Ecma 6 Cheat Sheet

Arrow functions

Lexical this, shoter functions.

```
let obj = {
  method: function () {
    return () => this;
  }
};
//Due to lexical scope obj.method()(

let fact = (n) => { return n === 0 ?

let fib = (n) => { return n < 2 ? n
```

Block Scope

Declares a block scope local variable, optionally initializing it to a value.

```
var aboutme = () => {
  {
    var investements = 1;
    const salary = 10;
  }
  console.log(investements,salary);
}
```

class

Syntactical sugar over prototype-based inheritance.

```
class Person {
  constructor(name) {
    this.name = name;
    this.movement = "walks";
  }

  move(meters) {
    console.log(`${this.name} ${this
  }
}

class Hero extends Person {
  constructor(name, movement) {
    this.name = name;
    this.movement = movement;
  }

  move() {
    super.move(500);
  }
}

let clark = new Person("Clark Kent")

let superman = new Hero("Superman",

clark.move(100);
//-> Clark Kent walks 100m.

superman.move();
//-> Superman flies 500m.

/* Make a note of:

class Base {}
class Derived extends Base {}

//Here, Derived.prototype will inher

let parent = {};
class Derived prototype parent {}
*/
```

Computed property names

An expression in brackets `[]`

```
var obj = {
  [foo + bar]: "o_0",
  [foo + baz]: "0_o",
  foo: "foo",
  bar: "bar",
  baz: "baz"
};

console.log(obj.foobar); //o_0
console.log(obj.foobaz); //0_o
```

Default Params

Initialize formal parameters with default values, if no value or undefined is passed.

```
let greet = (msg = "Hello", name = "
```

Destructuring

Extract data from arrays or objects.

```
var {foo, bar} = {foo: "lorem", bar:
//foo => lorem and bar => ipsum
```

Direct Proxy

Define custom behavior for fundamental operations of an object.

```
let NegativeIndices = (array) => {
  return new Proxy(array, {
    get: (receiver, name) => {
      let index;
      console.log('Proxy#get', array
      index = parseInt(name);
      if (!isNaN(index) && index < 0
        array[array.length + index];
      } else {
        return array[name];
      }
    }
  });
```

for-of loop

Loop over Iterator objects.

```
for (let element of [1, 2, 3]) {
  console.log(element);
}
```

Generators

The function* declaration defines a generator function, which returns a Generator object.

```
function *Counter(){
  var n = 0;
  while(1 < 2) {
      yield n;
      ++n;
  }
}

var CountIter = new Counter();

CountIter.next(); //{value: 0, done:
CountIter.next(); //{value: 1, done:
CountIter.next(); //{value: 2, done:
```

```
};

/*
* Negative array indices:
* array = NegativeIndices [4, 420, 4
* array[-1] is 42
*/
```

Map

Map object is a simple and effective key/value data-structure.

```
let m = new Map();
m.set('answer', 42);
m.get('answer'); //42
m.has('answer'); //true
m.delete('answer');// true
m.has('answer'); //false

m.set(keyFunc,() => "foo");
m.get(keyFunc)(); //"foo"
```

modules

Module format common to CommonJS and AMD.

```
/* In math.js */
export function div(x, y) {
  return x / y;
}
export var pi = 3.141593;

//In index.js
import {div, pi} from math;
```

Better Object Literal

Better as in the example.

```
var greet = {
  __proto__: theProtoObj,
  handler, //Instead of handler: han
  world: () => "Hello World!",
  toString() {
    return "Results: " + super.toStr
  }
};
```

property-method-assignment

Method syntax is supported in object initializers.

```
let person = {
  get name() {
    return this._name;
  },
  set name(val){
    console.log("Setting name: " + v
    this._name = val;
  }
};

/*
> person.name = "Hemanth"
"Hemanth"
"Setting name: Hemanth"

> person.name
"Hemanth"
```

Rest params

Variable number of arguments without using the arguments object.

```
let sortRestArgs = (...theArgs) => t

console.log(sortRestArgs(5,2,7,1)) /
```

Set

Store unique values of any type.

```
var cards = new Set();
cards.add('♠');
cards.add('♥');
cards.add('♦');
cards.add('♣');

cards.has('♠'); //true

cards.has('joker'); //false

cards.size; //4

cards.forEach((card) => console.log(

/*
Would log:
♠
♥
♦
♣
*/

cards.add('♠');

cards.size //Still four as ♠ was alr
```

Spread operator

Expanded in places with `...` for arguments or multiple elements.

```
var nodeList = document.querySelecto
var array = [...nodeList];
```

Symbol

Unique and immutable data type.

```
var Cat = (function() {
    var nameSymbl = Symbol('name');

    function Cat(name) {
        this[nameSymbl] = name;
    }

    Cat.prototype.getName = function
        return this[nameSymbl];
    };

    return Cat;
}());

var c = new Cat('milly');
console.log("Cat's name: " + c.getNa
delete c.name; //Even after deleting
console.log("Cat's name is still: "
```

Tail recursion

Tail Calls, Optimization.

```
let factorial = (n, acc = 1) => {
    if (n <= 1) return acc;
    return factorial(n - 1, n * acc)
}
//NO S.O!
factorial(133333337);
```

Template Literals

Better string formatting capabilities.

```
var First = "Hemanth";
var Last = " HM";
`${First} + ${Last} = ${First + Last
//"Hemanth + HM = Hemanth HM"
```

Unicode in Regex

Unicode aware regex.

```
var string = 'foo≣bar';
var match = string.match(/foo(.)bar/
console.log(match[1]);
//'≣'
```

WeakMap

key/value pairs, keys are objects and the values can be arbitrary values, references to key objects are held "weakly".

```
var wm = new WeakMap();

wm.set('life'); //TypeError: Invalid

wm.set('life', 'life'.length) //Type

var wmk = {};
```

```
wm.set(wmk, 'life');
wm.get(wmk); //"life"
wm.has(wmk); //true
wm.delete(wmk); //true
wm.has(wmk); //false
```

WeakSet

Store weakly held objects in a collection.

```
var ws = new WeakSet();
var foo = {};
var bar = {};

ws.add(window);
ws.add(foo);

ws.has(window); //true
ws.has(bar); //false, bar has not be

ws.delete(window); //removes window
ws.has(window); //false, window has

ws.clear(); //empty the whole WeakSe
```