

# CSC 121

## Computers and Scientific Thinking

David Reed  
Creighton University

---

### Chapter 9

#### 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
- Chapter 7 introduced simple user-defined functions
  - could encapsulate the statements associated with a button, call the function as needed

2

## General Function Form

to write general-purpose functions, we can extend definitions to include:  
1) parameters, 2) local variables, and 3) return statements

```
function FUNCTION_NAME(PARAMETER_1, PARAMETER_2, ..., PARAMETER_n)
// Assumes: DESCRIPTION OF ASSUMPTIONS MADE ABOUT PARAMETERS
// Returns: DESCRIPTION OF VALUE RETURNED BY FUNCTION
{
  var LOCAL_1, LOCAL_2, ..., LOCAL_n;           // optional
  STATEMENTS_TO_PERFORM_THE_DESIRED_COMPUTATION;
  return OUTPUT_VALUE;                          // optional
}
```

- *parameters* are variables that correspond to the function's inputs (if any)
  - parameters appear in the parentheses, separated by commas
- *local variables* are temporary variables that are limited to that function only
  - if require some temporary storage in performing calculations, then declare local variables using the keyword `var`, separated by commas
  - a local variable exists only while the function executes, so no potential conflicts with other functions
- a *return statement* is a statement that specifies an output value
  - consists of the keyword `return` followed by a variable or expression

3

## Functions with Inputs

most of the predefined function we have considered expect at least on input

e.g., `Math.sqrt` takes a number as input, and returns its square root as output

`Math.sqrt(9) → 3`

e.g., `Math.max` takes two numbers as inputs, and returns the maximum as output

`Math.max(7, 3) → 7`

in English, the word *parameter* refers to some aspect of a system that can be varied in order to control its behavior

- in JavaScript, a parameter is a variable (declared inside the function's parentheses) whose value is automatically initialized to the corresponding input value when the function is called
- parameters allow the same function to perform different (but related) tasks when called with different input values

```
function ChangeImage(imgSource)
// Assumes: imgSource is a file containing an image
// Results: the source of faceImg is set to imgSource
{
  document.getElementById('faceImg').src = imgSource;
}
```

the call `ChangeImage("happy.gif")` will assign the input "happy.gif" to the parameter `imgSource`, resulting in the image being assigned to that file

the call `ChangeImage("sad.gif")` will assign the input "sad.gif" to the parameter `imgSource`, resulting in the image being assigned to that file

4



## Multiple Inputs

if a function has more than one input,

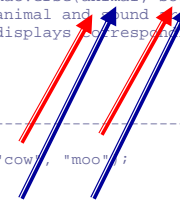
- parameters in the function definition are separated by commas
- input values in the function call are separated by commas
  
- values are matched to parameters by order
  - 1<sup>st</sup> input value in the function call is assigned to the 1<sup>st</sup> parameter in the function
  - 2<sup>nd</sup> input value in the function call is assigned to the 2<sup>nd</sup> 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");
```



7

## 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

by default, variables other than parameters are considered *global*, 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

8

## 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<sub>OldMacVerse</sub> and sound<sub>OldMacVerse</sub> are used in the function
- animal<sub>BODY</sub> and sound<sub>BODY</sub> are used in the BODY

9

## 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 should *declare* temporary 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
// Results: displays 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

    alert("You owe $" + totalTax);
}

```

since these variables are declared as local, they will not affect (or be affected by) any variables with the same names elsewhere in the page

10

# Functions with Return

displaying results using `document.write` or `alert` is OK for some functions

- for full generality, we need to be able to return an output value, which can then be used in other computations

```
e.g.,    number = Math.sqrt(9);
        amountOwed = IncomeTax(38000, 6500);
```

a return statement can be added to a function to specify its output value

- when the return statement is reached, the variable or expression is evaluated and its value is returned as the function's output
- general form: `return OUTPUT_VALUE;`

```
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;
}
```

since this function returns the value, it can be used in other computations, e.g., calculate amount owed in 4 payments:

```
payment =
IncomeTax(38000, 6500)/4;
```

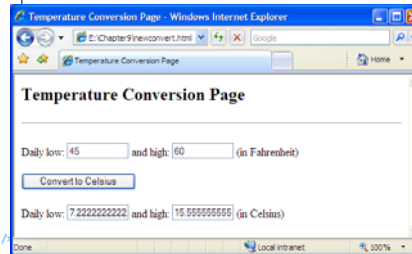
11

```
1. <html>
2. <!-- newconvert.html           Dave Reed -->
3. <!-- This page converts temperatures from Fahrenheit to Celsius. -->
4. <!-- ----- -->
5.
6. <head>
7. <title>Temperature Conversion Page</title>
8.
9. <script type="text/javascript">
10. function FahrToCelsius(tempInFahr)
11. // Assumes: tempInFahr is a temperature in Fahrenheit
12. // Returns: the equivalent temperature in Celsius
13. {
14.     var tempInCelsius;
15.     tempInCelsius = (5/9) * (tempInFahr - 32);
16.     return tempInCelsius;
17. }
18.
19. function ConvertFtoC()
20. // Assumes: lowFahrBox and highFahrBox contain degrees Fahrenheit
21. // Results: assigns lowCelsiusBox and highCelsiusBox the equivalent temperatures
22. {
23.     var lowTempInF, highTempInF, lowTempInC, highTempInC;
24.
25.     lowTempInF = parseFloat(document.getElementById('lowFahrBox').value);
26.     lowTempInC = FahrToCelsius(lowTempInF);
27.     highTempInF = parseFloat(document.getElementById('highFahrBox').value);
28.     highTempInC = FahrToCelsius(highTempInF);
29.
30.     lowTempInC = FahrToCelsius(lowTempInF);
31.     highTempInC = FahrToCelsius(highTempInF);
32.
33.     document.getElementById('lowCelsiusBox').value = lowTempInC;
34.     document.getElementById('highCelsiusBox').value = highTempInC;
35. }
36. </script>
37. </head>
38.
39. <body>
40. <h2>Temperature Conversion Page</h2>
41. <hr />
42. <p>
43.     Daily low: <input type="text" id="lowFahrBox" size="10" value="" />
44.     and high: <input type="text" id="highFahrBox" size="10" value="" />
45.     (in Fahrenheit)
46. </p>
47. <p>
48.     <input type="button" value="Convert to Celsius" onclick="ConvertFtoC();" />
49. </p>
50.
51.     Daily low: <input type="text" id="lowCelsiusBox" size="10" value="" />
52.     and high: <input type="text" id="highCelsiusBox" size="10" value="" />
53.     (in Celsius)
54. </p>
55. </body>
56. </html>
```

## Newconvert Page

if the same computation must be done repeatedly, a function can greatly simplify the page

- here, `FahrToCelsius` is called twice to convert two different temperatures



12

## 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

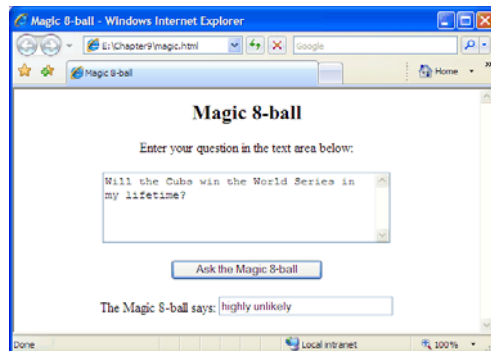
- the inputs
- the computation to be performed using those inputs
- the output

13

## Design Example

consider the task of designing an online Magic 8-ball® (Mattell, Inc.)

- must be able to ask a yes/no type question
- receive an answer (presumably, at random)



could use:

- a text area for entering the question (which could be several lines long)
- a text box for displaying the answer (which should be short)
- a button for initiating the action – which involves calling a function to process the question, select an answer, and display it in the text box

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
RandomNum	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)
RandomInt	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)
RandomChar	A nonempty string, e.g., <code>RandomChar("abcd")</code>	A random character taken from the string
RandomOneOf	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

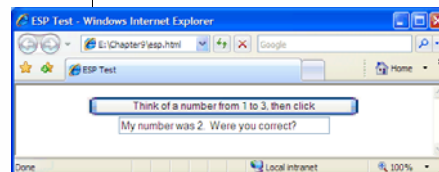
## Using random.js

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. <!-- esp.html Dave Reed -->
3. <!-- This page uses RandomInt to pick and display a random number. -->
4. <!-- ----- -->
5.
6. <head>
7. <title> ESP Test </title>
8. <script type="text/javascript"
9.   src="http://dave-reed.com/book/random.js">
10. </script>
11. <script type="text/javascript">
12.   function PickNumber()
13.   // Results: displays a random number in messageBox
14.   {
15.     var number;
16.
17.     number = RandomInt(1, 3);
18.
19.     document.getElementById("messageBox").value =
20.       "My number was " + number + ". Were you correct?";
21.   }
22. </script>
23. </head>
24.
25. <body>
26. <p style="text-align:center">
27.   <input type="button" value="Think of a number from 1 to 3, then click"
28.     onclick="PickNumber();" />
29. <br />
30.   <input type="text" id="messageBox" size="40" />
31. </p>
32. </body>
33. </html>
```



note: the `RandomOneOf` function from `random.js` would similarly be useful for the Magic 8-ball page (in selecting possible answers at random)

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 Firefox), where XXX is the entered 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