

CMSC388A

Web Application Development with JavaScript



JavaScript II

Department of Computer Science
University of MD, College Park

Slides material developed by Ilchul Yoon, Nelson Padua-Perez

Resources

- Check “YouTube Resources” at
<https://www.cs.umd.edu/~nelson/classes/resources/web/>

Type Conversions

- In JavaScript, you don't specify the type of variables
- Most of the time, implicit transformations will take care of transforming a value to the expected one
- **Example:**
 - `let age = 10;`
 - `let personsInfo = "John Age: " + age;`
- Mechanism to transform values explicitly:
 - **Converting number to string**
 - » `let stringValue = String(number);`
 - **Converting string to number**
 - » `let number = Number(stringValue);`
 - » `let number = parseInt(stringValue);`
 - » `let number = parseFloat(stringValue);`

Comparisons

- You can compare values by using the following operators
 - ===** Return true if the values are equal, false otherwise
(e.g., `x === y`)
 - !==** Returns true if the values are different, false otherwise
(e.g., `x !== y`)
- `==` and `!=` Not as strict as previous equality operators
- **Relational Operators**
 - `<` Less than
 - `>` Greater than
 - `<=` Less than or equal
 - `>=` Greater than or equal

Dialog Boxes – Basic Input/Output

- **document.writeln()/document.write()** - generates page content
- **Example:** WriteVsWriteIn.html
- We can perform input and output via dialog boxes
- Input via **prompt** - Returns a string
 - If you need to perform mathematical computations, you might need to convert the value read into a number explicitly
- **Example:** InputOutput.html
 - We can define several variables at the same time
 - **prompt** is a function that displays a dialog box with the specified title. It can be used to read any data. You can specify the default value after the title
 - You can read numbers and strings via **prompt**
- **window.alert()** - function used to display a message in a dialog box
- **window.open()** - generates a pop-up window with the specified website
- **Example:** Network.html
 - You have to execute it twice; once to allow pop-ups; the second time, the actual program is executed

Control Structures

- Constructs having syntax /semantic similar to Java
 - **while, do-while, for loops**
 - **if** statement
 - **cascaded if** statements
 - **break** statement
 - **switch** statement
- **Example:** SqrTable.html

Strict Mode

- Use the **strict** mode pragma
 - "use strict";
- If pragma is used outside of a function, it applies to all the script
- It can appear in a function

```
function computeAvg() {  
    "use strict";  
    ...  
}
```

- Enforces that variables have to be declared first
- Restricts use of reserved words (interface, package, private, ...)
- **Example:** Strict.html

Console

- Allow us to view JavaScript errors and user messages
- **console** object functions
 - **log** - General message
 - **info** - Informational message
 - **error** - Error message
 - **warn** - Warning message
 - **table** - Displays array in tabular form
- **In Chrome**
 - View → Developers → JavaScript console
 - Different icons are used for different console functions
 - You can explore JavaScript constructs by typing code at the console
- **Example:** ConsoleEx.html

Built-in Structural Types

- **Object** - generic object
- **Array** - list of values (numerically indexed)
- **Function**
- **Error** - runtime error
- **Date** - date/time
- **RegExp** - regular expression
- Many built-in types have a **literal form** that enables you to define a value without explicitly creating an object (using **new**)
- The typical function definition is based on a literal form

Primitive Wrapper Types

- JavaScript promptly coerces between primitives and objects when a property of the type is accessed
- Three wrapper types: **Boolean**, **String**, and **Number**
- Primitive wrapper types simplify working with primitives
- Wrapper types are automatically created when needed
- **Example:** Wrapper.html, WrapperType.html

Global Object

- ECMAScript defines a global object
- In JavaScript, **window** implements the global object
 - Recently, **globalThis** was added to the language as a standardized name for a global object
- **All functions and variables defined globally become part of the global object**
 - Try defining a variable and see if you can access it using the window object
 - **Example:** GlobalObject.html
- Some **functions** that are part of the Global object
 - isNaN()
 - parseFloat() - Let's try on the console parseInt("1.4in")
 - parseInt() - Let's try on the console parseInt("1.4in")
 - eval() - evaluates JS code represented as a string
 - isFinite()

Examples typed in
Chrome JS Console

```
> isNaN(2.3)
< false
> eval(2.3)
< 2.3
> eval("3*2");
< 6
> window.isNaN(Infinity)
< false
> isFinite(Infinity)
< false
> |
```

Global Object

- Some **properties** that are part of the Global object
 - NaN (also part of Number)
 - undefined
 - Object : Constructor for Object
 - Array : Constructor for Array
 - Function : Constructor for Function
 - Number : Constructor for Number
 - String : Constructor for String
 - Date : Constructor for Date
 - Error : Constructor for Error
 - RegExp : Constructor for RegExp
- ECMAScript also defines the Math object
 - Try Math.random(), Math.floor(), Math.PI

Examples typed in Chrome JS Console

```
> var k = Number(3);
< undefined

> k
< 3

> var k = new Number(3.1);
< undefined

> k
< ▼ Number {3.1} ⓘ
    ► __proto__: Number
    [[PrimitiveValue]]: 3.1

> var k = Number(3.1);
< undefined

> k
< 3.1

> window.isNaN("Hello everyone");
< true

> isNaN(3.29e10)
< false

>
```

Functions

- **Functions are objects**
- The name of a function is a reference value
- Functions can be passed and returned from other functions
- Functions can be defined inside of other functions
- Function literal form **declaration**

```
function name (<comma-separated list of parameters>) {  
    statements  
}
```

Note: parameters do not use let, const, or var (just parameter name)

Functions

- Functions are invoked by using the () operator
- Don't use **var**, **let**, **const** for parameters (e.g., function print(x, y))
- Parameters are **passed by value**
- There is **no mandatory main** function
- Returning values via **return**
- Recursion is supported (there is stack as in Java)

Three Approaches to Define Functions

- **Function declaration**
 - Read and available before any code is executed (hoisted)
 - When a function is **hoisted**, it is internally moved to the beginning of the current scope. So...
 - » Hoisting allows calling the function before its declaration (i.e., functions can appear in any order)
- **Function expression (anonymous function)**
 - `let myFunction1 = function(x, y) { return x * y };`
- **Using Function constructor**
 - `let myFunction2 = new Function("x", "y", "return x * y");`
- **Arrow Functions (lambda expressions)**
 - `let myFunction3 = (x, y) => x * y;`
- Function overloading is not possible as the second function definition will redefine the first one
- **Example:** DefiningFunctions.html, FunctionsAsData.html, DisplayValues.html