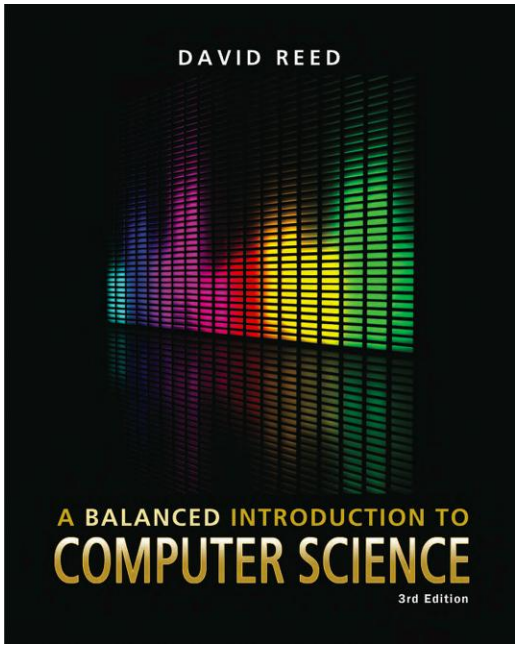
A Balanced Introduction to Computer Science, 3/E

David Reed, Creighton University

ISBN 978-0-13-216675-1



Chapter 15
JavaScript Strings

Strings as Objects

so far, your interactive Web pages have manipulated strings in simple ways

- use text box to input a word or phrase
- store that text in a (string) variable
- incorporate the text in a message, possibly using + to concatenate

strings are different from numbers and Booleans in that they are *objects*

- a *software object* is a unit of code that encapsulates both data and operations that can be performed on that data

- a string is a software object that models words and phrases

data:	a sequence of characters, enclosed in quotes
operations include:	make upper case, make lower case, determine the number of characters, access a particular character, search for a particular character, …

Object-Oriented Programming

objects are fundamental in the dominant approach to developing software systems: *object-oriented programming (OOP)*

- OOP encourages programmers to design programs around software objects
  - the programmer identifies the real-world objects involved in a system
    (e.g., for a banking program: bank account, customer, teller, …)
  - then designs and builds software objects to model these real-world objects

- OOP is effective for managing large systems, since individual objects can be assigned to different teams and developed independently
- OOP also supports code reuse, since the same or similar objects can be combined in different ways to solve different kinds of problems

example: a doorbell button

- has physical components/properties: color, shape, label, …
- has functionality: when you press the button, the bell rings

an HTML button is a software object that models a real-world button

- has physical components/properties: color, shape, label, …
- has functionality: when you click on the button, JavaScript code is executed

Properties and Methods

using object-oriented terminology,

- the characteristics of an object are called *properties*
  - e.g., a string object has a `length` property that identifies the number of characters in the string
- the operations that can be performed on the string are called *methods*
  - e.g., the `toLowerCase` method makes a copy of the string with all upper-case letters converted to lower-case

properties and methods are not new concepts

- a property is a *special kind* of a variable (it stores a value)
- a method is a *special kind* of function (it performs some action)

what is special is that they are associated with (or "belong to") an object

- e.g., each string object will have its own variable to stores it length

to access an object property, specify: object name, a period, property name

```
str1 = 'foo';
len1 = str1.length;
```

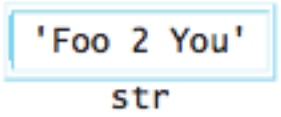
```
str2 = 'Hi there';
len2 = str2.length;
```

Properties and Methods

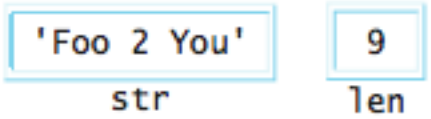
similarly, to call a method: object name, period, method call

- e.g., `str.toLowerCase()` calls the `toLowerCase` method on `str` (which *returns* a lowercase copy of the string)
- e.g., `str.toUpperCase()` calls the `toUpperCase` method on `str` (which *returns* an uppercase copy of the string)

```
str = 'Foo 2 You';
```




```
len = str.lenght;
```




```
upStr = str.toUpperCase ();
```

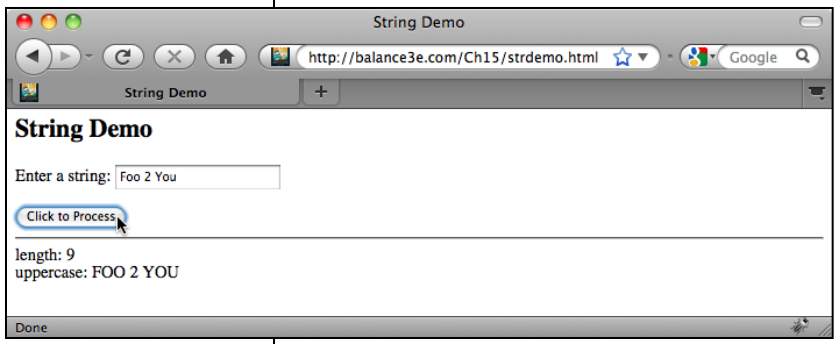

```
downStr = str.toLowerCase ();
```

note: the `toLowerCase` and `toUpperCase` methods do not change the string object they are called on (only an assignment can do that!)

- instead, they return modified copies of the string

String Manipulation Page

```
1. <html>
2. <!-- strdemo.html                                          Dave Reed -->
3. <!-- This page demonstrates several string properties and operations -->
4. <!-- ============================================================== -->
5.
6.  <head>
7.    <title> String Fun </title>
8.    <script type="text/javascript">
9.      function Process()
10.     // Assumes: strBox contains a string
11.     // Results: displays the outcome of string operations in outputDiv
12.     {
13.       var str;
14.
15.       str = document.getElementById('strBox').value;
16.
17.       document.getElementById('outputDiv').innerHTML =
18.         'length: ' + str.length + '<br>' +
19.         'uppercase: ' + str.toUpperCase() + '<br>';
20.     }
21.   </script>
22. </head>
23.
24. <body>
25.   <h2>String Demo</h2>
26.   <p>
27.     Enter a string: <input type="text" id="strBox" size=20 value="">
28.   </p>
29.   <input type="button" value="Click to Process" onclick="Process();">
30.   <hr>
31.   <div id="outputDiv"></div>
32. </body>
33. </html>
```



Common String Methods

useful methods exist that allow programmers to access and manipulate individual components of a string

- components are identifiable via *indices*, or numbers that correspond to the order in which individual characters occur in a string
- indices are assigned in ascending order from left to right, so that the first character in the string is at index 0

the `charAt` method provides access to a single character within the string

- it takes an index as an input and returns the character at that particular index

```
word = 'foo';
ch = word.charAt(0);                // ASSIGNS ch = 'f'
```

the `substring` method provides access to an entire sequence of characters within the string

- it takes two numbers as inputs, representing the starting (inclusive) and ending (exclusive) indices of the substring, and returns the substring

```
word = 'foo';
sub = word.substring(1, 3);         // ASSIGNS sub = 'oo'
```

String Access/Concatenation

recall: the concatenation operator (+) can join strings together

assuming the variable word stores a string value, what affect would the following assignment have?

```
word = word.charAt(0) + word.substring(1, word.length);
```

the following function takes a string as input and uses string method calls to create (and return) a capitalized version of that string

```
function Capitalize(str)
// Assumes: str is a word
// Returns: str with first letter capitalized, all others lowercase
{
    var firstLetter, restString, cap;
    firstLetter = str.charAt(0);                    // GET FIRST CHAR
    restString = str.substring(1, str.length);      // GET REST OF WORD
    cap = firstLetter.toUpperCase() + restString.toLowerCase();
                                                    // PUT BACK TOGETHER
    return cap;
}
```

Searching Strings

the `search` method traverses a string in order to locate a given character or substring

- it takes a character or string as input and returns the index at which the character or string first occurs (or -1 if not found)

```
str = 'banana';
num1 = str.search('n');   // ASSIGNS num1 = 2 since the character
                          //   'n' first occurs at index 2
num2 = str.search('ana'); // ASSIGNS num2 = 1 since the string
                          //   'ana' first occurs at index 1
num3 = str.search('z');   // ASSIGNS num3 = -1 since the character
                          //   'z' does not occur anywhere
```

simple application: determine whether a string is a single word or a phrase

- if the string contains no spaces, the call `str.search(' ')` will return -1, indicating that the string value consists of a single word
- if `str.search(' ')` returns a nonnegative value, then the presence of spaces signifies a phrase containing multiple words

General Searches

there are times when you want to search for a type of character, rather than a specific value

example: converting a word into Pig Latin

- if a word contains no vowels or begins with a vowel, the characters 'way' are appended to the end of the word

  nth → nthway apple → appleway

- if a word begins with a consonant, its initial sequence of consonants is shifted to the end of the word followed by 'ay'

  banana → ananabay cherry → errychay

in order to distinguish between these two cases, must `search` for the first vowel

- then, use the `substring` method to break the string into parts and the + operator to put the pieces back together (with 'ay')

  cherry → erry + ch + ay = errychay

General Searches

rather than having to search for vowels individually, an entire class of characters can be specified using /[. . .]/

`phrase.search(/[aeiou]/)`	returns the index of the first occurrence of a lowercase vowel in `phrase`; returns −1 if not found
`phrase.search(/[aeiouAEIOU]/)`	returns the index of the first occurrence of a lowercase or uppercase vowel in `phrase`; returns −1 if not found
`phrase.search(/[a-z]/)`	returns the index of the first occurrence of lowercase letter in `phrase`; returns −1 if not found
`phrase.search(/[a-zA-Z]/)`	returns the index of the first occurrence of lowercase or uppercase letter in `phrase`; returns −1 if not found
`phrase.search(/[0-9]/)`	returns the index of the first occurrence of a digit in `phrase`; returns −1 if not found
`phrase.search(/[ .,;:'!\?]/)`	returns the index of the first occurrence of a space or punctuation mark in `phrase`; returns −1 if not found

Strings and Repetition

some tasks involve repeatedly performing the same operations

- to accomplish such tasks, we can combine while loops with string methods such as `charAt` and `search`

example: a while loop used to access and process each character in a string

- the characters that comprise the string are concatenated one-by-one onto another string, resulting in an exact copy

```
str = 'abcd';
copy = '';                              // INITIALIZE copy TO EMPTY STRING

i = 0;                                  // START AT BEGINNING OF str
while (i < str.length) {                // AS LONG AS CHARS LEFT IN str
    copy = copy + str.charAt(i);        //   ADD CHAR TO END OF copy
    i=i+1;                              //   GO TO NEXT CHAR
}
```

	copy	i	str.charAt(i)
before loop	''	0	'a'
after 1st loop pass	'a'	1	'b'
after 2nd loop pass	'ab'	2	'c'
after 3rd loop pass	'abc'	3	'd'
after 4th loop pass	'abcd'	4	''

Example: Substitution Ciphers

a substitution cipher is a code for encrypting/decrypting messages

- one letter of the alphabet is substituted for another in the message
- Atbash cipher (500 B.C.) was used by Hebrew scribes
- Caesar cipher (50-60 B.C.) was used by Julius Caesar

Atbash cipher:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

⇓ ⇓ ⇓ ⇓ ⇓ ⇓ ⇓ ⇓ ⇓ ⇓ ⇓ ⇓ ⇓

ZYXWVUTSRQPONMLKJIHGFEDCBA

ABC → ZYX

HELLO → SVOOL

Caesar cipher:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

⇓ ⇓ ⇓ ⇓ ⇓ ⇓ ⇓ ⇓ ⇓ ⇓ ⇓ ⇓ ⇓

DEFGHIJKLMNOPQRSTUVWXYZABC

ABC → DEF

HELLO → KHOOR

substitution ciphers are easy to understand and use

- 26! ≈ 4 x 10^{26} possible substitution keys

Text Areas

a text area is similar to a text box but it can contain any number of text lines

general form of a text area element:

```
<textarea id='AREA_ID' rows=NUM_ROWS cols=NUM_COLS>
INITIAL_TEXT
</textarea>
```

- the ID attribute gives the element an identifier so that it can be referenced
- the ROWS attribute specifies the height (number of text lines) of the area
- the COLS attribute specifies the width (number of characters) of the area

unlike a text box, opening and closing tags are used to define a text area

- any text appearing between the tags will be the initial text in the text area
- otherwise, the contents of a text area are accessed/assigned in the same way

```
document.getElementById('AREA_ID').value
```

```
1.  <html>
2.  <!-- cipher.html                                    Dave Reed -->
3.  <!-- This page encodes messages using a simple substitution -->
4.  <!-- cipher (with message and key entered by the user)      -->
5.  <!-- ================================================== -->
6.
7.   <head>
8.     <title> Substitution Cipher </title>
9.
10.    <script type="text/javascript">
11.      function Encode()
12.      // Assumes: the message to be encoded is in messageArea (all caps),
13.      //          the key for encoding is in keyBox (all caps)
14.      // Results: the coded version of message is displayed in outputDiv
15.      {
16.        var message, key, alphabet, coded, i, ch, index;
17.
18.        message = document.getElementById('messageArea').value;
19.        key = document.getElementById('keyBox').value;
20.        alphabet = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
21.        coded = '';
22.
23.        i = 0;
24.        while (i < message.length) {      // FOR AS MANY LETTERS AS THERE ARE
25.            ch = message.charAt(i);       // ACCESS EACH LETTER IN MESSAGE
26.            index = alphabet.indexOf(ch); // FIND ITS POSITION IN ALPHABET
27.            if (index == -1) {            // IF NOT A CAPITAL LETTER,
28.               coded = coded + ch;        // THEN ADD IT UNCHANGED
29.            }                             // OTHERWISE,
30.            else {                        // ADD THE CORRESPONDING LETTER
31.               coded = coded + key.charAt(index); // IN THE KEY STRING
32.            }
33.            i = i + 1;
34.        }
35.
36.        document.getElementById('outputDiv').innerHTML = coded;
37.      }
38.    </script>
39.  </head>
40.
41.  <body>
42.    <h2>Substitution Cipher</h2>
43.    <p>
44.      Key: <input type="text" id="keyBox" size=26
45.                  value="ZYXWVUTSRQPONMLKJIHGFEDCBA">
46.    </p>
47.    <p>
48.      Enter your message below: <br>
49.      <textarea id ="messageArea" rows=8 cols=30></textarea> <br>
50.        <input type="button" value="Encode the Message" onclick="Encode();">
51.    </p>
52.    <hr>
53.    <div id="outputDiv"></div>
54.  </body>
55. </html>
```

Encoding a message

pseudocode:

for as many letters as there are in the message

- get the next character in the message
- find its position in the alphabet
- find the corresponding letter in the key
- use that letter to encode the current letter in the message