JS Frontend Dev Notes

chat.openai.com/chat

write javascript notes with code used in frontend development

Here are some common JavaScript concepts and code snippets used in front-end development:

1. Variables:

javascript

```
var name = "John Doe";
let age = 30;
const country = "USA";
```

2. Data Types:

javascript

```
// Primitive Types:
let name = "John Doe"; // String
let age = 30; // Number
let isMale = true; // Boolean
let address = null; // Null
let sym = Symbol(); // Symbol (ES6)

// Reference Types:
let hobbies = ["reading", "music"]; // Array
let person = {firstName: "John", lastName: "Doe"}; // Object
```

3. Conditional Statements:

```javascript
let age = 30;
if (age < 18) {
  console.log("minor");
} else if (age >= 18 && age < 60) {
  console.log("adult");
} else {
  console.log("senior citizen");
}
```

4. Loops:

```javascript
// For Loop:
for (let i = 0; i < 5; i++) {
  console.log(i);
}

// For-Each Loop:
let hobbies = ["reading", "music"];
hobbies.forEach(function(hobby) {
  console.log(hobby);
});

// While Loop:
let i = 0;
while (i < 5) {
  console.log(i);
  i++;
}
```

5. Functions:

javascript

```javascript
// Function Declaration:
function sayHello() {
  console.log("Hello");
}

// Function Expression:
let sayGoodbye = function() {
  console.log("Goodbye");
};

// Arrow Function (ES6):
let sayHi = () => console.log("Hi");
```

6. Event Handling:

javascript

```javascript
// Add Event Listener:
let btn = document.querySelector("button");
btn.addEventListener("click", function() {
  console.log("Button clicked");
});

// Event Handler Property:
let btn = document.querySelector("button");
btn.onclick = function() {
  console.log("Button clicked");
};
```

These are some of the basics of JavaScript used in front-end development.

More code and comments on DOM manipulations

Sure! Here's some more information on manipulating the Document Object Model (DOM) in JavaScript:

1. Selecting Elements:

```
javascript

// Get Element by ID:
let header = document.getElementById("header");

// Get Elements by Class Name:
let items = document.getElementsByClassName("item");

// Get Elements by Tag Name:
let links = document.getElementsByTagName("a");

// Query Selector:
let header = document.querySelector("#header");
let firstItem = document.querySelector(".item:first-child");
```

2. Modifying Element Content:

```
css

// Get Text Content:
let header = document.getElementById("header");
let text = header.textContent;

// Set Text Content:
header.textContent = "New Header";

// Get HTML Content:
let header = document.getElementById("header");
let html = header.innerHTML;

// Set HTML Content:
header.innerHTML = "<h1>New Header</h1>";
```

3. Modifying Element Styles:

less

```
// Get Style:
let header = document.getElementById("header");
let color = header.style.color;

// Set Style:
header.style.color = "red";
header.style.backgroundColor = "#ccc";
header.style.fontSize = "20px";

// Add/Remove CSS Class:
header.classList.add("new-class");
header.classList.remove("new-class");
header.classList.toggle("new-class");
```

4. Creating and Deleting Elements:

javascript

```
// Create Element:
let newHeader = document.createElement("h1");
newHeader.textContent = "New Header";

// Append Element:
let body = document.querySelector("body");
body.appendChild(newHeader);

// Prepend Element:
let body = document.querySelector("body");
body.insertBefore(newHeader, body.firstChild);

// Remove Element:
let header = document.getElementById("header");
header.remove();
```

These are some of the basic operations you can perform on the DOM using JavaScript. With these, you can create dynamic and interactive websites.

also for Hoisting Scoping this keword closure method Map like reduce , filter etc call back function promises Async Awaiy Js

Sure! Here is an explanation of these additional JavaScript concepts:

1. Hoisting:

Hoisting is a behavior in JavaScript where variables and function declarations are moved to the top of their scope, regardless of where they are defined in the code. This means that you can use a variable or call a function before it is declared in the code. However, only the declaration is hoisted, not the assignment.

javascript

```javascript
// Example of Hoisting:
console.log(name); // undefined
var name = "John Doe";

// Function Declarations are also hoisted:
sayHello(); // "Hello"
function sayHello() {
  console.log("Hello");
}
```

2. Scoping:

Scoping determines the visibility and accessibility of variables and functions in different parts of the code. In JavaScript, there are two types of scopes: global scope and local scope. Variables declared outside of any function have global scope and can be accessed from anywhere in the code. Variables declared inside a function have local scope and are only accessible within the function.

```javascript
// Global Scope:
let name = "John Doe";
function sayHello() {
  console.log(name); // "John Doe"
}

// Local Scope:
function sayHello() {
  let name = "Jane Doe";
  console.log(name); // "Jane Doe"
}
```

3. `this` keyword:

The `this` keyword refers to the current object in the context of the code. Its value can change depending on where it is used in the code and how the function is called. In a global context, `this` refers to the global object (`window` in the browser). In a method of an object, `this` refers to the object itself.

```javascript
// Example of `this` in a global context:
console.log(this); // window

// Example of `this` in a method:
let person = {
  name: "John Doe",
  sayHello: function() {
    console.log("Hello, my name is " + this.name);
  }
};
person.sayHello(); // "Hello, my name is John Doe"
```

4. Array Methods:

JavaScript arrays have several built-in methods that can be used to manipulate and process the data in the arrays. These methods include `map` , `reduce` , `filter` , `sort` , etc.

- `map` : This method creates a new array with the result of calling a provided function on every element in the calling array.

typescript

```
let numbers = [1, 2, 3, 4, 5];
let doubledNumbers = numbers.map(function(number) {
  return number * 2;
});
console.log(doubledNumbers); // [2, 4, 6, 8, 10]
```

`reduce` : This method reduces an array to a single value by iteratively combining elements using a provided function.

javascript

```
let numbers = [1, 2, 3, 4, 5];
let sum = numbers.reduce(function(accumulator, currentValue) {
  return accumulator + currentValue;
});
console.log(sum); // 15
```

`filter` : This method creates a new array with all elements that pass the test implemented by the provided function.

javascript

```
let numbers = [1, 2, 3
```

complete please

typescript

```
, 4, 5];
let evenNumbers = numbers.filter(function(number) {
  return number % 2 === 0;
});
console.log(evenNumbers); // [2, 4]
```

5. Callback Functions:

A callback function is a function that is passed as an argument to another function and is executed after the outer function has completed. Callback functions are commonly used in JavaScript to handle asynchronous events and operations.

scss

```
// Example of a callback function:
function greeting(name, callback) {
  console.log("Hello, " + name);
  callback();
}

function sayGoodbye() {
  console.log("Goodbye");
}

greeting("John Doe", sayGoodbye);
// Output:
// Hello, John Doe
// Goodbye
```

6. Promises:

A Promise in JavaScript is an object representing the eventual completion or failure of an asynchronous operation. Promises provide a cleaner and more elegant way to handle asynchronous operations compared to callbacks.

javascript

```
// Example of a Promise:
let promise = new Promise(function(resolve, reject) {
  let result = true;
  if (result) {
    resolve("Success");
  } else {
    reject("Failure");
  }
});

promise
  .then(function(value) {
    console.log(value); // "Success"
  })
  .catch(function(error) {
    console.log(error);
  });
```

7. Async/Await:

Async/Await is a modern way to handle asynchronous operations in JavaScript. It provides a cleaner and more readable syntax compared to Promises and callbacks. The `async` keyword is used to declare an asynchronous function, and the `await` keyword is used to wait for a Promise to be resolved before moving on to the next line of code.

javascript

```
// Example of Async/Await:
async function getData() {
  let response = await fetch("https://api.example.com/data");
  let data = await response.json();
  console.log(data);
}

getData();
```

These are the essential concepts and code examples for front-end development in JavaScript.