HyperionDev

Full-Stack Web Development

HyperionDev

Functions and DOM Manipulation in JavaScript

Full Stack Web Development Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Full Stack Web Development Session Housekeeping cont.

- For all **non-academic questions**, please submit a query: **www.hyperiondev.com/support**
- Report a **safeguarding** incident: **www.hyperiondev.com/safeguardreporting**
- We would love your **feedback** on lectures: **Feedback on Lectures**

Objectives

- Understand the concept of functions and how to define them in JavaScript.
- Differentiate between parameters and arguments.
- Understand Promises, Async & Await
- Perform basic DOM manipulation using JavaScript.

Introduction to Functions in JavaScript

What is a Function?

- A function is a reusable block of code designed to perform a specific task.
- Functions help to organize and simplify your code by breaking down complex operations into manageable chunks.

Defining a Function

JS script.js > ...

```
1   // Syntax
2
3   function functionName(parameters) {
4       // code to be executed
5   }
6
```

JS script.js > ...

```
1   // Example
2
3   function greet(name) {
4       return "Hello, " + name + "!";
5   }
6
```

- Explanation: In this example, greet is the function name, name is a parameter, and the function returns a greeting message.

Invoking a Function

- Functions are invoked (called) by using their name followed by parentheses.
- **Example**:

```
JS script.js
1   // Example
2
3   console.log(greet("Alice")); // Output: "Hello, Alice!"
4
```

- **Explanation**: Here, "Alice" is the argument passed to the greet function.

Parameters vs. Arguments in JavaScript Functions

- What are Parameters?
  - Parameters are the names listed in the function definition that act as placeholders for the values that will be passed to the function when it is called.
  - They allow functions to be more flexible and reusable by enabling the same function to operate on different data inputs.
  - Think of parameters as variables that are defined within the function and are used to refer to the values that will be provided later.

Defining Parameters

- Parameters are specified within the parentheses of a function definition.
- **Example**:

```
JS script.js > ...
1   // Example
2
3   function multiply(x, y) {
4       return x * y;
5   }
6
```

- Explanation: In this example, x and y are parameters. They are placeholders for the values that will be multiplied when the function is called.

What are Arguments?

- Arguments are the actual values or expressions that are passed to the function when it is called.
- These values replace the placeholders (parameters) defined in the function and are used to perform the functions' operations.
- When passing arguments to a function, they must match the order of the parameters defined in the function.
- Example:

JS script.js > ...

```
1  // Example
2
3  function multiply(x, y) {
4      return x * y;
5  }
6
7  let result = multiply(5, 3); // Output: 15
8
```

Promises

- A Promise is an object that represents the eventual completion (or failure) of an asynchronous operation, and its resulting value. Promises provide a way to register callbacks to be called when the async operation completes or fails.
- Promises have three states:
  - Pending: The initial state, neither fulfilled nor rejected.
  - Fulfilled: This means that the operation was completed successfully.
  - Rejected: This means that the operation failed.
- Here is an example of a basic promise:

```
JS script.js > ...
1  const myPromise = new Promise((resolve, reject) => {
2      setTimeout(() => {
3          resolve("Hello from myPromise!");
4      }, 1000);
5  });
6
7  myPromise.then(console.log);
8
```

Async & Await

- Async & Await are JavaScript keywords that are used to handle promises. The async keyword defines an asynchronous function, which returns a promise. The await keyword is used to wait for a promise to be resolved before moving on to the next line of code.
- Here is an example of an asynchronous function using async & await:

```
JS script.js > ...
1   async function myAsyncFunction() {
2       const myPromise = new Promise((resolve, reject) => {
3           setTimeout(() => {
4               resolve("Hello from myAsyncFunction!");
5           }, 1000);
6       });
7       const result = await myPromise;
8       console.log(result);
9   }
10
11  myAsyncFunction();
```

Introduction to DOM Manipulation

- What is the DOM?
  - DOM stands for Document Object Model, representing the structure of an HTML document as a tree of objects.
  - JavaScript allows you to interact with and manipulate these objects to create dynamic and interactive web pages.
- **Real-World Example**: Imagine a webpage with a form where users can submit their information. The DOM allows you to capture the input data, validate it, and even provide real-time feedback (e.g., highlighting fields with errors) without reloading the page. This interaction is crucial for building modern web applications like forms, interactive dashboards, and dynamic content displays.

Selecting DOM Elements

- To manipulate the DOM, the first step is to select the elements you want to interact with. JavaScript provides several methods for this purpose, with the most commonly used ones being `document.getElementById()` and `document.querySelector()`.

document.getElementById()

- This method is used to select a single element based on its unique id attribute.
- Syntax:

```
JS script.js > ...
1   const element = document.getElementById("myElement");
2
```

- **Explanation**: The getElementById() method searches the DOM for an element with the specified id and returns a reference to it. The id must be unique within the document, making this method a precise way to select a specific element.
- **Example Use Case:** If you have a `<div>` with `id="header"`, you can select it and change its content or style directly:

```
JS script.js > ...
1   const header = document.getElementById("header");
2   header.textContent = "Welcome to My Site";
```

document.querySelector()

- This method is used to select the first element that matches a given CSS selector.
- Syntax:

```
JS script.js > ...
1   const element = document.querySelector(".myClass");
```

- **Explanation**: The querySelector() method is more versatile than getElementById() because it allows you to use any valid CSS selector to find an element. This includes class selectors (.myClass), id selectors (#myId), tag selectors (div), and even more complex selectors like attribute selectors or pseudo-classes.
- Example Use Case:

```
JS script.js > ...
1   const firstContainer = document.querySelector(".container");
2   firstContainer.style.backgroundColor = "lightgray";
```

Modifying DOM Elements

- Once you've selected an element, you can modify its content and style to update the web page dynamically. Here are two common modifications:
- Changing Text Content:
  - You can change the text displayed within an HTML element using the textContent or innerHTML properties.
- Syntax:

```
JS script.js
1   document.getElementById("message").textContent = "New Text";
```

- Example Use Case: Suppose you have a paragraph with the id="message", and you want to update its content based on user interaction:

```
JS script.js > ...
1   const message = document.getElementById("message");
2   message.textContent = "Thank you for submitting your details!";
```

Changing Styles

- You can dynamically change the appearance of an element by modifying its CSS properties via the style object.
- Syntax:

```
JS script.js
1   document.getElementById("message").style.color = "blue";
```

- The style property allows you to access and modify the inline styles of an element. Each CSS property can be changed by assigning a new value to it, such as changing the text color, background color, font size, and more.
- **Example Use Case**: If you want to change the color of a message based on a user's action, you could do the following:

```
JS script.js > ...
1   const message = document.getElementById("message");
2   message.style.color = "red";
3   message.style.fontSize = "20px";
```

Activity

- Write JavaScript code to change the text content of an HTML element and modify its style based on user input.

JS script.js > ...

```
1  document.getElementById("changeText").addEventListener("click", function() {
2      document.getElementById("message").textContent = "Text has been changed!";
3      document.getElementById("message").style.color = "green";
4  });
```

Hyperiondev

Questions and Answers

HyperionDev

Thank You for attending!