Write a function sumAsync that takes two numbers as arguments and uses a callback to return their sum after a delay of lsecond.

Solution:

```
function sumAsync(a, b, callback) {
  setTimeout(() => {
    const result = a + b;
    callback( result);
  }, 1000);
}

sumAsync(3, 7, (result) => {
  console.log(result); // Output: 10
});
```

Create a function getData that returns a Promise. The Promise should resolve after 2 seconds with a message "Data fetched successfully."

Solution:

JavaScript .....

```
function getData() {
  return new Promise((resolve) => {
    setTimeout(() => {
      resolve("Data fetched successfully.");
    }, 2000);
  });
}

getData().then((result) => {
  console.log(result); // Output: Data fetched successfully.
});
```

Write an asynchronous function fetchData that uses the Fetch API to retrieve data from a given URL and returns the parsed JSON response.

API to be used
https:L/isonplaceholdertvpicodecom/todos/j

```
async function fetchData(url) {
  canst response = await fetch(url);
  canst data = await response.json();
  return data;


fetchData("https://jsonplaceholder.typicode.com/todos/1").then((data) => {
  console.log(data);
// Output: { userid: 1, id: 1, title: 'delectus aut autem', completed: false
});
```

Write an asynchronous function fetchData that uses the Fetch API to retrieve data from a given URL (https:/A son laceholder. icode.com todos) and returns the parsed JSON response.

Solution:

```
async function fetchData(url) {

  const response = await fetch(url);
  const data = await response.json();
  return data;
}

fetchData("https://jsonplaceholder.typicode.com/todos/1").then((data) => {
  console.log(data);
});
// Output: { userid: 1, id: 1, title: 'delectus aut autem', completed: false }
```

Implement a function multiplyWithCallback that takes an array of numbers and a callback function. The function should multiply each element of the array by 2 and pass the result to the callback.

Solution

```
JavaScript
function multiplyWithCallback(numbers, callback) {
        multipliedArray = numbers.map((num) => num * 2);
  callback(multipliedArray);
}

canst inputArray = [1, 2, 3, 4];
multiplyWithCallback(inputArray, (result) => {
  console.log(result);  // Output: [2, 4, 6, 8]
})
```

Create a function fetchUserDataAndPosts that takes a user ID and fetches the user details and their posts using separate API calls. Use promise chaining to ensure the posts are retrieved only after the user details are fetched. Return an object with user details and posts.

Solution:

```
function fetchUserDataAndPosts(userid) {
  return fetch("https://jsonplaceholder.typicode.com/users/${userid}")
    .then((response) => response.json())
    .then((userData) => {
      return fetch(
        "https://jsonplaceholder.typicode.com/posts?userid=${userid}"

        .then((response) => response.json())
        .then((posts) => ({ user: userData, posts }));
    });
}
```

```
fetchUserDataAndPosts (1 ).then((result) => {
  console.log(result); // Output: {user: {...}, posts: [...] }
});
```

Write a function fetchMultipleData that takes an array of URLs and uses Promise.all() to fetch data from all the URLs concurrently. Return an array of responses.

API to be used
Change todo id for each API call
https://jsonplaceholder. icode.com todos

Solution:

```
JavaScript
function fetchMultipleData (urls) {
  const promises = urls.map( (url) =>
    fetch(url) .then( (response) => response.json())
  );

  return  Promise.all (promises);
}

canst urlsToFetch = [
  "https://jsonplaceholder. typicode.com/todos/1 ",
  "https://jsonplaceholder. typicode.com/todos/2" ,
  "https://jsonplaceholder. typicode.com/todos/3" ,
];

fetchMultipleData (urlsToFetch) .then ((responses) => {
  console.log (responses); JI Array of responses from each URL
});
```

Create a function racePromises that takes an array of promises and returns the result of the first promise that resolves or rejects. Use Promise.race() to implement this.

Solution:

```
function racePromises (promises) {
  return  Promise.race (promises);
}
```

```
canst promisel = new Promise((resolve) =>
  setTimeout( () => resolve ("Promise 1 resolved"), 1000)
);
canst promise2 = new Promise((_, reject) =>
  setTimeout( () => reject ("Promise 2 rejected"), 500)
);

racePromises([promisel, promise2])
  .then( (result) => {
    console.log( result);
  })
  .catch( (error) => {
    console.error (error);
  });
```