

CMSC335

Web Application Development with JavaScript



JavaScript2

Department of Computer Science

University of MD, College Park

Slides material developed by Ilchul Yoon, Nelson Padua-Perez

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:
 - **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
- We can perform input and output via dialog boxes
- Input via **prompt**
 - **Returns a string**
 - If you need to perform a mathematical computation you might need to explicitly convert the value read into a number
- **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 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 with the specified website
- **Example:** Network.html
 - You have to execute twice; once to allow pop-ups; 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

- Allows for error checking both globally or within a function
- Use the strict mode pragma
 - “use strict”;
- If **pragma** used outside of a function, it applies to all the script
- It can appear in a function

```
function computeAvg() {  
    “use strict”;  
}
```
- Variables have to be declared first
- Cannot use 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 practice JavaScript by typing code at the console
- **Example:** ConsoleEx.html

Built-in Structural Types

- **Object** - generic object
- **Array** - list of values (numerically indexed)
- **Function** - (non-data structure)
- **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
- Some **functions** that are part of the Global object
 - isNaN()
 - parseFloat()
 - parseInt()
 - 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

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 declaration

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

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

Three Approaches to Define Functions

- **Function declaration**
 - Read and available before any code is executed
 - 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**
 - `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 second function definition will redefine the first one
- **Example:** [DefiningFunctions.html](#), [FunctionsAsData.html](#)

One-Dimensional Arrays

- **Array** : Collection of values that can be treated as a unit or individually
 - **A special type of objects**

```
var a = new Array(4);
```

- **Indexing** : access an element using []
 - First element associated with index 0 (e.g., a[0])
- An element of an array can be of any type and an array can hold different types of elements
- The **length** property represents the length of the array (e.g., a.length)
- Try printing the contents of an array by using **alert**

Definition of One-Dim Arrays

- **Using literal form**

- Comma separated list of elements within square brackets

- let a = [2, 3, 5];

- let b = []; /* Empty array */

- **Using Array constructor**

- let c = new Array();

- let e = new Array(4); /* Defines array of size 4 */

- **Example:** ArraysOneDim.html, ArraysLengthProp.html

Two-Dimensional Arrays

- Can be passed to or returned from functions like one-dim arrays
- Any modifications to the array in the function will be permanent
- You can have ragged arrays
- **Example:** ArraysTwoDim.html

String Methods

- **Comparison based on < and >**
- **concat** - returns a new string representing concatenation of strings
- **includes** - determines whether one string is found within another
- **startsWith** – determines whether string begins with characters from another string
- **endsWith** – determines whether string ends with characters from another string
- **indexOf** - index of first character in string (or -1 if not found)
- **lastIndexOf** - index of last occurrence of character in the string (or -1 if not found)
- **repeat** - returns string repeated **n** times
- **splice** - extracts section of a string
- **split** - splits a string into array of strings
- **toLowerCase/toUpperCase**
- **trim** - trims whitespaces
- **Example:** StringMethods.html
- **Reference**
 - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

Getting String Characters

- The function **charAt** or **[]** allows us to access the character associated with a particular index position in a string
 - Access is similar to array indexing (first character at 0)
- **Example:**

```
let x = "Wednesday";  
let secondCharacter = x.charAt(1); /* Variable has "e " */  
let lengthOfString = x.length;    /* Variable has 9 */
```
- **Example:** CharAt.html

More about Functions

- Functions can be passed and returned from other functions (one more example)
- **Example:** ProcessValues.html

JavaScript Errors

- You may get a blank page when there is an error
- Use console to see error
- Additional debugging information:
- <http://www.cs.umd.edu/~nelson/classes/resources/JavaScript/JavaScriptDebugging/>

JavaScript References

- Excellent source of information

<https://developer.mozilla.org/en-US/docs/Web/JavaScript>

- Equivalent of Java API:

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference>

- The previous reference provides excellent examples describing the functionality of methods. Let's take a look at a couple of methods and the provided examples