

CSC 121
Computers and Scientific
Thinking

Fall 2005

Abstraction and
User-Defined Functions

1

Abstraction



abstraction is the process of ignoring minutiae and focusing on the big picture

- in modern life, we are constantly confronted with complexity
- we don't necessarily know how it works, but we know how to use it

e.g., how does a TV work? a car? a computer?

we survive in the face of complexity by abstracting away details

- to use a TV/car/computer, it's not important to understand the inner workings
- we ignore unimportant details and focus on those features relevant to using it
- e.g., TV has power switch, volume control, channel changer, ...

JavaScript functions (like `Math.sqrt`) provide computational abstraction

- a function encapsulates some computation & hides the details from the user
- the user only needs to know how to call the function, not how it works
- in order to create our own abstractions, we must learn how functions work

2

User-defined Functions

JavaScript's predefined functions represent a collection of useful, general-purpose abstractions

- the programmer can add additional abstractions via user-defined functions
- once defined, a user-defined function can be used the same way as a predefined function
- e.g., consider converting a temperature from Fahrenheit to Celsius

```
tempInCelsius = (5/9) * (tempInFahr - 32);
```

this expression & assignment could be used whenever we want to convert

- requires remembering the formula every time

instead, we could define a function to encapsulate the calculation

```
function FahrToCelsius(tempInFahr)
// Assumes: tempInFahr is a temperature in Fahrenheit
// Returns: the equivalent temperature in Celsius
{
    return (5/9) * (tempInFahr - 32);
}
```

could then call that function whenever a conversion was needed

```
freezing = FahrToCelsius(32);           current = FahrToCelsius(78);
```

3

Dissecting the Function

```
function FahrToCelsius(tempInFahr)
// Assumes: tempInFahr is a temperature in Fahrenheit
// Returns: the equivalent temperature in Celsius
{
    return (5/9) * (tempInFahr - 32);
}
```

the 1st line specifies that we are defining a function named `FahrToCelsius` that takes one input

- the variable name in parentheses is known as a *parameter*
- when the function is called, the provided input is assigned to the parameter
e.g., for the call `FahrToCelsius(32)`, the value 32 would be assigned to `tempInFahr` for the calculation

the 2nd and 3rd lines are *comments* that describe the behavior of the function

- anything to the right of `//` is ignored by the browser (so not strictly required)
- but, you should ALWAYS have comments in any user-defined function to make the code easier to read and understand

the actual code that carries out the function's computation is enclosed in `{ }`

- here, the function definition contains only one statement, but could be more
- a `return` statement specifies the value that should be returned by the function (i.e., it's output)

4

General Function Form

in general, the form of a user-defined function is as follows:

```
function FUNCTION_NAME(PARAMETER_1, PARAMETER_2,..., PARAMETER_n)
// Assumes: DESCRIPTION OF ASSUMPTIONS MADE ABOUT PARAMETERS
// Returns: DESCRIPTION OF VALUE RETURNED BY FUNCTION
{
    STATEMENTS_TO_PERFORM_(AND_RETURN)_THE_DESIRED_COMPUTATION;
}
```

- function names follow the same rules as variables: consist of letters, digits, and underscores and must start with a letter
- note that there may be:
 - more than one parameter, separated by commas
 - no parameters at all, in which case the function name is followed by ()
 - more than one statement inside the curly braces (more later)

```
function difference(num1, num2)
// Assumes: num1 and num2 are numbers
// Returns: the absolute difference between num1 and num2
{
    return Math.abs(num1 - num2);
}
```

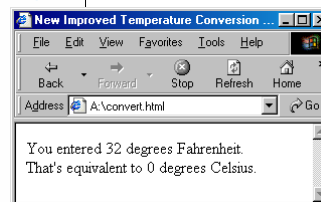
5

Temperature Conversion Page

to include a user-defined function in a page, the simplest way is to include its definition in SCRIPT tags in the HEAD

- once the function is defined in the HEAD, it can be called just like any other function

```
1. <html>
2. <!-- convert.html                                Dave Reed -->
3. <!-- This page converts a temperature from Fahrenheit to Celsius. -->
4. <!--
5.
6. <head>
7. <title> New Improved Temperature Conversion </title>
8. <script type="text/javascript">
9.     function FahrToCelsius(tempInFahr)
10.         // Assumes: tempInFahr is a temperature in Fahrenheit
11.         // Returns: the equivalent temperature in Celsius
12.         {
13.             return (5/9) * (tempInFahr - 32);
14.         }
15.     </script>
16. </head>
17.
18. <body>
19.     <script type="text/javascript">
20.         tempF = prompt("Enter the temperature (in Fahrenheit):", "32");
21.         tempF = parseFloat(tempF);
22.
23.         tempC = FahrToCelsius(tempF);
24.
25.         document.write("You entered " + tempF + " degrees Fahrenheit.<br />");
26.         document.write("That's equivalent to " + tempC +
27.             " degrees Celsius.");
28.     </script>
29. </body>
30. </html>
```



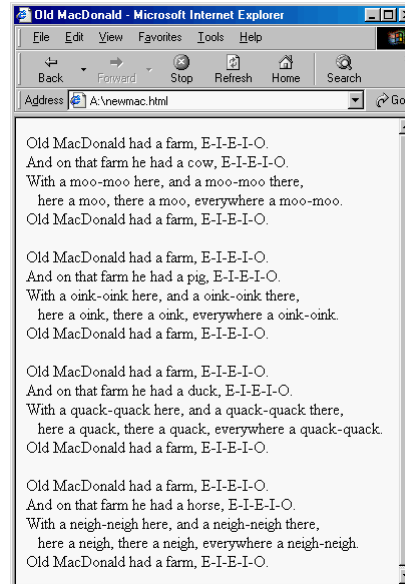
note: if a page requires more than one user-defined function, they can all be enclosed in the same pair of SCRIPT tags

6

Old MacDonald

each call to the OldMacVerse function displays the verse with the specified animal & sound

- to rearrange the verses, simply rearrange the calls in the BODY
- to add a new verse, simply add a new call in the BODY
- to make a change to all verses, e.g., change indentation, simply make the change once in the function definition



9

Multiple Inputs

if a function has more than one input,

- parameters in the function definition are separated by commas
- input values in the function call are separated by commas
- values are matched to parameters by order
 - 1st input value in the function call is assigned to the 1st parameter in the function
 - 2nd input value in the function call is assigned to the 2nd parameter in the function
 - ...

```
function OldMacVerse( animal, sound )
// Assumes: animal and sound are strings
// Results: displays corresponding Old MacDonald verse
{
    ...
}

-----
OldMacVerse( "cow", "moo" );
OldMacVerse( "moo", "cow" );
```

10

Designing Functions

functions do not add any computational power to the language

- a function definition simply encapsulates other statements

still, the capacity to define and use functions is key to solving complex problems, as well as to developing reusable code

- encapsulating repetitive tasks can shorten and simplify code
- functions provide units of computational abstraction – user can ignore details
- functions are self-contained, so can easily be reused in different applications

when is it worthwhile to define a function?

- if a particular computation is complex—meaning that it requires extra variables and/or multiple lines to define
- if you have to perform a particular computation repeatedly within a page

when defining a function, you must identify

- the inputs
- the computation to be performed using those inputs
- the output

11

Parameters and Locals

parameters play an important role in functions

- they facilitate the creation of generalized computations
- i.e., the function defines a formula, but certain values within the formula can differ each time the function is called

Technically, a parameter is a *local variable*, meaning it exists only inside its particular function

- when the function is called, memory cells are allocated for the parameters and each input from the call is assigned to its corresponding parameter
- once a parameter has been assigned a value, you can refer to that parameter within the function just as you would any other variable
- when the function terminates, the parameters "go away," and their associated memory cells are freed

variables that appear in the BODY of the page are *global variables*, meaning they exist and can be accessed by JavaScript code anywhere in the page

- note: it is possible to use the same name to refer to a local variable and a global variable
 - within the function, the local variable is accessible
 - outside that function, the global variable is accessible

12

Local vs. Global

```

1. <html>
2. <!-- testmac.html Dave Reed -->
3. <!-- This page displays two verses of OldMacDonald. -->
4. <!------->
5.
6. <head>
7. <title> Old MacDonald Test </title>
8. <script type="text/javascript">
9.   function OldMacVerse(animal, sound)
10.    // Assumes: animal and sound are strings
11.    // Results: displays corresponding Old MacDonald verse
12.    {
13.      document.write("<p>Old MacDonald had a farm, E-I-E-I-O.<br />");
14.      document.write("And on that farm he had a " + animal +
15.        " E-I-E-I-O.<br />");
16.      document.write("With a " + sound + "-" + sound +
17.        " here, and a " + sound + "-" + sound +
18.        " there.<br />");
19.      document.write("&nbsp;&nbsp;&nbsp; here a " + sound + ", there a " +
20.        sound + ", everywhere a " + sound + "-" +
21.        sound + "<br />");
22.      document.write("Old MacDonald had a farm, E-I-E-I-O.</p>");
23.    }
24.  </script>
25. </head>
26.
27. <body>
28. <script type="text/javascript">
29.   animal = prompt("Enter the name of an animal:", "");
30.   sound = prompt("What sound does it make?", "");
31.
32.   OldMacVerse(animal, sound);
33.   OldMacVerse("duck", "quack");
34. </script>
35. </body>
36. </html>

```

here, the variable names
animal and sound are

- used for parameters in the function definition
(local variables)
- used for variables in the BODY
(global variables)

we can think of these as completely separate variables, identifiable via a subscript

- animal_{OldMacVerse} and sound_{OldMacVerse} are used in the function
- animal_{BODY} and sound_{BODY} are used in the BODY

13

Declaring Local Variables

we have seen that variables are useful for storing intermediate steps in a complex computation

- within a user-defined function, the programmer is free to create new variables and use them in specifying the function's computation
- however, by default, new variables used in a function are global
 - but what if the same variable name is already used elsewhere?

to avoid name conflicts, the programmer can *declare* variables to be local

- a variable declaration is a statement that lists all local variables to be used in a function (usually the first statement in a function)
- general form:

```
var LOCAL_1, LOCAL_2, . . . , LOCAL_n;
```

```

function IncomeTax(income, itemized)
// Assumes: income >= 0, itemized >= 0
// Returns: flat tax (13%) due after deductions
{
  var deduction, taxableIncome, totalTax;

  deduction = Math.max(itemized, 4150);
  taxableIncome = Math.max(income - deduction, 0);
  totalTax = 0.13*taxableIncome

  return totalTax;
}

```

14

random.js

general-purpose functions can be grouped together in a *library*

- a library is a text file that contains one or more function definitions
- once the functions are defined in the library, that library can be loaded into pages as needed

e.g., the `random.js` library contains useful functions for generating random values

| Function | Inputs | Output |
|--------------------------|--|--|
| <code>RandomNum</code> | Two numbers (low and high limits of a range) e.g., <code>RandomNum (2, 4.5)</code> | A random number from the range low (inclusive) to high (exclusive) |
| <code>RandomInt</code> | Two integers (low and high limits of a range) e.g., <code>RandomInt(1, 10)</code> | A random integer from the range low to high (both inclusive) |
| <code>RandomChar</code> | A nonempty string e.g., <code>RandomChar("abcd")</code> | A random character taken from the string |
| <code>RandomOneOf</code> | A list of options (in brackets, separated by commas) e.g., <code>RandomOneOf(["yes", "no"])</code> | A random value taken from the list of options |

to load a library of functions in a page, use a special pair of SCRIPT tags

```
<script type="text/javascript"
src="URL_OR_LOCAL_FILENAME">
</script>
```

15

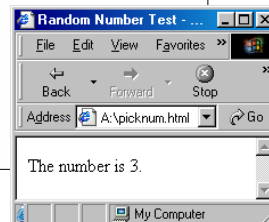
Using random.js

in the page below, the `random.js` library is accessed via the Web

- you can download the file and store it on your own machine
- then, simply specify the file name in the SRC attribute (the default is that the file is in the same folder as the Web page that includes it)

```

1. <html>
2. <!-- picknum.html                                     Dave Reed -->
3. <!-- This page uses RandomInt to pick and display a random number. -->
4. <!------->
5.
6. <head>
7. <title> Random Number Test </title>
8. <script type="text/javascript"
9.   src="http://www.prenhall.com/reed/random.js">
10. </script>
11. </head>
12.
13. <body>
14. <script type="text/javascript">
15.   x = RandomInt(1, 3);
16.   document.write("The number is " + x + ".");
17. </script>
18. </body>
19. </html>
```



16

Errors to Avoid

When beginning programmers attempt to load a JavaScript code library, errors of two types commonly occur:

1. if the SCRIPT tags are malformed or the name/address of the library is incorrect, the library will fail to load
 - this will not cause an error in itself, but any subsequent attempt to call a function from the library will produce
 - "Error: Object Expected" (using Internet Explorer)
 - or
 - "Error: XXX is not a function" (using Navigator), where XXX represents the typed function name
2. when you use the SRC attribute in a pair of SCRIPT tags to load a code library, you cannot place additional JavaScript code between the tags
 - think of the SRC attribute as causing the contents of the library to be inserted between the tags, overwriting any other code that was erroneously placed there

```
<script type="text/javascript" src="FILENAME">
  ANYTHING PLACED IN HERE WILL BE IGNORED
</script>
```
 - if you want additional JavaScript code or another library, you must use another pair of SCRIPT tags