www.fabioluiz.de

ES6 Cheat Sheet

'let' and 'const'

Past:

```
var x = 1;
var y = 2;
var sum = x + y;
```

Now instead of 'var', we have 'let' for variables and 'const' for constants.

```
const PERCENTAGE = 0.25;
let oldSalary = 50000;
let newSalary = oldSalary * (1 + PERCENTAGE);
//you can change oldSalary and newSalary as you want,
//but if you try to change PERCENTAGE you got an error
PERCENTAGE = 0.1; //TypeError: Assignment to constant variable.
```

Template strings

```
let name = "Fabio";
let oldStyleConcat = "Hi! Welcome " + name + ".";
let templateString = `Hi! Welcome ${name}.`
```

Arrow functions

```
const multiply = function(x, y){
    return x * y;
};

//now as arrow function
const multiply = (x, y) => {
    return x * y;
};

//this sample has just one expression, so we may omit the return and braces
const multiply = (x, y) => x * y;

//if there is just one parameter, the parenthesis may be omited as well
const squared = number => number * number;
console.log(squared(5)); //output: 25

//another example
const logIt = () => console.log("function without argument must alwas use
parenthesis");
logIt();
```

Problem

```
const gang = {
    developers: ['Fabio', 'Bento', 'Luiz'],
    gangName: 'Back-End developers',
    printMembers: function(){
        return this.developers.map(function(developer){
            //here you reach a point in the code base
            //where keyword 'this' DO NOT represent a gang instance
            return `${this.gangName} has ${developer} as member.`;
        });
    }
};

gang.printMembers();//so it throws TypeError: Cannot read property 'gangName' of undefined
```

ES6 solution

```
const gang = {
    developers: ['Fabio', 'Bento', 'Luiz'],
    gangName: 'Back-End developers',
    printMembers: function(){

        return this.developers.map(developer =>
            //arrow functions use lexical this, so at this point
            //this === gang
            `${this.gangName} has as ${developer} member.`
                                  );
    }
};

gang.printMembers();//now it works!
```

Object Literals

```
const gang = {
    developers: ['Fabio', 'Bento', 'Luiz'],
    gangName: 'Back-End developers',
    //ES5: printMembers: function(){
    //ES6: if the value is a function the 'function' key word and colon may be omited
    printMembers(){
        return this.developers.map(developer => `${this.gangName} has as ${developer}
member.`);
    }
};

gang.printMembers();

function saveFile(url, data){
    //ES5
    //var fileToSave = {url: url, data: data, method: 'POST'};

    //ES6
    //if the key/value has the same name, condense it.
    let fileToSave = {url, data, method: 'POST'};
}
```

Default functions argument

```
//define default values with the parameters
function pointsMessage(points, playerName = 'anonymous'){
    return `${playerName} made ${points} points`;
}

console.log(pointsMessage(10));//output: anonymous made 10 points
console.log(pointsMessage(25, 'Fabio'));//output: Fabio made 25 points
```

Rest and Spread

```
//rest argument accept undefined quantity of parameters
function proccessAll(...parameters){
    //logic
    console.log(parameters);
}

proccessAll(1);//output: [1]
proccessAll(1, 2);//output: [1, 2]
proccessAll(1,2,3,4,5,6,7,8,9);//output: [1, 2, 3, 4, 5, 6, 7, 8, 9]

//concatenation with spread operator
let coolColors = ['blue','green','gray'];
let warmColors = ['yellow','red','orange'];
let allColors = [...coolColors, ...warmColors, 'pink', 'black', 'white'];
console.log(allColors);
//output: ["blue","green","gray","yellow","red","orange","pink","black","white"]
```

Destructuring

```
let person = {
    name: 'Fabio',
    age: 31
}

const { name } = person;
const { age } = person;
//same is: const { name, age } = person;
console.log(`${name} is ${age} years old.`);//output: Fabio is 31 years old.
```

Destructuring arguments

```
let product = {
    description: 'TV',
    price: 1200
}

function showProduct({ description , price }){
    console.log(`The ${description} costs ${price}.`);//output: The TV costs 1200.
}

showProduct(product);
```

Destructuring arrays

```
const products = [
       'TV',
  'Computer',
  'Smartphone'
];

const [ firstProduct, secondProduct, thirdProduct, fourthProduct ] = products;
console.log(firstProduct);//output: TV
console.log(fourthProduct);//output: undefined

const { length } = products;
console.log(length);//output: 3
```

Destructuring arrays of objects

```
const products = [
  { description: 'TV', price: 1200 },
  { description: 'Smartphone', price: 500 },
  { description: 'Computer', price: 2000 }
];

//brackets takes the first element end price takes the price of this element
const [{ price }] = products;
console.log(price);//output: 1200 (TV's price)
```

Another example

```
const products = {
    descriptions: ['TV', 'Computer', 'Smartphone']
};

const { descriptions: [firstProductDescription] } = products;
console.log(firstProductDescription);//output: TV
```

Classes

```
class Animal {

    constructor({ feet }){
        this.feet = feet;
    }

    walk(){
        return `Walking with ${this.feet} feet`;
    }
}

class Dog extends Animal{
    constructor({ feet, name }){
        super(feet);
        this.feet = feet;
        this.name = name;
    }

    bark(){
        return `${this.name} barks: auau!`;
    }
}

const dog = new Dog({ feet: 4, name: 'Max' });
dog.bark();//output: Max barks: auau!
dog.walk();//output: Walking with 4 feet
```

Promises, fetch, then and catch

```
const url = 'https://jsonplaceholder.typicode.com/posts/';

fetch(url)
 .then(response => console.log(response.json()))
 .then(data => console.log(data))
 .catch(error => console.log(error));
//ATTENTION: this catch from native ES6 fetch is reached just if the
//domain is not found, so it DO NOT catch 404 errors, for example.
//Thus may be better to use third part libraries like axios
```

Array helper methods

Past:

```
let prices = [10, 15, 55, 20];

for (var i = 0; i < prices.length; i++) {
    //do something with the array elements
    console.log(prices[i]);
}
```

Now with ECMAScript 6…

For… of loops

```
const products = [
    'TV', 'Computer', 'Smartphone'
];

for(let product of products){
    console.log(product);
};
```

forEach

```
prices.forEach(price => {
    console.log(price);
});

//if it is just one line, braces are not needed:
prices.forEach(price =>
    console.log(price)
);

//as you see it is a function inside the forEach, so it is the same:
function logIt(price) {
    console.log(price);
}
prices.forEach(logIt);
```

Map

```
let doubled = prices.map(price => {
    return price * 2
});
console.log(doubled);//output: [20, 30, 110, 40]

//if it is just one line, besides braces, the 'return' may also be ommited
let doubled = prices.map(price =>
    price * 2
);
```

Filter

```
let evenPrices = prices.filter(price => {
    return price % 2 == 0;
});

console.log(evenPrices);//output: [10,20]
```

Find

```
let computers = [
  {os:"Mac", price: 2000},
  {os:"Windows", price: 1000},
  {os:"Linux", price: 800}
];
let chosenOne = computers.find(computer => {
    return computer.os === "Linux"
});

console.log(chosenOne);//output: {"os":"Linux","price":800}
```

Every

```
let employees = [
  {name:"Jhon", salary: 35000},
  {name:"Smith", salary: 46000},
  {name:"Linda", salary: 50000},
];

//The average salary is 43600.
//All employees earn greater or equal the average?
let allHaveFairSalary = employees.every(employee => {
    return employee.salary >= 43600
});

console.log(allHaveFairSalary);//output: false (as you can see Jhon)
```

Some

```
//Someone earns to much?
let someoneHasHighSalary = employees.some(employee => {
    return employee.salary > 49000
});

console.log(someoneHasHighSalary);//output: false (as you can see Linda)
```

Reduce

```
let employees = [
  {name:"Jhon", salary: 35000},
  {name:"Smith", salary: 46000},
  {name:"Linda", salary: 50000},
];

//sum all salaries using the traditional 'for' way
var sum = 0;
for(var i = 0; i < employees.length; i++){
    sum += employees[i].salary;
}
console.log(sum);//output: 131000

//now using reduce
let result = employees.reduce((sum, employee) => {
    return sum + employee.salary
}, 0);//initialize 'sum' with 0 (zero)

console.log(result);//output: 131000
```

Generator

TODO