Javascript

October 15, 2017

JavaScript ("JS" for short) is a full-fledged dynamic programming language that, when applied to an HTML document, can provide dynamic interactivity on websites. - Invented by Brendan Eich, co-founder of Mozilla project. - JavaScript is case-sensitive and uses the Unicode character set. - JavaScript borrows most of its syntax from Java, but is also influenced by Awk, Perl and Python. - ECMAScript 2015 or ES6 is the latest code standard

0.1 Basics

0.1.1 Keywords

Reserved keywords as of ECMAScript 2015: **33**

```
break    case     catch    class    const     continue    debugger    default    delete
export   extends  finally  for      function  if          import      in         instanceo
super    switch   this     throw    try       typeof      var         void       while
```

Future reserved Keywords - Always reserved : `enum` - reserved ONLY when they are found in strict mode code

```
implements     interface    let    package    private    protected    public    static
```

- reserved ONLY when found in module code: `await`

Future reserved keywords in older standards (ECMAScript 1 till 3):

```
abstract     boolean        byte    char        double    final          float
goto         int            long    native      short     synchronized   throws
transient    volatile
```

0.1.2 Comments

```
In [1]: /*
        Multi-line
        comments
        */
```

```
Out[1]: undefined

In [2]: // single-line comment

Out[2]: undefined
```

0.1.3 Variables

declaring the variable

```
In [3]: var myVariable;

Out[3]: undefined

In [4]: typeof myVariable

Out[4]: 'undefined'
```

assigning to variable

```
In [5]: myVariable = 'apple';

Out[5]: 'apple'

In [7]: typeof myVariable   // data type of the object

Out[7]: 'string'
```

retrieving the value by cxalling it

```
In [8]: myVariable;

Out[8]: 'apple'
```

changing the datatype of created variable

```
In [9]: myVariable = 123;

Out[9]: 123

In [10]: typeof myVariable

Out[10]: 'number'
```

creating and assigning a variable

```
In [11]: var myVal = 12343;

Out[11]: undefined

In [12]: myVal;
```

```
Out[12]: 12343
```

variable hoisting It doesn't work in strict mode

```
In [101]: bla = 2;
          var bla;
          // ...

          // is implicitly understood as:

          var bla;
          bla = 2;
```

```
Out[101]: 2
```

Declaring and initializing more than one variable

```
In [102]: var a = 90, b = 80, c = 70, d = 60;
```

```
Out[102]: undefined
```

```
In [103]: d;
```

```
Out[103]: 60
```

Assigning two variables with single string value

```
In [104]: var a = 'A';
          var b = a;

          // Equivalent to:

          var a, b = a = 'A';
```

```
Out[104]: undefined
```

```
In [105]: var x = y, y = 'A';       // x === undefined && y === 'A'
          console.log(x + y);       // undefinedA
```

```
undefinedA
```

```
Out[105]: undefined
```

```
In [107]: var q = 'A', p = q ;       // p === 'A' && p === q
          console.log(p + q);        // AA
```

```
AA
```

```
Out[107]: undefined
```

```
In [108]: var m = n = o = 999;
          m;
```

```
Out[108]: 999
```

0.1.4 Constants

```
In [109]: const PI = 3.14;

Out[109]: undefined

In [110]: PI;

Out[110]: 3.14

In [111]: PI = 7898;  // constants can't be changed


        evalmachine.<anonymous>:1

        PI = 7898;

           ^



        TypeError: Assignment to constant variable.

            at evalmachine.<anonymous>:1:4

            at ContextifyScript.Script.runInThisContext (vm.js:25:33)

            at Object.runInThisContext (vm.js:97:38)

            at run ([eval]:617:19)

            at onRunRequest ([eval]:388:22)

            at onMessage ([eval]:356:17)

            at emitTwo (events.js:106:13)

            at process.emit (events.js:191:7)

            at process.nextTick (internal/child_process.js:758:12)

            at _combinedTickCallback (internal/process/next_tick.js:73:7)
```

0.1.5 Data types in javascript

The latest ECMAScript standard defines SEVEN data types: - six premitive data types 1. Boolean : *true* and *false* 2. null : null is not same as Null, NULL, ... (case-sensitive) 3. Undefined: value is undefined 4. Number : 12, -0.5 5. string : 'apple', "mango" 6. Symbool(new in ECMAScript 2015) : - Object

String: A sequence of text known as a string. To signify that the variable is a string, you should enclose it in quote marks.

```
In [13]: var myString = 'apple';
         myString;

Out[13]: 'apple'

In [14]: myString = 'bat'
         myString;

Out[14]: 'bat'

In [15]: var myString = "someThing";   // single and double quotes are treated the same
         myString;

Out[15]: 'someThing'

In [165]: myString.length

Out[165]: 9

In [166]: `In JavaScript '\n' is a line-feed.`; // Basic literal string

Out[166]: 'In JavaScript \'\n\' is a line-feed.'

In [167]: // Multiline strings
          `In JavaScript template strings can run
           over multiple lines, but double and single
           quoted strings cannot.`

Out[167]: 'In JavaScript template strings can run\n over multiple lines, but double and single'

In [168]: console.log(`In JavaScript template strings can run
           over multiple lines, but double and single
           quoted strings cannot.`)

In JavaScript template strings can run
 over multiple lines, but double and single
 quoted strings cannot.


Out[168]: undefined

In [169]: // String interpolation
          var name = 'Bob', time = 'today';
          `Hello ${name}, how are you ${time}?`

Out[169]: 'Hello Bob, how are you today?'

In [170]: "Hello ${name}, how are you ${time}?" \\ observe quotes
```

```
Out[170]: 'Hello ${name}, how are you ${time}?'
```

Using special characters in strings

```
In [172]: console.log('one line \n another line')

one line
 another line


Out[172]: undefined

In [174]: console.log('one line \t another line')

one line          another line


Out[174]: undefined

JavaScript special characters
------------------------------------------------------
Character   Meaning
------------------------------------------------------
\0          Null Byte
\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 XX
                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 X
                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. See Unico
\u{XXXXX}     Unicode code point escapes.
                For example, \u{2F804} is the same as the simple Unicode escapes \uD87E\uDC04.

In [175]: var quote = "He read \"The Cremation of Sam McGee\" by R.W. Service.";
          console.log(quote);

He read "The Cremation of Sam McGee" by R.W. Service.


Out[175]: undefined
```

```
In [176]: var home = 'c:\\temp';
          home;

Out[176]: 'c:\\temp'

In [177]: console.log(home);

c:\temp


Out[177]: undefined
```

can also be used as line-continution operator

```
In [178]: var str = 'this string \
          is broken \
          across multiple \
          lines.'
          console.log(str);   // this string is broken across multiplelines.

this string is broken across multiple lines.


Out[178]: undefined

In [179]: var poem =
          'Roses are red,\n\
          Violets are blue.\n\
          Sugar is sweet,\n\
          and so is foo.'

          console.log(poem);

Roses are red,
Violets are blue.
Sugar is sweet,
and so is foo.


Out[179]: undefined

In [180]: poem;

Out[180]: 'Roses are red,\nViolets are blue.\nSugar is sweet,\nand so is foo.'
```

ECMAScript 2015 introduces a new type of literal, namely template literals

```
In [181]: var poem =
          `Roses are red,
          Violets are blue.
          Sugar is sweet,
          and so is foo.`

          console.log(poem);
```

```
Roses are red,
Violets are blue.
Sugar is sweet,
and so is foo.


Out[181]: undefined

In [182]: poem;

Out[182]: 'Roses are red, \nViolets are blue. \nSugar is sweet, \nand so is foo.'
```

Number: A number. Numbers don't have quotes around them.

Numbers can be expressed in - decimal (base 10) - sequence of digits without leading 0(zero) - hexadecimal (base 16) - with leading 0x or 0X; 0-9; a-f and A-F - octal (base 8) - with leading 0o or 0O; digits 0-7 - binary (base 2) - with leading 0b or 0B; digits 0 and 1 only

```
0, 117 and -345               (decimal, base 10)
015, 0001 and -0o77           (octal, base 8)
0x1123, 0x00111 and -0xF1A7   (hexadecimal, "hex" or base 16)
0b11, 0b0011 and -0b11        (binary, base 2)

In [16]: var myVal = 10;    // int
         myVal;

Out[16]: 10

In [17]: typeof myVal

Out[17]: 'number'

In [18]: myVal = -1.5;     // float
         myVal;

Out[18]: -1.5

In [19]: typeof myVal

Out[19]: 'number'

In [136]: var myVal = -0o77; // octal
          myVal;

Out[136]: -63

In [137]: typeof myVal;

Out[137]: 'number'

In [138]: var myVal = -0xF1A7; // hexadecimal
          myVal;
```

```
Out[138]: -61863

In [139]: typeof myVal;

Out[139]: 'number'

In [140]: var myVal = -0b11; // binary
          myVal;

Out[140]: -3

In [141]: typeof myVal;

Out[141]: 'number'
```

floating point value - syntax: `[(+|-)][digits][.digits][(E|e)[(+|-)]digits]`

```
In [142]: var myVal = -3.1E+12; // floating - point  // small 'e' also works same
          myVal;

Out[142]: -3100000000000

In [143]: typeof myVal;

Out[143]: 'number'
```

Boolean: A **true**/**false** value. The words true and false are special keywords in JS, and don't need quotes.

```
In [20]: myBool = true;     // It was created even without using 'var' keyword; but recommended
         myBool;

Out[20]: true

In [21]: typeof myBool

Out[21]: 'boolean'

In [22]: var myBool = true;
         myBool;

Out[22]: true

In [23]: myBool = false; myBool; // multiple statements in same line

Out[23]: false

In [24]: myBool = 'TRUE'; // changing the datatype; dynamic typed language
         myBool;

Out[24]: 'TRUE'
```

```
In [25]: typeof myBool

Out[25]: 'string'
```

Array: A structure that allows you to store multiple values in one single reference.

```
In [26]: var myArray = [1, 12.9, 'apple', "banana", true]

Out[26]: undefined

In [27]: typeof myArray  // Object type

Out[27]: 'object'

In [28]: myArray;

Out[28]: [ 1, 12.9, 'apple', 'banana', true ]

In [29]: myArray[0]    // Indexing in javascript starts with 0

Out[29]: 1

In [30]: myArray[3]

Out[30]: 'banana'

In [31]: myArray[-2]    // JS doesn't support negative indices

Out[31]: undefined

In [32]: myArray[:]                          // indices based slicing is not supported in Javas

        evalmachine.<anonymous>:1

        myArray[:]                          // indices based slicing is not supported in Javasc
                ^


        SyntaxError: Unexpected token :

            at createScript (vm.js:56:10)

            at Object.runInThisContext (vm.js:97:10)

            at run ([eval]:617:19)

            at onRunRequest ([eval]:388:22)
```

```
    at onMessage ([eval]:356:17)

    at emitTwo (events.js:106:13)

    at process.emit (events.js:191:7)

    at process.nextTick (internal/child_process.js:758:12)

    at _combinedTickCallback (internal/process/next_tick.js:73:7)

    at process._tickCallback (internal/process/next_tick.js:104:9)
```

```
In [128]: var fish = ['Lion', , 'Angel'];
          fish;

Out[128]: [ 'Lion', , 'Angel' ]

In [129]: fish[1];                // when no value is provided, it will be 'undefined'

Out[129]: undefined

In [130]: var myList = ['home', , 'school', ];
          myList[1];

Out[130]: undefined

In [131]: myList[3];

Out[131]: undefined

In [132]: myList[4];

Out[132]: undefined

In [133]: myList[999]; // it is not throwing exception

Out[133]: undefined

In [134]: var myList = ['home', , 'school' ];
          myList[999];

Out[134]: undefined

In [135]: var myList = ['home', , 'school', , ];  // only last comma is ignored
          myList;

Out[135]: [ 'home', , 'school',  ]
```

NOTE: Trailing commas can create errors in older browser versions and it is a best practice to remove them.

Object: Basically, anything. Everything in JavaScript is an object, and can be stored in a variable.

An object literal is a list of zero or more pairs of property names and associated values of an object, enclosed in curly braces ({}).

```
In [144]: var sales = 'Toyota';

          function carTypes(name) {
            if (name === 'Honda') {
              return name;
            } else {
              return "Sorry, we don't sell " + name + ".";
            }
          }

          // decalaring and assigning to  'car' object
          var car = { myCar: 'Saturn', getCar: carTypes('Honda'), special: sales };

          console.log(car.myCar);   // Saturn
          console.log(car.getCar);  // Honda
          console.log(car.special); // Toyota
```

```
Saturn
Honda
Toyota
```

```
Out[144]: undefined
```

```
In [145]: var car = { manyCars: {a: 'Saab', 'b': 'Jeep'}, 7: 'Mazda' };

          console.log(car.manyCars.b); // Jeep
          console.log(car[7]); // Mazda
```

```
Jeep
Mazda
```

```
Out[145]: undefined
```

```
In [147]: var unusualPropertyNames = {
            '': 'An empty string',
            '!': 'Bang!'
          }
```

```
Out[147]: undefined
```

```
In [148]: console.log(unusualPropertyNames.'');   // SyntaxError: Unexpected string
```

```
        evalmachine.<anonymous>:1
        console.log(unusualPropertyNames.'');   // SyntaxError: Unexpected string
                                            ^^


        SyntaxError: Unexpected string
            at createScript (vm.js:56:10)
            at Object.runInThisContext (vm.js:97:10)
            at run ([eval]:617:19)
            at onRunRequest ([eval]:388:22)
            at onMessage ([eval]:356:17)
            at emitTwo (events.js:106:13)
            at process.emit (events.js:191:7)
            at process.nextTick (internal/child_process.js:758:12)
            at _combinedTickCallback (internal/process/next_tick.js:73:7)
            at process._tickCallback (internal/process/next_tick.js:104:9)

In [149]: console.log(unusualPropertyNames['']);  // An empty string
An empty string

Out[149]: undefined
In [150]: console.log(unusualPropertyNames.!);    // SyntaxError: Unexpected token !

        evalmachine.<anonymous>:1
        console.log(unusualPropertyNames.!);    // SyntaxError: Unexpected token !
                                            ^
```

```
SyntaxError: Unexpected token !

    at createScript (vm.js:56:10)

    at Object.runInThisContext (vm.js:97:10)

    at run ([eval]:617:19)

    at onRunRequest ([eval]:388:22)

    at onMessage ([eval]:356:17)

    at emitTwo (events.js:106:13)

    at process.emit (events.js:191:7)

    at process.nextTick (internal/child_process.js:758:12)

    at _combinedTickCallback (internal/process/next_tick.js:73:7)

    at process._tickCallback (internal/process/next_tick.js:104:9)
```

```
In [151]: console.log(unusualPropertyNames['!']); // Bang!
```

```
Bang!
```

```
Out[151]: undefined
```

0.1.6 Enhanced Object literals

In ES2015, object literals are extended to support setting the prototype at construction, shorthand for foo: foo assignments, defining methods, making super calls, and computing property names with expressions. Together, these also bring object literals and class declarations closer together, and let object-based design benefit from some of the same conveniences.

```
In [152]: var foo = {a: 'alpha', 2: 'two'};
          console.log(foo.a);    // alpha
          console.log(foo[2]);   // two
          //console.log(foo.2);  // Error: missing ) after argument list
          //console.log(foo[a]); // Error: a is not defined
          console.log(foo['a']); // alpha
          console.log(foo['2']); // two
```

```
alpha
two
alpha
```

```
two
```

```
Out[152]: undefined
```

```
In [155]: var obj = {
              // __proto__
              __proto__: 'theProtoObj',
              // Shorthand for handler: handler
              //handler,
              // Methods
              toString() {
               // Super calls
               return 'd ' + super.toString();
              },
              // Computed (dynamic) property names
              [ 'prop_' + (() => 42)() ]: 42
          };
```

```
Out[155]: undefined
```

```
In [158]: typeof obj
```

```
Out[158]: 'object'
```

```
In [157]: obj.__proto__;
```

```
Out[157]: {}
```

```
In [160]: Object.keys(obj)  // 'Object' is built-in object
```

```
Out[160]: [ 'toString', 'prop_42' ]
```

```
In [161]: obj.toString
```

```
Out[161]: [Function: toString]
```

```
In [162]: obj.toString()
```

```
Out[162]: 'd [object Object]'
```

```
In [163]: obj.prop_42
```

```
Out[163]: 42
```

0.1.7 RegExp literals

- A regex literal is a pattern enclosed between slashes.

```
In [164]: var re = /ab+c/;
          typeof re;
```

```
Out[164]: 'object'
```

0.1.8 Data type Conversions

```
In [112]: x = 'The answer is ' + 42;
          x;

Out[112]: 'The answer is 42'

In [113]: typeof x

Out[113]: 'string'

In [114]: y = 42 + ' is the answer';
          typeof y;

Out[114]: 'string'

In [115]: y;

Out[115]: '42 is the answer'

In [116]: '37' - 7 // 30

Out[116]: 30

In [117]: '37' + 7 // "377"

Out[117]: '377'
```

0.1.9 Converting strings to numbers

If a number is stored as string, then **parseInt()** and **parseFloat()** can be used to convert back to number.

```
In [118]: x = '12.2';
          typeof x;

Out[118]: 'string'

In [119]: y = parseInt(x);    // truncates the decimal part
          y;

Out[119]: 12

In [120]: typeof y;

Out[120]: 'number'

In [123]: parseInt('12.9');   // truncates the decimal part

Out[123]: 12
```

```
In [121]: z = parseFloat(x);  // retains the decimal part also
          z;

Out[121]: 12.2

In [122]: typeof z;

Out[122]: 'number'
```

An alternative method of retrieving a number from a string is with the **+ (unary plus)** operator:

```
In [124]: '1.1' + '1.1'

Out[124]: '1.11.1'

In [125]: (+'1.1') + (+'1.1')  # parantheses are not required here

Out[125]: 2.2

In [127]: +'1.1' + +'1.1'

Out[127]: 2.2

In [126]: result = (+'1.1') + (+'1.1');
          typeof result;

Out[126]: 'number'
```

0.1.10 Operators

```
+     Addition Operator
-     Subtraction Operator
*     Multiplication Operator
/     Division Operator

=     Assignment Operator
===   Equality Operator (Value equivalence)
!     NOT
!==   Does not equal

val++ post increment
++val pre increment
val-- post decrement
--val pre decrement
```

```
In [35]: 6 + 9;

Out[35]: 15

In [36]: 6 + 9.0;
```

```
Out[36]: 15

In [37]: 6 + 9.5;

Out[37]: 15.5

In [38]: [6+9, 6+9.0, 6+9.5];

Out[38]: [ 15, 15, 15.5 ]

In [39]: [6 - 9, 6 - 9.0, 6 - 9.5];

Out[39]: [ -3, -3, -3.5 ]

In [40]: [6 * 9, 6 * 9.0, 6 * 9.5];

Out[40]: [ 54, 54, 57 ]

In [41]: [10/2, 10/2.0, 10/2.5];

Out[41]: [ 5, 5, 4 ]

In [42]: 10/3

Out[42]: 3.3333333333333335

In [43]: var myVariable = 3;  // assignment Operation

Out[43]: undefined

In [44]: myVariable === 4;  // value equvalence check

Out[44]: false

In [45]: var a = 3;
         var b = 3;

Out[45]: undefined

In [46]: a === b;

Out[46]: true
```

? How to check Object level equivalence.
? what is the importance of ==

```
In [47]: var myVariable = 3;

Out[47]: undefined

In [48]: (myVariable === 3);
```

```
Out[48]: true

In [49]: !(myVariable === 3);

Out[49]: false

In [50]: myVariable !== 3;

Out[50]: false
```

JavaScript is not a **Strictly-typed** language.

```
In [51]: 2 + 2;

Out[51]: 4

In [52]: 2 + "2";   // integer is converted to string, and string concatenation takes place

Out[52]: '22'

In [53]: 2.2 + '2';

Out[53]: '2.22'

In [54]: '33' + "44";

Out[54]: '3344'
```

Pre/post increments and decrements

```
In [55]: var myVal = 10;
         console.log("myVal is "+myVal)

myVal is 10


Out[55]: undefined

In [56]: myVal;

Out[56]: 10

In [57]: myVal++;  // result value before addition

Out[57]: 10

In [58]: myVal;

Out[58]: 11

In [59]: ++myVal; // results value after addition
```

```
Out[59]: 12

In [61]: console.log("myVal   =", myVal,"\nmyVal-- =", myVal--, "\nmyVal   =", myVal)

myVal   = 11
myVal-- = 11
myVal   = 10


Out[61]: undefined

In [65]: console.log("myVal      =", myVal,"\n--nmyVal   =", --myVal, "\nmyVal      =", myVal)

myVal      = 7
--nmyVal   = 6
myVal      = 6


Out[65]: undefined
```

0.2 Mixed Operators

```
+=      Addition assignment
-=      Subtraction assignment
*=      Multiplication assignment
/=      Division assignment

In [84]: x = 3;
         x += 4; // same as x = x + 4

Out[84]: 7

In [85]: x = 6;
         x -= 3; // same as x = x - 3

Out[85]: 3

In [86]: x = 2;
         x *= 3; // same as x = x * 3

Out[86]: 6

In [88]: x = 10;
         x /= 3; // same as x = x / 3

Out[88]: 3.3333333333333335
```

0.2.1 Comparision Operators

```
===     Strict equality ( checks value and datatype)
!==     Strict-non-equality
==      loose equality ( checks value only)
!=      loose non-equaility
<       Less than
>       Greater than
<=      Less than or equal to
>=      Greater than or equal to
```

NOTE: == and != are different from ===/!== , though all are valid.

```
In [89]: 1 == '1'

Out[89]: true

In [90]: 1 === '1'

Out[90]: false

In [91]: 1 != '1'

Out[91]: false

In [92]: 1 !== '1'

Out[92]: true

In [93]: 2 < 3

Out[93]: true

In [94]: 2 <= 3

Out[94]: true

In [100]: 2 <= '2'

Out[100]: true
```

? Numbers strings boolean which is greater

0.2.2 semi-colon

Required: When two statements are on the same line

```
In [ ]: var i = 0; i++          // <-- semicolon obligatory
                                //     (but optional before newline)
        var i = 0               // <-- semicolon optional
            i++                 // <-- semicolon optional
```

OPTIONAL: After the statements

```
In [52]: var i;                          // variable declaration
         i = 5;                          // value assignment
         i = i + 1;                      // value assignment
         i++;                            // same as above
         var x = 9;                      // declaration & assignment
         var fun = function() {var d = 23;};   // var decl., assignmt, and func. defin.

         //alert("hi");                    // function call. alert doesn't work in node.js; but
```

```
Out[52]: 6
```

AVOID: After a closing curly bracket }

```
In [57]: /*

         // NO semicolons after }:
         if  (...) {...} else {...}
         for (...) {...}
         while (...) {...}

         // BUT:
         do {...} while (...);

         // function statement:
         function (arg) { ...} // NO semicolon after }


         // BUT, the below statement is exception
         var obj = {};

         */
```

```
Out[57]: undefined
```

AVOID: After the round bracket of an if, for, while or switch statement

```
In [58]: if (0 === 1); { console.log("hi") }

         // equivalent to:

         if (0 === 1) /*do nothing*/ ;
         console.log ("hi");
```

```
hi
hi
```

```
Out[58]: undefined
```

```
for (var i=0; i < 10; i++)  {/*actions*/}       // correct
for (var i=0; i < 10; i++;) {/*actions*/}       // SyntaxError
```

0.3 Operator Precedence

```
-------------------------------------------------------
Operator type           Individual operators
-------------------------------------------------------
member                     . []
call / create instance   () new
negation/increment       ! ~ - + ++ -- typeof void delete
multiply/divide             * / %
addition/subtraction       + -
bitwise shift             << >> >>>
relational               < <= > >= in instanceof
equality                   == != === !==
bitwise-and                 &
bitwise-xor                 ^
bitwise-or               |
logical-and                 &&
logical-or               ||
conditional                 ?:
assignment               = += -= *= /= %= <<= >>= >>>= &= ^= |=
comma                       ,
```

```
In [66]: var a = 1;
         var b = 2;
         var c = 3;
```

```
Out[66]: undefined
```

```
In [67]: a + b * c     // 7     // default precedence
```

```
Out[67]: 7
```

```
In [68]: a + (b * c)   // 7    // evaluated by default like this
```

```
Out[68]: 7
```

now overriding precedence using Grouping Operator ()

```
In [69]: (a + b) * c   // 9   // addition before multiplication
```

```
Out[69]: 9
```

```
In [70]: a * c + b * c // 9   // this is equivalent to the above statement
```

```
Out[70]: 9
```

0.4 Conditions

if , else, else if

```
In [71]: var iceCream = 'chocolate';
         if (iceCream === 'chocolate') {
           console.log('Yay, I love chocolate ice cream!');
         } else {
           console.log('Awwww, but chocolate is my favorite...');
         }
```

```
Yay, I love chocolate ice cream!
```

```
Out[71]: undefined
```

```
In [72]: num1 = 12; num2 = 23;

         if (num1 < num2)
         { console.log("num1 is lesser than num2");}
         else if (num1 > num2)
         { console.log("num1 is greater than num2");}
         else
         { console.log("num1 is equal to num2");}
```

```
num1 is lesser than num2
```

```
Out[72]: undefined
```

0.4.1 Comprehensions

There are two types of comprehensions: 1. Array comprehensions

```
[for (x of y) x]
```

2. Generator comprehensions

```
(for (x of y) y)
```

```
In [82]: [for (i of [1, 2, 3]) i * i];   // doesn't work in node.js
```

```
         evalmachine.<anonymous>:1

         [for (i of [1, 2, 3]) i * i];

          ^^^
```

```
SyntaxError: Unexpected token for

    at createScript (vm.js:56:10)

    at Object.runInThisContext (vm.js:97:10)

    at run ([eval]:617:19)

    at onRunRequest ([eval]:388:22)

    at onMessage ([eval]:356:17)

    at emitTwo (events.js:106:13)

    at process.emit (events.js:191:7)

    at process.nextTick (internal/child_process.js:758:12)

    at _combinedTickCallback (internal/process/next_tick.js:73:7)

    at process._tickCallback (internal/process/next_tick.js:104:9)
```

```
In [83]: var abc = ['A', 'B', 'C'];
          [for (letters of abc) letters.toLowerCase()];
```

```
evalmachine.<anonymous>:2

[for (letters of abc) letters.toLowerCase()];

 ^^^



SyntaxError: Unexpected token for

    at createScript (vm.js:56:10)

    at Object.runInThisContext (vm.js:97:10)

    at run ([eval]:617:19)

    at onRunRequest ([eval]:388:22)

    at onMessage ([eval]:356:17)
```

```
    at emitTwo (events.js:106:13)

    at process.emit (events.js:191:7)

    at process.nextTick (internal/child_process.js:758:12)

    at _combinedTickCallback (internal/process/next_tick.js:73:7)

    at process._tickCallback (internal/process/next_tick.js:104:9)
```

1 Control flow and error handling

Block statement - group of statements

```
{
  statement_1;
  statement_2;
  .
  .
  .
  statement_n;
}
```

Important: JavaScript prior to ECMAScript2015 does not have block scope. Variables introduced within a block are scoped to the containing function or script, and the effects of setting them persist beyond the block itself. In other words, block statements do not define a scope. "Standalone" blocks in JavaScript can produce completely different results from what they would produce in C or Java.

```
In [183]: var x = 1;
          {
            var x = 2;
          }
          console.log(x); // outputs 2
```

```
2
```

```
Out[183]: undefined
```

Starting with ECMAScript2015, the **let** variable declaration is block scoped.
Falsy Values - false - undefined - null - 0 - NaN - empty string ("")
All other values, including all objects, evaluate to true when passed to a conditional statement.

```
In [184]: Boolean(false);
```

```
Out[184]: false
```

```
In [185]: Boolean(undefined);

Out[185]: false

In [186]: Boolean(null);

Out[186]: false

In [187]: Boolean(0);

Out[187]: false

In [188]: Boolean(NaN);

Out[188]: false

In [189]: Boolean("");

Out[189]: false

In [190]: if ( 2 > 3){
              2;
          }
          else {
              3;
          }

Out[190]: 3

In [191]: if ( 2 > 3){
              2;
          }
          else if ( 3 > 2) {
              3;
          }
          else {
              "both";
          }

Out[191]: 3

In [192]: if ( 2 > 3){
              2;
          }
          else if ( 3 > 2) {
              3;
          }

Out[192]: 3
```

NOTE: Observe that 'else' case is optional.

```
In [193]: var b = new Boolean(false);

Out[193]: undefined

In [194]: if (b) // this condition evaluates to true
          {
              "condition is true";
          }

Out[194]: 'condition is true'

In [195]: if (b == true) // this condition evaluates to false
          {
              "condition is true";
          }
          else {
              "condition is false";
          }

Out[195]: 'condition is false'

In [196]: if (!b) // this condition evaluates to false
          {
              "condition is true";
          }
          else {
              "condition is false";
          }

Out[196]: 'condition is false'
```

Switch statement

```
switch (expression) {
  case label_1:
    statements_1
    [break;]
  case label_2:
    statements_2
    [break;]
    ...
  default:
    statements_def
    [break;]
}
```

```
In [199]: fruittype = 'Bananas';

          switch (fruittype) {
```

```
            case 'Oranges':
              console.log('Oranges are $0.59 a pound.');
              break;
            case 'Apples':
              console.log('Apples are $0.32 a pound.');
              break;
            case 'Bananas':
              console.log('Bananas are $0.48 a pound.');
              break;
            case 'Cherries':
              console.log('Cherries are $3.00 a pound.');
              break;
            case 'Mangoes':
              console.log('Mangoes are $0.56 a pound.');
              break;
            case 'Papayas':
              console.log('Mangoes and papayas are $2.79 a pound.');
              break;
            default:
             console.log('Sorry, we are out of ' + fruittype + '.');
          }
```

```
Bananas are $0.48 a pound.
```

```
Out[199]: undefined
```

2 Exception Handling

You can throw exceptions using the throw statement and handle them using the try...catch statements.

Exception types 1. ECMAScript exceptions 2. DOMException and DOMError

throw statement : Use the throw statement to throw an exception.

SYNTAX: throw expression;

```
In [200]: /*
          throw 'Error2';   // String type
          throw 42;         // Number type
          throw true;       // Boolean type
          throw {toString: function() { return "I'm an object!"; } };
          */
```

```
Out[200]: undefined
```

```
In [201]: throw 'Error2';
```

```
In [202]: throw 42;
```

```
In [203]: throw new Error("This is network error");

        evalmachine.<anonymous>:1

        throw new Error("This is network error");

        ^



        Error: This is network error

            at evalmachine.<anonymous>:1:7

            at ContextifyScript.Script.runInThisContext (vm.js:25:33)

            at Object.runInThisContext (vm.js:97:38)

            at run ([eval]:617:19)

            at onRunRequest ([eval]:388:22)

            at onMessage ([eval]:356:17)

            at emitTwo (events.js:106:13)

            at process.emit (events.js:191:7)

            at process.nextTick (internal/child_process.js:758:12)

            at _combinedTickCallback (internal/process/next_tick.js:73:7)
```

```
In [211]: try{
              throw new Error("This is some error");
          }
          catch(ex) {
              //console.log(ex);
              console.log(ex.message);
          }
This is some error
```

```
Out[211]: undefined

In [213]: try{
              throw new Error("This is some error");
          }
          catch(ex) {
              //console.log(ex);
              console.log(ex.message);
          }
          finally{
              console.log("completed execution");
          }

This is some error
completed execution


Out[213]: undefined

In [214]: try{
              2+4;
          }
          catch(ex) {
              console.log(ex.message);
          }
          finally{
              console.log("completed execution");
          }

completed execution


Out[214]: 6
```

3 Loops and iteration

3.1 for statement

```
for ([initialExpression]; [condition]; [incrementExpression])
  statement
```

```
In [216]: for (i=0; i<7; i++){
              console.log(i);
          }

0
1
2
3
```

```
4
5
6
```

```
Out[216]: undefined
```

```
In [217]: for (i=0; i<7; ++i){
              console.log(i);
          }
```

```
0
1
2
3
4
5
6
```

```
Out[217]: undefined
```

```
In [218]: for (j=7; j>=0; --j) {
              console.log(j);
          }
```

```
7
6
5
4
3
2
1
0
```

```
Out[218]: undefined
```

3.2 do-while statement

```
do
  statement
while (condition);
```

the do loop iterates at least once

```
In [219]: var i = 0;
          do {
            i += 1;
            console.log(i);
          } while (i < 5);
```

```
1
2
3
4
5
```

```
Out[219]: undefined
```

3.3 while statement

```
while (condition)
  statement
```

```
In [220]: var n = 0;
          var x = 0;
          while (n < 3) {
            n++;
            x += n;
          }
```

```
Out[220]: 6
```

```
In [2]: i = 0;
        while (true) {
          console.log('Hello, world!');
          i++;
          if (i == 8){break;} // absence of this line leads to INFINITE LOOP
        }
```

```
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
Hello, world!
```

```
Out[2]: undefined
```

3.4 labeled statement

A label provides a statement with an identifier that lets you refer to it elsewhere in your program.

```
label :
   statement
```

```
In [6]: evenLabel:
            console.log("This is even number");

        oddLable:
            console.log("This is odd number");

        var i = 0;
        while (i < 10){
            if (i%2 == 0){
               break evenLabel;
            }
        }


        evalmachine.<anonymous>:10

               break labelmark;

                     ^^^^^^^^^



        SyntaxError: Undefined label 'labelmark'

            at createScript (vm.js:56:10)

            at Object.runInThisContext (vm.js:97:10)

            at run ([eval]:617:19)

            at onRunRequest ([eval]:388:22)

            at onMessage ([eval]:356:17)

            at emitTwo (events.js:106:13)

            at process.emit (events.js:191:7)

            at process.nextTick (internal/child_process.js:758:12)

            at _combinedTickCallback (internal/process/next_tick.js:73:7)

            at process._tickCallback (internal/process/next_tick.js:104:9)
```

3.5 break statement

```
break [label];
```

- When you use break without a label, it terminates the innermost enclosing while, do-while, for, or switch immediately and transfers control to the following statement.
- When you use break with a label, it terminates the specified labeled statement.

```
In [11]: var a = 'abcdpef';

         for (var i = 0; i < a.length; i++) {
           if (a[i] == 'p') {
             break;
           }
           else {
               console.log(a[i]);
           }
         }
```

```
a
b
c
d
```

```
Out[11]: undefined
```

```
In [10]: a[0];
```

```
Out[10]: 'a'
```

```
In [13]: var x = 0;
         var z = 0;
         labelCancelLoops: while (true) {
           console.log('Outer loops: ' + x);
           x += 1;
           z = 1;
           while (true) {
             console.log('Inner loops: ' + z);
             z += 1;
             if (z === 5 && x === 5) {
               break labelCancelLoops;
             } else if (z === 5) {
               break;
             }
           }
         }
```

```
Outer loops: 0
Inner loops: 1
Inner loops: 2
Inner loops: 3
Inner loops: 4
```

```
Outer loops: 1
Inner loops: 1
Inner loops: 2
Inner loops: 3
Inner loops: 4
Outer loops: 2
Inner loops: 1
Inner loops: 2
Inner loops: 3
Inner loops: 4
Outer loops: 3
Inner loops: 1
Inner loops: 2
Inner loops: 3
Inner loops: 4
Outer loops: 4
Inner loops: 1
Inner loops: 2
Inner loops: 3
Inner loops: 4
```

```
Out[13]: undefined
```

3.6 continue statement

```
continue [Label];
```

- when used without a label, it will terminate the current iteration of the loop
- when used with label, it goes to the label

```
In [14]: var i = 0;
         var n = 0;
         while (i < 5) {
           i++;
           if (i == 3) {
             continue;
           }
           n += i;
         }
```

```
Out[14]: 12
```

```
In [15]: var i = 0;
         var j = 10;
         checkiandj:
           while (i < 4) {
             console.log(i);
             i += 1;
```

```
            checkj:
              while (j > 4) {
                console.log(j);
                j -= 1;
                if ((j % 2) == 0) {
                  continue checkj;
                }
                console.log(j + ' is odd.');
              }
              console.log('i = ' + i);
              console.log('j = ' + j);
          }
```

```
0
10
9 is odd.
9
8
7 is odd.
7
6
5 is odd.
5
i = 1
j = 4
1
i = 2
j = 4
2
i = 3
j = 4
3
i = 4
j = 4
```

```
Out[15]: undefined
```

3.7 for - in statement

```
for (variable in object) {
  statements
}
```

```
In [19]: for (var y in "GOOD") {
             console.log(y);
         }
```

```
0
1
```

```
2
3


Out[19]: undefined

In [20]: for (var s in [1,2,3,4,5]) {
             console.log(s);
         }

0
1
2
3
4


Out[20]: undefined
```

3.8 for - of statement

```
for (variable of object) {
  statement
}

In [21]: var arr = [3, 5, 7];
         arr.foo = 'hello';

         console.log(arr);

         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
         }

[ 3, 5, 7, foo: 'hello' ]
0
1
2
foo
3
5
7


Out[21]: undefined
```

4 Functions

- Primitive parameters (such as a number) are passed to functions by value; the value is passed to the function, but if the function changes the value of the parameter, this change is not reflected globally or in the calling function.
- If you pass an object (i.e. a non-primitive value, such as Array or a user-defined object) as a parameter and the function changes the object's properties, that change is visible outside the function

```
In [22]: function hello()                // function without input arguments and no return
         {
             console.log("HELLO WORLD!");
         }

Out[22]: undefined

In [23]: hello()

HELLO WORLD!


Out[23]: undefined

In [24]: function hello(name)                // function with input arguments and no return
         {
             console.log("HELLO WORLD!"+name);
         }

Out[24]: undefined

In [25]: hello("Udhay")

HELLO WORLD!Udhay


Out[25]: undefined

In [26]: function hello(name)                // function with input arguments and return
         {
             return "HELLO WORLD!"+name;
         }

Out[26]: undefined

In [27]: hello("Udhay")

Out[27]: 'HELLO WORLD!Udhay'

In [28]: var hw = hello("Udhay")
         console.log(hw)
```

```
HELLO WORLD!Udhay


Out[28]: undefined

In [29]: function multiply(num1,num2) {
           var result = num1 * num2;
           return result;
         }

Out[29]: undefined

In [30]: multiply(4,7);

Out[30]: 28

In [31]: multiply(0.5,3);

Out[31]: 1.5

In [33]: var num1 = 100;

         function numChange(num1){
             console.log(num1);
             num1 = 99;
             console.log(num1);
         }

         numChange(num1)
         console.log("outside:"+num1);

100
99
outside:100


Out[33]: undefined

In [34]: function myFunc(theObject) {
           theObject.make = 'Toyota';
         }

         var mycar = {make: 'Honda', model: 'Accord', year: 1998};
         var x, y;

         x = mycar.make; // x gets the value "Honda"

         myFunc(mycar);
         y = mycar.make; // y gets the value "Toyota"
                         // (the make property was changed by the function)
```

```
Out[34]: 'Toyota'

In [35]: mycar;

Out[35]: { make: 'Toyota', model: 'Accord', year: 1998 }
```

4.1 Function Expressions

- While the function declaration above is syntactically a statement, functions can also be created by a function expression. Such a function can be anonymous; it does not have to have a name.

```
In [37]: var square = function(number) { return number * number; };
         var x = square(4);
         x;

Out[37]: 16
```

However, a name can be provided with a function expression

```
In [38]: var factorial = function fac(n) { return n < 2 ? 1 : n * fac(n - 1); };

         console.log(factorial(3));

6


Out[38]: undefined
```

Function expressions are convenient when passing a function as an argument to another function.

```
In [39]: function map(f, a) {
           var result = [], // Create a new Array
               i;
           for (i = 0; i != a.length; i++)
             result[i] = f(a[i]);
           return result;
         }

Out[39]: undefined

In [40]: var multiply = function(x) { return x * x * x; }; // Expression function.
         map(multiply, [0, 1, 2, 5, 10]);

Out[40]: [ 0, 1, 8, 125, 1000 ]
```

In JavaScript, a function can be defined based on a condition.

```
In [42]: var myFunc;
         var num = 0;

         if (num === 0) {
           myFunc = function(theObject) {
             theObject.make = 'Toyota';
           }
         }
```

```
Out[42]: [Function: myFunc]
```

A method is a function that is a property of an object.

4.2 Function Scope

```
In [43]: // The following variables are defined in the global scope
         var num1 = 20,
             num2 = 3,
             name = 'Chamahk';

         // This function is defined in the global scope
         function multiply() {
           return num1 * num2;
         }

         multiply(); // Returns 60

         // A nested function example
         function getScore() {
           var num1 = 2,
               num2 = 3;

           function add() {
             return name + ' scored ' + (num1 + num2);
           }

           return add();
         }

         getScore(); // Returns "Chamahk scored 5"
```

```
Out[43]: 'Chamahk scored 5'
```

4.3 Scope and the function stack

4.3.1 Recursions

```
In [45]: function loop(x) {
           if (x >= 10) // "x >= 10" is the exit condition (equivalent to "!(x < 10)")
```

```
            return;
          console.log(x);
          loop(x + 1); // the recursive call
        }
        loop(0);
```

```
0
1
2
3
4
5
6
7
8
9
```

```
Out[45]: undefined
```

4.3.2 Nested functions and closures

- You can nest a function within a function. The nested (inner) function is private to its containing (outer) function. It also forms a closure.
- The inner function can be accessed only from statements in the outer function.
- The inner function forms a closure: the inner function can use the arguments and variables of the outer function, while the outer function cannot use the arguments and variables of the inner function.

```
In [46]: function addSquares(a, b) {
           function square(x) {
             return x * x;
           }
           return square(a) + square(b);
         }
         a = addSquares(2, 3); // returns 13
         b = addSquares(3, 4); // returns 25
         c = addSquares(4, 5); // returns 41

         console.log(a,b,c);
```

```
13 25 41
```

```
Out[46]: undefined
```

Since the inner function forms a closure, you can call the outer function and specify arguments for both the outer and inner function:

```
In [47]: function outside(x) {
           function inside(y) {
             return x + y;
           }
           return inside;
         }
         fn_inside = outside(3); // Think of it like: give me a function that adds 3 to whatev
         result = fn_inside(5); // returns 8

         result1 = outside(3)(5); // returns 8
```

```
Out[47]: 8
```

4.3.3 Multiply-nested functions

```
In [48]: function A(x) {
           function B(y) {
             function C(z) {
               console.log(x + y + z);
             }
             C(3);
           }
           B(2);
         }
         A(1); // logs 6 (1 + 2 + 3)
```

```
6


Out[48]: undefined
```

4.3.4 Name conflicts

- When two arguments or variables in the scopes of a closure have the same name, there is a name conflict.
- More inner scopes take precedence, so the inner-most scope takes the highest precedence, while the outer-most scope takes the lowest. This is the scope chain.
- The first on the chain is the inner-most scope, and the last is the outer-most scope.

```
In [50]: function outside() {
           var x = 10;
           function inside(x) {
             return x;
           }
           return inside;
         }
         result = outside()(20); // returns 20 instead of 10

         console.log("result ="+result);
```

```
result =20
```

```
Out[50]: undefined
```

4.4 Promises

Starting with ECMAScript2015, JavaScript gains support for **Promise** objects allowing you to control the flow of deferred and **asynchronous operations**.

A Promise is in one of these states:

- **pending:** initial state, not fulfilled or rejected.
- **fulfilled:** successful operation
- **rejected:** failed operation.
- **settled:** the Promise is either fulfilled or rejected, but not pending.

```
In [1]: //syntax
        const myFirstPromise = new Promise((resolve, reject) => {
          // do something asynchronous which eventually calls either:
          //
          //   resolve(someValue); // fulfilled
          // or
          //   reject("failure reason"); // rejected
        });
```

```
Out[1]: undefined
```

```
In [19]: function my_first_promise(number){
             return new Promise((resolve, reject) => {
                 if (number > 5) {
                     console.log('--------resolving');
                     resolve(number);
                 } else {
                     console.log('--------rejecting');
                     reject(number);
                 }
             });
         }
```

```
Out[19]: undefined
```

```
In [24]: my_first_promise(6)
             .then((number)=>{
                 console.log("number:", number)
             })
             .catch((err)=>{
                 console.log("err:", err)
             });
```

```
--------resolving


Out[24]: Promise { <pending> }

number: 6


In [25]: my_first_promise(2)
             .then((number)=>{
                 console.log("number:", number)
             })
             .catch((err)=>{
                 console.log("err:", err)
             });

--------rejecting


Out[25]: Promise { <pending> }

err: 2


In [27]: my_first_promise(6)
             .then((fullfilled, notfullfilled)=>{
                 if(notfullfilled){
                     console.log('notfullfilled:', notfullfilled);
                 } else {
                     console.log("fullfilled:", fullfilled)
                 }
             })
             .catch((err)=>{
                 console.log("err:", err)
             });

--------resolving


Out[27]: Promise { <pending> }

fullfilled: 6


In [ ]:
```