

CSC 121 Computers and Scientific Thinking

Fall 2005

Interactive 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)

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)

2

Simple Dynamic Page

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

```

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("Hello " + firstName + ", welcome to my Web page.");
15.   </script>
16.
17.   <p>
18.     Whatever else you want to appear in your Web page...
19.   </p>
20. </body>
21. </html>

```

3

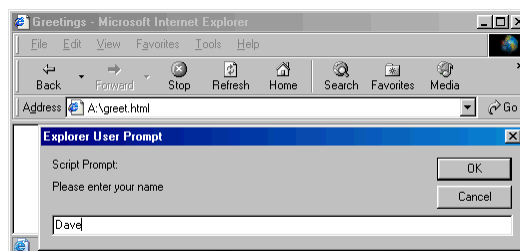
Assignment Statement

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



4

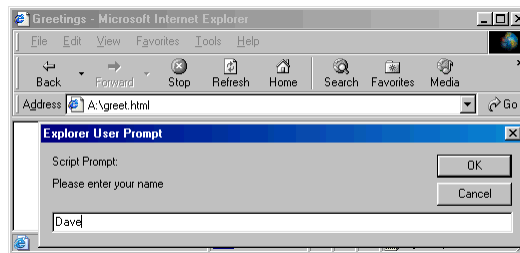
Assignment Statement (cont.)



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



5

Write Statement

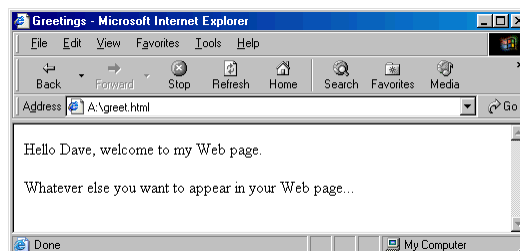


when a write statement is executed by the browser

- 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("Hello " + firstName + ", welcome to my Web page.");
```

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



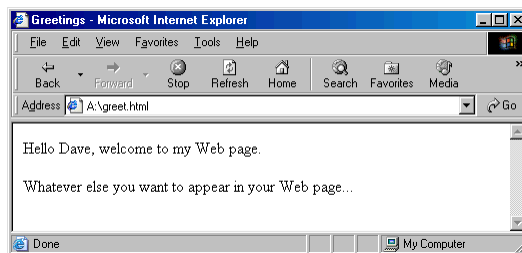
6

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



7

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

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

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

which would be displayed by the browser as

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

8

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**

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	delete	function	null	throw
boolean	do	goto	package	throws
break	double	if	private	transient
byte	else	implements	protected	true
case	enum	import	public	try
catch	export	in	return	typeof
char	extends	instanceof	short	var
class	false	int	static	void
const	final	interface	super	volatile
continue	finally	long	switch	while
debugger	float	native	synchronized	with
default	for	new	this	

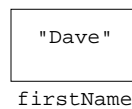
Variabl es & Memory Cel l s

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

11

Prompts wi th Defaul ts

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");
```

12

Ol d MacDonal d

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("<ps>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.</ps>");
24. </script>
25. </body>
26. </html>

```

13

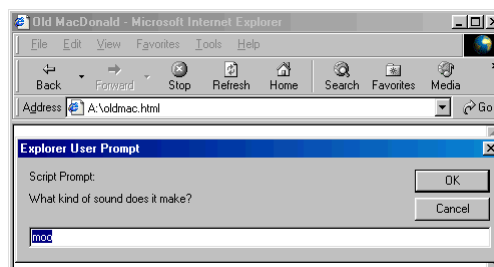
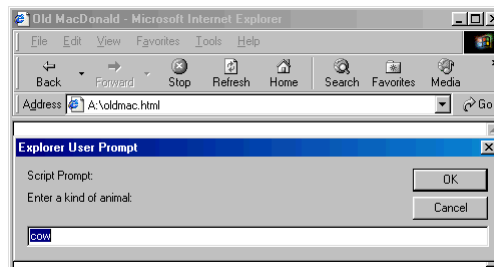
Ol d MacDonal d (cont.)

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



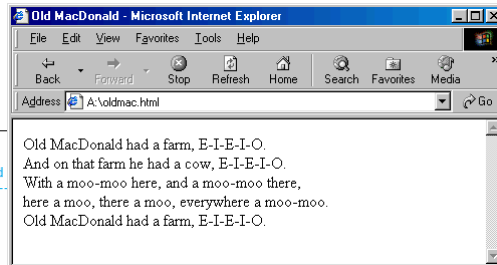
14

Old MacDonald (cont.)

```

1. <html>
2. <!-- oldmac.html
3. <!-- Web page that displays a verse of Old
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("<sp>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>

```



15

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

16

Variabl es and Expressi ons

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

<code>x = 24;</code>	<input type="text" value="24"/>	
	x	
<code>y = (100 * 10) + 24;</code>	<input type="text" value="24"/>	<input type="text" value="1024"/>
	x	y
<code>x = y - 1;</code>	<input type="text" value="1023"/>	<input type="text" value="1024"/>
	x	x

17

Mi xed 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"
```

18

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

19

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 → 5, -2.4 → 2.4, 17 → 17, ...

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

- 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*

20

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

21

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

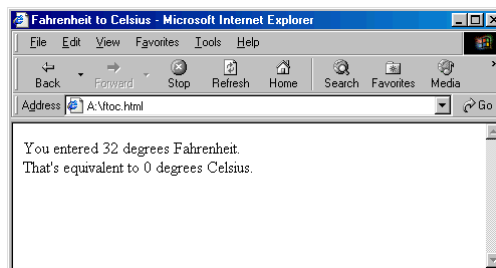
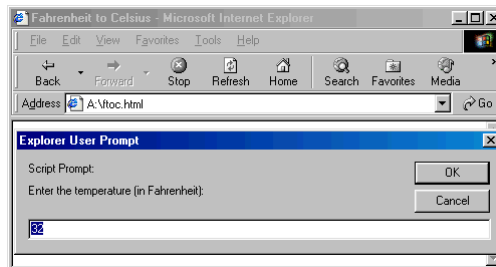
```
1. <html>
2. <!-- ftoc.html                               Dave Reed -->
3. <!--                                           -->
4. <!-- Converts a temperature from Fahrenheit to Celsius. -->
5. <!------->
6.
7. <head>
8.   <title>Fahrenheit to Celsius</title>
9. </head>
10.
11. <body>
12.   <h3 style="text-align:center">Temperature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

22

Conversion Page



note that the prompt has a default value of 32



23

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>
```

24

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 square root of a number
e.g., `Math.max(3.2, 1.8) → 3.2`

```

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

```

25

Other Useful Functions

`Math.pow` raises a number to a power

`Math.pow(2, 10)` → $2^{10} = 1024$
`Math.pow(2, -1)` → $2^{-1} = 0.5$
`Math.pow(9, 0.5)` → $9^{0.5} = 3$

`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
 .
 .
 .

26

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