SyntaxEDIT

```
var a, b, rest;
[a, b] = [1, 2];
console.log(a); // 1
console.log(b); // 2

[a, b, ...rest] = [1, 2, 3, 4, 5]
console.log(a); // 1
console.log(b); // 2
console.log(rest); // [3, 4, 5]

({a, b} = {a:1, b:2})
console.log(a); // 1
console.log(b); // 2
```

```
var a = 5;
var b = 10;
console.log("Fifteen is " + (a + b) + " and\nnot " + (2 * a + b) + ".");
// "Fifteen is 15 and
// not 20."

var a = 5;
var b = 10;
console.log(`Fifteen is ${a + b} and\nnot ${2 * a + b}.`);
// "Fifteen is 15 and
// not 20."


function tag(strings, ...values) {
  console.log(strings.raw[0]);
  // "string text line 1 \n string text line 2"
}

tag`string text line 1 \n string text line 2`;


String.raw`Hi\n${2+3}!`;
// "Hi\n5!"
```

```
var a = 5;
var b = 10;

function tag(strings, ...values) {
  console.log(strings[0]); // "Hello "
  console.log(strings[1]); // " world "
  console.log(strings[2]); // ""
  console.log(values[0]);  // 15
  console.log(values[1]);  // 50

  return "Bazinga!";
}

tag`Hello ${ a + b } world ${ a * b }`;
// "Bazinga!"
```

Template strings provide syntactic sugar for constructing strings. This is similar to string interpolation features in Perl, Python and more. Optionally, a tag can be added to allow the string construction to be customized, avoiding injection attacks or constructing higher level data structures from string contents.

```
// Basic literal string creation
`This is a pretty little template string.`

// Multiline strings
`In ES5 this is
 not legal.`

// Interpolate variable bindings
var name = "Bob", time = "today";
`Hello ${name}, how are you ${time}?`

// Unescaped template strings
```

```
String.raw`In ES5 "\n" is a line-feed.`

// Construct an HTTP request prefix is used to interpret the replacements and construction
GET`http://foo.org/bar?a=${a}&b=${b}
    Content-Type: application/json
    X-Credentials: ${credentials}
    { "foo": ${foo},
      "bar": ${bar}}`(myOnReadyStateChangeHandler);
```

In ES6, we would use export and import. For example, this is our library in the ES6 module.js file:

```
export var port = 3000
export function getAccounts(url) {
  ...
}
```

In the importer ES6 file main.js, we use import {name} from 'my-module'syntax. For example,

```
import {port, getAccounts} from 'module'
console.log(port) // 3000
```

Or we can import everything as a variable service in main.js:

```
import * as service from 'module'
console.log(service.port) // 3000
```

PATTERN 7: EXPORT A NAMED PROTOTYPE

```
        // qux.js
    var Qux = function () {};
    Qux.prototype.log = function () {
      console.log('baz!');
    };
    exports.Qux = Qux;
    // app.js
    var Qux = require('./qux.js').Qux;
    var qux = new Qux();
    qux.log();
```

PATTERN 6: EXPORT AN ANONYMOUS PROTOTYPE

```
 // doo.js
    var Doo = function () {};
    Doo.prototype.log = function () {
        console.log('doo!');
    }
    module.exports = Doo;

// app.js
    var Doo = require('./doo.js');
    var doo = new Doo();
    doo.log();
```

PATTERN 5: EXPORT A NAMED OBJECT

```
 // baz.js
    var Baz = function () {};
    Baz.prototype.log = function () {
      console.log('baz!');
    };

    exports.Baz = new Baz();

    // app.js
    var baz = require('./baz.js').Baz;
    baz.log();
```

PATTERN 4: EXPORT AN ANONYMOUS OBJECT

```
  // buz.js
 var Buz = function () {};
 Buz.prototype.log = function () {
   console.log('buz!');
 };
 module.exports = new Buz();

// app.js
 var buz = require('./buz.js');
 buz.log();
```

PATTERN 3: EXPORT A NAMED FUNCTION

```
 // first.js
 exports.fiz = function () {
   console.log('fiz!');
 }

// app.js
 var FOO = require('./first.js').fiz;
 FOO();
```

PATTERN 2: EXPORT AN ANONYMOUS FUNCTION

```
// bar.js
 module.exports = function () {
   console.log('bar!');
 }
```

```
// app.js
    var bar = require('./bar.js');
    bar();
```

PATTERN 1: DEFINE A GLOBAL

```
// foo.js
    foo = function () {
      console.log('foo!');
    }

    // app.js
    require('./foo.js');
    foo();
```

```
var msg=require('./foo.js')
console.log(msg.hello())
console.log(msg.bye())
```

```
module.exports.hello = function() {return 'hello'}

module.exports.bye = function() {return 'bye'}
```

exports is an alias to module.exports.
node automatically creates it as a convenient shortcut.
For assigning named properties, use either one

```
// Expression bodies

odds = evens.map(v => v + 1)
pairs = evens.map(v => ({ even: v, odd: v + 1 }))
nums = evens.map((v, i) => v + i)

// Statement bodies
nums.forEach(v => {
  if (v % 5 === 0)
    fives.push(v);
});

// Lexical this
var bob = {
  _name: "Bob",
  _friends: [],
  printFriends() {
    this._friends.forEach(f =>
      console.log(this._name + " knows " + f));
  }
};
```

```
const ask8=(question,yes,no)=>{
            if(confirm(question))yes()
            else no()
}
undefined
ask8("Do you agree?",()=>alert("yes"),()=>alert("no"));
undefined
```

```
let age=prompt("Enter your age")
undefined
age
"6"
let welcome =(age<18)? ()=>alert("hello") : ()=>alert("greetings");
undefined
welcome()
undefined
```

```
var
people=[{name:'eshan',age:20},{name:'suha',age:7},{name:'anish',age:20},{name:'vibha',age:19}
]
undefined
function teenager(person){
    return person.age > 10 && person.age < 2
}
undefined
var firstTeenager=people.find(teenager)
undefined
firstTeenager.name
```

```
var people=[{name:'eshan',age:20},{name:'suha',age:7},{name:'anish',age:20}]
undefined
```

```
people[0].name
"eshan"
```

```
var double=coll.map(v=> v*2)
undefined
double
(5) [20, 40, 60, 80, 100]
function print(val){
   console.log(val)
}
undefined
coll.forEach(print)
VM1926:2 10
VM1926:2 20
VM1926:2 30
VM1926:2 40
VM1926:2 50
undefined
```

```
var coll=[10,20,30,40,50]
undefined
var sum=coll.reduce((a,b)=>a+b)
undefined
sum
150
var even=coll.filter(v=> v%2 ==0)
undefined
even
(5) [10, 20, 30, 40, 50]
var double=coll.map(v=> v*2)
undefined
double
(5) [20, 40, 60, 80, 100]
```

```
var max=(a,b)=>a>b?a:b
undefined
max(100,200)
200
max(1000,200)
1000
```

```
let empty=()=>{}
undefined
empty
()=>{}
(()=>'foobar')()
"foobar"
var simple=a=>a>15 ? 15 : a1;
undefined
var simple=a=>a>15 ? 15 : a;
undefined
simple(16)
15
simple(14)
14
simple(11)
11
simple(16)
15
```

```
var coll=[1,2,3,4]
undefined
var arr=()=>coll[0]
undefined
arr
()=>coll[0]
```

```
arr()
1
function foo(n){
    var f=()=>coll[0]+n;
    return f()
}
undefined
foo(3)
4
```

```
function show(a){
    return a*100;
}
undefined
show(10)
1000
const a1=function(a){
    return a*100;
}
undefined
a1(30)
3000
const a2=(a)=>{
    return a*100;
}
undefined
a2(30)
3000
const a3=(a)=>a*100
undefined
a3(40)
4000
const a4=a=>a*100
undefined
a4(50)
5000
```

Arrows are a function shorthand using the => syntax. They are syntactically similar to the related feature in C#, Java 8 and CoffeeScript. They support both expression and statement bodies. Unlike functions, arrows share the same lexical this as their surrounding code.

Here’s the list of the top 10 best ES6 features for a busy software engineer (in no particular order):
Default Parameters in ES6
Template Literals in ES6
Multi-line Strings in ES6
Destructuring Assignment in ES6
Enhanced Object Literals in ES6
Arrow Functions in ES6
Promises in ES6
Block-Scoped Constructs Let and Const
Classes in ES6
Modules in ES6

1995: JavaScript is born as LiveScript
1997: ECMAScript standard is established
1999: ES3 comes out and IE5 is all the rage
2000–2005: XMLHttpRequest, a.k.a. AJAX, gains popularity in app such as Outlook Web Access (2000) and Oddpost (2002), Gmail (2004) and Google Maps (2005).
2009: ES5 comes out (this is what most of us use now) with forEach, Object.keys, Object.create (specially for Douglas Crockford), and standard JSON
2015: ES6/ECMAScript2015 comes out; it has mostly syntactic sugar, because people weren’t able to agree on anything more ground breaking (ES7?)

ECMA [European Computer Manufacturers Association]

ECMAScript 2015 is the newest version of the ECMAScript standard. This standard was ratified in June 2015. ES2015 is a significant update to the language, and the first major update to the language since ES5 was standardized in 2009. Implementation of these features in major JavaScript engines is underway now.