FIT100 Concepts of Algorithmic Thinking

Functions and Methods:

Good Things Come in Small Packages

D.A. Clements

Functions

FIT100

- A function is a package for an algorithm; once written, it can be used over and over.
- Professional developers have their own script libraries they bring to each job they work on.

Anatomy of a Function

FIT100

- Functions are packages for algorithms
- Three parts
  - Name
  - Parameters
  - Definition
- These parts are the *function declaration*

Pick a Name

FIT100

- Name is the identifier for the function
  - Commonly used to describe what the function does
- Function declaration form:

function <name> (<*parameter list*>)
{
  <*statement list*>
}

Parameters

FIT100

- Parameters are the input values the function will compute on
- Parameters are given names
- If more than one, they are separated by commas
- Parameter names follow usual rules for identifiers

```
function convertC2F ( tempInC )
{
   <statement list>
}
```

Definition

FIT100

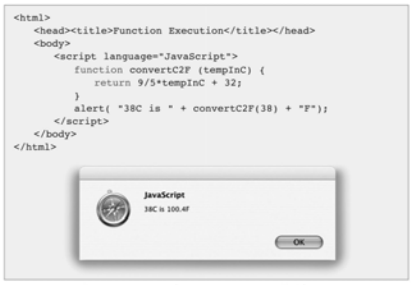
- Definition is the algorithm written in a programming language
- To say what the answer/result is, JavaScript uses the statement: **return** <expression>

```
function convertC2F ( tempInC )
{
   return 9.0 / 5.0 * tempInC + 32;
}
```

- "Calling" a function is to run or execute it
  - Write the function's name, put the input values (*arguments*) in the parentheses

```
convertC2F( 38 );
```

FIT1



```
<html>
  <head><title>Function Execution</title></head>
  <body>
    <script language="JavaScript">
      function convertC2F (tempInC) {
        return 9/5*tempInC + 32;
      }
      alert( "38C is " + convertC2F(38) + "F");
    </script>
  </body>
</html>
```

JavaScript
38C is 100.4F

Figure 20.1. *The* `convertC2F()` *function in JavaScript called from an* `alert()`.

FIT100

Declaration vs. Call

- A function's declaration is different from its call (use)
- Functions are declared once
- Functions can be called as many times as their answers are needed

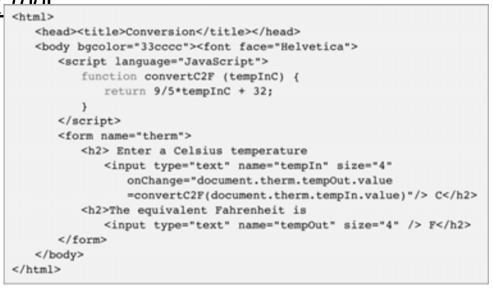
Declare once.
Call many times.

FIT100

Forms and Functions

- Construct a web page in which to run a function
- Event handlers usually call functions
- The value in an input window, or text box, can be used as an *argument* to a function

FIT100



```
<html>
  <head><title>Conversion</title></head>
  <body bgcolor="33cccc"><font face="Helvetica">
    <script language="JavaScript">
      function convertC2F (tempInC) {
        return 9/5*tempInC + 32;
      }
    </script>
    <form name="therm">
      <h2> Enter a Celsius temperature
        <input type="text" name="tempIn" size="4"
          onChange="document.therm.tempOut.value
          =convertC2F(document.therm.tempIn.value)"/> C</h2>
      <h2>The equivalent Fahrenheit is
        <input type="text" name="tempOut" size="4" /> F</h2>
    </form>
  </body>
</html>
```

Figure 20.2 The HTML/JavaScript source for the Conversion application.

FIT100

Calling to Customize a Page

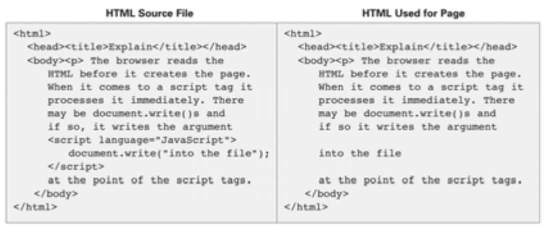
- Three ways to get the result of a function call to print on the monitor
  - 1) Before the page is created
    - For example, with the alert() call (slide 7)
  - 2) Interactively after the page is displayed
    - For example, the Conversion application (slide 10)
  - 3) While the page is being loaded
    - For example, document.write() built-in function
- Calling functions while the browser is

FIT100

Calling to Customize a Page

- How a browser builds a page:
  - ∗ Reads through HTML file, figuring out all tags and preparing to build page
  - ∗ Removes JavaScript tags and all text between them, and does whatever the JavaScript tells it to do
    - It could tell the browser to put some text back in the file, as in *document.write()*

FIT100



HTML Source File

```
<html>
  <head><title>Explain</title></head>
  <body><p> The browser reads the
     HTML before it creates the page.
     When it comes to a script tag it
     processes it immediately. There
     may be document.write()s and
     if so, it writes the argument
     <script language="JavaScript">
        document.write("into the file");
     </script>
     at the point of the script tags.
   </body>
</html>
```

HTML Used for Page

```
<html>
  <head><title>Explain</title></head>
  <body><p> The browser reads the
     HTML before it creates the page.
     When it comes to a script tag it
     processes it immediately. There
     may be document.write()s and
     if so it writes the argument

     into the file

     at the point of the script tags.
   </body>
</html>
```

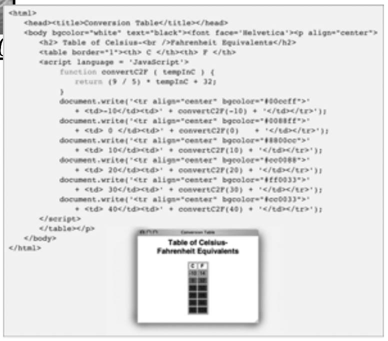
Figure 20.3. *An HTML source file containing a JavaScript* `document.write()`, *and the HTML text used by the browser to create the page.*

FIT100

Calling to Customize a Page

- Suppose we want a table of temperature conversions for a web page with a column for Celsius and a column for Fahrenheit
- Put *document.write()* within the <script> </script> tags to create the rows of the table
- Put Celsius values in first column cells, second column cells can call

FIT100



```
<html>
   <head><title>Conversion Table</title></head>
   <body bgcolor="white" text="black"><font face='Helvetica'><p align="center">
      <h2> Table of Celsius-<br />Fahrenheit Equivalents</h2>
      <table border="1"><th> C </th><th> F </th>
      <script language = 'JavaScript'>
          function convertC2F ( tempInC ) {
             return (9 / 5) * tempInC + 32;
          }
          document.write('<tr align="center" bgcolor="#00ccff">'
             + <td>-10</td><td>' + convertC2F(-10) + '</td></tr>');
          document.write('<tr align="center" bgcolor="#0088ff">'
             + <td> 0 </td><td>' + convertC2F(0)    + '</td></tr>');
          document.write('<tr align="center" bgcolor="#8800cc">'
             + <td> 10</td><td>' + convertC2F(10) + '</td></tr>');
          document.write('<tr align="center" bgcolor="#cc0088">'
             + <td> 20</td><td>' + convertC2F(20) + '</td></tr>');
          document.write('<tr align="center" bgcolor="#ff0033">'
             + <td> 30</td><td>' + convertC2F(30) + '</td></tr>');
          document.write('<tr align="center" bgcolor="#cc0033">'
             + <td> 40</td><td>' + convertC2F(40) + '</td></tr>');
      </script>
      </table></p>
   </body>
</html>
```

Conversion Table
Table of Celsius-
Fahrenheit Equivalents
C
F
-10
14

Figure 20.4 Source text and image for the Conversion Table computation.

FIT100

Writing Functions, Using Functions

- Flipping Electronic Coins
  - A coin flip is an unpredictable event whose two outcomes are "equally probable"
  - Computers can generate pseudo-random numbers
    - An algorithm that produces a sequence of numbers that passes the statistical tests for randomness
    - We can just call them random numbers

FIT100

Flipping Electronic Coins

- *Math.random()* is JavaScript's built-in function for generating random numbers
  - Each time it is called, it generates a random number between 0 (inclusive) and 1 (exclusive)
- A function to flip electronic coins:

```
function coinFlip() {
  return Math.round( Math.random() );
```

FIT100

Flipping Electronic Coins (cont'd)

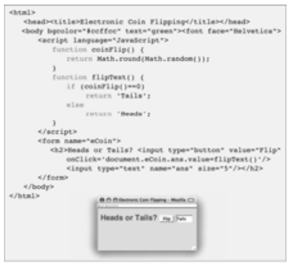
- coinFlip() returns with equal probability a 0 or a 1
- Next improvement is to return the text Heads or Tails rather than numbers

```
function flipText() {
  if ( coinFlip() == 0 )
     return 'Tails';
   else
     return 'Heads';
```

FIT100

Flipping Electronic Coins (cont'd)

- Even more useful to give outcome in response to pressing a button on a web page



```
<html>
  <head><title>Electronic Coin Flipping</title></head>
  <body bgcolor="#ccffcc" text="green"><font face="Helvetica">
    <script language="JavaScript">
      function coinFlip() {
        return Math.round(Math.random());
      }
      function flipText() {
        if (coinFlip()==0)
          return 'Tails';
        else
          return 'Heads';
      }
    </script>
    <form name="eCoin">
      <h2>Heads or Tails? <input type="button" value="Flip"
        onClick='document.eCoin.ans.value=flipText()'/>
        <input type="text" name="ans" size="5"/></h2>
    </form>
  </body>
</html>
```

Figure 20.5 The JavaScript and image for the Electronic Coin-Flipping page.

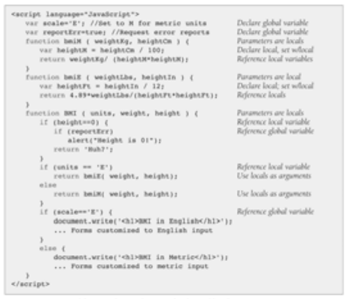
FIT100

Scoping: When to Use Names

- Scope of a name defines how "far" from its declarations it can be used
- General rule for scoping:
  - Variable names declared in a function can be used only within that function (they are *local to the function*)
    - Parameters are considered local variables
  - Variable names declared outside any function can be used throughout the

FIT100

An Annotated Example



```
<script language="JavaScript">
  var scale='E'; //Set to M for metric units          Declare global variable
  var reportErr=true; //Request error reports          Declare global variable
  function bmiM ( weightKg, heightCm ) {               Parameters are locals
    var heightM = heightCm / 100;                      Declare local, set w/local
    return weightKg/ (heightM*heightM);                Reference local variables
  }
  function bmiE ( weightLbs, heightIn ) {              Parameters are local
    var heightFt = heightIn / 12;                      Declare local; set w/local
    return 4.89*weightLbs/(heightFt*heightFt);         Reference locals
  }
  function BMI ( units, weight, height ) {             Parameters are locals
    if (height==0) {                                   Reference local variable
      if (reportErr)                                   Reference global variable
        alert("Height is 0!");
      return 'Huh?';
    }
    if (units == 'E')                                  Reference local variable
      return bmiE( weight, height);                    Use locals as arguments
    else
      return bmiM( weight, height);                    Use locals as arguments
    }
    if (scale=='E') {                                  Reference global variable
      document.write('<h1>BMI in English</h1>');
      ... Forms customized to English input
    }
    else {
      document.write('<h1>BMI in Metric</h1>');
      ... Forms customized to metric input
  }
</script>
```

Figure 20.7. Annotated functions showing the scope of each variable reference.

FIT100

Scoping

- *Local variables* come into existence when a function begins, and when it ends, they vanish
- *Global variables* are around all the time
- If information must be saved from one function call to the next, it must be in a *global variable*

FIT100

Global/Local Scope Interaction

- Where a global variable and a local variable have the same name:

```
var y=0;
…
function tricky (x) {
  var y;
   y = x;
  …
}
```

FIT100

Global/Local Scope Interaction (cont'd)

- y is globally declared and can be referenced anywhere
- y is also declared as a local variable in the tricky() function
- They are two different variables
- Which y is assigned the parameter x?
  - The local y, because it is declared in the function's scope, making it the

FIT100

Recap: Two Reasons to Write Functions

- Packaging algorithms into functions
  - Reuse
    - Building blocks of future programming
    - Make them as general as possible
  - Complexity management
    - Help us keep our sanity while we're solving problems

FIT100

Built-in JavaScript functions

METHODS

FIT100

Methods

- Methods are built-in JavaScript for commonly used code:
  - window.open()
  - alert();
  - prompt();
  - confirm();
  - document.write();

FIT100

Methods—Write your own

- You can even write your own methods and "attach" them to objects

FIT100

End papers…

Why is programming fun?

- Third is the fascination of fashioning complex puzzle-like objects of interlocking moving parts and watching them work in subtle cycles, playing out the consequences of principles built in from the beginning. The programmed computer has all the fascination of the pinball machine or the jukebox mechanism, carried to the ultimate.