# 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 $\varepsilon_0 = 8{,}9 \times 10^{-12}$ $C^2/N \cdot m^2$. 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 $k = 8.99 \times 10^9$ $\mathrm{N{\cdot}m^2/C^2}$ e $q$ a carga elétrica. A carga elementar é $1.6 \times 10^{-19}$ 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 $E_x(x,y,z) = kq\mathrm{sen}\theta\cos\varphi/(x^2+y^2+z^2)$ , $E_y(x,y,z) = kq\mathrm{sen}\theta\mathrm{sen}\varphi/(x^2+y^2+z^2)$ e $E_z(x,y,z) = kq\cos\theta/(x^2+y^2+z^2)$, onde $k = 8.99 \times 10^9$ $\mathrm{N{\cdot}m^2/C^2}$ e $q$ a carga elétrica. A carga elementar é $1.6 \times 10^{-19}$ 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 **a** = ($a_x$, $a_y$, $a_z$) e **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 **c** = **a** × **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.