ES5 & ES6 Advanced JavaScript

The Complete Web Developer in 2018

The Complete Web Developer in 2018
Zero to Mastery
Andrei Neagoie
Lecture Notes by Stephanie

ES5 and ES6

ECMAScript = JavaScript

Currently at ES7 version, but ES6 is still fairly new. We can write all our coding using latest versions regardless of browser compatibility because of….

BABEL

BABEL - reads JavaScript file line by line, and then changes it to a version of JavaScript that works on all browsers. Allows us to write our code using the latest version and convert it to a browser-compatible version when we are done.

https://babeljs.io/

Advanced JavaScript Variables

let + const *(ES5/ES6 only)*

"**const**" for variables that will not change (constants)
"**let**" for variables that change (variables)

"const" is good for teamwork, preventing accidental reuse/ reassignment of a variable

When using "const" for objects, we CAN change the properties of the object, you just can't reassign the entire object variable…

```
> const obj = {
      player: "bobby",
      experience: 100,
      wizardLevel: false
  }
```

```
> obj
◂ ▸{player: "bobby", experience: 100, wizardLevel: false}
```

Cannot reassign…

```
> obj = {
      game: "monopoly"
  }
```

```
✖ ▸Uncaught TypeError: Assignment to constant variable.        VM229:1
      at <anonymous>:1:5
```

But we **CAN** change the properties for const object…

```
> obj.player = "stephanie";
  obj.experience = 80;
  obj.wizardLevel = true; // allowed to change properties of objects
```

```
> obj
◂ ▸{player: "stephanie", experience: 80, wizardLevel: true}
```

Scope of “let” and “const” (ES5/ES6 only)

With variable, we can only create a new scope within a function. But with 'let', we can create a new scope within an if-statement.

Use “let” to create scope:

```
> const player = 'bobby';
  let experience = 100;
  let wizardLevel = false;

  console.log("outside1",wizardLevel);

  if (experience > 90) {
      let wizardLevel = true; // scope is only within if statemt
      console.log("inside2",wizardLevel);
  }

  console.log("outside3",wizardLevel);
```

```
outside1 false                    VM188:5
inside2 true                      VM188:9
outside3 false                    VM188:12
```

Doesn’t work if we don’t say “let” within the if-statement:

```
> const player = 'bobby';
  let experience = 100;
  let wizardLevel = false;

  console.log("outside1",wizardLevel);

  if (experience > 90) {
      wizardLevel = true; // scope is root scope
      console.log("inside2",wizardLevel);
  }

  console.log("outside3",wizardLevel);
```

```
outside1 false                    VM42:5
inside2 true                      VM42:9
outside3 true                     VM42:12
```

“Const” will also create new scope…

```
> const player = 'bobby';
  let experience = 100;
  let wizardLevel = false;

  console.log("outside1",wizardLevel);

  if (experience > 90) {
      const wizardLevel = true; // scope is only within if statemt
      console.log("inside2",wizardLevel);
  }

  console.log("outside3",wizardLevel);
```

```
outside1 false                    VM829:5
inside2 true                      VM829:9
outside3 false                    VM829:12
```

If we use var:

```
> const player = 'bobby';
  let experience = 100;
  var wizardLevel = false;

  console.log("outside1",wizardLevel);

  if (experience > 90) {
      var wizardLevel = true; // scope is root scope
      console.log("inside2",wizardLevel);
  }

  console.log("outside3",wizardLevel);
```

```
outside1 false                    VM52:5
inside2 true                      VM52:9
outside3 true                     VM52:12
```

Destructuring (ES5/ES6 only)

Say we have this object..

```
> const obj = {
      player: "bobby",
      experience: 100,
      wizardLevel: false
  }
```

We can do this…

```
  const player = obj.player;
  const experience = obj.experience;
  let wizardLevel = obj.wizardLevel;
```

Or use this notation:

```
  const { player, experience } = obj; // same as above!
  let { wizardLevel } = obj;
```

Note: these new variables do NOT update as the object properties change!

Object Properties: Dynamic Property Values (ES5/ES6 only)

```
> const name = 'john snow';
  const obj2 = {
      [1+2]: 'girl',
      [name]: 'hello',
      ['ray' + 'smith']: 'hihi',
  }
```

```
> name
<- "john snow"
> obj2
<- ▶ {3: "girl", john snow: "hello", raysmith: "hihi"}
```

Object Properties: Property matches Value (ES5/ES6 only)

```
const a = "simon";
const b = true;
const c = {};
```

When property is same as value…

```
const obj3 = {
    a: a, // property matches value
    b: b,
    c: c
}
```

```
> obj3
◂ ▶ {a: "simon", b: true, c: {…}}
```

Or use this notation for same result:

```
> const obj4 = {
    a, b, c
}
```

```
> obj4
◂ ▶ {a: "simon", b: true, c: {…}}
```

Template Strings (ES5/ES6 only)

- Skip escape character for quotes
- Denote variables within strings more conveniently
- Dynamic variables allowed

Normally, must be careful using quotes within a string…
denote that they are okay using the JavaScript escape character, backwards backslash \

```
const name = "Sally";
const age = 34;
const pet = "horse";

const greeting = "Hello " + name + " you\'re pet is a " + pet + "!";
```

```
> greeting
<- "Hello Sally you're pet is a horse!"
```

Use back ticks ``` (above tab key) so that we can use double and single quotes within a string while skipping the escape character, and denote our variables more conveniently (can be dynamic!)…

```
> const name = "Sally";
  const age = 34;
  const pet = "horse";

  const greetingBest = `Hello ${name}, aren't you ${age-10}? You have
  a ${pet}`;
```

```
> greetingBest
<- "Hello Sally, aren't you 24? You have a horse"
```

Default Arguments for Function (ES5/ES6 only)

Create default arguments for function in case no parameters are passed to funxn:

```
> function greet (name='', age=30, pet='cat'){
      return `Hello ${name}, aged ${age-10}? You have a ${pet}`;
  }
```

Default parameters…

```
> greet()
<· "Hello , aged 20? You have a cat"
```

Pass parameters to funxn…

```
> greet("Tracy", 34, "monkey")
<· "Hello Tracy, aged 24? You have a monkey"
```

Advanced JavaScript Types: *Symbol* (ES5/ES6 only)

Symbols create a completely unique type, so that you can make sure there's never going to be any conflict.

```
> let sym1 = Symbol();
  let sym2 = Symbol('foo');
  let sym3 = Symbol('foo');
```

```
> sym2
< Symbol(foo)
> sym3
< Symbol(foo)
> sym2===sym3
< false
```

The symbol value is used as an indentifier mostly for object properties… because sometimes you don't want object properties (if you have thousands of them) to collide and be the same ones because then they'll get bugs.

Arrow Functions (ES5/ES6 only)

```
(param1, param2, …, paramN) => { statements }
(param1, param2, …, paramN) => expression
// equivalent to: => { return expression; }

// Parentheses are optional when there's only one parameter name:
(singleParam) => { statements }
singleParam => { statements }

// The parameter list for a function with no parameters should be written with a pair of
parentheses.
() => { statements }
```

Old way to define a function (statements)…

```
> function add (a,b) {
      return a + b;
  }
```

New way…

```
  const add2 = (a,b) => a + b;
```

Same result:

```
> add(4,3)
< 7
> add2(4,3)
< 7
```

Old way to define function (expressions)…

```
function whereAmI(username, location) {
    if (username && location) {
        return "I am not lost";
    } else {
        return "I am totally lost!";
    }
}
```

New way… (use { })

```
> const whereAmI2 = (username, location) => {
    if (username && location) {
        return "I am not lost";
    } else {
        return "I am totally lost!";
    };
}
```

Same results:

```
> whereAmI("stephanie","usa")
<· "I am not lost"
> whereAmI2("stephanie","usa")
<· "I am not lost"
> whereAmI("stephanie")
<· "I am totally lost!"
> whereAmI2("stephanie")
<· "I am totally lost!"
```

Exercise: ES5 and ES6

Section 13, Lecture 138

It's time to code some javascript! Get your sublime text ready for this exercise, and use Google Chrome javascript console to test your code. You can find the exercise file and the solution file attached. Good luck!

Also, after you have your solutions, try to run them from Babel right here, and see how it gets converted to a format (older javascript) that works on all browsers.

Resources for this lecture

exercise3.js
exercise3-SOLUTIONS.js

Change everything below to newer JavaScript and run in BABEL.

Question 1

```
> // Q1
  // let + const
  var a = 'test';
  var b = true;
  var c = 789;
  a = 'test2';
```

```
  // soln:
  let a = 'test';
  const b = true;
  const c = 789;
  a = 'test2';
```

BABEL:

```
1
2 let a = 'test';
3 const b = true;
4 const c = 789;
5 a = 'test2';
```

```
1 "use strict";
2
3 var a = 'test';
4 var b = true;
5 var c = 789;
6 a = 'test2';
```

Question 2

```
> // Q2
  // Destructuring
  var person = {
      firstName : "John",
      lastName  : "Doe",
      age       : 50,
      eyeColor  : "blue"
  };

  var firstName = person.firstName;
  var lastName = person.lastName;
  var age = person.age;
  var eyeColor = person.eyeColor;
```

```
> // soln:
  const person = {
      firstName : "John",
      lastName  : "Doe",
      age       : 50,
      eyeColor  : "blue"
  };

  const {firstName, lastName, age, eyeColor} = person;
```

```
> firstName
<- "John"
> lastName
<- "Doe"
> age
<- 50
> eyeColor
<- "blue"
```

BABEL:

```
1 const person = {
2     firstName : "John",
3     lastName  : "Doe",
4     age       : 50,
5     eyeColor  : "blue"
6 };
7
8 const {firstName, lastName, age,
  eyeColor} = person;
```

```
1  "use strict";
2
3  var person = {
4    firstName: "John",
5    lastName: "Doe",
6    age: 50,
7    eyeColor: "blue"
8  };
9  var firstName = person.firstName,
10     lastName = person.lastName,
11     age = person.age,
12     eyeColor = person.eyeColor;
```

```
> // Q3
  // Object properties
  var a = 'test';
  var b = true;
  var c = 789;

  var okObj = {
    a: a,
    b: b,
    c: c
  };
<· undefined
> // soln:
  var bestObj = { a, b, c };
<· undefined
> bestObj
<· ▶{a: "test", b: true, c: 789}
```

```
> // Q4
  // Template strings
  var firstName = "Sandy";
  var city = "Denver";
  var message = "Hello " + firstName + " have I met you before? I
  think we met in " + city + " last summer???";
<· undefined
> // soln:
  var message2 = `Hello ${firstName} have I met you before? I think we
  met in ${city} last summer???`;
<· undefined
> message
<· "Hello Sandy have I met you before? I think we met in Denver last su
  mmer???"
> message2
<· "Hello Sandy have I met you before? I think we met in Denver last su
  mmer???"
```

BABEL:

```
1
2
3 var message2 = `Hello ${firstName}
  have I met you before? I think we
  met in ${city} last summer???`;
4
```

```
"use strict";

var message2 = "Hello
".concat(firstName, " have I met
you before? I think we met in
").concat(city, " last
summer???");
```

```
> // Q5
  // default arguments
  // default age to 10;
  function isValidAge(age) {
      return age
  }
<· undefined
```

```
> // soln:
  function isValidAge2(age=10) {
      return age
  }
<· undefined
> // better soln:
  const isValidAge3 = (age = 10) => age;
<· undefined
```

```
> isValidAge()
<· undefined
> isValidAge2()
<· 10
> isValidAge3()
<· 10
> isValidAge(20)
<· 20
> isValidAge2(20)
<· 20
> isValidAge3(20)
<· 20
```

BABEL:

```
1 // better soln:
2
3 const isValidAge3 = (age = 10) =>
  age;
4
5
```

```
1 "use strict";
2
3 // better soln:
4 var isValidAge3 = function
  isValidAge3() {
5   var age = arguments.length > 0
  && arguments[0] !== undefined ?
  arguments[0] : 10;
6   return age;
7 };
```

```
> // Q6
  // Symbol
  // Create a symbol: "This is my first Symbol"

  // soln:
  const sym1 = Symbol("my first symbol");
```

```
> // Q7
  // Arrow functions
  function whereAmI(username, location) {
      if (username && location) {
          return "I am not lost";
      } else {
          return "I am totally lost!";
      }
  }
```

```
> // soln:
  const whereAmI2 = (username, location) => {
      if (username && location) {
          return "I am not lost";
      } else {
          return "I am totally lost!";
      };
  }
```

```
> whereAmI("stephanie","usa")
<- "I am not lost"
> whereAmI2("stephanie","usa")
<- "I am not lost"
> whereAmI("stephanie")
<- "I am totally lost!"
> whereAmI2("stephanie")
<- "I am totally lost!"
```

BABEL:

```
1 // soln:
2 const whereAmI2 = (username,
  location) => {
3     if (username && location) {
4         return "I am not lost";
5     } else {
6         return "I am totally
  lost!";
7     };
8 }
9
```

```
"use strict";
2
3 // soln:
4 var whereAmI2 = function
  whereAmI2(username, location) {
5   if (username && location) {
6     return "I am not lost";
7   } else {
8     return "I am totally lost!";
9   }
10
11  ;
12 };
```