**JavaScript 6 version Features**

**let**

>let is a keyword.

>let used as alternative of var keyword.

>by using let we can create local var or global var

>let used for creating a variable with fixed datatype.

Syn: **let varname=value;**

ex: let a=10;

datatype of a is number, and we can't change datatype in rest of program.

**const**

>const is a keyword.

>const kw used for creating a constant variables.means once we assign any value to variable, we can't change.

Syn: **constvarname=value;**

ex: const a=10;

we can't change value of a var.

**Note: while using const keyword, don't use var& let keywords.**

**Spread operator**

> "**...**" is unary operator, we should use this operator as prefix.

>Its rest and spread an operator.

>the rest operator represents all remaining values/so on values.

>this we can use in methods and arrays.

**Array Syn: ...array 🡸spread**

**…collection🡸spread**

**Method Syn: function-name(...param)🡸 rest**

**de-structuring**

>destructing is used to retrieve each value of an array into separate/respective variables/array.

**Syn1: let [var1,var2,var3.....] =arrayname; <== left to right**

means: var1=arrayname[0];

var2=arrayname[1];

**Syn2: let [var1,var2, , , ,var3] =arrayname; <== left to right (with skip)**

means: var1=arrayname[0];

var2=arrayname[1];

var3=arrayname[4]; **<==** 2cell skipped

**Syn3: let [var1,var2, ...var3] =arrayname; <== left to right (with skip)**

means: var1=arrayname[0];

var2=arrayname[1];

var3=rest of cells (2-last)

**backtickoperator (template lit)**

>this feature is used to create a string with multiple lines of text.

>multiline string should be enclosed/represented with **backtick** (**` `**)

**Syn: var=`**line1

line2

line3....**`;**

“ “+

**String interpolation**

>string interpolation replaces the expressions in the string with actual values of the specified variables.

> sub dynamically var/expr/fun-call in between strings

> operator is **${ }**

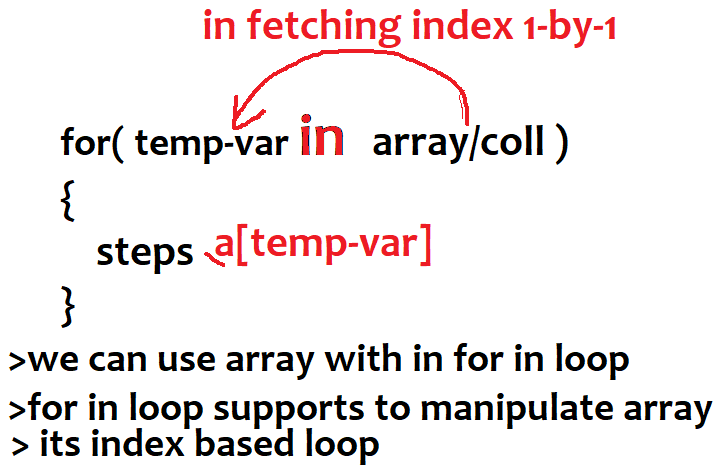
>string should be enclosed with in "**backtick**" (**` `**), but not **""** and **' '**.

**Syn:** ${var} ${expr} ${fun call}

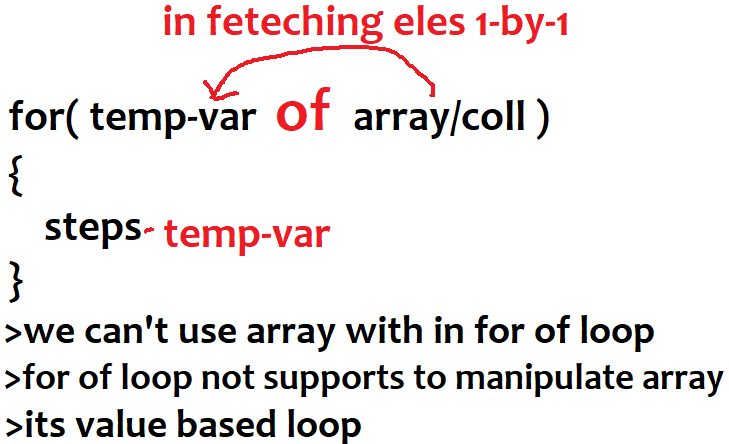
**`**text **${var}** text **${expr}** text **${fun}** ...**`**

**Use:**while printing data & assigning values to variable

**Forin loop**

****

**for of loop**



**Arrow functions**

ES6 arrow functions provide you with an alternative way to write a shorter syntax compared to the function expression.

Specifying parameters:

() => statement

() =>{ ... } *// no parameter*

param =>{ ... } *// one parameter, an identifier*

(param1, param2…) =>{ ... } *// several parameters*

Specifying a body:

x =>{ **return** x \* x } *// block*

x => x \* x *// expression, equivalent to previous line*

**No line break after arrow function parameters**

ES6 forbids a line break between the parameter definitions and the arrow of an arrow function:

**const** func1 = (x, y) *//SyntaxError*

=> {

return x + y;

};

**const** func2 = (x, y) =>*// OK*

{

return x + y;

};

**const** func3 = (x, y) =>{ *// OK*

return x + y;

};

**const** func4 = (x, y) *// SyntaxError*

=> x + y;

**const** func5 = (x, y) =>*// OK*

x + y;

Line breaks *inside* parameter definitions are OK:

**const** func6 = ( *// OK*

x,

y

) => {

**return** x + y;

};

let numbers = [4,2,6];

numbers.sort((a,b) => b - a);

console.log(numbers); // [6,4,2]

**The following example uses an arrow function as an argument of the**[**map()**](https://www.javascripttutorial.net/javascript-array-map/)**method that transforms an array of strings into an array of the string’s lengths.**

let names = ['Apple', ‘Orange’, 'Mango’, ‘Banana’];

let lengths = names.map(name =>name.length);

console.log(lengths);

**If you use an expression in the body of an arrow function, you don’t need to use the curly braces.**

let square = x => x \* x;

**JavaScript arrow functions and object literal:**

letsetColor = function (color) {

return {value: color}

};

letbackgroundColor = setColor('Blue');

console.log(backgroundColor.value); *// "Blue"*

**object literal from an arrow function**

letsetColor = color => ({value: color });

When You Should Not Use Arrow Functions?

An [arrow function](https://www.javascripttutorial.net/es6/javascript-arrow-function/) doesn’t have its own [this](https://www.javascripttutorial.net/javascript-this/) value and the arguments object. Therefore, you should not use it as an event handler, a method of an object literal, a prototype method, or when you have a function that uses the arguments object.