JavaScript Built-in objects

ITI - Assiut Branch

Eng. Sara Ali

JavaScript Objects

- **JavaScript Objects fall into 4 categories:**

1.Custom Objects

Objects that you, as a JavaScript developer, create and use.

2.Built – in Objects

Objects that are provided with JavaScript to make your life as a JavaScript developer easier.

3.BOM Objects “Browser Object Model”

It is a collection of objects that are accessible through the global objects window. The browser objects deal with the characteristic and properties of the web browser.

4.DOM Objects “Document Object Model”

Objects provide the foundation for creating dynamic web pages. The DOM provides the ability for a JavaScript script to access, manipulate, and extend the content of a web page dynamically.

JavaScript Built-in Objects

- String
- Number
- Array
- Date
- Math
- Boolean
- RegExp
- Error
- Object

String Object

- The JavaScript String object is a global object that is used to store strings. A string is a sequence of letters, numbers, special characters and arithmetic values or combination of all.
- Because strings must be written within quotes, JavaScript will misunderstand this string:

```
let text = "We are the so-called "Vikings" from the north.";
```

- The string will be chopped to "We are the so-called ".
- The solution to avoid this problem, is to use the **backslash escape character**.

Code	Result	Example
\'	'	let text= 'It\'s alright.';
\"	"	let text= "Welcome to \"Javascript\".";
\\	\	let text = "The character \\ is called backslash.";

String Object (Cont.)

- Normally, JavaScript strings are primitive values, created from literals:

```
let x= "John";
```

- But strings can also be defined as objects with the keyword new:

```
let y= new String("John");

// typeof x will return string
// typeof y will return object
```

- **String Properties**

Property	Description	Example
length	Returns the length of a string.	var str = “Hello”; str.length; //5

String Object(Cont.)

- **String Methods**

  Methods that fall into three categories:

  1. Manipulate the contents of the String
  2. Manipulate the appearance of the String
  3. Convert the String into an HTML element

String Object(Cont.)

- **1.Methods that Manipulate the contents of the String**

```
var myStr = "Let's see what happens!";
```

Method	Description	Example
charAt(index)	Returns the character at the specified index	myStr.charAt(0); //L
indexOf(string)	Returns the position of the first found occurrence of a specified value in a string	myStr.indexOf("at"); //12
lastIndexOf(string)	Returns the position of the last found occurrence of a specified value in a string	myStr.lastIndexOf("a"); //16

substring(index,index)	Extracts the characters from a string, between two specified indices	myStr.substring(1, 7); //et's s
		myStr.substring(2); //t's see what happens!

String Object(Cont.)

- **1.Methods that Manipulate the contents of the String (cont.)**

```
var myStr = "Let's see what happens!";
```

Method	Description	Example

replace(string)	Searches for a match between a substring (or regular expression) and a string, and replaces the matched substring with a new substring	myStr.replace(/e/,”?”); //L?t's see what happens!
		
		myStr.replace(/e/g,”?”); //L?t's s?? what happ?ns!

concat(string)	Joins two or more strings, and returns a copy of the joined strings	myStr.concat(“ myDate”); //Let's see what happens! myDate
endsWith()	Checks whether a string ends with a specified substring.	myStr.endsWith('!'); //true

String Object(Cont.)

- **1.Methods that Manipulate the contents of the String (cont.)**

```
var myStr = "Let's see what happens!";
```

Method	Description	Example

includes()	Checks whether a string contains the specified substring.	myStr.includes(“what") //true
		myStr.includes(“what“, 12) //false as start search from index 12

match()	Matches a string against a regular expression, and returns an array of all matches.	myStr.match(/at/); //at
		myStr.match(/a/); //a
		myStr.match(/a/g); //a,a

repeat()	Returns a new string which contains the specified number of copies of the original string.	myStr.repeat(2); //Let's see what happens!Let's see what happens!
search()	Searches a string against a regular expression, and returns the index of the first match.	myStr.search('see'); //6

String Object(Cont.)

- **1.Methods that Manipulate the contents of the String (cont.)**

```
var myStr = "Let's see what happens!";
```

Method	Description	Example

slice()	Extracts a portion of a string and returns it as a new string.	myStr.slice(0,3); //Let
		myStr.slice(-1); //! Last chacter

split()	Splits a string into an array of substrings.	myStr.split(" "); //Let's,see,what,happens!
trim()	Removes whitespace from both sides of a string. It does not change the original string.	var myStr = “ Hello “; myStr.trim(); //Hello

String Object(Cont.)

2.Methods that Manipulate the appearance of the String

Method	Example	Returned value
bold()	"hi".bold()	hi
fontcolor()	"hi".fontcolor("green")	<FONT COLOR="green“>hi</FONT>
fontsize()	"hi".fontsize(1)	<FONT SIZE="1">hi</FONT>
italics()	"hi".italics()	<I>hi</I>
strike()	"hi".strike()	<STRIKE>hi</STRIKE>
sup()	"hi".sup()	^{hi}
sub()	"hi".sub()	_{hi}
toLowerCase()	"UPPERcase".toLowerCase()	uppercase
toUpperCase()	"UPPERcase".toUpperCase()	UPPERCASE

String Object(Cont.)

3.Methods that Convert the String into an HTML element

var myStr = “Click me”;

Method	Example	Returned value
link(string)	“Click me".link("linktext") Or myStr. link("linktext")	Click me

Number Object

- The JavaScript Number object acts as a wrapper for primitive numeric values. JavaScript has only one kind of number data type and it doesn't distinguish between integers and floating-point values.
- **To create a Number Object**

```
var n=10; //recommended for number declaration

//OR

var n = new Number(10); //not recommended for number declaration

//OR

n = new Number(); //if not assigned a value initially n = 0

n=10; // value changed to n=10
```

Number Object (Cont.)

- **Number Properties**

Property	Description
MAX_SAFE_INTEGER	Represents the maximum safe integer in JavaScript (2^{53} - 1).
MAX_VALUE	Returns the largest numeric value representable in JavaScript, approximately 1.79E+308. Values larger than MAX_VALUE are represented as Infinity.
MIN_SAFE_INTEGER	Represents the minimum safe integer in JavaScript (-(2^{53} - 1)).
MIN_VALUE	Returns the smallest positive numeric value representable in JavaScript, approximately 5e-324. It is closest to 0, not the most negative number. Values smaller than MIN_VALUE are converted to 0.
NEGATIVE_INFINITY	Represents the negative infinity value.
NaN	Represents "Not-A-Number" value.
POSITIVE_INFINITY	Represents the infinity value.

Number Object (Cont.)

- **Number Methods**

Method	Description
toFixed(x)	Fixed-point representation of a number object as a string. Rounds the returned value. n = 34.8896; n.toFixed(6); //34.889600
toPrecision(x)	Formats any number so it is of "x" length var n = 34.8896; n.toPrecision (3); //34.9
toExponential(x)	Exponential notation of a number object as a string. Rounds the returned value. var n = 56789; n.toExponential(2); // "5.68e+4"
toString()	Converts from decimal system to any other system when passing its base as parameter var n= 10; var x=n.toString(2); //1010
	Returns a string representing the Number object. var numStr = n.toString(); //”10”
valueOf()	returns the primitive value of a Number object as a number data type. var n=new Number(10); n.valueOf() ; //10

Number Object (Cont.)

- **Number Methods**

Method	Description

isInteger()	Checks whether the passed value is an integer.
	Number.isInteger(123) //true
	Number.isInteger(-123) //true
	Number.isInteger(0.5) //false
	Number.isInteger('123') //false

isSafeInteger()	Checks whether a value is a safe integer.
	A safe integer is an integer that can be from (2^{53} - 1) to -(2^{53} - 1))
	Number.isSafeInteger(-123) //true
	Number.isSafeInteger(0) //true
	Number.isSafeInteger(0.5) //false

Math Object

- The JavaScript Math object is used to perform mathematical tasks. The Math object is a static object.
- **Math Properties**

Property	Description
Math.E	Returns Euler's number, the base of natural logarithms, e, approximately 2.718
Math.LN2	Returns the natural logarithm of 2, approximately 0.693
Math.LN10	Returns the natural logarithm of 10, approximately 2.302
Math.LOG2E	Returns the base 2 logarithm of e, approximately 1.442
Math.LOG10E	Returns the base 10 logarithm of e, approximately 0.434
Math.PI	Returns the ratio of the circumference of a circle to its diameter (i.e. π). The approximate value of PI is 3.14159
Math.SQRT1_2	Returns the square root of 1/2, approximately 0.707
Math.SQRT2	Returns the square root of 2, approximately 1.414

Math Object (Cont.)

- **Math Methods**

Method	Example	Returned value
cos(n)	Math.cos(.4)	0.921060994002885102 8
sin(n)	Math.sin(Math.PI)	0
tan(n)	Math.tan(1.5 * Math.PI)	infinity
acos(n)	Math.acos(.5)	1.04719755119659763 1
asin(n)	Math.asin(1)	1.57079632679489655 8
atan(n)	Math.atan(.5)	0.463647609000806093 5
exp(n)	Math.exp(8)	2980.957987041728302
sqrt(n)	Math.sqrt(9801)	99
log(n)	Math.log(5)	1.60943791243410028 2

Math Object (Cont.)

- **Math Methods (cont.)**

Method	Example	Returned value
max(x,y,…)	Math.max(1 , 700)	700
min(x,y,…)	Math.min(1 , 700,2)	1
pow(x,n)	Math.pow(2, 3)	8
abs(n)	Math.abs(-6.5)	6.5
random()	Math.random()	.7877896
floor(n)	Math.floor(8.9)	8
ceil(n)	Math.ceil(8.1)	9
round(n)	Math.round(.567)	1

Boolean Object

- **The Boolean object is an object wrapper around the boolean value true or false.**

```
var n = 0;

Or

Var n =false;

Or

var n = new Boolean(false);
```

- **Boolean Methods**

Method	Description	Example
toString()	Converts a Boolean value to a string, and returns the result.	var n=false; n.toString(); //"false" var n=0; n.toString(); //"0"
valueOf()	Returns the primitive value of a Boolean object.	var n=new Boolean(true);n.valueOf(); //true

Array Object

- The JavaScript Array object is a global object that is used in the construction of arrays. An array is a special type of variable that allows you to store multiple values in a single variable.
- **To declare an array:**

```
<script>
        var colorArray = new Array();
        colorArray [0]="red";
        colorArray [1]="blue";
        colorArray [2]="green;
//OR
        var colorArray = new  Array("red","blue","green");
//OR
        var colorArray=[];
        var colorArray=["red","blue","green"];
</script>
```

Array Object (Cont.)

Array Properties

Property	Description	Example
length	returns the number of elements in an array.	var arr = [“red”,”blue”] arr.length //2

Array Methods

var arr1=new Array(“A”,”B”,”C”); var arr2 = new Array(“1”,”2”,”0”)

Method	Description	Example
concat(array)	Joins two or more arrays, and returns a copy of the joined arrays	arr1.concat(arr2); //A,B,C,1,2,0

join()	Joins all elements of an array into a string	arr1.join(); //A,B,C
		arr1.join(“*”); //A*B*C

reverse()	Reverses the order of the elements in an array	arr1.reverse(); //C,B,A

Array Object (Cont.)

- **Array Methods (cont.)**

var arr1=new Array(“A”,”B”,”C”); var arr2 = new Array(“1”,”2”,”0”)

Method	Description	Example
pop()	Removes the last element of an array, and returns that element	arr1.pop(); //C, and the length becomes 2
push(element)	Adds new elements to the end of an array, and returns the new length	arr1.push(“D”); //4 (Length of the array) //and arr1[3]=“D”

shift()	Removes the first element of an array, and returns that element	arr1.shift(); // A
		//arr1[0] =“B” & arr[1]=“C”

unshift(element)	Adds new elements to the beginning of an array, and returns the new length	arr1.unshift(“D”); //4 //arr1[0]=“D”

slice(index)	Selects a part of an array, and returns the new array	arr1.slice(1); // B,C
		arr1.slice(2); //C

sort()	Sorts the elements of an array alphapitacally	var arr2 = [1,2,10]; arr2.sort() ; //1,10,2

Array Object (Cont.)

- **Array Methods (cont.)**

```
var arr1=new Array(“A”,”B”,”C”);
```

Method	Description	Example
copyWithin()	Copies part of an array to another location in the same array and returns it.	arr1.copyWithin(2, 0); //A,B,A
includes()	Determines whether an array includes a certain element.	Arr1.includes('C') //true

Array Object (Cont.)

- Associative Arrays:
  - Associative array is just like an ordinary array, except that instead of the indices being numbers, they're strings
  - Although the keys for an associative array have to be strings, the values can be of any data type, including other arrays or associative arrays.

```
<script>
        var assocArray = new Array( );
        assocArray["Ahmed"] = "Excellent";
        assocArray["Tareq"] = "pass";
</script>
```

Date Object

- The JavaScript Date object is a global object that is used to work with dates and times. The Date objects are based on a time value that is the number of milliseconds since January 1, 1970 UTC.

```
<script>
    var d = new Date(); // holds current date
    var d = new Date(dateString); // Ex. "October 13, 2014 11:13:00"
    var d = new Date(year, month, day, hours, minutes, seconds, milliseconds);
</script>
```

Date Object (Cont.)

- **Date Properties**

Property	Numeric Range
seconds, minutes	0 - 59
hours	0 - 23
day	0 - 6 (0 = Sunday, 1 = Monday, and so on)
date	1 - 31
month	0 - 11 (0 = January, 1 = February, and so on)
year	1900 90 2000

Date Object (Cont.)

- **Date Methods**
  - The Date object methods fall into three categories (get, set, to)

1.get Methods

var myDate= new Date ("November 25,2006 11:13:55");

Method	Example	Returned value
getDate()	myDate.getDate()	25
getMonth()	myDate.getMonth()	10
getFullYear()	myDate. getFullYear()	2006
getDay()	myDate.getDay()	0
getHours()	myDate.getHours()	11
getMinutes()	myDate.getMinutes()	13
getSeconds()	myDate.getSeconds()	55
getTime()	myDate.getTime()	11:13:55

Date Object (Cont.)

- **Date Methods (cont.)**

  2.set Methods

```
var newDate = new Date ();
var myDate= new Date ( "November 25,2006 11:13:55");
```

Method	Example
setDate(number)	newDate.setDate(25)
setHours(number)	newDate.setHours(14)
setMinutes(number)	newDate.setMinutes(50)
setMonth(number)	newDate.setMonth(7)
setSeconds(number)	newDate.setSeconds(7)
setTime(TimeString)	newDate.setTime(myDate.getTime())
setFullYear(number)	newDate.setFullYear(88)

Date Object (Cont.)

- **Date Methods (cont.)**

  2.to Methods

var myDate= new Date ("November 25,2006 11:13:55");

Method	Example	Returned value
toUTCString()	myDate.toUTCString()	Sat, 25 Nov 2006 09:13:55 GMT
toLocaleString()	myDate.toLocaleString( )	11/25/2006, 11:13:55 AM (Based on date format in your OS)
toLocaleTimeString()	myDate.toLocaleTimeS tring()	11:13:55 AM
toLocaleDateString()	myDate.toLocaleDateSt ring()	11/25/2006
toString()	myDate.toString()	Sat Nov 25 2006 11:13:55 GMT+0200
toDateString()	myDate.toDateString()	Sat Nov 25 2006

Regular expression Object

- A Regular Expression is a way of representing a pattern you are looking for in a string.
- A Regular Expression lets you build patterns using a set of special characters. Depending on whether or not there's a match, appropriate action can be taken.
- Regular expressions is often used for the purposes of validation.
- **Syntax**

/ pattern / [flag] ; **or** **new RegExp(“pattern” [, “flag”])**

- **Eaxmple:**

```
var myRegExp = new RegExp("j.*t", "i"); or
var myRegExp = /j.*t/i;
```

j.*t is the regular expression pattern. It means, "Match any string that starts with j, ends with t and has zero or more characters in between". The asterisk * means "zero or more of the preceding“. The dot (.) means "any character“.

Regular expression Object (Cont.)

- **Regular Expression Modifiers (Mode / Flag)**
  - Modifiers can be used to perform case-insensitive more global searches:
  - Optional parameter, indicates the mode in which the Regular Expression is to be used:

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)
m	Perform multiline matching

Regular expression Object (Cont.)

- **Regular Expression Patterns**

Character	Description	Example
.	Any character	/a.*a/ matches "aa", "aba", "a9qa", "a!?_a",
^	Start	/^a/ matches “apple”, “abcde”..
$	End	/z$/ matches “abcz”, “az”..
|	Or	/abc|def|g/ matches lines with "abc", "def", or "g"
[ ]	Match any one character between the brackets	/[a-z]/ matches any lowercase letter
[^]	Match any one character not between the brackets	/[^abcd]/ matches any character but not a, b, c, or d

Regular expression Object (Cont.)

- **Regular Expression Patterns (cont.)**

Character	Description	Example	
*	0 or more	/Goo*gle/ →"Gogle", "Google", "Gooogle", "Goooogle"	
+	1 or more	/Goo+gle/ →"Google", "Gooogle", "Goooogle"	
?	0 or 1	/Goo?gle/ →"Gogle", "Google",	
{min, max}	{min,} → min or more	{2,} 2 or more	/a(bc){2,4}/ →"abcbc", "abcbcbc", or "abcbcbcbc"
	{,max} → up to max	{,6} up to 6	
	{val} → exact value	{3} exactly 3	

- /d (any digit)
- () (group)

Regular expression Object (Cont.)

- **Regular Expression Methods**

  - **test()**

    It searches a string for a pattern, and returns true or false, depending on the result.

```
var reg=/j.*t/ ;
var t= reg.test("Javascript"); //false case sensitive
```

- **String Methods that Accept Regular Expressions as Parameters :**
  - match() - returns an array of matches.
  - search() - returns the position of the first match.
  - replace() - allows you to substitute matched text with another string.
  - split() - also accepts a RegExp when splitting a string into array elements.

Error Object

- Whenever an error occurs, an instance of the Error object is created to describe the error.

- **Error objects can be created in 2 ways:**
  - Explicitly:

```
var newErrorObj = new Error();
```

  - Implicitly:

    thrown using the throw statement

- **Error Properties**

Property	Description
fileName	URI of the file containing the script throwing the error
lineNumber	Source code line number of error
message	Plain-language description of error (ECMA)
name	Error type (ECMA)
number	Microsoft proprietary error number

Error Object (Cont.)

- Error Name Values

  Six different values can be returned by the error name property:

  var e = new Error();

Error Name	Description
EvalError	An error has occurred in the eval() function
RangeError	A number "out of range" has occurred
ReferenceError	An illegal reference has occurred
SyntaxError	A syntax error has occurred
TypeError	A type error has occurred
URIError	An error in encodeURI() has occurred

Using **instanceOf** when catching the error lets you know if the error is one of these built-in types.

Error Object (Cont.)

- Example:

```
var e = new EvalError('jaavcsritp is _not_ how you spell it');
document.write(e.name); //EvalError
document.write(e. message); // jaavcsritp is _not_ how you spell it
```

- Error Handling:
  1. try…catch
  2. onerror

1- try…catch

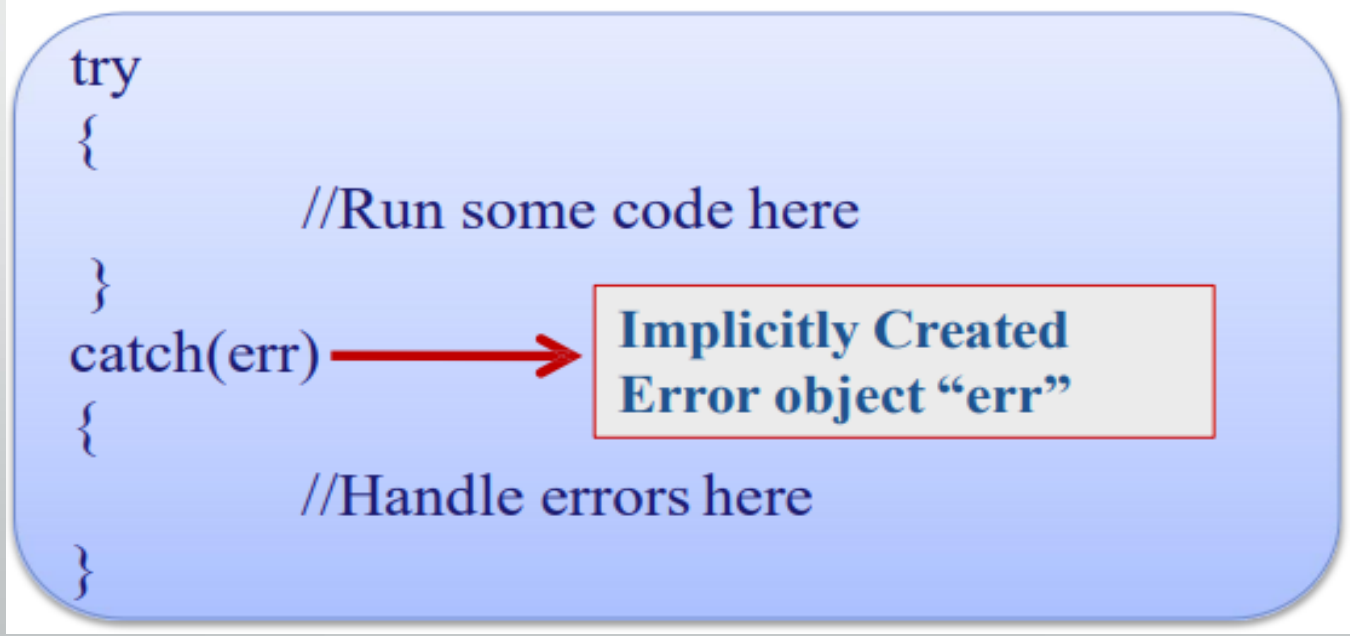
- The try...catch statement allows you to test a block of code for errors.
- The try block contains the code to be run.
- The catch block contains the code to be executed if an error occurs.

Error Object (Cont.)

- **Error Handling (cont.):**

  1- try…catch

  - The try...catch statement allows you to test a block of code for errors.
  - The try block contains the code to be run.
  - The catch block contains the code to be executed if an error occurs.



Error Object (Cont.)

- **Error Handling (cont.):**

  1- try…catch

  - If you have any functionality that needs to be processed regardless of success or failure, you can include this in the finally block (will executed even if there break, throw, or return statements in catch block.

```
try
{
        //Run some code here
 }
catch(err)
{
        //Handle errors here
}
finally
{
        //Here code that will be executed on both cases
}
```

Error Object (Cont.)

- **Error Handling (cont.):**

  1- onerror

  - The old standard solution to catch errors in a web page.
  - The onerror event is fired whenever there is a script error in the page.
  - onerror event can be used to retrieve additional information about the error.

```
onerror=handleErr
function handleErr(msg,url,l)
{
        //Handle the error here
                return true;
}
```

msg : Contains the message explaining why the error occurred.

url : Contains the url of the page with the errorscript .

l : Contains the line number where the error occurred

Error Object (Cont.)

- **Throw Error**
  - The throw statement allows you to create an exception.
  - Syntax:

```
throw(exception)
```

  - The exception can be a string, integer, Boolean or an object.

```
try{
        if(x<100)
                throw “less100"
        else if(x>200)
                throw “more200"
    }
catch(er){
                if(er==“less100")
                    alert("Error! The value is too low")
                if(er == “more200")
                    alert("Error! The value is too high")
          }
```

THANK YOU