Ecmascript Core Reference

By John Bentley

Key

This is text is for general narration.

The next paragraph.

This is a rule to follow.

```
This contains any code or other examples.
```

This justifies the rule.

> This is a quote that also justifies the rule.

This is the source for the rule (if there are not multiple source that refer to different parts of the example, justification, or justification quote).

Another rule to follow.

Concepts

ECMA Script Hosts, Object Models, and Contexts

EcmaScript only runs in a host. Generally this is a web browser or Windows Script host. The hosts expose object models. In a web browser this is the Document Object Model (the DOM). In windows script host the object model is called Windows Script Host object model.

EcmaScript, from a host, can access external object models, eg the FileSystemObject object model (available under windows)..

Version

The reference originally written at 12 Aug 2004 for ECMA - 262, 3rd Edition (December 1999).

http://www.ecma-international.org/publications/standards/Ecma-262.htm

ECMAScript version history at
https://en.wikipedia.org/wiki/ECMAScript#Versions

(Wikipedia 2019, "ECMAScript", https://en.wikipedia.org/wiki/ECMAScript)

Basics

Lines

Termination: it's good habit to terminate javascript statements with semicolons ";"

In javascript semicolons are optional but they might be required in the future.

*(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), P 51*

Continuation: a single statement may span multiple lines.

```
document.write("<p>" +
  "This paragraph " +
  "over multiple lines." +
  "</p>");
```

(Netscape 2000, "Core JavaScript Reference 1.5", https://tecfa.unige.ch/guides/js/jsref15/)

Case

EcmaScript is case-sensitive.

*(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), p 23*

Comments

```
// or /* ... */
```

Datatypes

The datatypes:

Datatype	**Examples**
Numbers	42, 3.1415, -3.1E12, .1e12, 2E-12

String	"Nice"
Boolean	true, false
Null	null
Undefined	undefined
Function	[A function definition]

There is no Date datatype. You use, instead, the inbuilt Date object and it's properties and methods.

Variables

The value type of a variable (eg string, number, boolean, etc) can change during execution.

(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), p 61*

Variables: global (outside a function) V local (inside a function); declarations without var are always global; always declare a variable with a var; global scope is for the document only.

(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), p 74*

Initialize variables.

```
// This works
var j = 0;
for (i = 0; i < 10; i++) {
  j += 2;
}
document.write("<p>Script surrounded by
  plain comments. J = " + j + " </p>");

// This doesn't work
var j;
for (i = 0; i < 10; i++) {
  j += 2;
}
document.write("<p>Script surrounded by plain
  comments. J = " + j + " </p>");
```

Constants

```
const prefix = '212';
```

The scope rules for constants are the same as those for variables.

Expressions

Expressions are evaluated left to right

```
3 + 3 + "3" // evaluates to "63"
```

*(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), p 66*

DataType Conversion

Convert a string to a number with parseInt() or parseFloat()

```
parseInt ("33.85") // evaluates to 33
```

*(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), p 66*

Convert a number to a string by concatenating an empty string.

```
("" + 2500) // evaluates to 2500
```

*(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), p 66*

Conversion	**Trick**	**Readable**
To Number	(+x)	Number(x) [slow]
To Boolean	!!x	Boolean(x)
To String	("" + x)	String(x);

(comp.lang.javascript n.d., "Notes on the Comp.Lang.Javascript FAQ", Accessed 2021-02-17. http://jibbering.com/faq/faq_notes/faq_notes.html#toc)
https://jibbering.com/faq/faq_notes/type_convert.html

Values

Undefined: A variable that has not been assigned a value is of type undefined.

```
undefinded ≡ false
```

A null value is interpreted to be the same as false in a condition, while the presence of any non-zero value is the same as true in a condition.

```
null ≡ false
non-zero ≡ true
```

(Goodman and Morrison 2004, **JavaScript Bible 5th Edition**), p 86

True V False

```
false: zero, null, undefined, zero-length string
   ("")
true: non-zero, a string with a character
   (including " ")
```

Literals

A string literal is zero or more characters enclosed in double (") or single (') quotation marks.

```
"foo"
'bar'
```

(Mozilla Developer Network 2021, "JavaScript Guide", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide) *https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#string_literals*

Special Characters

String special characters

Character	**Meaning**
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab
\'	Apostrophe or single quote
\"	Double quote
\\	Backslash character (\).
\XXX	The character with the Latin-1 encoding specified by up to three octal digits XXX between 0 and 377. For example, \251 is the octal sequence for the copyright symbol.
\xXX	The character with the Latin-1 encoding specified by the two hexadecimal digits XX between 00 and FF. For example, \xA9 is the hexadecimal sequence for the copyright symbol.
\uXXXX	The Unicode character specified by the four hexadecimal digits XXXX. For example, \u00A9 is the Unicode sequence for the copyright symbol.

Numbers

When rounding a number avoid using the native functions Number.toFixed() and Number.toPrecision(). Use the custom function Math.bankersRound().

The native rounding functions produces *different* results in different browers.

Value	toFixed(1) FireFox 1.0.6	toFixed(1) IE 6.0
4.55	*4.5*	*4.6*
4.65	4.7	4.7
-4.55	*-4.5*	*-4.6*
-4.65	-4.7	-4.7
Rounding Goal	Accuracy	Accuracy
Rounding Direction	Toward Odd (Bankers Round)	Away From Zero

*(Bentley 2021, *EcmaScript Library > Core*, C:\Users\John\Documents\Sda\Code\EcmaScript\Libraries\StandardLibrary\Core), roundingFunctions.html & Math.js*

Use the custom Math.bankersRound() instead of the native Math.round().

Math.round() only works to round integers. That is, you can't specify the number of decimal places. It also employs rounding towards positve infinity rather than bankers rounding:

Value	toRound(1) (both browsers)
-5.5	-5
-4.5	-4
-3.5	-3
4.5	5
5.5	6
Rounding Goal	Accuracy
Rounding Direction	Toward Positive Infinity

*(Bentley 2021, *EcmaScript Library > Core*, C:\Users\John\Documents\Sda\Code\EcmaScript\Libraries\StandardLibrary\Core), roundingFunctions.html*

The custom Math.bankersRound() function works like this:

Value	Custom Math.bankersRound(num,1)
4.55	4.6
4.65	4.6
-4.55	-4.6
-4.65	-4.6
Rounding Goal	Accuracy
Rounding Direction	Toward Even (Bankers Round)

(Bentley 2021, *EcmaScript Library > Core*, *C:\Users\John\Documents\Sda\Code\EcmaScript\Libraries\StandardLibrary\Core), Math.js*

Enums (Simulated)

Within an object instance, assign several properties their own number.

```
  var DatePrecisionEnum = {
  year: 1,
  month: 2,
  day: 4,
  full: 8,
};
...
switch (this.precision) {
  case DatePrecisionEnum.year:
    // format() as a custom function, not native javascript.
    dateString = this.realDate.format("yyyy");
    break;
  case DatePrecisionEnum.day:
    dateString = this.realDate.format("yyyy-MM-dd");
    break;
  case DatePrecisionEnum.full:
    dateString = this.realDate.toISOString();
    break;
  default:
    throw new Error('Unexpected case reached.');
}
```

C:\Users\John\Documents\Sda\Code\EcmaScript\Examples\Web\dateTimeDemo.xhtml

Statements

If

Ususal syntax.

```
if (condition) {
   statements1
}
[else {
   statements2
}]
```

Can do with the warning that "form is known to contribute to mistakes in projects where many programmers are working on the same code"

```
if (condition)
    statement1
[else
    statement2]
```

Douglas Crookford > JSLint The JavaScript Verifier (http://www.jslint.com/lint.html)

If inline syntax

```
(someVariable ? doThis : elseDoThis);
```

Switch

In a switch statement match multiple cases like this:

```
switch (selectedPlanetName){
  case "Saturn":
  case "Pluto":
    eval(selectedPlanetName + ".showPlanet()");
    break;
  default:
    planets[lst.options[i].text].showPlanet();
}
```

For

Basic For statement. Repeats until a specified condition evaluates to false.

```
for (var i = 0; i < regionalOffices.length; i++) {
  regionalOffices[regionalOffices[i].city] = regionalOffices[i];
}
```

(Mozilla Developer Network 2019, "JavaScript Reference", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference), "Loops and iteration", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration

> The for...in statement iterates a specified variable over all the enumerable properties of an object.

```
function dump_props(obj, obj_name) {
  var result = '';
  for (var i in obj) {
    result += obj_name + '.' + i + ' = ' + obj[i] + '<br>';
  }
  result += '<hr>';
  return result;
}
```

(Mozilla Developer Network 2019, "JavaScript Reference", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference), "Loops and iteration", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration

> The for...of statement creates a loop Iterating over iterable objects (including Array, Map, Set, arguments object and so on), invoking a custom iteration hook with statements to be executed for the value of each distinct property.

```
var arr = [3, 5, 7];
arr.foo = 'hello';

for (var i in arr) {
   console.log(i); // logs "0", "1", "2", "foo"
}

for (var i of arr) {
   console.log(i); // logs 3, 5, 7
}
```

(Mozilla Developer Network 2019, "JavaScript Reference", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference), "Loops and iteration", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Loops_and_iteration

Date Time

Date Time values are stored as M illiseconds since 1970-01-01T00:00:00+00:00. Date times before this are negative.

Local Formats, Browser Determined	
theDateTime [Defaults to toString()]	Mon Jan 1 21:30:00 UTC+1100 2001
toString()	Mon Jan 1 21:30:00 UTC+1100 2001
toDateString()	Mon Jan 1 2001
toTimeString()	21:30:00 UTC+1100
toISOStrong()	2001-01-01T10:30:00.000Z
Local Formats, System Determined	

toLocaleString()	Monday, 1 January 2001 21:30:00
toLocaleDateString()	Monday, 1 January 2001
toLocaleTimeString()	21:30:00
Universal Time Coordinated Formats	
toUTCString()	Mon, 1 Jan 2001 10:30:00 UTC
toGMTString()	Mon, 1 Jan 2001 10:30:00 UTC
Other Formats	
getTime() [milliseconds since 1970-01-01 00:00:00 UTC]	978345000485
getTimezoneOffset [Minutes offset from UTC]	-660

(Goodman and Morrison 2004, *JavaScript Bible 5th Edition**), p892*

Date Time values are generated based on the visitor's client, not on server settings.

(Goodman and Morrison 2004, *JavaScript Bible 5th Edition**), p892*

Creation

General.

```
Var dateObjectName = new Date([Parameters]);

new Date("Month dd, yyyy hh:nn:ss");
new Date("Month dd, yyyy");
new Date(yyyy,mm,dd,hh,nn,ss);
new Date(yyyy,mm,dd);
new Date(milliseconds);
```

Create negative, BCE, dates with two leading zeros.

```
var myDate = new Date("-000379");
```

Now

```
var now = new Date();
now.toISOString();

// One line
outptMessage(new Date().toISOString());
```

Formats

Custom Formats (By John Bentley)	
getDateTimeFile(theDate)	20010101_2130
getDateTimeIso(theDate)	2001-01-01T21:30:00
getDateTimeXslSafe(theDate)	20010101213000
getDateTimeLocalFriendly(theDate)	Mon, 01 Jan 2001 21:30 AUS Eastern Daylight Time (UTC+1100)
getOffsetAsHoursMinutes(-570)	+0930
getOffsetAsHoursMinutes(570)	-0930

Bentley > EcmaScriptLibrary > DateTime.js

Parsing and UTC

Parsings can be done in ISO format.

```
var myDatePrecise = new Date("2016-05-20T03:40:00+00:00");
var myDateImprecise = new Date("2016-05-20");
var myDateYearOnly = new Date("2016");

myDatePrecise.toISOString(); // etc.

2016-05-20T03:40:00.000Z
2016-05-20T00:00:00.000Z
2016-01-01T00:00:00.000Z
```

Bentley. 2016-05-20 20:47.
C:\Users\John\Documents\Sda\Code\EcmaScript\Examples\Web\dateTimeDemo.xhtml

Each Static function returns milliseconds.

```
var theDateTime1 = new Date(Date.parse("Jan 01,
  1970 10:59:59"));

var theDateTime2 = new
  Date(Date.UTC(1970,01,01,00,00,01,234));
```

(Goodman and Morrison 2004, *JavaScript Bible 5th Edition* *), p899*

You can use jquery UI datepicker.parseDate to extract a date in ISO format

```
// 'yy' really means 'yyyy' as in '2000'.
$.datepicker.parseDate('yy-mm-dd', '2007-01-26');
```

http://docs.jquery.com/UI/Datepicker/parseDate

Arithmetic

Work in milliseconds.

```
var oneMinute = 60 * 1000;
var oneHour = oneMinute * 60;
var oneDay = oneHour * 24;
var oneWeek = oneDay * 7;
```

(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), p900*

Arrays

Zero based; can contain different datatypes; can grow in size at any time.

Simple

Declare:

```
var states = new Array();

// Number of elements.
// No need to do this but you can.
var states = new Array(4);
```

Assign

```
states[0] = "vic";
states[1] = "nsw";
states[2] = "qld";
states[3] = "tas";

// You can do this but, by itself,
// makes looping through array with a
// numeric index immpossible. See Simulated Hash table bellow.
states["vicState"] = "vic";
states["nswState"] = "nsw";
states["qldState"] = "qld";
states["tasState"] = "tas";
```

Declare and Assign

```
// Streamlined avoids a bug.
var states = ["vic","nsw","qld","tas"];

var states = [];

var states = new Array("vic","nsw","qld","tas");
var columnHeads = "Qty,Desc,Price".split(",");
```

Declare And Assign with Object Initialiser.

```
var uranus =
  {
    name : "Uranus",
    day : "Happy",
    year : 3445,
    isCool : false,
    'HotFactor': 'boiling'
  };

var output = "";

output += "<p>Arrary Results:<ul>";
output += "<li>uranus['name'] = " + uranus['name'] + "</li>";
output += "<li>uranus['isCool'] = " + uranus['isCool'] + "</li>";
output += "<li>uranus['year'] = " + uranus['year'] + "</li>";
output += "<li>uranus['HotFactor'] = " + uranus['HotFactor'] + "</li>";
output += "</ul></p>";

document.writeln(output);
```

You can also use object initializers to create arrays…
The syntax for an object using an object initializer is:

objectName = {property1:value1, property2:value2,..., propertyN:valueN}

where objectName is the name of the new object, each property I is an identifier (either a name, a number, or a string literal),

Netscape > Core JavaScript Guide 1.5> Using Object Initializers

Delete Entry

Makes entry undefined, not removed.

```
delete states[2];
```

This completely removes

```
Array.prototype.cut = function(index) {
  if (index>=this.length) return;
  if (index==this.length-1) { this.length-=1; return; }
  for (var i=index+1; i<this.length; i++) this[i-1]=this[i];
  this.length-=1;
}

// Usage
states.cut(2);
```

See Bentley > EcmaScriptLibrary > (from comp.lang.javascript, Jon Gilkison, 1999/11/03, Re: remove array element from javascript array?)

Access

```
arr[3];
states["tas"];
```

Array of Objects

Object definition

```
function aircraft(typeCode, manufacturer, model,
  seats){

  this.typeCode = typeCode;
  this.manufacturer = manufacturer;
  this.model = model;
  this.seats = seats;
}
```

Create

```
var fleet = new Array()
```

Assign

```
fleet[0] = new aircraft("B734", "Boeing",
  "737-400", 168);
fleet[1] = new aircraft("C182R", "Cessna",
  "182RG", 4);
fleet[2] = new aircraft("BE58", "Beechcraft",
  "Baron 58", 6);
```

Access

```
document.write("<p>" + fleet[2].manufacturer +
  "</p>");
```

(Goodman and Morrison 2004, *JavaScript Bible 5th Edition**), p915*

Array of Arrays

Create

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

Assign

```
fleetArrayOfArrays[0] = new Array("B734",
  "Boeing", "737-400", 168);
fleetArrayOfArrays[1] = new Array();
fleetArrayOfArrays[2] = new Array("BE58",
  "Beechcraft", "Baron 58", 6);

fleetArrayOfArrays =
[ ["B737", "Boeing", "737-400", 168],
  ["C182RG", "Cessna", "182RG", 4],
  ["BE350", "Beechcraft", "King Air 58", 6] ]
```

Access

```
for (i=0; i < fleetArrayOfObjects.length; i++){
  document.write("<li>" );
  for(j=0; j<fleetArrayOfArrays[i].length; j ++)
  {
    document.write(
      fleetArrayOfArrays[i][j]+" ");
  }

  document.write("</li>");
}
```

*(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), p915*

Simulated Hast Table: access values through either numeric or string index.

```
states[0] = "vic";
states[1] = "nsw";
states[2] = "qld";
states[3] = "tas";
states["vicState"] = "vic";
states["nswState"] = "nsw";
states["qldState"] = "qld";
states["tasState"] = "tas";

// Or
for (var i = 0; i < regionalOffices.length; i++) {
  regionalOffices[regionalOffices[i].city] = regionalOffices[i];
}
```

*(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), p915*

Array Reference

For a full list see (Mozilla Developer Network 2019, "JavaScript Reference", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference), "Array", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Array.prototype.map() ... creates a new array with the results of calling a provided function on every element in the calling array.

```
var new_array = arr.map(function callback(currentValue[, index[, array]]) {
    // Return element for new_array
}[, thisArg])
```

```
var personsArray = ['John','George','Paul','Ringo'];
// John,George,Paul,Ringo

var personsWordLengthArray = personsArray.map(function(item) {
  return item.length;
})
// 4,6,4,5
```

C:\Users\John\Documents\Sda\Code\EcmaScript\Examples\Core\arrays.xhtml
(Mozilla Developer Network 2019, "JavaScript Reference", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference) https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map

Functions

Returning

The function is terminated on the first return encountered.

```
  function returnValue () {
    return 5;
    return 6;
  }
  function test () {
    alert(returnValue());
  }
// 5 is returned.
```

Creating

Three ways of creating a function:

1. Normal Function

```
function functionName([arg1] ... [,argN]) {
    statements(s)
}
```

2. Anonymous Function

```
var functionName = new Function (["argName1"
  ... [,"argNameN"],
  "stament1; ... [statementN]"]);

// Calling an Anonymous function like the normal
functionName([arg1] ... [,argN])
```

3. Lambda Expression: assign a function to an object property.

```
// Lambda Expression with an inline function definition: define function inline
and directly assign to an object property.
document.forms[0].txtAge.onchange =
  function (event){
    isNumber(document.forms[0].txtAge)
  }

// Lambda Expression with function name assignment: assign a function name to an
object property.
function doSomething() {
  alert("Something is happening");
}

function main() {
  document.forms[0].btnDoSomething.onclick =
    doSomething;
}
```

A function, the only type of subroutine, is capable of returning a value but is not required to.

(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), p 73*

Function Parameters and Arguments

Calling

You must call a function using brackets.

```
function getMyString(inside) {
  return "#" + inside + "#";
}

// Do this
var str = getMyString('Cool');

// Not this
var str = getMyString 'Cool';
```

C:\Users\John\Documents\Sda\Code\EcmaScript\Examples\Core\functionBrackets.xhtml

Optional Arguments

```
function setRequiredMark(jqueryElement, add) {
  if (add === undefined) {
    add = true;
  }
..
}
```

Parameter Arrays

Use the "arguments" array to access all arguments passed to a function regardless of whether there is explicit parameters or none.

```
doStuff('Now','this', 64, 'is', 'cool');

function doStuff() {
  var args = doStuff.arguments;
  var message = "";
  for (var i = 0; i < args.length; i++) {
    message += "Argument " + i + ": " + args[i] + "<br />";
  }
  document.getElementById("output").innerHTML = message;
}
```

(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), p984*

If you want an array to be interpreted as an argument list use 'apply':

```
function demoParamArray() {
  var places = new Array("New Zealand" ,"Thailand", "France");
  doStuff.apply(this, places);
}
```

*(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), p987*

Operators

Basic

Arithmetic Operators	**(+, -, *, /, %, ++, --, unary -, unary +)**
Assignment Operators	(=, *=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=, |=)
Bitwise Operators	(&, |, ^, ~, <<, >>, >>>)
Comparison Operators	(==, !=, ===, !==, >, >=, <, <=)
Logical Operators	(&&, ||, !)
String Operators	(+ and +=)
Member Operators	(object.property and object["property"])

https://developer.mozilla.org/en/Javascript/Reference

&& as well as || are short circuit evaluated.

This

The `this` Operator references either the containing xhtml object; or the custom object.

```
// Containing XHTML object
<input type="text" name="txtEntry
    onchange="process(this)" />

// Custom object, property assignment
// in constructor.
function aircraft(typeCode, manufacturer){
  this.typeCode = typeCode;
  this.manufacturer = manufacturer;
}

// Custom object, property reference within
// custom object method
function showAircraft() {
  return "Aircraft: " + this.manufacturer;
}
```

Classes and Objects

Create

Define and initialise using a constructor

Define a function to act as method.

```
function briefPlanetDetails() {
  var result = "";
  result += "Planet " + this.name
    + " Diameter: " + this.diameter;

  var fra = parent.fra2
  fra.document.write(result);
  fra.document.close();
}
```

Define Custom Object.

```
function planet(name, diameter, distance,
  year, day) {

  this.name = name;
  this.diameter = diameter;
  this.distance = distance;
  this.year = year;
  this.day = day;
  // Assign Method to a function
  this.showPlanet = briefPlanetDetails;
}
```

Create Object.

```
var Saturn = new planet("Saturn", 4500, 34,
  90, "15 hours");
```

Define and initialise without a constructor

Initialise by property and method assignment

Define and Create object without a constructor - initialise by property. More useful for single instances.

```
Pluto = new Object();
Pluto.name = "Pluto";
Pluto.diameter = 678;
Pluto.distance = 2348;
Pluto.year = 678;
Pluto.day = "34 hours, 2 nanoseconds";

// Must define a method if you are going to
// call it.
Pluto.showPlanet = outputPlanet;

// Or you can define a method on the fly
Pluto.farewell = function() { alert("Bye everybody!"); }
```

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Basics#Dot_notation

Initialise by object initialiser (object literal)

Useful for database or xml records.

```
var Uranus = {name:"Uranus", day:"Happy",
   diameter:1234, distance:342, year:3445,
   showPlanet:outputPlanet,
   describePlanet: function () { alert(this.name + ' ' + this.diameter}
}
```

*(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), p1001*

Constructor default values

Define default values for an object.

```
function aircraft(manufacturer, model, variant,
  typeCode, range) {

  this.manufacturer = manufacturer || "Boeing";
  this.model        = model        || "737";
  this.variant      = variant      || "- 400";
  this.typeCode     = typeCode     || "B734";
  this.range        = range        || 3813;
}
```

*(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), p1004*

Access

Access an object properties externally (from outside the class) through either: dot notation; or brack notation.

```
// Define object
const person = {
  ...
  age: 32,
  ...
};

// Dot notation
addOutput(person.age);
```

```
// Bracket notation
addOutput(person['age']);
```

(Mozilla Developer Network 2019, "JavaScript", https://developer.mozilla.org/en-US/docs/Web/JavaScript), Learn web development, JavaScript, Introducing JavaScript objects, JavaScript object basics, Dot Notation, https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Basics#Dot_notation C:\Users\John\Documents\Sda\Code\EcmaScript\Examples\Core\objects.xhtml

Access an objects isntance members, properties and methods, internally (from within the class) using the `this` keyword.

```
const person2 = {
  name: 'Brian',
  greeting: function() {
    alert('Hi! I\'m ' + this.name + '.');
  }
}

person2.greeting()

// "Hi! I'm Brian."
```

(Mozilla Developer Network 2019, "JavaScript", https://developer.mozilla.org/en-US/docs/Web/JavaScript), Learn web development, JavaScript, Introducing JavaScript objects, JavaScript object basics, What is "this"? https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Basics#What_is_this C:\Users\John\Documents\Sda\Code\EcmaScript\Examples\Core\objects.xhtml

Object of objects

Create an object containing another object.

```
// Create
const person = {
  name : {
    first: 'Bob',
    last: 'Smith'
  },
  age: 32,
  gender: 'male',
  interests: ['music', 'skiing'],
  ...
};
// name is an object.

// Access via dot notation
person.name.first
person.name.last

// Access via dot notation
person['name']['first']
person['name']['last']
```

(Mozilla Developer Network 2019, "JavaScript", https://developer.mozilla.org/en-US/docs/Web/JavaScript) "Object Basics", "Dot Notation", https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Basics#Dot_notation

Object Member Scope

The scope, private V public, of an object's members, properties and methods, can be achieved with the following ...

```

// declaration
function myClass(){

    // variables
    var privateVar = "I am private";
    this.publicVar = "I am public";

    // functions
    var privateFunction = function(){
        alert("I am private");
    }

    this.publicFunction = function(){
        alert("I am public");
    }
}

// usage
```

```
var myInstance = new myClass();

myInstance.publicVar = "new value";
alert(myInstance.privateVar);    // undefined!

myInstance.publicFunction();
myInstance.privateFunction();      // error!
```

Pavel Simakov > 26 September, 2005 > Javascript Refactoring For Safer Faster Better AJAX
http://www.softwaresecretweapons.com/jspwiki/Wiki.jsp?page=JavascriptRefactoringForSaferFasterBetterAJAX

Use self, by convention, to allow private functions to access public members

```
// declaration
function myClass(){
    // Instance Members
    // variables
    var self = this;
    var privateVar = "I am private";
    this.publicVar = "I am public";

    // functions
    var privateFunction = function(){
        // publicVar undefined.
        alert("Access publicVar In privateFunction. this- " + this.publicVar);

        // publicVar is accessible.
        alert("Access publicVar In privateFunction. self- " + self.publicVar);
        return "I am private";
    }

    this.publicFunction = function(){
        privateFunction();
        return "I am public";
    }
}
```

Douglas Crockford 2001 > Private Members in JavaScript >
http://www.crockford.com/javascript/private.html

Instance V Static methods

```
// declaration
function myClass(){
    // Instance Members
    // variables
    var privateVar = "I am private";
    this.publicVar = "I am public";

    // functions
    var privateFunction = function(){
        return "I am private";
    }

    this.publicFunction = function(){
        return "I am public";
    }
}

// Static Member declaration
myClass.staticProperty = 4;
myClass.staticMethod = function() {
  return "String there at any time";
}
// Invoke
alert("myClass.staticProperty")
alert("myClass.staticMethod()");

// Instance member declarations
myClass.prototype.staticMethodWithArg = function(arg) {
  outPairLine("the arg ", this.publicVar + " " + arg);
}

// Invoke
var myInstance = new myClass();
myInstance.staticMethodWithArg("myArg");
```

Add properties or methods *bookmark*

Add a property or method to an existing object instance

```
fleet["B744"].slogan = "Fat Jumbo";
```

Add a property or method to an existing class.

```
aircraft.prototype.slogan =
  "Fly high in the sky."
// All aircraft now have this slogan
```

If you change the value of a constructor's prototype property then any instance variable property retains their setting.

```
// Instance property set
fleet["B744"].slogan ="Fat Jumbo";

// Object Property added and set
aircraft.prototype.slogan = "Fly high in the sky."
// B744.slogan is still "Fat Jumbo"
```

(Goodman and Morrison 2004, *JavaScript Bible 5th Edition**), p1004*

Callback Function

```
function coolClass(callbackFunction) {
  this.coolFunction = function () {
    callbackFunction(8);
  }
}

// This function must have a specific
// argument signature: that defined
// by the class.
function calledBack(secretNumber) {
  alert("The Secret Number is: " + secretNumber);
}

var coolObject = new coolClass(calledBack);
coolObject.coolFunction();
```

Bentley > EcmaScriptLibrary > callbackFunctions.html

Inheritance

To derive an object from another, in the derived class: set a property to the object; then call the constructor of the base class.

```
// Base Class
function aircraft(manufacturer, model, variant,
  typeCode, range) {

  this.manufacturer = manufacturer || "Boeing";
  this.model        = model        || "737";
  this.variant      = variant      || "- 400";
  this.typeCode     = typeCode     || "B734";
  this.range        = range        || 3813;
  this.getName = getAircraftName;
  this.showAircraft = showAircraft;
}

// Derived Class
function jet(manufacturer, model, variant, typeCode, range, seats, thrust) {
  this.aircraftInfo = aircraft;
  this.aircraftInfo(manufacturer, model,
    variant, typeCode, range);
  this.seats = seats || "missing";
  this.thrust = thrust || "missing";
}
```

To get derived instances to inherit properties and methods that are set on the base class using "prototype" do the following. Explicitly connect the prototype of the derived class to an instance of a base class.

```
// Base Class
function aircraft(manufacturer, model, variant, typeCode, range) {
  this.manufacturer = manufacturer || "Boeing";
  this.model        = model        || "737";
  this.variant      = variant      || "- 400";
  this.typeCode     = typeCode     || "B734";
  this.range        = range        || 3813;
  this.getName = getAircraftName;
  this.showAircraft = showAircraft;
}

// Derived Class
function jet(manufacturer, model, variant, typeCode, range, seats, thrust
  this.aircraftInfo = aircraft;
  this.aircraftInfo(manufacturer, model,
    variant, typeCode, range);
  this.seats = seats || "missing";
  this.thrust = thrust || "missing";
}

// Create Object.
var fleet = new Array();

// Explictily connect derived prototype to an instance of the base class.
jet.prototype = new aircraft();

fleet[0] = new aircraft("Cessna", "Skylane",
  "C182RG", "C182RG", 2102);
```

```
fleet[1] = new jet(null, "747", null, "B744",
  3566, 510, 145666);
fleet[2] = new jet("Learjet", "Learjet", "45",
  "LJ45", 4074, 16, 34000);
fleet[3] = new aircraft("Beechcraft", "Baron",
  "58", "BE58", 2715);
// B734 from default values
fleet[4] = new jet();
```

*(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), p1004*

Also .... ??

```
BjaxerNodeList.prototype = new Array();
BjaxerNodeList.prototype.constructor = Array;
```

See sarissa_ieemu_xpath.js

Extend Classes and Objects

Extend a native or custom object instance or class with a new property or new method. Do this by assigning a custom property to a value or "lambda" expression (inline or function name).

Extend Object (Instance)

Extend Object Instance with a property.

```
// Create Object Instance.
var myBirthday = new Date(71, 02, 04)

// Extend Object Instance
myBirthday.cake = "dope";

// Invoke.
document.writeln("The Cake: " + myBirthday.cake);
```

Bentley > EcmaScriptLibrary > extendingObjectsAndClasses.html
*(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), p167*

Extend Object Instance with a method.

```
// Create Object Instance.
var myBirthday = new Date(71, 02, 04)

// Extend object instance using either "lambda expressions":
// Here we use an inline function assignment.
// We could have used a function name assignment.
myBirthday.happyFormat = function(formatType) {
  if (formatType == "good") {
    return this.toDateString() + " It's your birthday!";
  } else {
    return  " Angst.";
  }
}
```

```
// Invoke.
document.write(myBirthday.happyFormat("good"));
```

Bentley > EcmaScriptLibrary > extendingObjectsAndClasses.html
(Goodman and Morrison 2004, *JavaScript Bible 5th Edition**), p167*

Extend Class Method

Extend Class with a method. All object instances of the class can now use the method.

```
// Create Object Instance.
var christmas = new Date(71, 11, 25);

function coolFormat(formatType) {
  if (formatType == "drugs") {
    return this.toDateString() + " Wow I can't see! <br />";
  } else {
    return  " normal.";
  }
}
// Extend object instance using either "lambda expressions":
// Here we use a function name assignment.
// We could have used an inline function assignment.
Date.prototype.crazyFormat = coolFormat;

// Invoke.
document.write(christmas.crazyFormat("drugs"));

var now = new Date();
// All object instances can now use the custom Method.
document.writeln(now.crazyFormat("drugs"));
```

Bentley > EcmaScriptLibrary > extendingObjectsAndClasses.html

Extend Class Property

Extend a native or custom class property, in NN6+, by using the following syntax. Define both a getter and a setter even for readonly or writeonly properties.

```
//Works in NN6+ (including Mozilla).
HTMLElement.prototype.__defineGetter__(
  "innerText", function () {

  var rng = document.createRange();
  rng.selectNode(this);
  return rng.toString();
})

HTMLElement.prototype.__defineSetter__(
  "innerText", function (txt) {

  var rng = document.createRange();
  rng.selectNodeContents(this);
  rng.deleteContents();
  var newText = document.createTextNode(txt);
  this.appendChild(newText);
  return txt;
})

// Invoke
alert(document.getElementById('result').innerText);
```

*(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), p1002*

You can't override existing properties (Eg document.getElementById('result').innerHTML). ?

Define Getters and Setters for object Properties.

```
js> o = new Object;
[object Object]
js> o = {a:7, get b() {return this.a+1; }, set c(x) {this.a = x/2}};
[object Object]
js> o.a
7
js> o.b
8
js> o.c = 50
js> o.a
25
js>
```

```
js> var d = Date.prototype;
js> d.year getter= function() { return this.getFullYear(); };

js> d.year setter= function(y) { return this.setFullYear(y); };

These statements use the getter and setter in a Date object:

js> var now = new Date;
js> print(now.year);
2000
js> now.year=2001;
987617605170
js> print(now);
Wed Apr 18 11:13:25 GMT-0700 (Pacific Daylight Time) 2001
```

http://devedge-temp.mozilla.org/library/manuals/2000/javascript/1.5/guide/obj.html#1018325

Namespaces

```
var Lib = {};
Lib.YourNamespace =  function (){};
Lib.YourNamespace.cool = function () {
  return "cool string";
};
window.onload = function () {
    document.getElementById("output").innerHTML =
      Lib.YourNamespace.cool();
};
```

See: C:\Data\Sda\Code\EcmaScript\Libraries\StandardLibrary\EcmaScriptLibrary\namespaces.html

Error Handling

Intro

In ECMAScript Let's speak of "Errors" rather than "Exceptions"

In Javascript "Exception" and "Error" are intermingled.

(Goodman and Morrison 2004, *JavaScript Bible 5th Edition**), p949*

Errors have names associated with them

```
catch (err) {
  switch (err.name) {
    case "INVALID_CHARACTER_ERR":
...
```

Browsers, by default, are designed to hide ECMAScript errors from the user. Fine.

Only use a try-catch-finally clause for the least number of statements as possible.

Always ensure an unexpected error goes to; if you are going to report an error report it to:
1. the User to Developer Error Reporter; AND
2. the Broswer Error Reporter.

Implementing Web Error Handling

Case 1: Report Errors without using a try-catch-finally clause.

Report to the browser error reporter: do nothing.

```
...
// error producing statement
var x = errorOutsideTryClause;
...
```

Most often this is all the error handling you need to do!

This error will just be displayed in the browsers error reporter. For Mozilla & Mozilla firefox this will be displayed in the JavaScritp console. For MS IE this will be displayed after you click on the error icon in the left side of the bottom status bar.

Report also to the User-to-Developer error reporter:

```
// 1. Copy  ~\WebScriptLibrary\errorHandling.js
// to your directory
// 2. In your .html file reference
// errorHandling.js.
<script type="text/javascript" src="errorHandling.js"></script>

  // 3. Somewhere in the execution path:
  ...
  function main() {
    // Error still gets reported by the browsers
    // default error reporter.
    window.onerror = reportErrorUserToDeveloper;

  }
  //-->
  </script>
  </head>
<body onload="main()">
...
```

Case 2: Throw and Catch custom errors (err.name & err.message not supported by IE5.0. Therefore, don't use this technique until IE5.0 is no longer a factor):

As of 20 Jun 2004 using err.name is useless for catching browser DOM errors.

Too many discrepancies. You can, however, use it for custom errors.

(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), p953*

Using navtive Error object. Throw and Catch in Web, simple, design pattern.

```
function getLetterSimple(letterIndex) {
  try {
    var inp = parseInt(letterIndex, 10);
    if (inp < 1 || inp > 5) {
      throw new Error("Enter only 1 through 5.");
    }
  }
  catch (err)  {
    // Throwing an error 3: Catch the specific error using err.message
    switch (err.message) {
      case "Enter only 1 through 5." :
            alert(err.message + " And Don't do it again!");
            break;
      default:
        // Throwing an error 4: This ensures the unanticpated error is reported
        // to the Browser Error Reporter
        throw err;
    }

    return false;
  } finally {
    // Executes no matter what.
  }

  // Executes if error not thrown & no prior 'return' executed.
  return letters[inp - 1];
}
```

See C:\Data\Sda\Code\EcmaScript\Libraries\StandardLibrary\WebScriptLibrary\errorHandling.html

Using: custom getErrorObj(). Throw and Catch in Web, full, design pattern.

```
// 1. Copy  ~\WebScriptLibrary\errorHandling.js
// to your directory
// 2. In your .html file reference
// errorHandling.js.
<script type="text/javascript" src="errorHandling.js"></script>

<!-- This defines:
// Throwing an Error 1: define getErrorObj
// Summary: Use this to throw errors.
// Example:
//          if (inp < 1 || inp > 5) {
//            throw new getErrorObj("Enter only 1 through 5.");
//          }
function getErrorObj(message) {
  var err = new Error(message);
  err.name = "MyError";
  return err;
}
-->

<script type="text/javascript">
var letters = new Array("A","B","C","D","E");

function getLetterFull(letterIndex) {

      try {
        var inp = parseInt(letterIndex, 10);
        if (isNaN(inp)) {
          // Throwing an error 2: Throw a getErrorObj
          throw getErrorObj("Entry was not a number.");
        }

        if (inp < 1 || inp > 5) {
          throw new getErrorObj("Enter only 1 through 5.");
        }
      }
      catch (err)  {
        // Throwing an error 3: Catch the custom errors with err.name
        // then specific custom errors with err.message.
        switch (err.name) {
          case "MyError" :
            switch (err.message) {
              case "Entry was not a number." :
                // We intercept the error, inform the user, and recover.
                alert(err.message + " Enter an Alphabetic!");
                break;

              // Treat all remaining custom errors in the same manner.
              default:
                alert(err.message);
                break;
            }
            break;
        default:
          // Throwing an error 4: This ensures the unanticpated error is
reported
          // to the Browser Error Reporter
          throw err;
        }
        return false;
      } finally {
        // Executes no matter what.
      }

      // Executes if error not thrown & no prior 'return' executed.
      return letters[inp - 1];
    }
```

Custom error messages caught do not get reported. They get "recovered from" which might include a message box to the user. Unanticipated Errors *are* reported to the broswer error reporter and, if programmed, the User-To-Developer reporter.

See C:\Data\Sda\Code\EcmaScript\Libraries\StandardLibrary\WebScriptLibrary\errorHandling.html

*(Based on) (Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), p956\Listing 31-10*

Using: custom getErrorObj(). Throw and Catch in Web, simple, design pattern.

```
function getLetterSimple(letterIndex) {
  try {
    var inp = parseInt(letterIndex, 10);
    if (inp < 1 || inp > 5) {
      throw new getErrorObj("Enter only 1 through 5.");
    }
  }
  catch (err)  {
    // Throwing an error 3: Catch the specific error using err.message
    switch (err.message) {
      case "Enter only 1 through 5." :
            alert(err.message + " And Don't do it again!");
            break;
      default:
        // Throwing an error 4: This ensures the unanticpated error is reported
        // to the Browser Error Reporter
        throw err;
    }

    return false;
  } finally {
    // Executes no matter what.
  }

  // Executes if error not thrown & no prior 'return' executed.
  return letters[inp - 1];
}
```

See C:\Data\Sda\Code\EcmaScript\Libraries\StandardLibrary\WebScriptLibrary\errorHandling.html

Case 3: Report on uncaught errors in the catch clause.

Just use "throw err" to report to the Browser Error Reporter.

From above:

```
       catch (err)  {
          ...
          default:
            // Throwing an error 4: This ensures the unanticpated error is 
reported
            // to the Browser Error Reporter.
            throw err;
          }

          return false;
      } finally {
```

... And use the following to also report to the User-To-Developer Error Reporter:

```
  // 3. Somewhere in the execution path:
  ...
  function main() {
    // Error still gets reported by the browsers
    // default error reporter.
    window.onerror = reportErrorUserToDeveloper;

  }
  //-->
```

Implementing Windows Script Host Error Handling

Throw and Catch in Windows Script Host design pattern (*fix)

```

var letters = new Array("A","B","C","D","E");

// Throwing an Error 1: define getErrorObj
// Summary: Use this to throw errors.
// Example:
//           if (inp < 1 || inp > 5) {
//             throw new getErrorObj("Enter only 1 through 5.");
//           }
function getErrorObj(msg) {
  var err = new Error(WScript.ScriptName + " Error: " + msg);
  err.name = "My_Error";
  return err;
}

function myFunction() {
  try {
    // Error prone statements.

  } catch (err) {
    switch (err.description) {
      case "Specific Error Description":
        break;
      default:
        // Report unhandled error to interface
        throw err;
    }
    return false;
  } finally {
    // Executes no matter what.
  }
  // Executes if error not thrown.
  return true;
}
```

Regular Expressions

Cheat Sheet

An alternate cheat sheet.

(Mozilla Developer Network 2021, "Regular Expression Syntax Cheatsheet - JavaScript | MDN", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions/Cheatsheet)
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions/Cheatsheet

Metacharacters

Character Type Metacharacters

Character	**Matches**	**Example**
\d	Numeral 0 through 9	/\d\d\d/ matches “212” and “415” but not “B17”
\D	Non-numeral	/\D\D\D/ matches “ABC” but not “212” or “B17”
\s	Single white space	/over\sbite/ matches “over bite” but not “overbite” or “over bite”
\S	Single non-white space	/over\Sbite/ matches “over-bite” but not “overbite” or “over bite”
\w	Letter, numeral, or underscore	/A\w/ matches “A1” and “AA” but not “A+”
\W	Not letter, numeral, or underscore	/A\W/ matches “A+” but not “A1” and “AA”
.	Any character except newline	/.../ matches “ABC”, “1+3”, “A 3”, or any three characters
[...]	Character set	/[AN]BC/ matches “ABC” and “NBC” but not “BBC”
[x-x]	Character set Range	[0-9] matches any digit.
[^...]	Negated character set	/[^AN]BC/ matches “BBC” and “CBC” but not “ABC” or “NBC”

Not to be confused with the metacharacters listed above are the escaped string characters for tab (\t), newline (\n), carriage return (\r), formfeed (\f), and vertical tab (\v).

*(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), BC194*

To match a metacharacter literally escape with a backslash "\"

```
/\./ matches the period rather than any character.
```

Boundary/Positional Metacharacters

Character	Matches	Example
\b	Word boundary	/\bor/ matches “origami” and “or” but not “normal” /or\b/ matches “traitor” and “or” but not “perform” /\bor\b/ matches full word “or” and nothing else
\B	Word non-boundary	/\Bor/ matches “normal” but not “origami” /or\B/ matches “normal” and “origami” but not “traitor” /\Bor\B/ matches “normal” but not “origami” or “traitor”
^	At beginning of a string	/^Fred/ matches “Fred is OK” but not “I’m with Fred” or line or “Is Fred here?”
$	At end of a string or line	/Fred$/ matches “I’m with Fred” but not “Fred is OK” or “Is Fred here?”

(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), BC194*

For example, you may want to make sure that a match for a Roman numeral is found

Counting Metacharacters

Character	Matches Last Character	Example
*	Zero or more times	/Ja*vaScript/ matches “JvaScript”, “JavaScript”, and “JaaavaScript” but not “JovaScript”
?	Zero or one time	/Ja?vaScript/ matches “JvaScript” or “JavaScript” but not “JaaavaScript”
+	One or more times	/Ja+vaScript/ matches “JavaScript” or “JaavaScript” but not “JvaScript”
{n}	Exactly n times	/Ja{2}vaScript/ matches “JaavaScript” but not “JvaScript” or “JavaScript”
{n,}	n or more times	/Ja{2,}vaScript/ matches “JaavaScript” or “JaaavaScript” but not “JavaScript”
{n,m}	At least n, at most m time	s /Ja{2,3}vaScript/ matches “JaavaScript” or “JaaavaScript” but not “JavaScript”

(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), BC194*

Make the search "non-greedy".

Operators

Character	Meaning	Example
x*? x+?	By default quantifiers like * and + are "greedy", meaning that they try to match as much of the	given a string like "some <foo> <bar> new </bar> </foo> thing"

x?? x{n}? x{n,}? x{n,m}?	string as possible. The ? character after the quantifier makes the quantifier "non-greedy": meaning that it will stop as soon as it finds a match.	/<.*>/ will match "<foo> <bar> new </bar> </foo>" /<.*?>/ will match "<foo>"

(Mozilla Developer Network 2021, "Regular Expression Syntax Cheatsheet - JavaScript | MDN", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions/Cheatsheet) *https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions/Cheatsheet*

Backreferences

Character	**Meaning**	**Example**
\n	Matches a previously captured group in the search string. Match groups start from 1 (the whole match)	/ (B)A\1E/ matches 'BABE', but not 'BASE'

$n	Matches a previously captured group in the replace string.
	
	Match groups start from 1 (the whole match)
		var str = 'nice';
		var re = /(n)ic(e)/;
		str.replace(re," $1$2t");
		
		// net

Flags

Flag	**Meaning**
g	Match all instances not just the first. (For Replace Operation only).
i	Peform a case-insensitive match.
m	A multiline input string should be treated as multiple lines. If the m flag is used, ^ and $ match at the start or end of any line within the input string instead of the start or end of the entire string.

Netscape > Core JavaScript Guide 1.5

Properties and Methods

All the following objects (and their properties) will be effected by a search.

Instance object properties:

Properties	*Methods*

re.source
re.global
re.ignoreCase
re.lastIndex
	compile()
	exec()
	test()

The static RegExp object properties reveal what, if any, regular expression pattern matching has just taken place in the window.

RegExp Object Properties

Properties	*Methods*

// String to search
RegExp.input
RegExp.multiline
RegExp.lastMatch
RegExp.lastParen
 	
// Before match
RegExp.leftContext
 	
// After match
RegExp.rightContext
 	
RegExp.$1
...
RegExp.$9	

Object returned by re.exec method

```
matchArray[0]
matchArray.index   // Starting index of match
matchArray.input
```

matchArray is null if no match found.

Coding

Basics

Properly implemented from WinIE5.5.

Create either using:

1. A regex literal with forward slashes "/".

```
var myRegExpression = /hello/gi;
```

2. A the regex constructor.

```
var myRegExpression = new Regex('hello', 'gi');
```

3. A regex literal passed to the regex constructor (Starting with ECMAScript 6).

```
var myRegExpression = new Regex(/hello/, 'gi');
```

*(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), BC192*
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp

Regular expression matching is case-sensitive by default.

*(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), BC193*

Flags.

```
// search for 'web' globally and
// case insensitively.
var re = /web/gi;

// or
re = new RegExp("web", "gi");
re = new Regexp(/web/, "gi");
```

*(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), BC193*

Performing Searches

```
var matchArray = re.exec(mainString);

// Shortcut syntax for above
var matchArray = re(mainString);
```

The parentheses around the segments of the regular expression instruct JavaScript to assign each found value to a slot in the matchArray object. Starting index 1. Index 0 refers to entire match.

```
// String to Search
var stringToSearch = "One day 'cunt' will not be taboo 342 and 563free speech
will be too cool."

// Regular Expression.
var re = new RegExp("(ay).*(ll).*(\d\d)")

// Perform matching
var matchArray = re.exec(stringToSearch)
if (matchArray == null) {
  throw new Error("No match in " + stringToSearch);
}

// Results
```

```
matchArray[0] // "ay 'cunt' will not be taboo 342 and 563"
matchArray[1] // "ay"
matchArray[2] // "ll"
matchArray[3] // "63"
```

(Goodman and Morrison 2004, *JavaScript Bible 5th Edition*), BC202*

Basic Matching

Boolean test. Returns false if no match.

```
regexObject.test(string)

// Test if 'john' in string
/.*(john).*/.test(string);
```

Index test. Returns -1 if no match.

```
string.search(regexObject)
```

Pure string method. Returns -1 if no match.

```
stringToSearch.indexOf("stringToFind")
```

Replace

Simple

```
var path = "/some/path";
path = path.replace("/","");
// output: some/path
```

The first argument of String.prototype.replace() either takes a:

1. Regex literal.

```
str = str.replace(/(\s+)/g,'#');
```

2. A regex variable.

```
var re = /(\s+)/g;
str = str.replace(re,'#');
```

3. A regex string.

```
str = str.replace('(\s+)','#','g');
```

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/replace
C:\Users\John\Documents\Sda\Code\EcmaScript\Examples\CoreExamples\regex.xhtml

$1 ... $9 stand for the capture patterns, defined by the parentheses "(,)" in the regular expression.

```
function insertCommas() {
  var frm = document.forms.frmMain;
  var re = /(-?\d+)(\d{3})/;
  var strLargeNumber = frm.txtEntry.value;
  while (re.test(strLargeNumber)) {
     strLargeNumber = strLargeNumber.replace(re,"$1,$2");
  }
  frm.txtResult.value = strLargeNumber;
}
```

Use options like g for global.

```
function removeCommas() {
  var frm = document.forms["frmMain"];
  // g means global: replace all instances not just the first.
  var re = /,/g;
  var subjectString = frm.txtEntry.value;
  subjectString = subjectString.replace(re,"");
  frm.txtRemoveResult.value = subjectString;
}
```

Inline Syntax

```
  // Reduce multiple spaces down to one.
  // g means global: replace all instances not just the first.
  subjectString = subjectString.replace(/(\s+)/g,' ');
```

Script Example Template

Script example template [master is the linked file in the source].

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
<head>
  <meta charset="utf-8" />
  <title>Script Example Template</title>

  <!-- <link rel="stylesheet" href="../web/webScriptExamples.css" /> -->

  <script>
  /* <![CDATA[ */

    function addOutput(message) {
      var outputElement = document.getElementsByTagName("output")[0];
      outputElement.innerHTML += message + "<br />";
    }

    function main () {
      addOutput("Fun");
    }

    window.onload = main;

  /* ]]> */
  </script>
</head>
<body>
  <h1>Script Example Template</h1>
  <p>Output:</p>
  <output></output>
</body>
</html>
```

C:\Users\John\Documents\Sda\Code\EcmaScript\Examples\Core\script-example-template.xhtml

See Also

An essential part of this quick reference is:

(Goodman and Morrison 2004, **JavaScript Bible 5th Edition**) > Appendix A: JavaScript and Browser Object Quick Reference > Core JavaScript/JScript/ECMAScript Quick Reference.

Edition additions

Lookup

For additions by Editions ...

*(Hoban [2014] 2019, *Overview of ECMAScript 6 Features.*, https://github.com/lukehoban/es6features)* https://github.com/lukehoban/es6features

(ECMA International 2018, "ECMAScript 2018 Language Specification: ECMA-262, 9th Edition, June 2018", http://www.ecma-international.org/ecma-262/9.0/index.html), "Introduction", http://www.ecma-international.org/ecma-262/9.0/index.html

5th

Object/Class features

Accessor properties. Write methods to set or get a property.

```
var obj = {
    get foo() {
        return 'getter';
    },
    set foo(value) {
        console.log('setter: '+value);
    }
};

// Here’s the interaction:

> obj.foo = 'bla';
setter: bla
> obj.foo
'getter'
```

*(Rauschmayer 2014, *Speaking JavaScript: An In-Depth Guide for Programmers*, http://speakingjs.com/es5/index.html) " Accessors (Getters and Setters)",* http://speakingjs.com/es5/ch17.html#getters_setters

Reflective creation and inspection of objects; Program control of propery attributes.

Category	**New Method**
Getting and setting prototypes	
	Object.create()
	Object.getPrototypeOf()
Managing property attributes via property descriptors	
	Object.defineProperty()
	Object.defineProperties()

	Object.create()
	Object.getOwnPropertyDescriptor()
Listing properties	
	Object.keys()
	Object.getOwnPropertyNames()
Protecting objects	
	Object.preventExtensions()
	Object.isExtensible()
	Object.seal()
	Object.isSealed()
	Object.freeze()
	Object.isFrozen()

*(Rauschmayer 2014, *Speaking JavaScript: An In-Depth Guide for Programmers*, http://speakingjs.com/es5/index.html), "Chapter 25. New in ECMAScript"* *http://speakingjs.com/es5/ch25.html*

New Function method:

```
Function.prototype.bind()
```

*(Rauschmayer 2014, *Speaking JavaScript: An In-Depth Guide for Programmers*, http://speakingjs.com/es5/index.html), "Chapter 25. New in ECMAScript"* *http://speakingjs.com/es5/ch25.html*

New type specific methods

New type methods.

Addiontal array manipulation
Array.isArray()
Array.prototype.every()
Array.prototype.filter()
Array.prototype.forEach()
Array.prototype.indexOf()
Array.prototype.lastIndexOf()
Array.prototype.map()
Array.prototype.reduce()
Array.prototype.some()

*(Rauschmayer 2014, *Speaking JavaScript: An In-Depth Guide for Programmers*, http://speakingjs.com/es5/index.html), "Chapter 25. New in ECMAScript"* *http://speakingjs.com/es5/ch25.html*

JSON support
JSON.parse()
JSON.stringify()
Boolean.prototype.toJSON()

`Boolean.prototype.toJSON()`
`String.prototype.toJSON()`
`Date.prototype.toJSON()`

*(Rauschmayer 2014, *Speaking JavaScript: An In-Depth Guide for Programmers*, http://speakingjs.com/es5/index.html), "Chapter 25. New in ECMAScript"* *http://speakingjs.com/es5/ch25.html*

Dates
`Date.now()`
`Date.prototype.toISOString()`

*(Rauschmayer 2014, *Speaking JavaScript: An In-Depth Guide for Programmers*, http://speakingjs.com/es5/index.html), "Chapter 25. New in ECMAScript"* *http://speakingjs.com/es5/ch25.html*

String features

New String function: `String.prototype.trim()`

```
// setOutput() is a custom function.
var myString = "Nice ";
setOutput(myString);
setOutput(myString.trim());
setOutput(String.trim(myString));

// Output
Nice ;Nice;Nice;
```

*(Rauschmayer 2014, *Speaking JavaScript: An In-Depth Guide for Programmers*, http://speakingjs.com/es5/index.html), "Chapter 25. New in ECMAScript"* *http://speakingjs.com/es5/ch25.html*

New String access to characters via bracket operator `[...]`.

```
var myString = "The Sun";
return myString[0]; // Returns "T".
```

*(Rauschmayer 2014, *Speaking JavaScript: An In-Depth Guide for Programmers*, http://speakingjs.com/es5/index.html), "Chapter 25. New in ECMAScript"* *http://speakingjs.com/es5/ch25.html* *C:\Users\John\Documents\Sda\Code\EcmaScript\Examples\Core\strings.xhtml*

String literals can be multilined if end of continuing lines escaped with a backslah. Spaces in the string are retained so you may need to build the lines of the string at the very start of each coding line (as desired).

```
var myString = "Lorem, ipsum \
                dolor sit \
amet";
```

```
// Output
Lorem, ipsum                 dolor sit amet
```

*(Rauschmayer 2014, *Speaking JavaScript: An In-Depth Guide for Programmers*, http://speakingjs.com/es5/index.html), "Chapter 25. New in ECMAScript"*
http://speakingjs.com/es5/ch25.html
C:\Users\John\Documents\Sda\Code\EcmaScript\Examples\Core\strings.xhtml

Strict Mode

Strict Mode. Makes the parser perform tighter checks, forbidding some features.

```
'use strict';

// or

"use strict";
```

*(Rauschmayer 2014, *Speaking JavaScript: An In-Depth Guide for Programmers*, http://speakingjs.com/es5/index.html), "Chapter 25. New in ECMAScript"*
http://speakingjs.com/es5/ch25.html
http://www.ecma-international.org/ecma-262/9.0/index.html#sec-directive-prologues-and-the-use-strict-directive
http://www.ecma-international.org/ecma-262/9.0/index.html#sec-strict-mode-code
http://www.ecma-international.org/ecma-262/9.0/index.html#sec-strict-mode-of-ecmascript

Other Syntactic Changes

When defining and creating an object without a constructor - initialise by object initialiser (See [Define and initialise without a constructor]()) - you can use the "new" and "function" reserved words as: unquoted property keys in the object initialiser; and reference the property after the dot operator.

```
> var obj = { new: 'abc' };
> obj.new
'abc'
```

*(Rauschmayer 2014, *Speaking JavaScript: An In-Depth Guide for Programmers*, http://speakingjs.com/es5/index.html), "Chapter 25. New in ECMAScript"*
http://speakingjs.com/es5/ch25.html

Trailing commas are permitted in array literals and object literals.

```
var myArray = ['c','a','a','a','b','b','a',];  //  OK.

// OK
var uranus =
  {
    name : "Uranus",
    day : "Happy",
    year : 3445,
    isCool : false,
    'HotFactor': 'boiling',
  };
```

*(Rauschmayer 2014, *Speaking JavaScript: An In-Depth Guide for Programmers*, http://speakingjs.com/es5/index.html), "Chapter 25. New in ECMAScript"*
http://speakingjs.com/es5/ch25.html
C:\Users\John\Documents\Sda\Code\EcmaScript\Libraries\StandardLibrary\Core\array.html
C:\Users\John\Documents\Sda\Code\EcmaScript\Examples\Web\arrayCreationUsingObjectInitialiser.htm

6th ES2015

Arrow function expression

Arrow function expressions are shorthands for creating functions. But they have limitations:

- Ill suited as methods;
- Cannot be used as constructors.

```
// ## Basic Syntax

(param1, param2, …, paramN) => { statements }
(param1, param2, …, paramN) => expression
// equivalent to: => { return expression; }

// Parentheses are optional when there's only one parameter name:
(singleParam) => { statements }
singleParam => { statements }

// The parameter list for a function with no parameters should be written with a
pair of parentheses.
() => { statements }


// ## Advanced Syntax

// Parenthesize the body of function to return an object literal expression:
params => ({foo: bar})

// Rest parameters and default parameters are supported
(param1, param2, ...rest) => { statements }
(param1 = defaultValue1, param2, …, paramN = defaultValueN) => {
statements }

// Destructuring within the parameter list is also supported
var f = ([a, b] = [1, 2], {x: c} = {x: a + b}) => a + b + c;
f(); // 6
```

```
var myFunction = null;

// Long hand
myFunction = function fun(myString) {
  return "cool01 " + myString;
}

//
// Basic Syntax
//
```

```
// (param1, param2, …, paramN) => { statements }
myFunction = (myString) => {return "cool02 " + myString;}

// (param1, param2, …, paramN) => expression
// equivalent to: => { return expression; }
myFunction = (myString) => "cool03 " + myString;

// When there is only one parameter, we can remove the surrounding parenthesies:
myFunction = myString => "cool04 " + myString;

addOutput(myFunction("times"));
```

(Mozilla Developer Network 2019, "JavaScript Reference", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference), "Arrow functions", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions
C:\Users\John\Documents\Sda\Code\EcmaScript\Examples\Core\whatsNew-06th-ES2015.xhtml

Modules (import/export)

Modules (containining functions, variables, objects, etc) can be exported and imported in a consuming javascript file (another module). Both export and import statements are required for this.

*(Hoban [2014] 2019, *Overview of ECMAScript 6 Features.*, https://github.com/lukehoban/es6features)*
https://github.com/lukehoban/es6features#modules
(Mozilla Developer Network 2019, "JavaScript Reference", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference)
"Export", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export
"import", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import

The importing script must include a `type="module"`.

```
// Consuming HTML doc
<script type="module">
/* <![CDATA[ */
...
```

(Mozilla Developer Network 2019, "JavaScript Reference", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference)
"Export", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export
"import", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import

Basic example.

// modules/my-maths-module.js

```
export function myGreatFunction() {
  return "myGreatFunction test string";
}

export const fooFactor = 13.4;
```

// Consuming HTML doc

```
<script type="module">
```

```
/* <![CDATA[ */
import * as MyMaths from "./modules/my-maths-module.js";

function addOutput(message) {
  var outputElement = document.getElementsByTagName("output")[0];
  outputElement.innerHTML += message + "<br />";
}
function main () {
  moduleImportDemo();
}
window.onload = main;

function moduleImportDemo() {
  addOutput(MyMaths.myGreatFunction());
  addOutput("FooFactor: " + MyMaths.fooFactor);
}
/* ]]> */
</script>
```

C:\Users\John\Documents\Sda\Code\EcmaScript\Examples\Core\whats-new-06th-es2015\import-export.xhtml

Exports are named or default.

Syntax.

(Mozilla Developer Network 2019, "JavaScript Reference", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference)
"Export", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export
"import", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import

Class declarations

Lexical block scoping

Iterators and generators

Promises for asynchronous programming

Destructuring patters

Proper tail calls

Additional data abstractions (Maps, sets, arrays of binary values)

Additional support for Unicode supplemental characters in strings and regexes.

Built-ins made extensible via subclassing.

7th ES2016

Software tools including Ecmarkup, Ecmarkdown, and Grammarkdown

Exponentiation operator

A new method to Array.prototype called **includes**.

8th ES2017

Async Functions.

Shared Memory.

Atomics.

New static methods on Object: Object.values, Object.entries, and Object.getOwnPropertyDescriptors.

9th ES2018

Asynchronous iteration support via the AsyncIterator protocol and async generators.

Four new regular expression features: the dotAll flag, named capture groups, Unicode property escapes, and look-behind assertions.

For Object properties the rest parameter and spread operator.

References, Zotero

Inline citations:
(Bentley 2019, **EcmaScript Library*, C:\Users\John\Documents\Sda\Code\EcmaScript\Libraries\StandardLibrary);

(Mozilla Developer Network 2019, "JavaScript", https://developer.mozilla.org/en-US/docs/Web/JavaScript),

(Mozilla Developer Network 2019, "JavaScript Reference", https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference),

(comp.lang.javascript n.d., "Notes on the Comp.Lang.Javascript FAQ", Accessed 2021-02-17. http://jibbering.com/faq/faq_notes/faq_notes.html#toc)

Bentley, John. 2019. **EcmaScript Library**. C:\Users\John\Documents\Sda\Code\EcmaScript\Libraries\StandardLibrary. 2019.

———. 2021. **EcmaScript Library > Core**. C:\Users\John\Documents\Sda\Code\EcmaScript\Libraries\StandardLibrary\Core. 2021.

comp.lang.javascript. n.d. "Notes on the Comp.Lang.Javascript FAQ". Accessed 2021-02-17. http://jibbering.com/faq/faq_notes/faq_notes.html#toc. n.d.

ECMA International. 2018. "ECMAScript 2018 Language Specification: ECMA-262, 9th Edition, June 2018". http://www.ecma-international.org/ecma-262/9.0/index.html. 2018-06.

Goodman, Danny, and Michael Morrison. 2004. **JavaScript Bible 5th Edition**. 5 edition. Indianapolis, Ind: Wiley, 2004-03-24.

Hoban, Luke. [2014] 2019. **Overview of ECMAScript 6 Features.**. https://github.com/lukehoban/es6features. (March 3, 2014) 2019-03-28.

Mozilla Developer Network. 2019. "JavaScript". MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/JavaScript. 2019.

———. 2019. "JavaScript Reference". MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference. 2019.

———. 2021. "JavaScript Guide". https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide. 2021-01-19.

———. 2021. "Regular Expression Syntax Cheatsheet - JavaScript | MDN". https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions/Cheatsheet. 2021-04-28.

Netscape. 2000. "Core JavaScript Reference 1.5". https://tecfa.unige.ch/guides/js/jsref15/. 2000-09-28.

Rauschmayer, Axel. 2014. **Speaking JavaScript: An In-Depth Guide for Programmers**. 1 edition. http://speakingjs.com/es5/index.html. Sebastopol, CA: O'Reilly Media, 2014-03-24.

Wikipedia. 2019. "ECMAScript". *Wikipedia*. https://en.wikipedia.org/wiki/ECMAScript. 2019-03-05. Page Version ID: 886252683.

Document Licence

Ecmascript Core Reference © 2021 by John Bentley is licensed under Attribution-NonCommercial-ShareAlike 4.0 International