WHAT'S NEW IN JAVASCRIPT

(ES 2015/2016/2017)

RAJU GANDHI

ES2016

LET

```
(function() {
    console.log('Before Block: a is ' + a);
    //console.log('Before Block: b is ' + b); // ReferenceError
    {
        // b still gets hoisted
        console.log('b is Hoisted ' + b);
        var a = 1;
        let b = 2;
        console.log('Inside Block: a is ' + a);
        console.log('Inside Block: b is ' + b);
    }
    console.log('After Block: ' + a); // 1
    //console.log('After Block: ' + b); // ReferenceError
})();
```

CONST

```
(function() {
    console.log('Before Block: a is ' + a);
   {
        var a = 1;
         const b = 2;
         console.log('Inside Block: a is ' + a);
         console.log('Inside Block: b is ' + b);

         //const b = 3; // TypeError
         //b = 4; // Does NOT ERROR, but also does NOTHING
    }
    console.log('After Block: ' + a); // 1
})();
```

LET AND CONST

- LET IS BLOCK SCOPED
- CONST HAS THE SAME SEMANTICS AS LET

DEFAULT VALUES

```javascript
// Parameters can have default values
function defaultOne(x = 1, y = 2) {
    return [x, y];
}

console.log("No args: " + defaultOne());
console.log("x = 10: " + defaultOne(10));
console.log("Both args: " + defaultOne(10, 20));

// parameters can see values of preceding parameters
function defaultOne(x = 1, y = x * 2) {
    return [x, y];
}

console.log("No args: " + defaultOne())
console.log("x = 10: " + defaultOne(10));
console.log("Both args: " + defaultOne(10, 30));
```

ARROW FUNCTIONS

```
// single args don't need parentheses
clear();
let fatOne = val => { return val };
console.log(fatOne(10)); // 10
```

```
// multiple args have to have parentheses
clear();
let fatTwo = (a, b) => { return [a, b] }
console.log(fatTwo('js', 'next'));

// no args have to have parentheses
// without curly brackets 'return' is implicit
clear();
let fatFive = () => 'I have no args';
console.log(fatFive()); // I have no args
```

```
function Person(name) {
    this.name = name;
    this.sayHello = () => "Hello! My name is " + this.name;
}

let me = new Person("raju");

// CANNOT Change 'this'
let obj = {};
me.sayHello.call(obj); // "Hello! My name is raju"
```

USECASES

```
[1,2,3,4]
  .filter(n => n > 2)
  .map(n => n * n)
  .reduce(((acc, n) => acc + n), 0);
```

ARROW FUNCTIONS

- 'THIS' IS BOUND TO THE CONTAINING SCOPE
- SEMANTICALLY EQUIVALENT OF "FUNCTION () { ... }.BIND(THIS)"
- CANNOT BE 'NEW'-ED
- HAVE NO 'PROTOTYPE' OR 'CONSTRUCTOR'

DESTRUCTURING

OBJECTS

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

// alternatively
let {x, y} = point;

console.log(x); // 1
console.log(y); // 2
```

ARRAYS

```
let color = [255, 255, 100, 0.5];
let [r, g, , a] = color
console.log(r); // 255

// nested works just as well
clear();
let [[, c], y] = [['a', 'b'], 2];
console.log(c); // 'b'

// easy swap
clear();
let x = 10, y = 20;
[y, x] = [x, y];
console.log(x); // 20
console.log(y); // 10
```

USECASES

```
clear();
function destruct({name, age}) {
    return "My name is " + name + " and I am " + age;
}

const me = { name: "Raju", age: 34 };
// My name is Raju and I am 34
console.log(destruct(me));
```

USECASES

```
clear();
// destructuring in forEach (short version)
let students = [{name: 'raju', age: 34}, {name: 'bob', age: 25}];
students.forEach(({name, age}) => console.log(name+' '+age));


// destructuring in forEach (long version)
clear();
let students = [{name: 'raju', age: 34}, {name: 'bob', age: 25}];
students.forEach(({name:n, age:a}) => console.log(n+' '+a));
```

USECASES

```
// destructuring with regex
clear();
const PHONE_MATCH =  /^(\d{3})-(\d{3})-(\d{4})$/;
let [, city, area, local]  = PHONE_MATCH.exec("614-555-1234");
console.log([city, area, local]);
```

SPREAD OPERATOR

```
let arr = [1,2,3];

console.log(arr); // [1, 2, 3]
console.log(...arr); // 1, 2, 3
```

USECASES

```
let arr = [1,2,3];

// useful where you have multiple arity functions
let other = [4,5,6];
other.push(...arr);
console.log(other); // [4, 5, 6, 1, 2, 3]

// great for concatenating arrays
let combo = [10, 9, 7, ...other];
console.log(combo); // [10, 9, 7, 4, 5, 6, 1, 2, 3]
```

SPREAD ARGS

```javascript
function something(a, ...b) {
    return [a, b];
}

console.log(something(1, 2, 3, 4)); // [1, [2, 3, 4]]
```

USECASES

```
function something(a, ...b) {
    return [a, b.map(n => n * 2)];
}
console.log(something(1, 2, 3, 4)); // [1, [4, 6, 8]]

// with spread operator
function somethingElse(a, ...b) {
    return [a, ...b.map(n => n * 2)];
}
console.log(something(1, 2, 3, 4)); // [1, 4, 6, 8]
```

SPREAD ARGS

- FINAL (???) NAIL IN THE COFFIN FOR 'ARGUMENTS'
- ARE ARRAYS
- CAN BE USED IN 'ARROW' FUNCTIONS

MAPS

```
const myMap = new Map();
// set - Type specific (unlike objects)
myMap.set(true, "Boolean True");
myMap.set("true", "String True");

// get
console.log(myMap.get(true));
console.log(myMap.get("true"));
```

WEAK MAPS

```
// declare a map of users to passwords
const userPwds = new WeakMap();

function User(userSuppliedPwd) {
    const attrs = { pwd: userSuppliedPwd };
    userPwds.set(this, attrs);
}

User.prototype.authenticate = function(pwd) {
    const storedPwd = userPwds.get(this);
    return pwd == storedPwd.pwd;
}

let raju = new User("password");
console.dir(raju);
console.log("Can I authenticate? " + raju.authenticate("password"));
// if the user gets deleted
// raju = null;
// the weak map will 'eventually' drop the associated password
console.log(userPwds.has(raju));
```

SETS

```
clear();
var mySet = new Set();

mySet.add(true);
mySet.add("true");
mySet.add(true);

console.log("Size: " + mySet.size);

// get
console.log(mySet.has(true));
console.log(mySet.has("true"));
```

WEAK SETS

```
// DOES NOT WORK WITH ANY BROWSER
// Just like WeakMaps you can ONLY add Objects
let ws = new WeakSet();
let obj = {};

ws.add(obj);
ws.has(obj); // returns true

ws.delete(obj);

ws.clear();
```

MUCH MORE ...

OBJECT LITERALS

SYMBOLS

ITERATORS & GENERATORS

CLASSES

MODULES

…

STRING TEMPLATES
PROXIES
PROPER TAILS CALLS
UNICODE SUPPORT

...

RESOURCES

MDN DOCS

COMPATIBILITY MATRIX

EXPLORING ES6

EXPLORING ES 7 & 8

UNDERSTANDING ECMASCRIPT 6

MOZILLA ES6 FEATURES IN DEPTH ARTICLE SERIES

BABEL (TRANSPILING)

CREDITS

THEME - HTTPS://SPEAKERDECK.COM/PHILHAWKSWORTH/EXCESSIVE-ENHANCEMENT-GOTHAMJS

THANKS