JavaScript

ES6

EcmaScript 2015

LUXOFT

LXFT LISTED NYSE

let operator

```
function func() {
    if (true) {
        let tmp = 123;
    }
    console.log(tmp); // ReferenceError: tmp is not defined
  }

function func() {
    if (true) {
        var tmp = 123;
    }
    console.log(tmp); // 123
  }
```

let operator

```
function func() {
    let foo = 5;
    if (···) {
       let foo = 10; // shadows outer `foo`
       console.log(foo); // 10
    }
    console.log(foo); // 5
  }
```

const

```
let foo = 'abc';
foo = 'def';
console.log(foo); // def

const foo2 = 'abc';
foo2 = 'def'; // TypeError
```

Arrow function

```
f = v => v + 1;                          var f = function (v) { return v + 1; })
```

Usage example:

```
var arr = [1,2,3];
arr.forEach(i=>console.log(i));
```

Arrow function with multiple parameters

```
f = (x,y) => x+y;
f(1,2) === 3;
```

Arrow function with function body

```
f = (x,y) => {
  console.log(x,y);
  return x+y;
}
```

Property Shorthand

```
obj = { x, y }
```

same as `obj = { x: x, y: y };`

Computed Property Names

```
obj = {
  foo: "bar",
  [ "prop_" + foo() ]: 42
}
```

```
obj = { foo: "bar" };
obj[ "prop_" + foo() ] =
42;
```

Method Properties

```
obj = {
  foo (a, b) { … },
  bar (x, y) { … },
  *quux (x, y) { … }
}
```

```
obj = {
  foo: function (a, b) { … },
  bar: function (x, y) { … },
  // quux: no equivalent in ES5 …
};
```

Array matching

```
var list = [ 1, 2, 3 ]
var [ a, , b ] = list
[ b, a ] = [ a, b ]
```

```
var list = [ 1, 2, 3 ];
var a = list[0], b = list[2];
var tmp = a; a = b; b = tmp;
```

Object matching

```
var { op, lhs, rhs } =
getASTNode()
```

```
var tmp = getASTNode();
var op = tmp.op;
var lhs = tmp.lhs;
var rhs = tmp.rhs;
```

Fail-soft matching

```
var list = [ 7, 42 ]
var [ a = 1, b = 2, c = 3, d ] = list
// a === 7 b === 42
// c === 3 d === undefined
```

```
var list = [ 7, 42 ];
var a = list[0] !== undefined ? list[0] : 1;
var b = list[1] !== undefined ? list[1] : 2;
var c = list[2] !== undefined ? list[2] : 3;
var d = list[3] !== undefined ? list[3]
      : undefined;
```

Array: new functions

```
[ 1, 3, 4, 2 ].find(x => x > 3) // 4
```

```
[ 1, 3, 4, 2 ].filter(function (x) {
  return x > 3; })[0]; // 4
```

Object assigning

```
var dst = { quux: 0 }
var src1 = { foo: 1, bar: 2 }
var src2 = { foo: 3, baz: 4 }
Object.assign(dst, src1, src2)

dst.quux === 0
dst.foo === 3
dst.bar === 2
dst.baz === 4
```

```
var dst = { quux: 0 };
var src1 = { foo: 1, bar: 2 };
var src2 = { foo: 3, baz: 4 };
Object.keys(src1).forEach(function(k) {
  dst[k] = src1[k]; });
Object.keys(src2).forEach(function(e) {
  dst[k] = src2[k]; });
```

String searching

```
"hello".startsWith("ello", 1) // true
"hello".endsWith("hell", 4) // true
"hello".includes("ell") // true
"hello".includes("ell", 1) // true
"hello".includes("ell", 2) // false
```

```
"hello".indexOf("ello") === 1; // true
"hello".indexOf("hell") === (4 - "hell".length);
"hello".indexOf("ell") !== -1; // true
"hello".indexOf("ell", 1) !== -1; // true
"hello".indexOf("ell", 2) !== -1; // false
```

Set

```
let s = new Set()
s.add("hello").add("goodbye").add("hello")
s.size === 2
s.has("hello") === true
for (let key of s.values()) // insertion order console.log(key)
```

Map

```
let m = new Map()
m.set("hello", 42)
m.set(s, 34)
m.get(s) === 34
m.size === 2
for (let [ key, val ] of m.entries()) console.log(key + " = " + val)
```

WeakSet/WeakMap

```
var weakSet = new WeakSet()
a = {}; // only objects allowed
weakSet.add(a);
weakSet.has(a); // true
a = null; // now a can be garbage collected
for (e in weakSet) console.log(e); // not working: WeakSet is not itarable
```

String Interpolation

```
var customer = { name: "Foo" }
var card = { amount: 7,
  product: "Bar",
  unitprice: 42 }
message = `Hello ${customer.name},
want to buy ${card.amount}
${card.product} for a total of
${card.amount * card.unitprice}
bucks?`
```

```
var customer = { name: "Foo" };
var card = { amount: 7,
    product: "Bar",
    unitprice: 42 };
message = "Hello " + customer.name + ",\n" +
"want to buy " + card.amount + " " +
card.product + " for\n" + "a total of " +
(card.amount * card.unitprice) + " bucks?";
```

New number functions

```
Number.isNaN(42) === false
Number.isNaN(NaN) === true

Number.isFinite(Infinity) === false
Number.isFinite(-Infinity) === false
Number.isFinite(NaN) === false
Number.isFinite(123) === true

Number.isSafeInteger(42) === true
Number.isSafeInteger(9007199254740992) === false

console.log(0.1 + 0.2 === 0.3) // false
console.log(Math.abs((0.1 + 0.2) - 0.3) < Number.EPSILON)
// true
```

Default Parameter Values

```
function f (x, y = 7, z = 42) {
  return x + y + z
}
f(1) === 50
```

```
function f (x, y, z) {
  if (y === undefined) y = 7;
  if (z === undefined) z = 42;
  return x + y + z;
}
f(1) === 50;
```

Rest Parameters

```
function f (x, y, ...a) {
  return (x + y) * a.length
}

f(1, 2, "hello", true, 7) === 9
```

```
function f (x, y) {
  return (x + y) * (a.length-2);
}

f(1, 2, "hello", true, 7) === 9;
```

Spread Operator

```
var params = [ "hello", true, 7 ]
var other = [ 1, 2, ...params ] // [ 1, 2, "hello", true, 7 ]
f(1, 2, ...params) === 9
```

Using this in callbacks

```
arr = [1,2,3];
arr.summarize = function() {
   this.sum = 0;
   this.forEach(function(e) { this.sum = this.sum+e; } );
   // Callbacks are executed in their own context, this points to function, not arr
}
```

workaround:

```
arr.summarize = function() {
   this.sum = 0;
   var self = this;
   this.forEach(function(e) { self.sum = self.sum+e; } );
}
```

another workaround:

```
   this.forEach(function(e) { this.sum = this.sum+e; }.bind(this) );
```

lexical scoping “this”

```
arr.summarize = function() {
    this.sum = 0;
    this.forEach(e=>{ this.sum = this.sum+e; });
}
```

Using classes

```
class Shape {
    constructor (id, x, y) {
        this.id = id
        this.move(x, y)
    }
    move (x, y) {
        this.x = x
        this.y = y
    }
}
```

```
var Shape = function (id, x, y) {
    this.id = id;
    this.move(x, y);
};
Shape.prototype.move = function (x, y) {
    this.x = x;
    this.y = y;
};
```

Inheritance

```
class Rectangle extends Shape {
    constructor (id, x, y, width, height) {
        super(id, x, y)
        this.width  = width
        this.height = height
    }
}
class Circle extends Shape {
    constructor (id, x, y, radius) {
        super(id, x, y)
        this.radius = radius
    }
}
```

Base class access

```
class Shape {
    …
    toString () {
        return `Shape(${this.id})`
    }
}
class Rectangle extends Shape {
    constructor (id, x, y, width, height) {
        super(id, x, y)
        ...
    }
    toString () {
        return "Rectangle > " + super.toString()
    }
}
class Circle extends Shape {
    constructor (id, x, y, radius) {
        super(id, x, y)
        ...
    }
    toString () {
        return "Circle > " + super.toString()
    }
}
```

Static members

```
class Circle extends Shape {
    static defaultCircle () {
        return new Circle("default", 0, 0, 100)
    }
}
var defRectangle = Rectangle.defaultRectangle()
var defCircle    = Circle.defaultCircle()
```

Getters/setters

```
class Rectangle {
    constructor (width, height) {
        this._width  = width
        this._height = height
    }
    set width  (width)  { this._width = width               }
    get width  ()       { return this._width               }
    set height (height) { this._height = height             }
    get height ()       { return this._height              }
    get area   ()       { return this._width * this._height }
}
var r = new Rectangle(50, 20)
r.area === 1000
```

Modules import/export

```
//lib/math.js
export function sum (x, y) { return x + y }
export var pi = 3.141593

//  someApp.js
import * as math from "lib/math"
console.log("2π = " + math.sum(math.pi, math.pi))

//  otherApp.js
import { sum, pi } from "lib/math"
console.log("2π = " + sum(pi, pi))
```

Marking a value as the default exported value

```
//  lib/mathplusplus.js
export * from "lib/math"
export var e = 2.71828182846
export default (x) => Math.exp(x)

//  someApp.js
import exp, { pi, e } from "lib/mathplusplus"
console.log("e^{π} = " + exp(pi))
```

Practice

Exercise 1, 2

Promises: built-in support

```
function msgAfterTimeout (msg, who, timeout) {
 return new Promise((resolve, reject) => {
   setTimeout(() => resolve(`${msg} Hello ${who}!`), timeout)
 })
}

msgAfterTimeout("", "Foo", 100).then((msg) =>
    msgAfterTimeout(msg, "Bar", 200)
).then((msg) => {
    console.log(`done after 300ms:${msg}`)
});
```

Practice

Exercise 3

Generators

```
function* range (start, end, step) {
    while (start < end) {
        yield start
        start += step
    }
}

for (let i of range(0, 10, 2)) {
    console.log(i) // 0, 2, 4, 6, 8
}
```

```
function* genFunc() {
        yield 'a';
        yield 'b';
        return 1;
    }
genObj = genFunc();
genObj.next() // {value: "a", done: false}
genObj.next() // {value: "b", done: false}
genObj.next() //  {value: 1, done: true}
arr = [...genFunc()]; // ['a', 'b']
```

Generators: example of use

```
function* objectEntries(obj) {
    // In ES6, you can use strings
    // or symbols as property keys,
    // Reflect.ownKeys() retrieves both
    let propKeys = Reflect.ownKeys(obj);

    for (let propKey of propKeys) {
        yield [propKey, obj[propKey]];
    }
}

let jane = { first: 'Jane', last: 'Doe' };
for (let [key,value] of objectEntries(jane)) {
    console.log(`${key}: ${value}`);
}
// Output:
// first: Jane
// last: Doe
```

Generators: recursion

```
function* foo() {
    yield 'a';
    yield 'b';
}
function* bar() {
    yield 'x';
    yield* foo();
    yield 'y';
}

// Collect all values yielded by bar() in an
array
let arr = [...bar()];
// ['x', 'a', 'b', 'y']
```

Generators: yielding arrays

```
function* bla() {
    yield 'sequence';
    yield* ['of', 'yielded'];
    yield 'values';
}

let arr = [...bla()];
// ['sequence', 'of', 'yielded', 'values']
```

throw() signals an error

```
function* genFunc1() {
    try {
        console.log('Started');
        yield; // (A)
    } catch (error) {
        console.log('Caught: ' + error);
    }
}

> let genObj1 = genFunc1();

> genObj1.next()
Started
{ value: undefined, done: false }

> genObj1.throw(new Error('Problem!'))
Caught: Error: Problem!
{ value: undefined, done: true }
```

Practice

Exercise 4

Generators for async calls

```
function asyncAdd(x, y) {
  setTimeout(function() { it.next(x+y);},
1000);
}

function *process() {
  var res = yield asyncAdd(1,2);
  var res2 = yield asyncAdd(res,3);
  console.log(res2);
}

it = process();
it.next();
```

```
function asyncAdd(x, y, f) {
  setTimeout(function() { f(x+y);},
1000);
}

asyncAdd(1, 2, function(res) {
  asyncAdd(res, 3, function(res) {
     console.log(res);
  })
});
```

Generators for async calls

```
function add(x,y) {
  return new Promise(function(resolve,reject) {
    setTimeout(()=>resolve(x+y), 1000);
  });
}

run(function *main() {
   var res1 = yield add(1,2);
   var res2 = yield add(res1, 3);
   console.log(res2);
});

function run(g) {
   var it = g(), ret;
   var iterate = (val)=>{
        ret = it.next(val);
        if (!ret.done) ret.value.then( iterate ); // wait on the promise
           else setTimeout(()=>iterate(ret.value), 0);  // avoid synchronous
recursion
   }
   iterate();
}
```

Practice

Exercise 5

Generators for async calls: add reject processing

```
function run(g) {
    var it = g(), ret;
    var exception = (e)=>it.throw(e);
    var iterate = (val)=>{
          ret = it.next(val);
          if (!ret.done) ret.value.then( iterate, exception); // wait on the promise
             else setTimeout(()=>iterate(ret.value), 0);  // avoid synchronous recursion
    }
    iterate();
}
```

Generators for async calls with exceptions

```
function add(x,y) {
  return new Promise(function(resolve,reject) {
    setTimeout(()=>x>0?resolve(x+y):reject("x should be >0"), 1000);
  });
}

run(function *main() {
   try {
     var res1 = yield add(0,2);
   } catch(err) {
     console.log( "Error: " + err );
     return;
   }

   var res2 = yield add(res1, 3);
   console.log(res2);
});
```

Practice

Exercise 6

Example: fetchJson with promises

```
function fetchJson(url) {
    return fetch(url)
    .then(request => request.text())
    .then(text => {
        return JSON.parse(text);
    })
    .catch(error => {
        console.log(`ERROR: ${error.stack}`);
    });
}

fetchJson('http://example.com/some_file.json')
.then(obj => console.log(obj));
```

Example: fetchJson with co library

```
const fetchJson = co(function* () {
    try {
        let request = yield fetch(url);
        let text = yield request.text();
        return JSON.parse(text);
    }
    catch (error) {
        console.log(`ERROR: ${error.stack}`);
    }
});
```

Poposed features of EcmaScript 2017 / ES7+

ES7 async/await

```
function add(x,y) {
   return new Promise(function(resolve,reject) {
      setTimeout(()=>x>0?resolve(x+y):reject("x should be >0"), 1000);
   });
}

async function main() {
    var res = await add(1, 2);
    var res2 = await add (res, 3);
    console.log( res2 ); //6
}

main();
```

Example: fetchJson with async/await (ES7+)

```
async function fetchJson(url) {
    try {
        let request = await fetch(url);
        let text = await request.text();
        return JSON.parse(text);
    }
    catch (error) {
        console.log(`ERROR: ${error.stack}`);
    }
}
```

async declaration:

- Async function declarations: async function foo() {}
- Async function expressions: const foo = async function () {};
- Async method definitions: let obj = { async foo() {} }
- Async arrow functions: const foo = async () => {};

Practice

Exercise 7

Exponentiation operator

x ** y

produce the same result as Math.pow(x,y)

Trailing commas in function parameters and arrays/objects

```
let obj = {
 first: 'Jane',
 last: 'Doe',
};

let arr = [
            'red',
            'green',
            'blue',
        ];

console.log(arr.length); // 3
```

```
function foo(
                param1,
                param2,
) {}

foo(
     'abc',
     'def',
);
```

Decorators: @readonly

```
function readonly(target, key, descriptor) {
    descriptor.writable = false;
    return descriptor;
}
class Meal {
    @readonly
    entree='salad';
}
// this is the same as
Object.defineProperty(Meal.prototype, ’entree',

    // this is descriptor:
    { value: 'salad',  enumerable: false, configurable: true,  writable: false  });

// let’s check it!
var dinner = new Meal();
dinner.entree = ’soup'; // Cannot assign to read only property
```

Decorators: enrich class

```
function superhero(target) {
    target.isSuperhero = true;
    target.power = "flight";
}

@superhero
class MySuperHero {}
console.log(MySuperHero.isSuperhero); // true
```

Decorators: enrich class with parameter

```
function superhero(isSuperhero) {
    return function (target) {
        target.isSuperhero = isSuperhero
    }
}

@superhero(true)
class MySuperheroClass { }
console.log(MySuperheroClass.isSuperhero); // true

@superhero(false)
class MySuperheroClass { }
console.log(MySuperheroClass.isSuperhero); // false
```

Decorators: enrich class objects

```
@makesPhonecalls
class Cellphone {
    constructor() {
        this.model = "Samsung"
        this.storage = 16
    }
}
function makesPhonecalls(target) {
    let callNumber = function(number) {
        return `calling ${number}`
    }
    // Attach it to the prototype
    target.prototype.callNumber = callNumber
}
```

Decorators: limit access

```
function adminOnly(user) {
    return function (target) {
        if (!user.isAdmin) {
            log('You do not have sufficient privileges!');
            return false;
        }
    }
}
@adminOnly(currentUser)
function deleteAllUsers() {
    users.delete().then((response) => {
        log('You deleted everyone!');
    });
}
```

Decorator as a wrapper

```
@logger
function logMe() {
    console.log('I want to be logged');
}
// Decorator function for logging
function logger(target, name, descriptor) {
    // obtain the original function
    let fn = descriptor.value;
    // create a new function that wraps the original function
    let newFn  = function() {
        console.log('starting %s', name);
        fn.apply(target, arguments);
        console.log('ending %s', name);
    };
    // we then overwrite the origin descriptor value and return new
    descriptor.value = newFn;
    return descriptor;
}
```

Decorator as a wrapper – customization with parameters

```
@logger('custom message starting %s', 'custom message ending %s')
function logMe() {
  console.log('I want to be logged');
}
function logger(startMsg, endMsg) {
  return function(target, name, descriptor) {
    let fn = descriptor.value;
    let newFn  = function() {
      console.log(startMsg, name);
      fn.apply(target, arguments);
      console.log(endMsg, name);
    };
    descriptor.value = newFn;
    return descriptor;
  }
}
```