

Jargon Review

- *Variable* – “container” to hold data in a script
- *Identifier* – the “name” of a variable
- *Literal* – data that appears directly in source code (e.g. 5, “thomas”, true)
- Type – describes the format/nature of the data
 - Primitive JavaScript types: string, number, Boolean
 - Complex JavaScript types: function, array, object

Jargon Review Contd.

- There are strong, weak, and untyped languages
 - JavaScript is a weakly typed language for better or worse
- When you *declare* a variable you are setting up space to hold data
 - `var x;`
- When you *assign* a variable you put some data into it (associate it with something)
 - `x = "foo";`

Variables

- Variables are defined with the **var** keyword
 - `var myName = "Fred";`
 - `var x;`
 - `var x, y, z;`
 - `var a = 1, x="hello", longerName;`
- Variable names or more appropriate identifiers should start with a letter and may be followed by any number of letters, digits, underscores, or dashes
 - Variable identifiers must not contain special characters or white space

Objects

- We'll see in JavaScript there are four built-in object types
 1. Type related – String, Number, Boolean, Array, Object
 2. Built-in – Date, Math, RegExp
 3. Browser – Window, Navigator
 4. Document – Document, Form, Image, etc.

Objects

- A fifth type would be user created, but that is somewhat rare given the coding style used by many JS programmers as well as the applicability of OOP principles to extremely small scripts.
- In general objects are created with the **new** operator as well as the appropriate object constructor
 - `var myArray = new Array(5);`
 - `var myLocation = new Object();`
`myLocation.city = "San Diego";`
`myLocation.state = "California";`

Arrays

- An array is an ordered list that can contain primitive and complex data types.
- Array can be declared explicitly or created using the Array object constructor (Arrays are objects too!)
 - `var myArray = [5, 67, 45243, "hello"];`
 - `var myArray2 = new Array(); // var myArray2 = [];`
- Arrays are zero based indexed
- Arrays elements are accessed with the `[]` operator
 - `alert myArray[0]; alert (myArray[1]);`
`alert (myArray[3]);`
 - `alert (myArray[4]);`
 - `myArray[4] = "hi there";`

Arrays

- As we can see with assigning to new locations, arrays are extended automatically
 - `myArray[500] = "wow!";`
 - Sparse arrays may result! Be careful about memory issues even though JavaScript does use garbage collection.
- Arrays are objects and they have many properties and methods
 - `alert(myArray.length);`
 - `myArray.reverse();`

Functions

- Functions are objects and thus are a data type
- Function are declared like so:

```
function functionname(parameter list)  
{  
    function statement(s)  
    optional return statement(s)  
}
```

- Functions are called by invoking function name with a list of parameters

```
functionname(parameter1, ... parameter n);
```


Form Validation Basics

- What to check?
 - Field blank
 - Field numeric
 - Field in range (min to max)
 - Field in some specified format (xxx-xx-xxx)
- Try to use form fields to keep errors from happening
 - State as 2 char text box vs a pull-down
- When should the form validation happen?
 - As the errors happen?
 - Between fields?
 - When the form is submitted?

Form Validation Basics Contd.

```
<form action="dosomething.cgi"  
      method="post"  
      onsubmit="return validate( )">  
  various fields  
  <input type="submit" value="send" />  
</form>
```

- The function validate should be built to return true if the fields are ok and false otherwise
- Avoid using the script to perform the submit, the action is for free and makes the form work when script is off

Pseudo-Code for Validation

```
function validate() {  
    errorMessage = ""  
    inError = false;  
  
    determine fields to check  
    check each field  
  
    if field is bad {  
        add to errorMessage reporting on field;  
        set inError = true;  
    }  
  
    When done checking fields  
    if inError == true {  
        display errorMessage  
        focus firstfield in error  
        return false  
    }  
    otherwise  
        return true //everything is good  
}
```

Form Validation Tips

- Don't hardcode the checks
 - Consider passing in the fields or an array with the fields and the appropriate validations
- Do the validation all at once to limit user annoyance
 - Flag each field in error somehow (color, message, etc.)
- Provide reasonable error messages that show how to create the problem

Form Validation Tips Contd.

- Each form field is slightly different
 - Text
 - Password
 - Textarea
 - Select and multi-select
 - Checkbox
 - Radio
 - Reset and submit
- The main problem for most folks is the radio group because of the array and the awkwardness of accessing the **options[]** array in a select value.
- See demos for the nitty gritty details of every type of form field and how it can be accessed
 - <http://www.javascriptref.com/examples/ch14/index.htm>

Form Validation Tips Contd.

- Do not confuse **id** and **name**, they are two different things in XHTML though they do overlap for backwards compatability
 - **id** must be unique but **name** does not need to be
 - Be very careful on radio and selects
- Be aware of unset defaults for attributes like **size**, **maxlength**, etc.
- Add some usability improvements on page load, reset, etc. to make users happy

Form Validation Tips Contd.

- Do not reinvent the wheel many scripts are out there for years that will do more than you will ever need to do
 - Ancient history even:
<http://developer.netscape.com/docs/examples/index.html?content=javascript.html>