**JAVASCRIPT NOTES**

***CLASS 2***

***—-------------------***

// ASYNCHRONOUS / SYNCHORONOUS ??

// Is JavaScript Sync/Async????

// JS is Synchronous by default.

// SINGLE THREADED / MULTI THREADED ??

// Is JS Single Threaded / Multi Threaded??

// JS is Single Threaded.

/\*

CASE 1: ASYNCHRONOUS

She can keep the glass, go and do some other work and

when I am done drinking milk,

she can come and collect the glass

CASE 2: SYNCHORONOUS

She will wait here until i finish the milk.

Then she will take the glass back,

\*/

// Execution flow / context

// Call Stack

// Event Loops

// Execution Context

// JS executes any statement line by line

/\*

var a = 10;

console.log(a);

console.log("hello");

console.log(b);

var name="vikas";

console.log(c);

function print(){

console.log("welcome")

}

function add(num){

return num+5;

}

var c = add(7);

\*/

// Execution Context

// Memory --> All the variable and the functions declarations

// a: 10

//name: "vikas"

// add: function(){....}

// print: function(){

// console.log("welcome")

//}

// Code --> The part where all calculations happen

// JavaScript Call Stack

/\*

// whenever it is basic statement, it executes line by line.

console.log("line1");

console.log("line2");

console.log("line3");

// in any variable declaration/variable assignment,

// it executes line by line

var a = 10;//10

a = a+5;//15

a = 30;//30

a = a-10;//20

console.log(a);//20

\*/

// Stack -- Last In First Out

console.log("line 0"); // executed 1st

main(); //executed 2nd

console.log("line 4"); //executed 7th

function sub(){

console.log("line3");//executed 5th

}

function main(){

console.log("line1"); //excecuted 3rd

sub();//executed 4th

console.log("line2");//executed 6th

}

// line0

// line1

// line3

// line2

// line4

—-------

// whenever you pass a function as a argument/parameter

//to another function, it is called a

//callback function

console.log("1"); //1 3 5 9 2 8 7 6 4

main(); // 1 3 5 9 2 7 8 6 4

console.log("2"); // 1 3 5 7 8 9 2 6 4

// 1 3 7 8 5 2 6 9 4

function main(){

console.log("3");

setTimeout(()=> {

console.log("4");

}, 1000)

console.log("5");

setTimeout(()=> {

console.log("7");

}, 0)

setTimeout(()=> {

console.log("6");

}, 100)

setTimeout(()=> {

console.log("8");

},0)

console.log("9");

}

// 1 3 5 2 4 6

// 1 3 5 6 4 2

// 1 3 4 5 6 2

// 1 3 5 2 6 4 -- right

// 1 3 6 4 5 2

// 1 3 2 5 4 6

***—-------------------------—-------------------------—-------------------------—-------------------------***

***DAY 3 :-***

/\* function is a block if code

--> Makes the code cleaner

--> Improves reusability of the code

--> Saves development time

--> Functions should be created in a way

that it should do a single task

--> Functions can recieve an input/argument

--> Functions can give an output

How to create functions :-

1. function declaration

\*/

function sum(num1, num2){

return num1 + num2;

}

console.log(sum(4,5));

console.log(typeof sum);

/\*

2. function expressions

\*/

// --- function expression

var multiply = function (num1, num2){

return num1 \* num2;

}

console.log(multiply(4,5));

console.log(typeof multiply);

// ---- function expression with arrow notation

var subtract = (num1, num2) => {

return num1- num2;

}

console.log(subtract(4,5));

console.log(typeof subtract);

/\*

Anonymous function : a func without a name

function (num1, num2){

return num1 \* num2;

}

\*/

—

var x = sum(4,5);

console.log(x);

function sum(num1, num2){

return num1 + num2;

}

---

function sum(num1, num2){

return num1 + num2;

}

var x;

x = sum(4,5)

console.log(x);

//- functional hoisiting

// only function declaration gets hoisted

//-------

var z;

console.log(z);

z = 10;

console.log(z);

// ----

console.log(subtract(4,5));

var subtract = (num1, num2) => {

return num1- num2;

}

/\*console.log(multiply(4,5));

var multiply = function (num1, num2){

return num1 \* num2;

}

----

var multiply;

console.log(multiply(4,5));

multiply = function (num1, num2){

return num1 \* num2;

}

\*/

—-

/\*

Scope of a variable in JS.

--> var has functional scope

--> let, const have block level scope

--> Lexical scope

\*/

var x = 10;

main();

function main(){

console.log(x);//10

var y = 20;

sub();

function sub(){

var x = 40;

console.log(x);//10

console.log(y);//20

console.log(z);//undefined

}

}

// Lexical scope of variables in JS -->

//Every function has it's own variables in its excution context

// if a variable is not present in its execution context,

// it will keep check in its parent's EC until it reaches

// the global EC

//--------

/\*

var sum = () => {

console.log("Hello");//"hello

}

console.log(typeof sum);//"function"

console.log(sum);// function body

console.log(sum());/// hello, undefined

console.log(typeof sum());// hello, "undefined"

\*/

—

// Closures -- The inner/child function has the access to

// the scope of the outer/parent function even when

// the outer function is already executed

// GEC, OUTER scope, inner scope

function outer (){

var firstname = "Vikas";

var inner = () => {

console.log(firstname);

}

return inner;

}

var abcd = outer();//line1

console.log(abcd);//line2

abcd();//line3

—---

function main(){

var counter = 0;

var increase = () => {

counter++;

}

var decrease = () => {

counter--;

}

var display = () => {

console.log('The value of counter is--' + counter);

}

return {

increase: increase,

decrease: decrease,

display: display

}

}

var data = main();

data.display();//0

data.increase();

data.increase();

data.display();//2

data.increase();

data.increase();

data.increase();

data.increase();

data.decrease();

data.decrease();

data.decrease();

data.display();//

—---

function main () {

// 0\* , 1\*, 2\*, 3\*,4\*,5\*,6\*,7\*,8\*,9\*,10?

for (var index = 0; index < 10; index++){

(function(val){

setTimeout(()=> {

console.log(val);

}, 100)

})(index)

}

//console.log("index is ----"+ index);

}

main();

// 0 .. 9 after 1 sec

// 10 times 9 after 1. sec

// 10 times 10

index = 10

cbArr = [()=> {

console.log(index);

}, ()=> {

console.log(index);

}, ()=> {

console.log(index);

},()=> {

console.log(index);

},()=> {

console.log(index);

},()=> {

console.log(index);

},()=> {

console.log(index);

},()=> {

console.log(index);

},()=> {

console.log(index);

},()=> {

console.log(index);

}]

—-

(function (num) {

console.log(num);

})(5);

/// iify

// immediately invoke functions

—-

—---------------------

DAY 5

// not declared

// not defined / undefined

var x; // declared, but not defined

console.log(y);// undeclared

console.log(typeof x);

if (typeof x != "undefined"){

console.log(x);

}

//console.log(x);

/\*

if (x is defined){

print x;

}

\*/

/\*

if (!x){

console.log("x is not defined")

}

\*/

/\*

if (x !== undefined){

console.log(x);

}

\*/

var x;

console.log(typeof x); // "undefined"

console.log(typeof y); // "undefined"

console.log(x); //undefined

try{

console.log(y);

}

catch (error){

console.log("Hello, y is not defined.")

}

finally{

console.log("finallly is always running")

}

console.log("Welcome")

/\*

try{

fetch("......")

.then(res => {return res.json();})

.catch()

}

catch(error){

al

\*/

try{

if(10 == 10){//true/false

console.log("hello");

}else{

console.log(y);

}

}

catch(error){

console.log(error.message);

}

// server call

// Race condition

// undeclared cases

// internet connection

// server is blocked

// server has been moved

// api location hss been moved

// url is wrong

// CORS

—----

—---------

const promise1 = new Promise ((resolve, reject) => {

const x = 21;

setTimeout(()=> {

if (x % 2 !=0){

resolve();

}else{

reject();

}

}, 1000);

})

const promise2 = new Promise ((resolve, reject) => {

const x = 21;

setTimeout(()=> {

if (x % 2 !=0){

resolve();

}else{

reject();

}

}, 5000);

})

Promise.all([promise1, promise2]).then(values => {

console.log("All done"); // [3, 1337, "foo"]

}, ()=> {

console.log("All done again");

});

promise1.then(()=> {

console.log("success promise1");

}, ()=> {

console.log("error promise1");

})

promise2.then(()=> {

console.log("success promise2");

}, ()=> {

console.log("error promise2");

})

// Promise.all

//"error promise1"

//"success promise2"

// promise1 -- success promise1, error promise1

// promise2 -- success promise2, error promise2

// both are done --> All done

/\*

const x = new Promise ((resolve, reject) => {

setTimeout(()=> {

resolve();

}, 1000)

})

x.then(()=> {

console.log("success");

}, ()=> {

console.log("error");

})

\*/

var arr = [1,2,3,4,5];

// [1,4,9,16,25]

// Map function --

// applies on an array and takes a callback, applies the callback

//on every element in the array and returns a new array

const square = (element) => {

return element\*element;

}

var newArr = arr.map(square)

console.log(newArr);

// Filter -- works on an array, takes a callback function

// and returns a new array with elements which fulfill the

//callback function

// [1,3,5];

var filterArr = arr.filter(element => {

if (element %2 !=0){

return true;

}else{

return false;

}

})

console.log(filterArr);

// Reduce -- works on an aray, takes a callback function

// and returns a accumulator/reduced value.

var reducedValue = arr.reduce((element,sum) => {

return sum = element + sum;

})

console.log(reducedValue);

—------------

// polyphil for MAP function

// Array --- map, filter, reduce, join

// Integer -- parseInt

// Boolean

// String

// Number

//any new array is part of the Array prototype chain

// An object in JS through which you can inherit any other

// object/class/function

// this is a special object in JS which points to

// the reference from where it is called.

Array.prototype.myMap = function (cbFn) {

const newArr = [];

this.forEach(element => {

const res = cbFn(element);

newArr.push(res);

})

return newArr;

}

const square = (element) => {

return element\*element;

}

const arr = [1,2,3,4,5];

const result = arr.myMap(square)

//const result = myMap(arr, square)

// var newArr = arr.map(square)

console.log(result);

—-------

//console.log(this);

// Window object is the super scope in JS

// ALl global variables, global functions are a

//part of window object

// var x = 10;

//console.log(x);

//console.log(window.x);

// "this" always gives the context of the place it is called

// from

/\*

function main(){

console.log(this);

}

main();

\*/

//--

/\*

function main(){

function sub(){

console.log(this);

}

return sub;

}

var subFn = main();

subFn();

\*/

//------

var obj = {

name: "Vikas",

subObj: {

adrress: "city",

display2: function(){

console.log(this);

}

},

display: function(){

console.log(this);

console.log(this.name);

}

}

//obj.display();

obj.subObj.display2();

—

25th April

// TIme complexity

// Space complexity

// Searching

//[10,4,9,30,7]

// Write a program if 30 is present in the array,

const arr = [10,4,9,30,7];

function checkExists (arr, value){

let isPresent = false;

for (let index = 0; index < arr.length; index++){

if (arr[index] == value){

isPresent = true;

break;

}

}

if(isPresent){

console.log("Present");

}else{

console.log("Not Present");

}

}

checkExists(arr,30);

checkExists(arr,10);

checkExists(arr,0);

checkExists(arr,100);

—

Binary search

const arr = [4,7, 9,10, 30];

checkExistsBinarySearch (arr,40);

checkExistsBinarySearch (arr,10);

function checkExistsBinarySearch(arr,value){

let left = 0;

let right = arr.length - 1;

let isPresent = false;

while(right>=left){

let mid = Math.floor((left + right) / 2);

if (arr[mid] == value){

isPresent = true;

break;

}

if (value > arr[mid]){

// check in the right sub half

left = mid +1;

}else{

// check in the left sub half

right = mid -1;

}

}

if (isPresent){

console.log("present");

}else{

console.log("not present");

}

}

—--

26/04/2022

–

const arr = [9, 11, 15, 7, 5, 3];

const selectionSort = (arr) => {

for (let n1 = 0; n1 < arr.length-1; n1++){

for (let n2 = n1+1; n2 < arr.length; n2++){

// if n1 elem greater than n2 element, swap them

if (arr[n1] > arr[n2]){

const temp = arr[n1];

arr[n1] = arr[n2];

arr[n2] = temp;

}

}

}

console.log(arr);

}

selectionSort(arr);

—

const arr = [9, 11, 15, 7, 5, 3];

const bubbleSort = (arr) => {

for (let n1 = 0; n1 < arr.length; n1++){

for (let n2 = 0; n2 < arr.length-n1-1; n2++){

// if x elem greater than x+1 element, swap them

if (arr[n2] > arr[n2+1]){

const temp = arr[n2];

arr[n2] = arr[n2+1];

arr[n2+1] = temp;

}

}

}

console.log(arr);

}

bubbleSort(arr);

–

// Given a sorted array containing integers(1-n),

// find first missing positive integer.

const arr = [1,2,3,4,5,6,7,9,10];

const firstMissingPositive = (arr) => {

let missingNumber = null;

for (let index = 0; index< arr.length; index++){

if (arr[index] != index+1){

missingNumber = index+1;

break;

}

}

console.log(missingNumber);

}

// O(n)

firstMissingPositive(arr);

—

// Given an unsorted array containing integers(1-n),

// find first missing positive integer.

const arr = [10,3,4,9,5,2,8,6,1]; //47

const firstMissingPositiveForUnsorted = (arr, n) => {

const expectedSum = n\*(n+1) /2;

let actualSum =0;

for (let index =0; index<arr.length; index++){

actualSum = actualSum + arr[index];

}

console.log(expectedSum - actualSum);

}

// sum of nos from 1 to 10 -- 1 + 2 + 3 + 4 + 5

// ===. n (n+1)/2

// 5 \* 6 /2. == 15

// 10 \* 11 /2 == 55

firstMissingPositiveForUnsorted(arr,10);

// 1. Sort the array -- //quick/merge -- O(n log n)

// 2. Then apply previous logic -- O (n)

// --- O(n log n)

//// O(n)

—-

27 April

// Find first missing positive number in unsorted array

//(not 1-n).

// All elements are unique.

const arr = [-3, 5, 7, 1, 4, 0, -9]; // 2

// [-3, 5, 7, 1, 4, 2, 0, -9]; // 3

// [-3, 5, 3, 7, 1, 4, 2, 0, -9]; // 6

const firstMissingPositive = (arr) => {

}

firstMissingPositive(arr);

// [-3, 5, 3, 7, 1, 4, 2, 0, -9]

–

const arr = [-3, 5, 3, 7, 1, 4, 2, 0, 9];

const getHighest = (arr) => {

let highest = Number.NEGATIVE\_INFINITY;

for (let index = 0; index<arr.length; index++ ){

if(arr[index] > highest){

highest = arr[index];

}

}

console.log(highest);

}

getHighest(arr);

//[-3, 5, 3, 7, 1, 4, 2, 0, -9]; => 7

// [-3,-9,-1] ==> -1

// [0,-1,-3] ==> 0

// [] --> null

// [1] --> 1

—---

// [3,19,7, 4, 16] => [16, 4 , 7, 19, 3]

// [10, 20, 30, 40, 50, 60] => [60,50,40,30,20,10]

const arr = [10, 20, 30, 40, 50, 60];

const reverseArr = (arr) => {

let left = 0, right = arr.length-1;

while (left < right){

const temp = arr[left];

arr[left] = arr[right];

arr[right] = temp;

left++;

right--;

}

console.log(arr);

}

reverseArr(arr);

—--

// [0,1,1,0,0,0,1,1,0] => [0,0,0,0,0,1,1,1,1]

const arr = [0,1,1,0,0,0,1,1,0];

// sort an array having only 0-1-2 --> [0,0,1,2,2,1,1,0,2,1]

var sortZeroAndOnes = (arr) => {

let low = 0, high = arr.length - 1;

while (low <= high){

if (arr[low] ==0){

low++;

}else{

const temp = arr[low];

arr[low] = arr[high];

arr[high] = temp;

high--;

}

}

console.log(arr);

}

sortZeroAndOnes(arr);

—

—-

28 april class

const arr1 = [4, 19, 3, 5, 6]

const arr2 = [7, 13, 5, 23,6];

// find the intersection of the two arrays

//output => [5,6]

const findIntersection = (arr1, arr2) => {

const newArr = [];

const arr1Map = {};

for (let index=0; index<arr1.length; index++){

const item = arr1[index];

arr1Map[item] = 1;

}

for (let index2=0; index2<arr2.length; index2++){

const item = arr2[index2];

if (arr1Map[item]){

newArr.push(item);

}

}

console.log(newArr);

}

findIntersection(arr1, arr2);

—----

console.log(Boolean(0));

console.log(Boolean(1));

console.log(Boolean(100));

console.log(Boolean(-100));

console.log(Boolean(undefined));

console.log(Boolean(null));

console.log(Boolean([]));

console.log(Boolean({}));

console.log(Boolean(""));

console.log(Boolean("Hello. welcome"));

console.log(Boolean(" "));

console.log(Boolean("x"));

—-

// Merge two sorted Arrays to form a new merged Array

// 3,5,6,8,9,4,6,7,10,11

const arr1 = [3, 4, 6, 8,9];

const arr2 = [5, 6, 7, 10,11];

// [3,4,5,6,6,7,8,9,10,11] -- with duplicates

// [3,4,5,6,7,8,9,10,11]-- without duplicates

const mergeArrays = (arr1, arr2) => {

const newArr = [];

let p1 = 0;// pointer for arr1

let p2 = 0;// pointer for arr2

//[1,2,3,4,5] [1,2,3,4,5] == n + m-1 = 2n

while(p1 < arr1.length && p2 < arr2.length){

if (arr1[p1] < arr2[p2]){

newArr.push(arr1[p1]);

p1++;

}else{

newArr.push(arr2[p2]);

p2++;

}

}

if(p1 < arr1.length){

while (p1<arr.length){

newArr.push(arr1[p1]);

p1++;

}

}

if(p2 < arr2.length){

while (p2 < arr2.length){

newArr.push(arr2[p2]);

p2++;

}

}

console.log(newArr); // O(n+m) ~~> O(n)

}

mergeArrays(arr1,arr2);

—-

// [1,7,10,3, 25, 9]

// part 1 - dividing in 2 halves in every iteration -- O(log n)

// [1,7,10] [3,25,9]

// [1,7] [10] [3,25] [9]

// [1] [7] [10] [3] [25] [9]

// part 2---- merging O(n)

// [1,7] [3,10] [9,25]

// [1,3,7,10] [9,25]

// [1,3,7,9,10,25]

//-- O(n log n)

—----

// Rotate an array "k" times clockwise

const arr = [3,7,10, 6, 4, 21];

const rotateKTimes = (arr,k) => {

const newArr = [];

const effectiveCount = k % arr.length;

for (let index = effectiveCount ; index<arr.length; index++){

newArr.push(arr[index]);

}

for (let index = 0 ; index<effectiveCount; index++){

newArr.push(arr[index]);

}

console.log(newArr);

}

rotateKTimes(arr,2);

rotateKTimes(arr,8);

rotateKTimes(arr,14);

// time -- O(N)

// space -- O(N)

// rotate it 2 times clockwise

// 3,7,10, 6, 4, 21

// 7, 10, 6, 4, 21, 3 -- 1 rotation

// 10, 6, 4, 21, 3, 7 -- 2nd rotation

// 6, 4, 21, 3, 7, 10 -- 3rd rotation

// 4, 21, 3, 7, 10, 6 -- 4th rotation

// 21, 3, 7, 10, 6, 4 -- 5th rotation

// 3, 7, 10, 6, 4, 21 -- 6th rotation

// 7, 10, 6, 4, 21, 3 -- 7th rotation

// 3,7,10, 6, 4, 21

// it should be rotated 2 times --> // 10, 6, 4, 21, 3, 7 -- 2nd rotation

—--

// Rotate an array "k" times clockwise

const arr = [3,7,10, 6, 4, 21];

const rotateKTimes = (arr,k) => {

const newArr = [];

const effectiveCount = k % arr.length;

for (let index = effectiveCount ; index<arr.length; index++){

newArr.push(arr[index]);

}

for (let index = 0 ; index<effectiveCount; index++){

newArr.push(arr[index]);

}

console.log(newArr);

}

rotateKTimes(arr,2);

rotateKTimes(arr,8);

rotateKTimes(arr,14);

// time -- O(N)

// space -- O(N)

// rotate it 2 times clockwise

// 3,7,10, 6, 4, 21

// 7, 10, 6, 4, 21, 3 -- 1 rotation

// 10, 6, 4, 21, 3, 7 -- 2nd rotation

// 6, 4, 21, 3, 7, 10 -- 3rd rotation

// 4, 21, 3, 7, 10, 6 -- 4th rotation

// 21, 3, 7, 10, 6, 4 -- 5th rotation

// 3, 7, 10, 6, 4, 21 -- 6th rotation

// 7, 10, 6, 4, 21, 3 -- 7th rotation

// 3,7,10, 6, 4, 21

// it should be rotated 2 times --> // 10, 6, 4, 21, 3, 7 -- 2nd rotation

—--

05 May

—

​​const arr = [13, 7, 11, 19, 5, 7];

// given an array of unique prime numbers,

// where any one number is repeated twice.

// find the repeated number

const getRepeatedPrime = (arr) => {

let product = 1;

//for (let index = 0; index < arr.length; index++){

// product = product \* arr[index];

// }

console.log(product);

for (let index = 0; index < arr.length; index++){

product = product \* arr[index];

if (product % (arr[index] \* arr[index]) == 0){

console.log(arr[index]);

break;

}

}

}

getRepeatedPrime(arr);

—-------------

—-------------------------

/\*

Input : arr[] = {10, 20, 10, 5, 15}

Output : prefixSum[] = {10, 30, 40, 45, 60}

Given an array arr[] of size n,

its prefix sum array is another array

prefixSum[] of the same size, such that the

value of prefixSum[i] is arr[0] + arr[1] + arr[2] … arr[i].

\*/

const arr = [10, 20, 10, 5, 15];

const prefixSum = (arr) => {

let previousSum = 0;

for (let index = 0; index<arr.length; index++){

arr[index] = previousSum + arr[index];

previousSum = arr[index];

}

console.log(arr);

}

prefixSum(arr);

—----------------

—--------------------------------

// find the pivot element

/\*

The pivot index is the index

where the sum of all the numbers strictly to the left of the index

is equal to the sum of all the numbers strictly to the index's right.

If the index is on the left edge of the array, then the left sum is 0 because there are no elements to the left. This also applies to the right edge of the array.

Return the leftmost pivot index. If no such index exists, return -1.

\*/

/\*

[1,7,3,6,5,6]

for 1 --> left Sum = 0, right sum = 27

for 7 --> leftsum = 1 rightsum = 20

for 3 --> leftSUm = 8. righsum = 17

for 6 --> leftSum = 11 rightsum = 11 --> answer ---> 3 the index of 6

\*/

const arr1 = [1,7,3,6,5,6];

const arr2 = [1,2,3];

const arr3 = [2,1,-1];

const pivotIndex = arr => {

let finalIndex = -1;

let leftSum = 0;

let totalSum = 0;

for (let index = 0; index < arr.length; index++){

totalSum = totalSum + arr[index];

}

for (let index = 0; index < arr.length; index++){

const rightSum = totalSum - arr[index] - leftSum;

if(leftSum == rightSum){

finalIndex = index;

break;

}else{

leftSum = leftSum + arr[index];

}

}

console.log(finalIndex);

}

pivotIndex(arr1);

pivotIndex(arr2);

pivotIndex(arr3);

/\*

[1,2,3]

for 1 -- letsum = 0 rightSUm = 5

for 2 --- leftsum = 1. righsum = 3

for 3 -- leftsum = 3. righrsum = 0

if no pivot index , return -1

\*/

/\*

[2,1,-1]

for 2, leftsum = 0. rightsum = 0 ; pivot Index ---> 0

\*/

—--------

—--------------

5th may 2022

–

const message = "hello, my name is Vikas!"

// always written within double quotes

// it can contain any character - alhpabet, number, symbol, space

console.log(2 + "2"); //22

console.log(2 - "2"); //0

console.log(2 - "2bcd"); //0

console.log("251");

console.log(Number("251"));

console.log(Number("abcd")); // NaN --> Not a number

console.log(typeof NaN); // "number"

const arr = [1,2,3,4,5];

arr[2] = 7;

console.log(arr);//1,2,7,4,5

const name = "Vikas";

console.log(name.length);

console.log(name[0]);

console.log(name[3]);

name[2] = "x";

console.log(name); //

—-----

const str = "I love my India";

// q1. Reverse the whole string

// ---> "aidnI ym evol I"

// q2. Reverse the words in the string

// --> "India my love I"

// q3. Reverse individual words

// "I evol ym aidnI"

//console.log(str.split(" "));

//console.log(str.split("o"));

//console.log(str.split("I"));

//console.log(str.split(""));

function reverseString1(str){

const arr = str.split(""); //--> O(n)

const reverseArr = arr.reverse(); //--> O(n)

console.log(reverseArr.join("")); // --> O(n)

}

function reverseStringWithWords1(str){

const arr = str.split(" "); //--> O(n)

const reverseArr = arr.reverse(); //--> O(n)

console.log(reverseArr.join(" ")); // --> O(n)

// str.split(" ").reverse().join(" ");

}

function reverseStringWithWords2(str){

const arr = str.split(" ");

let newStr = "";

for (let index = arr.length-1; index>=0; index--){

newStr = newStr + arr[index];

if(index!=0){

newStr = newStr + " ";

}

}

console.log(newStr);

}

function reverseString2(str){

let newStr = "";

for (let index = str.length-1; index>=0; index--){

newStr = newStr + str[index];//"" + "a" ==> "a" + "i" ==> "ai" ==> "ai" + "d" ==> "aid"

}

console.log(newStr);

}

reverseString1(str);

reverseString2(str);

reverseStringWithWords1(str);

reverseStringWithWords2(str);

—-----

// Listen --> Silent

// abcde --> ebcda --> abdec --> bdcea

//Q. you are given two words -->

// check whether they are anagrams or not?

// Q. given a string -->

// how will you check if it contains duplicates?

// Q. In any string , print the character that occurs

// the maximum number of times

const str = "abcdeffcdghdeee";

const maxChar = (str) => {

const freqObj = {};

for (let index = 0; index < str.length; index++){

const char = str[index];

if(!freqObj[char]){

freqObj[char] = 1;

}else{

const freq = freqObj[char];

freqObj[char] = freq+1;

}

}

console.log(freqObj);

let MAX\_FREQ = 0;

let MAX\_FREQ\_LETTER = "";

const keysArr = Object.keys(freqObj);

for (let index = 0; index< keysArr.length; index++){

const key = keysArr[index];

const freq = freqObj[key];

if(freq > MAX\_FREQ){

MAX\_FREQ = freq;

MAX\_FREQ\_LETTER = key;

}

}

console.log(MAX\_FREQ\_LETTER);

}

maxChar(str);

—

// Listen --> Silent

// abcde --> ebcda --> abdec --> bdcea

//Q. you are given two words -->

// check whether they are anagrams or not?

// Q. given a string -->

// how will you check if it contains duplicates?

// Q. In any string , print the character that occurs

// the maximum number of times

// Q. In any string , print top "k" characters that occur

//the maximum number of times

const str = "abcdeffcdghdeee";

const maxChar = (str) => {

const freqObj = {};

for (let index = 0; index < str.length; index++){

const char = str[index];

if(!freqObj[char]){

freqObj[char] = 1;

}else{

const freq = freqObj[char];

freqObj[char] = freq+1;

}

}

console.log(freqObj);

let MAX\_FREQ = 0;

let MAX\_FREQ\_LETTER = "";

const keysArr = Object.keys(freqObj);

for (let index = 0; index< keysArr.length; index++){

const key = keysArr[index];

const freq = freqObj[key];

if(freq > MAX\_FREQ){

MAX\_FREQ = freq;

MAX\_FREQ\_LETTER = key;

}

}

console.log(MAX\_FREQ\_LETTER);

}

maxChar(str);

—

6th May

–

// Q.1 In any string , print top "k" characters that occur

//the maximum number of times

//Q.2 -------

const nums = [-2,1,-3,4,-1,2,1,-5,4]

const maximumSum = (arr) => {

let MAX = Number.NEGATIVE\_INFINITY;

let currentSum = 0;

for (let index = 0; index<arr.length; index++){

currentSum = currentSum + arr[index];

if(MAX < currentSum){

MAX = currentSum;

}

if (currentSum < 0){

currentSum = 0;

}

}

console.log(MAX);

}

maximumSum(nums);

// Given an integer array nums,

//Q.2.a find the maximum sum.

//Q.2.b find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

// A subarray is a contiguous part of an array.

// [-4, -7, -9 , -1] // -1

—-

//arrays

// number

// boolean

// string

// null

// undefined

// Object

// In JS, an Object has a type Object,

//an array also has type Object

// Null also has type object

// AN object in JS is a data type that essentially

// points to a memory location

// and can store a key/value pair like a map

/\*

// Pass by value

let a = 23;

let b = a;

b = 26;

console.log(b);//26

console.log(a);//23

let x = {"name": "vikas", age : 29};

let y = x;

y.name = "Anul";

console.log(y);//

console.log(x);//

let i = {"name": "vikas", age : 29};

let j = {...i};

j.name = "Anul";

console.log(j);//

console.log(i);//

const getKey = () => {

return "isIndian";

}

const human = {

name: "vikas",

age: 29,

isIndian: true,

displayName: () => {

console.log(this.name);

}

};

const x = getKey();//"isIndian"

//console.log(human.x);//undefined

// console.log(human['x']);

console.log(human[x]);

delete human.name;

delete [human['name']];

delete [human[x]];

\

// DOT operator

console.log(human.age);//29

console.log(human.name);//"Vikas"

// square bracket operator

console.log(human['age']);

console.log(human['name']);//"Vikas"

\*/

const arr = ["a", "b", "c", "d", "a","e", "b","c","c","e"];

const freqMap = {};

for (let index = 0; index< arr.length; index++){

const key = arr[index];

if (!freqMap[key]){

freqMap[key] = 1;

}else{

freqMap[key] = freqMap[key] + 1;

}

}

console.log(freqMap);

/\*

freqMap['a'] = 1;

freqMap.b = 1;

freqMap.c = 1;

freqMap.d = 1;

freqMap['a'] = freqMap['a'] + 1;

console.log(freqMap);

\*/

—-

/\*

Given an array and a constant "k",

Find any pair of x and y

where x + y = k

\*/

const arr = [3, 7, 9, 14, 10, 16, 2, 12, 19]

const k = 16;

const getPairs = (arr, k) => {

const tempMap = {};

const pairsArr = [];

for (let index = 0; index < arr.length; index++ ){

const item = arr[index]; //3

const pairItem = k - item; //13

if(tempMap[pairItem]){

pairsArr.push([item, pairItem]);

}else{

tempMap[item] = 1;

}

}

console.log(pairsArr);

}

getPairs(arr,k);

—------

7th May

const arr = [36, 41, 56, 35, 44, 33, 34, 92, 43, 32, 42];

//[1, 9, 3, 10, 4, 20, 2];

const maxConSubsequence = arr => {

const freqMap = {};

for (let index = 0; index< arr.length; index++){

freqMap[arr[index]] = true;

}

let MAX\_LENGTH = Number.NEGATIVE\_INFINITY;

for (let index = 0; index< arr.length; index++){

const num = arr[index] - 1;

let currentLength = 0;

if (!freqMap[num]){

// then start checking for arr[index] + 1;

let curr = 1;

currentLength = currentLength + 1;

while (freqMap[arr[index] + curr]){

currentLength = currentLength + 1;

curr++;

}

if(MAX\_LENGTH < currentLength){

MAX\_LENGTH = currentLength;

}

}

}

console.log("MAX\_SUM--"+ MAX\_SUM);

}

maxConSubsequence(arr);

—--

//Recursion

// Recursion means -- A function calling itself

// you can also call the function from itself

// Recursion must have a base/breaking condition

// print nums from 1 to 20 without using loops

const print = num => {

if(num> 20){

return;

}

else{

console.log(num);

print(num+1)

}

}

print(1);

/\*

main (5);

function main(num){

const newNum = num + 2;

console.log(newNum);

subFn(newNum); // F2 from F1

}

function subFn(num){

num = num+10;

console.log(num);

// if (num < 100){

subFn(num);

// }

}

\*/

/// you can also call the function from itself

–

/\*

Given a number n, print its factorial using recursion

factorial -

if n is 5 - 5 \* 4 \* 3 \* 2 \* 1

5! -->. 5 \* 4!

-->. 5 \* 4 \* 3!

--> 5 \* 4 \* 3 \* 2!

--> 5 \* 4 \* 3 \* 2 \* 1!

---> 5 \* 4 \* 3 \* 2 \* 1

\*/

const factorial = (num) => {

if(num == 0 || num== 1){

return 1;

}

return num \* factorial(num-1);

}

console.log(factorial(5));

console.log(factorial(10));

/\*

const print = num => {

if(num> 5){

return;

}

else{

const value = print(num+1);

console.log(value);

return num;

}

}

const val = print(1);

console.log(val);

\*/

—------

9 may

// SLIDING WINDOW ALGORITHM

// Best Time to Buy & Sell Stock

/\*

Input: prices[] = [7, 1, 5, 3, 6, 4]

Output: 5

Explanation:

The lowest price of the stock is on the 2nd day, i.e. price = 1. Starting from the 2nd day, the highest price of the stock is witnessed on the 5th day, i.e. price = 6.

Therefore, maximum possible profit = 6 – 1 = 5.

prices2 = [7,6,5,4,3,2,1]

Output = 0;

prices3 = [5,5,5,5]

output = 0

prices4 = [10]

output = 0

prices5 = []

output = 0

prices6 = [1,2,3,4,5,6]

ouput = 5

\*/

const prices = [7, 1, 5, 3, 6, 4];

const calculateMaxProfit = (arr) => {

let maxProfit = 0;

let minPrice = Number.POSITIVE\_INFINITY;

for (let index = 0; index< arr.length; index++){

if(arr[index] < minPrice){

minPrice = arr[index];

}else{

const currentProfit = arr[index] - minPrice;

if(currentProfit > maxProfit){

maxProfit = currentProfit;

}

}

}

return maxProfit;

}

console.log(calculateMaxProfit(prices));

console.log(calculateMaxProfit([7,6,5,4,3,2,1]));//0

console.log(calculateMaxProfit([5,5,5,5]));//0

console.log(calculateMaxProfit([10]));//0

console.log(calculateMaxProfit([]));//0

console.log(calculateMaxProfit([1,2,3,4,5,6,9]));

—---

—-----

10 May

/\*

Given a string s,

find the length of the longest substring

without repeating characters.

Input: s = "abcabcbb"

Output: 3

Explanation: The answer is "abc", with the length of 3.

s2: "abcdabcbb"

output: 4

s3: "abcdefghijklmnopqrstuvxyz"

output: 26

s4: "xxxxxxxxxxxxxxx"

output: 1

s5: ""

output: 0

s6: "pwawkew"

output: 4

s7: "abcdabcd"

ouput: 4

s8: "abcdaxyzeibod"

ouput: 10

\*/

//"abcabcbb"

const longestSubstr = (str) => {

let p1 =0, p2 = 0;

let charMap = {};

let MAX\_LENGTH = 0;

let currentlength = 0;

while ( p2<str.length){

const letter = str[p2];

if (charMap[letter] == undefined){

charMap[letter] = p2;

currentlength++;

p2++;

if(currentlength > MAX\_LENGTH){

MAX\_LENGTH = currentlength;

}

}

else{

const pos = charMap[letter];

p1 = pos+1;

p2 = p1;

charMap = {};

currentlength = 0;

}

}

return MAX\_LENGTH;

}

console.log(longestSubstr("abcabcbb"));//4 --> wrong //3

console.log(longestSubstr("abcdabcbb"));//5 --> wrong //4

console.log(longestSubstr("abcdefghijklmnopqrstuvwxyz"));//26

console.log(longestSubstr("xxxxxxxxxxxxxxx"));//2 --> wrong //1

console.log(longestSubstr(""));//0

console.log(longestSubstr("pwawkew"));//4

console.log(longestSubstr("abcdabcd"));//5 --> wrong //4

console.log(longestSubstr("abcdaxyzeibod"));//10

–

const newArr = [1,7,5,4, 19, 90, 34];

/\*

newArr.sort((a,b) => {

if(a < b){

return -100;//any negative number

}

else if (b < a){

return 500; // any positive number

}

else

return 0;

})

console.log(newArr);

\*/

newArr.sort((a,b) => {

return a-b;

//case 1: if a < b, e.g. 4,7 a-b = -3 which is a -ve num

// case 2: if a > b, e.g 14, 8, 14 - 8 = 6, +ve num

//case 3: a-b will return 0

})

console.log(newArr);

newArr.sort((a,b) => {

return b-a;

})

console.log(newArr);

—

const arr = [1,4,2,7,9,6];

arr.push(3);

console.log(arr);

arr.sort();

console.log(arr);

let name = "Vikas";

name = name.replace("i", "o");

console.log(name);

const months = ['March', 'Feb', 'Jan', 'Dec'];

months.sort();

console.log(months);

// "c" "axzscdkheirf"

// aznjdnfdnjfk cwweeww

//"1", "30, "4", "21" , "1000000"

const array1 = [1, 30, 4, 21, 100000];

array1.sort();

console.log(array1);//1,4,21,30,100000

//[1, 100000, 21, 30, 4]

/\* const callbackFn = () => {

}

// sort (callbackFn)

// arr.map( (x) => {

return x\*x;

})

// sort ((a,b)=>{

if (output < 0){

// sort a before b

}

else if (output > 0){

// sort b before a

}

else{

// if output == 0

/// wont change anything

}

})

callbackFN --> compare function

in this compare fn, we give two inputs/args

a,b

\*/

const newArr = [1,undefined,7,5,undefined,4,undefined];

newArr.sort((a,b) => {

return -1;

})

console.log(newArr);

newArr.sort((a,b) => {

return 1;

})

console.log(newArr);

console.log("hello");

newArr.sort((a,b) => {

return -1;

})

console.log(newArr);