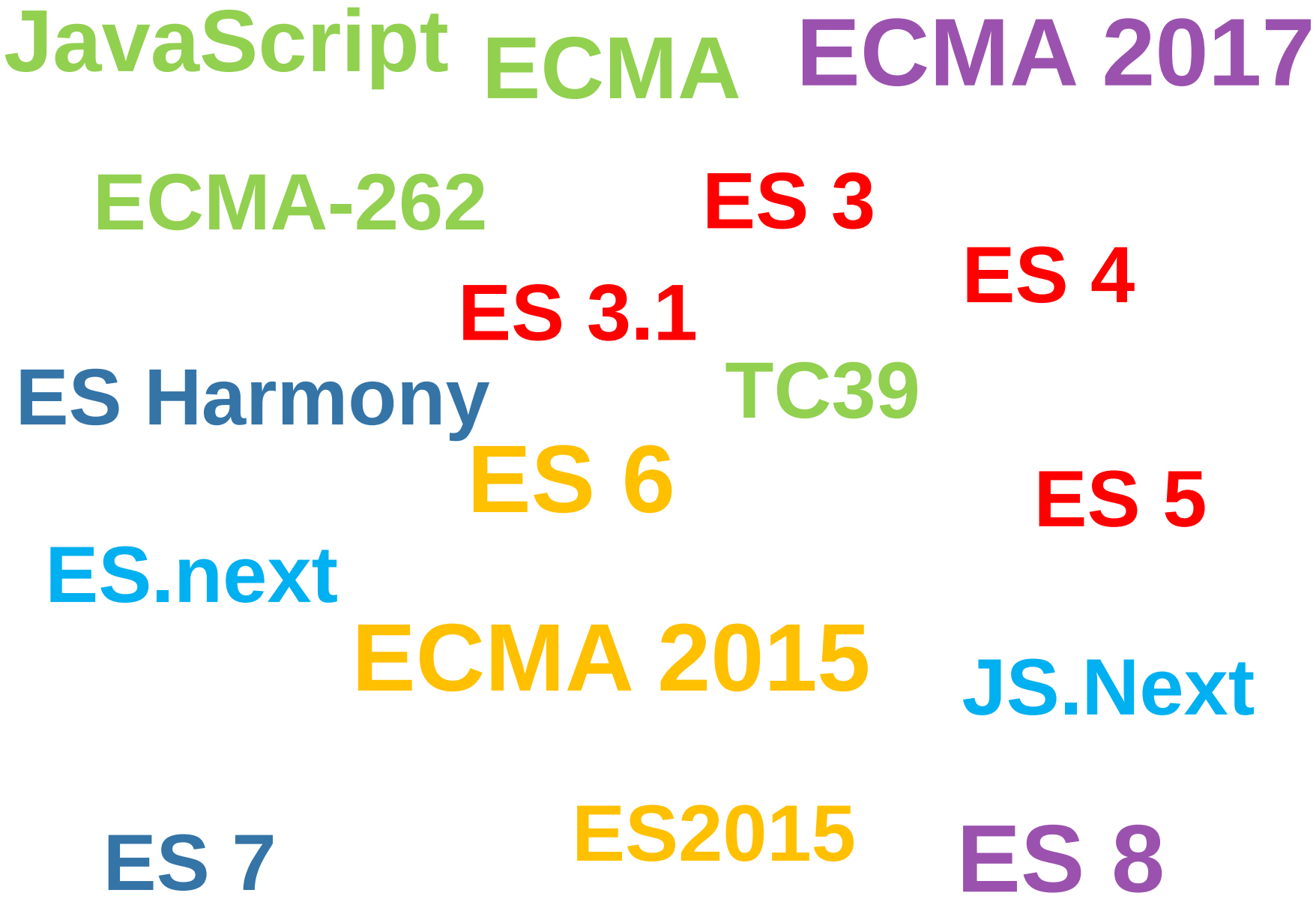
ES.next

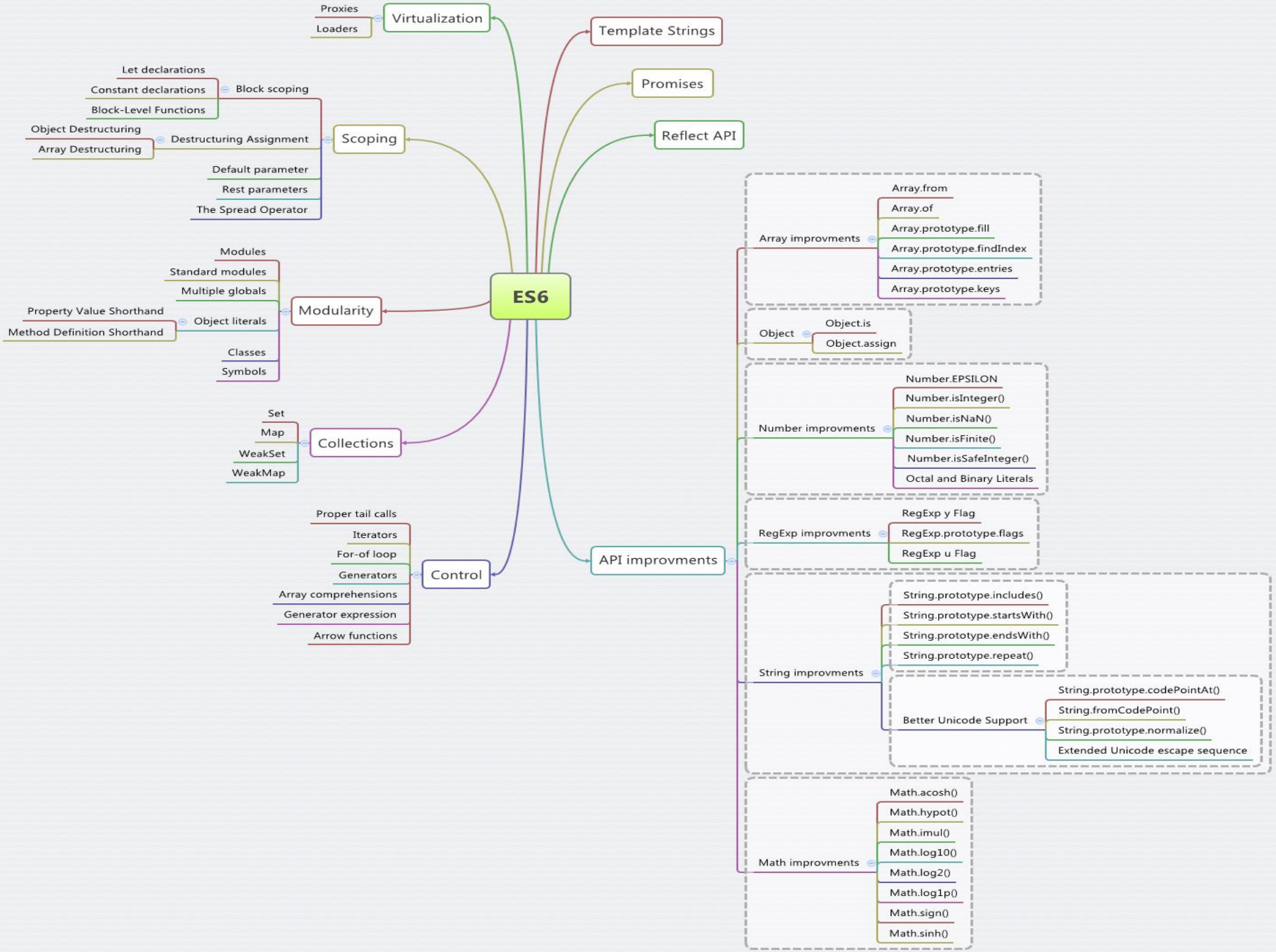
Amazing New Features In JavaScript

Eng. Niveen Nasr El-Den
SD & Gaming CoE
iTi

Day 1

Amazing New Features In JavaScript


JavaScript
ECMA
ECMA 2017
ECMA-262
ES 3
ES 4
ES 3.1
ES Harmony
TC39
ES 6
ES 5
ES.next
ECMA 2015
JS.Next
ES2015
ES 7
ES 8


ES6
Virtualization
Proxies
Loaders
Template Strings
Promises
Reflect API
Scoping
Block scoping
Let declarations
Constant declarations
Block-Level Functions
Destructuring Assignment
Object Destructuring
Array Destructuring
Default parameter
Rest parameters
The Spread Operator
Modularity
Modules
Standard modules
Multiple globals
Object literals
Property Value Shorthand
Method Definition Shorthand
Classes
Symbols
Collections
Set
Map
WeakSet
WeakMap
Control
Proper tail calls
Iterators
For-of loop
Generators
Array comprehensions
Generator expression
Arrow functions
API improvments
Array improvments
Array.from
Array.of
Array.prototype.fill
Array.prototype.findIndex
Array.prototype.entries
Array.prototype.keys
Object
Object.is
Object.assign
Number improvments
Number.EPSILON
Number.isInteger()
Number.isNaN()
Number.isFinite()
Number.isSafeInteger()
Octal and Binary Literals
RegExp improvments
RegExp y Flag
RegExp.prototype.flags
RegExp u Flag
String improvments
String.prototype.includes()
String.prototype.startsWith()
String.prototype.endsWith()
String.prototype.repeat()
Better Unicode Support
String.prototype.codePointAt()
String.fromCodePoint()
String.prototype.normalize()
Extended Unicode escape sequence
Math improvments
Math.acosh()
Math.hypot()
Math.imul()
Math.log10()
Math.log2()
Math.log1p()
Math.sign()
Math.sinh()

ES6 Features

- let + const
- default Parameters
- rest parameters
- spread operator
- Destructuring (array/object)
- Arrow Functions
- Enhanced object literals
- Template strings
- for..of
- Data Structure/Collection
  - map
  - set
  - weakmap
  - weakset
- Binary and Octal literals
- Classes
- iterators
- Generators
- Symbols
- Proxies
- Modules
- Module loaders
- Promises
- math API
- number API
- string API
- array API
- object API
- etc…

http://es6-features.org/#Constants

https://github.com/lukehoban/es6features

https://kangax.github.io/compat-table/es6/

ES6 Keywords

- break
- case
- class
- catch
- const
- continue
- constructor
- debugger
- default
- delete
- do
- else
- export
- extends
- finally
- for
- function
- get
- if
- import
- in
- instanceof
- let
- new
- null
- return
- set
- super
- static
- switch
- this
- throw
- try
- typeof
- var
- void
- while
- with
- yield
- **etc…**

let & const

- ES6 represents block scope via let, const.
  - Block starts by { and ends by }
- Variables declared with let and const don’t hoist.
- Variables defined by let can be reassigned
- Variable defined by const cant be reassigned

let & const Example

```
function closureTest() {
    var arr = [];
    for (var i = 0; i < 3; i++) {
        arr.push(function () {
            console.log(i);
        });
    }
    return arr;
}
```

```
const pi=3.14;
```

```
function closureTest() {
    var arr = [];
    for (let i = 0; i < 3; i++) {
        arr.push(function () {
            console.log(i);
        });
    }
    return arr;
}
```

Closure & its Conflicts

- Reminder: Closure is a local variable for function that kept in memo even after function is returned
- Closure’s conflicts happen when using for loop
- We used to solve closure’s conflict via IIFE
- Nowadays we can use let to overcome closure problem with loops

Arrow Function

- Arrow functions bind to the scope of where they are defined, not where they are called.
  - AKA lexical binding
  - Unlike functions, arrows share the same lexical this as their surrounding code.
  - It finds this from its enclosing scope.

```
var closureConstructor = {
    m: "hello",
    disp: function () {
        var that = this;
        setTimeout(function () {
            console.log(that.m)
        }, 1000)
    }
}
```

```
var closureConstructor = {
        m: "hello",
        disp: function () {
            setTimeout(() => {
                console.log(this.m)
            }, 1000)
        }
    }
```

Arrow Function

- Arrow functions bind to the scope of where they are defined, not where they are called.
  - AKA lexical binding
  - Unlike functions, arrows share the same lexical this as their surrounding code
  - It finds this from its enclosing scope.

```
var closureConstructor = {
    m: "hello",
    disp: function () {
        var that = this;
        setTimeout(function () {
            console.log(that.m)
        }, 1000)
    }
}
```

```
var closureConstructor = {
        m: "hello",
        disp () { //Method initialization shorthand
            setTimeout(() => {
                console.log(this.m)
            }, 1000)
        }
    }
```

…rest Parameter

- Used in function definition and must be the last parameter in function definition
- It gathers all remaining arguments into an array element.

```
function displayNames(p1, ...arr) {
    for (let i in arr) {
        console.log(arr[i])
    }

}

var fruits = ["apple", "strawberry",
      "banana", "orange", "mango"];

displayNames(10, fruits)
```

...spread Operator

- Used in function invocation

```
function dis(f, n) {
    console.log(f + "" + n)
}
var rest = ["ahmed", "ali"]
dis(...rest)
```

- Used also in Array

```
var fruits = ["apple", "strawberry", "banana", "orange", "mango"];

var newFruits = ["kiwi", ...fruits]
// ["kiwi", "apple", "strawberry", "banana", "orange", "mango"]

var anotherNewFruits = ["kiwi", ...fruits, "others"]
// ["kiwi", "apple", "strawberry", "banana", "orange", "mango", "others"]
```

Destructuring Assignment

- Array Destructuring

```
let users = ["ali", "nour", "kareem"];
let [a, b, c] = users;
console.log( a, b, c );
```

- Object Destructuring

```
var User = function (id, firstName,
lastName) {
    return {
        id: id,
        firstName: firstName,
        lastName: lastName
    };

}
```

```
var me = new User(10,”Ahmed”,”Ali”)
console.log(me.firstName)
```

```
var {id,firstName} = new User(10,”Ahmed”,”Ali”)

console.log(id)
console.log(firstName)
```

Order doesn’t matter

Must be with the same name of the object properties

Object shorthand literal creation

```
var User = function (id, firstName, lastName) {
    return {
        id: id,
        firstName: firstName,
        lastName: lastName
    };

}
```

```
var User = function (id, firstName, lastName) {
    return {
        id,
        firstName,
        lastName
    };

}
```

String API Improvements

Method name
endsWith()
startsWith()
includes()
repeat()
search()
trim()
trimRight()
trimLeft()

```
var str = “ES6 ”;

str.repeat(5);
// “ES6 ES6 ES6 ES6 ES6”;
```

```
var str = “Hi ES6”;

str.includes(“ES6”);//true
```

Template Strings

- Template strings are string literals allowing embedded expressions using `` and ${}

```
function User(first, last,){

let fullName = `${first} ${last}`;
console.log(fullName);

}
```

Array API Improvements

- These methods are a JavaScript extension to the ECMA-262 standard.

Name	Example
every(testingfn)	Tests every element similar to &&; It returns true if all elements meet the testing function
some(testingfn)	Like every() but it applies the manner of ||; It returns true if one element meets the testing function
map(callback)	Creates a new array with the results of executing the callback function on array elements
filter(testingfn)	Creates a new array with the arrays elements that pass the testing function
find(callback)	Search for an element inside calling array object
forEach(callback)	Calls a callback function on each element in the array, similar to for loop

Example

```
[1, 2, 3, 4].find(num => num === 2);

let fruits = ["apple", "strawberry", "banana",
          "orange", "mango"];

fruits.forEach(val => console.log(val));

fruits.map((val) => {
    return "i like " + val
});
```

Assignments