Basics for React

let keyword:-

① Var → declaration gives functional scope. (It can be declared after using variable also)

Let → gives block scope (It is like variables declaration in c, c++, etc...) should be used after declaration of variables only.

```
Ex:- var a=1;
     var b=2;
     if (a===1) {
        var a=10;
        var b=20;
        console.log(a);
        console.log(b);
     }
     console.log(a);
     console.log(b);
```

Output:-
```
10
20
10
2
```

②

We can declare	But we cannot declare
var c=1;	let d=1;
var c=2;	let d=2;
in same fun or block	in same block

Const keyword:-

* Behaves same as let, but the assigned value cannot be changed in the same scope.
* Const keyword ~~know~~ used for constants.
* The constant should be initialize while declaring only.

```
Ex:- const a=10; ✓
     const a;
     a=10; ×
```

Arrow functions :-

```
var getRegValue = function(){
    return 10;
}
console.log(getRegValue());
```

/* regular function */

OUTPUT

10

Now arrow functions

1. const getArrowvalue = () => {
    return 10;
}
console.log(getArrowvalue()); //10.

/* Arrowfunctions format)

2. Const getArrowvalue = () => 10;
console.log(getArrowvalue()); //10

/* We can remove return & curly braces for one statement if its a return */

3. const getArrowvalue = n => 10*n;
console.log(getArrowvalue(2)); //20

/* For single argument no need parathesis */

Default Parameters :-

ex:-
```
let getValue = function (value=10){
    console.log(value);
};
getValue(); //10
getValue(20); //20
```

/* Without default parameter it gives undefined as output if we won't call fun without arguments */

* We can also do as below.

Ex:-
```
let getvalue = function(value=10, bonus=value*0.1){
    console.log(value+bonus);
};
```

/* We can also replace 0.1 with a variable */
(or) function calling also

Note:-

console.log(arguments.length); gives no of arguments passed to a function.

* In above example arguements as (value=bonus*10, bonus=1) is not possible.

Rest operator:-

Ex:-
```
let displayColors = function(){
    for(let i in arguments){
        console.log(arguments[i]);
    }
}
displayColors('Red');          // Red
displayColors('Red','Blue');   // Red
                               Blue.
```

Ex:-
```
let displayColors = function(){
    for(let i in arguments){
        console.log(arguments[i]);
    }
}
let Message = "ya";
displayColors(Message,'Red');
/* ya
   Red */
displayColors(message,'Red',
              'Blue');
/* ya
   Red
   Blue */
```

* Above are written to understand arguments Array (default Array).

* But the problem is :-

Ex:

```
let displayColors = function(){
    console.log(message);
    for(let i in arguments){
        console.log(arguments[i]);
    }
}
let message = "ya";
displayColors(message, 'Red'); /* ya
                                  ya
                                  Red */

displayColors(message, 'Red', 'Blue'); /* ya
                                          ya
                                          Red
                                          Blue */
```

* To solve that Repetation use rest operator (...) :-

Now program is :-

```
let displayColors = function(message, ...colors){
    console.log(message);
    for(let i in colors){
        console.log(colors[i]);
    }
}
let message = "ya";
displayColors(message, 'Red'); /* ya
                                  Red */
displayColors(message, 'Red', Blue'); /* ya
                                         Red
                                         Blue */
```

* List of arguments are converted in to an array (here colors)
* There we come to know that rest operator takes variable no of parameters.

Spread operator :-

* It is opposite to rest operator.
* It is used to split away into individual items.
* spread is used at calling a function, rest operator is used at defining a function.

Ex:-

```
let displayColors = function (message, ...colors ){ // here ... is rest operator.
        console.log (message);
        for(let i in colors){
            console.log(colors[i]);
        }
    }
let message = "list of colors";
let colorArray = ['Orange','yellow', 'Indigo'];
displayColors (message, ... colorArray); // here ...spread operator.
```

OUTPUT :-

```
list of colors
Orange
yellow
Indigo
```

Object literals :-

①
```
let firstname = 'ok';
let lastname = 'boss';

let person = {
    firstname : firstname,
    lastname : lastname
};
```
/* object literal */

instead of writing as above we can write as

```
let person = {
    firstname,
    lastname
};                    /*because value name & variable name) & key
                        name are same */

console.log(person.firstname); // ok
console.log(person.lastname); // boss..
```

②

```
function createPerson(firstname, lastname, age){
    let fullname = firstname + " " + lastname;
    return {
        firstname,
        lastname,
        fullname,
        isSenior: function(){
            return age>60;
        }
    }
}

let P = createPerson("ok", "boss", 40);
console.log(P.firstname);    // ok
console.log(P.lastname);     // boss
console.log(P.fullname);     // ok boss
console.log(P.isSenior());   // false.
```

/* no need key and value pair, because variable and key names are same */

/* assigning fun to key so name of fun is key name */

/* returning a object */

/* It also works with arrow fun in return statement and also

```
isSenior(){
    return age>60;
}
```

can work.
*/

③

```
let In = "kanta";
let person = {
    "mani": "firstname",
    [In]: "lastname".
};
console.log(person);          // {mani: "firstname", kanta: "lastname"}.
console.log(person["mani"]);  // firstname
```

Destructuring :-

destructuring arrays :-

① let employee = ["abc", "xyz", "male"];

let [fname, lname, gender] = employee;

```
console.log(fname);  // abc
Console.log(lname);  // xyz
console.log(gender); // male.
```

② let employee = ["abc", "xyz"];

a) let [fname, lname, gender = "male"] = employee;

```
console.log(gender);  // male.
```

b) let [fname, lname, gender] = employee;

```
console.log(gender);  // undefined.
```

③ let employee = ["abc", "xyz", "male"];

a) let [, , gender] = employee; // Here we need only gender so,...

```
console.log(gender)  // male
```

b) let [fname, ...elements] = employee;

```
console.log(fname)  // abc.
console.log(elements)  // ["xyz", "male"]
```

destructuring objects :-

*It is same as destructuring arrays but we use curly braces and property name (key name) and variable name must be same.

Ex:-

```
let info = {
    fname : "mani",
    lname : "kanta"
};
```

```
let {fname, lname} = employee;
console.log(fname);        // mani
console.log(lname);        // kanta.
```

(or)

use alias for variable names.

```
let {fname :f, lname : l} = employee;
console.log(f);        //mani
console.log(l);        //kanta.
```

String Templates:-

①
```
let user = "chandler";
let greet = "welcome" + user + "sir";
```
(or)
```
let greet = `welcome ${user} sir`;   /* ` ` backtick is used */
console.log(greet); // welcome chandler sir.
```

② Multiple lines

```
let greet = `welcome sir to
            our show`;
console.log(greet);    // welcome sir to
                       our show
```