**1. Write script for following methods of Array object:**

**a. Find()**

const array1 = [5, 12, 8, 130, 44, 38, 27];

const found = array1.find(element => element > 10);

console.log(found);

**b. Filter()**

const array1 = [5, 12, 8, 130, 44, 38, 27];

console.log(array1.filter((element) => {

return element >= 20 && element <= 40;

})

);

**c. Map()**

const map1 = new Map();

map1.set('a', 1);

map1.set('b', 2);

map1.set('c', 3);

console.log(map1);

console.log(map1.get('b'));

map1.set('c',25);

console.log(map1.get('c'));

**d. Reduce()**

const array2 = [7, 10, 6, 4, 8];

console.log(array2.reduce( (a, b) => a \* b, 5 ));

**e. Slice()**

const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];

const citrus = fruits.slice(1, 3);

console.log(citrus);

const fav = fruits.slice(-3,-1);

console.log(fav);

**f. Splice()**

const fruits1 = ["Banana", "Orange", "Apple", "Mango"];

fruits1.splice(2, 1, "Lemon", "Kiwi"); //add 2 elements at position 2 overwrites one element

console.log(fruits1);

const fruits2 = ["Banana", "Orange", "Apple", "Mango", "Kiwi"];

fruits2.splice(2, 1);//remove one element starting from index 2

console.log(fruits2);

**2. Study closure in JavaScript and write script for the same.**

Ans.

JavaScript has lexical scoping i.e. a variable declared outside is automatically available to the function.

Example:

let i = 1;

const f = ()=>{

console.log(i);

};

f(); //returns 1

console.dir(f)

[[Scopes]]: Scopes[2]

0: Script

f: ()=>{ console.log(i);}

i: 1

The word lexical refers to the fact that lexical scoping uses the location where a variable is declared within the source code to determine where that variable is available. Nested functions have access to variables declared in their outer scope.

But this cannot be always the case as function can be defined in one scope and then executed in some other scope. So, we use closure.

**Example:**

if(true){

let i = 1;

const f = ()=>{

console.log(i);

};

}

f(); //Error: Uncaught ReferenceError: f is not defined

A closure is the combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment). This environment consists of any local variables that were in-scope at the time the closure was created.

In other words, the function retains the variable it needs for execution. The value of the variable is preserved inside the function as a property called closure. So, that we can associate the data with the function and create as many copies of it as required.

**Example 1:**

var addTo = function(passed){

var add = function(inner){

return passed + inner;

};

return add;

};

//declare as many functions with value

var addThree = new addTo(3);

console.dir(addThree);

//addThree is defined with the passed value - so it has closure property

console.log("Sum is "+ addThree(10));

//addThree retains the value of passed and so only the inner value is sent

Output:

ƒ add(inner)

arguments: null

caller: null

length: 1

name: "add"

....

[[Scopes]]: Scopes[2]

0: Closure (addTo)

passed: 3

.....

**Output:**

Sum is 13

**Example 2:**

Now lets take the example further and add one more function.

var addTo = function(passed){

var add = function(inner){

var display = function(){

console.log(passed + inner);

}

return display;

};

return add;

};

var addThree = new addTo(3);

// Scope of addThree

// 0: Closure (addTo) {passed: 3}

// 1: Global

var thenAddTen = new addThree(10);

//scope of thenAddTen

// 0: Closure (add) {inner: 10}

// 1: Closure (addTo) {passed: 3}

// 2: Global

thenAddTen();

**Output:** 13

**3. Write a JavaScript function to merge two arrays and removes all duplicates elements**

**Test data :**

**var array1 = [1, 2, 3];**

**var array2 = [2, 30, 1];**

**console.log(merge\_array(array1, array2));**

**[3, 2, 30, 1]**

Ans.

var array1 = [1, 2, 3];

var array2 = [2, 30, 1];

function merge\_array(array1,array2){

var result = [];

array1.forEach(x => result.push(x));

array2.forEach(x => {if(result.indexOf(x) === -1){

result.push(x);

}});

return result;

}

console.log(merge\_array(array1, array2));

**4. Write a pattern that matches e-mail addresses.**

**The personal information part contains the following ASCII characters.**

** Uppercase (A-Z) and lowercase (a-z) English letters.**

** Digits (0-9).**

** Characters ! # $ % & ' \\* + - / = ? ^ \\_ ` { | } ~**

** Character . ( period, dot or full stop) provided that it is not the first or last character and it will not come one after the other.**

Ans.

/^[a-zA-Z0-9!#$%&'\\*+-/=?^\\_`{|}~]{1}[a-zA-Z0-9!#$%&'\\*+-/=?^\\_`{|}~.]+[a-zA-Z0-9!#$%&'\\*+-/=?^\\_`{|}~]{1}@[a-zA-Z0-9]{1}[a-zA-Z0-9.]+[a-zA-Z0-9]{1}.[a-zA-Z]{2,4}

**5. Write a JavaScript function to get the values of First and Last name of the following form:**

**Sample HTML file**

**<!DOCTYPE html>**

**<html><head>**

**<meta charset=utf-8 />**

**<title>Return first and last name from a form -**

**w3resource</title>**

**</head><body>**

**<form id="form1" onsubmit="getFormvalue()">**

**First name: <input type="text" name="fname" value="David"><br>**

**Last name: <input type="text" name="lname" value="Beckham"><br>**

**<input type="submit" value="Submit">**

**</form>**

**</body>**

**</html>**

Ans.

<html><head>

<meta charset=utf-8 />

<title>Return first and last name from a form -

w3resource</title>

</head>

<body>

<form id="form1" onsubmit="getFormvalue();">

First name: <input type="text" name="fname" value="David"><br>

Last name: <input type="text" name="lname" value="Beckham"><br>

<input type="submit" value="Submit">

</form>

<script>

function getFormvalue(){

var fname = document.getElementById("form1").childNodes[1].value;

var lname = document.getElementById("form1").childNodes[4].value;

document.write(fname+" "+lname);

}

</script>

</body>

</html>

**6. Write a JavaScript program to remove items from a dropdown list.**

Ans.

<html><head>

<meta charset=utf-8 />

<title>Remove items from a dropdown list</title>

</head><body><form>

<select id="colorSelect">

<option>Red</option>

<option>Green</option>

<option>White</option>

<option>Black</option>

</select>

<input type="button" onclick="removecolor()" value="Select and

Remove">

</form>

<script>

function removecolor(){

var option = document.getElementById("colorSelect");

option.remove(option.selectedIndex);

}

</script>

</body>

</html>

**7. Write a JavaScript program to highlight the bold words of the paragraph, on mouse over a**

**certain link.**

**Test data:**

**Link text is – Click here to highlight bold words.**

Ans.

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

</head>

<body>

<a href="http://google.com" id="link" onmouseover="myFunction()">Click here to highlight bold words.</a>

<script>

function myFunction(){

document.getElementById("link").style.fontWeight = "900";

}

</script>

</body>

</html>

**8. Write a JavaScript function to get the month name from a particular date.**

**Test Data :**

**console.log(month\_name(new Date("10/11/2009")));**

**console.log(month\_name(new Date("11/13/2014")));**

**Output :**

**"October"**

**"November"**

Ans.

function month\_name(date1){

var month = date1.getMonth();

switch(month){

case 0:

return 'January';

break;

case 1:

return 'February';

break;

case 2:

return 'March';

break;

case 3:

return 'April';

break;

case 4:

return 'May';

break;

case 5:

return 'June';

break;

case 6:

return 'July';

break;

case 7:

return 'August';

break;

case 8:

return 'September';

break;

case 9:

return 'October';

break;

case 10:

return 'November';

break;

case 11:

return 'December';

break;

default:

return 'Date is wrong!!'

}

}

**9. Write a JavaScript function to test whether a date is a weekend. Go to the editor**

**Note : Use standard Saturday/Sunday definition of a weekend.**

**Test Data :**

**console.log(is\_weekend('Nov 15, 2014'));**

**console.log(is\_weekend('Nov 16, 2014'));**

**console.log(is\_weekend('Nov 17, 2014'));**

**Output :**

**"weekend"**

**"weekend"**

**Undefined**

Ans.

function is\_weekend(date1){

var date = new Date(date1);

var day = date.getDay();

switch(day){

case 0:

case 6:

return 'weekend';

break;

case 1:

case 2:

case 3:

case 4:

case 5:

return ;

break;

default:

return 'Date is wrong!!'

}

}

**10. Write a JavaScript program to display the reading status (i.e. display book name, author name and reading status) of the following books.**

**var library = [**

**{**

**author: 'Bill Gates',**

**title: 'The Road Ahead',**

**readingStatus: true**

**},**

**{**

**author: 'Steve Jobs',**

**title: 'Walter Isaacson',**

**readingStatus: true**

**},**

**{**

**author: 'Suzanne Collins',**

**title: 'Mockingjay: The Final Book of The Hunger Games',**

**readingStatus: false**

**}];**

Ans.

var library = [

{

author: 'Bill Gates',

title: 'The Road Ahead',

readingStatus: true

},

{

author: 'Steve Jobs',

title: 'Walter Isaacson',

readingStatus: true

},

{

author: 'Suzanne Collins',

title: 'Mockingjay: The Final Book of The Hunger Games',

readingStatus: false

}];

for(x in library){

var i = parseInt(x);

if(library[i].readingStatus == true){

console.log(`I am reading the book ${library[i].title} by ${library[i].author}.`);

}

else{

console.log(`I have not read the book ${library[i].title} by ${library[i].author}.`);

}

}

**11. Write a JavaScript program to create a Clock.**

**Note: The output will come every second.**

**Expected Console Output:**

**"14:37:42"**

**"14:37:43"**

**"14:37:44"**

**"14:37:45**

Ans

setInterval(clock, 1000);

function clock() {

let d = new Date();

let hh = d.getHours();

let mm = d.getMinutes();

let ss = d.getSeconds();

hh = (hh < 10) ? "0" + hh : hh;

mm = (mm < 10) ? "0" + mm : mm;

ss = (ss < 10) ? "0" + ss : ss;

console.log(hh + ":" + mm + ":" + ss + " ");

}

**12. Write a JavaScript program to sort an array of JavaScript objects.**

**Sample Object:**

**var library = [**

**{**

**title: 'The Road Ahead',**

**author: 'Bill Gates',**

**libraryID: 1254**

**},**

**{**

**title: 'Walter Isaacson',**

**author: 'Steve Jobs',**

**libraryID: 4264**

**},**

**{**

**title: 'Mockingjay: The Final Book of The Hunger Games',**

**author: 'Suzanne Collins',**

**libraryID: 3245**

**}];**

Ans.

var library = [

{

title: 'The Road Ahead',

author: 'Bill Gates',

libraryID: 1254

},

{

title: 'Walter Isaacson',

author: 'Steve Jobs',

libraryID: 4264

},

{

title: 'Mockingjay: The Final Book of The Hunger Games',

author: 'Suzanne Collins',

libraryID: 3245

}];

library.sort(function (a, b) {

return a.libraryID - b.libraryID;

});

console.log("Sorted by library id -- ");

console.log(library);

// sort by title

library.sort(function(a, b) {

var titleA = a.title.toUpperCase();

var titleB = b.title.toUpperCase();

if (titleA < titleB) {

return -1;

}

if (titleA > titleB) {

return 1;

}

return 0;

});

console.log("Sorted by library title -- ");

console.log(library);

// sort by author

library.sort(function(a, b) {

var authorA = a.author.toLowerCase();

var authorB = b.author.toLowerCase();

if (authorA < authorB) {

return -1;

}

if (authorA > authorB) {

return 1;

}

return 0;

});

console.log("Sorted by library author -- ");

console.log(library);

**13. Study Promises in JavaScript and provide script for the same.**

Ans.

A Promise is a JavaScript object that links producing code and consuming code. "Producing code" is code that can take some time. "Consuming code" is code that must wait for the result.

**Promise Syntax**

let myPromise = new Promise(function(myResolve, myReject) {

// "Producing Code" (May take some time)

myResolve(); // when successful

myReject(); // when error

});

// "Consuming Code" (Must wait for a fulfilled Promise)

myPromise.then(

function(value) { /\_ code if successful \_/ },

function(error) { /\_ code if some error \_/ }

);

When the producing code obtains the result, it should call one of the two callbacks:

Result --------------> Call

Success --------------> myResolve(result value)

Error --------------> myReject(error object)

The Promise object supports two properties: state and result.

* While a Promise object is "pending" (working), the result is undefined.
* When a Promise object is "fulfilled", the result is a value.
* When a Promise object is "rejected", the result is an error object.

The methods promise.then(), promise.catch(), and promise.finally() are used to associate further action with a promise that becomes settled.

The .then() method takes up to two arguments; the first argument is a callback function for the resolved case of the promise, and the second argument is a callback function for the rejected case.

A .catch() is really just a .then() without a slot for a callback function for the case when the promise is resolved.

**Example:**

let myPromise = new Promise(function(myResolve, myReject) {

setTimeout(function() { myResolve("Hello there "); }, 3000);

console.log("Hi !!");

});

myPromise.then(value => { return value + 'again'; })

.then(value => { console.log(value) })

.catch(err => { console.log(err) });

**Output:**

Hi !!

Hello there again