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:

```
KLU
```

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.