What is ES6?

ES6 stands for ECMAScript 6.

ECMAScript was created to standardize JavaScript, and ES6 is the 6th version of ECMAScript, it was published in 2015, and is also known as ECMAScript 2015.

ES6 comes with significant changes to the JavaScript language. It brought several new features like, let and const keyword, rest and spread operators, template literals, classes, modules and many other

Why Should I Learn ES6?

React uses ES6, and you should be familiar with some of the new features like:

* [Arrow Functions](https://www.w3schools.com/react/react_es6_arrow.asp)
* [Variables](https://www.w3schools.com/react/react_es6_variables.asp) (let, const, var)
* [Array Methods](https://www.w3schools.com/react/react_es6_array_methods.asp) like Map, Reduce and Filter
* [Destructuring](https://www.w3schools.com/react/react_es6_destructuring.asp)
* [Modules](https://www.w3schools.com/react/react_es6_modules.asp)
* [Ternary Operator](https://www.w3schools.com/react/react_es6_ternary.asp)
* [Spread Operator](https://www.w3schools.com/react/react_es6_spread.asp)

History:

The ECMAScript specification is the standardized specification of scripting language, which is developed by **Brendan Eich (**He is an American technologist and the creator of JavaScript programming language**)**of**Netscape (**It is a name of brand which is associated with Netscape web browser's development**).**

Initially, the ECMAScript was named **Mocha,**later **LiveScript,**and finally, **JavaScript.**In December 1995, **Sun Microsystems**and **Netscape**announced the JavaScript.

Prerequisite :

Before learning ES6, you should have a basic understanding of JavaScript.

**Diff between JavaScript and ES6 :**

**Using JavaScript:**

Largest number:-

var arr = [3, 6, 2, 56, 32, 5, 89, 32];

var largest = arr[0];

for (var i = 0; i < arr.length; i++) {

if (arr[i] > largest ) {

largest = arr[i];

}

}

console.log(largest);

**Using ES6:**

let numbers = [3, 7, 2, 8, 5];

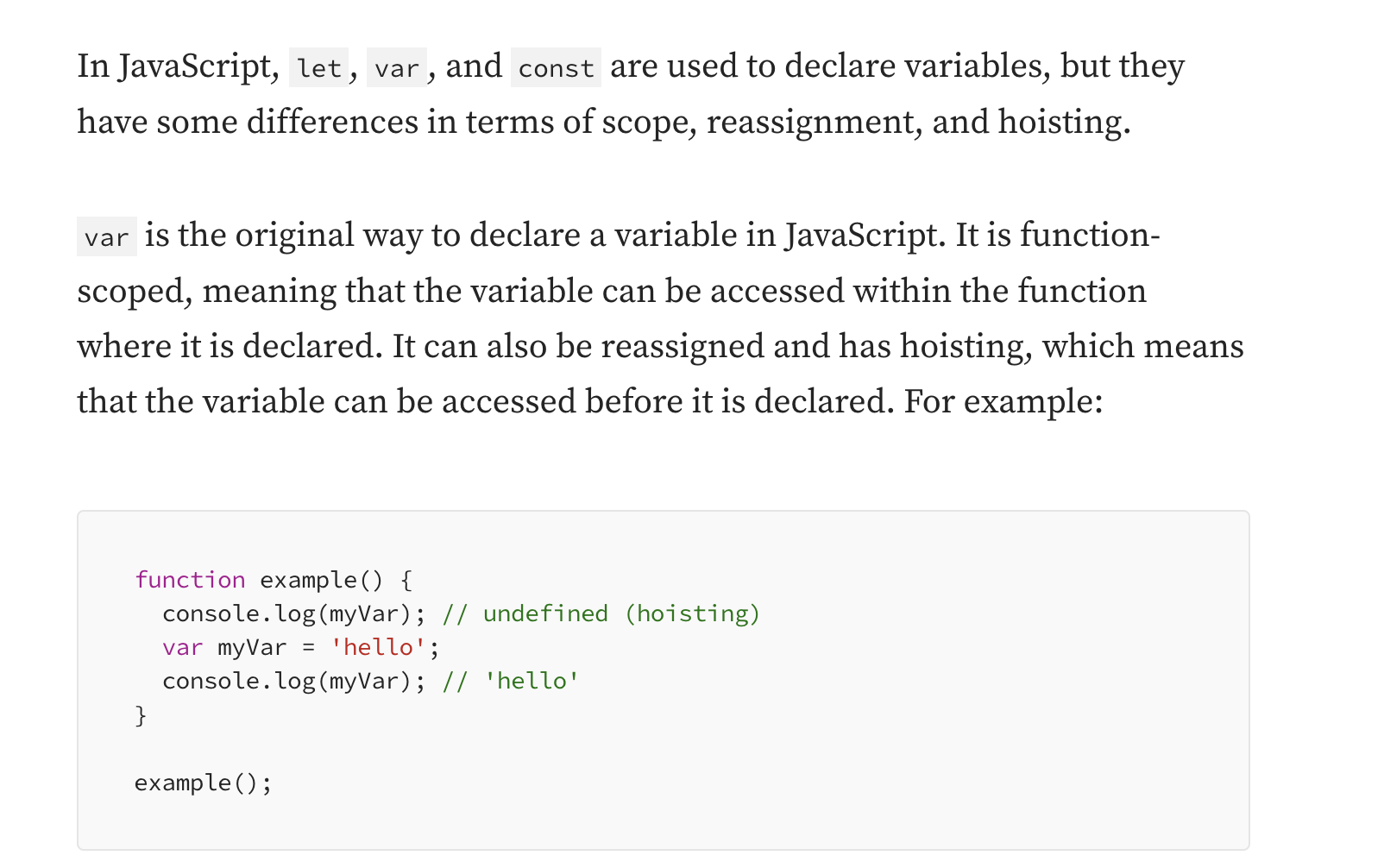
let max = Math.max(...numbers);

console.log(max);

**Difference between ES5 and ES6 :**

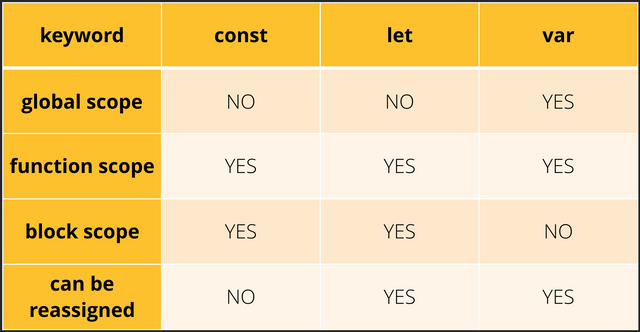
| **ES5** | **ES6** |
| --- | --- |
| ECMA script is a trademarked scripting language specification defined by Ecma International. The fifth edition of the  same is known as ES5 | ECMA script is a trademarked scripting language specification defined by Ecma International. The sixth edition of the same is known as ES6 |
| It was introduced in 2009. | It was introduced in 2015. |
| There is only one way to define the variables by using the var keyword. | There are two new ways to define variables that are let and const. |
| It has a lower performance as compared to ES6. | It has a higher performance than ES5. |
| Object manipulation is time-consuming in ES5. | Object manipulation is less time-consuming in ES6. |

**Diff b/w Var vs Let vs Const :**

****

**A screenshot of a computer program

Description automatically generated**



Local Scope:

The local scope is limited to the code or function in which the variable is declared.

function print() {

var number = 50;

  var square = number \* number;

  console.log(square);

}

print();

console.log(number);

Global scope:

Global scope is the entire program. Global variables can be used anywhere throughout the program.

var number = 50;

function print() {

  var square = number \* number;

  console.log(square);

}

print();

console.log(number);

Arrow Function:

Arrow function {()=>} is concise way of writing JavaScript functions in shorter way. Arrow functions were introduced in the ES6 version. They make our code more structured and readable. Arrow functions are anonymous functions i.e. functions without a name but they are often assigned to any variable.

ES5 Function:

function greet(name){

return "hello " + name;

}

console.log(greet("vijay"));

ES6 Function:

let greet = (name) => {

return "hello " + name;

}

console.log((greet("Vijay")));

ES6 Operators:

The operator can be defined as a symbol that tells the system to implement a particular operation.

The operators are used in the expressions for evaluating different operands.

An expression is a kind of statement that returns a value. The expression includes:

* **Operators:** It is responsible for performing some operations on operands
* **Operands:** It represents the data.

**For example:** Suppose an expression like**x+y.** In this expression,**x**and **y**are the **operands,** and the addition **(+)** symbol is the addition operator.

## **Types of Operators**

* Arithmetic Operators
* Relational Operators
* Logical Operators
* Assignment Operators
* Bitwise Operators
* Type Operators
* Miscellaneous Operators

### **Arithmetic Operators**

Arithmetic operators are the basic mathematical operators that are available in JavaScript [ES6](https://www.javatpoint.com/es6). These operators are responsible for performing all mathematical operations such as addition, subtraction, etc. in JavaScript.

|  |  |
| --- | --- |
| **Operators** | **Functions** |
| **+ (Addition)** | It returns the sum of the value of operands |
| **- (Subtraction)** | It returns the difference between the value of operands |
| **\* (Multiplication)** | It returns the product of operands values. |
| **/ (Division)** | It is used to perform division, and it returns quotient. |
| **% (Modulus)** | It also performs division and returns the remainder. |
| **++ (Increment)** | It increments the value of a variable by one. |
| **- (Decrement)** | It decrements the value of a variable by one. |

**For example**

var x = 30;

var y = 20 ;

console.log("Addition: " + (x + y) );

console.log("Subtraction: " + (x - y) );

console.log("Multiplication: " + (x \* y) );

console.log("The Division will give you the quotient: " + (x / y) );

console.log("Modulus will give you the Remainder: " + (x % y) );

// pre-increment

console.log("Value of x after pre-increment: "  + (++x) );

// post-increment

console.log("Value of x after post-increment: " + (x++) );

// pre-decrement

console.log("Value of y after pre-decrement: "  + (--y) );

// post-decrement

console.log("Value of y after post-decrement: " + (y--) );

### **Relational Operators**

Relational operators are used for comparing the two values and return either true or false based on the expression. These operators are sometimes called Comparison Operators.

|  |  |
| --- | --- |
| **Operator** | **Function** |
| **> (Greater than)** | It returns true if the left operand is greater than right operand else it returns false. |
| **< (Less than)** | It returns true if the left operand is smaller than right operand else it returns false. |
| **>= (Greater than or equal to)** | It returns true if the left operand is greater than or equal to right operand else it returns false. |
| **<= (Less than or equal to)** | It returns true if the left operand is smaller than or equal to right operand else it returns false. |
| **== (Equality)** | It returns true if the value of both operands is the same else it returns false. |
| **!= (Not Equal to)** | It returns true if the value of operands is not the same else it returns false. |

**For example:**

var x = 20;

var y = 15;

console.log(5<6<7);

console.log(7>6>5);

console.log("Value of x: " + x);

console.log("Value of y: " + y);

var result = x > y;

console.log("x is greater than y: " + result);

result = x < y;

console.log("x is smaller than y: " + result);

result = x >= y;

console.log("x is greater than or equal to  y: " + result);

result = x <= y;

console.log("x is smaller than or equal to y: " + result);

result = x == y;

console.log("x is equal to y: " + result);

result = x != y;

console.log("x not equal to  y: " + result);

### **Logical Operators**

Logical operators are generally used for combining two or more relational statements. They also return Boolean values.

|  |  |
| --- | --- |
| **Operators** | **Description** |
| **&& (Logical AND)** | This operator returns true if all relational statements that are combined with && are true, else it returns false. |
| **|| (Logical OR)** | This operator returns true if at least one of the relational statements that are combined with || are true, else it returns false. |
| **! (Logical NOT)** | It returns the inverse of the statement's result. |

**For example:**

var x = 30;

var y = 80;

console.log("Value of x = " + x );

console.log( "Value of y = " + y );

 var result = ((x < 40) && (y <= 90));

console.log("(x < 40) && (y <= 90): ", result);

var result = ((x == 50) || (y > 80));

console.log("(x == 50) || (y > 80): ", result);

var result = !((x > 20) && (y >= 80));

console.log("!((x > 20) && (y >= 80)): ", result);

Assignment Operators

Assignment operators are used for assigning a value to the variable. The operand on the left side of the assignment operator is a variable, and the operand on the right side of the assignment operator is a value.

The right-side value must be of the same data-type of the left-side variable; otherwise, the compiler will raise an error.

|  |  |
| --- | --- |
| **Operators** | **Functions** |
| **= (Simple Assignment)** | It simply assigns the value of the right operand to the left operand |
| **+= (Add and Assignment)** | This operator adds the value of the right operand to the value of the left operand and assigns the result to the left operand. |
| **-= (Subtract and Assignment)** | This operator subtracts the value of the right operand from the value of the left operand and assigns the result to the left operand. |
| **\*= (Multiply and Assignment)** | This operator multiplies the value of the right operand to the value of the left operand and assigns the result to the left operand. |
| **/= (Divide and Assignment)** | This operator divides the value of the right operand to the value of the left operand and assigns the result to the left operand. |

For Example

var x = 20;

var y = 40;

x = y;

console.log("After assignment the value of x is:  " + x);

x += y;

console.log("x+=y: " + x);

x -= y;

console.log("x-=y: " + x);

x \*= y;

console.log("x\*=y: " + x);

x /= y;

console.log("x/=y: " + x);

x %= y;

console.log("x%=y: " + x);

### **Bitwise Operators**

Bitwise operators are used for performing the bitwise operations on binary numerals or bit patterns that involve the manipulation of individual bits. Bitwise operators perform the operation on the binary representation of arguments

Generally, bitwise operators are less used and relevant for the applications and hyper-performance programs.

|  |  |
| --- | --- |
| **Operator** | **Description** |
| **Bitwise AND (&)** | It compares every bit of the first operand to the corresponding bit of the second operand. If both of the bits are 1, then the result bit will set to 1, else it will set to 0. |
| **Bitwise OR (|)** | It compares every bit of the first operand to the corresponding bit of the second operand. If both of the bits are 0, then the result bit will set to 0, else it will set to 1. |
| **Bitwise XOR (^)** | It takes two operands and does XOR on each bit of both operands. It returns 1 if both of the two bits are different and returns 0 in any other case. |
| **Bitwise NOT (~)** | It flips the bits of its operand, i.e., 0 becomes 1 and 1 becomes 0. |
| **Left shift (<<)** | It shifts the value of the left operand to the left by the number of bits specified by the right operand. |
| **Sign-propagating Right shift (>>)** | It shifts the value of the left operand to the right by the number of bits specified by the right operand. This is sign-propagating because the bits that we are adding from the left depends upon the sign of the number (0 represents positive, and 1 represents negative). |

**For example:**

In this example, we are using all logical operators that are listed above.

var x = 70; /\* 70 = 0100 0110 \*/

var y = 80; /\* 80 = 0101 0000 \*/

var res = 0;

console.log("Value of 70 in binary 0100 0110" );

console.log("Value of 80 in binary 0101 0000" );

res = x & y;       /\* 64 = 0100 0000 \*/

console.log("Value of x & y = %d\n", res );

res = x | y;       /\* 86 = 0101 0110 \*/

console.log("Value of x | y = %d\n", res );

res = x ^ y;       /\* 22 = 0001 0110 \*/

console.log("Value of x ^ y = %d\n", res );

res = ~x;          /\*-71 = - 10111001\*/

console.log("Value of ~ x = %d\n", res );

res = x << 2;     /\* 280 = 1000 11000 \*/

console.log("Value of x << 2 = %d\n", res );

res = x >> 2;     /\* 17 = 0001 0001 \*/

console.log("Value of x >> 2 = %d\n", res );

### **Miscellaneous Operators**

These are the operators that perform different operations in different circumstances.

|  |  |
| --- | --- |
| **Operators** | **Description** |
| **+ (Concatenation Operator)** | It applies to strings and appends the second string to first. |
| **- (Negation Operator)** | It changes the sign of the value. |
| **? (Conditional Operator)** | It is used for representing the conditional expression. It is also called a **ternary operator.** |

Let us try to understand the miscellaneous operators in detail:

### **The Negation Operator (-)**

It is used to change the sign of the value.

**For example:**

var num1 = 80;

var num2 = -num1;

console.log("Value of num1 = " +num1); // It will give 80

console.log("Value of num2 = " +num2); // It will give -80

### **The Concatenation Operator (+)**

It applies on strings and appends the second string to first. You can understand it by using the following example:

**Example:**

var str1 = 'Hello' + 'World';

var str2 = 'Welcome ' + 'Back';

console.log(str1);

console.log(str2);

### **The Conditional Operator (?)**

This operator represents the conditional expression. It is also called the **'ternary operator.'**

**Syntax:**

1. variablename = (condition) ? value1 : value2

Where,

**condition:** It refers to the conditional expression.

**value1:** If the condition is true, then this value will be returned.

**value2:** If the condition is false, then this value will be returned.

**Example:**

var num1 = 30;

var num2 = 20;

var res = num1 > num2 ? "Yes 30 is greater than 20" : "No It's not";

console.log(res);

Type Operators

It is a unary operator that returns the data type of the operand.

**Syntax:**

1. typeof(operand)

You can see the data types and values in the following table that are returned by the **typeof** [operator in JavaScript](https://www.javatpoint.com/javascript-operators):

|  |  |
| --- | --- |
| **Type** | **String Returned by typeof** |
| String | "string" |
| Boolean | "boolean" |
| Number | "number" |
| Object | "object" |

**Example:**

var a = 20;

var b = **true**;

var c = 'Hello';

var d = 'true';

var e;

console.log("Variable a is " +typeof(a));

console.log("Variable b is " +typeof(b));

console.log("Variable c is a " +typeof(c));

console.log("Variable d is a " +typeof(d));

console.log("Variable e is " +typeof(e));

ES6 Template Literals:

ES5:

var name = "Cognizant";

console.log("welcome to" + name + "!Have a nice day");

// multiline

var name = "Cognizant";

var res = "welcome \n to "+ name ;

// console.log("welcome to" + name + "!Have a nice day");

console.log(res);

ES6:

const name = "Cognizant";

console.log(`welcome to ${name} !Have a nice day`); // tilt

// multiline

const name = "Cognizant";

console.log(`welcome to

${name}

!Have a nice day`); // tilt

ES6 Arrays:

[Array in JavaScript](https://www.javatpoint.com/javascript-array) is an object which is used to represent a collection of similar type of elements. It allows you to store more than one value or a group of values in a single variable name.

We can store any valid values such as objects, numbers, strings, functions, and also other arrays, which make it possible to create complex data structures like an array of arrays or an array of objects.

### **Syntax**

var array\_name = **new** Array();  // By using the new keyword

var array\_name = [value1, value2,....valueN];  //By using Array literals

                or,

var array\_name;   //Declaration

array\_name=[value1, value2,…..valueN]; //Initialization

## **JavaScript Arrays**

JavaScript supports the following categories of arrays.

* Multidimensional array
* Passing arrays to functions
* Return array from functions

### **ES6 Multidimensional Arrays**

[ES6](https://www.javatpoint.com/es6) also supports the multidimensional array concept. A multidimensional array can be defined as an array reference to another array for its value.

Ex:

var multi = [[2,3,4],[4,9,16]]

console.log(multi[0][0])

console.log(multi[0][1])

console.log(multi[0][2])

console.log(multi[1][0])

console.log(multi[1][1])

console.log(multi[1][2])

### **Passing Array to function:**

Passing array as an argument to a function, you have to specify the array name (a reference to an array) without brackets. Let us try to understand it with the following example.

**Example**

var rainbow = ["Violet", "Indigo", "Blue", "Green", "Yellow", "Orange", "Red"];

function show(rainbow) {

for(var i = 0;i<rainbow.length;i++) {

console.log(rainbow[i]);

}

}

show(rainbow);

Return Array from function

It allows a function to return an array.

**Example**

function show() {

return new Array("Blue", "Red", "Green", "Yellow") ;

}

var colors = show()

for(var i = 0;i<colors.length;i++) {

console.log(colors[i])

}

ES6 Arrays Methods:

1. Map
2. Filter
3. Reduce

Map:

Const arr = [5,1,3,2,6];

If we want to transform this array means transform each and every value of this array to get a new array.

Ex:

//double – [10,2,6,4,12];

//triple – [15,3,9,6,18];

//Binary[“101”, “001”, “011”,”010”,”110];

Filter:

Is basically used to filter.

Const arr = [5,1,3,2,6];

Suppose if we want to filter value let’s say value is grater than 4.

// to filter out even numbers

// to filter out odd numbers

// to filter out all numbers which is divisible by 3.

Reduce:

Take all the elements of an array and come up with a single value out of them.

Const arr = [5,1,3,2,6];

//sum or max

const user = [

{ firstname: "Vijay", lastname: "kumar", age: 22 },

{ firstname: "Nithin", lastname: "kumar", age: 45 },

{ firstname: "Ravi", lastname: "Chandra", age: 21 },

{ firstname: "Raghu", lastname: "veeru", age: 67 }, ];

Destructuring in ES6:

The destructuring assignment syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

Object destructuring

It is similar to array destructuring except that instead of values being pulled out of an array, the properties (or keys) and their corresponding values can be pulled out from an object.

When destructuring the objects, we use keys as the name of the variable. The variable name must match the property (or keys) name of the object. If it does not match, then it receives an **undefined** value. This is how JavaScript knows which property of the object we want to assign.

In object destructuring, the values are extracted by the keys instead of position (or index).

**const** num = {

x: 100,

y: 200

};

**const** {x, y} = num;

console.log(x); // 100

console.log(y); // 200

### **Example - Basic Object destructuring assignment**

**const** student = {

name: 'Arun',

position: 'First',

rollno: '24'

};

**const** {name, position, rollno} = student;

console.log(name); // Arun

console.log(position); // First

console.log(rollno); // 24

ES6 Array destructuring

Destructuring means to break down a complex structure into simpler parts. With the syntax of destructuring, you can extract smaller fragments from objects and arrays. It can be used for assignments and declaration of a variable.

Destructuring is an efficient way to extract multiple values from data that is stored in arrays or objects. When destructuring an array, we use their positions (or index) in an assignment.

### **Example1:**

var arr = ["Hello", "World"]

// destructuring assignment

var [first, second] = arr;

console.log(first); // Hello

console.log(second); // World

### **Example2:**

var colors = ["Violet", "Indigo", "Blue", "Green", "Yellow", "Orange", "Red"];

// destructuring assignment

var[color1, color2, color3] = colors;

console.log(color1); // Violet

console.log(color2); // Indigo

console.log(color3); // Blue

### **Example**

If you want to choose random elements from the given array then in array destructuring you can perform it as follows:

var colors = ["Violet", "Indigo", "Blue", "Green", "Yellow", "Orange", "Red

// destructuring assignment

var[color1, ,color3, ,color5] = colors; //Leave space for unpick elements

console.log(color1); // Violet

console.log(color3); // Blue

console.log(color5); // Yellow

In the above example, we have defined an array named **colors** which has seven elements. But we have to show three random colors from the given array that are **Violet, Blue,** and **Yellow**. These array elements are in positions **0, 2,** and **4**.

During destructuring, you have to leave the space for unpick elements, as shown in the above example. Otherwise, you will get unwanted results.

## **Array destructuring and Rest operator**

By using the rest operator (…) in array destructuring, you can put all the remaining elements of an array in a new array.

Let us try to understand it with an example.

### **Example**

var colors = ["Violet", "Indigo", "Blue", "Green", "Yellow", "Orange", "Red

// destructuring assignment

var [a,b,...args] = colors;

console.log(a);

console.log(b);

console.log(args);

## **Swapping Variables**

The values of the two variables can be swapped in one destructuring expression. The array destructuring makes it easy to swap the values of variables without using any temporary variable.

### **Example**

var x = 100, y = 200;

[x, y] = [y, x];

console.log(x); // 200

console.log(y); // 100

# **ES6 Rest Parameter**

The rest parameter is introduced in ECMAScript 2015 or [ES6](https://www.javatpoint.com/es6), which improves the ability to handle parameters. The rest parameter allows us to represent an indefinite number of arguments as an array. By using the rest parameter, a function can be called with any number of arguments.

The rest parameter is prefixed with three dots (...). Although the syntax of the rest parameter is similar to the spread operator, it is entirely opposite from the spread operator. The rest parameter has to be the last argument because it is used to collect all of the remaining elements into an array.

### **Syntax**

function fun(a, b, ...theArgs) {

  // statements

}

### **Example**

function show(...args) {

  let sum = 0;

**for** (let i of args) {

      sum += i;

  }

  console.log("Sum = "+sum);

}

show(10, 20, 30);

# **ES6 Spread Operator:**

[ES6](https://www.javatpoint.com/es6) introduced a new operator referred to as a spread operator, which consists of three dots (...). It allows an iterable to expand in places where more than zero arguments are expected. It gives us the privilege to obtain the parameters from an array.

Spread operator syntax is similar to the rest parameter, but it is entirely opposite of it. Let's understand the syntax of the spread operator.

### **Syntax**

var variablename1 = [...value];

The three dots (...) in the above syntax are the spread operator, which targets the entire values in the particular variable.

Example:

const team1= ["user1","user2"];

const team2= ["user3"];

const team3= ["user4"];

const team4= ["user5", 'user6','user7'];

// const newTeam = team1.concat(team2);

// console.log(newTeam);

const myTeam = [...team1,...team2,...team3,...team4];

console.log(myTeam);

**When to use Spread and concat:**

Example:

const team1= ["user1","user2"];

const team2= ["user3"];

const team3= ["user4"];

const team4= ["user5", 'user6','user7'];

// const newTeam = team1.concat(team2);

// console.log(newTeam);

const myTeam = [...team1,...team2,...team3,...team4];

const newTeamMember = "newTeamMember";

console.log(team1.concat(newTeamMember));

console.log([...team1,...newTeamMember]);

**Ex2:copying array and adding to array:**

const productlist1 = ['phone', 'adopter'];

const newProduct = [...productlist1];

console.log(productlist1);

console.log(newProduct);

newProduct.push('camera');

console.log(productlist1);

console.log(newProduct);

ES6 Classes

Classes are an essential part of object-oriented programming (OOP). Classes are used to define the blueprint for real-world object modeling and organize the code into reusable and logical parts.

Before ES6, it was hard to create a class in [JavaScript](https://www.javatpoint.com/javascript-tutorial). But in ES6, we can create the class by using the **class** keyword. We can include classes in our code either by class expression or by using a class declaration.

A class definition can only include **constructors** and **functions**. These components are together called as the data members of a class. The classes contain **constructors** that allocates the memory to the objects of a class. Classes contain **functions** that are responsible for performing the actions to the objects.

Example:

**class** Student {

   constructor(name, age) {

**this**.n = name;

**this**.a = age;

   }

   stu() {

      console.log("The Name of the student is: ", **this**.n)

      console.log("The Age of the student is: ",**this**. a)

   }

}

var stuObj = **new** Student('Peter',20);

stuObj.stu();

In the above example, we have declared a class **Student**. The constructor of the class contains two arguments **name** and **age,** respectively. The keyword **'this'** refers to the current instance of the class. We can also say that the above constructor initializes two variables **'n' and 'a'** along with the parameter values passed to the constructor.

The function **stu()** in the class will print the values of **name** and **age**.

Example:

class person {

constructor(uname, uage){

this.name = uname;

this.age=uage

}

details() {

console.log(`The age of ${this.name} is ${this.age} yea`)

}

}

const p= new person('vijay', 30);

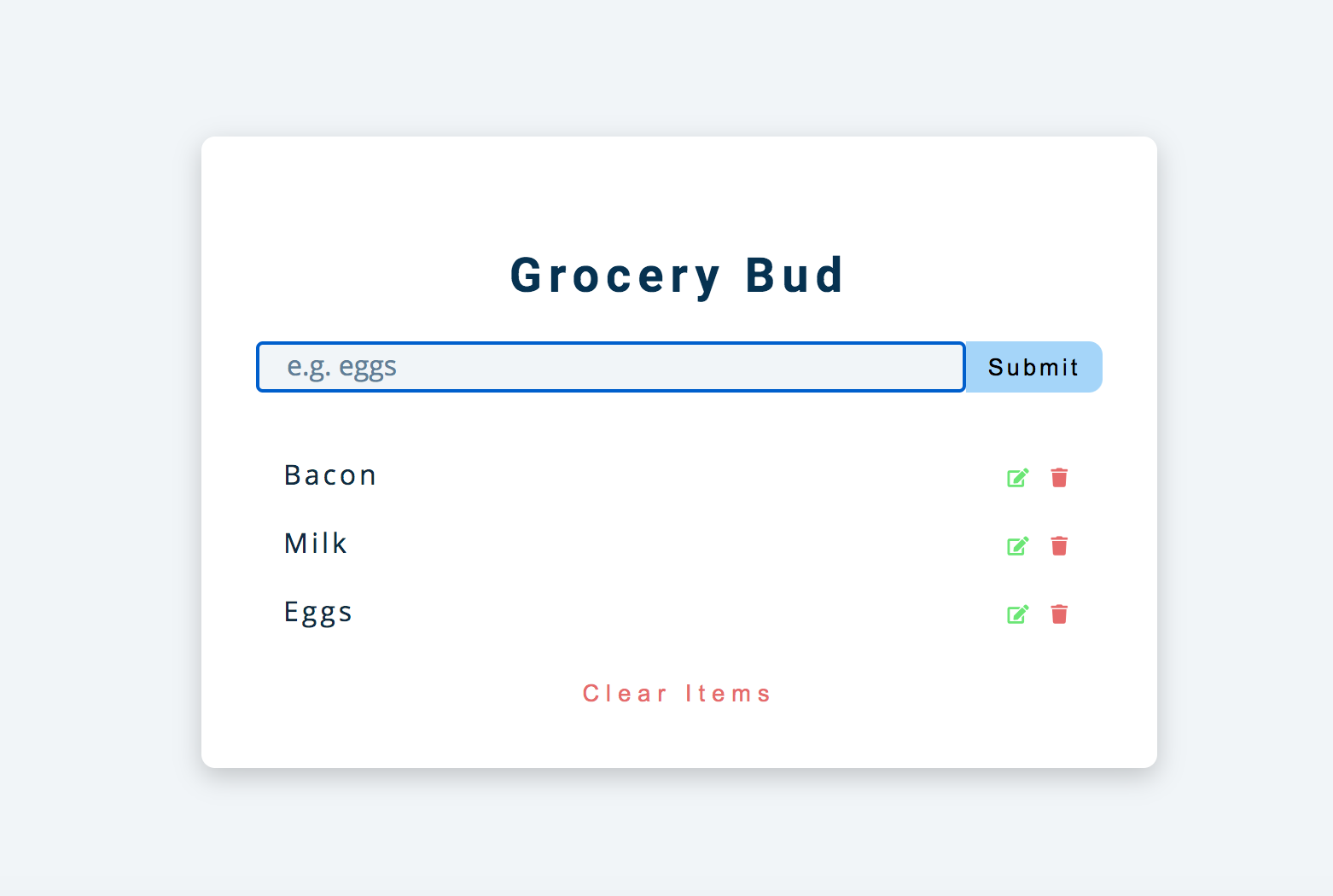
p.details();

const q= new person('Ajay', 35);

q.details();

Assignment:

### How to create a grocery list?



In this you will learn how to update and delete items from a grocery list and create a simple CRUD (Create, Read, Update, and Delete) application.

CRUD plays a very important role in developing full stack applications. Without it, you wouldn't be able to do things like edit or delete posts on your favourite social media platform.

# **ES6 Promises**

A Promise represents something that is eventually fulfilled. A Promise can either be rejected or resolved based on the operation outcome.

[ES6](https://www.javatpoint.com/es6) Promise is the easiest way to work with asynchronous programming in [JavaScript](https://www.javatpoint.com/javascript-tutorial). Asynchronous programming includes the running of processes individually from the main thread and notifies the main thread when it gets complete. Prior to the Promises, **Callbacks** were used to perform asynchronous programming.

## **Callback**

A Callback is a way to handle the function execution after the completion of the execution of another function.

A Callback would be helpful in working with events. In Callback, a function can be passed as a parameter to another function.

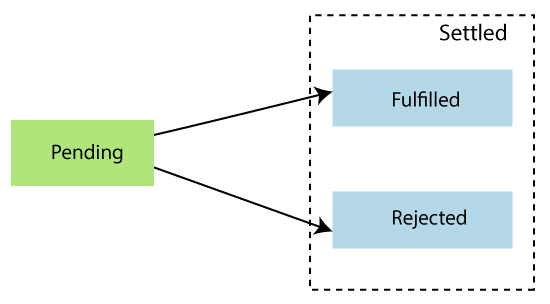
## **Why Promise required?**

A Callback is a great way when dealing with basic cases like minimal asynchronous operations. But when you are developing a web application that has a lot of code, then working with Callback will be messy. This excessive Callback nesting is often referred to as **Callback hell**.

To deal with such cases, we have to use **Promises** instead of **Callbacks**.

## **How Does Promise work?**

The Promise represents the completion of an **asynchronous operation**. It returns a single value based on the operation being **rejected** or **resolved**. There are mainly three stages of the Promise, which are shown below:



**Pending -** It is the initial state of each Promise. It represents that the result has not been computed yet.

**Fulfilled -** It means that the operation has completed.

**Rejected -** It represents a failure that occurs during computation.

Once a Promise is fulfilled or rejected, it will be immutable. The **Promise()** constructor takes two arguments that are **rejected** function and a **resolve** function. Based on the asynchronous operation, it returns either the first argument or second argument.

## **Creating a Promise**

In JavaScript, we can create a Promise by using the **Promise()** constructor.

**Syntax**

1. **const** Promise = **new** Promise((resolve,reject) => {....});

**Example**

let Promise = **new** Promise((resolve, reject)=>{

    let a = 3;

**if**(a==3){

        resolve('Success');

    }

**else**{

        reject('Failed');

    }

})

Promise.then((message)=>{

    console.log("It is then block. The message is: ?+ message)

}).**catch**((message)=>{

console.log("It is Catch block. The message is: ?+ message)

})

# **ES6 Validations**

Validation is the process of checking whether the information provided by the front-end user is correct or not as per the requirements. If the data provided by the client was incorrect or missing, then the server will have to send that data back to the client and request for resubmitting the form with the correct information.

Generally, the validation process is done at the server-side, which sends that validation information to the front-end user. This process wastes the execution time and user time.

[JavaScript](https://www.javatpoint.com/javascript-tutorial) gives us a way to validate the form's data before sending it to the server-side. Validation in forms generally performs two functions, which are as follows:

* **Basic validation-** It is a process to make sure that all of the mandatory fields are filled or not.
* **Data Format validation-** As its name implies, it is the process to check that entered data is correct or not. We have to include appropriate logic for testing the correctness of data.

## **Data Format Validation**

In data format validation, there is the checking of the entered data for the correct value. It validates already filled input fields that whether the filled information is correct or not.