Cheatography

ES6 Cheatsheet Cheat Sheet

by camilo.herbert via cheatography.com/66323/cs/16561/

Constants

```
> const EULER = 2.7182818284
> EULER = 13
> EULER
> 2.7182818284 // Value stays the same
```

Warning! If array or object, the reference is kept constant. If the constant is a reference to an object, you can still modify the content, but never change the variable.

```
> const CONSTANTS = []
> CONSTANTS.push(EULER)
> CONSTANTS
> [ 2.7182818284 ]
> CONSTANTS = { ‘euler’: 2.7182818284 }
> CONSTANTS
> [ 2.7182818284 ]
```

Object Notation

```
// Computed properties
> let key = new Date().getTime()
> let obj = { [key]: “value” }
> obj
> { '1459958882881': 'value' }
// Object literals
balloon = { color, size };
// Same as
balloon = {
 color: color,
 size: size
}
// Better method notations
obj = {
 foo (a, b) { … },
 bar
```

Destructuring Arrays

```
> let [a, b, c, d] = [1, 2, 3, 4];
> console.log(a);
> 1
> b
> 2
```

Destructuring Objects

```
> let luke = { occupation: 'jedi',
 father: 'anakin' }
> let {occupation, father} = luke
> console.log(occupation, father)
> jedi anakin
```

Spread Operator

```
// Turn arrays into comma separated
// values and more
> function logger (...args) {
 console.log(‘%s arguments’,
 args.length)
 args.forEach(console.log)
 // arg[0], arg[1], arg[2]
}
```

let vs var

```
> var average = 5
> var average = (average + 1) / 2
> average
> 3
> let value = ‘hello world’
> let value = ‘what is new’
// -> throws TypeError: Identifier 'value' has
already been declared
```

Be aware of Temporal Dead Zones:

```
> console.log(val) // -> 'undefined'
> var val = 3
> console.log(val) // -> 3
```

Because it's equivalent to:

let vs var (cont)

```
> var val
> console.log(val)
> val = 3
> console.log(val)
```

Variables declared with "let/const" do not get hoisted:

```
> console.log(val)
// -> Throws ReferenceError
> let val = 3
> console.log(val) // -> 3
```

Promises

```
new Promise((resolve, reject) => {
        request.get(url, (error, response,
            body) => {
            if (body) {
                resolve(JSON.parse(body));
            } else {
                resolve({});
            }
        })
    }).then(() => { ...
    })
    .catch((err) =>
        throw err)
// Parallelize tasks
Promise.all([
    promise1, promise2, promise3
]).then(() => {
    // all tasks are finished
})
```

Classes, Inheritance, Setters, Getters

```
class Rectangle extends Shape {
    constructor(id, x, y, w, h) {
        super(id, x, y)
        this.width = w
        this.height = h
    }
    // Getter and setter
    set width(w) {
        this._width = w
    }
    get width() {
        return this._width
    }
}
class Circle extends Shape {
    constructor(id, x, y, radius) {
        super(id, x, y)
        this.radius = radius
    }
    do_a(x) {
        let a = 12;
        super.do_a(x + a);
    }
    static do_b() { ...
    }
}
Circle.do_b()
```

Binary, Octal and Hex Notation

```
> 0b1001011101 // 605
> 0o6745 // 3557
> 0x2f50a // 193802
```

Arrow Function

```
> setTimeout(() => {
… console.log('delayed')
… }, 1000)
```

Equivalent with Anonymous Function

```
> setTimeout(function () {
… console.log('delayed')
… }.bind(this), 1000)
```

Equivalent with IIFE

```
> (function () {
… var cue = 'Luke, I am your
father'
… console.log(cue) // 'Luke, I am
-
… }())
> console.log(cue) // Reference
Error
```

New Scoped Functions

```
> {
… let cue = 'Luke, I am your
father'
… console.log(cue)
… }
> 'Luke, I am your father'
```

String Interpolation, Thanks to Template Literals

```
> const name = 'Tiger'
> const age = 13
> console.log(`My cat is named
${name} and is
${age} years old.`)
> My cat is named Tiger and is 13
years old.
// We can preserve newlines…
let text = ( `cat
dog
nickelodeon`
)
```

Default Params

```
> function howAreYou (answer =
'ok') {
 conso
```