

CSC 121

Computers and Scientific Thinking

David Reed
Creighton University

JavaScript and Dynamic Web Pages

1

Static vs. Dynamic Pages



recall: a Web page uses HTML tags to identify page content and formatting information

HTML can produce only *static pages*

- static pages look the same and behave in the same manner each time they are loaded into a browser

in 1995, researchers at Netscape developed JavaScript, a language for creating *dynamic pages*

- Web pages with JavaScript can change their appearance:
 - over time (e.g., a different image each time that a page is loaded), or
 - in response to a user's actions (e.g., typing, mouse clicks, and other input methods)

2

Programming Languages

JavaScript is a *programming language*

- a *programming language* is a language for specifying instructions that a computer can execute
- each *statement* in a programming language specifies a particular action that the computer is to carry out
(e.g., changing an image or opening a window when a button is clicked)

some programming languages are general-purpose

- popular languages include C++, Java, J#

JavaScript was defined for a specific purpose: *adding dynamic content to Web pages*

- can add JavaScript statements to a Web page using the HTML tags

```
<script type="text/javascript"> . . . </script>
```

- when the browser displays the page, any statements inside the SCRIPT tags are executed and the result is displayed

3

Simple Dynamic Page

below is a simple Web page with dynamic content

- *note*: dynamic content can be mixed with static HTML content

it demonstrates two types of JavaScript statements

- an `assignment` statement that asks the user for input and stores that input
- a `write` statement that writes text into the HTML page

```

1. <html>
2. <!-- greet.html                               Dave Reed -->
3. <!-- Web page that displays a personalized greeting. -->
4. <!-- ----- -->
5.
6. <head>
7.   <title> Greetings </title>
8. </head>
9.
10. <body>
11.   <script type="text/javascript">
12.     firstName = prompt("Please enter your name", "");
13.
14.     document.write("<p>Hello " + firstName + ", welcome to my Web page.</p>");
15.   </script>
16.
17.   <p>
18.     Whatever else you want to appear in your Web page...
19.   </p>
20. </body>
21. </html>
```

4

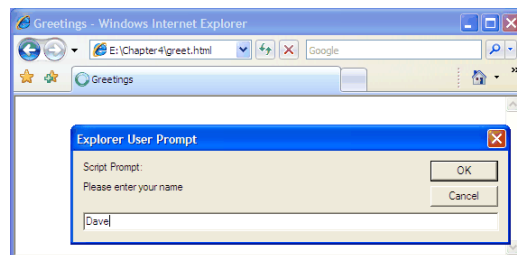
Assignment Statement

when an assignment statement involving `prompt` is executed by the browser

- a separate window is opened with a text box for the user to enter text
- when the user is done typing, he/she can click on the OK button

```
firstName = prompt("Please enter your name", "");
```

- when OK is clicked, the text entered is assigned to a variable
 - a *variable* is a name used to symbolize a dynamic value
 - here, the variable `firstName` is used to store the text entered by the user



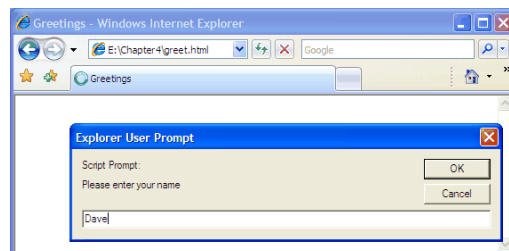
5

Assignment Statement (cont.)

the general form of an assignment statement using a prompt is

```
VARIABLE = prompt("PROMPT MESSAGE", "");
```

- the variable name can vary depending on the task at hand
 - here, the variable is used to store the user's first name, so `firstName` is a meaningful name
- the prompt message that appears in the window can likewise change
 - here, the message "Please enter your name" tells the user what is expected



6

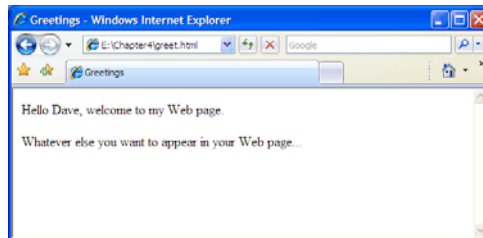
Write Statement

when a write statement is executed by the browser

- the message specified in the statement is written into the HTML page
- a message can include
 - a string literal – text enclosed in quotes
 - a variable
 - a combination of strings and variables, connected via '+'

```
document.write("<p>Hello " + firstName +
               ", welcome to my Web page.</p>");
```

- when a variable is encountered, the browser substitutes the value currently assigned to that variable



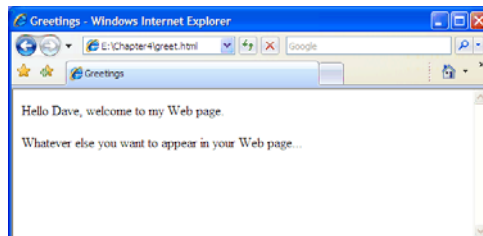
7

Write Statement (cont.)

the general form of a write statement is

```
document.write("MESSAGE TO BE DISPLAYED " + VARIABLE +
               " MORE MESSAGE" + ...);
```

- note that the statement can be broken across lines, as long as no string literal is split (i.e., the beginning and ending quotes of a string must be on same line)
- the pieces of the message are displayed in sequence, with no spaces in between
 - if you want spaces, you have to enter them in the text



8

Formatted Output

the output produced by a write statement is embedded in the page

- the browser displays this output just as it does any other text
- if the text contains HTML tags, the browser will interpret the tags and format the text accordingly

```
document.write("<p>Hello <i>" + firstName +  
              "</i>, welcome to my Web page.</p>");
```

assuming the variable `firstName` has been assigned "Dave", the browser would execute the statement to produce

```
<p>Hello <i>Dave</i>, welcome to my Web page.</p>
```

which would be displayed by the browser as

```
Hello Dave, welcome to my Web page.
```

9

Syntax Errors

an error in the format of an HTML or JavaScript statements is known as a *syntax error*

- some syntax errors are ignored by the browser
 - e.g., misspelling an HTML tag name
- most JavaScript syntax errors will generate an error message

```
document.write("This example is illegal since the  
              string is broken across lines");
```

yields: **Error: unterminated string literal**

```
document.write("The value of x is " x);
```

yields: **Error: missing) after argument list**

10

JavaScript Variables

a variable name can be any sequence of letters, digits, and underscores (but must start with a letter)

- valid: `tempInFahr` `SUM` `current_age` `Sum2Date` `x`
- invalid: `2hotforU` `salary$` `two words` `"sum_to_date"`

variable names are case sensitive, so `Sum` and `SUM` are treated as different variables

Reserved Words That Shouldn't Be Used as Variable Names				
abstract	document	if	package	throw
boolean	double	implements	parent	throws
break	else	import	private	top
byte	enum	in	protected	transient
case	export	instanceof	public	true
catch	extends	int	return	try
char	false	interface	screen	typeof
class	final	length	self	var
const	finally	location	short	void
continue	float	long	static	volatile
debugger	for	name	super	while
default	function	native	switch	window
delete	goto	new	synchronized	with
do	history	null	this	

11

Variables & Memory Cells

computers keep track of the values that variables represent by associating each variable with a specific piece of memory, known as a *memory cell*

- when a JavaScript assignment is executed,


```
firstName = prompt("Please enter your name", "");
```
- the value entered by the user (e.g., "Dave") is stored in a memory cell associated with the variable `firstName`



- any future reference to the variable name evaluates to the value stored in its associated memory cell

12

Another Example

once you create a variable, you can repeatedly assign values to it

- only the most recent value is retained in memory

EXAMPLE: suppose we want to prompt the user for two different foods

- if only one food is needed at a time, we can reuse the same variable

```

1. <html>
2. <!-- food.html                               Dave Reed -->
3. <!-- Web page that prompts for and displays food preferences. -->
4. <!-- =====>
5.
6. <head>
7.   <title> Who's Hungry? </title>
8. </head>
9.
10. <body>
11.   <script type="text/javascript">
12.     food = prompt("What is your favorite food?", "");
13.     document.write("<p>Your favorite food is " + food + "</p>");
14.
15.     food = prompt("What is your least favorite food?", "");
16.     document.write("<p>Your least favorite food is " + food + "</p>");
17.   </script>
18. </body>
19. </html>

```

13

Reusing Variables

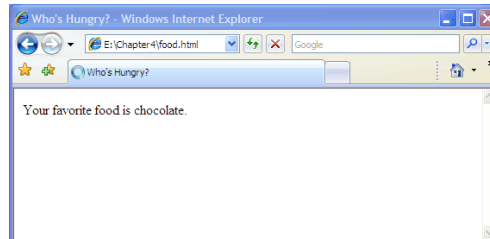
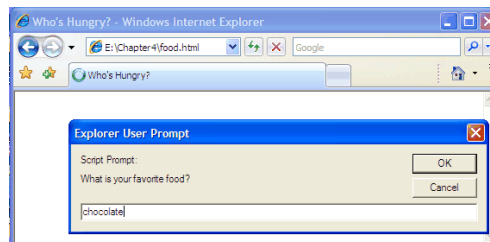
```

food = prompt("What is your favorite food?", "");
document.write("<p>Your favorite food is " + food + "</p>");

```

the first pair of statements

- stores the user's favorite food
- displays that food in the page



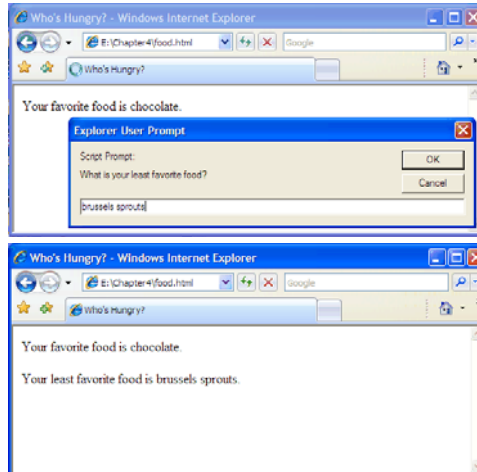
14

Reusing Variables (cont.)

```
food = prompt("What is your least favorite food?", "");
document.write("<p>Your least favorite food is " + food + "</p>");
```

the second pair of statements

- stores the user's least favorite food (overwriting the old value)
- displays that food in the page



15

Prompts with Defaults

so far, all prompts have been of the form

```
VARIABLE = prompt("PROMPT MESSAGE", "");
```

sometimes it makes sense to provide default values for prompts

- can specify a string literal instead of ""
- this string will appear in the prompt box when it appears
 - if the user wants to accept the default value, can just click OK

EXAMPLE: suppose we wanted to create a page that displays a verse of the children's song, *Old MacDonald had a Farm*

- the page should be able to display any verse
- can accomplish this by prompting the user for the animal and sound
- can specify default values so that it is easy to display a common verse

```
animal = prompt("Enter a kind of animal:", "cow");
sound = prompt("What kind of sound does it make?", "moo");
```

16

Old MacDonald

this page prompts the user for the animal and sound ("cow" and "moo", by default), then displays a verse using those values

- `
` tags are embedded to break the output onto separate lines

```

1. <html>
2. <!-- oldmac.html          Dave Reed -->
3. <!-- Web page that displays a verse of Old MacDonald. -->
4. <!-- ----- -->
5.
6. <head>
7. <title> Old MacDonald </title>
8. </head>
9.
10. <body>
11. <h3 style="text-align:center">Old MacDonald Had a Farm</h3>
12.
13. <script type="text/javascript">
14.     animal = prompt("Enter a kind of animal:", "cow");
15.     sound = prompt("What kind of sound does it make?", "moo");
16.
17.     document.write("<p>Old MacDonald had a farm, E-I-E-I-O.<br />");
18.     document.write("And on that farm he had a " + animal + ", E-I-E-I-O.<br />");
19.     document.write("With a " + sound + "-" + sound + " here, and a " +
20.         sound + "-" + sound + " there,<br />");
21.     document.write(" here a " + sound + ", there a " + sound +
22.         " everywhere a " + sound + "-" + sound + ".<br />");
23.     document.write("Old MacDonald had a farm, E-I-E-I-O.</p>");
24. </script>
25. </body>
26. </html>

```

17

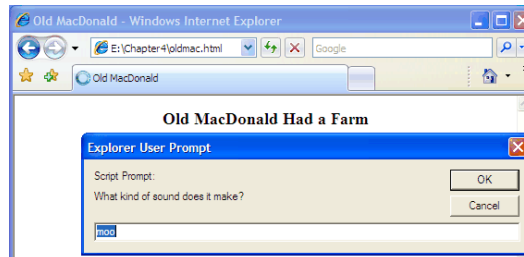
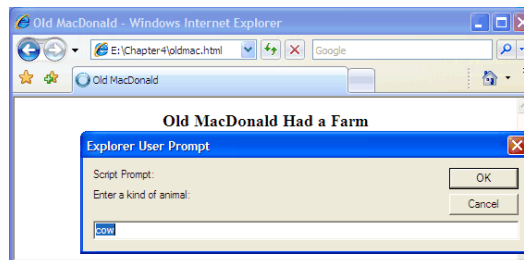
Old MacDonald (cont.)

the default values automatically appear in the prompt boxes

- the user can click OK to accept the defaults

OR

- type new values into the prompt box



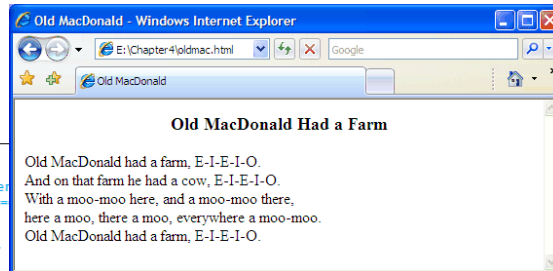
18

Old MacDonald (cont.)

```

1. <html>
2. <!-- oldmac.html
3. <!-- Web page that displays a ver
4. <!-- =====
5.
6. <head>
7.   <title> Old MacDonald </title>
8. </head>
9.
10. <body>
11.   <h3 style="text-align:center">Old MacDonald Had a Farm</h3>
12.
13.   <script type="text/javascript">
14.     animal = prompt("Enter a kind of animal:", "cow");
15.     sound = prompt("What kind of sound does it make?", "moo");
16.
17.     document.write("<p>Old MacDonald had a farm, E-I-E-I-O.<br />");
18.     document.write("And on that farm he had a " + animal + ", E-I-E-I-O.<br />");
19.     document.write("With a " + sound + "-" + sound + " here, and a " +
20.       sound + "-" + sound + " there,<br />");
21.     document.write(" here a " + sound + ", there a " + sound +
22.       ", everywhere a " + sound + "-" + sound + "<br />");
23.     document.write("Old MacDonald had a farm, E-I-E-I-O.</p>");
24.   </script>
25. </body>
26. </html>

```



19

Localizing Changes

so far, we have used variables to store values read in via prompts

another common use is to store values used repeatedly in a page

- suppose we wanted to change the spelling of the refrain in Old MacDonald ("E-I-E-I-O" → "Eeyigh-Eeyigh-Oh")
- as is, would need to find and update all occurrences in the verse

- instead, could use a variable to store the refrain

```

refrain = "E-I-E-I-O";
document.write("<p>Old MacDonald had a farm, " + refrain + "<br />");
document.write("And on that farm he had a " + animal + ", " +
  refrain + "<br />");
. . .

```

- now, to update the value in the entire verse, simply must change the assignment

```

refrain = "Eeyigh-Eeyigh-Oh";
document.write("<p>Old MacDonald had a farm, " + refrain + "<br />");
document.write("And on that farm he had a " + animal + ", " +
  refrain + "<br />");
. . .

```

20

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

```
word = "howdy" + " doo";
```

"howdy doo"

word

```
x = 50/4;
```

12.5

x

21

Variables and Expressions

similarly, expressions can appear in write statements

- note: parentheses can be used to make sub-expression grouping explicit

```
document.write(3 + 7);
```

→ writes 10

```
document.write("The sum of is " + (3 + 7));
```

→ writes The sum is 10

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

```
x = 24;
```

24

x

```
y = (100 * 10) + 24;
```

24

x

1024

y

```
x = y - 1;
```

1023

x

1024

y

22

Number Representati on

useful facts about JavaScript numbers

- to improve readability, very large or very small number are displayed in *scientific notation*: Xe^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 $1e-323 \dots 1e309$, 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

23

Mixed Expressi ons

in JavaScript, the + operator serves two purposes

- when applied to numbers, + means addition
- when applied to strings, + means concatenation
- what about a mixed expression?

when applied to a string and a number,

- the number is converted to a string (effectively, by placing quotes around it),
- then string concatenation is performed

```
"We're number " + 1 → "We're number " + "1"
                    → "We're number 1"
```

note: expressions involving + are evaluated left-to-right

- this can have consequences in the way mixed expressions are evaluated
- *ADVICE: always use parentheses to group nested sub-expressions*

```
3 + 2 + " is the sum" → (3 + 2) + " is the sum"
                    → 5 + " is the sum"
                    → "5" + " is the sum"
                    → "5 is the sum"
"the sum is " + 3 + 2 → ("the sum is " + 3) + 2
                    → ("the sum is " + "3") + 2
                    → "the sum is 3" + 2
                    → "the sum is 3" + "2"
                    → "the sum is 32"
```

24

Prompting for Numbers

special care must be taken when prompting the user for number values

- recall that `prompt` always returns a string, even if the user enters only digits
- e.g., if the user enters 500 at a prompt, then the value "500" is returned

```
myNumber = prompt("Enter a number", "");
document.write("One more is " + (myNumber + 1));
```

- if the user entered 12 at the prompt, what would be displayed?
- the message displayed would be `One more is 121` **WHY?**
 - the prompt returns "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" then concatenated
 - 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, ...
- this is accomplished in JavaScript using the `parseFloat` function

25

Functions

in mathematics, a *function* is a mapping from inputs to a single output

- e.g., the absolute value function maps one number to another
 $-5 \rightarrow 5, -2.4 \rightarrow 2.4, 17 \rightarrow 17, \dots$

$$|n| = \begin{cases} n & \text{if } n \geq 0 \\ -n & \text{if } n < 0 \end{cases}$$

- similarly, the `parseFloat` function maps strings of digits to numbers
`"500" → 500, "1.314" → 1.314, "0" → 0, ...`

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
- the user does not need to know how the function works, only how to apply it
 - applying a function to inputs is known as *calling the function*
 - the output of a function call is known as the *return value*

26

parseFloat

a function call can appear anywhere in a JavaScript expression

- when the expression is evaluated, the return value for that call is substituted

```
myNumber = prompt("Enter a number", "");
myNumber = parseFloat(myNumber);
document.write("One more is " + (myNumber + 1));
```

- the 1st statement prompts the user and stores their input (say "12") in myNumber
- the 2nd statement calls parseFloat to convert the string to a number (12) and then reassigns that number back to myNumber
- the 3rd statement uses the number value 12 to display `One more is 13`

note, the following is not an error (but probably not what was intended)

```
myNumber = prompt("Enter a number", "");
parseFloat(myNumber);
document.write("One more is " + (myNumber + 1));
```

- the call to parseFloat returns a number, but nothing is done with that number
- NOTE: the only way to change the value of a variable is via an assignment statement

27

Temperature Conversion

the following page prompts the user for a temperature (in Fahrenheit), stores the input as a number, then converts that temperature to Celsius

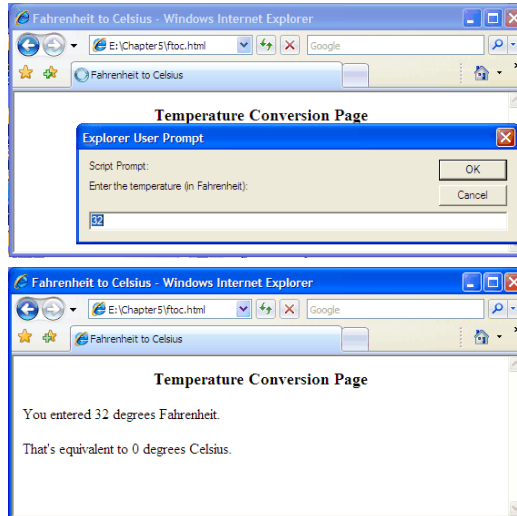
```
1. <html>
2. <!-- ftoc.html Dave Reed -->
3. <!-- Converts a temperature from Fahrenheit to Celsius. -->
4. <!-- =====>
5.
6. <head>
7. <title>Fahrenheit to Celsius</title>
8. </head>
9.
10. <body>
11. <h3 style="text-align:center">Temperature Conversion Page</h3>
12.
13. <script type="text/javascript">
14.     tempInFahr = prompt("Enter the temperature (in Fahrenheit):", "32");
15.     tempInFahr = parseFloat(tempInFahr);
16.
17.     tempInCelsius = (5/9) * (tempInFahr - 32);
18.
19.     document.write("<p>You entered " + tempInFahr + " degrees Fahrenheit.</p>");
20.     document.write("<p>That's equivalent to " + tempInCelsius +
21.         " degrees Celsius.</p>");
22. </script>
23. </body>
24. </html>
```

28

Conversion Page



note that the prompt has a default value of 32



29

Common Pattern



many tasks that we will consider have the same basic form

1. prompt the user for numbers
2. store them in variables
3. perform some calculation(s) using those numbers
4. display the results

not surprisingly, there is a pattern to the code

```
<script type="text/javascript">
  number1 = prompt("PROMPT MESSAGE", "");
  number1 = parseFloat(number1);
  number2 = prompt("PROMPT MESSAGE", "");
  number2 = parseFloat(number2);
  . . .
  numberN = prompt("PROMPT MESSAGE", "");
  numberN = parseFloat(numberN);

  answer = SOME EXPRESSION INVOLVING number1, ..., numberN;

  document.write("MESSAGE INVOLVING answer");
</script>
```

30

Predefined Functions

JavaScript provides an extensive library of predefined mathematical functions

- `Math.sqrt` returns the square root of a number
e.g., `Math.sqrt(9) → 3`
- `Math.max` returns the maximum of two numbers
e.g., `Math.max(3.2, 1.8) → 3.2`

```

1. <html>
2. <!-- mathtest.html           Dave Reed -->
3. <!-- This page tests the Math.sqrt function. -->
4. <!-- ===== -->
5.
6. <head>
7. <title>Function Tester</title>
8. </head>
9.
10. <body>
11. <h3 style="text-align:center">Math Function Tester</h3>
12.
13. <script type="text/javascript">
14.   number = prompt("Enter a number", 0);
15.   number = parseFloat(number);
16.
17.   result = Math.sqrt(number);
18.   document.write("<p>Math.sqrt(" + number + ") = " + result + "</p>");
19. </script>
20. </body>
21. </html>

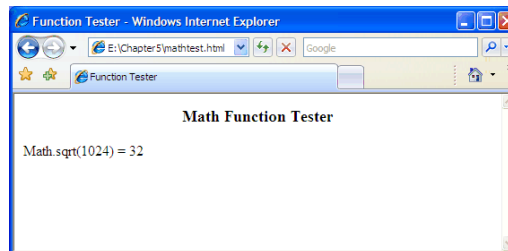
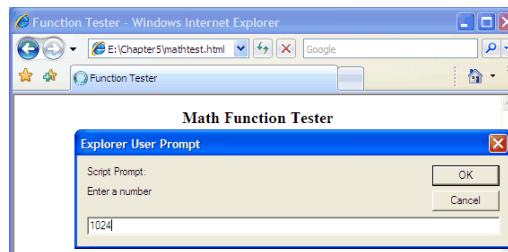
```

31

Tester Page

this page could be modified to test a variety of functions

- change the function call in the page
- enter various inputs and observe the corresponding outputs



32

Other Useful Functions

`Math.pow` raises a number to a power

```
Math.pow(2, 10)    → 210 = 1024  
Math.pow(2, -1)   → 2-1 = 0.5  
Math.pow(9, 0.5)  → 90.5 = 3
```

`Math.random` generates a random number in the range [0...1)

- note: this function has no inputs; it returns a different number each call

```
Math.random()     → 0.33008525626748814  
Math.random()     → 0.213335955823927  
Math.random()     → 0.8975001737758223  
.  
.  
.
```

33

Errors and Debugging

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 write statements*

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

34