**Text Boxes**

HTML event handlers enable the user to interact with the page
- e.g., move the mouse over an image to change it
- e.g., click on a button to display a text message in a page division

For greater control, the user must be able to enter information into the page
- e.g., enter words to complete a fill-in-the-blank story
- e.g., enter grades to calculate a course average

A **text box** is an HTML element that is embedded in the page

```
<input type="text" id="BOX_ID" size=NUM_CHARS value="INITIAL_CONTENTS">
```

- the user can enter text directly in the box
- a JavaScript statement can then access the contents of the text box by accessing its VALUE attribute

```
document.getElementById('BOX_ID').value
```
Greetings Page

1. <!doctype html>
2. <!-- greet.html
3. <!-- Web page that displays a personal greeting
4. <!-- ---------------------------------------->
5. 6. <html>
6.   <head>
7.     <title> Greetings </title>
8.   </head>
9.  
10. 11. <body>
12.   <h2>Greetings</h2>
13.   <p>
14.     Enter your name: <input type="text" id="nameBox" size=12 value="">
15.   </p>
16.   <input type="button" value="Click for Greeting"
17.     onclick="document.getElementById('outputDiv').innerHTML=
18.         'Hello' + document.getElementById('nameBox').value + 
19.         ', welcome to my page.<br>Do you mind if I call you' +
20.         document.getElementById('nameBox').value + '?';">
21.  
22.   <hr>
23.   <div id="outputDiv"></div>
24. </body>
25. </html>

since value="" ,
the text box
is initially
empty

the user can
enter his/her
name in the
text box
when the button is clicked, the text in the box is accessed and incorporated into the message.
Mixing Text & Expressions

when displaying a complex message involving text and box contents, special care must be taken

- any part of the message enclosed in quotes is treated as plain text (including formatting tags)
- expressions that access the contents of a text box must be evaluated by the browser → cannot be enclosed in quotes
- all of the pieces of the message are concatenated together using '++'
a Web page can have numerous text boxes
- each must have a unique ID
- text assigned to the VALUE attribute is automatically displayed
- useful whenever a default value is natural
**JavaScript Variables**

JavaScript assignments have been used to directly assign values to attributes

```javascript
document.getElementById('mysteryImg').width = 100;
```

assignments can also be used for indirect actions via variables

a *variable* is a name used to symbolize a dynamic (changeable) value

- each variable is associated with a memory cell
- when a value is assigned to a variable, that value is stored in the corresponding memory cell

```javascript
userName = 'Dave';
```

- any subsequent reference to a variable evaluates to the value stored in its memory cell

```javascript
document.getElementById('outputDiv').innerHTML = 'Hi ' + userName;
```
**Variable Names**

A variable name can be any sequence of letters, digits, and underscores, as long as it starts with a letter.

- Variable names should be chosen to be descriptive of its purpose.

---

<table>
<thead>
<tr>
<th>Reserved words that shouldn’t be used as variable names because they are already used by JavaScript or the browser.</th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract</td>
</tr>
<tr>
<td>all</td>
</tr>
<tr>
<td>anchor</td>
</tr>
<tr>
<td>area</td>
</tr>
<tr>
<td>boolean</td>
</tr>
<tr>
<td>break</td>
</tr>
<tr>
<td>button</td>
</tr>
<tr>
<td>byte</td>
</tr>
<tr>
<td>case</td>
</tr>
<tr>
<td>catch</td>
</tr>
<tr>
<td>char</td>
</tr>
<tr>
<td>class</td>
</tr>
<tr>
<td>const</td>
</tr>
<tr>
<td>continue</td>
</tr>
<tr>
<td>date</td>
</tr>
<tr>
<td>debugger</td>
</tr>
</tbody>
</table>
Variables for Reuse

variables can simplify code by giving a short name to a complex expression

- e.g., in greetings page, can assign the text box contents to a variable
- then, can use the variable repeatedly in the message

```html
<input type="button" value="Click for Greeting"
    onclick="userName=document.getElementById('nameBox').value;
             document.getElementById('outputDiv').innerHTML =
             'Hello ' + userName + ', welcome to my page.<br>' +
             'Do you mind if I call you ' + userName + '?';">
```

example application: a fill-in-the-blank story page

- page contains text boxes with label such as Name, Color, Animal, Place
- the user enters word/phrase choices in these boxes
- those choices are inserted into a story and displayed in a page division

- because the words/phrases may be used several times in the story, it will make the code shorter (and less error prone) to first assign box contents to variables, then use the variables in the story
Common Pattern

many pages will follow the same basic pattern

- text boxes allow the user to enter words/phrases
- at the click of a button, the text box contents are accessed & stored in variables
- a message incorporating the variable values is displayed

the JavaScript code executed at the button click similarly follows a pattern

```
VAR1 = document.getElementById('BOX_ID1').value;
VAR2 = document.getElementById('BOX_ID2').value;
...
VARn = document.getElementById('BOX_IDn').value;

document.getElementById('outputDiv').innerHTML = MESSAGE_INTEGRATING_STRING_LITERALS_AND_VARIABLES;
```
Variables for Temps

sometimes, a variable is needed to store a value so it is not lost

when the button is clicked, the IMG sources are swapped

- the leftImg source is saved in a variable
- then, can overwrite leftImg source with rightImg source
- finally, can assign stored value to rightImg

```html
1. <!doctype html>
2. <!-- swap.html     Dave Reed -->
3. <!-- Web page that swaps two images at the click of a button. -->
4. <!-- __________________________________________________ -->
5. 
6. <html>
7. <head>
8. <title>Image Swapper</title>
9. </head>
10. 
11. <body>
12.  <div style="text-align:center">
13.   <h2>Image Swapper</h2>
14.   </p>
15.   <img id="leftImg" src="http://balance3e.com/Images/happy.gif">
16.   <img id="rightImg" src="http://balance3e.com/Images/sad.gif">
17.   </p>
18.   <input type="button" value="Swap Images" onclick="saved=document.getElementById('leftImg').src; document.getElementById('leftImg').src=document.getElementById('rightImg').src; document.getElementById('rightImg').src=saved;">">
19. </div>
20. </body>
21. </html>
```
Data Types

each unit of information processed by a computer belongs to a general category or data type
- e.g., string, number, Boolean (either true or false)

each data type is associated with a specific set of predefined operators that may be used by programmers to manipulate values of that type
- e.g., we have seen string concatenation via +
- similarly, standard operators are predefined for numbers
  - addition (+), subtraction (-), multiplication (*), division (/)

variables can be assigned various kinds of numerical values, including mathematical expressions formed by applying operators to numbers
- when an expression appears on the right-hand side, the expression is evaluated and the resulting value is assigned to the variable on the left-hand side

```plaintext
phrase = 'howdy' + 'doo';

x = 50/4;
```
Variables and Expressions

if a variable appears in an expression, the value currently assigned to that variable is substituted

\[
\begin{align*}
x &= 24; \\
y &= (100 * 10) + 24; \\
x &= y - 1;
\end{align*}
\]

when you read an assignment statement, refrain from using \textit{equals} for '='

- '=' does not represent equality, it represents assignment
- read it as \textit{gets}

\[
\begin{align*}
x &= x + 1; & \rightarrow & \quad x \textit{ gets } x + 1;
\end{align*}
\]
useful facts about JavaScript numbers

- to improve readability, very large or very small number are displayed in
  scientific notation: $X\times10^Y$ represents the value $X \times 10^Y$
  - e.g., $1e24 \rightarrow 1 \times 10^{24} \rightarrow 1000000000000000000000000$

- JavaScript stores all numbers in memory cells of a fixed size (64 bits)
  - as a result, only a finite number of values can be represented
  - e.g., $1e308$ can be represented, but $1e309$ is treated as Infinity
  - $1e-323$ can be represented, but $1e-324$ is treated as 0

- even within the range $1e-323 \ldots 1e309$, not all numbers can be represented
  - note that between any two numbers lie infinitely more numbers!
  - JavaScript can represent approximately 17 significant digits
  - e.g., $0.9999999999999999$ can be represented exactly
  - $0.9999999999999999$ is rounded up to 1
Text Boxes and Numbers

special care must be taken when accessing numbers from a text box

- The content of a text box is always accessed as a string (sequence of characters)
- E.g., if the user enters 500 in a box, then the value '500' is accessed

```javascript
myNumber = document.getElementById('numBox').value;
alert('One more is ' + (myNumber + 1));
```

- If the user entered 12 in the box, what would be displayed?
- The alert message would be *One more is 121*  
  - The box content is accessed as '12', which is stored in `myNumber`
  - The parenthesized sub-expression `(myNumber + 1)` is evaluated first
  - Since this is a mixed expression, the number 1 is converted to '1'
  - The result, '121', is then concatenated to the end of 'One more is'

What is needed is a mechanism for converting strings of digits into numbers

- E.g., '500' → 500, '1.314' → 1.314, ...
parseFloat Function

in mathematics, a function is a mapping from inputs to a single output
- e.g., the absolute value function: |-5| → 5, |17.3| → 17.3

from a programmer's view, a function is a "unit of computational abstraction"
- there is some computation required to calculate the output given the input(s)
- a JavaScript function encapsulates that computation and hides the details
  - applying a function to inputs is known as calling the function
  - the output of a function call is known as the return value

the parseFloat function takes a string as input and returns a number
- For example:
  - parseFloat('10') -> 10, parseFloat("1.11") -> 1.11
  - If document.getElementById('myID').value -> "1234",
    parseFloat(document.getElementById('myID').value) -> 1234
calling the `parseFloat` function on the text in the box converts it to a number. This number is then assigned to `amount`.
Common Pattern

similarly, Web pages that compute a value will follow the same basic pattern

- text boxes allow the user to enter numbers
- at the click of a button, the text box contents are accessed, `parseFloat` is applied to convert to numbers, and the numbers are stored in variables
- a computation involving those numbers is performed
- the result of the computation is displayed in the page

the JavaScript code executed at the button click similarly follows a pattern

```javascript
VAR1 = parseFloat(document.getElementById('BOX_ID1').value);
VAR2 = parseFloat(document.getElementById('BOX_ID2').value);
... 
VARn = parseFloat(document.getElementById('BOX_IDn').value);
RESULT = EXPRESSION_INVOLVING_VARIABLES;

document.getElementById('outputDiv').innerHTML = 
MESSAGE_INTEGRATING_STRING_LITERALS_AND_RESULT;
```
in computer jargon, the term **bug** refers to an error in a program

the process of systematically locating and fixing errors is **debugging**

three types of errors can occur

1. **syntax errors**: typographic errors
   - e.g., omitting a quote or misspelling a function name
   - since the browser catches these, they are usually "easy" to identify and fix

2. **run-time errors**: occur when operations are applied to illegal values
   - e.g., attempting to multiply a string or divide by zero
   - also caught by the browser, which either produces an error message or else returns a special value (string multiplication produces NaN, for "Not a Number"; division by zero produces Infinity)

3. **logic errors**: flaws in the design or implementation of a program
   - whenever your program produces the wrong result
   - since they are not caught by the browser (the program is legal, just not what you wanted), logic errors are hardest to identify

useful technique for identifying bugs: **diagnostic alert statements**

- at various intervals in the code, display the values of key variables using alert
- you can then isolate at what point the program is going wrong