Mastering URI Handling in JavaScript: EncodeURI() and DecodeURI()

encodeURI() 3
Usage: 4
Example: 4
decodeURI() 4
Usage: 4
Example: 5
Common Use Cases: 5
Important Note: 5
Summary: 6
10 coding exercises **6**
Exercise 1: Encode a Simple URI 6
Exercise 2: Decode a URI 7
Exercise 3: Encode User-Generated Input 8
Exercise 4: Decode User-Generated Input 9
Exercise 5: Encode Query Parameters 10
Exercise 6: Decode Query Parameters 11
Exercise 7: URL Builder 13
Exercise 8: URL Parser 14
Exercise 9: Safely Decode User Input 16
Exercise 10: Encode and Decode a Full URL 17
Quiz Questions: **18**

Let's explore two essential functions in JavaScript - encodeURI() and decodeURI(). Understanding these functions is crucial when dealing with Uniform Resource Identifiers (URIs) such as URLs.

1. encodeURI() - Safeguarding Your URIs:

encodeURI() is a handy function that ensures proper formatting and safety when dealing with entire URIs. Whether you're constructing URLs dynamically or handling user input, this function encodes special characters, making your URIs compliant and secure.

2. decodeURI() - Unraveling the Encoded:

On the flip side, decodeURI() comes to the rescue when you need to decode an already encoded URI. It helps transform encoded strings back into readable and usable forms, a valuable tool for extracting information from URIs or handling user-generated content.

Applications in Real-World Scenarios:

User Input Handling: Safely encode and decode user input to prevent security vulnerabilities and enhance user experience.

Building Dynamic URLs: Construct URLs with dynamic content while ensuring proper encoding of special characters.

API Interactions: Encode and decode URLs when interacting with APIs, ensuring seamless communication.

Best Practices:

Query Parameters: When dealing with query parameters, consider using encodeURI() for entire URIs and encodeURIComponent() for individual components.

Error Handling: Safely decode user input by employing try-catch blocks to handle potential decoding errors.

Conclusion:

Mastering encodeURI() and decodeURI() is fundamental for robust URI handling in JavaScript. Whether you're a seasoned developer or just starting, incorporating these functions into your toolkit will elevate your web development skills.

In JavaScript, the decodeURI() and encodeURI() functions are used for working with Uniform Resource Identifiers (URIs), which include URLs. These functions help handle special characters within URIs to ensure proper encoding and decoding. Let's explore each function in detail, along with coding examples.

encodeURI()

The encodeURI() function is used to encode a complete URI by replacing certain characters with their respective percent-encoded values. This

function is typically used to ensure that a URI is properly formatted and safe for use in a web context.

Usage:

```
encodedURI = encodeURI(uri);
```

Example:

```
const originalURI = 'https://www.example.com/path with spaces/page?query=hello world#fragment';

const encodedURI = encodeURI(originalURI);

console.log(encodedURI);

// Output:
https://www.example.com/path%20with%20spaces/page?query=hello%20world#fragment
```

In this example, spaces are replaced with `%20`, ensuring that the URI is valid and can be used in a browser.

decodeURI()

The `decodeURI()` function is used to decode a URI that has been previously encoded using `encodeURI()`. This function is useful when you need to extract information from a URI or when dealing with user input that might contain encoded URIs.

Usage:

```
decodedURI = decodeURI(encodedURI);
```

Example:

```
const encodedURI =
'https://www.example.com/path%20with%20spaces/page?query=hello%20world#fragment';

const decodedURI = decodeURI(encodedURI);

console.log(decodedURI);

// Output: https://www.example.com/path with spaces/page?query=hello world#fragment
```

In this example, the previously encoded URI is decoded back to its original form, making it readable and usable in your JavaScript code.

Common Use Cases:

Handling User Input: When dealing with user-generated content that might include URLs, it's essential to encode and decode the URIs to prevent security issues and ensure proper functionality.
Building Query Parameters: When constructing URLs with dynamic data, you should use `encodeURI()` to ensure that special characters in the data don't interfere with the structure of the URL.
Working with APIs: When sending or receiving data from APIs, encoding and decoding URLs can be crucial to avoid issues related to special characters in URIs.

Important Note:

While `encodeURI()` is suitable for encoding entire URIs, if you need to encode individual components (such as query parameters), you should use `encodeURIComponent()`.

Summary:

- encodeURI() is used to encode entire URIs, ensuring proper formatting and safety.
- decodeURI() is used to decode previously encoded URIs, making them readable and usable in JavaScript.

These functions play a crucial role in web development, particularly when working with URLs and handling user input that involves URI components. Understanding when and how to use encodeURI() and decodeURI() is essential for writing robust and secure JavaScript code.

10 coding exercises

10 coding exercises to help you practice using encodeURI() and decodeURI() in JavaScript. Each exercise comes with step-by-step descriptions, code examples, and solutions.

Exercise 1: Encode a Simple URI

Task: Encode the given URI.

Steps:

1. Declare a variable with a simple URI.
2. Use encodeURI() to encode the URI.
3. Log the encoded URI.

Code:

```
const originalURI = 'https://www.example.com/path with spaces/page?query=hello world#fragment';

// Encoding the URI

const encodedURI = encodeURI(originalURI);

// Logging the result

console.log(encodedURI);
```

Solution:

```
// Output:
https://www.example.com/path%20with%20spaces/page?query=hello%20world#fragment
```

Exercise 2: Decode a URI

Task: Decode the given encoded URI.

Steps:

1. Declare a variable with an encoded URI.
2. Use `decodeURI()` to decode the URI.
3. Log the decoded URI.

Code:

```
const encodedURI =
'https://www.example.com/path%20with%20spaces/page?query=hello%20world#fragment';

// Decoding the URI

const decodedURI = decodeURI(encodedURI);

// Logging the result

console.log(decodedURI);
```

Solution:

```
// Output: https://www.example.com/path with spaces/page?query=hello world#fragment
```

Exercise 3: Encode User-Generated Input

Task: Create a function that takes a user's input and encodes it for a URL.

Steps:

1. Create a function `encodeUserInput` that takes user input as a parameter.
2. Use `encodeURI()` to encode the user input.
3. Return the encoded string.

Code:

```
function encodeUserInput(input) {

 return encodeURI(input);
```

```
}

const userInput = 'This is user input with spaces!';

// Encoding user input

const encodedUserInput = encodeUserInput(userInput);

// Logging the result

console.log(encodedUserInput);
```

Solution:

```
// Output: This%20is%20user%20input%20with%20spaces!
```

Exercise 4: Decode User-Generated Input

Task: Create a function that takes an encoded URI component and decodes it.

Steps:

1. Create a function `decodeUserInput` that takes an encoded string as a parameter.
2. Use `decodeURI()` to decode the input.
3. Return the decoded string.

Code:

```
function decodeUserInput(encodedInput) {

 return decodeURI(encodedInput);

}

const encodedUserInput = 'This%20is%20user%20input%20with%20spaces!';

// Decoding user input

const decodedUserInput = decodeUserInput(encodedUserInput);

// Logging the result

console.log(decodedUserInput);
```

Solution:

```
// Output: This is user input with spaces!
```

Exercise 5: Encode Query Parameters

Task: Create a function that takes an object with query parameters and encodes them for a URL.

Steps:

1. Create a function `encodeQueryParams` that takes an object as a parameter.
2. Use `encodeURI()` to encode each key-value pair in the object.
3. Return the encoded query string.

Code:

```
function encodeQueryParams(queryObject) {
 const encodedPairs = Object.entries(queryObject)
 .map(([key, value]) => `${encodeURIComponent(key)}=${encodeURIComponent(value)}`);
 return encodedPairs.join('&');
}
const queryParams = {
 name: 'John Doe',
 age: 30,
 city: 'New York',
};
// Encoding query parameters
const encodedQueryParams = encodeQueryParams(queryParams);
// Logging the result
console.log(encodedQueryParams);
```

Solution:

```
// Output: name=John%20Doe&age=30&city=New%20York
```

Exercise 6: Decode Query Parameters

Task: Create a function that takes an encoded query string and decodes it into an object.

Steps:

1. Create a function `decodeQueryParams` that takes an encoded query string as a parameter.
2. Split the query string into key-value pairs.
3. Use `decodeURI()` to decode each key and value.
4. Return an object with the decoded key-value pairs.

Code:

```
function decodeQueryParams(encodedQueryString) {

 const decodedPairs = encodedQueryString.split('&')

 .map(pair => {

 const [key, value] = pair.split('=');

 return [decodeURIComponent(key), decodeURIComponent(value)];

 });

 return Object.fromEntries(decodedPairs);

}

const encodedQueryString = 'name=John%20Doe&age=30&city=New%20York';

// Decoding query parameters
```

```
const decodedQueryParams = decodeQueryParams(encodedQueryString);

// Logging the result

console.log(decodedQueryParams);
```

Solution:

```
// Output: { name: 'John Doe', age: '30', city: 'New York' }
```

Exercise 7: URL Builder

Task: Create a function that constructs a URL with encoded query parameters.

Steps:

1. Create a function buildURL that takes a base URL and an object of query parameters.
2. Use encodeURI() and encodeQueryParams() to build the complete URL.
3. Return the final URL.

Code:

```
function buildURL(baseURL, queryParams) {

 const encodedParams = encodeQueryParams(queryParams);

 return `${encodeURI(baseURL)}?${encodedParams}`;

}

const baseURL = 'https://api.example.com/data';
```

```
const queryParams = {
 category: 'technology',
 page: 1,
 filter: 'new',
};
// Building the URL
const finalURL = buildURL(baseURL, queryParams);
// Logging the result
console.log(finalURL);
```

Solution:

```
// Output: https://api.example.com/data?category=technology&page=1&filter=new
```

Exercise 8: URL Parser

Task: Create a function that parses a URL and returns an object with decoded components.

Steps:

1. Create a function `parseURL` that takes a URL as a parameter.
2. Use `decodeURI()` to decode the entire URL.

3. Extract and decode individual components like protocol, host, path, query, and fragment.
4. Return an object with the decoded components.

Code:

```
function parseURL(url) {

 const decodedURL = decodeURI(url);

 const urlObject = new URL(decodedURL);

 return {

 protocol: urlObject.protocol,

 host: urlObject.host,

 path: urlObject.pathname,

 query: decodeQueryParams(urlObject.search.slice(1)),

 fragment: decodeURIComponent(urlObject.hash.slice(1)),

 };

}

const sampleURL =
'https://www.example.com/path%20with%20spaces/page?query=hello%20world#fragment';

// Parsing the URL

const parsedURL = parseURL(sampleURL);
```

```
// Logging the result

console.log(parsedURL);
```

Solution:

```
// Output: { protocol: 'https:', host: 'www.example.com', path: '/path with spaces/page', query: { query: 'hello world' }, fragment: 'fragment' }
```

Exercise 9: Safely Decode User Input

Task: Create a function that safely decodes user input, handling potential decoding errors.

Steps:

1. Create a function safelyDecode that takes an encoded string as a parameter.
2. Use a try-catch block to handle potential decoding errors.
3. Return the decoded string or an error message.

Code:

```
function safelyDecode(encodedInput) {

 try {

 return decodeURI(encodedInput);

 } catch (error) {

 return `Error decoding: ${error.message}`;
```

```
 }

}

const possiblyEncodedInput = 'This%20is%20a%20malformed%20string%';

// Safely decoding user input

const safelyDecodedInput = safelyDecode(possiblyEncodedInput);

// Logging the result

console.log(safelyDecodedInput);
```

Solution:

```
// Output: Error decoding: URI malformed
```

Exercise 10: Encode and Decode a Full URL

Task: Create a function that takes a full URL, encodes it, and then decodes it back.

Steps:

1. Create a function `encodeAndDecodeURL` that takes a full URL as a parameter.
2. Use `encodeURI()` to encode the URL.
3. Use `decodeURI()` to decode the encoded URL.
4. Return both the encoded and decoded URLs.

Code:

```
function encodeAndDecodeURL(fullURL) {

 const encodedURL = encodeURI(fullURL);

 const decodedURL = decodeURI(encodedURL);

 return { encoded: encodedURL, decoded: decodedURL };

}



const fullURL = 'https://www.example.com/path with spaces/page?query=hello
world#fragment';



// Encoding and decoding the URL

const result = encodeAndDecodeURL(fullURL);

// Logging the result

console.log(result);
```

Solution:

```
// Output: { encoded:
'https://www.example.com/path%20with%20spaces/page?query=hello%20world#fragment',
decoded: 'https://www.example.com/path with spaces/page?query=hello world#fragment' }
```

Quiz Questions:

Question: What is the purpose of the encodeURI() function in JavaScript?

A) To encode individual URI components

B) To encode entire URIs

C) To decode URIs

D) To check if a URI is valid

Answer: B) To encode entire URIs

Question: Which function is used to decode a URI that has been encoded using encodeURI()?

A) decodeURIComponent()

B) decodeURI()

C) decodeURIComponent()

D) decodeURIFull()

Answer: B) decodeURI()

Question: What is the primary use case for the encodeURI() function?

A) Building complex queries

B) Safely decoding user input

C) Encoding query parameters

D) Ensuring proper formatting of entire URIs

Answer: D) Ensuring proper formatting of entire URIs

Question: How can you decode a URI component that might contain special characters?

A) Use `decodeURI()`

B) Use `decodeURIComponent()`

C) Use `encodeURI()`

D) Use `encodeURIcomponent()`

Answer: B) Use `decodeURIComponent()`

Question: What does the `decodeURI()` function return if there is an error in decoding?

A) `null`

B) An empty string

C) The original encoded string

D) An error message

Answer: C) The original encoded string

Question: When should you use encodeURI() instead of encodeURIComponent()?

A) When encoding entire URIs

B) When encoding individual query parameters

C) When dealing with user input

D) Both A and B

Answer: A) When encoding entire URIs

Question: What is the purpose of the encodeQueryParams() function in Exercise 5?

A) To encode individual query parameters

B) To encode an entire URI

C) To encode user input

D) To encode user-generated content

Answer: A) To encode individual query parameters

Question: Why is it important to decode user input safely, as shown in Exercise 9?

A) To prevent security vulnerabilities

B) To ensure a better user experience

C) To improve code readability

D) Both A and B

Answer: D) Both A and B

Question: In Exercise 8, what does the parseURL function return?

A) An array of URL components

B) A string with URL components separated by commas

C) An object with decoded URL components

D) The original URL string

Answer: C) An object with decoded URL components

Question: Which function is suitable for decoding an entire URL, including its components, as shown in Exercise 8?

A) decodeURI()

B) decodeURIComponent()

C) decodeURIFull()

D) parseURL()

Answer: A) decodeURI()