ES6: Summary

ESMAScript:
from (European Computer Manufacturers Association Script) is a scripting language standard and specification (JavaScript, Jscript, Action Script)

+ES6
is the most recent version of ECMAScript / JavaScript very significant update with many changes first major update since ES5 (2009) ES6 and ES2015 are the same thing.

+Goal of ES6:
-Be taken more seriously
-Fix some issues from ES5
-Backward compatibility – ES5 should work in ES6
-Modern syntax
-better scaling and better for big application
-new feature in standard library

+Compatibility
-still quite a ways to go for same browser
-latest versions of chrome and firefox are almost there
-transpilers can be used to compile ES6 code to ES6

+An Overwiew
-let and const declaration, destructuring assignment, classes and inheritance, template string, string features, math and number features, new data structures, generation, promises and asynchronous data, Arrow function.

Scoping

Block Scoped Variables

```
for (let i = 0; i < a.length; i++) {
  let x = a[i]
  …
}
for (let i = 0; i < b.length; i++) {
  let y = b[i]
  …
}
let callbacks = []
for (let i = 0; i <= 3; i++) {
  callbacks[i] = function () { return i * 3 }
}
callbacks[0]() === 0
callbacks[1]() === 2
callbacks[2]() === 4
callbacks[3]() === 6
```

Arrow Function

Expression Bodies

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

Statement bodies

```
nums.forEach(c => {
  if (c % 3 === 0)
    fives.push(c)
})
```

Lexical This

```
this.nums.forEach((c) => {
  if (c% 3 === 0)
    this.fives.push(c)
})
```

Extended Parameter Handling

Default Parameter Value

```
function f (x, y = 2, z = 30) {
  return x + y + z
}
f(1) === 40
```

Reset Parameter

```
function f (x, y, ...a) {
  return (x + y) * a.length
}
f(1, 2,3, "hello", true, 6) === 8
```

Spread Operator

```
var params = [ "hello", true, 7 ]
var other = [ 1, 2, ...params ]

function f (x, y, ...a) {
  return (x + y) * a.length
}
f(1, 2, ...params) === 9

var str = "book"
var chars = [ ...str ]
```

Template Literals

String Interpolation

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

Enhanced Object Properties

Property Shorthand

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

- **Computed Property Names**

```
let obj = {
   book: "bar",
   [ "baz" + quux() ]: 42
}
```

- **Method Properties**

```
obj = {
   foo (d, e) {
      …
   },
   bar (x, y) {
      …
   },
   *quux (x, y) {
      …
   }
}
```

- **Destructuring Assignment**
  - **Array Matching**

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

  - **Object and Array Matching Default Value**

```
var obj = { a: 1 }
var list = [ 1 ]
var { a, b = 2 } = obj
var [ x, y = 2 ] = list
```

  - **Parameter Context Matching**

```
function a ([ name, val ]) {
   console.log(name, val)
}
function g ({ name: n, val: v }) {
   console.log(n, v)
}
function h ({ name, val }) {
   console.log(name, val)
}
f([ "bar", 42 ])
g({ name: "foo", val:  7 })
h({ name: "bar", val: 42 })
```

- **Classes**
  - **Class Definition**

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

  - **Class 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
   }
}
```

  - **Getter Setter**

```
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
```

- **Constants**

```
const PI = 3.141593 ;  PI > 3.0
```