**1. Filter unique array members using Set**.

let arr=[1,2,3,4,5,6,7,8,9,1,2,3,4,5,6,7,8,9];

const arr2= new Set(arr);

console.log(arr2);

**2. Find the possible combinations of a string and store them in a MAP?**

function getCharMap(str)

{

let map = new Map();

let idx = 0;

for(let c of str.split("")) {

map.set(c, idx++);

}

return map;

}

function jumble(str){

if (str.length < 2)

return str;

let permutations = [];

let charMap = getCharMap(str);

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

let char = str[i];

if (charMap.get(char) != i)

continue;

let remainingString = str.slice(0, i) + str.slice(i + 1, str.length);

for (let subPermutation of jumble(remainingString)) permutations.push(char + subPermutation) }

return permutations; }

console.log(jumble("apple"));

**3. Write a program to implement inheritance upto 3 classes.The Class must be public variables and static functions.**

class vehicle {

constructor(wheels, engine) {

this.wheels = wheels;

this.engine = engine;

}}

class car extends vehicle{

constructor(wheelCount, engineType, transmission) {

super(wheelCount, engineType);

this.transmission = transmission;

}}

class nissan extends car {

constructor(wheelCount, engineType, transmission, model) {

super(wheelCount, engineType, transmission);

this.model = model;

}

getCarInfo = () => {

return {

wheels: this.wheels,

engine: this.engine,

transmission: this.transmission,

model: this.model };

}}

let gtrModel = new nissan(4, "v8", "automatic", "GT-R");

console.log(gtrModel.getCarInfo());

**4. Write a program to implement a class having static functions**

class instanceCounter {

constructor() {

if(instanceCounter.instances)

instanceCounter.instances += 1;

else

instanceCounter.instances = 1; }

static getInstanceCount() {

return instanceCounter.instances; }

}

let i1 = new instanceCounter();

let i2 = new instanceCounter();

let i3 = new instanceCounter();

let i4 = new instanceCounter();

let i5 = new instanceCounter();

console.log(instanceCounter.getInstanceCount());

7. Write a program to flatten a nested array to single level using arrow functions.

const flattenService = (toBeFlattened, flattened) => { if(Array.isArray(toBeFlattened))

toBeFlattened.forEach(ele => flattenService(ele, flattened));

else

flattened.push(toBeFlattened);

}

const flatten = arr => {

let flattened = [];

arr.forEach(ele => flattenService(ele, flattened));

return flattened;}

let nested = [1, 2, [3, 4, [5, 6]]];

console.log( flatten(nested) );

**8. Implement a singly linked list in es6 and implement addFirst() addLast(), length(), getFirst(), getLast(). (without using array)**

class node {

constructor(data) {

this.data = data;

this.next = null;

}}

class linkedList {

constructor(firstNode) {

this.head = firstNode;

this.last = firstNode;

}

addfirst = (node) => {

node.next = this.head;

this.head = node;

}

addLast = (node) => {

this.last.next = node;

this.last = node;

}

getLength = () => {

let currentNode = this.head;

let length = 0;

while(currentNode) {

length ++;

currentNode = currentNode.next;

}

return length; }

printlist = () => {

let currentNode = this.head;

while(currentNode) {

console.log(currentNode.data);

currentNode = currentNode.next;

}

}

getFirst = () => this.head;

getLast = () => this.last; }

let n1 = new node(1);

let n2 = new node(2);

let n3 = new node(3);

let n4 = new node(4);

let n5 = new node(5);

let n6 = new node(6);

let list = new linkedList(n1);

list.addLast(n2);

list.addLast(n3);

list.addfirst(n4);

list.addfirst(n5);

list.addfirst(n6);

list.printlist();

console.log("length: " + list.getLength());

**9. Implement Map and Set using Es6**

let map = new Map();

map.set(1, 'qwer');

map.set(2, 'ert');

map.set(3, 'fsdf');

let set = new Set();

set.add("1");

set.add("2");

set.add("3");

**10. Implementation of stack (using linked list)**

class stack {

constructor() {

this.top = null;

}

static stackNode = data => ({data: data, next: null})

push = (data) => {

let node = stack.stackNode(data);

node.next = this.top;

this.top = node;

}

pop = () => {

let top = this.top.data;

this.top = this.top.next;

return top;

}

printStack = () => {

let current = this.top;

while(current) {

console.log(current.data);

current = current.next;

}

}}

let s = new stack();

s.push(1);

s.push(2);

s.push(3);

s.push(4);

s.push(5);

s.push(6);

s.push(7);

s.printStack();

console.log("popping " + s.pop());

console.log("popping " + s.pop());

console.log("popping " + s.pop());

console.log("popping " + s.pop());

s.printStack();