# ES6 Cheat Sheet

by arrow96 via cheatography.com/77976/cs/19118/

## Let and Const

| **let and const** | **Hoisting problem** |
|---|---|
| **let** -- allows block scoping and **hoisting problem** in ES5 is solved in ES6.Variables declared with the var keyword can not have Block Scope. Variables declared inside a block {} can be accessed from outside the block. | When we use a undeclared variable with **var keyword** in ES5 we get **undefined variable name** error. This is the example for hositing problem. |
| **const** -- It does NOT define a constant value. It defines a constant reference to a value. Because of this, we cannot change constant primitive values, but we can change the properties of constant objects. | Whereas when we use **let keyword** , hoisting problem is solved in ES6. We get the Error as **Reference Error <variable name> not defined**. |

## This keyword

The JavaScript this keyword refers to the object it belongs to. It has different values depending on where it is used: In a method, this refers to the owner object. Alone, this refers to the global object

**The 4 rules of finding out the value of this keyword**

Rule 1 : When the **keyword this** is not inside the declared object then it refers to the **global object**

Rule 2 : When the **keyword this** is inside the declared object , then it refers to the closest parent object

Rule 3 : whenever the context of the object changes , we use **call , apply and bind** to set the value of this explicitly.

Rule 4 : Whenever we create a object using **new keyword** inside the function definition, the **this keyword** refers to the new object that is being created

## Arrow Functions

Arrow function or fat arrow function -- shorter version of syntax when compared to the normal function

We cannot manipulate the value of **this** keyword inside the arrow function when we use **call,apply or bind**

We do not have access to the **prototype** field when we declare the function using **fat arrow symbol**

## Default function parameters

when we set up default function parameters we get access to the functions and the variables in the context

```
data=( pri ce, cos t=0.07 )=>{ consol e.l og( pri ce* cost) }
data(5.00)
```

## Rest and spread operator

Rest

It allows to convert the no of parameters into an array

It is denoted by "..." in the function definition or function expression

```
a = (...da ta) =>{ consol e.l og( data) } a(2,3, 3,3,3)(5) [2, 3, 3, 3,
 3, 3, 3]
```

The rest parameters must be at the end

Ref -- https://javascript.info/rest-paramet-ers-spread-operator

## Object Literal

It is shorthand for initializing the object properties and also method

Ref -- https://dev.to/sarah_chima/enhanc-ed-object-literals-in-es6-a9d

## Prototype

All JavaScript objects inherit properties and methods from a prototype.

When we create the constructor function , **prototype** property is created for that constr-uctor function

The only inconvenience of using prototypes is that there is no easy way to create private methods or variables.

Ref -- https://stackoverflow.com/questions/-8433459/what-s-the-purpose-of-prototype

Ref -- https://idiallo.com/javascript/why-u-se-prototypes

## for of loop

```
//for of loop is used in
iterable
var a = [1,2,2 ,2,2];
for ( let i of a) {
consol e.l og(i);
}
```

## Octal and binary Literals

```
var a = 0O12; //octal literals
either O or o is allowed
consol e.l og( a)//12
var f = 0b111;
consol e.l og(f);
```

## Template literals

It can create the multiline strings

## new.target

The new.target property lets you detect whether a function or constructor was called using the new operator. In constr-uctors and functions instantiated with the new operator, new.target returns a reference to the constructor or function. In normal function calls, new.target is undefined.

Ref -- https://developer.mozilla.org/en-US/-docs/Web/JavaScript/Reference/Operat-ors/new.target

Example

class A{
constructor(){
this.data = 55;

```
console.log("Inside the base")
console.log(new.target.dumm())
}
}

class B extends A{
constructor(){
super()
console.log(new.target)
console.log(typeof B)
this.data = 66;
console.log(this.data)
}
static dumm(){
return 57;
}
}
```