WHAT'S NEW IN JAVASCRIPT

(ES 2015/2016/2017/2018)

RAJU GANDHI

OBJECT LITERALS

```
(function () {
  console.log("Demonstrates shorthand property creation");
  let counter = 0,
    todos = [
      { id: ++counter, text: "Learn ES6" },
      { id: ++counter, text: "Practice! Practice!" },
      { id: ++counter, text: "Teach it" }
    ];

  // similar to { todos: todos }
  const ret = { todos };
  console.log(JSON.stringify(ret));
  return ret;
})();
```

```
(function () {
  console.log("Demonstrates shorthand accessors");
  const obj = {
    _name: 'Hulk',
    // getter/setters
    get name() {
      return this._name;
    },
    set name(name) {
      this._name = name
    }
  };

  console.log(JSON.stringify(obj));
  console.log(obj.name); // invokes getter
})();
```

```
(function () {
  console.log("Demonstrates computed property/method");
  const providePrefix = (name) => {
    return `my${name.substring(0, 1).toUpperCase()}${name.substring(1, name.length)}`;
  }

  const obj = {
    _name: 'Hulk',
    // computed property name
    [`fullName`]: 'Hulk-a-mania',
    // computed method name
    [providePrefix('name')]() {
      return this._name;
    }
  };

  console.log(JSON.stringify(obj));
  console.log(obj.fullName);
  console.log(obj.myName());
})();
```

SYMBOL

```
(function () {
  console.log('Demonstrates construction and immutability');
  // the string 'foo' is descriptional
  const fooSym = Symbol('foo');
  console.log(fooSym.toString());
  // is universal, and immutable
  console.log(Symbol('foo') === Symbol('foo'));

  // is a datatype
  console.log(typeof fooSym);
})();
```

```
(function () {
  // no constructor!!!
  try {
    new Symbol()
  } catch (e) {
    // TypeError: Symbol is not a constructor
    console.log(e);
  }

  // global registry
  // Symbol.for will look up a symbol with a key,
  // and create one if it does not exist
  const fooSym = Symbol.for('foo');
  console.log(fooSym === Symbol.for('foo'));

  console.log(Symbol.keyFor(fooSym));
})();
```

```
(function () {
  class HyphenSplitter {
    [Symbol.split](txt, n) {
      const result = txt.split('-')
        .map(s => `(${s})`);

      if (result.length > n) {
        return result.splice(0, n);
      }
      return result;
    }
  }

  const ssn = '123-45-6789';
  console.log(JSON.stringify(
    ssn.split(new HyphenSplitter(), 2)));
})();
```

ITERATORS

```
// creates an iterator over arrays
// returns an array of [key, value] by default
clear();
let arr = [10, 20, 30];
let iter = Iterator(arr);

console.log(iter.next()); // [0,10]
console.log(iter.next()); // [1,20]
console.log(iter.next()); // [2,30]
```

```
// the for .. of loop automatically invokes 'next'
clear();
let arr = [10, 20, 30];

for (let i of arr) {
  console.log(i);
}
```

```
// Maps
for (let [k, v] of new Map([["a", 1], ["b", 2]])) {
  console.log(k, "->", v);
}

// Sets
for (let k of new Set(["a", "b", "a"])) {
  console.log(k);
}

// Strings
for (let c of "ES6 Rocks!!") {
  console.log(c);
}
```

```
// you can define custom iterators for your own
// objects by tacking a iterator property on it
// The iterator should be a function that
// returns an object that responds to the 'next' call
var iterableObj = {
  upperLimit: 10,
  [Symbol.iterator]: function () {
    // iterator is a function
    var that = this;
    // that returns an object
    return {
      cur: 0,
      // that has the next function defined
      next: function () {
        if (this.cur < that.upperLimit) {
          let ret = this.cur;
          this.cur++;
          return { value: ret, done: false };
        }
        this.cur = 0;
        return { done: true };
      }
    }
  }
}
for (let i of iterableObj) console.log(i); // 0,1,2,…,9
```

GENERATORS

```
function* someGenerator() {
  yield 1;
  yield 2;
  yield 3;
}
```

```
let gen = someGenerator();
console.log(gen.next().value); //1
console.log(gen.next().value); //2
console.log(gen.next().value); //3
console.log(JSON.stringify(gen.next())); //{"done":true}
```

WHAT'S THE POINT?

YOU

```
{
  function* fibonacci() {
    let start = 0, next = 1;
    yield start;
    yield next;
    while (true) {
      let result = start + next;
      start = next, next = result;
      yield result;
    }
  }

  let f = fibonacci();
  console.log('--- FIBONACCI ---------------------');
  console.log(f.next().value); // 0
  console.log(f.next().value); // 1
  console.log(f.next().value); // 1
  console.log(f.next().value); // 2
  console.log(f.next().value); // 3
  console.log(f.next().value); // 5
  console.log(f.next().value); // 8
}
```

GENERATORS

- USE THE FUNCTION* SYNTAX
- ACTIVATE THE “YIELD" AND “YIELD*”

KEYWORDS

PROMISE

```
{
  clear();
  console.log("Demonstrates resolution");
  const p = new Promise(function (res, rej) {
    setTimeout(res, 1000, 'woohoo!');
  });

  p.then(function onSuccess(msg) {
    console.log('Success', msg);
  }, function onError(err) {
    console.err(err);
  });
}
```

```
{
  clear();
  console.log("Demonstrates rejection");
  const p = new Promise(function (res, rej) {
    setTimeout(rej, 1000, 'No way hose!');
  });

  p.then(function onSuccess(msg) {
    console.log('Success', msg);
  }, function onError(err) {
    console.error('Error', err);
  });
}
```

```
{
  clear();
  console.log("Demonstrates all");
  const p1 = new Promise(function (res, rej) {
    setTimeout(res, 1000, 'resolved last');
  }),
    p2 = Promise.resolve('Success'),
    p3 = Promise.resolve(true);

  Promise.all([p1, p2, p3])
    .then(function onSuccess(msg) {
      console.log('Success', msg);
      // you get back an array
      console.log(msg instanceof Array);
    }, function onError(err) {
      console.error('Error', err);
    });
}
```

```
{
  clear();
  console.log("Demonstrates race");
  const p1 = new Promise(function (res, rej) {
    setTimeout(res, 1000, 'resolved last');
  }),
    p2 = new Promise(function (res, rej) {
      setTimeout(res, 500, 'resolved first');
    });

  Promise.race([p1, p2])
    .then(function onSuccess(msg) {
      // you get back a value
      console.log('Success', msg);
      console.log(typeof msg);
    }, function onError(err) {
      console.error('Error', err);
    });
}
```

CLASSES

```
class Todo {
  constructor(text) {
    this.text = text;
  }

  toString() {
    return `You should ${this.text}`;
  }

  get isDone() {
    this.done;
  }
}

(function () {
  let t = new Todo("Buy Milk");
})();
```

```
class MyList extends Array {
  constructor(...items) {
    super(...items)
  }
  // other functions here
}

(function () {
  let x = new MyList(3, 4, 5);
})();
```

CLASSES

- ACTIVATE "CLASS" & "EXTENDS" KEYWORDS
- "CONSTRUCTOR" IS A SPECIAL METHOD
- SUGARS PROTOTYPAL INHERITANCE
- SEVERAL TYPES (DATE, ARRAY ETC) WITHIN JAVASCRIPT ARE NOW SUBCLASSABLE

MODULES

EXPORTING

```
// string-utils.js
export function encrypt(msg) {
  return msg.split('').reverse().join('');
}
```

```
export const MY_CONSTANT = "Some Constant";
var someOtherVar = "Is not exported";
```

```
// string-utils.js
function encrypt(msg) {
  return msg.split('').reverse().join('');
}
```

```
const MY_CONSTANT = "Some Constant";
var someOtherVar = "Is not exported";

export { encrypt, MY_CONSTANT };
```

```
export { encrypt, MY_CONSTANT };
// OR rename
export { encrypt as doNotTrackMe, MY_CONSTANT as SOME_CONSTANT };
```

IMPORTING

```
// if there are a lot of exports from a file you can
// import them all together as

// everything gets tacked on global ns
import * as str from 'string_utils.js';

// to avoid that you can do
import 'string_utils.js' as str;

// or import only certain "exports"
import { doNotTrackMe } from 'string_utils.js';

// rename the import
import { doNotTrackMe as myEncrypt } from 'string_utils.js';
```

TEMPLATE STRINGS

```
let name = "Raju";
let greeting = `Hola! My name is ${name}`;

// can be any expression
let loudGreetings = `HOLA! MY NAME IS ${name.toUpperCase()}`;
```

```
// multi-line strings
let longStr = `This is a
doc string can be a multi-line
string
`
```

```
let firstName = "raju"
let lastName = "gandhi"

function handler(str, ...subs) {
  // Gets the "raw" string as an array that looks like
  // ["Hello", " ", "!"]

  // This array has a "raw" property that gives you the
  // String as is (e.g for `Hello \n` the "raw" will be
  // ["Hello \\n"], whereas str will be ["Hello \n"]
  // You can get to it by "str.raw"

  // The substitutions are passed in as the remaining args
  console.log(subs[0]); // "raju"
  console.log(subs[1]); // "gandhi"
}

// This is NOT a regular function call!
handler`Hello ${firstName} ${lastName}!`
```

```
// Can be used for L10N
myButton.innerText = msg`Open`;
```

```
// Dynamic regex generation
re`\d+(${localeSpecificDecimalPoint}\d+)?`
```

TEMPLATE STRINGS

- NEW SYNTAX (BACKTICK)
- CAN HAVE A "HANDLER" FUNCTION (SANITIZATION, LOCALIZATION, DYNAMIC TEMPLATES)
- ARE EVALUATED IMMEDIATELY

MUCH MORE ...

PROPER TAILS CALLS
NEW OBJECT, MATH, STRING, NUMBER APIS

...

ES7

EXPONENTIAL

```
console.log(2 ** 3);
```

```
var a = 2,
    a **= 3;
```

ES8

ASYNC/AWAIT

```
const a = () => {
  return new Promise((res, rej) => setTimeout(res, 1000, 'a'));
};
const b = (arg) => {
  console.log('in b', arg);
  return new Promise((res, rej) => setTimeout(res, 1000, 'b'));
};
const c = (arg) => {
  console.log('in c', arg);
  return Promise.resolve('c');
};
```

```
// UTILIZING PROMISES
a().then(b).then(c).then(console.log);

// UTILIZING ASYNC/AWA
(async () => {
  const a1 = await a();
  const b1 = await b(a1);
  const c1 = await c(b1);
  console.log(c1);
})();
```

SHARED MEMORY/ ATOMICS

ESG*

OBJECT SPREAD

```
// we have an object
const raju = {
  name: 'raju',
  address: {
    street: '1 inifinity drive',
    state: 'OH',
    zip: [
      12345,
      6789
    ]
  }
}
// use spread operator to conjoin one on to another
const resume = {
  profession: 'developer',
  interests: 'devops',
  ...raju,
};

console.log('Conjoined object', resume);
```

OBJECT REST

```
const resume = {
  profession: 'developer',
  interests: 'devops',
  name: 'raju',
  address: {
    street: '1 inifinity drive',
    state: 'OH',
    zip: [12345, 6789]
  }
};

const {
  name,
  ...rest
} = resume;

console.log('Name is', name);
console.log('Rest is', rest);
```

REGEX

```
// named groups
const RE_ZIP = /(?<major>[0-9]{5})-(?<minor>[0-9]{4})/;
const matchObj = RE_ZIP.exec('45701-1234');

const {groups: {major, minor}} = matchObj;

console.log(major, minor);
```

```
// positive look ahead
const RE_POS_LA = /(?=\d{10})(\d{3})/;
const areaCode = RE_POS_LA.exec('6141234567');
console.log(areaCode[1]); // 614
```

```
// positive look behind
const RE_URL = /(?<=http:\/\/)[\w/\.]+(?=\?)/;
const urlMatch = RE_URL.exec('http://www.google.com/search?hl=en');
console.log(urlMatch[0]); // www.google.com/search
```

```
// negative look ahead
const RE_DIR = /home\/(?!photos)(.*)/;
const dirMatches = RE_DIR.exec('home/documents/vacation.jpg');
console.log(dirMatches[1]); // documents/vacation.jpg

const dirDoesNotMatch = RE_DIR.exec('home/photos/vacation.jpg');
console.log(dirDoesNotMatch); // null
```

```
// negative look behind
const RE_LOGIN = /(?<!http:\/\/)[\w:]+(?=@)/
const loginMatch = RE_LOGIN.exec('ftp://username:password@hostname/');

console.log(loginMatch[0]); // username:password
```

ASYNC

GENERATORS/ITERATORS

```
async function* getPosts(...args) {
  for (let i of args) {
    const res = await fetch(`https://jsonplaceholder.typicode.com/posts/${i}`);
    const json = await res.text();
    yield json;
  }
}

async function asyncIterator() {
  for await (let i of getPosts(1, 2, 3, 4, 5)) {
    console.log(i);
  }
}
```

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