# JS Advanced: Exam Preparation 2

**Link to contest**: <https://alpha.judge.softuni.org/contests/js-advanced-regular-exam-21-october-2023/4403>

# Problem 01.Tech Troubleshoot Hub

**Environment Specifics**

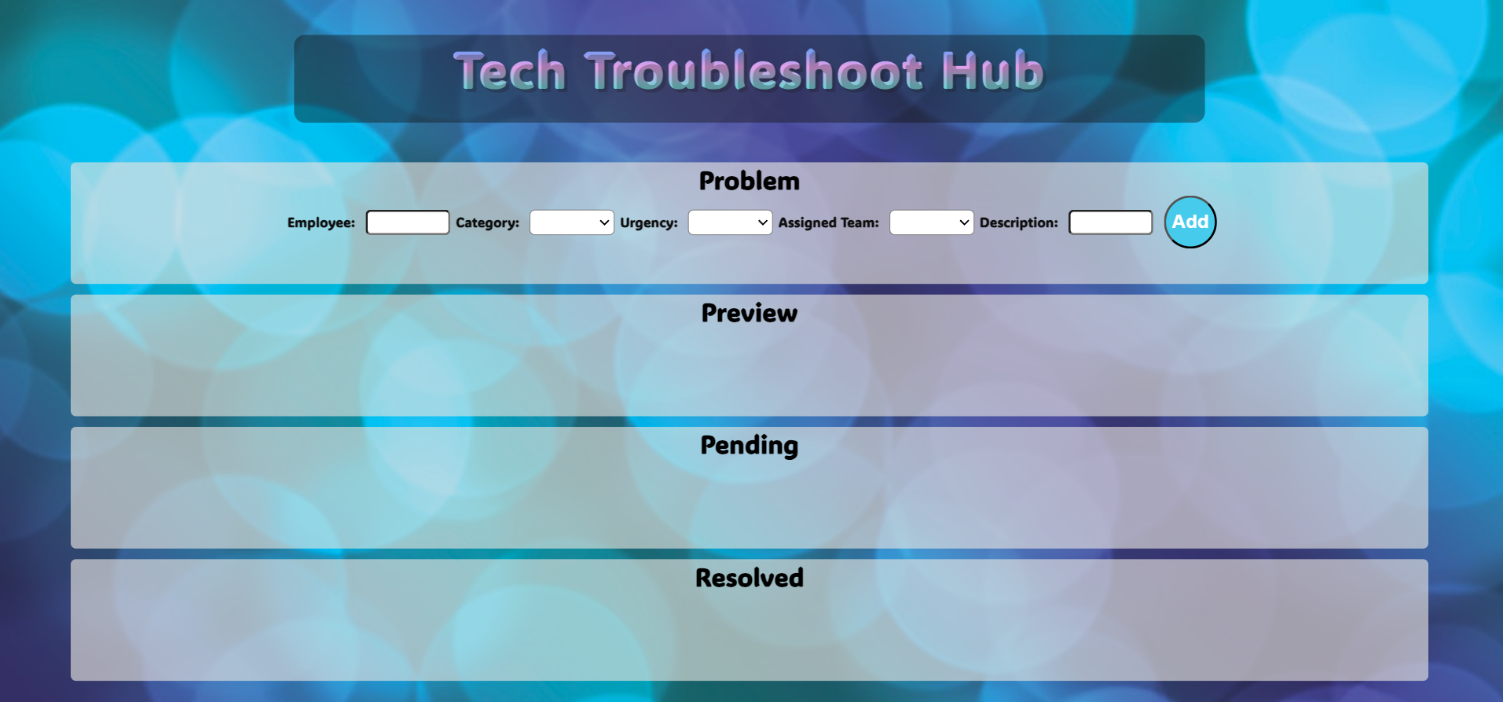
Please be aware that every JS environment may **behave differently** when executing code. Certain things that work in the browser are not supported in **Node.js**, which is the environment used by **Judge**.

The following actions are **NOT** supported:

* **.forEach()** with **NodeList** (returned by **querySelector()** and **querySelectorAll()**)
* **.forEach()** with **HTMLCollection** (returned by **getElementsByClassName()** and **element.children**)
* Using the **spread-operator** (**...**) to convert a **NodeList** into an array
* **append()** in Judge (use only **appendChild()**)
* **prepend()**
* **replaceWith()**
* **replaceAll()**
* **closest()**
* **replaceChildren()**
* Always turn the collection into a **JS array** (forEach, forOf, et.)

If you want to perform these operations, you may use **Array.from()** to first convert the collection into an array.

**Use the provided skeleton to solve this problem.**

**Write the missing functionality** of this user interface. The functionality is divided in the following steps:

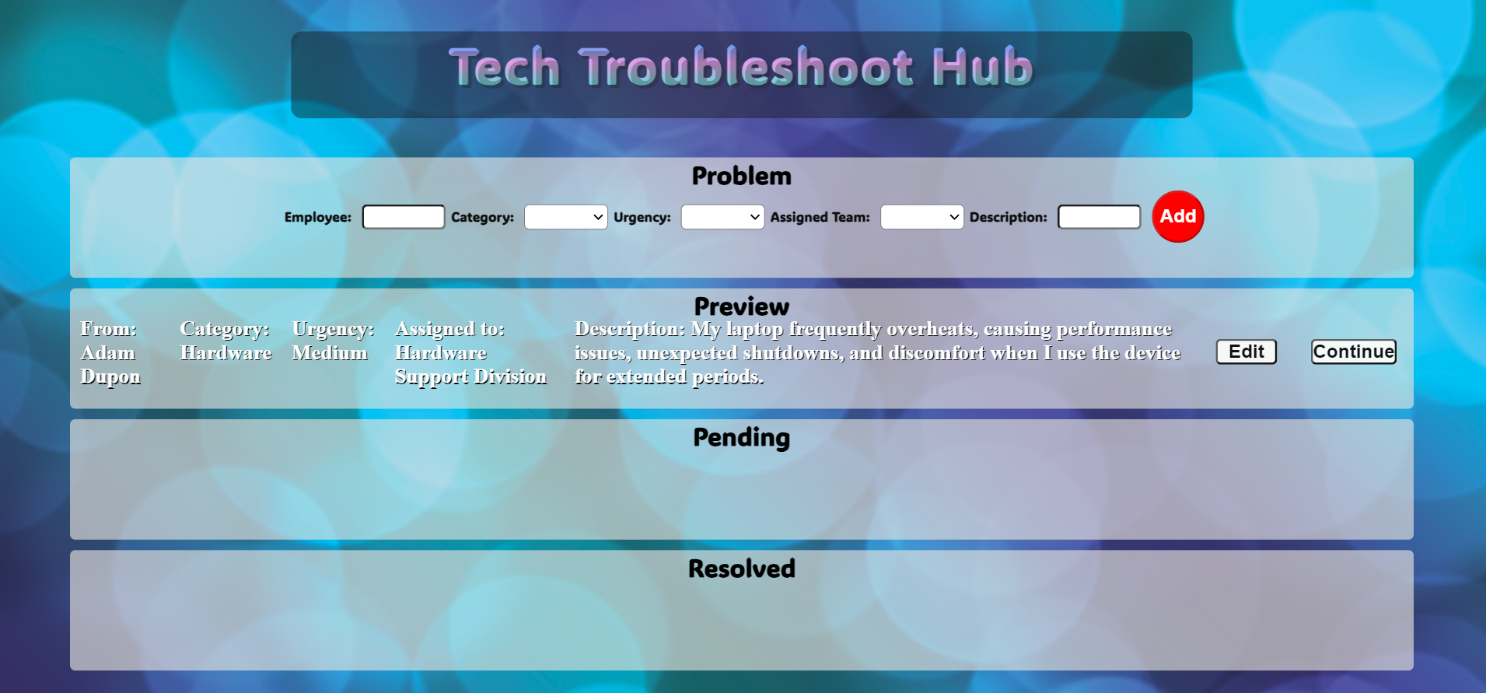
**Your Task**

**Write the missing JavaScript code** to make the **Tech Troubleshoot Hub** work as expected:

All fields **(Employee, Category, Urgency, Assigned Team,** and **Description)** are **filled with the correct input**

* **Employee, Category, Urgency, Assigned Team,** and **Description** are **non**-**empty** **strings**. If any of them is empty, the program should not do anything.

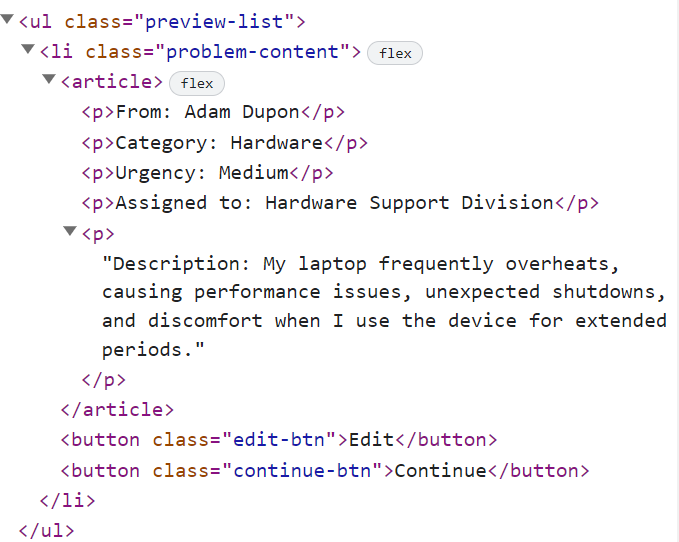
1. **Getting the information from the form**



• On clicking the **["Add"]** button, the information from the input fields is listed in the "**Preview**" section by adding a **list item** to the **".preview-list"** unordered list.

* The list item should follow the same text format and order as shown in the provided picture.
* Upon clicking the button, the **input** fields must be **cleared**, and the **["Add"]** button should be **disabled**. Additionally, the **"Edit"** and **"Continue"** buttons need to be added.

The HTML structure looks like this:



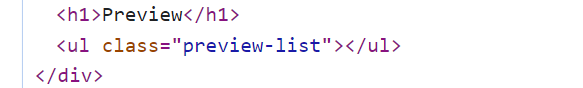
1. **Editing the information**

**The functionality here is the following:**

* **When the "**Edit" **button is clicked, all of the information is loaded in the input fields from step 1 and all buttons in Preview section are removed while the** ["Add"] **button is enabled again.**

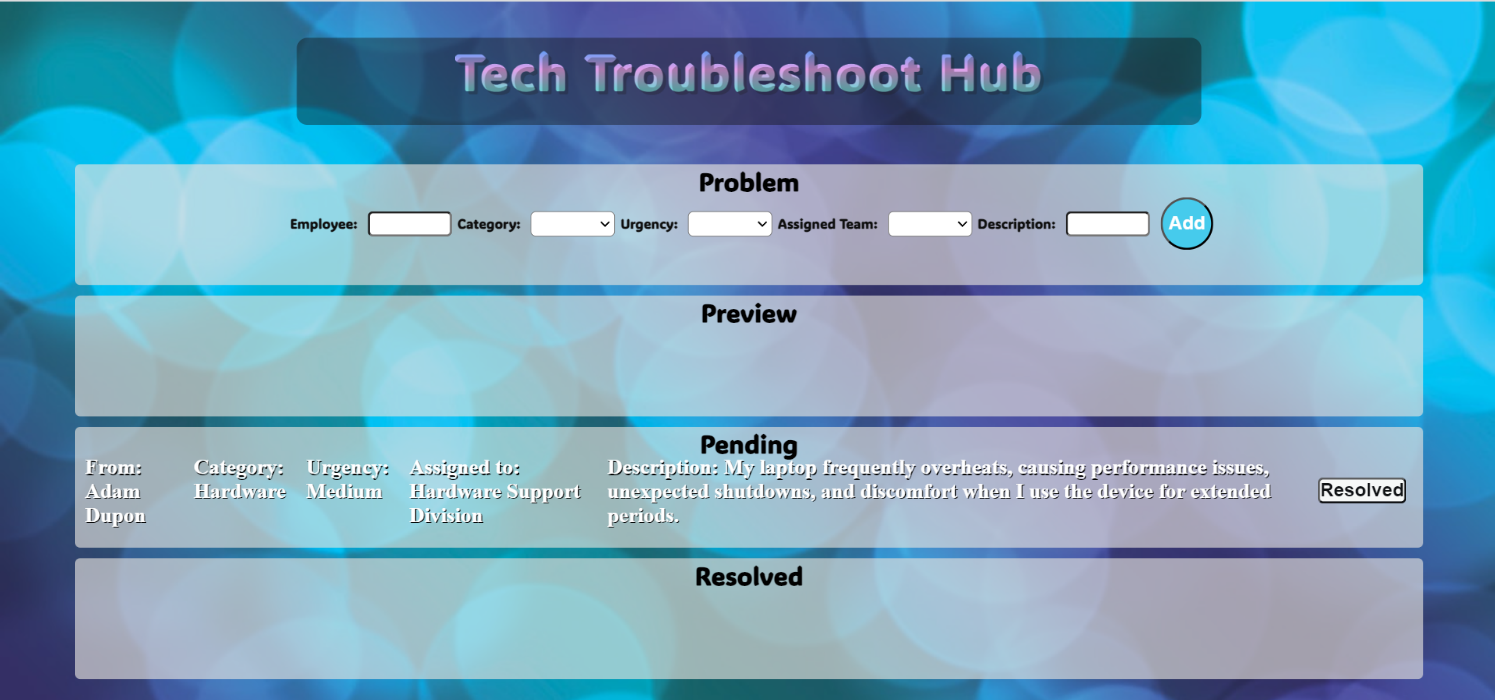
****

* **The list items must be removed from the** "preview-list" and **all of the information must go back to the input fields again**.

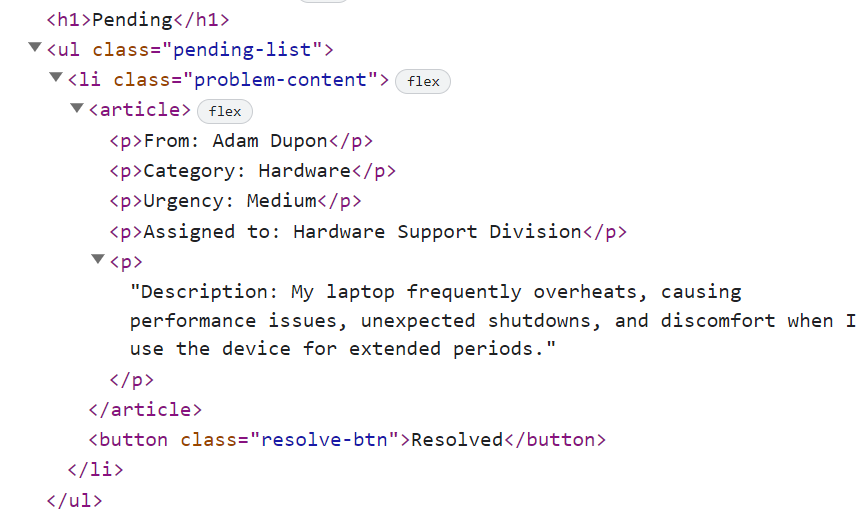
****

1. **Continue to pending**

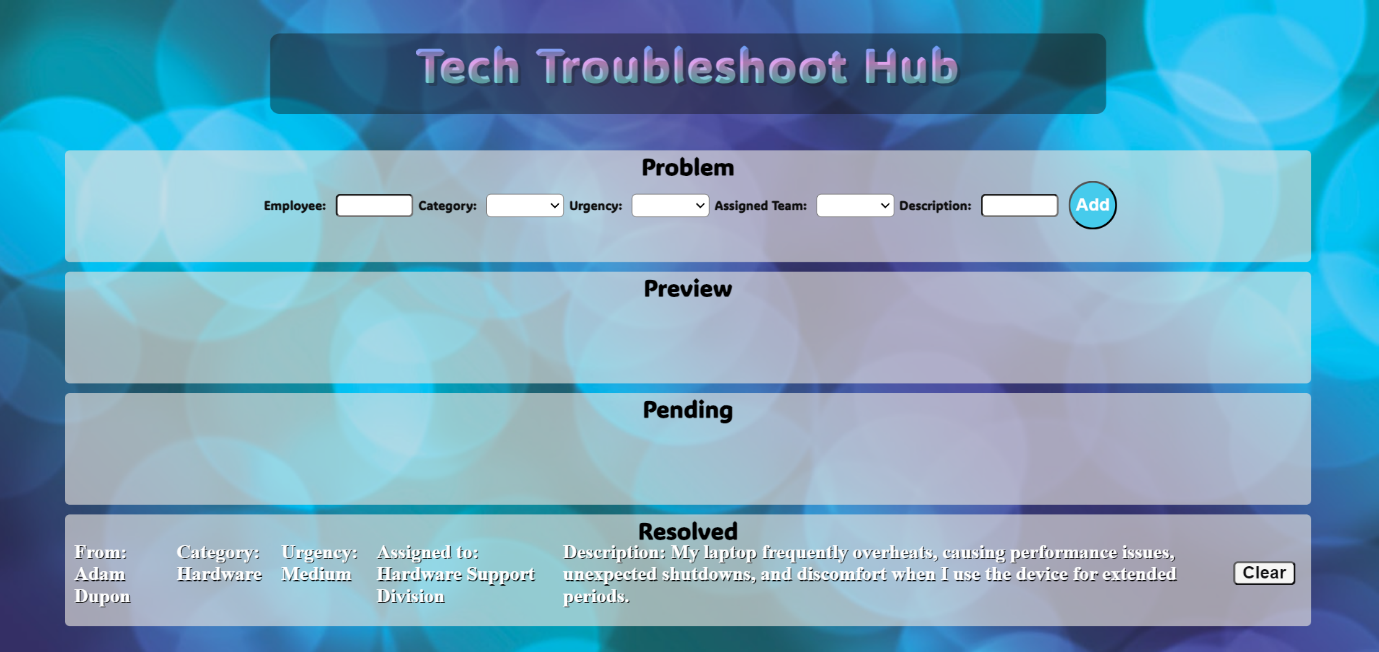
* **When the** "Continue" **button is clicked,** **the information from** "preview-list" unordered list must be transferred to "pending-list" in the same HTML structure**. For you, this means removing everything inside of the ul with class =** "preview-list" **and adding in** "pending-list", **the list item with same information** and the "Resolved" button must be **added.**

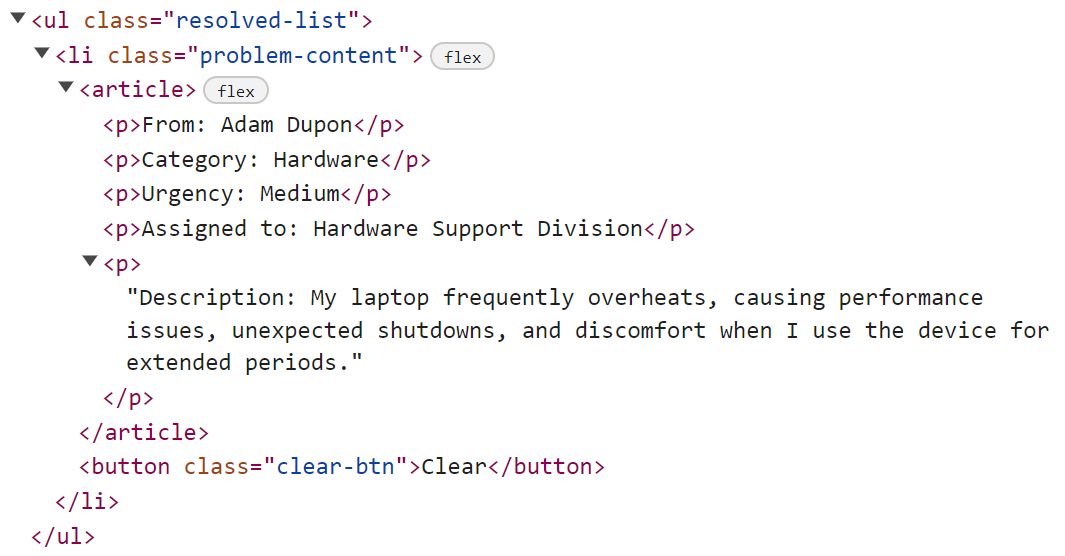
****

* **This is HTML** structure **of** "pending-list" unordered list**:**



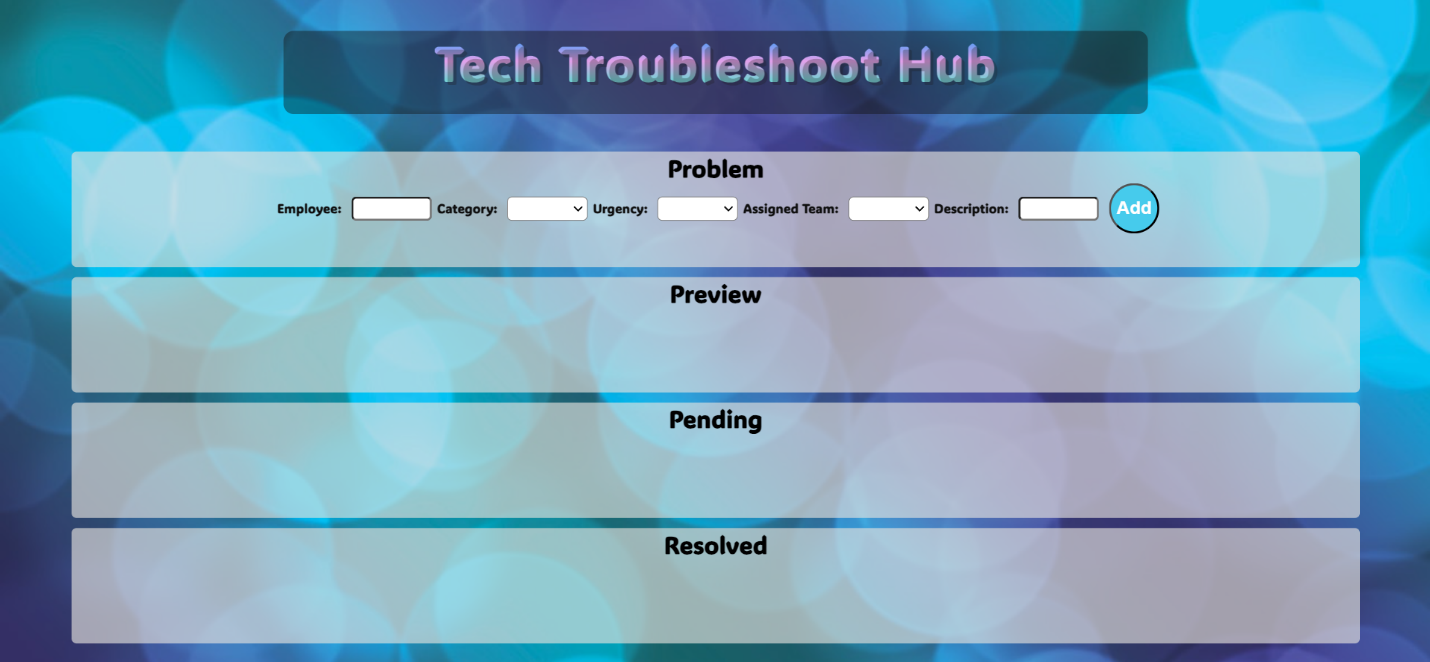
1. **Add to resolved**

* **When the** "Resolved" **button is clicked,** **the information from** "pending-list" unordered list must be transferred to "resolved-list" in the same HTML structure**. For you, this means removing everything inside of the ul with class =** "pending-list" **and adding in** "resolved-list", **the list item with same information** and the "Clear" button must be **added.**
* **This is HTML** structure **of** "resolved-list" unordered list**:**



1. **Clear information**

* **When the** "Clear" **button is clicked,** **the list item must be removed, from the** "resolved-list.
* "Clear" button:

****

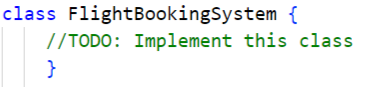
## Submission

Submit only yours **solution()** function.

# Problem 02. Flight Booking System

*You are developing a Flight Booking System for a travel agency. Create a class called "FlightBookingSystem" to manage flight bookings and passenger records.*

## Flight Booking System



Implement a **class FlightBookingSystem**, which supports the functionality described below.

### Functionality

### Constructor

The constructor has **4** properties:

* **agencyName - a string**
* **flights - an empty array**
* **bookings - an empty array**
* **bookingsCount – 0**

**At the initialization** of the **FlightBookingSystem** class, the **constructor** accepts only the **agencyName!**

### Methods

#### addFlight (flightNumber, destination, departureTime, price)

This method adds a new flight to the system. It accepts 4 arguments:

* **flightNumber - a unique string identifier for the flight.**
* **destination - a string representing the flight's destination.**
* **departureTime - a string representing the departure time.**
* **price - a number representing the ticket price.**

If a flight with the same flight number already exists in the flights array, **return** the following message:

**`Flight ${flightNumber} to ${destination} is already available.`**

Otherwise, add the new flight to the flights array as an object and **return** the following message:

**`Flight ${flightNumber} to ${destination} has been added to the system.`**

#### bookFlight (passengerName, flightNumber)

This method allows a passenger to book a flight. It accepts 2 arguments:

* **passengerName - a string representing the name of the passenger.**
* **flightNumber - a string representing the flight number for booking.**

If the flight with the specified flight number is not found in the flights array, **return** the following message:

**`Flight ${flightNumber} is not available for booking.`**

Otherwise, add the booking to the bookings array as an object and **increment** **bookingsCount** by **1**. **Return** the following message:

**`Booking for passenger ${passengerName} on flight ${flightNumber} is confirmed.`**

#### cancelBooking (passengerName, flightNumber)

This method allows a passenger to cancel a flight booking. It accepts 2 arguments:

* **passengerName - a string representing the name of the passenger.**
* **flightNumber - a string representing the flight number to cancel.**

If the booking with the specified passenger name and flight number is not found in the bookings array, **throw error** with the following message:

**`Booking for passenger ${passengerName} on flight ${flightNumber} not found.`**

Otherwise, remove the booking from the bookings array, decrement **bookingsCount** by **1** and **return** the following message:

**`Booking for passenger ${passengerName} on flight ${flightNumber} is cancelled.`**

#### showBookings (criteria)

Accept 1 argument:

* **criteria – a string representing the booking criteria ("all", "cheap", "expensive").**

This method **returns** information based on the booking **criteria**:

If the bookings array is **empty**, **throw error** with the following message:

**`No bookings have been made yet.`**

If the criteria is "**all**", **return** a list of all bookings in the following format:

* On first line show the following message:

**`All bookings(${bookingsCount}):`**

* On the following lines, display information about each booking:

**`${passengerName} booked for flight ${flightNumber}.`**

If the criteria is "**cheap**":

If there are no "**cheap**" flights, **return** :

**"No cheap bookings found."**

Else**, return** a list of all bookings under or exactly the price of $100 in the following format:

* On first line show the following message:

**"Cheap bookings:"**

* On the following lines, display information about each booking:

**`${passengerName} booked for flight ${flightNumber}.`**

If the criteria is "**expensive**":

If there are no "**expensive**" flights, **return** :

**"No expensive bookings found."**

Else**, return** a list of all bookings over the price of $100 in the following format:

* On first line show the following message:

**"Expensive bookings:"**

* On the following lines, display information about each booking:

**`${passengerName} booked for flight ${flightNumber}.`**

#### Examples

|  |
| --- |
| **Input 1** |
| const system = new FlightBookingSystem("TravelWorld");    console.log(system.addFlight("AA101", "Los Angeles", "09:00 AM", 250));    console.log(system.addFlight("BB202", "New York", "10:30 AM", 180));    console.log(system.addFlight("CC303", "Chicago", "11:45 AM", 120));    console.log(system.addFlight("AA101", "Los Angeles", "09:00 AM", 250)); |

|  |
| --- |
| **Output 1** |
| Flight AA101 to Los Angeles has been added to the system.  Flight BB202 to New York has been added to the system.  Flight CC303 to Chicago has been added to the system.  Flight AA101 to Los Angeles is already available. |

|  |
| --- |
| **Input 2** |
| const system = new FlightBookingSystem("TravelWorld");    console.log(system.addFlight("AA101", "Los Angeles", "09:00 AM", 250));    console.log(system.addFlight("BB202", "New York", "10:30 AM", 180));    console.log(system.bookFlight("Alice", "AA101"));    console.log(system.bookFlight("Bob", "BB202"));    console.log(system.bookFlight("Charlie", "CC303")); |

|  |
| --- |
| **Output 2** |
| Flight AA101 to Los Angeles has been added to the system.  Flight BB202 to New York has been added to the system.  Booking for passenger Alice on flight AA101 is confirmed.  Booking for passenger Bob on flight BB202 is confirmed.  Flight CC303 is not available for booking. |

|  |
| --- |
| **Input 3** |
| const system = new FlightBookingSystem("TravelWorld");    console.log(system.addFlight("AA101", "Los Angeles", "09:00 AM", 250));    console.log(system.addFlight("BB202", "New York", "10:30 AM", 180));    console.log(system.bookFlight("Alice", "AA101"));    console.log(system.bookFlight("Bob", "BB202"));    console.log(system.cancelBooking("Alice", "AA101")); |

|  |
| --- |
| **Output 3** |
| Flight AA101 to Los Angeles has been added to the system.  Flight BB202 to New York has been added to the system.  Booking for passenger Alice on flight AA101 is confirmed.  Booking for passenger Bob on flight BB202 is confirmed.  Booking for passenger Alice on flight AA101 is cancelled. |

|  |
| --- |
| **Input 4** |
| const system = new FlightBookingSystem("TravelWorld");    console.log(system.addFlight("AA101", "Los Angeles", "09:00 AM", 250));    console.log(system.addFlight("BB202", "New York", "10:30 AM", 180));    console.log(system.bookFlight("Alice", "AA101"));    console.log(system.bookFlight("Bob", "BB202"));    console.log(system.showBookings("all")); |

|  |
| --- |
| **Output 4** |
| Flight AA101 to Los Angeles has been added to the system.  Flight BB202 to New York has been added to the system.  Booking for passenger Alice on flight AA101 is confirmed.  Booking for passenger Bob on flight BB202 is confirmed.  All bookings(2):  Alice booked for flight AA101.  Bob booked for flight BB202. |

|  |
| --- |
| **Input 5** |
| const system = new FlightBookingSystem("TravelWorld");    console.log(system.addFlight("AA101", "Los Angeles", "09:00 AM", 250));    console.log(system.addFlight("BB202", "New York", "10:30 AM", 180));    console.log(system.bookFlight("Alice", "AA101"));    console.log(system.bookFlight("Bob", "BB202"));    console.log(system.showBookings("expensive"));    console.log(system.showBookings("cheap")); |

|  |
| --- |
| **Output 5** |
| Flight AA101 to Los Angeles has been added to the system.  Flight BB202 to New York has been added to the system.  Booking for passenger Alice on flight AA101 is confirmed.  Booking for passenger Bob on flight BB202 is confirmed.  Expensive bookings:  Alice booked for flight AA101.  Bob booked for flight BB202.  No cheap bookings found. |

# Problem 3. Online Store

**Your Task**

Using Mocha and Chai, write JavaScript unit tests to test an object named **onlineStore**. You may use the following code as a template:

|  |
| --- |
| describe(**"*Tests* …"**, **function**() {  describe(**"*TODO* …"**, **function**() {  ***it***(**"*TODO …*"**, **function**() {  *//* ***TODO:*** …  });  });  *//* ***TODO:*** …  }); |

The **onlineStore** object represents an online shopping platform and contains the following functionality:

**isProductAvailable(product, stockQuantity) -**  A function that accepts two parameters: a **string** representing a product and a **number** representing the stock quantity.

* If the **stockQuantity** is **less** than or **equal** to 0, and the product is considered out of stock, the function should **return** a message:

**`Sorry, ${product} is currently out of stock.`**

* If the **stockQuantity** is **greater** than 0, the product is available, and the function should return:

**`Great! ${product} is available for purchase.`**

There is a need for validation for the input, the **product** parameter should be an **string**, and the **stockQuantity**  should be a **number**. In case of invalid parameters, the function should **throw an error**:

**"Invalid input."**

* **canAffordProduct(productPrice, accountBalance)-** A function that accepts two parameters: a **number** representing the product price and a **number** representing the account balance.
* The function should calculate if the user can afford to buy the product by **subtracting** the **product** price from the account **balance**.
* If the result is **less** than 0, the user doesn't have enough funds, and the function should **return**:

**"You don't have sufficient funds to buy this product."**

* If the result is **greater** than or **equal** to 0, the purchase is successful, and the function should **return**:

**`Product purchased. Your remaining balance is $${remainingBalance}.`**

* You need to validate the input; if **productPrice** and **accountBalance** are **not** numbers, the function should **throw an error:**

**"Invalid input."**

* **getRecommendedProducts(productList, category)** A function that accepts two parameters: an **array** of **objects** representing products and a **string** representing a category.
  + The **productList** array stores objects with product names and categories (e.g., **[{ name: "Camera", category: "Photography" }, ...])**.
* The function should find and **return** product names that match the specified **category** in the format:  
  **`Recommended products in the ${category} category: ${product}, ${product}`**
  + If there are no recommended products in the specified category, the function should return:

**`Sorry, we currently have no recommended products in the ${category} category.`**

There is a need for validation for the input, the **productList** parameter should be an **array**, and the **category** should be a **string**. In case of invalid parameters, the function should **throw an error**:

**"Invalid input."**

**JS Code**

To ease you in the process, you are provided with an implementation that meets all of the specification requirements for the **onlineStore** object:

|  |
| --- |
| onlineStore.js |
| const onlineStore = {      isProductAvailable(product, stockQuantity) {      if (typeof product !== "string" || typeof stockQuantity !== "number") {              throw new Error("Invalid input.");          }          if (stockQuantity <= 0) {          return `Sorry, ${product} is currently out of stock.`;        } else {          return `Great! ${product} is available for purchase.`;        }      },      canAffordProduct(productPrice, accountBalance) {        if (typeof productPrice !== "number" || typeof accountBalance !== "number") {          throw new Error("Invalid input.");        }          let remainingBalance = accountBalance - productPrice;          if (remainingBalance < 0) {          return "You don't have sufficient funds to buy this product.";        } else {          return `Product purchased. Your remaining balance is $${remainingBalance}.`;        }      },      getRecommendedProducts(productList, category) {        let recommendedProducts = [];          if (!Array.isArray(productList) || typeof category !== "string") {          throw new Error("Invalid input.");        }          productList.forEach((product) => {          if (product.category === category) {            recommendedProducts.push(product.name);          }        });          if (recommendedProducts.length === 0) {          return `Sorry, we currently have no recommended products in the ${category} category.`;        } else {          return `Recommended products in the ${category} category: ${recommendedProducts.join(", ")}`;        }      },    }; |

**Submission**

Submit your tests inside a **describe()** statement, as shown above.