JS Advanced: Exam Preparation 2

Problem 1. Car Parts

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**.

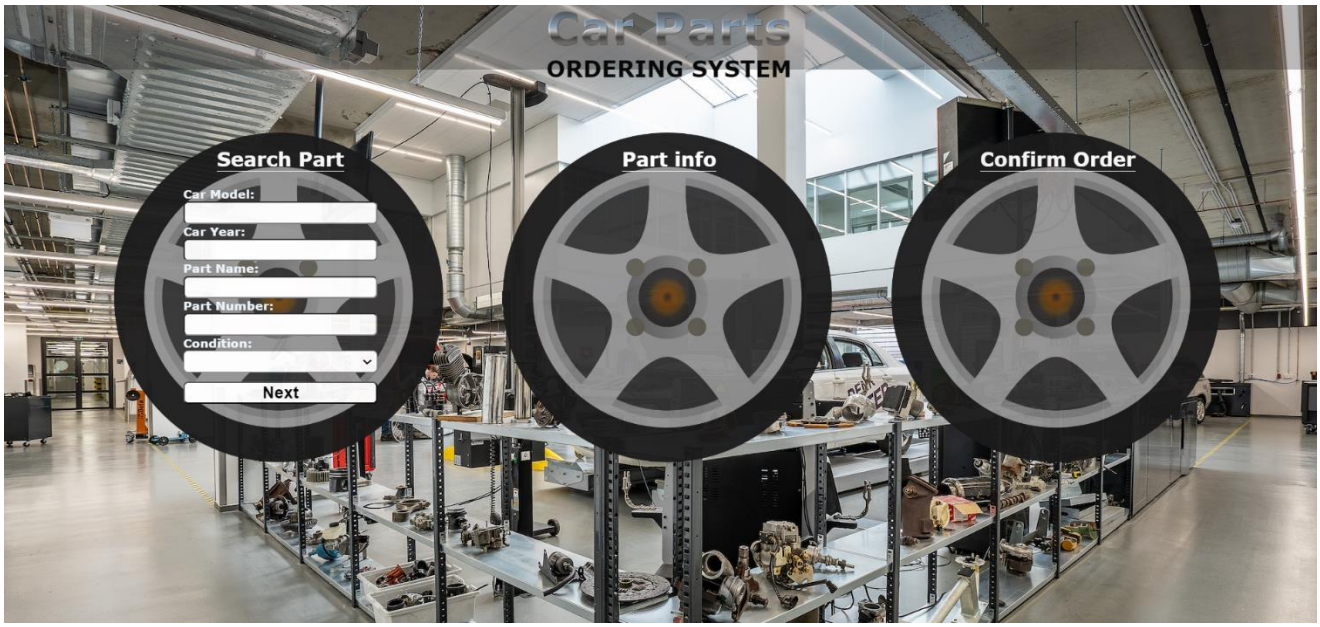
The following actions are **NOT** supported:

- `.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:



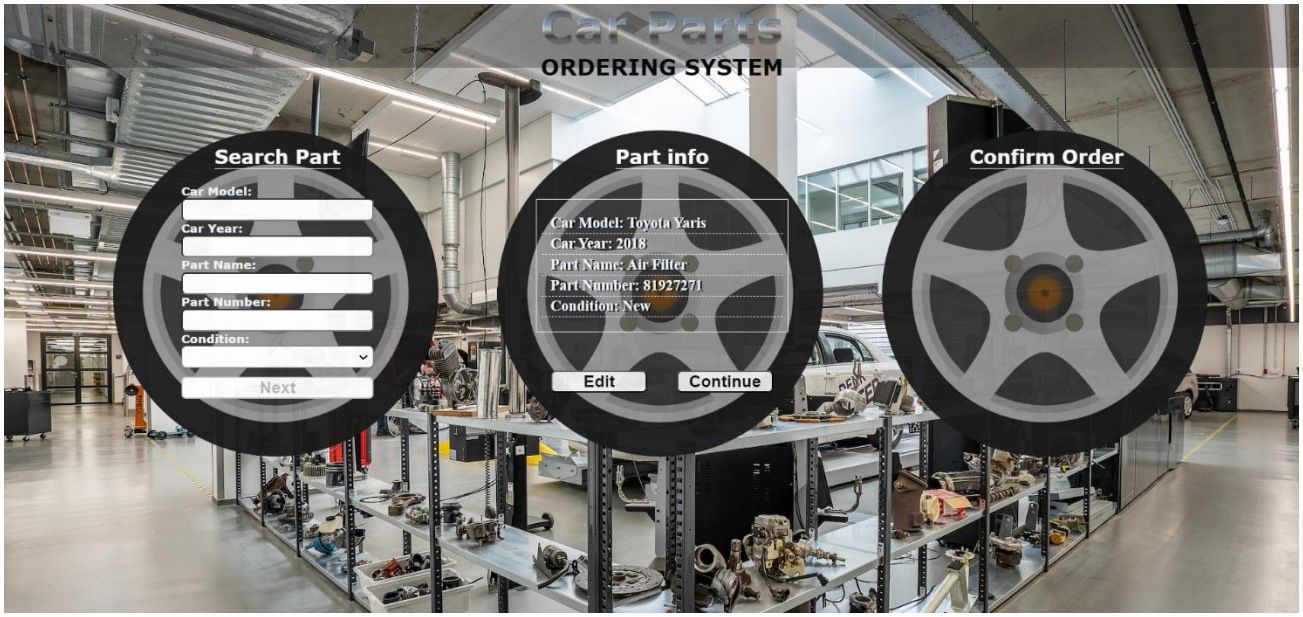
Your Task

Write the missing JavaScript code to make the **Car Parts** work as expected:

All fields **(Car Model, Car Year, Part Name, Part Number,** and **Condition)** are **filled with the correct input**

- **Car Model, Car Year, Part Name, Part Number,** and **Condition** are **non-empty strings** and **Car Year** must be between **1980** and **2023**. If any of them is empty or invalid, the program should not do anything.

1. Getting the information from the form



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

- To complete the ordering cycle at the end, you need to perform the following actions:
  - Change the **visibility** property of the image element with the ID **"complete-img"** to **"hidden"**.
  - Change the **text content** of the paragraph element with the ID **"complete-text"** to an empty string. These actions are necessary to hide the image and clear the text content as is by default in the beggining.
- 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 **["Next"]** button should be **disabled**. Additionally, the **"Edit"** and **"Continue"** buttons need to be added.

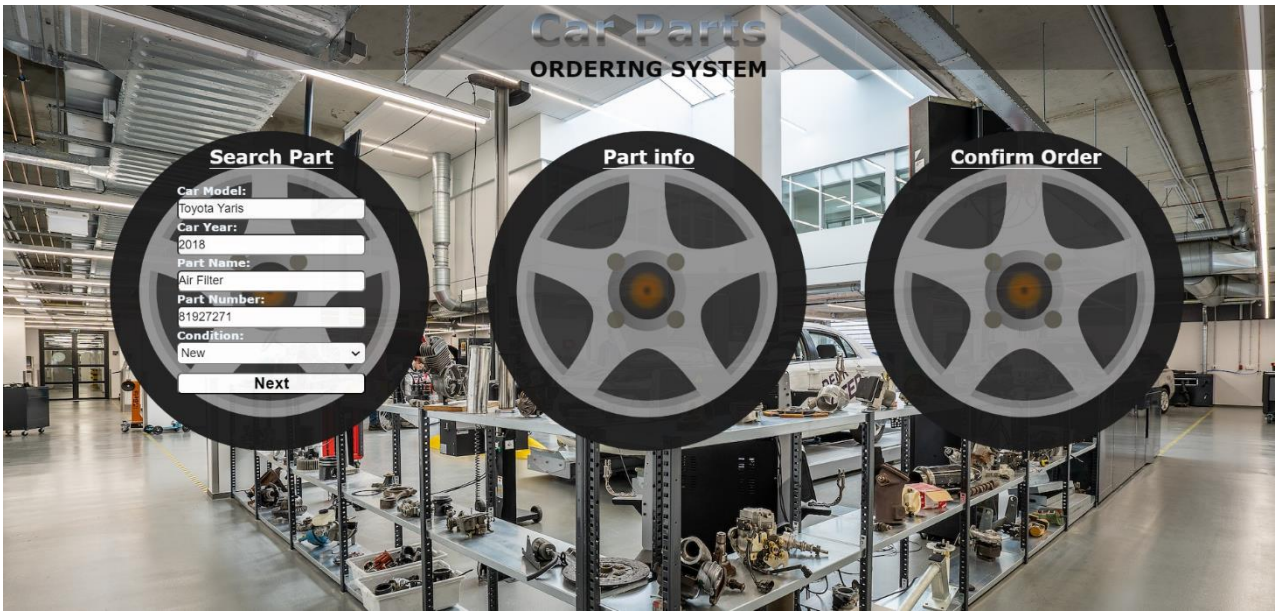
The HTML structure looks like this:

```
<h1>Part info</h1>
<ul class="info-list">
  <li class="part-content">
    <article>
      <p>Car Model: Toyota Yaris</p>
      <p>Car Year: 2018</p>
      <p>Part Name: Air Filter</p>
      <p>Part Number: 81927271</p>
      <p>Condition: New</p>
    </article>
    <button class="edit-btn">Edit</button>
    <button class="continue-btn">Continue</button>
  </li>
</ul>
```

2. Editing 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 **Part info** section are **removed** while the **["Next"]** button is **enabled** again.

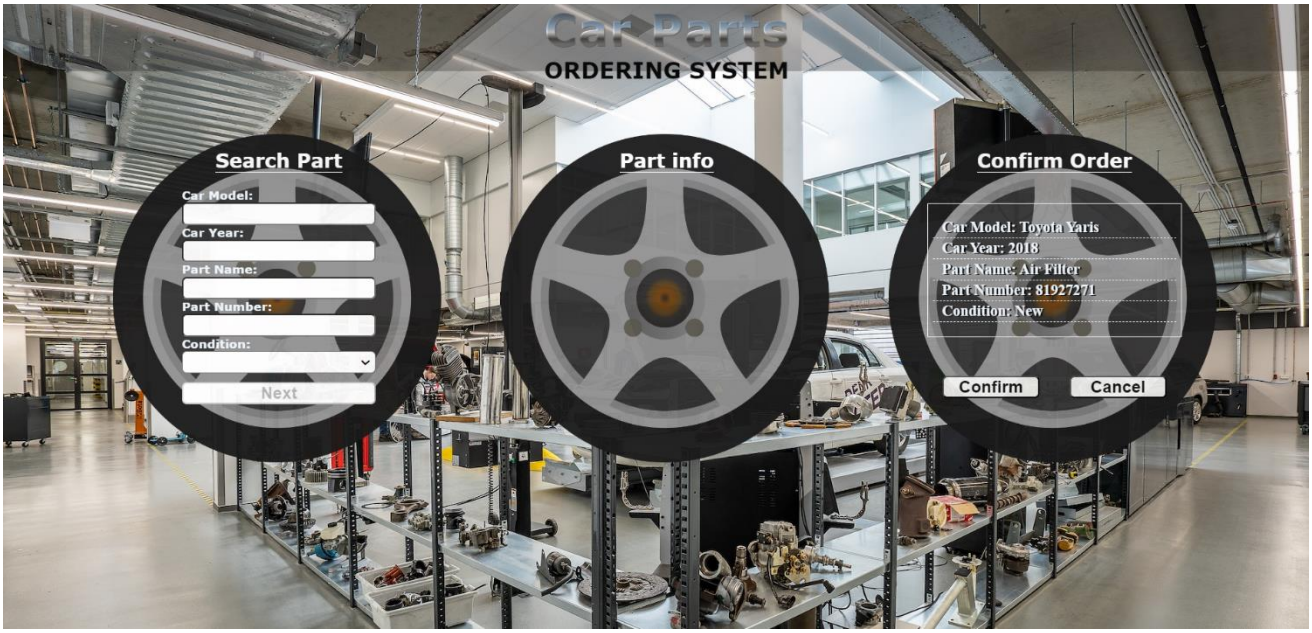


- The **list items** must be **removed** from the **"info-list" and** all of the **information** must go back to the **input fields** again**.**

```
  <h1>Part info</h1>
  <ul class="info-list"></ul>
</div>
```

3. Continue

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

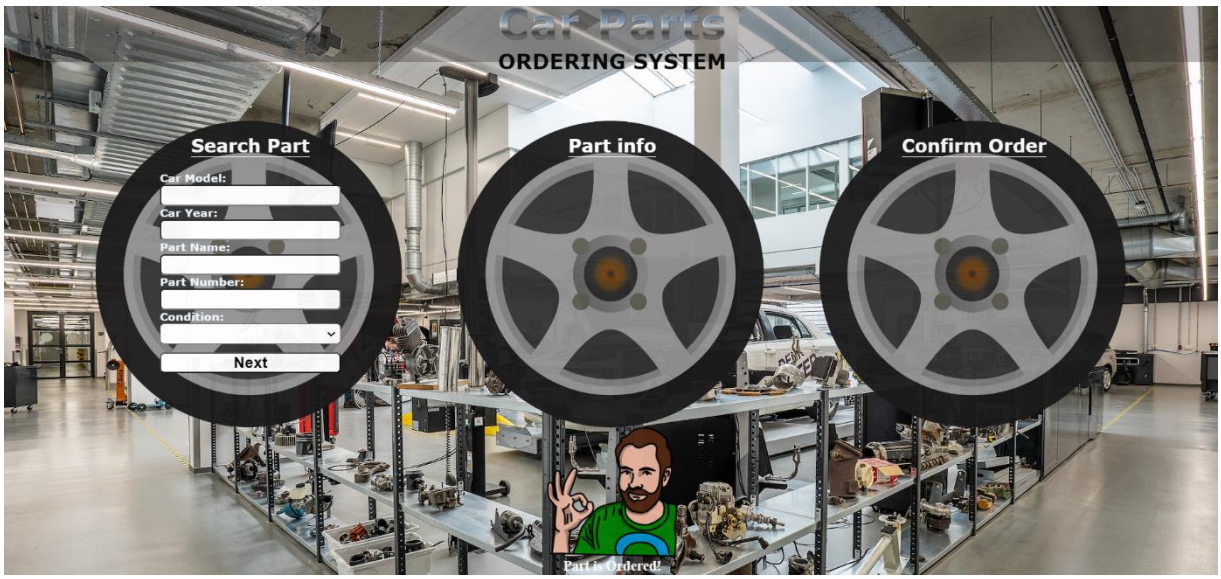


- This is HTML structure of **"confirm-list"** unordered list:

```
<h1>Confirm Order</h1>
<ul class="confirm-list">
  <li class="part-content">
    <article>
      <p>Car Model: Toyota Yaris</p>
      <p>Car Year: 2018</p>
      <p>Part Name: Air Filter</p>
      <p>Part Number: 81927271</p>
      <p>Condition: New</p>
    </article>
    <button class="confirm-btn">Confirm</button>
    <button class="cancel-btn">Cancel</button>
  </li>
</ul>
```

4. Confirm

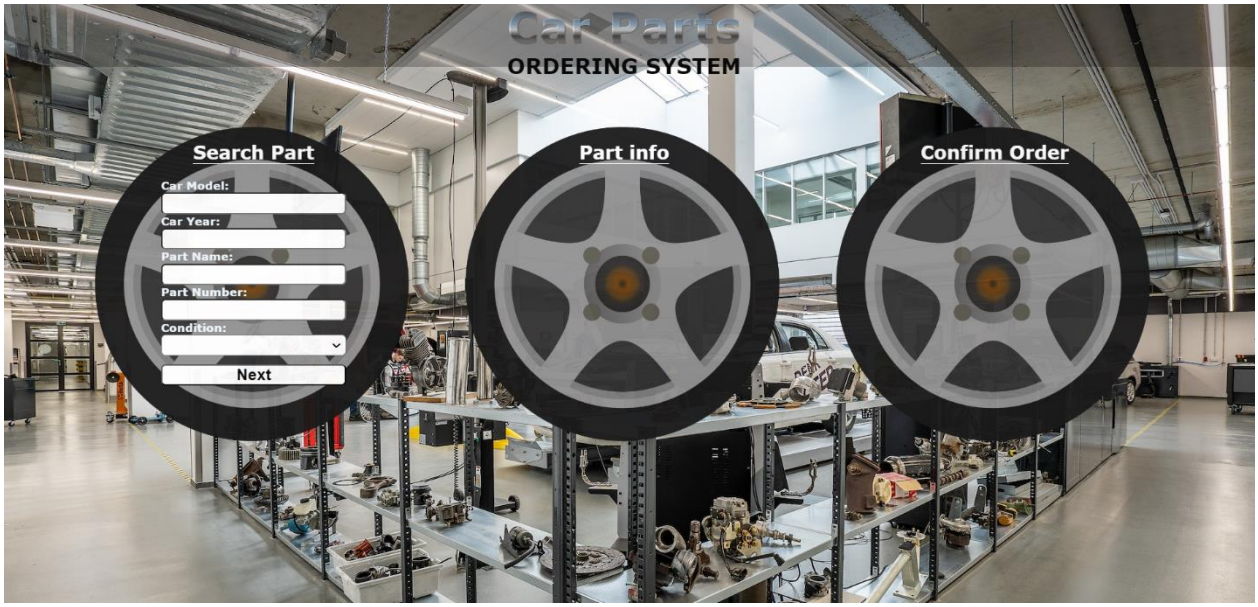
- When the **"Confirm"** button is clicked, the **list item** must be **removed,** from the **"confirm-list",** the **["Next"]** button is **enabled** again. You must change the **visibility** property of image element with **id "complete-img"** to **"visible"** and **add** text content **"Part is Ordered!"** to paragraph whit **id "complete-text".**
  "Confirm" button:



5. Cancel

- When the **"Cancel"** button is clicked, the **list item** must be **removed,** from the **"confirm-list",** the **["Next"]** button is **enabled** again.

"Cancel" button:



Submission

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

Problem 2. Job Offers

```
class JobOffers {
    //TODO Implement this class
}
```

Write a **class Job Offers**, which supports the described functionality below:

Functionality

Constructor

Should have these **3** properties:

- `employer - string`
- `position - string`
- `jobCandidates - empty array`

At the initialization of the **JobOffers** class, the **constructor** accepts the **employer** and **possition.**

Hint: You can add more properties to help you finish the task.

jobApplication(candidates)

This method adds people to the candidates list. The method takes **one argument**: **candidates (array of strings)**.

- **Every element** into this array is information about person in the format:

  `"{name}-{education}-{yearsExperience}"`

  - They are **separated** by slash symbol `"-"`. **Example**: [`"John Doe-Bachelor-10", "Peter Parker-Master-5", "Daniel Jones- Bachelor-18"…`]
  - If the **name** of the current person is already present in `jobCandidates` array, update the old `yearsExperience` **only if** the current one is **higher**.
  - Otherwise, should **add** the person, with properties: `{name, education, yearsExperience}` to the `jobCandidates` **array**.
- In **all cases**, you must **finally return a string** in the following format:

  `"You successfully added candidates: {name1}, {name2}, …{nameN}."`

  Note: When returning the **string**, keep in mind that the different **names** of **persons must** be:

  - **Unique** - for instance:
    - `" You successfully added candidates: John Doe, Peter Parker, Daniel Jones "` - is a correctly returned string
    - `" You successfully added candidates: John Doe, Peter Parker, John Doe "` - is not a correctly returned string
  - **Separated** by **comma** and **space** `(, )`

jobOffer(chosenPerson)

With this method, employer can sign contracts with person from the candidates list. The method takes **one argument**: **chosenPerson (string)**.

- The string about the selected player is in the format:

  `"{name}-{minimalExperience}"`
- Check:
  - If the **name** of the current person is not present in `jobCandidates` array, an **error** with the following message should be **thrown**:

"{name} is not in the candidates list!"

- If the **minimalExperience** selected by the employer for a given person **is more** than the value recorded in the array **jobCandidates**, an **error** with the following message should be **thrown**:

"{name} does not have enough experience as {position}, minimum requirement is {minimalExperience} years."

- Otherwise, if the above conditions are not met, you must replace **yearsExperience** with the string "**hired**"
- Finally, you need to **return** the string in the following format:

"Welcome aboard, our newest employee is {name}."

salaryBonus(name)

With this method, we make sure that the candidates have **proper education (Bachelor degree or Master degree)** for the position. The method takes **one argument**:

- **name (string)**

- If the submitted **name** is not present in the **jobCandidates** array, an **error** with the following message should be **thrown**:

"{name} is not in the candidates list!"

- If the education recorded in the **jobCandidates** array is **Bachelor degree, return** the following message:

"{name} will sign a contract for {employer}, as {position} with a salary of $50,000 per year!"

- If the education of the person is **Master degree, return** the following message:

"{name} will sign a contract for {employer}, as {position} with a salary of $60,000 per year!"

- If the education recorded in the **jobCandidates** array is different than **Bachelor degree** or **Master degree**, **return** the following message:

"{name} will sign a contract for {employer}, as {position} with a salary of $40,000 per year!"

candidatesDatabase()

This method checks if there are any candidates in the database, if not **throw an error:**

"Candidate Database is empty!"

- Otherwise, **returns all candidates**, The first line shows the following message:

"Candidates list:"

- On the new line, display information about each candidate sorted in **ascending** order of **name**:

  "{name}-{yearsExperience}"

Example

Input 1
`let Jobs = new JobOffers ("Google", "Strategy Analyst");` `console.log(Jobs.jobApplication(["John Doe-Bachelor-10", "Peter Parker-Master-5", "Daniel Jones- Bachelor-18"]));`

Output 1
You successfully added candidates: John Doe, Peter Parker, Daniel Jones.

Input 2

`let Jobs = new JobOffers ("Google", "Strategy Analyst");`
`console.log(Jobs.jobApplication(["John Doe-Bachelor-10", "Peter Parker-Master-5", "Daniel Jones- Bachelor-18"]));`
`console.log(Jobs.jobOffer("John Doe-8"));`
`console.log(Jobs.jobOffer("Peter Parker-4"));`
`console.log(Jobs.jobOffer("John Jones-8"));`

Output 2
You successfully added candidates: John Doe, Peter Parker, Daniel Jones. Welcome aboard, our newest employee is John Doe. Welcome aboard, our newest employee is Peter Parker. Uncaught Error Error: John Jones is not in the candidates list!

Input 3
`let Jobs = new JobOffers ("Google", "Strategy Analyst");` `console.log(Jobs.jobApplication(["John Doe-Bachelor-10", "Peter Parker-Master-5", "Daniel Jones- Bachelor-18"]));`

```
console.log(Jobs.jobOffer("John Doe-8"));
console.log(Jobs.jobOffer("Peter Parker-4"));
console.log(Jobs.salaryBonus("John Doe"));
console.log(Jobs.salaryBonus("Peter Parker"));
```

Output 3

```
You successfully added candidates: John Doe, Peter Parker, Daniel Jones.
Welcome aboard, our newest employee is John Doe.
Welcome aboard, our newest employee is Peter Parker.
John Doe will sign a contract for Google, as Strategy Analyst with a salary of $50,000 per year!
Peter Parker will sign a contract for Google, as Strategy Analyst with a salary of $60,000 per year!
```

Input 4

```
let Jobs = new JobOffers ("Google", "Strategy Analyst");
console.log(Jobs.jobApplication(["John Doe-Bachelor-10", "Peter Parker-Master-5","Jordan Cole-High School-5", "Daniel Jones- Bachelor-18"]));
console.log(Jobs.jobOffer("John Doe-8"));
console.log(Jobs.jobOffer("Peter Parker-4"));
console.log(Jobs.jobOffer("Jordan Cole-4"));
console.log(Jobs.salaryBonus("Jordan Cole"));
console.log(Jobs.salaryBonus("John Doe"));
console.log(Jobs.candidatesDatabase());
```

Output 4

```
You successfully added candidates: John Doe, Peter Parker, Jordan Cole, Daniel Jones.
Welcome aboard, our newest employee is John Doe.
Welcome aboard, our newest employee is Peter Parker.
Welcome aboard, our newest employee is Jordan Cole.
Jordan Cole will sign a contract for Google, as Strategy Analyst with a salary of $40,000 per year!
John Doe will sign a contract for Google, as Strategy Analyst with a salary of $50,000 per year!
Candidates list:
Daniel Jones-18
```

```
John Doe-hired
Jordan Cole-hired
Peter Parker-hired
```

Problem 3. Unit Testing

Your Task

Using **Mocha** and **Chai** write **JS Unit Tests** to test a variable named **Lottery**, which represents an object. You may use the following code as a template:

```
describe("Tests …", function() {
    describe("TODO …", function() {
        it("TODO …", function() {
            // TODO: …
        });
     });
     // TODO: …
});
```

The object that should have the following functionality:

- **`buyLotteryTicket (ticketPrice,ticketCount,buy)`** - A function that accepts **three** parameters: **number**, **number**, and **boolean**.
    - There is a **need for validation** for the input, in case of submitted **invalid** parameters, **throw** an error **`"Invalid input!"`**
    - If the value of the boolean **`buy`** is **false**, **throw** an error:

      `"Unable to buy lottery ticket!"`
    - To be valid, the **ticket purchase** must meet the **following requirement**:
        - If the **`ticketPrice`** is **greater** than **0**, and **`ticketCount`** is **greater** or **equal** to **1, and** the type of **`ticketPrice`** and **`ticketCount`** **is number, return** the string:

          `"You bought ${ticketCount} tickets for ${totalPrice}$."`,

          where **`totalPrice`** is **`ticketPrice`** multiplied by **`ticketCount.`**

- **`checkTicket (ticketNumbers,luckyNumbers)`** - A function that accepts two parameters: **array** and **array.**
    - There is a **need for validation** for the input, in case of submitted **invalid** parameters, **throw** an error **`"Invalid input!"`**
    - To be valid, the **ticket** must meet the **following requirement**:

        - Both **ticketNumbers** and **luckyNumbers** must be **arrays** with exact length of **6** numbers inside.
    - After validation the function compares the numbers from the ticket with the winning numbers.
        - If there is **from 3 to 5** winning numbers in the ticket, **return** the following message:
          "Congratulations you win, check your reward!"
        - If **all 6** are winning numbers, **return** the following message:
          "You win the JACKPOT!!!"

- **secondChance (ticketID, secondChanceWinningIDs)** - A function that accepts two parameters: **number** and **array**.
    - There is a **need for validation** for the input, in case of submitted **invalid** parameters, **throw** an error **"Invalid input!"**
    - To be valid, the **ticket** must meet the **following requirement**:
        - **ticketID** must be from type **number.**
        - **secondChanceWinningIDs** must be **array.**
    - After validation the function checks whether the **ticketID** is included in the **secondChanceWinningIDs** array.
        - If there is a match, **return** the following message:
          "You win our second chance prize!"
        - Else, **return** the following message:
          "Sorry, your ticket didn't win!"

JS Code

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

Lottery.js

```
const lottery = {
  buyLotteryTicket(ticketPrice, ticketCount, buy) {
    if (buy === false) {
      throw new Error("Unable to buy lottery ticket!");
    } else {
      if (
        ticketPrice <= 0 ||
        ticketCount < 1 ||
        typeof ticketPrice !== "number" ||
        typeof ticketCount !== "number"||
        typeof buy !== "boolean"
      ) {
        throw new Error("Invalid input!");
      } else {
        let totalPrice = ticketPrice * ticketCount;
        return `You bought ${ticketCount} tickets for ${totalPrice}$.`;
      }
    }
  },
 checkTicket(ticketNumbers, luckyNumbers) {
    if (
      !Array.isArray(ticketNumbers) ||
      !Array.isArray(luckyNumbers) ||
      ticketNumbers.length !== 6 ||
      luckyNumbers.length !== 6
    ) {
      throw new Error("Invalid input!");
    }

    const uniqueTicketNumbers = ticketNumbers.filter(
      (number, index, array) => array.indexOf(number) === index
    );
    let winningNumbers = 0;

    for (const number of uniqueTicketNumbers) {
      if (luckyNumbers.includes(number)) {
```

```
        winningNumbers++;
      }
    }

    if (winningNumbers >= 3 && winningNumbers < 6) {
      return "Congratulations you win, check your reward!";
    } else if (winningNumbers === 6) {
      return "You win the JACKPOT!!!";
    }
  }
  ,
  secondChance(ticketID, secondChanceWinningIDs) {
    if (typeof ticketID !== "number" || !Array.isArray(secondChanceWinningIDs)) {
      throw new Error("Invalid input!");
    }
    if (secondChanceWinningIDs.includes(ticketID)) {
      return "You win our second chance prize!";
    } else {
      return "Sorry, your ticket didn't win!";
    }
  },
};


module.exports = lottery;
```

Submission

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