**JS**

1.

**# Subiect 2 (2.5 pts)**

**# Tematica: Javascript**

**# Avand urmatoarea functie function `bowdlerize(input, dictionary)` unde:**

- `input` este un string (ex. "This is a cat")

- `dictionary` este un vector ce contine o serie de string-uri.

**# Completati urmatoarele taskuri:**

- `input` trebuie sa fie de tip `string`. Daca este dat un input de alt tip se va arunca un `Error` cu mesajul `Input should be a string`; (0.5 pts)

- `dictionary` este un vector de elemente de tip `string`. Daca cel putin un element nu este `string` se va arunca un `Error` cu mesajul `Invalid dictionary format`; (0.5 pts)

- Daca `dictionary` contine cuvinte, vor fi inlocuite in `input` cu prima litera, urmată de  o serie de caractere `\*` urmate de ultima litera. Lungimea cuvantului rezultat este acceasi ca a celui initial (ex. 'test' va deveni 't\*\*t') ; (0.5 pts)

- Se va returna un nou string, `input` nefiind modificat; (0.5 pts)

- Functia returneaza rezultatul corect si pentru cuvinte care incep cu litere mari. (0.5 pts)

Rezolvare:

function bowdlerize(input, dictionary)

{

        if (typeof input != "string") {

          throw new Error("Input should be a string");

        }

        dictionary.forEach(word => {

          if (typeof word != "string") {

            throw new Error("Invalid dictionary format");

          }

        });

        let sentence = input.split(" ");

        let censored = input;

        dictionary.forEach(wordD => {

          sentence.forEach(wordS => {

              let lowercase = wordS.toLowerCase();

            if (wordD === wordS || wordD === lowercase) {

              let newWord = wordS[0]+"\*".repeat(wordS.length-2)+wordS[wordS.length-1];

              censored = censored.replace(wordS,newWord);

            }

          });

        });

        return censored;

}

const app = {

    bowdlerize

};

module.exports = app;

2.

**# Subiect 2 (2.5 pts)**

**# Tematica: Javascript**

**# Avand urmatoarea functie  `removeOrderItem(orderInfo, position)` unde:**

- `orderInfo` este un obiect cu proprietatile total si items

- `position` este un numar intreg ce determina un element din items

**# Completati urmatoarele taskuri:**

- validati proprietatea `items` sa fie de tip `array`. Daca este dat un input de alt tip se va arunca eroare cu mesajul `Items should be an array`; (0.5 pts)

- fiecare obiect din `items` este are proprietatile price si quantity. Daca cel putin un element nu respecta formatul se va arunca mesajul `Malformed item`; (0.5 pts)

- parametrul `position` este valid in relatie cu vectorul items (0.5 pts)

- se va returna obiectul orderInfo din care s-a eliminat elementul de pe pozitia data (0.5 pts)

- functia returneaza totalul corect pentru comanda in urma eliminarii elementului de pe pozitia data. (0.5 pts)

Rezolvare:

function removeOrderItem(orderInfo, position){

    if(!Array.isArray(orderInfo.items)){

        throw new Error('Items should be an array')

    }

    for(let el of orderInfo.items){

        if(!el.price||!el.quantity){

        throw new Error('Malformed item')

        }

    }

    if(position<0||position>=orderInfo.items.length){

        throw new Error('Invalid position')

    }

    orderInfo.total-= orderInfo.items[position].price\*orderInfo.items[position].quantity

    orderInfo.items.splice(position,1)

    return orderInfo

}

const app = {

    removeOrderItem

};

module.exports = app;

3.

**# Subiect 2 (2.5 pts)**

**# Tematica: Javascript**

**# Avand urmatoarea functie function `calculateFrequencies(input, stopWords)` unde:**

- `input` este un string sau un obiect String (ex. "This is a cat")

- `stopWords` este un vector ce contine o serie de string-uri sau obiecte String.

**# Completati urmatoarele taskuri:**

- `input` trebuie sa fie de tip `string` sau `String`. Daca este dat un input de alt tip se va arunca un `Error` cu mesajul `Input should be a string`; (0.5 pts)

- `dictionary` este un vector de elemente de tip `string`sa `String`. Daca cel putin un element nu este `string` se va arunca un `Error` cu mesajul `Invalid dictionary format`; (0.5 pts)

- functia calculeaza frecventele relative ale cuvintelor in input si returneaza un dictionar care are cuvintele drept chei si frecventele drept valori (ex. pentru string-ul 'orange cat' rezultatul va fi {orange : 0.5, cat : 0.5}); (0.5 pts)

- daca stopWords contine cuvinte, ele vor fi ignorate pentru calcularea frecventelor (ex. pentru 'the orange cat' cu 'the' in stopWords rezultatul va fi {orange : 0.5, cat : 0.5}); (0.5 pts)

- functia returneaza rezultatul corect si pentru cuvinte care incep cu o litera mare, care sunt considerate identice cu varianta fara litera mare. (0.5 pts)

Rezolvare:

function remove\_stopwords(input,stopWords) {

    var res = []

    var words = input.split(' ');

    for(var i=0;i<words.length;i++) {

        if(!(stopWords.includes(words[i])||stopWords.includes(words[i].toLowerCase())))

        {

            res.push(words[i]);

        }

    }

    return(res.join(' '));

  }

function calculateFrequencies(input, stopWords){

    var s="";

      if (typeof(input) !== typeof(s)) {

    throw new Error("Input should be a string");

  }

  stopWords.forEach(word => {

    if (typeof(word) !== typeof(s)) {

      throw new Error("Invalid dictionary format");

    }

  });

 var result=remove\_stopwords(input,stopWords).toLowerCase();

    let splittedInput = result.split(" ");

    let dict = new Object();

    let nr = 0;

    splittedInput.forEach(wordInput => {

    if(dict.hasOwnProperty(wordInput))

    {

         dict[wordInput] ++;

         nr++;

    }

    else

    {

        dict[wordInput] = 1;

        nr++;

    }

    });

    splittedInput.forEach(obj =>{

        if(nr!==0){

            dict[obj] = dict[obj]/nr;

        }

    })

    console.log("$$$nr ",nr);

    return dict;

}

const app = {

    calculateFrequencies

};

module.exports = app;

Rezolvare netestata:

function calculateFrequencies(input, dictionary){

    if(typeof input === 'string' || input instanceof String){

        for(let element in dictionary){

            if(dictionary[element] instanceof String || typeof dictionary[element] === 'string'){

            }else{

                throw Error('Invalid dictionary format')

            }

        }

    }else{

        throw Error('Input should be a string')

    }

}

const app = {

    calculateFrequencies

};

module.exports = app;

4.

**# Subiect 2 (2.5 pts)**

**# TOPIC: Javascript**

**# Dată fiind funcția `function processString(input)`, care inițial tokenizează string-ul `input` în mai multe `tokens` separate de spațiu, rezolvați următoarele cerințe:**

- Dacă oricare `token` nu este un `number` sau un `Number`, funcția ar trebui să arunce `Error`

- Dacă oricare `token` nu este un `number` sau un `Number`, funcția ar trebui să arunce `Error` cu mesajul `Item is not a number`; (0.5 pts)

- Dacă `input` are lungime 0 funcția ar trebui să returneze 100; (0.5 pts)

- Token-urile `impare` sunt ignorate; (0.5 pts)

- Funcția returnează 100 minus suma tuturor `token`-urilor pare; (0.5 pts)

Rezolvare:

function processString(input){

    if(input === ''){

        return 100

    }

    var newInput = input.split(' ')

    var ok =1

    newInput.forEach(i => {

        if(isNaN(i)){

            ok = 0

        }

    })

    if(ok === 0){

        throw new Error('Item is not a number')

    } else{

        var sum = 0

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

            if(newInput[i] % 2 === 0){

                sum += parseFloat(newInput[i])

            }

        }

        return 100 - sum

    }

}

const app = {

    processString: processString

}

module.exports = app

5.

**# Subiect 2 (2.5 pts)**

**# TOPIC: Javascript**

**# Avand urmatoarea functie `function applyBlackFriday(products, discount)` unde:**

- `products` un vector de obiecte cu urmatorul format {name: string, price: number};

- `discount` un numar ce reprezinta discount-ul ce va fi aplicat preturilor produselor.

- Functia trebuie sa returneze un vector cu preturile reduse pentru toate produsele.

**# Completati urmatoarele cerinte:**

- Functia trebuie sa returneze un promise; (0.5 pts)

- `discount` trebuie sa fie un numar, in caz contrar apelati `reject` in promise cu `Error` si mesajul `Invalid discount`; (0.5 pts)

- `discount` trebuie sa fie mai mare ca 0 si mai mic egal cu 10, in caz contrar apelati `reject` in promise cu `Error` si mesajul `Discount not applicable`; (0.5 pts)

- `products` trebuie sa contina produse cu formatul specificat, in caz contrar se va apela `reject` cu `Error` si mesajul `Invalid array format`; (0.5 pts)

Un produs: {name: string, price: number}

- Functia trebuie sa returneze un vector cu preturile reduse pentru toate produsele; (0.5 pts)

Rezolvare:

function applyBlackFriday(products, discount){

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

        if(typeof discount !== 'number'){

            reject(new Error('Invalid discount'))

        } else if(discount < 0 || discount > 10){

            reject(new Error('Discount not applicable'))

        } else if(products.find(p => typeof p.name !== 'string' || typeof p.price !== 'number')){

            reject(new Error('Invalid array format'))

        } else{

            resolve(products.map(p => {

                return {

                    name : p.name,

                    price: p.price - discount/100 \* p.price

                }

            }))

        }

    })

}

const app = {

    applyBlackFriday: applyBlackFriday

};

module.exports = app;

Rezolvare netestata:

function applyBlackFriday(products, discount){

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

        if(typeof discount=="number") {

            products.forEach((e)=>

            {

                if(typeof e.name!="string" && typeof e.price!="number" )

                {  var eroare1='Invalid array format';

                      reject(new  Error(eroare1));

                }

            });

        if(discount<=10 && discount>0)

        {      resolve(products.map(e=>{

                    return{name: e.name,price: e.price- e.price\*discount/100};

                }));

        }

        else

        {

            var  eroare2="Discount not applicable";

             reject(new Error(eroare));

        }

        }

        else

        { var  eroare="Invalid discount";

             reject(new Error(eroare));

        }

     })

}

const app = {

    applyBlackFriday: applyBlackFriday

};

module.exports = app;

6.

- funcția translate primește ca parametrii un string și un obiect

 - funcția aruncă excepții dacă tipurile nu sunt respectate (mesajul "InvalidType")

 - obiectul dicționar are în cheie valoarea inițială și în valoare traducerea ei

 - valorile din dicționar sunt string-uri

 - funcția înlocuiește fiecare cheie din dicționar găsită în textul inițial cu valoarea tradusă

Rezolvare:

function translate(text, dictionary){

    if(typeof text !== 'string' || !dictionary || typeof dictionary !== 'object' ||  dictionary.constructor !== Object){

        throw new Error('InvalidType');

    }

    else {

    var words = text.split(' ');

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

        if(Object.keys(dictionary).find(key => key === words[i])){

        if(text.search(` ${words[i]}`) !== -1){

                text = text.split(` ${words[i]}`).join(` ${dictionary[words[i]]}`);

        }

        else if(text.search(` ${words[i]} `) !== -1){

                text = text.split(` ${words[i]} `).join(` ${dictionary[words[i]]} `);

        } else if(text.search(`${words[i]} `) !== -1){

                text = text.split(`${words[i]} `).join(`${dictionary[words[i]]} `);

        }

        }

        }

    return text;

    }

    // TODO: implementați funcția

    // TODO: implement the function

}

module.exports.translate = translate

Rezolvare netestata:

function translate(text, dictionary){

    // TODO: implementați funcția

    // TODO: implement the function

    if( typeof text!=="string")

    {

        throw new Error("InvalidType");

    }

    if(typeof dictionary!=="object" || !dictionary)

    {

        throw new Error('InvalidType');

    }

    for (let prop in dictionary){

        if (typeof dictionary[prop] !== 'string'){

            throw new Error('TypeError')

        }

    }

    let result = text.split(' ')

    for (let prop in dictionary){

        let position = result.indexOf(prop)

        if (position !== -1){

            result[position] = dictionary[prop]

        }

    }

    return result.join(' ')

}

module.exports.translate = translate

7.

**# Subiectul 2 (2.5 pts)**

**# Tematica: Javascript**

**# Avand urmatoarea functie `function addTokens(input, tokens)` unde:**

- `input` este un string ce poate sa contina "...". De exemplu: Subsemnatul ..., dominiciliat in ...;

- `tokens` un vector de tokenuri.

- Functia trebuie sa inlocuiasca toate `...` din `input` cu valorile corespunzatoare din `tokens` sub urmatorul format `${tokenName}`, in ordinea in care exista in vector;

**# Complete the following tasks:**

- `input` trebuie sa fie de tip `string`. Daca alt tip este pasat ca si parametru aruncati `Error` cu mesajul `Input should be a string`; (0.5 pts)

- `input` trebuie sa aiba cel putin 6 caractere ca si lungime. Daca dimensiunea `input-ului` este mai mica de 6, aruncati `Error` cu mesajul `Input should have at least 6 characters`; (0.5 pts)

- `tokens` este un vector de elemente cu urmatorul format: `{tokenName: string}`. Daca urmatorul format nu este respectat, aruncati `Error` cu urmatorul mesaj `Invalid array format`; (0.5 pts)

- Daca `input` nu contine `...` returnati valoarea initiala a `input-ului`; (0.5 pts)

- Daca `input` contine `...`, inlocuiti-le cu valorile specifice si returnati noul `input`; (0.5 pts)

Rezolvare:

function addTokens(input, tokens){

    if(typeof input !== 'string'){

        throw new Error('Invalid input')

    } else if(input.length <= 6){

        throw new Error('Input should have at least 6 characters')

    } else if(tokens.find(a => typeof a.tokenName !== 'string' )){

        throw new Error('Invalid array format')

    } else if(input.search('...') !== -1){

        for(let e of tokens){

            let a="${"+e.tokenName+"}";

            input = input.replace("...", a)

        }

    }

    return input

}

const app = {

    addTokens: addTokens

}

module.exports = app;

Rezolvare netestata:

function addTokens(input, tokens){

     if( typeof input!=="string"){

         throw new Error("Invalid input");

     }

    else

    {

        if(input.length <6)

        {

            throw new Error("Input should have at least 6 characters");

        }

    else {

        tokens.forEach((e)=>

         {

         if( typeof e.tokenName !=="string")

         {

             throw new Error("Invalid array format");

         }

         else

         {

         }

        })

        var inputnou;

        if(input.includes("..."))

        { tokens.forEach((e)=>{

            inputnou=input.replace("...", "${" + e.tokenName +"}");});

            return inputnou;

        }

        else

        {return input;

        }

    }

    }

}

const app = {

    addTokens: addTokens

}

module.exports = app;

8.

 - funcția capitalize primește ca parametrii un string și un array

 - dicționarul conține o serie de termeni

 - in textul ințial cuvintele sunt separate de spațiu

 - fiecare termen din dicționar trebuie să apară capitalizat în rezultat

 - rezultatul este un string nou, fără modificarea celui inițial

 - dacă textul nu este un string sau dicționarul nu este un array de string-uri se va arunca o excepție (mesajul TypeError)

Rezolvare:

function capitalize(text, dictionary){

    if(typeof text !== 'string' || !Array.isArray(dictionary)){

        throw new Error('TypeError')

    } else if(dictionary.find(d => typeof d !== 'string')){

        throw new Error('TypeError')

    } else if(dictionary.length === 0){

        return text

    } else{

        text = text.split(" ")

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

            for(var j=0; j < text.length; j++){

                if(text[j].toUpperCase() === dictionary[i].toUpperCase()){

                    var value = dictionary[i].charAt(0).toUpperCase() + dictionary[i].slice(1)

                    text.splice(j, 1, value)

                    var modified = text.toString()

                    modified = modified.replace(/,/g , " ")

                }

            }

        }

        return modified

    }

}

module.exports.capitalize = capitalize

Rezolvare neverificata:

function capitalize(text, dictionary){

  if(typeof text!=="string" || !Array.isArray(dictionary))

  {

    throw new Error("TypeError");

  }

  if (dictionary.filter((e) => typeof e !== 'string').length){

        throw new Error('TypeError')

    }

  let result = text

    let items = result.split(' ')

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

        if (dictionary.indexOf(items[i]) !== -1){

            items[i] = items[i][0].toUpperCase() + items[i].slice(1, items[i].length)

        }

    }

    return items.join(' ')

}

module.exports.capitalize = capitalize

9.

- Functia trebuie sa returneze un Promise; (0.5 pts)

// - Daca `discount` nu este numar, functia trebuie sa apeleze `reject` cu `Error` si mesajul `Invalid discount`; (0.5 pts)

// - `vehicles` este un vector ce contine elemente cu urmatorul format: `{make: string, price: number}` (Example: [{make: "Audi A5", price: 15000}]). Daca este pasat un vector cu obiecte invalide, se apeleaza `reject` cu `Error` si mesajul `Invalid array format`; (0.5 pts)

// - Functia trebuie sa apeleze `reject` cu `string` cu `Discount too big` daca `discount` este mai mare de 50% fata de cel mai mic pret din `vehicles`; (0.5 pts)

// - Functia trebuie sa apeleze `resolve` cu un vector ce contine valorule modificate pentru fiecare element din `vehicles`; (0.5 pts)

Rezolvare:

function applyDiscount(vehicles, discount){

    return new Promise((resolve,reject) =>

    {

        if(typeof discount !== 'number')

        {

            reject(new Error('Invalid discount'));

        }

        let smallestPrice = vehicles[0].price;

        let newVechiles =[];

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

            if( typeof vehicles[i].make !== 'string' || typeof vehicles[i].price !== 'number'){

                reject(new Error('Invalid array format'));

            }

            else if(vehicles[i].price < smallestPrice){

                smallestPrice = vehicles[i].price;

            }

            newVechiles.push({price: vehicles[i].price -discount, make:vehicles[i].make});

        }

        if(discount > 0.5 \* smallestPrice){

            reject('Discount too big');

        }

        else {

            console.log(newVechiles);

            resolve(newVechiles);

        }

    })

}

const app = {

    applyDiscount: applyDiscount

};

module.exports = app;

Rezolvare netestata:

function applyDiscount(vehicles, discount){

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

          if(typeof discount=="number")

          {var minprice=9999999;

             vehicles.forEach((e)=>

             { if(typeof e.make!="string" && typeof e.price!="number")

                {

                var eroare2="Invalid array format";

                reject(new Error(eroare2));

                 }

                 else

                 {

                     if(e.price<minprice)

                     {

                         minprice=e.price;

                     }

                 }

             });

             if(discount> minprice\*0.5)

             {

                 reject("Discount too big");

             }

              else {resolve(vehicles.map(e=>{

                    return{make: e.make,price:e.price-discount};

                })); }

          }

          else

          {

              var eroare1='Invalid discount';

              reject(new Error(eroare1));

          }

      });

}

const app = {

    applyDiscount: applyDiscount

};

module.exports = app;

10.

**# Subiectul 2 (2.5 pts)**

**# Tematica: Javascript**

**# Avand urmatoarea functie `function applyBonus(employees, bonus)`, completati urmatoarele taskuri:**

- Functia trebuie sa returneze un Promise; (0.5 pts)

- Daca `bonus` nu este numar, functia trebuie sa apeleze `reject` cu `Error` si mesajul `Invalid bonus`; (0.5 pts)

- `employees` este un vector ce contine elemente cu urmatorul format: `{name: string, salary: number}` (Example: [{name: "John Doe", salary: 5000}]). Daca este pasat un vector cu elemente invalide, functia trebuie sa apeleze `reject` cu `Error` si mesajul `Invalid array format`; (0.5 pts)

- Functia trebuie sa apeleze `reject` cu `string` cu valoarea `Bonus too small` daca `bonus` este mai mic de 10% din salariul maxim din `employees` array; (0.5 pts)

- Functia trebuie sa apeleze `resolve` cu un vector ce contine salariile marite pentru fiecare angajat; (0.5 pts)

Rezolvare

function applyBonus(employees, bonus){

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

        if(typeof bonus !== 'number'){

            reject(new Error("Invalid bonus"))

        }

        if(employees.find(employee => typeof employee.name !== 'string' || typeof employee.salary !== 'number')){

            reject(new Error('Invalid array type'))

        }

        let max = 0;

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

            if(employees[i].salary > max){

                max = employees[i].salary

            }

        }

        if(bonus < 0.1 \* max){

            reject('Bonus too small')

        } else{

            resolve(employees.map(employee => {

                return {

                    name : employee.name,

                    salary: employee.salary  + bonus

                }

            }))

        }

    })

}

let app = {

    applyBonus: applyBonus,

}

module.exports = app;

Rezolvare netestata:

function applyBonus(employees, bonus){

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

        if(typeof bonus=="number") {

                 var maxsalary=0;

            employees.forEach((e)=>

            {

                if(typeof e.name!="string" && typeof e.salary!="number" )

                {  var eroare1='Invalid array format';

                      reject(new  Error(eroare1));

                }

                else

                {

                    if( e.salary > maxsalary)

                    {

                        maxsalary=e.salary;

                    }

                }

            });

        if(bonus< maxsalary\*0.1)

        {

            reject("Bonus too small");

        }

        resolve(employees.map(e=>{

                    return{name: e.name,salary:e.salary+bonus};

                }));

        }

        else

        { var  eroare="Invalid bonus";

             reject(new Error(eroare));

        }

     })

}

let app = {

    applyBonus: applyBonus,

}

module.exports = app;

11.

// # Avand urmatoarea functie function `calculateFrequencies(input, stopWords)` unde:

// - `input` este un string sau un obiect String (ex. "This is a cat")

// - `stopWords` este un vector ce contine o serie de string-uri sau obiecte String.

// # Completati urmatoarele taskuri:

// - `input` trebuie sa fie de tip `string` sau `String`. Daca este dat un input de alt tip se va arunca un `Error` cu mesajul `Input should be a string`; (0.5 pts)

// - `dictionary` este un vector de elemente de tip `string`sa `String`. Daca cel putin un element nu este `string` se va arunca un `Error` cu mesajul `Invalid dictionary format`; (0.5 pts)

// - functia calculeaza frecventele relative ale cuvintelor in input si returneaza un dictionar care are cuvintele drept chei si frecventele drept valori (ex. pentru string-ul 'orange cat' rezultatul va fi {orange : 0.5, cat : 0.5}); (0.5 pts)

// - daca stopWords contine cuvinte, ele vor fi ignorate pentru calcularea frecventelor (ex. pentru 'the orange cat' cu 'the' in stopWords rezultatul va fi {orange : 0.5, cat : 0.5}); (0.5 pts)

// - functia returneaza rezultatul corect si pentru cuvinte care incep cu o litera mare, care sunt considerate identice cu varianta fara litera mare. (0.5 pts)

Rezolvare:

function calculateFrequencies(input, dictionary){

    if(typeof input !== 'string')

    {

        throw new Error('Input should be a string');

    }

    if(Array.isArray(dictionary))

    {

        let cuvinte = input.split(' ');

        let n = 0;

        let cuvinteNoi = [];

        let obj = {};

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

        {

            if(typeof dictionary[i] !== 'string' )

            {

                throw new Error('Invalid dictionary format');

            }

        }

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

        {

            if(!dictionary.find(cuvant => cuvant === cuvinte[i].toLowerCase()))

            {

                n++;

                obj[cuvinte[i].toLowerCase()]=0;

                cuvinteNoi.push(cuvinte[i].toLowerCase());

            }

        }

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

        {

            obj[cuvinteNoi[i]] = 1/n;

        }

        return obj;

    }

    else

    {

        throw new Error('Invalid dictionary format');

    }

}

const app = {

    calculateFrequencies

};

module.exports = app;

12.

// # Avand urmatoarea functie `function textProcessor(input, tokens)` unde:

// - `input` este un string ce poate sa contina tokenuri (Example: "Hello ${user}" or "Hello")

// - `tokens` este un vector ce contine obiecte cu numele si valoarea tokenurilor.

// - Toate tokenurile sunt idetificate sub urmatorul format: `${tokenName}`

// # Completati urmatoarele taskuri:

// - `input` trebuie sa fie de tip `string`. Daca alt tip de date este pasat aruncati `Error` cu mesajul `Input should be a string`; (0.5 pts)

// - `input` trebuie sa aiba cel putin 6 caractere. Daca lungimea `inputului` este mai mica de 6, aruncati `Error` cu mesajul `Input should have at least 6 characters`; (0.5 pts)

// - `tokens` este un vector de elemente cu urmatorul format: `{tokenName: string, tokenValue: string}`. Daca acest format nu este respectat, aruncati `Error` cu urmatorul mesaj `Invalid array format`; (0.5 pts)

// - Daca `input-ul` nu contine niciun token trebuie sa returnati valoarea initiala a `inputului`; (0.5 pts)

// - Daca `input` contine tokenuri, inlocuiti-le cu valorile specifice si returnati noul `input`; (0.5 pts)

Rezolvare:

function textProcessor(input, tokens){

    if(!(input instanceof String || typeof input === 'string')){

        throw new Error('Input should be a string');

    }

    if(input.length <6) {

        throw new Error('Input should have at least 6 characters');

    }

    if(!Array.isArray(tokens)){

        throw new Error('Invalid array format!');

    } else {

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

            if(!(tokens[i].tokenName instanceof String || typeof tokens[i].tokenName === 'string') ||

            !(tokens[i].tokenValue instanceof String || typeof tokens[i].tokenValue === 'string') ){

                throw new Error('Invalid array format!');

            }

            else {

             let index =  input.search(tokens[i].tokenName);

             if(index !== -1){

               input = input.split('${'+tokens[i].tokenName+'}').join(tokens[i].tokenValue);

               //input = input.replace('${'+tokens[i].tokenName+'}',tokens[i].tokenValue);

             }

            }

        }

    }

    return input;

}

const app = {

    textProcessor: textProcessor

};

module.exports = app;

Rezolvare netestata:

function textProcessor(input, tokens){

    if(typeof input=="string")

    {

        if(input.length>=6)

        {

            tokens.forEach((e)=>

            {

                if( typeof e.tokenName=="string" && typeof e.tokenValue=="string" )

                {

                    var t= "${"+ e.tokenName + "}";

                    input= input.replace(t, e.tokenValue);

                }

                else

                {

                    throw new Error("Invalid array format");

                }

            });

            return input;

        }

        else

        {

        throw new Error("Input should have at least 6 characters");

        }

    }

    else

    {

        throw new Error("Input should be a string");

    }

}

const app = {

    textProcessor: textProcessor

};

module.exports = app;

13.

- funcția distance primește ca parametrii două array-uri

 - fiecare element poate apărea cel mult o dată într-un array; orice apariții suplimentare sunt ignorate

 - distanța dintre cele 2 array-uri este numărul de elemente diferite dintre ele

 - dacă parametrii nu sunt array-uri se va arunca o excepție ("InvalidType")

Rezolvare:

function distance(first, second){

    if(!(Array.isArray(first) || first instanceof Array) || !(Array.isArray(second) || second instanceof Array) ){

        throw new Error("InvalidType");

    }

    else {

        let dist = 0;

        let vectorCuElementelePrinCareAmTrecut = [];

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

            if(second.indexOf(first[i]) === -1 && vectorCuElementelePrinCareAmTrecut.indexOf(first[i]) === -1 ){

                dist = dist +1;

                vectorCuElementelePrinCareAmTrecut.push(first[i]);

            }

        }

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

            if(first.indexOf(second[i]) === -1 && vectorCuElementelePrinCareAmTrecut.indexOf(second[i]) === -1 ){

                dist = dist +1;

                vectorCuElementelePrinCareAmTrecut.push(second[i]);

            }

        }

        return dist;

    }

    //TODO: implementați funcția

    // TODO: implement the function

}

module.exports.distance = distance

Rezolvare neverificata:

function distance(first, second){

    if(!(first instanceof Array) || !(second instanceof Array)){

        throw new Error('InvalidType');

    }

    else{

        var duplicates = 0;

        var f = new Set(first);

        var s = new Set(second);

        first = [...f];

        second=[...s];

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

            for(var j=0; j<second.length;j++){

                if(first[i]===second[j]){

                    duplicates++;

                }

            }

        }

        var dist=(first.length+second.length)-(2\*duplicates);

        if(first.length==0 && second.length==0){

            return 0;

        }

        return dist;

    }

}

module.exports.distance = distance

capitalizare

14.

// # Dată fiind funcția `function processString(input)`, care inițial tokenizează string-ul `input` în mai multe `tokens` separate de spațiu, rezolvați următoarele cerințe:

// - Dacă oricare `token` nu este un `number` sau un `Number`, funcția ar trebui să arunce `Error`

// - Dacă oricare `token` nu este un `number` sau un `Number`, funcția ar trebui să arunce `Error` cu mesajul `Item is not a number`; (0.5 pts)

// - Dacă `input` are lungime 0 funcția ar trebui să returneze 100; (0.5 pts)

// - Token-urile `impare` sunt ignorate; (0.5 pts)

// - Funcția returnează 100 minus suma tuturor `token`-urilor pare; (0.5 pts)

Rezolvare:

function processString(input){

    var rezultat =100;

    var tokens = input.split(' ');

    if(input.length === 0){

        return 100;

    }

    for(let i=0;i<tokens.length;i++){

        if(isNaN(tokens[i])){

            throw new Error('Item is not a number');

        } else {

            let token = Number(tokens[i]);

            if(token % 2 ===0){

                rezultat-=token;

            }

        }

    }

    return rezultat;

    // TODO

}

const app = {

    processString: processString

}

module.exports = app

Rezolvare netestata:

function processString(input){

    if(input.length==0){

        return 100;

    }else{

        let sir=input.split(' ');

        let sum=0;

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

            if(!isNaN(sir[i])){

                if(parseInt(sir[i])%2==0){

                    sum+=parseInt(sir[i]);

                }

            }else{

                console.log(typeof parseInt(sir[i]));

                throw new Error("Item is not a number");

            }

        }

        return 100-sum;

    }

}

const app = {

    processString: processString

}

module.exports = app

15.

# Subiect 2 (2.5 pts)

# Tematica: Javascript

# Avand urmatoarea functie `function getAverageGrade(grades)`, completati urmatoarele taskuri:

- Funcția aruncă o eroare dacă există o notă negativă (0.5 pts)

- Funcția aruncă o eroare dacă nota nu este fie un număr fie 'A'; (0.5 pts)

- Funcția aruncă o eroare dacă nota nu este definită; (0.5 pts)

- Functia returnează nota medie; (0.5 pts)

- Funcția returnează nota 0 pentru o listă vidă de note (0.5 pts)

Rezolvare netestata:

function getAverageGrade(gradeItems){

     var sum=0;

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

    {

        if(typeof gradeItems[i].grade !=="number"  ||  gradeItems[i].grade=='A' || typeof gradeItems[i].grade ==="undefined" || gradeItems[i].grade <0 )

        {

            throw new Error("Invalid grade");

        }

        else

        {

             sum+=gradeItems[i].grade;

        }

    }

    if(gradeItems.length===0)

    {

        return 0;

    }

    else{

    return  sum/gradeItems.length;

    }

}

const app = {

    getAverageGrade: getAverageGrade

}

module.exports = app

16.

# Subiect 2 (2.5 pts)

# Tematica: Javascript

# Avand definită clasa `Shape` rezolvați următoarele taskuri:

- Dacă `Shape` este instanțiată direct și este apelată metoda `area`, se va arunca un obiect `Error` cu mesajul `not implemented`; (0.5 pts)

- Se va defini clasă `Square` care extinde `Shape`; Un `Square` poate fi instanțiat pe baza unui obiect cu o proprietate `width`; (0.5 pts)

- Dat fiind un `Square`, se va calcula corect aria acestuia; (0.5 pts)

- Se va defini clasă `Circle` care extinde `Shape`; Un `Circle` poate fi instanțiat pe baza unui obiect cu o proprietate `radius`; Dat fiind un `Circle`, se va calcula corect aria acestuia; (0.5 pts)

- Se va defini clasă `Rectangle` care extinde `Shape`; Un `Rectangle` poate fi instanțiat pe baza unui obiect cu o proprietate `width` și o proprietate `height`; Dat fiind un `Rectangle`, se va calcula corect aria acestuia; (0.5 pts)

Rezolvare neterminata:

class Shape{

    constructor(dimensions){

        this.dimensions = dimensions

    }

    area(){

        // TODO

    }

}

// TODO: Square, Circle, Rectangle

const app = {

  Shape: Shape,

  Square : Square,

  Circle : Circle,

  Rectangle : Rectangle

}

module.exports = app

17.

# Subject 2 (2.5 pts)

# TOPIC: Javascript

# Given the classes `Duck` and `RubberDuck` where:

- a `Duck` is constructed based on a string name

- a `Duck` can swim

- a `RubberDuck` can float

- a `RubberDuck` can't swim

# Complete the following tasks:

- `name` should be of type `string` or `String`. If another type is given an `Error` is thrown with the message `name must be string or String`; (0.5 pts)

- if a duck is instructed to swim it will return a string saying it is swimming (e.g. if the duck is named `Donald` then the returned string is `Donald is swimming`); (0.5 pts)

- a rubber duck is both a `Duck` and a `RubberDuck`; (0.5 pts)

- if a rubber duck is instructed to float it will return a string saying it floats (e.g. if the rubber duck is named `Donald` then the returned string is `Donald floats`); (0.5 pts)

- if a rubber duck is instructed to swim it will return a string saying cannot swim (e.g. if the rubber duck is named `Donald` then the returned string is `Donald can't swim, only float`); (0.5 pts)

Rezolvare netestata:

class Duck{

    constructor(name){

        if(name instanceof String || typeof name === 'string'){

            this.name = name

        }

        else throw Error('name must be string or String')

    }

    move(){

        return `${this.name} is moving`

    }

    swim(){

        return `${this.name} is swimming`

    }

}

class RubberDuck extends Duck{

    float(){

        return `${this.name} floats`

    }

    swim(){

        return `${this.name} can't swim, only float`

    }

}

const app = {

    Duck,

    RubberDuck

}

module.exports = app

18.

# Subiect 2 (2.5 pts)

# Tematica: Javascript

# Avand clasa `Queue` din fisierul `index.js` sa se implementeze urmatoarele functionalitati:

- clasa `Queue` trebuie sa contina o proprietate numita `items`, de tip `Array` care va fi initializata cu un vector gol (0.5 pts);

- se va implementa metoda `insert` ce va primi ca parametru un `element` care va fi adaugat in vector, conform principiului cozii;

- metoda `insert` va permite adaugarea in coada doar a elementelor de tip `string` si va arunca o eroare cu textul `Invalid Type` pentru elementele de alt tip.

- se va implementa metoda `extract` ce va returna un `element` din vector, conform principiului cozii;

- in cazul in care vectorul este gol si se apeleaza metoda `extract`, se va arunca o eroare cu textul `Invalid Operation`;

Rezolvare netestata:

class Queue {

 constructor(){

     this.items=[]

 }

 insert=(item)=>{

     if(typeof item ==='string')

        this.items.push(item)

        else throw new Error("Invalid Type")

 }

 extract=()=>{

     if(this.items.length===0){

         throw new Error("Invalid Operation")

     }else{

         return this.items.shift();

     }

 }

}

module.exports = Queue;

20.

Definiți un tip obiectual (Widget)

Funcția decorate adaugă la Widget o metodă numită enhance, care crește mărimea unui widget cu "n"

Dacă parametrul trimis nu este un număr, se aruncă o excepție ("InvalidType")

Metoda funcționează și asupra Widget-urilor deja declarate

Rezolvare netestata:

class Widget {

    constructor(name, size){

        this.name = name

        this.size = size

    }

    getDescription(){

        return `a ${this.name} of size ${this.size}`

    }

}

function decorate(){

    Widget.prototype.enhance = function (n){

        if(typeof n !== 'number'){

            throw new Error("InvalidType")

        } else{

            this.size += n

        }

    }

}

module.exports.decorate = decorate

module.exports.Widget = Widget

21.

Exista un tip obiectual definit (Bird)

Să se definească tipul Penguin.

Un pinguin este un tip copil pentru Bird și are în plus metoda swim(distance)

Un pinguin nu poate fi creat fără un nume de tip string

Un pinguin nu poate să zboare și va spune asta dacă i se cere

Dacă se apelează makeNest, un pinguin va apela metoda părintelui său

Vedeți testele pentru formatul exact al mesajelor

\*/

Rezolvare netestata:

class Bird {

    constructor(name){

        this.name = name

    }

    fly(distance){

        return `${this.name} flies ${this.distance}`

    }

    makeNest(){

        return `${this.name} makes a nest`

    }

}

class Penguin extends Bird{

    constructor(name){

        if(!name || typeof name !== 'string'){

            throw new Error('CreationError')

        }

        super(name)

    }

    swim(distance){

        return `${this.name} swims ${this.distance}`

    }

    fly(distance){

        return `${this.name} is a penguin and cannot fly`

    }

    makeNest(){

        super.makeNest()

    }

}

module.exports.Bird = Bird

// module.exports.Penguin = Penguin

**SIMPLE -de pus date.json in directorul public**

1.

**# Subiect 1 (2.5 pts)**

**# Tematica: Clienți și servere simple**

**# Dat fiind serverul `app.js` și fișierul `index.html` din directorul `public` directory:**

**# Satisfaceți următoarele cerințe:**

- fișierul `index.html`, care conține textul `A simple app` este livrat de server ca conținut static (0.5 pts);

- butonul cu id-ul `load` există în pagină și se poate da click pe el(0.5 pts);

- când se dă click pe butonul cu id-ul `load` se cere lista de `cars` de pe server; mașinile cu culoare `red` sunt încărcate în tabelul cu id `main` cu un `tr` pentru fiecare (0.5 pts);

- tabelul conține câte un `tr` pentru fiecare mașină încărcată (0.5 pts);

- sunt afișate doar mașini cu culoarea `red` (0.5 pts);

Rezolvare:

App.js

const express = require('express')

const app = express()

app.use(express.static('public'))

app.get('/cars', (req, res) => {

    res.status(200).json([{

        name :  'a',

        color : 'red'

    },{

        name :  'b',

        color : 'blue'

    }])

})

module.exports = app

index.html

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>A simple app</title>

    <script>

        document.addEventListener('DOMContentLoaded', (e)=>{

            let but=document.getElementById('load')

            but.addEventListener('click',async(e)=>{

                let prom=await fetch('http:/cars')

                let cars=await prom.json()

                let cont=cars.filter(e=>e.color=='red').map(e=>`<tr><td>${e.name}</td></tr>`)

                document.getElementById('main').innerHTML=cont.join('')

            })

        } )

    </script>

</head>

<body>

    A simple app

    <table id=main></table>

    <input type="button" id="load"/>

</body>

</html>

2.

**# Subiect 1 (2.5 pts)**

**# Tematica: Clienți și servere simple**

**# Dat fiind serverul `server.js` și fișierul `index.html` din directorul `public` directory:**

**# Satisfaceți următoarele cerințe:**

- fișierul `index.html`, care conține textul `A simple app` este livrat de server ca conținut static (0.5 pts);

- butonul cu id-ul `reload` există în pagină și se poate da click pe el(0.5 pts);

- când se dă click pe butonul cu id-ul `reload` nimic in filtru, toate elementele sunt returnate (0.5 pts);

- când se dă click pe butonul cu id-ul `reload` cu valoarea `red` in filtru, elementele roșii sunt returnate (0.5 pts);

- când se dă click pe butonul cu id-ul `reload` cu o valoare in filtru care nu corespunde culorii niciunui element, este returnată o listă vidă (0.5 pts);

Rezolvare:

App.js

const express = require('express')

const cors = require('cors')

const app = express()

app.use(express.static('public'))

app.use(cors())

app.get('/cars', (req, res) => {

    let filter = req.query.filter

    if (!filter){

        res.status(200).json([{

            name :  'a',

            color : 'red'

        },{

            name :  'b',

            color : 'blue'

        }])

    }

    else{

        res.status(200).json([{

            name :  'a',

            color : 'red'

        },{

            name :  'b',

            color : 'blue'

        }].filter((e) => e.color === filter))

    }

})

module.exports = app

index.html

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>A simple app</title>

    <script>

      window.onload = async () => {

        let btn = document.getElementById("reload");

        let filter = document.getElementById("filter");

        let tbl = document.getElementById("main");

        btn.addEventListener("click", async () => {

          {

            load(filter);

          }

        });

        let load = async (filter) => {

          try {

            const response = await fetch(

              `http://localhost:8080/cars?filter=${filter.value}`

            );

            const responseBody = await response.json();

            const data = responseBody;

            let rows = data.map(

              (e) => `

                        <tr>

                            <td>

                                ${e.name}

                            </td>

                            <td>

                                ${e.color}

                            </td>

                        </tr>

                    `

            );

            tbl.innerHTML = rows.join();

          } catch (err) {

            console.warn(err);

          }

        };

        load("");

      };

    </script>

  </head>

  <body>

    A simple app

    <input type="text" placeholder="filter" id="filter" />

    <table id="main"></table>

    <button id="reload">Reload</button>

  </body>

</html>

3.

**# Subiect 1 (2.5 pts)**

**# Tematica: Clienți și servere simple**

**# Dat fiind serverul `server.js` și fișierul `index.html` din directorul `public` directory:**

**# Satisfaceți următoarele cerințe:**

- fișierul `index.html`, care conține textul `A simple app` este livrat de server ca conținut static (0.5 pts);

- butonul cu id-ul `del` există în pagină și se poate da click pe el(0.5 pts);

- la încărcarea paginii sunt încărcate toate elementele în tabelul cu id `main` cu un `tr` pentru fiecare (0.5 pts);

- când se dă click pe butonul `del` se șterg elementele cu numele prezent în text input-ul cu id-ul `name` (0.5 pts);

- elementele cu numele selectat pentru ștergere nu mai apar în tabel (0.5 pts);

Rezolvare:

Index.html

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>A simple app</title>

    <script>

        window.onload = async () => {

            let btn = document.getElementById('del')

            let name = document.getElementById('name')

            btn.onclick = async () => {

                await deleteCar(name.value)

            }

            let deleteCar = async (name) => {

                try{

                    await fetch(`http://localhost:8080/cars/${name}`, {

                        method : 'delete'

                    })

                    load()

                }

                catch(err){

                    console.warn(err)

                }

            }

            let tbl = document.getElementById('main')

            let load = async () => {

                try{

                    let response = await fetch(`http://localhost:8080/cars`)

                    let data = await response.json()

                    let rows = data.map((e) => `

                        <tr id= "${e.name}">

                            <td>

                                ${e.name}

                            </td>

                            <td>

                                ${e.color}

                            </td>

                        </tr>

                    `)

                    //console.warn(rows)

                    tbl.innerHTML = rows.join('')

                }

                catch(err){

                    console.warn(err)

                }

            }

            load()

        }

    </script>

</head>

<body>

    A simple app

    <input type="text" placeholder="name" id="name" />

    <table id=main></table>

    <button id="del">Del</button>

</body>

</html>

App.js

const express = require('express')

const cors= require("cors")

const app = express()

app.use(cors())

app.use(express.static('public'))

app.locals.cars = [{

    name :  'a',

    color : 'red'

},{

    name :  'b',

    color : 'blue'

}]

app.get('/cars', (req, res) => {

    res.status(200).json(app.locals.cars)

})

app.delete('/cars/:name', (req, res) => {

    app.locals.cars = app.locals.cars.filter((e) => e.name !== req.params.name)

    res.status(202).json({message : 'accepted'})

})

module.exports = app

4.

**# Subiect 1 (2.5 pts)**

**# Tematica: Clienți și servere simple**

**# Dat fiind serverul `app.js` și fișierul `index.html` din directorul `public`:**

**# Satisfaceți următoarele cerințe:**

- Fișierul `index.html`, trebuie sa contina un element de tip `paragraf` cu textul `Webtech app` si sa fie furnizat ca si continut static din directorul `public` (0.5 pts);

- Butoanele cu id-urile `load` si `delete` există în pagină și nu se afla intr-o stare disabled (0.5 pts);

- La apasarea butonului `load` sunt încărcate toate elementele din fisierul `data.json` în tabelul cu id `table` cu un `tr` pentru fiecare element si 3 `td-uri` pentru fiecare proprietate (0.5 pts);

- Input-ul de tip `text` cu id-ul `name` exista in pagina (0.5pts);

- La apasarea butonului `delete` se șterg elementele cu numele prezent în text input-ul cu id-ul `name` (0.5 pts);

Rezolvare:

App.js

const express = require('express')

const cors= require("cors")

const app = express()

app.use(cors())

app.use(express.static('public'))

module.exports = app;

index.html

<!DOCTYPE html>

<html>

  <head>

    <title>Webtech</title>

  </head>

  <body>

    <table id="table"></table>

    <button id="load">Load</button>

    <button id="delete">Delete</button>

    <input type="text" id="name" />

    <script>

            window.onload = async () => {

              let table = document.getElementById("table");

              let p = document.createElement("p");

              p.innerText = "Webtech app";

              document.body.appendChild(p);

              let btLoad = document.getElementById("load");

              btLoad.addEventListener("click", async () => {

                try {

                  let response = await fetch("data.json");

                  let data = await response.json()

                  for (let element of data) {

                    let tr = document.createElement("tr");

                    let td1 = document.createElement("td");

                    let td2 = document.createElement("td");

                    let td3 = document.createElement("td");

                    tr.setAttribute("id",element.name)

                    td1.innerText = element.name;

                    td2.innerText = element.surname;

                    td3.innerText = element.age;

                    tr.appendChild(td1);

                    tr.appendChild(td2);

                    tr.appendChild(td3);

                    table.appendChild(tr);

                  }

                } catch (e) {

                  console.warn(e);

                }

              });

              let btDelete = document.getElementById("delete");

              btDelete.addEventListener("click",()=>{

                  let nameValue=document.getElementById("name")

                  let value=nameValue.value;

                  let ta=document.getElementById(`${value}`)

                  ta.parentNode.removeChild(ta);

              })

            };

    </script>

  </body>

</html>

5.

**# Subiect 1 (2.5 pts)**

**# Tematica: Clienți și servere simple**

**# Dat fiind serverul `app.js` și fișierul `index.html` din directorul `public` directory:**

**# Satisfaceți următoarele cerințe:**

- fișierul `index.html`, care conține textul `A simple app` este livrat de server ca conținut static (0.5 pts);

- fișierul `profil.json` livrat de catre server respecta structura descrisa in test (0.5 pts)

- pagina `index.html` conține titlu de rang 1 cu textul `Profil Influencer` (0.5 pts)

- detaliile profiluli (name, instagram, youtube) sunt afisate in paragrafe separate in div-ul cu id=content (0.5 pts)

- butonul convert afiseaza numarul de urmaritori pe instagram in milioane (ex: 5M); operatia se realizeaza doar pe client (0.5 pts)

Rezolvare:

Json

 {

        "name": "influencer",

        "instagram": 1000000,

        "youtube": 2000000

    }

App.js

const express = require('express')

const cors= require("cors")

const app = express()

app.use(cors())

app.use(express.static('public'))

module.exports = app

index.html

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>A simple app</title>

    <script>

        window.onload = async () => {

            let btnConvert = document.getElementById('convert')

            let data

            let response

            btnConvert.onclick = () => {

                convert()

            }

            let load = async () => {

                try {

                     response = await fetch(`/profile.json`)

                     data = await response.json();

                    let div = document.getElementById("content");

                    for (let elem in data) {

                        let p = document.createElement('p')

                        p.innerText = data[elem]

                        p.setAttribute('id', elem)

                        div.append(p)

                    }

                }

                catch (err) {

                    console.warn(err)

                }

            }

            let convert = () => {

                let p=document.getElementById("instagram")

                let value=(parseInt(p.innerText)/1000000)

                let newP=value+"M"

                p.innerText=newP;

            }

            load()

        }

    </script>

</head>

<body>

    <h1>Profil influencer</h1>

    <div id="content"></div>

    <input type="button" value="convert" id="convert" onclick="convert()" />

</body>

</html>