**ES6**

1. Default Parameters

function(height = 50, color = 'red', url = 'http://azat.co') {}

1. Template Literals

ES5 - var name = 'Your name is ' + first + ' ' + last + '.'

ES6 - var name = `Your name is ${first} ${last}.`

1. Multiline strings – use backticks
2. Destructuring Assignment

ES5

var username = req.username;

var password = req.password;

ES6

var { username, password } = req;

1. Shorthands for assignment getAccounts: getAccounts, becomes just getAccounts.

ES5

var servicebase = {port: 3000, url: 'azat.co'};

var getAccounts = function(){return [1,2,3]}

Es5

Var es5 = {

port\_: servicebase.port,

getAccounts: getAccounts,

};

ES6

var es6 = {

\_proto\_: servicebase,

getAccounts

};

1. Arrow Functions - Using arrows functions in ES6 allows us to stop using storing scopr into variable E.g var self = this.

ES5

var \_this = this

$('.btn').click(function(event){

\_this.sendData()

})

var ids = ['5632953c4e345e145fdf2df8','563295464e345e145fdf2df9']

var messages = ids.map(function(value) {

return"ID is " + value // explicit return

});

ES6

$('.btn').click((event) =>{

this.sendData()

})

var ids = ['5632953c4e345e145fdf2df8','563295464e345e145fdf2df9']

var messages = ids.map(value => `ID is ${value}`) // implicit return

1. Var is function scoped where let is block scoped

Es5

Function test() {

Var amount = 100;

{

Var amount = 1000;

}

Console.log(amount) // log 1000

}

Es6

Function test() {

var amount = 100;

{

let amount = 1000;

}

Console.log(amount) // log 100

}

1. Classes
2. Modules
3. No longer semicolon required at end of statements
4. Generators

function \*foo(x) {

var y = 2 \* (yield (x + 1));

var z = yield (y / 3);

return (x + y + z);

}

var it = foo( 5 );

console.log( it.next() ); // { value:6, done:false }

console.log( it.next( 12 ) ); // { value:8, done:false }

console.log( it.next( 13 ) ); // { value:42, done:true }

1. Spread operator
2. Promises
3. var x = 'global';

let y = 'global';

console.log(this.x); // "global"

console.log(this.y); // undefined

1. **let** bindings are not subject to **Variable Hoisting**, which means that **let** declarations do not move to the top of the current execution context. Referencing the variable in the block before the initialization results in a [ReferenceError](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/ReferenceError)(contrary to a variable declared with [var](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var" \l "var_hoisting), which will just have the undefined value).