

JavaScript

About JavaScript

- JavaScript *is not Java*, or even *related to Java*
 - The original name for JavaScript was “LiveScript”
 - The name was changed when Java became popular
- Statements in JavaScript resemble statements in Java, because both languages borrowed heavily from the C language
 - JavaScript should be fairly easy for Java programmers to learn
 - However, JavaScript *is* a complete, full-featured, complex language
- JavaScript is seldom used to write complete “programs”
 - Instead, small bits of JavaScript are used to add functionality to HTML pages
 - JavaScript is often used in conjunction with HTML “forms”
- JavaScript is *reasonably* platform-independent

Using JavaScript in a browser

- JavaScript code is included within `<script>` tags:
 - `<script type="text/javascript">`
 `document.write("<h1>Hello World!</h1>");`
 `</script>`
- Notes:
 - The **type** attribute is to allow you to use other scripting languages (but JavaScript is the default)
 - This simple code does the same thing as just putting `<h1>Hello World!</h1>` in the same place in the HTML document
 - The semicolon at the end of the JavaScript statement is optional
 - You need semicolons if you put two or more statements on the same line
 - It's probably a good idea to keep using semicolons

Dealing with old browsers

- Some old browsers do not recognize **script** tags
 - These browsers will ignore the **script** tags but will *display* the included JavaScript
 - To get old browsers to ignore the whole thing, use:

```
<script type="text/javascript">  
  <!--  
    document.write("Hello World!")  
  //-->  
</script>
```
 - The `<!--` introduces an HTML comment
 - To get JavaScript to ignore the HTML close comment, `-->`, the `//` starts a JavaScript comment, which extends to the end of the line

Where to put JavaScript

- JavaScript can be put in the `<head>` or in the `<body>` of an HTML document
 - JavaScript *functions* should be defined in the `<head>`
 - This ensures that the function is loaded before it is needed
 - JavaScript in the `<body>` will be executed as the page loads
- JavaScript can be put in a separate `.js` file
 - `<script src="myJavaScriptFile.js"></script>`
 - Put this HTML wherever you would put the actual JavaScript code
 - An external `.js` file lets you use the same JavaScript on multiple HTML pages
 - The external `.js` file cannot itself contain a `<script>` tag
- JavaScript can be put in HTML *form object*, such as a button
 - This JavaScript will be executed when the form object is used

Primitive data types

- JavaScript has three “primitive” types: **number**, **string**, and **boolean**
 - Everything else is an object
- Numbers are always stored as floating-point values
 - Hexadecimal numbers begin with **0x**
 - Some platforms treat **0123** as octal, others treat it as decimal
- Strings may be enclosed in single quotes or double quotes
 - Strings can contains **\n** (newline), **\"** (double quote), etc.
- Booleans are either **true** or **false**
 - **0**, **"0"**, empty strings, **undefined**, **null**, and **NaN** are **false** , other values are **true**

Variables

- Variables are declared with a **var** statement:
 - **var pi = 3.1416, x, y, name = "Dr. Dave" ;**
 - Variables names must begin with a letter or underscore
 - Variable names are case-sensitive
 - Variables are *untyped* (they can hold values of any type)
 - The word **var** is optional (but it's good style to use it)
- Variables declared within a function are local to that function (accessible only within that function)
- Variables declared outside a function are global (accessible from anywhere on the page)

Operators, I

- Because most JavaScript syntax is borrowed from C (and is therefore just like Java), we won't spend much time on it

- Arithmetic operators:

+ - * / % ++ --

- Comparison operators:

< <= == != >= >

- Logical operators:

&& || ! (&& and || are *short-circuit* operators)

- Bitwise operators:

& | ^ ~ << >> >>>

- Assignment operators:

+= -= *= /= %= <<= >>= >>>= &= ^= |=

Operators, II

- String operator:
+
- The conditional operator:
condition ? value_if_true : value_if_false
- Special equality tests:
 - `==` and `!=` try to convert their operands to the same type before performing the test
 - `===` and `!==` consider their operands *unequal* if they are of different types
- Additional operators (to be discussed):
`new` `typeof` `void` `delete`

Comments

- Comments are as in C or Java:
 - Between `//` and the end of the line
 - Between `/*` and `*/`
- Java's javadoc comments, `/** ... */`, are treated just the same as `/* ... */` comments; they have no special meaning in JavaScript

Statements, I

- Most JavaScript statements are also borrowed from C
 - Assignment: `greeting = "Hello, " + name;`
 - Compound statement:
`{ statement; ...; statement }`
 - If statements:
`if (condition) statement;`
`if (condition) statement; else statement;`
 - Familiar loop statements:
`while (condition) statement;`
`do statement while (condition);`
`for (initialization; condition; increment) statement;`

Statements, II

- The switch statement:

```
switch (expression){  
    case label :  
        statement;  
        break;  
    case label :  
        statement;  
        break;  
    ...  
    default : statement;  
}
```

- Other familiar statements:

- break;
- continue;
- The empty statement, as in ;; or { }

JavaScript is not Java

- By now you should have realized that you *already know* a great deal of JavaScript
 - So far we have talked about things that are *the same* as in Java
- JavaScript has some features that *resemble* features in Java:
 - JavaScript has Objects and primitive data types
 - JavaScript has qualified names; for example, `document.write("Hello World");`
 - JavaScript has Events and event handlers
 - Exception handling in JavaScript is *almost* the same as in Java
- JavaScript has some features *unlike* anything in Java:
 - *Variable names* are untyped: the type of a variable depends on the value it is currently holding
 - Objects and arrays are defined in quite a different way
 - JavaScript has `with` statements and a new kind of `for` statement

Exception handling, I

- Exception handling in JavaScript is *almost* the same as in Java
- **throw** *expression* creates and throws an exception
 - The *expression* is the value of the exception, and can be of *any* type (often, it's a literal String)
- **try** {
 statements to try
} **catch** (*e*) { *// Notice: no type declaration for e*
 exception-handling statements
} **finally** { *// optional, as usual*
 code that is always executed
}
- With this form, there is *only one* **catch** clause

Exception handling, II

- ```
try {
 statements to try
} catch (e if test1) {
 exception-handling for the case that test1 is true
} catch (e if test2) {
 exception-handling for when test1 is false and test2 is true
} catch (e) {
 exception-handling for when both test1 and test2 are false
} finally { // optional, as usual
 code that is always executed
}
```
- Typically, the test would be something like  

```
e == "InvalidNameException"
```

# Object literals

- You don't declare the *types* of variables in JavaScript
- JavaScript has object *literals*, written with this syntax:
  - `{ name1 : value1 , ... , nameN : valueN }`
- Example (from Netscape's documentation):
  - `car = {myCar: "Saturn", 7: "Mazda",  
getCar: CarTypes("Honda"), special: Sales}`
    - The fields are `myCar`, `getCar`, `7` (this is a legal field name) ,  
and `special`
    - `"Saturn"` and `"Mazda"` are Strings
    - `CarTypes` is a function call
    - `Sales` is a variable you defined earlier
  - Example use: `document.write("I own a " + car.myCar);`



# Three ways to create an object

- You can use an object literal:
  - `var course = { number: "CIT597", teacher: "Dr. Dave" }`
- You can use **new** to create a “blank” object, and add fields to it later:
  - `var course = new Object();`  
`course.number = "CIT597";`  
`course.teacher = "Dr. Dave";`
- You can write and use a constructor:
  - `function Course(n, t) { // best placed in <head>`  
`this.number = n;`  
`this.teacher = t;`  
`}`
  - `var course = new Course("CIT597", "Dr. Dave");`

# Array literals

- You don't declare the *types* of variables in JavaScript
- JavaScript has array *literals*, written with brackets and commas
  - Example: `color = ["red", "yellow", "green", "blue"];`
  - Arrays are *zero-based*: `color[0]` is "red"
- If you put two commas in a row, the array has an “empty” element in that location
  - Example: `color = ["red", , "green", "blue"];`
    - `color` has 5 elements
  - However, a single comma at the end is ignored
    - Example: `color = ["red", , "green", "blue",];` still has 5 elements

# Four ways to create an array

- You can use an array literal:  
`var colors = ["red", "green", "blue"];`
- You can use `new Array()` to create an empty array:
  - `var colors = new Array();`
  - You can add elements to the array later:  
`colors[0] = "red"; colors[2] = "blue"; colors[1]="green";`
- You can use `new Array(n)` with a single numeric argument to create an array of that size
  - `var colors = new Array(3);`
- You can use `new Array(...)` with two or more arguments to create an array containing those values:
  - `var colors = new Array("red","green", "blue");`

# The length of an array

- If **myArray** is an array, its length is given by **myArray.length**
- Array length can be changed by assignment beyond the current length
  - Example: **var myArray = new Array(5); myArray[10] = 3;**
- Arrays are **sparse**, that is, space is only allocated for elements that have been assigned a value
  - Example: **myArray[50000] = 3;** is perfectly OK
  - But indices must be between 0 and  $2^{32}-1$
- As in C and Java, there are no two-dimensional arrays; but you can have an array of arrays: **myArray[5][3]**

# Arrays and objects

- Arrays *are* objects
- `car = { myCar: "Saturn", 7: "Mazda" }`
  - `car[7]` is the same as `car.7`
  - `car.myCar` is the same as `car["myCar"]`
- If you *know* the name of a property, you can use dot notation: `car.myCar`
- If you *don't know* the name of a property, but you have it in a variable (or can compute it), you *must* use array notation: `car["my" + "Car"]`

# Array functions

- If **myArray** is an array,
  - **myArray.sort()** sorts the array alphabetically
  - **myArray.sort(function(a, b) { return a - b; })** sorts numerically
  - **myArray.reverse()** reverses the array elements
  - **myArray.push(...)** adds any number of new elements to the end of the array, and increases the array's length
  - **myArray.pop()** removes and returns the last element of the array, and decrements the array's length
  - **myArray.toString()** returns a string containing the values of the array elements, separated by commas

# The for...in statement

- You can loop through all the properties of an object with **for (variable in object) statement**;
  - Example: 

```
for (var prop in course) {
 document.write(prop + ": " + course[prop]);
}
```
  - Possible output: **teacher: Dr. Dave**  
**number: CIT597**
  - The properties are accessed in an *undefined* order
  - If you add or delete properties of the object within the loop, it is *undefined* whether the loop will visit those properties
  - Arrays *are* objects; applied to an array, **for...in** will visit the “properties” **0, 1, 2, ...**
  - Notice that **course["teacher"]** is equivalent to **course.teacher**
    - You must use brackets if the property name is *in a variable*

# The with statement

- **with** (*object*) *statement* ; uses the *object* as the default prefix for variables in the *statement*
- For example, the following are equivalent:
  - **with** (document.myForm) {  
    result.value = compute(myInput.value) ;  
}
  - document.myForm.result.value =  
    compute(document.myForm.myInput.value);
- One of my books hints at mysterious problems resulting from the use of **with**, and recommends against ever using it



# Functions

- Functions should be defined in the **<head>** of an HTML page, to ensure that they are loaded first
- The syntax for defining a function is:  
**function** *name*(*arg1*, ..., *argN*) { *statements* }
  - The function may contain **return** *value*; statements
  - Any variables declared within the function are local to it
- The syntax for calling a function is just  
*name*(*arg1*, ..., *argN*)
- Simple parameters are passed *by value*, objects are passed *by reference*

# Regular expressions

- A regular expression can be written in either of two ways:
  - Within slashes, such as `re = /ab+c/`
  - With a constructor, such as `re = new RegExp("ab+c")`
- Regular expressions are almost the same as in Perl or Java (only a few unusual features are missing)
- `string.match(regex)` searches *string* for an occurrence of *regex*
  - It returns `null` if nothing is found
  - If *regex* has the `g` (global search) flag set, `match` returns an array of matched substrings
  - If `g` is not set, `match` returns an array whose 0<sup>th</sup> element is the matched text, extra elements are the parenthesized subexpressions, and the `index` property is the start position of the matched substring