## Array

*[1,'abc',new Date(),['x','y'],true]*

```
var a=new Array;  // container of numbered things
assert(a.length == 0);  // they begin with zero elements
a=new Array(8);    // unless you give them dimension
assert(a.length == 8);
assert(a[0] == null);  // indexes are from 0 to length-1
assert(a[7] == null);  // uninitialized elements are null
assert(a[20] == null); // out-of-range elements equal null
a[20]='21st element';  // but writing out of range
assert(a.length == 21);  // just makes an array bigger

a[0]='a'; a[1]='cat'; a[2]=44;  // three identical
a=new Array('a','cat',44);       // ways to fill
a=['a','cat',44];                // up an array
assert(a.length == 3);
assert(a[0] == 'a' && a[1] == 'cat' && a[2] == 44);

assert([1,2,3] != [1,2,3]);  // Array compare is tricky.
// (Tech reason: objects compare by reference, not value.)
assert([1,2,3].join() == "1,2,3");  // So we'll use join() a lot
// to work with arrays because strings compare by value.

assert(a.join() == "a,cat,44"); // join turns array into string
assert(a.join("/") == "a/cat/44"); // default comma delimited

a="a,cat,44".split();        // split parses string into array
assert(a.join() == "a,cat,44");   // (we use join() to prove it)
a="a-cat-44".split("-");            // split() also defaults
assert(a.join("+") == "a+cat+44");   // to comma delimited
a="pro@sup.net".split(/[\.\@]/); // split can also use a
assert(a.join() == "pro,sup,net");      // ❖ regular expression
      // split("") turns a string into an array of characters
assert("the end".split("").join() == "t,h,e, ,e,n,d");

a=[2,36,111]; a.sort(); // case-sensitive string sort
assert(a.join() == '111,2,36');
a.sort(function(a,b) { return a-b; });     // numeric order
assert(a.join() == '2,36,111');
// Sort function should return -,0,+ signifying <,==,>
assert(("a").localeCompare("z") < 0); // sort function

a=[1,2,3]; a.reverse();  assert(a.join() == '3,2,1');
a=[1,2,3]; assert(a.pop() == 3); assert(a.join() == '1,2');
a=[1,2,3]; a.push(4);  assert(a.join() == '1,2,3,4');
a=[1,2,3]; assert(a.shift() == 1); assert(a.join() == '2,3');
a=[1,2,3]; a.unshift(0);  assert(a.join() == '0,1,2,3');
a=[1,2,3]; // splice(iStart,nDelete,xInsert1,xInsert2,...)
a.splice(2,0,'a','b');  assert(a.join() == '1,2,a,b,3');    // insert
a.splice(1,2);          assert(a.join() == '1,b,3');         // delete
a.splice(1,2,'Z');  assert(a.join() == '1,Z'); // insert & delete
var aleft=[1,2,3], aright=[4,5,6], aboth=aleft.concat(aright);
assert(aboth.join() == "1,2,3,4,5,6");

// slice(istart,iend+1) creates a new subarray
assert([6,7,8,9].slice(0,2).join() == '6,7');  // istart,iend+1
assert([6,7,8,9].slice(1).join() == '7,8,9');  // istart
assert([6,7,8,9].slice(1,-1).join() == '7,8');  //length added
assert([6,7,8,9].slice(-3).join() == '7,8,9');  // to - values
```

## Function

*function zed() { return 0; }*

```
function sum(x,y) {                         // definition
    return x+y;                             // return value
}
var n=sum(5,5);  assert(n == 10);           // call

function sum1(x,y) { return x+y };             // 3 ways to
var sum2=function(x,y) { return x+y; }          // define a
var sum3=new Function("x","y","return x+y;");  //function
assert(sum1.toString() ==   // reveals definition code, but
   "function sum1(x,y) { return x+y; }"); // format varies

function sumx() {                       // Dynamic arguments
    var retval=0;
    for (var i=0 ; i < arguments.length ; i++) {
        retval+=arguments[i];
    }
    return retval;
}
assert(sumx(1,2) == 3);
assert(sumx(1,2,3,4,5) == 15);
```

## Date

*Date()  new Date(1999,12-1,31,23,59)*

```
var dNow=new Date();  // seize the present moment
var dPast=new Date(2002,5-1,20,23,59,59,999);
// (year,month-1,day,hours,minutes,seconds,milliseconds)

assert(dNow.getTime() > dPast.getTime());
// Compare dates only by their getTime() or valueOf()
assert(dPast.getTime() == 1021953599999);
assert(dPast.getTime() == dPast.valueOf());
// Compute elapsed milliseconds by subtracting getTime()'s
var  nHours=(dNow.getTime()-dPast.getTime())/(3600000);

// Example date and time formats: (! all vary widely)
assert(dPast.toString() == 'Mon May 20 23:59:59 EDT 2002');
assert(dPast.toGMTString() ==
                          'Tue, 21 May 2002 03:59:59 UTC');
assert(dPast.toUTCString() ==
                          'Tue, 21 May 2002 03:59:59 UTC');
assert(dPast.toDateString() ==          'Mon May 20 2002');
assert(dPast.toTimeString() ==            '23:59:59 EDT');
assert(dPast.toLocaleDateString() ==
                                'Monday, 20 May, 2002');
assert(dPast.toLocaleTimeString() ==     '23:59:59 PM');
assert(dPast.toLocaleString() ==
                       'Monday, 20 May, 2002 23:59:59 PM');

var d=new Date(0);        // Dates count milliseconds
assert(d.getTime() == 0);  // after midnight 1/1/1970 UTC
assert(d.toUTCString() == 'Thu, 1 Jan 1970 00:00:00 UTC');
assert(d.getTimezoneOffset() == 5*60); // minutes west

// ⌘ terminology: getTime() is millisec after 1/1/1970
// getDate() is day of month, getDay() is day of week
// Same for setTime() and setDate(). There is no setDay().

d.setFullYear(2002);   assert(d.getFullYear() == 2002);
d.setMonth(5-1);       assert(d.getMonth() == 5-1);
d.setDate(31);         assert(d.getDate() == 31);
d.setHours(23);        assert(d.getHours() == 23);
d.setMinutes(59);      assert(d.getMinutes() == 59);
d.setSeconds(59);      assert(d.getSeconds() == 59);
d.setMilliseconds(999);
                       assert(d.getMilliseconds() == 999);
assert(d.getDay() == 5);  // 0=Sunday, 6=Saturday
d.setYear(99);   assert(d.getYear() == 99); // Y2K bugs
d.setYear(2001); assert(d.getYear() == 2001);

d.setUTCFullYear(2002);
                       assert(d.getUTCFullYear() == 2002);
d.setUTCMonth(5-1); assert(d.getUTCMonth() == 5-1);
d.setUTCDate(31);      assert(d.getUTCDate() == 31);
d.setUTCHours(23);     assert(d.getUTCHours() == 23);
d.setUTCMinutes(59); assert(d.getUTCMinutes() == 59);
d.setUTCSeconds(59); assert(d.getUTCSeconds() == 59);
d.setUTCMilliseconds(999);
                    assert(d.getUTCMilliseconds() == 999);
assert(d.getUTCDay() == 5);  // 0=Sunday, 6=Saturday

// Most set-functions can take multiple parameters:
d.setFullYear(2002,5-1,31);  d.setUTCFullYear(2002,5-1,31);
d.setMonth(5-1,31);          d.setUTCMonth(5-1,31);
d.setHours(23,59,59,999);    d.setUTCHours(23,59,59,999);
d.setMinutes(59,59,999);     d.setUTCMinutes(59,59,999);
d.setSeconds(59,999);        d.setUTCSeconds(59,999);

// If you must call more than one set function, it's
// probably better to call the longer-period function first.

d.setMilliseconds(0); // (following point too coarse for msec)
// Date.parse() works on the output of either toString()
var msec=Date.parse(d.toString()); // or toUTCString().
assert(msec == d.getTime());          // The formats of
msec=Date.parse(d.toUTCString());  // those strings vary
assert(msec == d.getTime());   // one computer to another.
```

### Client-side JavaScript can be many places

1. *Header:* `<head> <script> ▪▪▪ </script> </head>`
   *Runs first before body is loaded.*
2. *Include:* `<script src="http://url/filename.js"></script>`
   *Text file with JavaScript code in it. (Better for XHTML.)*
3. *Body:* `<body> <script> ▪▪▪ </script> </body>`
   *Generate HTML with document.writeln(raw_html_string);*
4. *Event:* `<element onevent=" ▪▪▪ ">`
   *All HTML attributes that begin with "on" take event code.*
5. *URL:* `<a href="javascript: ▪▪▪ ; void 0;">`
   *Executes on click. All one line. (void 0 avoids page fetch.)*
6. *Bookmarklet aka Favelet:* `javascript: ▪▪▪ ; void 0;`
   *Savable browser utility, more at www.bookmarklets.com*
7. *String:* `eval(" ▪▪▪ ");`
   *Executes expression or code, returns the final result*

### Type Symbols

| | |
|---|---|
| a | array |
| b | boolean |
| f | function |
| i | *integer* number |
| n | number |
| o | object |
| p | *property* |
| r | regular expression |
| s | string |
| T | *Type (constructor)* |
| x | *variable, any type* |
| ... | *parameters* |

### JavaScript Literals

```
n=2;
n=2.1;
n=2.1e3;
n=0xFF;
n=010;
s="string";
s='string';
b=false;
b=true;
a=[x, x, x];
f=function(p) { statements; };
o={p:x, p:x, p:x};
r=/regular expression/;
```

tighter — operator binding strength — looser

```
x=o.p;
o=a[i];
x=o["p"];
x=f(...);
o=new T(...);
-----------------
n=n++;
n=n--;
n=+n;
n=-n;
n=~n;
b=!b;
delete o.p;
s=typeof o;
x=void x;
-----------------
n=n*n;
n=n/n;
n=n%n;
-----------------
n=n+n;
n=n-n;
s=s+s;
s=s+n;
-----------------
n=n<<i;
n=n>>i;
n=n>>>i;
-----------------
b=n < n;
b=n <= n;
b=n >= n;
b=n > n;
-----------------
b=s < s;
b=s <= s;
b=s >= s;
b=s > s;
b=o instanceof T;
b=s in o;
-----------------
b=n == n;
b=n != n;
b=n === n;
b=n !== n;
b=s == s;
b=s != s;
b=s === s;
b=s !== s;
b=x === x;
b=x !== x;
-----------------
n=(n&n);
-----------------
n=(n^n);
-----------------
n=(n|n);
-----------------
b=b&&b;
-----------------
b=b||b;
-----------------
x=(b ? x : x);
x=x;       n<<=n;
n*=n;      n>>=n;
n/=n;      n>>>=n;
n%=n;      n&=n;
n+=n;      n^=n;
n-=n;      n|=n;
s+=s;
-----------------
x=x,x;
```

## assertiveness

```
// Unit testing is the best advance in programming since
// subroutines were invented.  See www.xprogramming.com.
// At the core is the assert() function (also called check()).
// The goals are faster fixes and focused, fearless progress.

// Here assert()'s do more: describe JavaScript in JavaScript.
// Why would you learn and lookup JavaScript using a book
// that's 95% English? As Berlitz® knew, to live is to immerse.

// All the code in this reference not only runs but tests itself.
// Try it:  www.visibone.com/javascript/unittest.html
function assert(fact) {           // assert() can be very simple
   if (!fact) alert("Assert failure!");
}
function assert(fact,details) { // this helps to tell them apart
   if (!fact) alert("Assert failure!  "+details);
}
function assert(fact,details) {  // this assert shows the name
   if (!fact) {                  // of the function it's called from
      var msg="Assert failure!  "+details;
      if (arguments.callee.caller != null) { // but not in Opera
         msg=msg+" in function "+
            arguments.callee.caller.toString().match(
               /function\s+(\w+)/)[1];
      }
      alert(msg);
   }
}
// JsUnit is an open source JavaScript unit test framework.
// It has very elaborate and useful assert()'s.  www.jsunit.net
```

## Number *2 1.5 2.5e3 0xFF 010*

```
assert(2+2==4);        // numbers are 64-bit floating point
assert(1.5==3/2);      // (no separate integer type ⓃⒺ)
assert(2.5e3==2500);   // 2.5 x 10з exponential notation
assert(0xFF==255);     // hexadecimal
assert(010==8);        // octal

assert(   2 + 2 == 4);    // addition          simple math
assert(  10 - 3 == 7);    // subtraction
assert(   3 * 8 == 24);   // multiplication
assert( 123 / 10 == 12.3);// real (not integer) division ⓃⒺ
assert(1234 % 100 == 34); // modulo (remainder)

var n=3;  n += 30; assert(n==33); // compute & store
var n=33; n -= 30; assert(n==3);  // x*=y is the same
var n=3;  n *= 20; assert(n==60); // as x=x*y
var n=38; n /= 10; assert(n==3.8);
var n=38; n %= 10; assert(n==8);

assert(-3+3==0); // negative number (unary minus)
var n=3; n++; assert(n==4); // increment
var n=3; n--; assert(n==2); // decrement

assert( 99 <  100);   // less than           comparisons
assert( 99 <= 100);   // less than or equal
assert(100 >   99);   // greater than
assert(100 >=  99);   // greater than or equal
assert(100 == 100);   // equal
assert( 99 != 100);   // not equal

assert(1000 << 3 == 8000); // shift left       32-bit math
assert(1000 >> 3 == 125);  // shift right, signed  ⧗=31-bit
assert(0xFFFF0000 >>> 8 == 0x00FFFF00); // unsigned
// 💣 Always use parentheses around terms with: & | ^
assert((0x55555555 & 0xFF00FFFF) == 0x55005555);// and
assert((0x55555555 | 0x00FF0000) == 0x55FF5555);// or
assert((0x55555555 ^ 0x00FF0000) == 0x55AA5555);// xor
// >>>0 converts to unsigned, avoiding 💣 sign extension
assert((~0x55555555) >>> 0 == 0xAAAAAAAA); // 1's compl.
assert((~0x55555555)       != 0xAAAAAAAA); // is signed!
var n=0x555;  n &= 0xF0F;  assert(n == 0x505);
var n=0x555;  n |= 0x0F0;  assert(n == 0x5F5);
var n=0x555;  n ^= 0x0F0;  assert(n == 0x5A5);
var n=-10;   n<<=1;  assert(n == -20); // shift left
var n=-10;   n>>=1;  assert(n == -5);  // signed right
var n=0x8;  n>>>=1;  assert(n == 0x4); // unsigned Ⓔ

assert(Number.MIN_VALUE < 1e-307); ✿ // special
assert(Number.MAX_VALUE > 1e308);    // numbers
assert(Number.NEGATIVE_INFINITY==1/0);       ⓃⒺ
assert(Number.POSITIVE_INFINITY==-1/0);
assert(isNaN(0/0)); // NaN stands for Not a Number
assert(0/0 != 0/0); // 💣 NaN is not equal to itself! ✿
assert(!isFinite(1/0));  assert(isFinite(1));
```

## Math *Math.PI Math.max() Math.round()*

```
assert(Math.abs(-3.2) == 3.2);
assert(Math.max(1,2) == 2 && Math.max(1,2,3,4) == 4);
assert(Math.min(1,2) == 1 && Math.min(1,2,3,0) == 0);
assert(0 <= Math.random() && Math.random() < 1);

assert(Math.ceil(1.5) == 2);   // round up, to the nearest
assert(Math.ceil(-1.5) == -1); // integer higher or equal
assert(Math.round(1.7) == 2);  // round to the nearest
assert(Math.round(1.2) == 1);     // integer, up or down
assert(Math.floor(1.5) == 1);// round down to the nearest
assert(Math.floor(-1.5) == -2); // integer lower or equal

var n;
n=Math.E;        assertApprox(Math.log(n),1);
n=Math.LN10;     assertApprox(Math.pow(Math.E,n),10);
n=Math.LN2;      assertApprox(Math.pow(Math.E,n),2);
n=Math.LOG10E;   assertApprox(Math.pow(10,n),Math.E);
n=Math.LOG2E;    assertApprox(Math.pow(2,n),Math.E);
n=Math.PI;       assertApprox(Math.sin(n/2),1);
n=Math.SQRT1_2;  assertApprox(n*n,0.5);
n=Math.SQRT2;    assertApprox(n*n,2);

assertApprox(Math.acos(1/2),Math.PI/3); // trig functions
assertApprox(Math.asin(1/2),Math.PI/6); // are in radians
assertApprox(Math.atan(1),Math.PI/4);
assertApprox(Math.atan2(1,1),Math.PI/4);
assertApprox(Math.cos(Math.PI/3),1/2);
assertApprox(Math.exp(1),Math.E);
assertApprox(Math.log(Math.E),1);  // (base e, not 10)
assertApprox(Math.pow(10,3),1000);
assertApprox(Math.sin(Math.PI/6),1/2);
assertApprox(Math.sqrt(25),5);
assertApprox(Math.tan(Math.PI/4),1);
```

```
// Math functions are accurate to 15 digits:
function assertApprox(a,b) {
    assert((b*0.999999999999999 < a) &&
        (a < b*1.00000000000001));
}
```

## Number ⇔ String *conversions*

```
// First, a subtle distinction in JavaScript comparisons:
assert(3  == "3");  // == Equals    flexible about type
assert(3  != "4");
assert(3 === 3);    // === Identical   must be the
assert(3 !== "3");        ⓃⒺ          same type

assert(256 == "256");     // Strings in a numeric context are
assert(256.0 == "256");   // converted to a number.  This is
assert(256 == "256.0");   // usually reasonable and useful.
assert("256" != "256.0"); // (String contexts, no convert! 💣)
assert(256 == "0x100");   // Hexadecimal 0x prefix works,
assert(256 == "0256");    // but no octal 0 prefix this way.
assert(256 != "256 xyz"); // No extraneous characters.

// Number ⇦ String
assert(256 === "256" - 0); //- converts string to number
assert("2560" === "256" + 0); //+ concatenates strings 💣
assert(256 === parseInt("256"));
assert(256 === parseInt("256 xyz"));   // extras forgiven
assert(256 === parseInt("0x100"));     // hexadecimal
assert(256 === parseInt("0400"));      // 0 for octal 💣
assert(256 === parseInt("0256",10));   // certain decimal
assert(256 === parseInt("100",16));    // hexadecimal
assert(256 === parseInt("400",8));     // octal
assert(25.6 === parseFloat("2.56e1"));
assert("256" === "256".valueOf()); // (no conversion help)
assert(isNaN(parseInt("xyz")));    // gibberish handling
assert(isNaN(parseFloat("xyz")));

// Number ⇨ String, explicit conversions
assert(256 + ""                  === "256");
assert((256).toString()          === "256");
assert((2.56).toString()         === "2.56");
assert((256).toString(16)        === "100");   🐞0 ✿real
assert((2.56).toFixed()          === "3");
assert((2.56).toFixed(3)         === "2.560");
assert((2.56).toPrecision(2)     === "2.6");
assert((256).toExponential(4)    === "2.5600e+2");
assert((1024).toLocaleString() === "1,024.00"); // ✿!
// Exotic numbers convert to strings in precise ways:
assert((-1/0).toString()         === "-Infinity");
assert((0/0).toString()          === "NaN");
assert((1/0).toString()          === "Infinity");
```

## Boolean *true false*

```
var t=true;   assert(t);
var f=false;  assert(!f);            // ! is boolean not
assert((true && false) == false);    // && is boolean and
assert((true || false) == true);     // || is boolean or
assert((true ? 'a' : 'b') == 'a');  // miniature if-else chooser
assert((false ? 'a' : 'b') == 'b'); // (💣 parentheses outside)
```

## String *'abc' "abc" "line\u000D\u000A"*

```
var s="string"; // double or single quotes, your choice ⓃⒺ
var s='string'; // but 💣 'can't always use apostrophe's'
assert("str" + "ing" == "string"); //+ concatenates Ⓔ
assert(s.length == 6); // all strings have a length property
assert(s.charAt(0) == "s"); // and are indexed from zero
assert(s.charAt(5) == "g"); // no character type ⓃⒺ
assert(s[5] == "g");   // only Netscape indexes strings
assert(s.charCodeAt(5) == 0x67); // ASCII character value
assert(String.fromCharCode(65,66,67) == "ABC");

assert(s.substring(2) == "ring");       // istart
assert(s.substring(2,4) == "ri");       // istart, iend+1 } same
assert(s.substring(4,2) == "ri");       // iend+1, istart }
assert(s.substring(-2) == "string");    // (negative values
assert(s.substring(2,-2) == "st");      // are just like zero)
assert(s.slice(2) == "ring");        // istart
assert(s.slice(2,4) == "ri");        // istart, iend+1
assert(s.slice(-2) == "ng");  // ✿ - same as 0 before IE5.5
assert(s.slice(1,-1) == "trin");
assert(s.substr(2) == "ring");   // istart
assert(s.substr(2,2) == "ri");   // istart, inum
assert(s.substr(-2,2) == "ng");   // ✿ - same as 0 in IE

assert('abc'.toUpperCase() == 'ABC');
assert('ABC'.toLowerCase() == 'abc');
assert('abc'.toLocaleUpperCase() == 'ABC');
assert('ABC'.toLocaleLowerCase() == 'abc');

assert('str'.concat('ing') == 'str'+'ing'); // two kindsa glue
assert(s.indexOf('ing') == 3); // find substring, -1=can't
assert('strings'.lastIndexOf('s') == 6); // find rightmost

// These involve ❖Regular Expressions and/or ❖Arrays
asssert(/ing/.test(s));
assert(s.search(/ing/) == 3);
assert('nature'.replace(/a/,'ur') == 'nurture');
assert('a:b:c'.split(':').join('..') == 'a..b..c');
assert('1-37/54'.match(/\d+/g).join() == '1,37,54');
RegExp.lastIndex=0;
assert(/o(.)r/.exec('courage').join() == 'our,u');

//💣 search expects a regular expression (where dot=any):
assert('imdb.com'.search(".") == 0);        // so you must
assert('imdb.com'.search(/./) == 0);       // not forget to
assert('imdb.com'.search(/\./) == 4);      // double-escape
assert('imdb.com'.search("\\.") == 4);  // your punctuation

s="\uFFFF"; // Ⓔ 16-bit hex Unicode              Slash
s="\xFF";   // hexadecimal ASCII            Characters
s="\377"; s="\77"; s="\7"; // 8-bit octal ✿16-bit
assert('\0' == '\u0000'); // NUL
assert('\b' == '\u0008'); // backspace (BS)
assert('\t' == '\u0009'); // tab (TAB)
assert('\f' == '\u000C'); // formfeed (FF)
assert('\r' == '\u000D'); // return (CR)
assert('\n' == '\u000A'); // newline (LF)
assert('\v' == '\u000B'); // vertical tab (VT)
assert("\"" == '"');       // double-quote
assert('\'' == "'");       // single-quote
assert("\\" == '\u005C'); // a single backslash

// Multi-line strings (backslash works, plus is better)
s="this is a \
test"; // Ⓝ (comments not allowed on the line above)
assert(s == "this is a test"); // ✿N4 inserts LF, ✿Opera CR
s="this is a " + // Ⓔ concatenate (it's a plus to have plus)
"better test";    // comments allowed on both of these lines
assert(s == "this is a better test");

// NUL isn't special, it's a character like any other Ⓔ
assert('abc\0def'.length == 7); // ✿ Opera ignores \0, try
assert('abc\0def' != 'abc\0xyz'); // String.fromCharCode(0)

// User-entered cookies or URLs must encode punctuation
assert(escape("that's all.") == "that%27s%20all.");  ✿@
assert(unescape("that%27s%20all.") == "that's all.");
// These are escaped: "%<>[\]^`{|}#$&,:;=?!'()~
// plus space.  Alphanumerics and these are not: *-._+/@
// encodeURI() translates   %<>[\]^`{|}
// encodeURIComponent() %<>[\]^`{|}#$&+,/:;=?
// decodeURI() and decodeURIComponent() the inverse
```

*Duplicate definitions are harmless, the latter prevails.* ⓃⒺ

```
var a=1  // Lines don't have to end with ; semicolons, ⓃⒺ
var b=2; // but using them consistently shows character.
// Multiple statements on the same line require semicolons
var c=3;  c+=a;  c+=b;  assert(c == 6);  // between them.
```