CSC 242 - Web Programming

Introduction to JavaScript

JavaScript

- JavaScript is a client-side scripting language – the code is executed by the web browser
- JavaScript is an *embedded* language – it relies on its host environment for IO
- JavaScript IO options:
  - `console.log`: writes output to the browser console
  - `alert`: writes output to pop-up window
  - `document.write`: writes output to the HTML document

The `script` Element

- JavaScript source code is placed in an HTML document within the `script` element
- An external JavaScript source file can be imported with the `src` attribute:

```
<script type="text/javascript"
        src="url/file.js">
</script>
```

- The `<noscript>` tag defines alternate content when JavaScript is disabled or not available

Basic Syntax

- A statement does not need to be terminated by a semicolon when it is the only statement on a line
- Line comments are denoted by `//`
- Block comments are denoted by `/* ... */`

Simple Example

```
<!DOCTYPE html>
<html>
<head>
  <title>Hello World</title>
</head>
<body>
  <script>
    document.write("Hello World");
  </script>
  <noscript>
    Your browser does not support or has
    disabled JavaScript
  </noscript>
<body>
</html>
```

JavaScript Variable Naming Rules

- A variable may include only the characters `a-z,A-Z,0-9,` the `$` symbol, and the underscore (`_`)
- No other characters are allowed in a variable name
- The first character in a variable name must be a letter, `$`, or `_`
- Variable names are case sensitive

JavaScript Types

- JavaScript data types:
  - Object
  - Function
  - Number: (Integer and Float)
  - String
  - Boolean
  - Null
  - Undefined
- JavaScript is dynamically typed – types of variables do not need to be declared
- JavaScript is weakly typed – some type conversions are automatic

The String Type

- The string type represents a sequence of characters
- The string type must be enclosed by single or double quotes
- The escape character is the backslash (\)
- The plus (+) operator performs string concatenation

Multi-line strings

- A string can be defined over multiple lines by escaping the newline character

```
name = "first_name \
     last name";
```

Arithmetic Operators

Operator	Description	Example
+	Addition	`a + 3`
-	Subtraction	`a - 3`
*	Multiplication	`a * 3`
/	Division	`a / 3`
%	Modulus	`a % 3`
++	Increment	`++a`
--	Decrement	`--a`

Assignment Operators

Operator	Example	Equivalent to
=	`a = 3`	`a = 3`
+=	`a += 3`	`a = a + 3`
+=	`a += 3`	`a = a + 'text'`
-=	`a -= 3`	`a = a - 3`
*=	`a *= 3`	`a = a * 3`
/=	`a /= 3`	`a = a / 3`
%=	`a %= 3`	`a = a % 3`

JavaScript Implicit Type Coercion

- The type of a variable is implicitly converted based on the context in which the variable is used

```
<script>
  x = "10"; // string
  y = 3.14; // number
  z = x * y; // number
</script>
```

- The `typeof` function returns a string representation of a variable's type

Explicit Type Casting Functions

- `parseInt()` cast to Int, Integer
- `Boolean()` cast to boolean
- `parseFloat()` cast to Float, Double, Real
- `String()` cast to string
- `split()` cast to array

Equality & Comparison Operators

Operator	Description	Example
==	equal to	`a == 3`
===	identical to	`a === 3`
!=	not equal to	`a != 3`
!==	not identical to	`a !== 3`
>	greater than	`a > 3`
<	less than	`a < 3`
>=	greater than or equal to	`a >= 3`
<=	less than or equal to	`a <= 3`

Logical Operators

Operator	Description	Example
`&&`	and	`a == 3 && b == 0`
`||`	or	`a == 3 || b == 0`
`!`	not	`!(a == b)`

Selection

- `if`, `else`, and `else if`

```
if       (a > 100) {document.write(">")}
else if  (a < 100) {document.write("<")}
else               {document.write("=")}
```

- `switch`

```
switch (page) {
    case ("Home"):
        document.write("Home");
        break;
    case ("About"):
        document.write("About");
        break;
    default:
        break;
```

Iteration

- `while` loops
- `do while` loops
- `for` loops
- Example:

```
for (var count = 1; count <= 10; ++count) {
  document.write("Count:" + count + "<br>");
}
```

- `break` and `continue`

Defining a JavaScript Function

```
function function_name([parameter [, ...]])
{
    // Statements

    [return]
}
```

- A definition starts with the word function
- Next is the name of the function, which must start with a letter or underscore, followed by any number of letters, numbers, or underscores
- Function names are case sensitive
- The parentheses are required
- Zero or more parameters, separated by commas
- A value can be returned from a function with the `return` keyword

Variable Scope

- *Local variables* are accessible in context in which they are defined
- *Global variables* are accessible from all parts of the code
- Function parameters have local scope
- The `var` keyword defines a local variable with a scope of the current function
- Example:

```
function test() {
    a = 123 // global
    var b = 456 // local
    if (a == 123) {
        var c = 789 // local
    }
}
```

JavaScript Objects

- A JavaScript object groups data with functions that manipulate it
- The data members of an object are referred to as properties
- The functions of an object are referred to as methods

JavaScript Object Literal Syntax

```
object_name = {
    property1: value1,
    property2: value2,
    method1: function (parameters) {
        function_body
    }
};
```

Accessing Object Properties and Methods

- Syntax to access properties

```
object_name.property_name;
// or
object_name["property_name"];
```

- Syntax to access methods

```
object_name.method_name(parameters);
```

JavaScript Numeric Arrays

- JavaScript numeric arrays are special objects with numeric indices
- Array creation syntax:

```
array_name = [item1, item2, ...];
```

- Array access syntax:

```
array_name[index];
```

- The `length` property of an array stores the number of elements in an array:

Some JavaScript Array Methods

- `toString`: converts an array to a string
- `join`: converts an array to a string with specified separator
- `pop`: removes the last element of the array
- `push`: adds a new element to the end of an array
- `sort`: sorts an array in place

JavaScript Associative Arrays

- JavaScript associative arrays are objects
- The use of named indices converts an array to an object
- The array methods and properties are incompatible with the object type