8. JS Objects & Functions

충남대학교 컴퓨터공학과
데이타베이스시스템 연구실

JavaScript Objects

JavaScript Objects

- In JavaScript, almost "everything" is an object.
  - Booleans can be objects (or primitive data treated as objects)
  - Numbers can be objects (or primitive data treated as objects)
  - Strings can be objects (or primitive data treated as objects)
  - Dates are always objects
  - Maths are always objects
  - Regular expressions are always objects
  - Arrays are always objects.
  - Functions are always objects.
  - Objects are objects.
- Primitive values are : string("John Doe"), number(3.14), true, false, null, and undefined.

JavaScript Objects (cont'd)

- **Objects are Variables Containing Variables** *Try it!*
  - JavaScript object is an unordered collection of variables called **named values**.

- **Object Properties**
  - The named values are called **properties**

Property	Property Value
firstName	John
lastName	Doe
age	50
eyeColor	blue

JavaScript Objects (cont'd)

Object Methods

- An object property containing a function definition.
  - Object properties can be both primitive values, other objects, and functions.

Property	Value
firstName	John
lastName	Doe
age	50
eyeColor	blue
fullName	function() {return this.firstName + " " + this.lastName;}

JavaScript Objects (cont'd)

- **Creating a JavaScript Object**
  - Different ways to create new objects
    - Define and create a single object, using an object literal. *Try it!*
    - Define and create a single object, with the keyword new. *Try it!*
    - Define an object constructor, and then create objects of the constructed type. *Try it!*

- **The *this* Keyword**
    - The thing called this, is the object that "owns" the JavaScript code.

- **Built-in JavaScript Constructors** *Try it!*

JavaScript Objects (cont'd)

- **Did You Know?**
  - There is no reason to create complex objects. Primitive values execute much faster. *Try it!*

- **JavaScript Objects are Mutable** *Try it!*

JavaScript Object Properties

Accessing JavaScript Properties

```
objectName.property          // person.age
```

Try it!

or

```
objectName["property"]       // person["age"]
```

Try it!

or

```
objectName[expression]       // x = "age"; person[x]
```

JavaScript Object Properties (cont'd)

- **JavaScript for …in Loop** *Try it!*
  - Loops through the properties of an objects
  - Syntax

```
for (variable in object) {
    code to be executed
}
```

- **Adding New Properties** *Try it!*

- **Deleting Properties** *Try it!*
  - **delete** keyword deletes a property from an object.

JavaScript Object Methods

- **Accessing Object Methods** *Try it!* *Try it!*
  - Create an object method with the following syntax

    ```
    methodName : function( ) { code lines }
    ```

  - Access an object method with the following syntax

    ```
    objectName.methodName( )
    ```

- **Using Built-In Methods**
  - Example

    ```
    var  message  = "Hello world!";
    var  x=message.toUpperCase( );
    ```

- **Adding New Methods** *Try it!*

JavaScript Object Prototypes

- **JavaScript Prototypes**
  - All JavaScript objects inherit the properties and method from their prototype.
  - Objects created using an object literal, or with new Object( ), inherit from a prototype called Object.prototype.

- **Creating a Prototype** *Try it!*

- **Adding a Property to an Object** *Try it!*

- **Adding a Method to an Object** *Try it!*

JavaScript Object Prototypes (cont'd)

- **Adding Properties to an Prototype** *Try it!* *Try it!*
- **Adding Methods to a Prototype** *Try it!*
- **Using the prototype Property**
  - JavaScript prototype property allows you to add new properties to an existing prototype *Try it!*
  - JavaScript prototype property allows you to add new methods to an existing prototype *Try it!*

JavaScript Function Definitions

- **Function Declarations** *Try it!*
  - Syntax

```
function functionName(parameters) {
    code to be executed
}
```

- **Function Expressions** *Try it!*
  - The function above is actually an anonymous function (a function without a name)

JavaScript Function Definitions (cont'd)

- **Function Hoisting**

```
myFunction (5);

function  myFunction (y) {
      return  y*y;
}
```

  - Hoisting is JavaScript's default behavior of moving declarations to the top.

- **Self-Invoking Function** *Try it!*

JavaScript Function Definitions (cont'd)

- **Functions Can Be Used as Values** *Try it!*

- **Functions are Objects**
  - The **typeof** operator in JavaScript returns "function" for functions.
  - JavaScript functions have both properties and methods. *Try it!* *Try it!*

JavaScript Function Parameters

Function Parameters and Arguments

```
functionName(parameter1, parameter2, parameter3) {
    code to be executed
}
```

- Function **parameters** are the **names** listed in the function definition
- Function **arguments** are the real **values** passed to (and received by) the function.

Parameter Rules

- JavaScript function definitions do not specify data types for parameters
- JavaScript functions do not perform type checking on the passed arguments
- JavaScript functions do not check the number of arguments received.

JavaScript Function Parameters (cont'd)

- **Parameter Defaults**
  - If a function is called with **missing arguments** (less than declared), the missing values are set to : **undefined** *Try it!*
  - If a function is called with too many arguments(more than declared), these arguments cannot be referred

- **The Arguments Object** ***Try it!***
  - JavaScript functions have a built-in object called the arguments object.
  - The argument object contains an array of the arguments used when the function was called (invoked).

JavaScript Function Invocation

- **JavaScript functions can be invoked in 4 different ways.**
- **Invoking a Function as a Function** *Try it!* *Try it!*
- **The Global Object** *Try it!*
  - When a function is called without an owner object, the value of **this** becomes the global object.
  - In a web browser the global object is the browser window.
- **Invoking a Function as a Method** *Try it!* *Try it!*

JavaScript Function Invocation (cont'd)

- **Invoking a Function with a Function Constructor** *Try it!*
  - If a function invocation is preceded with the **new** keyword, it is a constructor invocation

- **Invoking a Function with a Function Method**
  - Both methods takes an owner object as the first argument
    - call( ) takes the function arguments separately
    - apply( ) takes the function arguments in an array.

```
function myFunction(a, b) {
    return a * b;
}
myFunction.call(myObject, 10, 2);       // Will return 20
```

```
function myFunction(a, b) {
    return a * b;
}
myArray = [10,2];
myFunction.apply(myObject, myArray);    // Will also return 20
```

JavaScript Closures

- **Global Variables**
  - Local variables can only be used inside the function where it is defined. *Try it!*
  - Global variables can be used (and changed) by all scripts in the page (and in the window). *Try it!*

- **Variable Lifetime**
  - Global variables live as long as your application lives
  - Local variables have short lives. They are create when the function is invoked, and deleted when the function is finished.

JavaScript Closures (cont'd)

- **JavaScript Nested Functions**
  - Nested functions have access to the scope "above" them. *Try it!*
- **JavaScript Closures** *Try it!*
  - A closure is a function having access to the parent scope, even after the parent function has closed.