço considerado no modelo (o "mundo"); as variáveis `cW` e `cH` contém a largura e a altura do canvas. Como especificada, a instrução `setTransform` coloca o eixo *x* (horizontal) e o eixo *y* (vertical) no meio do canvas, com a origem no centro dele. Os extremos nas duas direções passam a ser −1 e +1. O sinal de menos no quarto parâmetro faz com que o eixo *y*, que originalmente aponta para baixo, passe a apontar para cima.

A instrução `var rpix = (xmax-xmin)/cW` define uma variável de conveniência, que contém o tamanho aproximado de 1 pixel no novo sistema de coordenadas. Esse tamanho pode ser utilizado como referência para definir a espessura mínima de uma linha e para o raio mínimo de um círculo (arco) a ser desenhado no canvas.

A função `passoBola()` limpa o canvas, verifica se a bola bateu no "gol" (desligando o temporizador, caso isso ocorra) ou nas paredes (invertendo o sinal da respectiva componente da velocidade, caso isso ocorra). Finalmente, atualiza as coordenadas e desenha a bola.

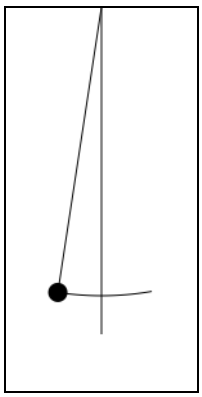
Note particularmente os usos dos operadores lógicos `&&` (E) e `||` (OU) para a construção da condição de fim e de reversão das velocidades.

Sua vez... (11-11)

Modifique o exemplo anterior de modo a (a) fazer com que a bola saia do centro do canvas e (b) incluir um segundo "gol" na lateral esquerda do canvas (oposto ao existente).

Exercícios

1. A figura abaixo ilustra um pêndulo simples. Reproduza-a utilizando os métodos de desenho em um canvas.



Algumas dicas:

a. Escolha uma escala proporcional. A figura foi gerada em um canvas 150 × 300 com as coordenadas na direção *x* variando de −0.5 a 0.5 e na direção *y* variando de 0 a 2.

b. O desenho tem quatro elementos: uma esfera, uma linha ligando o ponto de sustentação à esfera, uma linha vertical e um arco que, no exemplo, tem 10 graus à esquerda e 10 graus à direita da linha vertical.

c. No sistema de coordenadas do pêndulo, a linha vertical faz um ângulo de 270 graus com a origem dos ângulos.

d. Codifique cada um deles separadamente, isto é, com um `beginPath()` encerrado por um `stroke()` ou um `fill()`.

2. Anime o pêndulo do exercício anterior. Algumas dicas:

a. O ângulo que o pêndulo faz com a vertical em função do tempo pode ser dado por $\theta(t) = \theta_0 \cos(2\pi f t)$, onde θ_0 é o ângulo no instante $t = 0$ e f é a frequência.

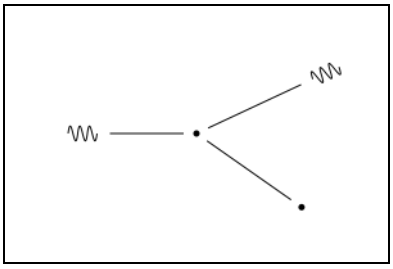
b. A frequência f depende do comprimento L do pêndulo e da aceleração da gravidade g: $f = (g/L)^{1/2}$.

c. As coordenadas x e y são dadas por: $x = L \operatorname{sen}\theta$ e $y = H - L \cos\theta$, onde H é a altura do suporte onde o pêndulo está preso.

d. Para uma animação realista, utilize o objeto `Date()` para obter o tempo inicial e o tempo de cada atualização, utilizando a diferença entre eles como valor de t. Não se esqueça que a diferença é dada em milissegundos (volte ao exemplo do cronômetro no capítulo sobre tempo).

3. Modifique o script do exercício anterior para incluir um amortecimento (volte ao problema do oscilador harmônico amortecido discutido no texto) e faça gráficos das coordenadas x e y do pêndulo em função do tempo.

4. Uma das formas da interação da radiação com a matéria é o chamado espalhamento Compton. Nele, uma partícula de luz, o fóton, colide com uma partícula carregada, por exemplo um elétron. Após a colisão, um novo fóton com uma energia menor sai com um ângulo e o elétron com outro ângulo em relação à direção de incidência. A diferença entre a energia do fóton incidente e do fóton espalhado, bem como os ângulos com que eles saem após a colisão podem ser calculados utilizando as leis da conservação da energia e do momento linear. A figura abaixo ilustra o processo:



Escreva um programa para desenhar a figura. Algumas dicas:

a. Foi definido um canvas 300 × 200 com uma escala de -15 a 15 na direção x e -10 a 10 na direção y.

b. O fóton e o elétron antes da colisão estão em (-10,0) e (0,0), respectivamente.

c. Os fótons podem ser desenhados concatenando curvas Bézier ou funções harmônicas

(seno ou cosseno).

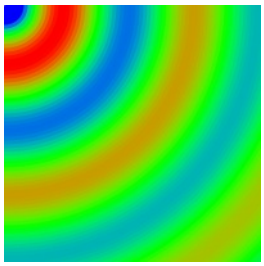
d. Para desenhar o fóton espalhado, o canvas foi rodado de 25 graus. No sistema rodado, as coordenadas do fóton são (10,0).

e. Para desenhar o elétron espalhado, o canvas foi rodado de -35 graus. No sistema rodado, as coordenadas do elétron espalhado são (10,0).

f. Utilize os métodos `save()` e `restore()` para preservar e recuperar o sistema de coordenadas original do canvas.

5. A figura ao lado ilustra um tipo de *passeio aleatório* (andar do bêbado), em que um objeto dá um passo de tamanho arbitrário, mas sempre igual, em uma direção aleatoriamente escolhida (um ângulo entre 0 e 2π sorteado ao acaso, por exemplo). Faça um programa que modele o passeio aleatório da figura. *Dica:* o canvas ao lado tem 100 pixels × 100 pixels, o início do passeio se dá em seu centro e cada passo tem 2 pixels.

6. Refaça o programa que modela uma onda bidimensional animada, feito utilizando uma tabela no capítulo anterior, mas agora utilizando o canvas.



7. Faça um relógio analógico que mostre o tempo real. Utilize os métodos do objeto `Date()` para obter horas, minutos e segundos do relógio do sistema e transforme seus valores em ângulos para os ponteiros.

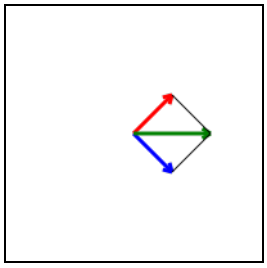
8. Volte ao capítulo sobre interação e utilize caixas de texto para construir um documento que contenha entradas para as componentes de dois vetores e saídas para as componentes do vetor soma resultante. Além disto, o script deve mostrar em um canvas os vetores fornecidos e o vetor resultante.

$\mathbf{v}_1$ = [3] **i** + [3] **j**

$\mathbf{v}_2$ = [3] **i** + [-3] **j**

SOMA

$\mathbf{v}_3$ = [6] **i** + [0] **j**



Bibliografia comentada

1. BOS, B., CELIK, T., HICKSON, I. (Eds.) **Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification, W3C Recommendation 07 June 2011.** Cambridge: W3C Consortium, 2011. Disponível em: http://www.w3.org/TR/CSS21/>. Acesso em: 29/04/2015.

   Este é o documento em que o consórcio internacional que gerencia a web propõe essencialmente como devem funcionar os navegadores no que se refere à interpretação das declarações de estilo. Não é um documento destinado a estudantes de programação, mas principalmente a desenvolvedores de navegadores e similares, mas vale a pena uma visita.

2. HICKSON, I. (Ed.) **HTML5: A vocabulary and associated APIs for HTML and XHTML. W3C Proposed Recommendation 16 September 2014.**. Cambridge: W3C (World Wide Web Consortium). Disponível em . Acesso em 29/10/2015.

   Este é o enorme documento em que o consórcio internacional que gerencia a web propõe essencialmente como devem funcionar os navegadores no que se refere à interpretação de HTML. Não é um documento destinado a estudantes de programação, mas principalmente a desenvolvedores de navegadores e similares, mas vale a pena uma visita.

3. LAURSEN, O., SCHNUR, D. **FLOT: an attractive JavaScript plotting for jQuery, v. 0.8.3.** Alborg: IOLA, 2015. Disponível em . Acesso em: 29/10/2015.

   É de onde você pode baixar o FLOT, *plug-in* para facilitar a geração de gráficos bonitos com funções ou dados. Vale a pena olhar em detalhes todos os exemplos para ter uma dimensão de suas potencialidades. Se você achar que precisa de mais, experimente buscar por HIGHCHARTS, por exemplo. Se realmente quiser se profissionalizar em visualização de dados, tente buscar pelo D3.

4. POWELL, T. A. **HTML & CSS: the complete reference (Fifth Edition).** New York: McGraw Hill, 2010.

   Um livro que talvez você queira ter quando não tem uma conexão com a internet. Aborda HTML e CSS em detalhes, mas não tem nada sobre JavaScript ou programação.

5. PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., FLANNERY, B. P. **Numerical Recipes (Third Edition)**. Cambridge: Cambridge University Press, 2007. Disponível em . Acesso em: 29/04/2015.

   Provavelmente a referência mais conhecida e citada sobre métodos numéricos. Tem de tudo, e códigos para tudo em várias linguagens. Ainda não há versão para JavaScript, mas como há para C e C++, a adaptação é fácil e direta.

6. REFSNES Data. **w3schools Tutorials**. 2015. Disponível em: . Acesso em 22/10/2015.

   Esse é "o" site de referência para HTML, CSS, JavaScript e várias outras tecnologias. E você nem precisa ter entre os seus favoritos: qualquer busca que incluir "JavaScript" como uma das palavras chave vai trazer um link para esse site no topo da lista.

7. SCHERER, C. **Métodos Computacionais da Física.** São Paulo: Livraria da Física, 2005.

   Um tradicional livro de física computacional, em português, mais voltado para aqueles interessados em seguir algum tipo de pesquisa em física utilizando métodos numéricos.

8. WIRFS-BROCK, A. (Ed.) **ECMA-262 ECMAScript Language Specification, 6 Edition, June 2015.** Geneva: ECMA International, 2015. . Acesso em: 06/10/2015.

   Este é o documento que contém a especificação do que é comumente chamado de JavaScript, cujo nome oficial é ECMA-262. É um documento longo, não escrito para ser lido por um estudante de programação, mas para os desenvolvedores de navegadores. No entanto, um passeio pelo documento pode dar uma boa dimensão do que é uma linguagem de programação e de como são pensadas as suas características.

Bibliografia comentada

1. BOS, B., CELIK, T., HICKSON, I. (Eds.) **Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification, W3C Recommendation 07 June 2011.** Cambridge: W3C Consortium, 2011. Disponível em: http://www.w3.org/TR/CSS21/>. Acesso em: 29/04/2015.

   Este é o documento em que o consórcio internacional que gerencia a web propõe essencialmente como devem funcionar os navegadores no que se refere à interpretação das declarações de estilo. Não é um documento destinado a estudantes de programação, mas principalmente a desenvolvedores de navegadores e similares, mas vale a pena uma visita.

2. HICKSON, I. (Ed.) **HTML5: A vocabulary and associated APIs for HTML and XHTML. W3C Proposed Recommendation 16 September 2014.**. Cambridge: W3C (World Wide Web Consortium). Disponível em . Acesso em 29/10/2015.

   Este é o enorme documento em que o consórcio internacional que gerencia a web propõe essencialmente como devem funcionar os navegadores no que se refere à interpretação de HTML. Não é um documento destinado a estudantes de programação, mas principalmente a desenvolvedores de navegadores e similares, mas vale a pena uma visita.

3. LAURSEN, O., SCHNUR, D. **FLOT: an attractive JavaScript plotting for jQuery, v. 0.8.3.** Alborg: IOLA, 2015. Disponível em . Acesso em: 29/10/2015.

   É de onde você pode baixar o FLOT, *plug-in* para facilitar a geração de gráficos bonitos com funções ou dados. Vale a pena olhar em detalhes todos os exemplos para ter uma dimensão de suas potencialidades. Se você achar que precisa de mais, experimente buscar por HIGHCHARTS, por exemplo. Se realmente quiser se profissionalizar em visualização de dados, tente buscar pelo D3.

4. POWELL, T. A. **HTML & CSS: the complete reference (Fifth Edition).** New York: McGraw Hill, 2010.

   Um livro que talvez você queira ter quando não tem uma conexão com a internet. Aborda HTML e CSS em detalhes, mas não tem nada sobre JavaScript ou programação.

5. PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., FLANNERY, B. P. **Numerical Recipes (Third Edition)**. Cambridge: Cambridge University Press, 2007. Disponível em . Acesso em: 29/04/2015.

   Provavelmente a referência mais conhecida e citada sobre métodos numéricos. Tem de tudo, e códigos para tudo em várias linguagens. Ainda não há versão para JavaScript, mas como há para C e C++, a adaptação é fácil e direta.

6. REFSNES Data. **w3schools Tutorials**. 2015. Disponível em: . Acesso em 22/10/2015.

   Esse é "o" site de referência para HTML, CSS, JavaScript e várias outras tecnologias. E você nem precisa ter entre os seus favoritos: qualquer busca que incluir "JavaScript" como uma das palavras chave vai trazer um link para esse site no topo da lista.

7. SCHERER, C. **Métodos Computacionais da Física.** São Paulo: Livraria da Física, 2005.

   Um tradicional livro de física computacional, em português, mais voltado para aqueles interessados em seguir algum tipo de pesquisa em física utilizando métodos numéricos.

8. WIRFS-BROCK, A. (Ed.) **ECMA-262 ECMAScript Language Specification, 6 Edition, June 2015.** Geneva: ECMA International, 2015. . Acesso em: 06/10/2015.

   Este é o documento que contém a especificação do que é comumente chamado de JavaScript, cujo nome oficial é ECMA-262. É um documento longo, não escrito para ser lido por um estudante de programação, mas para os desenvolvedores de navegadores. No entanto, um passeio pelo documento pode dar uma boa dimensão do que é uma linguagem de programação e de como são pensadas as suas características.

Loading [MathJax]/extensions/tex2jax.js

Apêndice A: equações e gráficos

Equações

Escrever equações matemáticas não é uma tarefa fácil em nenhum sistema. Mas graças à enorme quantidade de pessoas que usam e desenvolvem ferramentas para gerar conteúdos para a internet, existem diversas soluções para os mais variados gostos.

A maneira mais rápida e direta de se produzir equações em HTML é ser criativo com os recursos dos próprios elementos HTML e declarações de estilo. Isto em geral funciona para equações pequenas e simples, mas mesmo assim com resultados nem sempre muito satisfatórios. O exemplo a seguir mostra algumas possibilidades.

Listagem:

```
<i>V</i> =
(4/3)&amp;pi;<i>R</i><sup>3</sup>
```

$V = (4/3)\pi R^3$

```
<i>r</i> =
exp[<i>a</i> / (<i>b</i> <sup>_</sup>
<i>c</i>)<sup>1/2</sup>]
```

$r = \exp[a / (b - c)^{1/2}]$

```
<b>&tau;</b> =
<b>r</b> &amp;times; <b>F</b>
```

$\boldsymbol{\tau} = \mathbf{r} \times \mathbf{F}$

```
<b>a</b> =
<i>d</i><sup>2</sup><b>x</b>/
<i>d</i><i>t</i><sup>2</sup> =
<i>a</i><sub>x</sub> <b>i</b> +
<i>a</i><sub>y</sub> <b>j</b> +
<i>a</i><sub>z</sub> <b>k</b>
```

$\mathbf{a} = d^2\mathbf{x}/dt^2 = a_x\,\mathbf{i} + a_y\,\mathbf{j} + a_z\,\mathbf{k}$

```
<i>I</i> = (1/2)<i>M</i>
(<i>R</i><sub>1</sub><sup>2</sup> +
<i>R</i><sub>2</sub><sup>2</sup>)
```

$I = (1/2)M(R_1^2 + R_2^2)$

```
<font size="+2">&amp;int;</font>
<i>x</i><sup>2</sup> <i>d</i><i>x</i> =
<i>x</i><sup>3</sup>/3
```

$\int x^2\,dx = x^3/3$

Note a presença de vários caracteres especiais escritos com códigos que começam com "`&`" e terminam com um "`;`". No apêndice, você vai encontrar uma listagem com dezenas desses caracteres. Note, em particular, que o sinal de multiplicação, ×, não é um "x" e o sinal de menos, −, não é um hífen, "-".

π → π	μ → μ	ω → ω
× → ×	− → −	∫ → ∫

Sua vez.. (0-NaN)

Estude a sintaxe dos exemplos anteriores e construa sequências de texto com marcadores HTML de modo a produzir as equações abaixo:

$$x(t) = A\,\mathrm{sen}(\omega t) + B\cos(\omega t)$$

$$y = v_t\,[\,t - (m/k)(\,1 - e^{-(k/m)t}\,)\,]$$

É trabalhoso, prolixo e maçante escrever equações desta maneira. Além disso, equações mais complexas podem não ficar claras ou não serem apresentadas na forma que usualmente utilizamos. Considere, por exemplo, a equação a seguir que, além de difícil de especificar, produz uma equação de baixíssima qualidade editorial.

Listagem:

```
<i>&rho;</i> = <i>&rho;</i><sub>r</sub>
[ (<i>D</i><sub>b</sub><sup>3</sup>/3 –
<i>H</i><sub>r</sub> <i>D</i><sub>h</sub><sup>2</sup>/2) /
(<i>D</i><sub>b</sub><sup>3</sup>/3 –
<i>H</i> <i>D</i><sub>h</sub><sup>2</sup>/2)  ]
```

Resultado:

$$\rho = \rho_r \, [\, (D_b^3/3 - H_r \, D_h^2/2) \, / \, (D_b^3/3 - H \, D_h^2/2) \,]$$

O comitê internacional que propõe as especificações para a internet (W3C, ou *World wide web consortium*) desenvolveu, para isso, o MathML. O MathML é um poderoso esquema de marcação dos elementos de uma equação que, além de proporcionar uma formatação de qualidade gráfica elevada, permite marcar cada número e símbolo em uma equação individualmente, permitindo com que a eles sejam atribuídas declarações de estilo ou que sejam manipulados utilizando JavaScript. Isso, entretanto, também tem um preço: a sintaxe da marcação é extremamente prolixa e trabalhosa, deixando ainda o código-fonte muito difícil de compreender.

Um exemplo pode deixar isso mais claro. A equação para a distribuição angular da função de onda de um elétron no estado com momento angular $\ell = 1$ e projeção $m_\ell = 0$ é dada por:

$$\Theta 10 (\theta) = 3 4 \pi \cos \theta$$

que, em MathML, é especificada da seguinte maneira:

```
<math display="block">
  <msub>
    <mi mathvariant="normal">&#x0398;</mi>
    <mrow>
      <mn>10</mn>
    </mrow>
  </msub>
  <mo stretchy="false">(</mo>
  <mi>θ</mi>
  <mo stretchy="false">)</mo>
  <mo>=</mo>
  <msqrt>
    <mfrac>
      <mn>3</mn>
      <mrow>
        <mn>4</mn>
        <mi>&#x03C0;</mi>
      </mrow>
    </mfrac>
  </msqrt>
  <mi>cos</mi>
  <mo>&#x2061;</mo>
  <mi>&#x03B8;</mi>
</math>
```

Parece absurdo ter que fazer isso para especificar uma simples equação mas, acredite, dentro da filosofia geral de marcação dos elementos de um documento segundo sua funcionalidade, isso tem suas razões e usos. Se não fosse assim, tanta gente inteligente e bem paga não perderia tempo desenvolvendo isso.

Entretanto, na maioria das vezes deseja-se simplesmente escrever uma equação sem atribuir qualquer marcação especial ou funcionalidade a seus elementos. Devido à importância e versatilidade da internet hoje, diversas entidades produziram aplicativos e *plug-ins* que podem ser utilizados *online* e *offline* para descrever e renderizar equações das maneiras tradicionalmente utilizadas por usuários de diferentes áreas. Neste texto optamos por apresentar uma estratégia que faz uso de um *plug-in* gratuito, o `MathJax.js`, que permite ao usuário digitar suas equações como se estivesse utilizando o LaTeX, e que automaticamente disponibiliza a codificação da equação também em MathML.

O exemplo a seguir mostra como montar um documento que incorpora o *plug-in* do `MathJax.js` previamente baixado no computador do usuário e colocado na pasta `c:/MathJax` e como codificar a equação acima utilizando a sintaxe do LaTeX.

```
<html>
<head>
<script type="text/x-mathjax-config">
  MathJax.Hub.Config({
    extensions: ["tex2jax.js"],
    jax: ["input/TeX","output/HTML-CSS"],
    tex2jax: {inlineMath: [["$","$"],["\\(","\\)"]]}
  });
</script>
<script src="file:///c:/MathJax/MathJax.js"></script>
</head>
<body>

\begin{equation}
\rho = \left(
\frac{\frac{1}{3}{D}_{b}^{3} - \frac{1}{2}{D}_{h}^{2}{H}_{r}}
     {\frac{1}{3}{D}_{b}^{3} - \frac{1}{2} {D}_{h}^{2} H    }
\right) {\rho}_{r}
\end{equation}

</body>
</html>
```

Ao carregar o documento, o navegador deverá apresentar somente algo como:

$$\rho = \left(\frac{\frac{1}{3}D_b^3 - \frac{1}{2}D_h^2 H_r}{\frac{1}{3}D_b^3 - \frac{1}{2}D_h^2 H} \right) \rho_r$$

No exemplo, foi assumido que o MathJax está na pasta `c:/MathJax` do computador do usuário, mas é possível especificar outros endereços, como a página do próprio MathJax (cdn.mathjax.org) na internet.

O LaTeX vem sendo há décadas uma das principais plataformas utilizadas na edição de textos técnicos e científicos de excelente qualidade editorial. Suas possibilidades são muitas e inúmeros livros, tutoriais e exemplos sobre seu uso podem ser encontrados na internet, assim como sobre o MathML e o MathJax.

Gráficos

Textos técnicos e científicos também frequentemente incluem gráficos e, novamente, a internet está repleta de *plug-ins* para auxiliar na produção deles. Uma busca vai lhe revelar uma constelação de possibilidades, das gratuitas às comerciais, das mais simples às extremamente sofisticadas. Todas funcionam de modo semelhante, apenas variando na complexidade com que são especificados os seus recursos (mas não se iluda, algumas pedem um elevado grau de dedicação para obter algum resultado!).

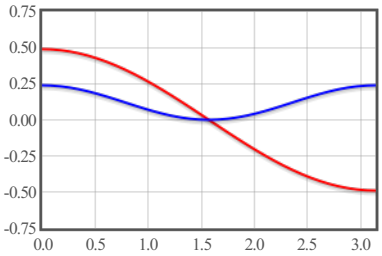
Neste documento utilizamos uma das mais simples, o Flot (www.flotcharts.org), para ilustrar como produzir gráficos "cotidianos" rapidamente. O exemplo a seguir mostra como fazer o gráfico da equação para a distribuição angular discutida na seção anterior.

Listagem:

```
<html>

<head>
<script src="file:///c:/flot/jquery.js"></script>
<script src="file:///c:/flot/jquery.flot.js"></script>
</head>

<body>

<div id='grT10' style='width:300;height:200;margin:auto'></div>

<script>

function T10(x) {
    return Math.sqrt(3/4/Math.PI)*Math.cos(x);
}

var ptsT10 = [];
var ptsT10sqr = [];

for (var theta=0;theta<=Math.PI;theta=theta+Math.PI/36) {
   ptsT10.push([theta,T10(theta)]);
   ptsT10sqr.push([theta,T10(theta)*T10(theta)]);
}
```

```
$.plot("#grT10",[ptsT10,ptsT10sqr],{ colors:["red","blue"] });

</script>

</body>
</html>
```

Resultado:



O exemplo se inicia com a inclusão, no cabeçalho (`head`) de dois arquivos previamente baixados no computador do usuário e colocados no diretório `c:/flot` (aqui também poderia ser feita uma referência ao site do Flot na internet).

O corpo do documento contém essencialmente dois blocos. No primeiro, apenas a definição da `div` que vai receber o gráfico. É obrigatória a atribuição de um `id` para a divisão, que será utilizado pela função que efetivamente desenha o gráfico, e de uma largura (`width`) e altura (`height`) em uma declaração de estilo (`style`).

No segundo bloco está o script que define a função de interesse. Em seguida, são declaradas duas matrizes (vetores), uma para armazenar os pontos para o gráfico da função de onda e outra para armazenar o gráfico do seu quadrado. O laço `for` utiliza o método `.push(...)` para adicionar a cada elemento das matrizes anteriormente declaradas uma matriz com apenas dois elementos que representam as coordenadas dos pontos do gráfico.

Na última linha do script é chamada a função `$.plot` do Flot, que recebe três parâmetros: o "endereço" da divisão onde o gráfico deve ser desenhado, uma matriz com a(s) matriz(es) de pontos a serem graficados e um objeto com as opções do gráfico. Este último parâmetro (o objeto com as opções do gráfico) pode ser definido previamente como uma variável bastante complexa, incluindo especificações sobre legendas, cores, tipos de gráficos (linhas, pontos, barras etc.), extremos e precisão das escalas, marcas de grade etc. Como usual, o leitor interessado pode encontrar detalhes e informações nos sites dos fornecedores dos *plug-ins* que optar utilizar.

Note que mesmo que se queira graficar apenas um conjunto de dados, é preciso colocar a matriz que contém os pontos dentro de outra matriz. Isto é, se o objetivo fosse graficar apenas a função de onda, a instrução seria:

```
$.plot("#grT10",[ptsT10]);
```

Note também que não foi passado o terceiro parâmetro com as opções do gráfico, o que faz com que as opções padrão sejam utilizadas.

Apêndice B: o valor de π

O valor aproximado de π pode ser computado comparando-se o número de pontos aleatoriamente distribuídos sobre um quadrado com o número de pontos aleatoriamente distribuídos sobre um círculo inscrito nele. A precisão com que o valor de π pode ser calculado cresce com o número de pontos utilizados.

Considere um círculo de raio R e área $A_c = \pi R^2$. O quadrado em que está inscrito terá lado $L = 2R$ e área $A_q = L^2 = 4R^2$. A razão entre as áreas é $A_c/A_q = \pi R^2/4R^2 = \pi/4$ e portanto $\pi = 4(A_c/A_q)$. Se os pontos estão distribuídos uniformemente sobre as áreas, o número de pontos dentro das figuras é proporcional a suas áreas, ou seja $A_c/A_q = N_c/N_q$ e $\pi = 4(N_c/N_q)$, onde N_c e N_q são o número de pontos sobre o círculo e sobre o quadrado, respectivamente.

O script abaixo determina o valor aproximado de π sorteando valores para as coordenadas x e y entre −1 e 1. Pontos com coordenadas entre −1 e 1 estão necessariamente dentro do quadrado, mas só estarão também dentro do círculo se sua distância até a origem, dada por $d = (x^2 + y^2)^{1/2}$, for menor do que o raio do círculo.

Listagem:

```
<table width='550'>
<tr>
<td>

 <div align="center">
 1 <input type="range" id="freqRng"
          min="1" max="100" value="50"
          onClick="atualiza()"> 100<br>
 <span id="freqVal"></span> pts/s
 </div>

 <div align="left">
 <br>
 N<sub>pts</sub> = <span id="npts"></span> <br>
 π<sub>calc</sub> = <span id="picalc"></span> <br>
 π<sub>real</sub> = <span id="pireal"></span> <br>
 Erro = <span id="pierro"></span>% <br>
 </div>

 <div align="center" style="margin-top:1em">
 <input type="button" value=" ZERA " onClick="zera()" />
 </div>

</td>
<td width="210">
 <div align="center">
 <canvas id='cnvPI' style='border: 1px solid;'
             width='200' height='200'></canvas>
 </div>
</td>
</tr>
</table>

<script>
var xmin = -1, xmax = +1, ymin = -1, ymax = +1;
var cnvPI = document.getElementById("cnvPI");
var ctxPI = cnvPI.getContext("2d");
var m11 = cnvPI.width/(xmax-xmin);
var m12 = 0;
var m21 = 0;
var m22 = cnvPI.height/(ymin-ymax);
var dx  = cnvPI.width/2;
var dy  = cnvPI.height/2;
var rpixPI = (xmax-xmin)/cnvPI.width;
ctxPI.setTransform(m11,m12,m21,m22,dx,dy);
ctxPI.lineWidth = rpixPI;
ctxPI.beginPath();
ctxPI.arc(0,0,1,0,2*Math.PI,true);
ctxPI.stroke();
document.getElementById("pireal").innerHTML = Math.PI;
var freq = parseInt(document.getElementById("freqRng").value);
```

```
var dtPI = parseInt(1000/freq);
document.getElementById("freqVal").innerHTML = freq;
var nQuad = 0;
var nCirc = 0;
var timerPI = setInterval("passo()",dtPI);

function zera() {
    nQuad = 0;
    nCirc = 0;
    document.getElementById("picalc").innerHTML = 0;
    document.getElementById("pierro").innerHTML = 0;
    document.getElementById("npts").innerHTML = 0;
    ctxPI.clearRect(-1,-1,2,2);
    ctxPI.beginPath();
    ctxPI.arc(0,0,1,0,2*Math.PI,true);
    ctxPI.stroke();
}

function atualiza() {
    var freq = parseInt(document.getElementById("freqRng").value);
    dtPI = parseInt(1000/freq);
    clearInterval(timerPI);
    timerPI = setInterval("passo()",dtPI);
    document.getElementById("freqVal").innerHTML = freq;
}

function passo() {
    var x = 2 * Math.random() - 1;
    var y = 2 * Math.random() - 1;
    var r = Math.sqrt(x*x + y*y);
    if (r < 1) nCirc++;
    nQuad++;
    ctxPI.beginPath();
    ctxPI.arc(x,y,rpixPI,0,2*Math.PI,true);
    ctxPI.fill();
    var picalc = 4*(nCirc/nQuad);
    document.getElementById("npts").innerHTML = nQuad;
    document.getElementById("picalc").innerHTML = picalc;
    document.getElementById("pierro").innerHTML =
            (picalc - Math.PI) / Math.PI * 100;
}
</script>
```

Resultado:

1 — 100

50 pts/s

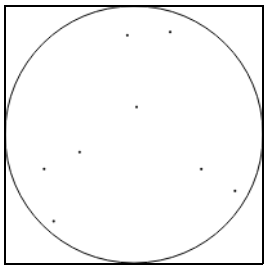
$N_{pts} = 8$

$\pi_{calc} = 4$

$\pi_{real} = 3.141592653589793$

Erro = 27.32395447351627%

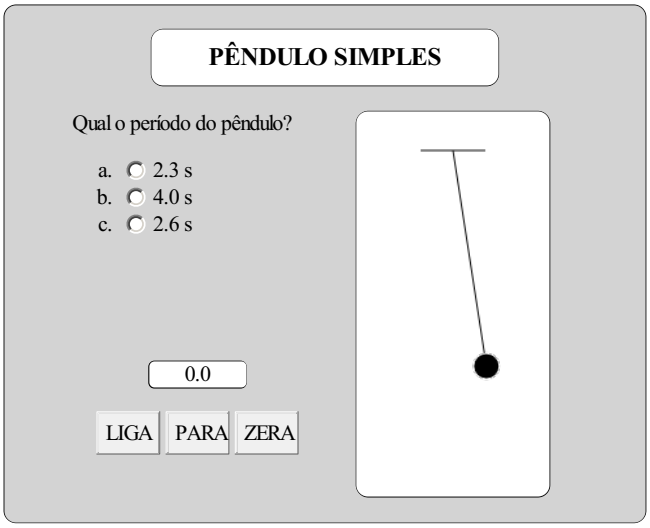
ZERA



Loading [MathJax]/extensions/tex2jax.js

Apêndice C: o pêndulo simples

O exemplo a seguir mostra como produzir uma questão de múltipla escolha para a qual o usuário deve (em princípio!) escolher uma resposta a partir de informações que obtém observando uma animação de um pêndulo simples.



O documento está essencialmente dividido em duas partes. A primeira é uma grande divisão que abarca uma tabela com as opções para o usuário, os controles do cronômetro e o canvas com o pêndulo. A segunda contém o JavaScript que gerencia tudo. No quadro a seguir o conteúdo da tabela e do script foram eliminados para tentar auxiliar na visualização desses dois grandes blocos.

```
<div style="width:500px; margin:auto;
            border:1px solid black; border-radius:10px;
            background-color:lightgray;
            padding-bottom:1em;
            margin-bottom:2em">

<p style="width:50%; margin:auto; margin-top:1em;
          border:1px solid black; border-radius:10px;
          background-color:white; color:black;
          text-align:center; padding:10px;
          font-size:14pt;font-weight:bold">
PÊNDULO SIMPLES
</p>

<table style="width:80%; margin:auto; margin-top:1em">
...
</table>

</div>

<script>
...
</script>
```

Esse exemplo procura mostrar com um pouco mais de detalhes como as declarações de estilo podem ser utilizadas para obter-se o efeito desejado. Muitas são relativamente óbvias, mas não deixe de pesquisar na internet o significado delas (por exemplo, `padding` e `margin`).

A tabela é formada por uma linha com duas células. Na primeira célula está a questão com

suas alternativas e os controles do cronômetro. Na segunda célula está o canvas com o pêndulo.

```
<table style="width:80%; margin:auto; margin-top:1em">
<tr>
  <td style="text-align:left;vertical-align:top;width:50%">

  Qual o período do pêndulo?
  <ol style="list-style-type:lower-alpha">
  <li>
    <input type="radio" name="resp" id="r1" onChange="verifica(1)">
    <span id="e1"></span> s
  </li>
  <li>
    <input type="radio" name="resp" id="r2" onChange="verifica(2)">
    <span id="e2"></span> s
  </li>
  <li>
    <input type="radio" name="resp" id="r3" onChange="verifica(3)">
    <span id="e3"></span> s
  </li>
  </ol>

  <div style="width:100%; margin:auto; margin-top:6em;
              text-align:center">
  <div id="cronoOut"
        style="width:75px; height:20px;
               margin:auto; margin-bottom:1em;
               text-align:center;
               background-color:white;
               border:1px solid black; border-radius:5px" >0.0</div>
  <input type="button" value="LIGA" style="width:50px"
         onClick="ligaPend()">
  <input type="button" value="PARA" style="width:50px"
         onClick="paraPend()">
  <input type="button" value="ZERA" style="width:50px"
         onClick="zeraPend()">
  <div>
  </td>

  <td align="center">
  <canvas id="cnv" width="150" height="300"
          style="background-color:white;
                 border:1px solid black; border-radius:10px">
  </canvas>
  </td>

</tr>
</table>
```

Para a construção das alternativas foi utilizada uma lista ordenada (`ol`) indexada por caracteres alfabéticos minúsculos (`lower-alpha`). Cada item da lista (`li`) possui um elemento tipo `radio` com seu `id` único e um *container* do tipo `span`, também com seu `id`, que vai receber, dinamicamente, os valores das alternativas sorteadas aleatoriamente. Por conveniência, na mesma célula foram colocados uma divisão que vai receber o tempo marcado pelo cronômetro e os seus controles, que respondem de diferentes maneiras ao clique do mouse. Na célula seguinte está o `canvas`, com seu respectivo `id` e declaração de estilo (tamanho, fundo, borda).

O bloco com o script contém uma longa lista de declarações de variáveis com suas respectivas inicializações e as declarações das funções que vão responder ao pressionar de um botão e ao temporizador.

```
var xmin = -0.75, xmax = +0.75, ymin = 0, ymax = 3;
var cnv = document.getElementById("cnv");
var ctx = cnv.getContext("2d");
var cnvW = cnv.width; var cnvH = cnv.height;
var scx = cnvW/(xmax-xmin);
var scy = cnvH/(ymax-ymin);
var dx  = cnvW/2;
var dy  = cnvH;
ctx.setTransform(scx,0,0,-scy,dx,dy);
ctx.lineWidth = 1/scx;

var H = ymin + (ymax-ymin)*0.9; // altura do pto de fixacao
var lmin = 0.2;    // tamanho minimo do pendulo
var lmax = 2.5; // tamanho maximo do pendulo
var L = lmin + (lmax-lmin)*Math.random(); // tamanho do pendulo
var g = 9.8;    // aceleracao da gravidade
var T = 2*Math.PI*Math.sqrt(L/g); // periodo do pendulo

var alt = 1 + Math.floor(Math.random()*3); // alternativa correta
document.getElementById("e" + alt).innerHTML = T.toFixed(1);
for (var i=1;i<=3;i++) {
    if (i!=alt) {
        var x = T * 0.8*Math.random() + i;
        document.getElementById("e" + i).innerHTML = x.toFixed(1);
    }
}
```

```
var radios = document.querySelectorAll('input[type=radio]');
for (var i = 0; i < radios.length; i++) {
    radios[i].checked = false;
}

var inicioPend;
var cronoPend;
var t = 0, dt = 0.1;
var timerPend = setInterval("movePend()",100);
```

O código acima está dividido essencialmente em quatro blocos. No primeiro estão as instruções que viabilizam a transformação de coordenadas. No segundo, as definições das variáveis associadas ao movimento do pêndulo propriamente dito. O terceiro, bastante trabalhoso de interpretar, a definição de qual alternativa será a correta e a distribuição dos valores para as demais (e, adicionalmente, um conjunto de instruções que garante que nenhuma alternativa estará marcada quando a página for recarregada em alguns navegadores). O quarto (e final) bloco define as variáveis associadas ao tempo e inicializa o temporizador que movimenta o pêndulo.

```
function verifica(opt) {
    if (opt==alt) alert("Parabéns!");
    else alert("Tente novamente!");
}

function ligaPend() {
    if (cronoPend) clearInterval(cronoPend);
    inicioPend = new Date();
    cronoPend = setInterval("cronometro()",100);
}

function paraPend() {
    clearInterval(cronoPend)
}

function zeraPend() {
    document.getElementById("cronoOut").innerHTML = "0.0";
}

function cronometro() {
    var agoraPend = new Date();
    var tempoPend =
        (agoraPend.getTime() - inicioPend.getTime())/1000;
    document.getElementById("cronoOut").innerHTML =
        tempoPend.toFixed(1);
}

function movePend() {
    var theta = Math.PI/20 * Math.cos(2*Math.PI/T*t);
    var x = L * Math.sin(theta);
    var y = H - L * Math.cos(theta);
    t = t + dt;

    // limpa a tela
    ctx.clearRect(-0.75,0,1.5,3);

    // desenha o teto
    ctx.beginPath();
    ctx.moveTo(-0.25,H);
    ctx.lineTo(+0.25,H);
    ctx.strokeStyle = "black";
    ctx.stroke();

    // desenha o fio
    ctx.beginPath();
    ctx.moveTo(0,H);
    ctx.lineTo(x,y);
    ctx.strokeStyle = "black";
    ctx.stroke();

    // desenha a esfera
    ctx.beginPath();
    ctx.arc(x,y,0.1,0,2*Math.PI,0);
    ctx.fillStyle = "black";
    ctx.fill();
    ctx.strokeStyle = "white";
    ctx.stroke();
}
</script>
```

Fica ao leitor o desafio de estudar, pesquisar e entender os detalhes e juntar esses fragmentos em um documento que efetivemante funcione.

Loading [MathJax]/extensions/tex2jax.js

Apêndice D: alguns caracteres especiais

¡	¡	Β	Β	γ	γ	•	•	∉	∉	⊄	⊄
©	©	Γ	Γ	δ	δ	…	…	∋	∋	⊆	⊆
®	®	Δ	Δ	ε	ε	′	′	∏	∏	⊇	⊇
¯	¯	Ε	E	ζ	ζ	″	″	∑	∑	⊕	⊕
°	°	Ζ	Z	η	η	ℑ	ℑ	−	−	⊗	⊗
±	±	Η	H	θ	θ	ℜ	ℜ	∗	∗	⊥	⊥
²	²	Θ	Θ	ι	ι	™	™	√	√	⋅	⋅
³	³	Ι	I	κ	κ	ℵ	ℵ	∝	∝	⟨	〈
µ	µ	Κ	K	λ	λ	←	←	∞	∞	⟩	〉
¶	¶	Λ	Λ	μ	μ	↑	↑	∠	∠	"	"
·	·	Μ	M	ν	ν	→	→	∧	∧	&	&
¹	¹	Ν	N	ξ	ξ	↓	↓	∨	∨	<	<
º	º	Ξ	Ξ	ο	ο	↔	↔	∩	∩	>	>
¼	¼	Ο	O	π	π	↵	↵	∪	∪	œ	œ
½	½	Π	Π	ρ	ρ	⇐	⇐	∫	∫	–	–
¾	¾	Ρ	P	ς	ς	⇑	⇑	∴	∴	—	—
¿	¿	Σ	Σ	σ	σ	⇒	⇒	∼	~	‘	‘
Å	Å	Τ	T	τ	τ	⇓	⇓	≅	≅	’	’
×	×	Υ	Y	υ	υ	⇔	⇔	≈	≈	“	“
Ø	Ø	Φ	Φ	φ	φ	∀	∀	≠	≠	”	”
å	å	Χ	X	χ	χ	∂	∂	≡	≡	†	†
÷	÷	Ψ	Ψ	ψ	ψ	∃	∃	≤	≤	‡	‡
ø	ø	Ω	Ω	ω	ω	∅	∅	≥	≥	‰	‰
ƒ	ƒ	α	α	ϑ	ϑ	∇	∇	⊂	⊂	‹	‹
Α	A	β	β	ϒ	ϒ	∈	∈	⊃	⊃	›	›