



College of Information Studies

University of Maryland Hornbake Library Building College Park, MD 20742-4345

Structured Programming

Week 3

INFM 603

Muddiest Points

- Emergent behavior of the Web

- HTML class attribute

```
<head>...  
<style type="text/css">  
    p.style1 { font-family:arial; color:blue}  
    p.style2 { font-family:serif; color:red}  
</style>  
</head>  
<body>  
<p class="style1">...</p>  
<p class="style2">...</p>
```

- The details of JavaScript

Programming in Four Parts

➤ Structured Programming

- Modular Programming
- Data Structures
- Object-Oriented Programming

Machine Language

- **Everything** is a binary number
 - Operations

Data

00001000 00010101 01010110

00001000

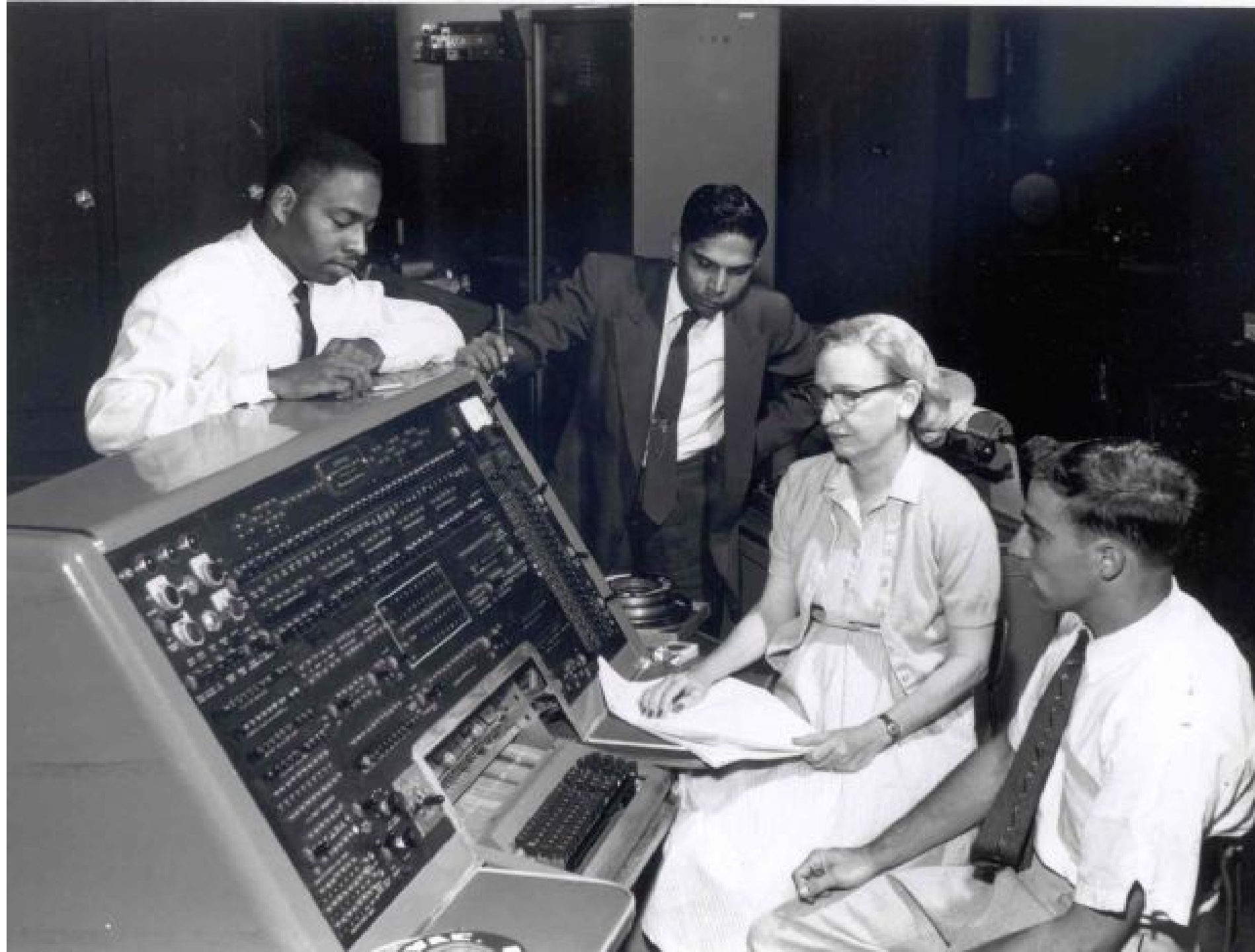
ADD

00010101

number to be added (21)

01010110

memory location to add it to (86)



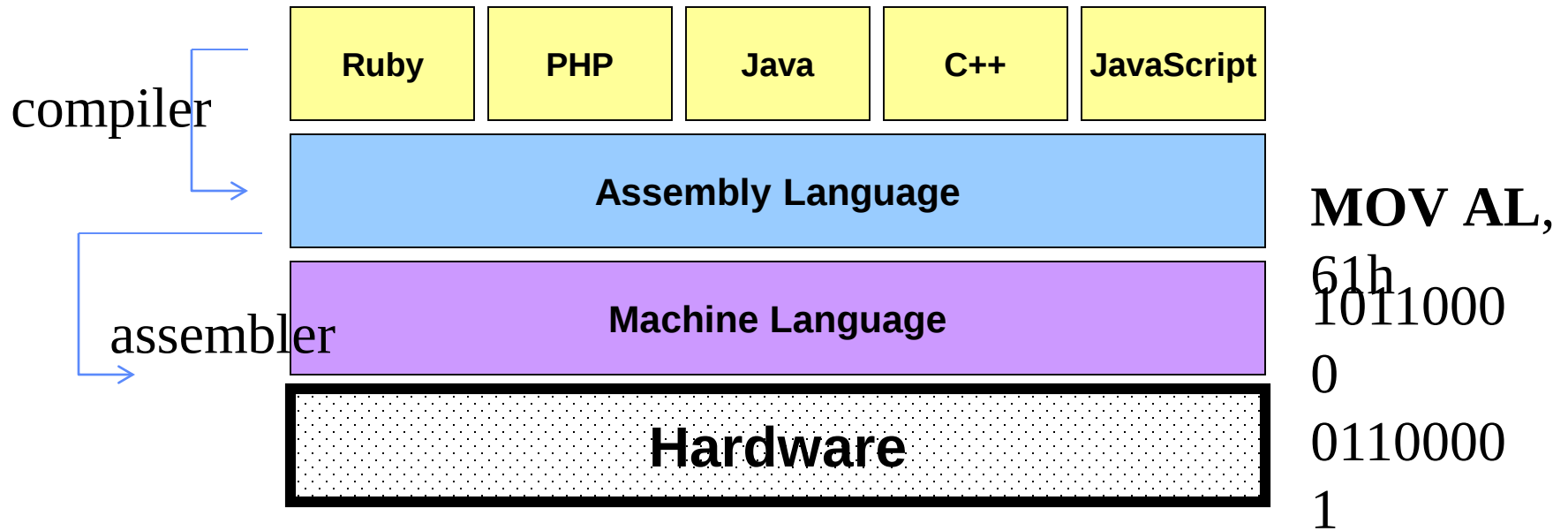
Assembly Language

- **Symbolic** instructions and addresses
 - Symbolic instruction “ADD”
 - Symbolic address “SUM1”
- For instance

ADD

21, SUM1

Programming Languages



Programming Languages

- High-level languages
 - Specifies algorithms at a more abstract level
 - Interpreter reads instructions, controls machine actions
 - Examples: JavaScript, PHP
- Declarative languages
 - Specifies desired results, but not the control flow
 - System decides how best to get that result
 - Examples: HTML, SQL, Excel

High level Languages

- Procedural (modular) programming
 - Group instructions into meaningful abstractions
 - C, Pascal, Perl
- Object oriented programming
 - Group “data” and “methods” into “objects”
 - Naturally represents the world around us
 - C++, Java, JavaScript, PHP, Ruby

Where does the JavaScript go?

```
<!DOCTYPE html>  
<html>  
<head>  
<meta charset=utf-8 />  
<title>My Title</title>
```

```
<script>  
...  
</script>
```

JavaScript in the header, processed
before the page is loaded

```
<script src="code.js">  
</script>
```

JavaScript in an external file,
processed before the page is loaded

```
</head>  
<body>
```

```
<script>  
...  
</script>
```

JavaScript in the body, processed
as the page is loaded

```
</body>  
</html>
```

Key Ideas

- State
 - Data as a representation of the world
- Control flow
 - Flowcharts
 - Pseudocode

Variables

- Data types = things that you can operate on
 - Boolean: true, false
 - Number: 5, 9, 3.1415926
 - String: “Hello World”
- Variables hold values of a particular data type
 - Represented as symbols (e.g., x)
 - Choose meaningful variable names
 - “Camel Case”: numberOfSquaresInBattleship
- In JavaScript, var declares a variable
 - var b = true; create a boolean b and set it to true
 - var n = 1; create a number n and set it to 1
 - var s = “hello”; create a string s and set it to “hello”

The Assignment Statement

- $x = 4$ means “set x to 4”
 - In APL, this would be written $x \leftarrow 4$
- In mathematics, $x = x + 1$ is nonsense
 - In programming, it means increment x by one
 - It is so common, we say $x++$ as shorthand
- $x == 4$ means “is x equal to 4?”
 - If you write $x = 4$ for that, you will regret it!

Expressions & Statements

- Things that you can do:
 - $-x$ reverse the sign of x (negation)
 - $6 + 5$ add 6 and 5
 - $2.1 * 3$ multiply two values
 - `"Hello" + "World"` concatenate two strings
- The simplest statements store results of expressions:
 - $x = 5$ set the value of x to be 5
 - $x += y$ $x = x + y$
 - $x *= 5$ $x = x * 5$
 - $x++$ increase value of x by 1
- In JavaScript, statements end with a semicolon (;)

Strings

- `var s = "Mr. Spock"`
- `s.length` is 9
- `s.toLowerCase()` is "mr. spock"
- `s.substr(3,4)` is " Spo"
- `s.indexOf("k")` is 8
- `s.split(" ")` is ["Mr.", "Spock"]
- `s.link(http://bit.ly.CUjV)` is
 "`<a href=http://bit.ly.CUjV>Mr. Spock</a>`"
- `s + "Captain Kirk"` is "Mr. SpockCaptainKirk"

Working with Strings

- When asking input from the user, the input is always read as a string
- To convert types you can do:
 - `var number = Number(stringValue);`
 - `var stringValue = String(number);`

Interaction

○ Input

- `var t = prompt("message here", "default");`
 - When asking input from the user, the input is always read as a string
 - To convert types:
 - `var number = Number(stringValue);`
 - `var stringValue = String(number);`

○ Output

- `document.writeln("message here");`
- `console.log("message here");`
- `alert ("message here");`

```
<!doctype html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Input/Output</title>
  </head>

  <body>
    <script type="text/javascript">
      document.writeln("Bill Calculation System <br />");

      var costPerCredit, numberOfCredits, tuitionCost;

      /* Reading values from the user */
      costPerCredit = prompt("Enter cost per credit:");
      numberOfCredits = prompt("Enter number of credits:");

      // Computing cost
      tuitionCost = costPerCredit * numberOfCredits;

      document.writeln("Tuition Cost:" + tuitionCost);
    </script>
  </body>
</html>
```

Basic Control Structures

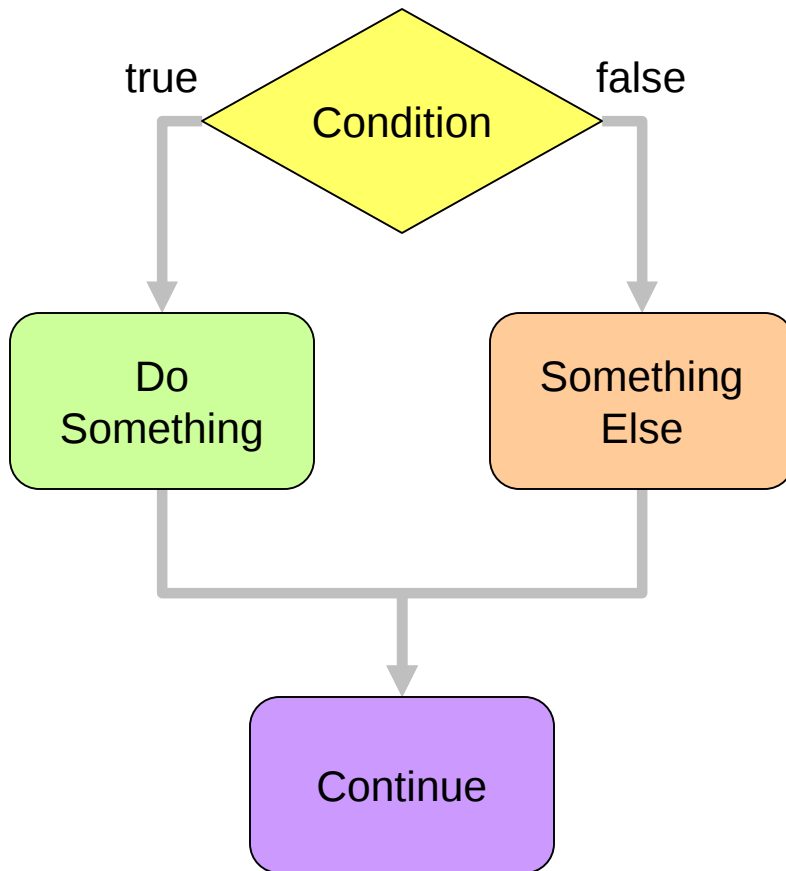
- Sequential
 - Perform instructions one after another
- Conditional
 - Perform instructions contingent on something
- Repetition
 - Repeat instructions until a condition is met

1: Sequential Control Structure



```
var a = 2;  
var b = 3;  
var c = a * b;
```

2: Conditional Control Structure



```
if (gender == "male") {  
    greeting = "It's a boy!";  
} else {  
    greeting = "It's a girl!";  
}
```

Note, the text in red is part of the “template” of the conditional

Note the indentation...

Nested if-else clauses

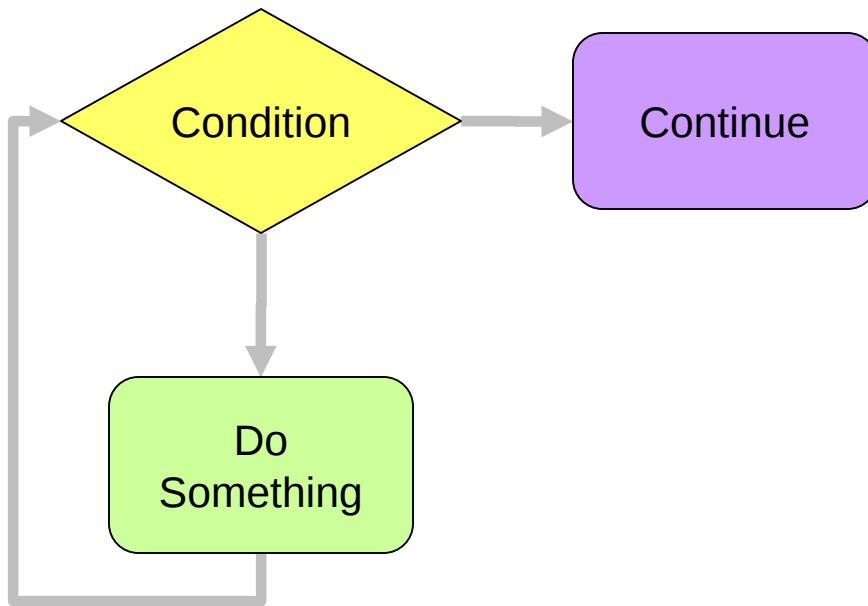
```
if ( expression ) {  
    if ( expression ) {  
        ...  
    } else {  
        ...  
    }  
} else {  
    ...  
}
```

Note this is where indentation become important...

Multiple if-else clauses

```
if ( expression ) {  
    ...  
} else if ( expression ) {  
    ...  
} else if ( expression ) {  
    ...  
} else {  
    ...  
}
```

3: Iterative Control Structure (Loop)



```
var n = 1;  
while (n <= 10) {  
    document.writeln(n);  
    n++;  
}
```

```
for (var n = 1; n <= 10; n++) {  
    document.writeln(n);  
}
```

Note, the text in red is part of the “template” of the loop

FYI: Computer scientists like to start at zero...

Boolean Operators

- `x == y` true if x and y are equal [use `==` not `=`]
- `x != y` true if x and y are not equal
- `x > y` true if x is greater than y
- `x <= y` true if x is smaller than or equal to y
- `x && y` true if both x and y are true
- `x || y` true if either x or y is true
- `!x` true if x is false

Design Tips

- Protect against unexpected values
 - Test the value of **all** user input
 - Test the value of critical function parameters
- Verify that every loop will **always** terminate
 - Include a bailout condition, and report it
- Always test for conditions explicitly
 - Trap unexpected conditions with the final else

Programming Tips

- Attention to detail!
 - Careful where you place that comma, semicolon, etc.
- Don't get cute with the logic or the layout
 - Reflect the structure of your problem clearly
 - Use standard “design patterns”
- Write a little bit of code at a time
 - Add some functionality, make sure it works, move on
- Debug by viewing the “state” of your program
 - Print values of variables using `document.writeln();`

Programming Tips

- Details are everything!
 - Careful where you place that comma, semi-colon, etc.
- Write a little bit of code at a time
 - Add a small new functionality, make sure it works, then move on
 - Don't try to write a large program all at once
 - If it doesn't work, revert back to previous version that worked
- Debug by outputting the state of the program
 - Simulate what you think the program is doing
 - Print out the value of variables using `document.writeln` or `console.log`
 - Is the value what you expected?
- Use the Chrome JavaScript console!

Documentation Tips

- Reflect your pseudocode in your code
 - Use meaningful variable names
 - Use functions for abstractable concepts
 - And name those functions well
 - Use comments to fill remaining gaps
- Add a comment to identify each revision
 - Give author, date, nature of the change
- Waste space effectively
 - Use indentation and blank lines to guide the eye

Algorithms

- A finite sequence of well-defined instructions designed to accomplish a certain task
- Named for the Persian mathematician Al-Khwarizmi

Group Exercise

- Calculate the value of a \$10,000 investment at the end of each year each year from a list of annual percentage gains or losses, and make a note in each year for which a constant 5% interest rate would outperform the variable rate investment.

2001	−11.9%
2002	−22.1%
2003	28.7%
2004	10.9%
2005	4.9%
2006	15.8%
2007	5.5%
2008	−37.0%
2009	26.5%
2010	15.1%

Pair Exercises

- Print every even number below 873 in the Fibonacci series (1, 1, 2, 3, 5 8, ... see Wikipedia).
- Print a 9x9 lower triangular matrix of asterisks.
- Prompt the user to enter a date (number of the month and number of the day), check to see if the date is valid (assume February has 28 days), and reprompt until a valid date is entered.

Some Math Functions

- `Math.abs()` – Absolute value
 - Example: `Math.abs(-10)`
- `Math.max()` – Maximum of two values
 - Example: `Math.max(10, 20)`
- `Math.sqrt()` – Square root
 - Example: `Math.sqrt(4)`
- `Math.random()` – Random value between 0 and less than 1
 - Example: `Math.random()`
- Constants
 - `Math.PI` – Mathematical constant pi

One More Example

- Write a JavaScript program that asks for a number (n) and writes an HTML table with two columns:
 - Column1: numbers 0 to n
 - Column2: square root of number

Square Root Table

Number	Square Root
0	0
1	1
2	1.4142135623730951
3	1.7320508075688772
4	2

For n=4

```
<!doctype html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Square Root Table</title>
  </head>
  <body>
    <script type="text/javascript">
      var currValue = 0;
      var maximumValue;
      maximumValue = Number(prompt("Enter maximum value"));

      document.writeln("<table border=\"10\">");
      document.writeln("<caption> Table</caption>");
      document.writeln("<tr><th>Number</th><th>2*Number</th></tr>");

      while (currValue <= maximumValue) {
        document.writeln("<tr><td>" + currValue + "</td><td>" + currValue*2 + "</td></tr>");
        currValue = currValue + 1;
      }

      document.writeln("</table>");
    </script>
  </body>
</html>
```

A Look Ahead

- Modular programming
 - Functions
- Data structures
 - Arrays
- Object-oriented programming
 - The document object
 - Events

Functions

- Reusable code for complex “statements”
 - Takes one or more values as “parameters”
 - Returns at most one value as the “result”

```
function convertToCelsius(f) {  
    var celsius = 5/9 * (f-32);  
    return celsius;  
}
```

```
var f = 60;  
c = convertToCelsius(f);
```

```
c = convertToCelsius(60);
```

```
function convertToCelsius(f) {  
    var celsius = 5/9 * (f-32);  
    return celsius;  
}
```

```
graph TD; Call[c = convertToCelsius(60);] -- "60" --> Entry(( )); Entry --> FunctionBody[function convertToCelsius(f) {  
    var celsius = 5/9 * (f-32);  
    return celsius;  
}]; FunctionBody -- "return celsius;" --> Exit(( )); Exit --> Call; style Entry fill:#add8e6,stroke:#000,stroke-width:1px; style Exit fill:#add8e6,stroke:#000,stroke-width:1px;
```

Scope of a Variable

- In JavaScript, *var* “declares” a variable
 - `var mystery;` create a variable without defining its type
 - `var b = true;` create a boolean *b* and set it to true
 - `var n = 1;` create an integer *n* and set it to 1
 - `var s = “hello”;` create a string *s* and set it to “hello”
- Variables declared in a function are “local”
 - Same name outside function refers to **different** variable
- All other variables are “global”

Writing JavaScript Functions

- Convenient to put in the <head> section
 - Use <!-- ... //--> to prevent display of code

```
...
<head>
<script language="JavaScript" type="text/javascript">
<!--
function calculate() {
    var num = eval(document.input.number.value);
    ...
    document.output.number.value = total;
}
//-->
</script>
</head>
...
```

Using JavaScript with Forms

HTML:

```
<form name="input" action="">
  Please enter a number:
  <input size="10" value=" " name="number"/>
</form>
<form name="output" action="">
  The sum of all numbers up to the number above is
  <input size="10" value=" " name="number" readonly="true"/>
</form>
```

JavaScript:

```
var num = eval(document.input.number.value);
document.output.number.value = 10;
```

Reads in a value from the first form
(*eval* method turns it into a number)

Changes the value in the second form

Arrays

- A set of elements
 - For example, the number of days in each month
- Each element is assigned an index
 - A number used to refer to that element
 - For example, $x[4]$ is the fifth element (count from zero!)
 - Arrays and repetitions work naturally together

Some Useful Predefined “Methods”

- `document.writeln(“...”);`
 - String gets **rendered** as HTML
 - Include “
” to force a line break
- `window.alert(“...”);`
 - String is **written verbatim** as text
 - Include “\n” to force a line break
- `foo = window.prompt(“...”);`
 - String is **shown verbatim** as text
 - Result is whatever string the user enters

Handling Events

- Events:
 - Actions that users perform while visiting a page
- Use event handlers to response events
 - Event handlers triggered by events
 - Examples of event handlers in Javascript
 - `onMouseover`: the mouse moved over an object
 - `onMouseout`: the mouse moved off an object
 - `onClick`: the user clicked on an object

Before You Go

On a sheet of paper, answer the following (ungraded) question (no names, please):

What was the muddiest point in today's class?