Web Programming

Tutorial 4

To begin this tutorial, please create an empty folder to put your HTML and JS files. When you finish, zip all your deliveries to submit to this tutorial's submission box. The zip file's name should follow this format: `tclass_sid.zip` where `tclass` is your tutorial class name (e.g. `tut01`, `tut02`, `tut03`, etc.) and `sid` is your student's ID (e.g. `2101040015`).

Activity 1 – A simple Promise

In a page named `simple_promise.html`, construct a promise that resolves after 5 seconds. The promise value should be `'promise one is done!'`.

Upon fulfillment, the message should be logged to the console.

Requirements:

- Write the solution using `Promise` chaining.
    1. Create a `Promise` that resolves after 5 seconds.
    2. When the `Promise` resolves, log `'promise one is done!'` to the console using `.then()`
- Write the solution using `async` and `await`
    1. Write an `async` function that calls the same `Promise`
    2. Use the `await` keyword to wait for the `Promise` to resolve, and then log `'promise one is done!'` to the console.

Activity 2 – Review Questions

Answer the following questions in a file called `review_question_ans.txt`. Briefly explain the theoretical foundation of your answer.

Question 1: What is the order of statements logged to the console? Try answering this without executing the code.

```
function orderExecutor(resolve, reject) {
  console.log('1');
  setTimeout(function () {
    resolve('2');
  }, 1000);
}
let p1 = new Promise(orderExecutor);
p1.then(console.log);
console.log('3');
```

Question 2: What happens if the button is not clicked within 5 seconds? Try answering this without executing the code.

```
function buttonExecutor(resolve, reject) {
  let myBtn = document.querySelector('button');
  myBtn.addEventListener('click', resolve);
  setTimeout(reject, 5000);
}
let betterClick = new Promise(buttonExecutor);
betterClick
  .then(function () { console.log('Option A'); })
  .catch(function () { console.log('Option B'); });
```

Question 3: What happens if this button is clicked after 5 seconds? Note the change to the event listener. Try answering this without executing the code.

```
function buttonExecutor(resolve, reject) {
  let myBtn = document.querySelector('button');
  myBtn.addEventListener('click', function () {
    resolve();
    console.log('clicked!');
  });
  setTimeout(reject, 5000);
}
let betterClick = new Promise(buttonExecutor);
betterClick
```

```
  .then(function () { console.log('Option A'); })
  .catch(function () { console.log('Option B'); });
```

Question 4: What value is logged to the console? Try answering this without executing the code.

```
function executor(resolve, reject) {
  resolve(1);
}
function add(value) {
  return value + 5;
}
function multiply(value) {
  return value * 6;
}
let myPromise = new Promise(executor);
myPromise
  .then(add)
  .then(multiply)
  .then(console.log);
```

Question 5: What value is logged to the console? Try answering this without executing the code.

```
function executor(resolve, reject) {
  resolve(1);
}

function add(value) {
  return new Promise(function (resolve, reject) {
    resolve(value + 5);
  });
}

let myPromise = new Promise(executor);
```

```
myPromise
  .then(add)
  .then(console.log);
```

Question 6: What is the order of statements logged to the console? Try answering this without executing the code.

```
function orderExecutor(resolve, reject) {
  console.log('1');
  resolve('2');
}
let p2 = new Promise(orderExecutor);
p2.then(console.log);
console.log('3');
```

Activity 3 – Rejecting a Promise with button click

In a page named `reject_promise_click.html`, there should be a `button` and a `div` for displaying text. When the page is opened, the div should display *"Promise resolved in x seconds"* where *x* will count from 5 to 1, with a delay of 1 second between numbers, before displaying the text *"Promise has resolved"*. If the user clicks on the `button` before the countdown finishes, the countdown should stop and the text *"Promise has failed to resolve"* should be displayed instead.

Activity 4 – JSON Handling

To begin this activity, please download `tut04-starter.zip` from the course website.

In the `tut04-starter` folder, these starter files (HTML, CSS, and part of JS file) provide the basic structure for the JSON handling. You need to implement the JavaScript logic to load JSON data and displays it after a 3-second countdown. This exercise will help you practice working with JSON data, handling asynchronous operations, and implementing a countdown timer in JavaScript.

You have to do following tasks:

1. Initialize Event Listener

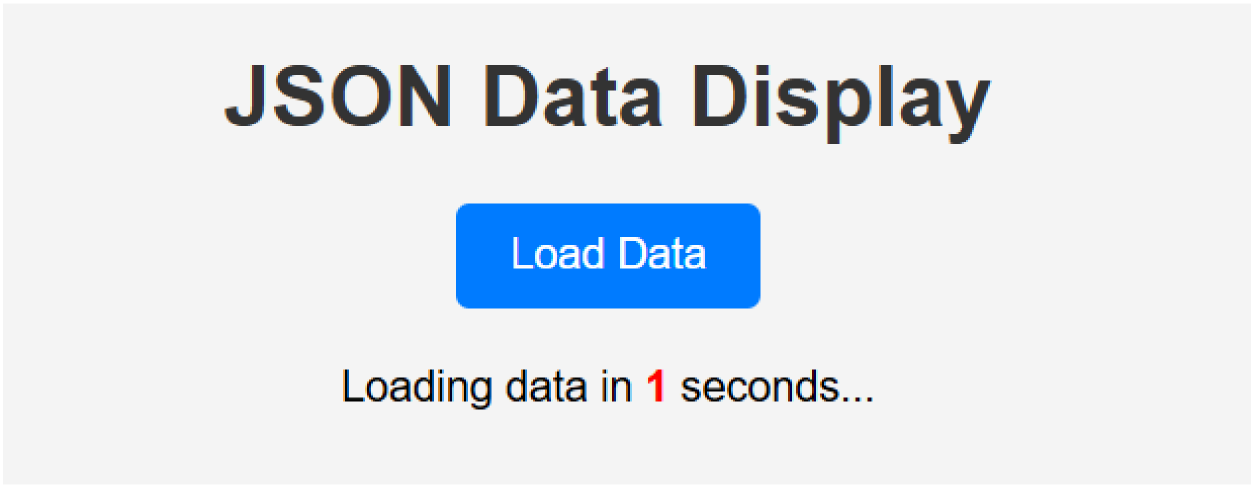
- Add an event listener to the button (`loadDataButton`) to call the `loadData` function when clicked.

2. Simulate Loading JSON Data

- Define a JSON object directly in the JavaScript file.
- Simulate a delay of 3 seconds before displaying the JSON data.

3. Countdown Timer

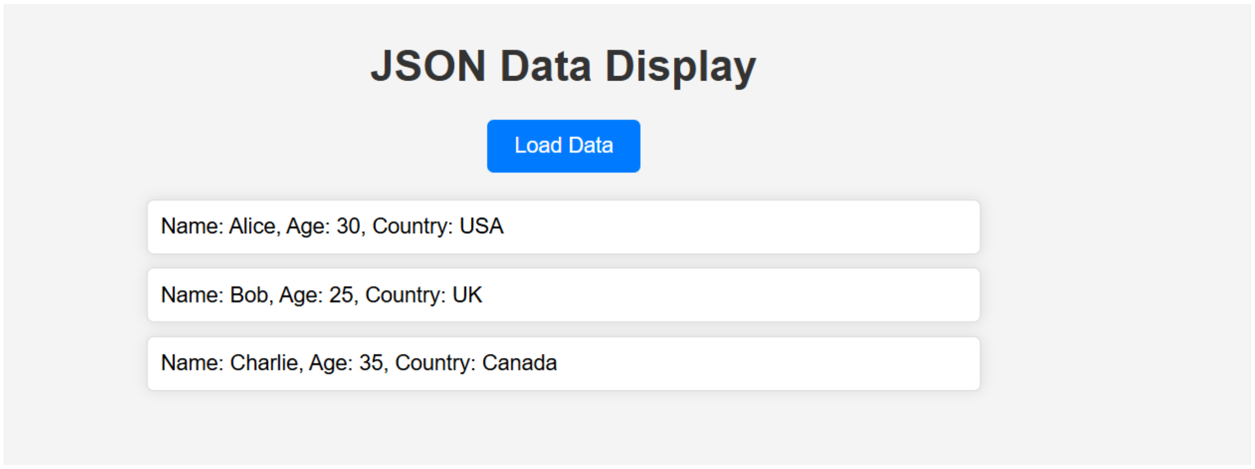
- Before displaying the data, show a countdown timer of 3 seconds.
- Display a message like "`Loading data in X seconds...`" where X is the current countdown value.
- After 3 seconds, display the simulated JSON data.


JSON Data Display

Loading data in 1 seconds...

4. Display Data

- Clear any previous content in the `dataContainer` element.
- Iterate over the `items` array in the JSON object.
- For each item, create a new div element displaying the item's `name`, `age`, and `country`.
- Add a class `data-item` to each div for styling purposes.



```
Name: Alice, Age: 30, Country: USA
Name: Bob, Age: 25, Country: UK
Name: Charlie, Age: 35, Country: Canada
```

→ **Delivery**: modified `script.js`, and after completing it, rename the folder `tut04-starter` to `json_handling` and move it into the folder you created for submission.