**Ecma Script 6(2015) Documentation**

***1. Var vs Let:***

* **Variable Overriding:** Problem with ‘var’ was it allows variable declaration without var, and even one can override the variable initialization as well.

x = 3.14;//this goes fine when there was var.

Where it should not be fine, beacause x was even not declared.

var username = ‘rims94’;

console.log(username);//returns ‘rims94’

var username = ‘rimo94’;

console.log(username);//returns ‘rimo94’

This piece of code also works as fine with ‘var’, whether it should not be, because we are initializing the username variable twice.

There we get ‘let’. It throws errors for both piece of code, maintaining all aspects of programming fine.

* **Scoping Issue:** When you declare a variable with the var keyword, it is declared globally, or locally if declared inside a function.

The let keyword behaves similarly, but with some extra features. When you declare a variable with the let keyword inside a block, statement, or expression, its scope is limited to that block, statement, or expression.

Ex:

var printNumTwo;

for (var i = 0; i < 3; i++) {

if (i === 2) {

printNumTwo = function() {

return i;

};

}

}

console.log(printNumTwo());

// returns 3

printNumTwo() prints 3 and not 2. This is because the value assigned to i was updated and the printNumTwo() returns the global i and not the value i had when the function was created in the for loop.

In the case of ‘let’ i is not defined because it was not declared in the global scope. It is only declared within the for loop statement. printNumTwo() returned the correct value because three different i variables with unique values (0, 1, and 2) were created by the let keyword within the loop statement.

'use strict';

let printNumTwo;

for (let i = 0; i < 3; i++) {

if (i === 2) {

printNumTwo = function() {

return i;

};

}

}

console.log(printNumTwo());

// returns 2

console.log(i);

// returns "i is not defined"

***2. Const:***

Const has all the awesome features that let has, with the added bonus that variables declared using const are read-only. They are a constant value, which means that once a variable is assigned with const, it cannot be reassigned.

"use strict";

const FAV\_PET = "Cats";

FAV\_PET = "Dogs"; // returns error

However, it is important to understand that objects (including arrays and functions) assigned to a variable using const are still mutable. Using the const declaration only prevents reassignment of the variable identifier.

"use strict";

const s = [5, 6, 7];

s = [1, 2, 3]; // throws error, trying to assign a const

s[2] = 45; // works just as it would with an array declared with var or let

console.log(s); // returns [5, 6, 45]

***3. Prevent Object Mutation:***

const declaration alone doesn't really protect your data from mutation. To ensure your data doesn't change, JavaScript provides a function Object.freeze to prevent data mutation.

Once the object is frozen, you can no longer add, update, or delete properties from it. Any attempt at changing the object will be rejected without an error.

let obj = {

name:"SayanGhosh",

review:"Cool"

};

Object.freeze(obj);

obj.review = "Hot Head"; // will be ignored. Mutation not allowed

obj.newProp = "Job"; // will be ignored. Mutation not allowed

console.log(obj);

// { name: "SayanGhosh", review:"Cool"}

***4. Arrow Functions to Write Concise Anonymous Functions:***

In javascript we can shorten the function declarations, that are not gonna be reuse in a code by arrow function.

let myFunc = function showValue(value) {

return value;

};

This is a simple function declaration in javascript. The javascript have syntactic sugar to shorten this.

let myFunc = (value)=>{

return value;

}

This is the same fuction call using the arrow function. Change is here we remove the function keyword along with the function name and add a ‘=>’ operator instead of that. Now this doesn’t seem to be huge help. But we can even more shorten this,here is an example.

let myFunc = value => value;

Yes, thats it, this does the same task as the previous two.

When there is only one parameter to pass through the function, its not mandatory to have the parenthisis, and even when there is only one line in the program statement,it also be can pass in the same line and without the curly braces.

Otherwise, a general arrow function look like the following.

let addTwoNos = (number1, number2) => {

let sum = number1+number2;

return sum;

};

We can also define default value of parameters in arrow function.

let addTwoNos = (number1=0, number2=0) => {

let sum = number1+number2;

return sum;

};

console.log(addTwoNos());//returns 0,as no arguments passed, so takes the default value

console.log(addTwoNos(10));//returns 10,as no 2nd arguments passed, so takes the default value

console.log(addTwoNos(10,20);//returns 30,as both arguments are given

***5. Rest Parameter as Function Parameters:***

In order to help us create more flexible functions, ES6 introduces the rest parameter for function parameters. With the rest parameter, you can create functions that take a variable number of arguments. These arguments are stored in an array that can be accessed later from inside the function.

function howMany(...args) {

return "You have passed " + args.length + " arguments.";

}

console.log(howMany(0, 1, 2)); // You have passed 3 arguments.

console.log(howMany("string", null, [1, 2, 3], { })); // You have passed 4 arguments.

The rest parameter eliminates the need to check the args array and allows us to apply map(), filter() and reduce() on the parameters array.

***6. Spread Operator:***

ES6 introduces the spread operator, which allows us to expand arrays and other expressions in places where multiple parameters or elements are expected.

The ES5 code below uses apply() to compute the maximum value in an array:

var arr = [6, 89, 3, 45];

var maximus = Math.max.apply(null, arr); // returns 89

We had to use Math.max.apply(null, arr) because Math.max(arr) returns NaN. Math.max() expects comma-separated arguments, but not an array. The spread operator makes this syntax much better to read and maintain.

const arr = [6, 89, 3, 45];

const maximus = Math.max(...arr); // returns 89

...arr returns an unpacked array. In other words, it spreads the array. However, the spread operator only works in-place, like in an argument to a function or in an array literal. The following code will not work:

const spreaded = ...arr; // will throw a syntax error