Exercise: JavaScript Async Functions II

1. Simulating Network Request with Fetch

Using the **`fetch`** API, write a function **`fetchData()`** which fetches data from https://swapi.dev/api/people/1 and logs the JSON response.

Requirements

- Use the fetch API to get data.
- Parse the response as JSON.
- Log the JSON response to the console.

Hints

- Define an **`async`** function.
- Inside the **`async`** function, use **`fetch`** to get data from the specified URL.
- Use the **`.json()`** method to parse the response.

2. Handling Fetch Errors

Using the **`fetch`** API, write a function **`fetchDataWithErrorHandling()`** which fetches data from https://swapi.dev/api/people/1 and handles potential errors using **`try/catch`**.

Requirements

- Use the fetch API to get data.
- Handle errors using try/catch.
- Log the JSON response or any errors.

3. Parallel Fetch Requests

Using the **`fetch`** API and **`Promise.all`**, write a function **`fetchParallel()`** which makes two parallel **`fetch`** requests to https://swapi.dev/api/people/1 and https://swapi.dev/api/people/2 and logs both results.

Requirements

- Make two parallel fetch requests.
- Use **`Promise.all`** to handle the responses.

4. Sequential Fetch Requests

Using the **`fetch`** API and **`async/await`**, write a function **`fetchSequential()`** which makes two sequential **`fetch`** requests to https://swapi.dev/api/people/1 and https://swapi.dev/api/people/2 and logs both results.

Requirements

- Make two sequential fetch requests.
- Log each result after it is received.

5. Multiple Promises

Using **`Promise.allSettled`**, write a **`multiplePromises()`** function which creates three promises where one resolves after **1** second, one resolves after **2** seconds, and one rejects after **3** seconds. Log the status and value or reason for each promise when all are settled.

Requirements

- Create three promises with specified delays.
- Use **`Promise.allSettled`** to handle all promises.
- Log the status and value or reason for each promise.

6. Retrying a Failed Promise

Using **`async`**/**`await`**, write a function **`startRetry()`** which creates a function that retries a promise up to 3 times if it fails. If the promise eventually resolves, log the result. If it fails after all retries, log the error.

Requirements

- Create a function that retries a promise up to 3 times.
- Log the result if the promise resolves.
- Log the error if the promise fails after all retries.

7. Throttling Promises

Using **`async`**/**`await`**, write a function **`throttlePromises()`** which creates a function that throttles promises so that only two promises are executed in parallel at any time. Ensure that once a promise is resolved, the next one starts.

Requirements

- Create a function that throttles promises with a specified concurrency limit.
- Ensure that only two promises are executed in parallel at any time.
- Log the results after all promises are resolved.

8. Timeout for Fetch Requests

Using **`async`**/**`await`**, write a function **`fetchWithTimeout()`** that fetches data from a URL with a timeout. If the fetch takes longer than the timeout, it should reject.

Requirements

- Create a function that fetches data with a specified timeout.
- Reject the promise if the fetch takes longer than the timeout.
- Log the result or error.

9. Async Function with Error Handling

Using **`async`**/**`await`**, write a class **`AsyncQueue()`** which creates a queue that processes asynchronous tasks one by one in sequence.

Requirements

- Create a queue that processes asynchronous tasks in sequence.
- Ensure the tasks are processed one by one.
- Log the completion of each task.

10. Combining Async/Await with Generators

Using **`async`**/**`await`** and generators, write a function **`startAsyncGenerator()`** that combines **`async`**/**`await`** with generators to handle a sequence of asynchronous tasks.

Requirements

- Create a function that combines async/await with generators.
- Ensure the function can handle a sequence of asynchronous tasks.
- Log the results of the tasks.