**codingame solution**

**exercice pipe’s**

<div id="fee">{{fee|number|percent:'2.2-3'}}</div>

<div id="amount">{{amount|currency:currency:'symbol':'9.2-2'}}</div>

<div id="time">{{timeOfTransaction|date:'ww: yyyy MMMMM dd hh-mm-ss'}}</div>

**luckyMoney**

function luckyMoney(money, giftees) {

if (money < 0 || money >= 100) return -1; //incorrect money input

if (giftees < 0 || giftees >= 10) return -2; //incorrect giftees input

if (money >= giftees \* 8) return giftees;

if (money < giftees) return 0; // not enough money

let b = money, g = giftees;

while (b >= 8 + (g - 1) && g > 0) { b -= 8; g--; }

if(g === 1 && b === 4) return 0;

return giftees - g;

}

**Voter component**

Dans display-component html:

<div id="lastVote">{{ answer }}</div>

<voter-component

[question]="question"

[yesAnswer]="yesAnswer"

[noAnswer]="noAnswer"

(output)="setVote($event)"

></voter-component>

Dans DisplayComponent:

public answer: string = 'Yes';

setVote(event: boolean) {

return (this.answer = event ? this.yesAnswer : this.noAnswer);

}

arbre

class Node { // keep these​​​​​​‌​​‌‌‌​‌​​​​​‌​​‌​‌​​​‌‌‌ fields

Node left, right;

int value;

public Node find(int v){

Node current = this;

while (current != null) {

if (current.value == v)

{ return current; }

current = v < current.value ? current.left : current.right; }

return null; }

}

class Node { // keep these​​​​​​‌​​‌‌‌​‌​​​​​‌​​‌​‌​​​‌‌‌ fields

Node left, right;

int value;

public Node find(int v){

Node current = this;

while (current != null) {

if (current.value == v)

{ return current; }

current = v < current.value ? current.left : current.right; }

return null; }

}

public static int *findSmallestInterval*(int[] numbers) {

// Write your code here

Arrays.*sort*(numbers);

int diff = **Integer**.**MAX\_VALUE**;

for (int i = 0; i < numbers.**length** - 1; i++)

if (numbers[i + 1] - numbers[i] < diff)

diff = numbers[i + 1] - numbers[i];

// Return min diff

return diff;

}

if(ts.**length**==0)

return 0;

int min =ts[0];

for (int i=1; i<ts.**length** ; i++){

min = min > **Math**.*abs*(0-ts[i]) ? **Math**.*abs*(0-ts[i]) : **Math**.*abs*(min);

}

return min;

**Counter Component**

import { Component, Output, Input } from '@angular/core';

import { Subject, of, interval, Observable } from 'rxjs';

import { filter, switchMap, map } from 'rxjs/operators';

<input type="number" id="intervalInput"/>

<button (click)="buttonClicked.next()">Set interval</button>

@Input() message: string;

public inputValue: string = '' ;

public buttonClicked: Subject<void> = new Subject();

@Output() tick: Observable<string> = this.buttonClicked.pipe(

map(()=>parseInt(this.inputValue = (<HTMLInputElement>document.getElementById("intervalInput")).value, 10)),

filter(value => Number.isInteger(value)),

switchMap(value => interval(value)),

map(() => this.message))

**Couples templates Person**

@Pipe({ name: "name" })

export class Name implements PipeTransform {

transform(person, reverseName: boolean = false) {

if (!reverseName) {

const first = person.firstName;

const last = person.lastName;

person.firstName = last;

person.lastName = first;

return person;

}

return person;

}

}

@Pipe({ name: "sort" })

export class Sort implements PipeTransform {

transform(person, reverseName: string = "backwards"): string {

if (reverseName == "same") {

return `${person.lastName} ${person.firstName}`;

}

if (reverseName == "backwards") {

person.firstName = person.firstName.split("").reverse().join("");

person.lastName = person.lastName.split("").reverse().join("");

return `${person.firstName} ${person.lastName}`;

}

return `${person.firstName} ${person.lastName}`;

}

}

[Lien stackBlitz](https://stackblitz.com/edit/angular-ivy-zx5o8k?file=src%2Fapp%2Fapp.component.ts)

**computeDayGains**

function computeDayGains(nbSeats, payingGuests, guestMovements) {

// Write your code here

// To debug: console.error('Debug messages...');

let availableSeats = nbSeats;

let guestSitting = [];

let waitingList = [];

let alreadyEntered = []

let dayGains = 0;

guestMovements.forEach(id => {

if (guestSitting.indexOf(id) == -1 && availableSeats > 0) {

if (alreadyEntered.indexOf(id) == -1)

dayGains += payingGuests[id];

guestSitting.push(id);

alreadyEntered.push(id);

availableSeats = availableSeats - 1;

} else if (guestSitting.indexOf(id) != -1) {

availableSeats += 1;

guestSitting.splice(guestSitting.indexOf(id), guestSitting.indexOf(id));

} else if (availableSeats == 0) {

waitingList.push(id);

}

});

return dayGains;

}

/\* Ignore and do not change the code below \*/

const nbSeats = parseInt(readline());

const nbGuests = parseInt(readline());

const nbMovements = parseInt(readline());

const payingGuests = readline().split(' ').map(j => parseInt(j)).slice(0, nbGuests);

const guestMovements = readline().split(' ').map(j => parseInt(j)).slice(0, nbMovements);

const oldWrite = process.stdout.write;

process.stdout.write = chunk => { console.error(chunk); return true }

const res = computeDayGains(nbSeats, payingGuests, guestMovements);

process.stdout.write = oldWrite;

console.log(res);

/\* Ignore and do not change the code above \*/

**Calc**

function calc(array, n1, n2) {

// Your code goes here

let sum=0;

for(let i=n1;i<=n2;i++){

sum+=array[i]

}

return sum;

}

**TestComponent**

HTML

<div \*ngIf="!inputData" id="noData">No Data</div>

<div id="dataList">

<div \*ngFor="let data of inputData" [ngStyle]="{'color': (data.length %2===0) ? 'green': 'red'}"> {{data}}</div>

</div>

TS

@Input() inputData: Array<string>

“”””””””””””””””””””””””””””””””””””””””””””

// Angular 15.x​​​​​​‌​​‌‌‌‌‌​​​​​‌​​‌‌‌‌​​‌​​ code import { Component, NgModule, Injectable, Inject, forwardRef } from '@angular/core'; // Do not modify the service as validation tests will not pass @Injectable({ providedIn: 'root' }) export class AccountingService{ private accountsWithBalance: Map = new Map(); public async createAccount(accountID: string): Promise { if (this.accountsWithBalance.has(accountID)){ throw Error("ACCOUNT\_EXISTS"); } this.accountsWithBalance.set(accountID, 0); } public async topUp(accountID: string, amount: number): Promise{ if (!this.accountsWithBalance.has(accountID)){ throw Error("ACCOUNT\_DOES\_NOT\_EXIST"); } this.accountsWithBalance.set(accountID, this.accountsWithBalance.get(accountID)+ amount); return this.accountsWithBalance.get(accountID); } } // Implement the component here @Component({ selector:'account-component', template: ` create topup `, }) export class AccountComponent { public cc:string=""; constructor(public accountingService:AccountingService){ } public createC(){ var inputValue = (document.getElementById("createAccountInput")).value; this.accountingService.createAccount(inputValue); } public depose(){ var val1 = (document.getElementById("topupAccountInput")).value; var val2 = (document.getElementById("topupAmountInput")).value; this.accountingService.topUp(val1,Number(val2)); } } // #region Preview @Component({ template: `` }) export class PreviewComponent { } // #endregion Preview // #region Module declaration - Do not Change @NgModule({ declarations: [PreviewComponent, AccountComponent], entryComponents: [PreviewComponent] }) export class PreviewModule { } // #endregion Module declaration

Correction   
@Component({

selector: 'account-component',

template: `

<div>

<h2>Create Account</h2>

<input id="createAccountInput" type="text" placeholder="Account ID">

<button id="createAccountButton" (click)="createAccount()">Create</button>

</div>

<div>

<h2>Top Up Account</h2>

<input id="topupAccountInput" type="text" placeholder="Account ID">

<input id="topupAmountInput" type="number" placeholder="Amount">

<button id="topupAccountButton" (click)="topUpAccount()">Top Up</button>

</div>

<div id="message">{{ message }}</div>

`,

})

export class AccountComponent {

public message: string = "";

constructor(public accountingService: AccountingService) {}

public async createAccount(): Promise<void> {

const accountID = (document.getElementById("createAccountInput") as HTMLInputElement).value;

try {

await this.accountingService.createAccount(accountID);

this.message = "Successfully added account";

} catch (error) {

this.message = error.message;

}

}

public async topUpAccount(): Promise<void> {

const accountID = (document.getElementById("topupAccountInput") as HTMLInputElement).value;

const amount = Number((document.getElementById("topupAmountInput") as HTMLInputElement).value);

if (amount <= 0) {

this.message = "INVALID INPUT";

return;

}

try {

const balance = await this.accountingService.topUp(accountID, amount);

this.message = "Current balance: " + balance;

} catch (error) {

this.message = error.message;

}

}

}

function countFrequencies(words) {

// Write your code here

// To debug: console.error('Debug messages...');

let x=words.sort();

let res=[];

let counnt=0;

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

counnt++;

if(words[i]!=words[i+1]){

res.push(counnt);

counnt=0

}

}

return res;

}

function reshape(n, str) {

str = str.replace(/\s/g, '');

var regex = new RegExp(".{" + n + "}", "g");

return str.replace(regex,"$&\n");}

function calculateTotalPrice(prices, discount) {

// Write your code here

// To debug: console.error('Debug messages...');

let total;

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

let ff=prices.reduce((a, b) => a + b, 0);

total=ff-(max\*discount)/100;

return Math.floor(total) ;

}

let result=[];

if((patternWidth>=1)&&(patternWidth<=400)&&

(patternHeight>=1)&&(patternHeight<=400)&&

(imageWidth>=1)&&(imageWidth<=400)&&

(imageHeight>=1)&&(imageHeight<=400)&&

(patternWidth<=imageWidth)&& (patternHeight<=imageHeight)){

let img\_x,img\_y,pattern\_x,pattern\_y,patttern\_not\_found,pattern\_is\_matching;

result=[-1,-1];

patttern\_not\_found=true;

for(img\_x=0;(patttern\_not\_found &&(img\_x<=(imageWidth-patternWidth)));++img\_x){

for(img\_y=0;(patttern\_not\_found &&(img\_y<=(imageHeight-patternHeight)));++img\_y){

pattern\_is\_matching=true;

for(pattern\_x=0;(pattern\_is\_matching && (pattern\_x<patternWidth));++pattern\_x){

for(pattern\_y=0;(pattern\_is\_matching && (pattern\_y<patternHeight));++pattern\_y){

if(image[(img\_x+pattern\_x)][(img\_y+pattern\_y)]!=pattern[pattern\_x][pattern\_y]){

pattern\_is\_matching=false;

}

}

}

if(pattern\_is\_matching){

patttern\_not\_found=false;

result=[img\_x,img\_y]

}

}

}

}

return result;

}

**createEE**   
const **createEE** = ({ fn, interval, signal }) => {

const **e** = new EventEmitter();

let timerId;

const **emitData** = () => {

try {

const **data** = fn();

**e**.emit("data", **data**);

} catch (error) {

**e**.emit("error", error);

}

};

const **stop** = () => {

clearTimeout(timerId);

**e**.emit("close");

};

**emitData**();

const **scheduleEmit** = () => {

timerId = setTimeout(() => {

**emitData**();

**scheduleEmit**();

}, interval);

};

if (signal) {

signal.addEventListener("abort", **stop**);

}

**scheduleEmit**();

return **e**;

};

promisify  
  
function promisify(callbackFunction) {

return async function (...args) {

return new Promise((resolve, reject) => {

const **callback** = function (result) {

resolve(result);

};

args.push(**callback**);

callbackFunction.apply(this, args);

});

};}

doCalculations  
  
async function doCalculations(inputs, workerPath = "/tmp/deps/worker") {

if (cluster.**isMaster**) {

const **numWorkers** = inputs.length;

const **results** = [];

const **promises** = inputs.map((input, index) => {

return new Promise((resolve, reject) => {

const **worker** = cluster.fork();

**worker**.on("message", (result) => {

**results**[index] = result;

resolve();

});

**worker**.on("error", (error) => {

reject(error);

});

**worker**.send(input);

});

});

return Promise.all(**promises**).then(() => **results**);

} else {

process.on("message", (input) => {

const **result** = calculate(input);

process.send(**result**);

});

}

}

function calculate(input) {

return input.toUpperCase();

}