ES6 and Beyond Cheat Sheet

toptal

Constants

```
> const EULER = 2.7182818284
> EULER = 13
> EULER
> 2.7182818284
```

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 ]
```

Binary, Octal and Hex Notation

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

New Types

Symbols, Maps, WeakMaps and Sets

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:

```
> 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
```

Arrow Function

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

Equivalent with Anonymous Function

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

New Scoped Functions

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

Equivalent with Immediately Invoked Function Expressions (IIFE)

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

Object Notation Novelties

```
// 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 (x, y) { … }
}
```

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’) {
 console.log(answer) // probably ‘ok’
}
```

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()
```

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]
}
```

Or Do a Better Push

```
> let arr = [1, 2, 3]
> [...arr, 4, 5, 6]
> [1, 2, 3, 4, 5, 6]
```

...Go Destructuring Like a Boss

```
> const [ cat, dog, ...fish ] = [
'schroedinger',  'Laika', 'Nemo', 'Dori']
> fish // -> ['Nemo', 'Dori']
```

...And Destructuring in the Future ES7

```
{a, b, ...rest} = { a:1, b:2, c:3, d:4 }
```

Async ES7

```
async function schrodinger () {
return new Promise((resolve, reject)
  => {
   const result = Math.random > 0.5
   setTimeout(() => {
     return result ? resolve('alive')
     : reject('dead')
   })
  })
}
```

Await ES7

```
try {
   console.log(await schrodinger())
   // -> 'alive'
} catch (err) {
   console.log(err)
   // -> 'dead'
}
```

Export ES7

```
export function sumTwo (a, b) {
    return a + b;
}
export const EULER = 2.7182818284
let stuff = { sumTwo, EULER }
export { stuff as default }
```

Importing ES7

```
import React from 'react'
import { EULER } from './myexports'
import * as stuff from './myexports'
// equivalent to
import stuff from './myexports'
// { sumTwo, EULER }
```

Generators

They return a objects that implement an iteration protocol. i.e. it has a next() method that returns { value: < some value>, done: <true or false> }.

```
function* incRand (max) { // Asterisk defines this as a generator
  while (true) {
    // Pause execution after the yield, resume
    // when next(<something>) is called
    // and assign <something> to x
    let x = yield Math.floor(Math.random() * max + 1);
    max += x;
  }
}
```

```
> var rng = incRand(2) // Returns a generator object
> rng.next() // { value: <between 1 and 2>, done: false }
> rng.next(3) // as above, but between 1 and 5
> rng.next() // NaN since 5 + undefined results in NaN
> rng.next(20) // No one expected NaN again?
> rng.throw(new Error('Unrecoverable generator state.'))
// Will be thrown from yield
```