

WEB222 - Web Programming Principles

Week 3: JavaScript Objects

Agenda

- JavaScript objects
- Standard Built-in objects
 - String,
 - RegExp,
 - Array

JavaScript Objects

- In JavaScript an "object" is a self-contained entity comprising of "properties" (variable), and "methods" (functions).
 - an object can store data in its properties - state
 - and perform actions with its methods - behavior.
- In JavaScript, you can create a new object without using a class.
 - The object does not belong to any class; it is the only one of its kind, a singleton

JavaScript Object Categories

- There are three kinds of JavaScript object:
 - **Built-in objects**
 - an intrinsic part of JavaScript, providing useful features, such as String, Array, Date, Math and JSON
 - **Host Objects**
 - objects that are supplied to JavaScript by the browser environment. e.g. DOM (Document Object Model) /BOM (Browser Object Model) objects, ie: window, document and form, etc...
 - **Custom objects**
 - user-defined object to store data and provide functionality in a single object

JavaScript Built-in Objects

- JavaScript provides many predefined, built-in objects that enable you to work with Strings and Dates, perform mathematical operations, and etc.:
 - String
 - RegExp
 - Array
 - Date
 - Math
 - Number
 - Boolean
 - JSON
- We'll cover String, Array, RegExp in this week.

JavaScript String Objects

- Strings enclosed within double or single quotes are used for holding data that can be represented in text format.
- Some of the most-used operations on strings are to
 - check their length, and to
 - concatenate strings using the + and += string operators.

String object - properties and methods

Member	Type	Example
length	property	stringName.length
charAt(n)	method	stringName.charAt(n)
charCodeAt(n)	method	stringName.charCodeAt(n)
concat(string2, string3)	method	stringName.concat(string2, string3)
indexOf('x')	method	stringName.indexOf('x')
lastIndexOf('x')	method	stringName.lastIndexOf('x')
split('x')	method	var arrayName = stringName.split('x')
substr(x,y)	method	stringName.substr(x,y) – x=from, y=length
substring(x,y)	method	stringName.substring(x,y) – x=from (inclusive) y=to (not inclusive)
toLowerCase()	method	Converts a string to lowercase.
toUpperCase()	method	Converts a string to uppercase.
trim()	method	Removes whitespaces from the left and right of a string.
prototype	property	Allows you to add properties and methods to an object

JS String object - property

➤ Length

- The length property returns the number of characters in a string.
- Syntax: **stringName.length**

Position/index » 012345

```
var myString    =    "WEB222";
```

myString.length // returns 6

JS String object - methods

➤ **charAt(index)**

- The method returns the character at the specific index.
- Characters in a string are indexed from left to right
 - Index start from 0 to one less than the length.
- The index of the last character in a string called myString is **myString.length - 1**
- If the index you supply is out of range, JavaScript returns an **empty string**
- Syntax: `stringName.charAt(index)`

Position/index » 012345

var **myString** = "WEB123" ;

```
myString.charAt(0) // returns 'W'  
myString.charAt(1) // returns 'E'  
myString.charAt(2) // returns "B"  
myString.charAt(3) // returns '1'  
myString.charAt(4) // returns '2'
```

```
myString.charAt(5) // returns "3"  
myString.charAt(6) // returns ""
```

NOTE: you can also use array index:
`myString[1] // returns 'N'`

JS String object - methods

➤ **charCodeAt(index)**

- The method returns the **unicode** of a character.
- Index can be a value from 0 to one less than the length.
- Syntax: `stringName.charCodeAt(index)`

Position/index » **012345**

```
var myString = "AZaz09";
```

```
console.log( myString.charCodeAt(0) ); // returns 65
console.log( myString.charCodeAt(1) ); // returns 90
console.log( myString.charCodeAt(2) ); // returns 97
console.log( myString.charCodeAt(3) ); // returns 122
console.log( myString.charCodeAt(4) ); // returns 48
console.log( myString.charCodeAt(5) ); // returns 57
console.log( myString.charCodeAt(6) ); // returns NaN
```

About Unicode

- Unicode provides a unique number for every character, no matter what the platform is.
- **UTF-8**: a character encoding has become the dominant for the World Wide Web.
- See unicodes table.

JS String object - methods

➤ concat(string2,string3,...)

- The concat(....) method concatenates the text of two or more strings.
- It is always recommended to use the operators (+, +=) instead of the concat method.
- Syntax: stringName.concat(string2, string3)

```
var myString0 = "My courses are: ";
```

```
var myString1 = "WEB222";
```

```
var myString2 = "OOP222";
```

```
myString0 = myString0.concat(myString1, " & ", myString2);
```

```
console.log(myString0); // My courses are: WEB222 & OOP244
```

JS String object - methods

➤ indexOf(subStr)

- returns the position at which the character or string begins. indexOf returns only the first occurrence of your character or string.
- If indexOf returns zero, the character or the string you are looking for begins at the 1st character.
- If indexOf returns **-1**, the character or string you searched for is not contained within the string.

Position/index » 012345

```
var myString = "WEB222";
```

```
console.log( myString.indexOf("W") );           // 0
console.log( myString.indexOf("EB") );           // 1
console.log( myString.indexOf('2') );            // 3
console.log( myString.indexOf('b2') );           // -1
console.log( myString.indexOf('3') );           // -1
```

JS String object - methods

➤ lastIndexOf(subStr)

- returns the position at which the **last occurrence** of your character or string – searching backwards.
- If `lastIndexOf` returns **-1**, the character or string you searched for is **not contained** within the string.

Position/index » 012345

```
var myString = "WEB222";
```

```
alert( myString.lastIndexOf("W") );        // returns 0
alert( myString.lastIndexOf("WEB") );      // returns 0
alert( myString.lastIndexOf('2') );        // returns 5
alert( myString.lastIndexOf('22') );      // returns 4
alert( myString.lastIndexOf('3') );        // returns -1
```

JS String object - methods

➤ **split(x)**

- The `split(' ')` uses the specified character(s) to break the argument string into an array.
- Syntax: `stringName.split(x)`
- Notes: the opposite operation: `join(x)` of an array

Position/index » 012345

```
var myString = "WEB 222";
var myArray1 = myString.split(' ');
```

Split on a blank will return the following:

```
myArray1[0] // element 0 returns "WEB"
myArray1[1] // element 1 returns "222"
```

split(x)

Position/index » 012345

```
var myString = "WEB 222";
var myArray2 = myString.split('2');
```

Split on 2 will return the following:

```
myArray2[0] // element 0 returns "WEB "
myArray2[1] // element 1 returns  "" (empty string)
myArray2[2] // element 2 returns  "" (empty string)
myArray2[3] // element 3 returns  "" (empty string)
```

var myArray3 = myString.split('22'); // split on 22 will return the following:

```
myArray3[0] // element 0 returns  "WEB "
myArray3[1] // element 1 returns  "2"
```

JS String object - methods

► substr(x, len)

- The substr(x, y) returns a sub string where:
 - ▶ x – start index
 - ▶ len – length: how many characters
- Syntax: **stringName.substr(x, len)**

```
Position/index    »    012345  
  
var myString      = "WEB222";  
  
console.log( myString.substr(1,4) ); // "EB22"  
console.log( myString.substr(4,4) ); // "22"  
console.log( myString.substr((myString.length-4),1) ); // "B"  
console.log( myString.substr(4) ); // "22"
```

JS String object - methods

➤ **substring(x, y)**

- The `substring(x,y)` returns a sub string where:
 - x starting from (index) - inclusive
 - y to (index) - not inclusive - if y < than x, then switch the 2 parameters

```
Position/index    »    012345
```

```
var myString      = "WEB222";
```

```
console.log( myString.substring(1,4) );    // "EB2"
console.log( myString.substring(4,4) );    // ""
console.log( myString.substring(4,2) );    // "B2"
console.log( myString.substring(-4,4) );   // "WEB2"
console.log( myString.substring(0,9) );    // "WEB222"
console.log( myString.substring(4) );      // "22"
```

JS String object - methods

➤ **toUpperCase()**

- Converts a string to upper case
- Syntax: `stringName.toUpperCase()`

Position/index » 012345

```
var myString = "Seneca";
```

```
console.log( myString.toUpperCase() );
console.log( myString ); // value not changed
```

```
myString = myString.toUpperCase();
console.log( myString ); // value changed
```

JS String object - methods

➤ **toLowerCase()**

- Converts a string to lower case
- Syntax: `stringName.toLowerCase()`

Position/index » 012345

```
var myString     = "WEB222";
```

```
console.log( myString.length );                        // 6
console.log( myString.toLowerCase() ); // web222
console.log( myString );                                // WEB222
```

JS String object - methods

➤ trim()

- The trim() method removes whitespace (blank characters) from the left and right of the string.
- trimLeft() & trimRight() methods work with some browsers but not others - Don't use them.
- Syntax: `stringName.trim()`

```
var myString      =      "      WEB 222      ";
console.log( myString.length );      // 14
```

```
myString = myString.trim();
console.log( myString );          // WEB 222
console.log( myString.length );    // 7
```

JS String object - property

➤ prototype

- Allows you to add properties and methods to an object

➤ e.g.

```
String.prototype.reverse = function () {  
    var rev = '';  
    for (var i = this.length - 1; i >= 0; i--)  
        rev += this[i]; // the string  
    return rev;  
};  
  
var myString = "WEB222";  
console.log( myString.reverse() ); // 222BEW
```

JavaScript RegExp Object

- Regular expressions are patterns used to match character combinations and perform search-and-replace functions in strings.
- In JavaScript, regular expressions are also objects.
- RegExp
 - is short for regular expression.
 - is the JavaScript built-in object.
- Regular Expressions Tutorial

Creating RegExp Object

➤ Syntax

```
var patt=/pattern/[modifiers];
```

or :

```
var patt=new RegExp(pattern[, modifiers]);
```

- **Pattern:** the text of the regular expression.
- **Modifiers:** if specified, modifiers can have any combination of the following values:
 - **g** - global match
 - **i** - ignore case-sensitivity
 - **m** - multiline;

RegExp Method – test(str)

- A RegExp method that tests if a string contains the (RegExp) pattern.
- It returns true or false.
- Example

```
var str = "Welcome to Toronto";
```

```
var patt = /Me/;  
var result = patt.test(str);  
console.log(result); // false
```

```
var result2 = /Me/i.test(str);  
console.log(result2); // true
```

String Method – match(RegExp)

- A String method that executes a search for a match in a string.
- It returns an array of the found text value. or null on a mismatch.
- Example 1:

```
var str = "Welcome to Toronto";
var patt1 = /to/i; // i: ignore case-sensitivity (returns first match)
// same as: var patt1 = new RegExp("to", "i");

var result = str.match(patt1);
console.log(result);          // to
console.log(result[0]);       // to
```

String Method – match(RegExp)

► Example 2:

```
var str = "Welcome to Toronto";
var patt1 = /to/g;           // g: do a global
search
var result = str.match(patt1);
console.log(result);        // to,to
```

► Example 3:

```
var str = "Welcome to Toronto";
var result = str.match(/to/ig);
// ig: global and case-insensitive

console.log(result );      // to, To, to
```

String Method – replace(RegExp, replacement)

- A String method that executes a search for a match in a string, and replaces the matched substring with a replacement substring.
- Syntax:
`replace(RegExp, replacement)`
- e.g.

```
var str = "Welcome to Toronto";
var patt = /to/i; // i: ignore case-sensitivity
var result = str.replace(patt, "<TO>");
console.log(result);
console.log(str.replace(/to/g, "<TO>")); // g: global search
console.log(str.replace(/to/ig, "<TO>"));
console.log(str);
```

String Method – split(RegExp)

- A String method that uses a **regular expression** or a **fixed string** to break a string into an array of substrings.
- E.g.

```
var str = "School of ICT-Seneca College";  
  
console.log(str.split(' - ')); // [ "School of ICT", "Seneca College" ]  
  
console.log(str.split(/ - /)); // [ "School of ICT", "Seneca College" ]  
  
console.log(str.split(/[ - ]/)); // [ "School", "of", "ICT", "Seneca", "College" ]
```

String Method – search(RegExp)

- A String method that tests for a match in a string. It returns the index of the match, or -1 if the search fails.
- **Syntax: search(RegExp)**
- Example:

```
var myString = "WEB222"

myString.search(/222/);    // returns 3
myString.search(/B/);     // returns 2
myString.search(/322/);   // returns -1
```

RegExp Examples

- To validate: at least 4 alphabetical alphabetic character
 - var pattern1 = /^[a-zA-Z]{4,}\$/;
- To validate: telephone format #####-###-###
 - var pattern2 = ^([0-9]{3}[-])[2][0-9]{4}\$;

Special characters in regular expressions

character	meaning
^	String begin
\$	String end
.	Match – one character
?	Match – zero or one (preceding character)
*	Match – zero or more
+	Match – one or more
{m, n}	Match – m or n number of preceding character
[]	Character sets delimiters []
\d	= [0-9], Match - digits
\D	= [^0-9], Match – non-digits
\w	= [A-Za-z0-9_], Match – alphanumeric
\s	= [\t\r\n], Match – whitespace

Array Object

- The JavaScript array is a global object that is used to store multiple values in a single variable..
 - In JavaScript, variables in an array may have **different data type**.
- JavaScript arrays are high-level, list-like data structure and are different from the arrays in C or Java.
- Index starts from **0**.

Creating Arrays

➤ Using an array literal (recommended)

- e.g. 1

```
var arrayName1 = [11, 15, 13, "blue", 24, 35.05] ;
```

- e.g. 2

```
var arrayName2 = []; // an empty array
```

➤ Using the new keyword (for simplicity, do not use this method)

- e.g. 1

```
var arrayName1 = new Array (11, 15, 13, "blue", 24, 35.05);
```

- e.g. 2

```
var arrayName2 = new Array();
```

```
arrayName2[2] = 100;
```

Array object - properties and methods

Member	Type	Example
length	property	arrayName.length
prototype	property	Allows you to add properties and methods to an Array object
concat()	method	arrayName.concat()
join()	method	arrayName.join() or arrayName.join("+")
pop()	method	arrayName.pop()
push()	method	arrayName.push()
reverse()	method	arrayName.reverse()
slice()	method	arrayName.slice(x,y)
sort()	method	arrayName.sort()
splice()	method	splice(x,y,[.....])

JS Array object - property

➤ **length**

- The length property returns the number of elements / occurrences in the array

```
var myAarray1 = [];
```

```
var myAarray2 = [11, 15, 13, "blue", 24, 35.05];
```

```
var myAarray3 = new Array();
```

```
myAarray3 [2] = "Green";
```

```
console.log( myAarray1.length ); // 0
```

```
console.log( myAarray2.length ); // 6
```

```
console.log( myAarray3.length ); // 3
```

JS Array object - methods

➤ **pop()**

- The pop() method removes an entry from the end of the array and return the removed element.
- Syntax: `arrayName.pop()`

➤ **shift()**

- The shift() method removes an entry from the beginning of the array and return the removed element.
- Syntax: `arrayName.shift()`

➤ **Example**

```
var colors = ["Red", "Green", "Blue", "White", "Black"];
var last  = colors.pop();
var first = colors.shift();
console.log(colors); // [ "Green", "Blue", "White" ]
console.log(last);   // Black
console.log(first);  // Red
```

JS Array object - methods

➤ **push()**

- adds an new entry to the end of the array
- Syntax: `arrayName.push(newEntry)`

➤ Example

```
var colors = ["Red", "Green", "Blue"];
colors.push("Pink");
console.log(colors); // [ "Red", "Green", "Blue", "Pink" ]
```

JS Array object - methods

➤ concat()

- The concat() method Concatenates two or more arrays and returns a new array
- Syntax: `arrayName.concat(anotherArray, ...)`

```
var array1 = ["Red", "Green", "Blue"];
```

```
var array2 = [1, 2, 3, "Yellow"];
```

```
var newArray = array1.concat(array2);
```

```
console.log(newArray); // [ "Red", "Green", "Blue", 1, 2, 3, "Yellow" ]
```

JS Array object - methods

➤ **join()**

- The join() method is used to join all the elements of an array into a **single string** separated by a specified string separator
 - if no separator is specified, the default is a **comma**.
 - Notes: the opposite operation: **split()** method of a String
- **Syntax:** **arrayName.join(str)**

```
var myArray = ["Red", "Green", "Blue", "Yellow"];
console.log( myArray ); // ["Red", "Green", "Blue", "Yellow"]

console.log( myArray.join() ); // "Red,Green,Blue,Yellow"
console.log( myArray.join(' ') ); // "Green Blue Yellow"
console.log( myArray.join("&") ); // "Red&Green&Blue&Yellow"
```

JS Array object - methods

➤ **reverse()**

- The elements in the array are reversed. First becomes last, second becomes second last, etc..
- Syntax: **arrayName.reverse()**

```
var myArray = ["Red", "Green", "Blue", "Yellow"];
console.log( myArray ); // [ "Red", "Green", "Blue", "Yellow" ]
```

```
myArray = myArray.reverse();
console.log( myArray ); // [ "Yellow", "Blue", "Green", "Red" ]
```

JS Array object - methods

➤ **sort()**

- The elements in the array are sorted based on their ASCII code.
- Syntax: `arrayName.sort()`

```
var array1 = ["Red", 2, "Green", "15", "Blue", 101, "Yellow"];
var array2 = [99, 2, 38, 15, 66, 101, 200];
```

```
console.log( array1.sort() ); // [101, "15", 2, "Blue", "Green", "Red", "Yellow"]
console.log( array2.sort() ); // [101, 15, 2, 200, 38, 66, 99]
```

JS Array object - methods

➤ slice()

- The slice() method extracts part of an array and returns a new array.
- Syntax:
 - `arrayName.slice(index1, index2)`
 - Index2: not included

```
var colors = ["Red", "Green", "Blue", "Yellow", "White"];
var subclrs = colors.slice(1, 3)
```

```
console.log( colors ); // [ "Red", "Green", "Blue", "Yellow", "White" ]
console.log( subclrs ); // [ "Green", "Blue" ]
```

Loop through an array in JS

- Use a simple **for loop**, e.g.

```
var i, len;  
var colors = ["Red", "Green", "Blue", "Yellow", "White"];  
for (i = 0, len = colors.length; i < len; i++) {  
    console.log(i + ": " + colors[i]);  
}
```

- Use JavaScript Array **forEach() Method**, e.g.

```
function myFunction(item, index) {  
    console.log(index + ": " + item);  
}  
  
var colors = ["Red", "Green", "Blue", "Yellow", "White"];  
  
colors.forEach(myFunction);
```

- Don't use for-each (for-in) loop for an array

- e.g. ~~for (var x in colors){...}~~

Resourceful Links

- [Standard built-in objects - JavaScript | MDN](#)
- [JavaScript Array object | MDN](#)

Thank you!

Any Questions?