

LECTURE-2

Javascript

- Data types
- Operators
- Control Statement
- Popup Boxes
- Functions

JAVASCRIPT – OVERVIEW

- This course is concerned with client side JS
 - Executes on client (browser)
- Scripting – NOT compile/link.
- Helps provide dynamic nature of HTML pages.
- Included as part of HTML pages as
 - Regular code (viewable by user)
 - A file present in some location.
- **NOTE:** Javascript is **NOT** the same as JAVA

A SIMPLE JAVASCRIPT PROGRAM

```
<html>
  <head>
    <title> A simple Javascript program
  </title>
</head>
<body>
  <!--The code below in “script” is Javascript code. -->
  <script>
    document.write (“A Simple Javascript program”);
  </script>
</body>
</html>
```

JAVASCRIPT CODE

- Javascript code in HTML
- Javascript code can be placed in
 - `<head>` part of HTML file
 - Code is **NOT** executed unless called in `<body>` part of the file.
 - `<body>` part of HTML file – executed along with the rest of body part.
 - Outside HTML file, location is specified.
 - Executed when called in `<body>`

WAYS OF DEFINING JAVASCRIPT CODE.

First:

```
<head>
  <script type="text/javascript">
    function foo(...) // defined here
  </script>
</head>

<body>
  <script type="text/javascript">
    foo(...) // called here
  </script>
</body>
```

Second:

```
<head>
  ...
</head>
<body>
  <script>
    function foo(...) // defined here
    {
      ..
    }
    foo() // Called here
  </script>
</body>
```

WAYS OF DEFINING JAVASCRIPT CODE, CONTD.

Third:

<head>

// Any general location, that can be accessed.

<script src="http://cnn.com/foo.js">

</script>

</head>

<body>

<script>

// Javascript code called here.

</script>

</body>

JAVASCRIPT – DATA TYPES

- Basic data types
 - **number**: E.g., 2, 3, 4, 10.6, 3.1415, 1.2e8, 3.2e-10
 - **string**: E.g., “abc”, “Bob Doe”
 - **boolean**: true, false
- Complex data types
 - Objects
 - Functions

JAVASCRIPT – BASIC TYPES

- Not a strongly typed language
 - `x = 5; x = "string"` is perfectly acceptable.
- Case sensitive.
- A variable has a “var” prefix.
 - “var x = 5” is same as just, “x = 5”.
- Re-declaration is possible
- Possible to mix and match while printing
 - `document.write (6 + 10 + “xyz”)` → prints 16xyz
 - `document.write (“xyz” + 6 + 10)` → prints xyz610

JAVASCRIPT – OBJECTS

- `var person = {
 firstName:"John",
 lastName:"Doe",
 email:"JohnDoe@gmail.com",
 age: 38
};`
- To access values, use
 - `object.propertyName` or
 - `object["propertyName"]`
- Example
 - `person[firstName]`
 - `person["firstName"]`

JAVASCRIPT – FINDING DATA TYPES

- Useful operator to find data types: `typeof`
 - `typeof(10) → number`
 - `typeof(3.1415) → number`
 - `typeof("abcd") → string`
 - `name = "Bob"; typeof(name) → string`
 - `typeof(true) → boolean`
 - `value = false; typeof(value) → false`
 - `var x = 10; typeof(x) → number`
 - `typeof(x) → undefined.`
 - `typeof(person) → object`

JAVASCRIPT – OPERATORS

- Arithmetic operators: Usual ones
 - +, -, *, /, %, ++, --
- Assignment operators
 - =, +=, -=, *=, /=, % =
- String operators:
 - + (concatenation operator)
- Comparison operators:
 - ==, !=, ===, >, <, >=, <=
- Logical operators
 - &&, ||, !
- Conditional operator
 - variable = (condition) ? value1 : value2

JAVASCRIPT – CONTROL STATEMENTS

if statement:

Example:

if (cond1)

<code-1>

else if (cond2)

<code-2>

...

else

<code-3>

if (x %2 == 0)

document.write("x is a multiple of 2");

else if (x%3 == 0)

document.write("x is a multiple of 3");

else

document.write("x is not a multiple of 2 or 3);

JAVASCRIPT – CONTROL STATEMENTS

for statement:

```
for (i = 0; i < n; i++)  
{  
    // code  
}
```

Example:

```
var x;  
var mycars = new Array();  
mycars[0] = "Saab";  
mycars[1] = "Volvo";  
mycars[2] = "BMW";  
for (x in mycars)  
{  
    document.write (mycars[x] + "<br />");  
}
```

JAVASCRIPT – CONTROL STATEMENTS ... CONTD.

do statement

```
do
{
    <code>
} while (cond);
```

while statement:

```
while (cond)
{
    <code>
}
```

switch statement:

```
switch (n)
{
    case n1:
        <code>
        break;
    ...
    default:
        <code>
}
```

JAVASCRIPT – POPUP BOXES

- Alert box
 - `alert (“alert text”);`
- Confirm box
 - `confirm (“confirm some text”);`
- Prompt box
 - `prompt (“prompt text”, “default value”)`

JS POPUP BOXES – GETTING VALUES

- Confirm box
 - `var i = confirm (“Press OK or Cancel”)`
- Prompt box
 - `var i = prompt (“Enter some value”, “default”);`
- Alert box
 - `alert (“alert text”);`

JAVASCRIPT FUNCTIONS

Syntax

```
function <functionName> (params)
{
    // code
}
```

Note: Parameters do **NOT** have variable type.

1. Recall: Function definition can be in
 - <head> part of HTML file.
 - <body> portion of HTML file
 - An external file.
2. “return” value of the function is optional.

FUNCTIONS – EXAMPLE I

```
<html>
<head>
  <title> Example of a simple function </title>
  <script type="text/javascript">
    function factorial (input)
    {
      product = 1;
      for (i=1; i <= input; i++)
        product *= i;
      document.write ("factorial of i " + product);
    }
  </script>
</head>
<body>
  <h1> Example of a simple function </h1>
  <script>
    factorial (9);
  </script>
</body>
</html>
```

FUNCTIONS – EXAMPLE2

```
<html>  
  <head>  
    <title>Browser Information example</title>  
    <script>  
      function BrowserInfoFn( )  
      {  
        var browser = navigator.appName;  
        var version = navigator.appVersion;  
        var ver = parseFloat (version);  
        document.write ("Broswer: " + browser + " version:" + version + " ver: " + ver + "<br />");  
      }  
    </script>  
  </head>  
  <body>  
    <h1>Browser Information example</h1>  
    <script>  
      BrowserInfoFn( );  
    </script>  
    <hr>  
  </body>  
</html>
```

SPECIAL FUNCTIONS IN JAVASCRIPT

<code>encodeURIComponent</code>	encodes special characters of a URI, except: , / ? : @ & = + \$ #
<code>encodeComponentURI</code>	Encodes special characters and , / ? : @ & = + \$ # of a URI
<code>decodeURI</code>	Opposite of <code>encodeURIComponent</code>
<code>decodeComponentURI</code>	Opposite of <code>decodeComponentURI</code>
<code>escape</code>	encodes special characters, except: * @ - _ + . /
<code>unescape</code>	Opposite of <code>escape</code> - decodes a string
<code>eval</code>	Evaluates and executes a string as Javascript code
<code>isFinite</code>	Finds out if argument is a finite, valid number
<code>isNaN</code>	Finds out if argument is not a number
<code>Number</code>	Converts a string to integer
<code>String</code>	Converts argument to string
<code>parseFloat</code>	Parses argument and returns a float value
<code>parseInt</code>	Parses argument and returns an integer value