CLIENT SIDE SCRIPTING	SERVER SIDE SCRIPTING
HTML, CSS, and javascript are used.	**PHP, Python, Java, Ruby are used.**
The source code is visible to the user.	**The source code is not visible to the user because its output of server-sideside is an HTML page.**
It does not provide security for data.	**It provides more security for data.**
Its main function is to provide the requested output to the end user.	**Its primary function is to manipulate and provide access to the respective database as per the request.**
It usually depends on the browser and its version.	**In this any server-side technology can be used and it does not depend on the client.**
It runs on the user's computer.	**It runs on the webserver.**

```
<script>
const person = {fname:"John", lname:"Doe", age:25};

let txt = "";
for (let x in person) {
  txt += person[x] + " ";
}

document.getElementById("demo").innerHTML = txt;
</script>
```

For in

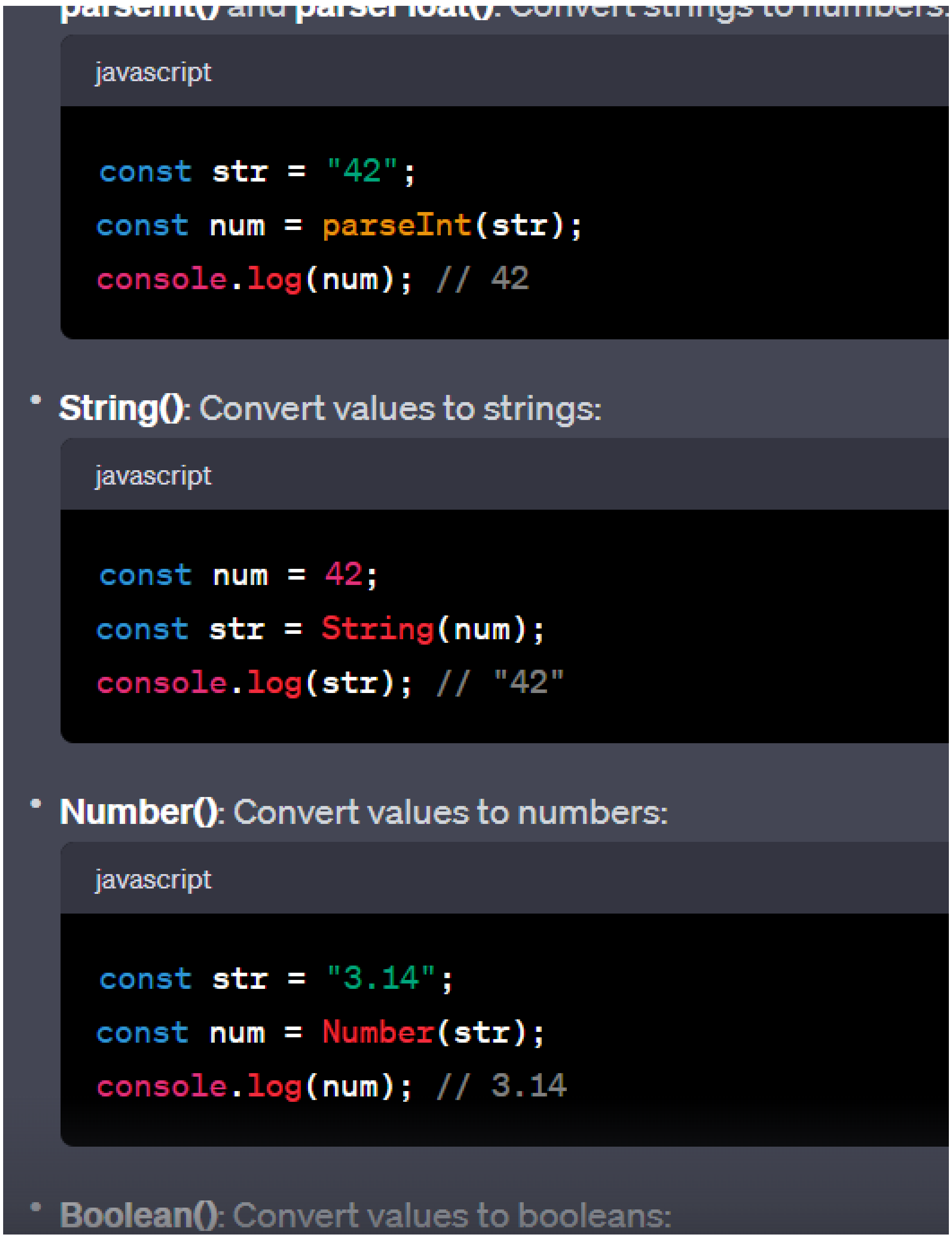
```
const cars = ["BMW", "Volvo", "Mini"];

let text = "";
for (let x of cars) {
  text += x;
}
```

```
const fruits = new Map([
  ["apples", 500],
  ["bananas", 300],
  ["oranges", 200]
]);
```

```
// Create a Map
const fruits = new Map();

// Set Map Values
fruits.set("apples", 500);
fruits.set("bananas", 300);
fruits.set("oranges", 200);
```

```
fruits.get("apples");
```

```
typeof "John"                 // Returns "string"
typeof 3.14                   // Returns "number"
typeof NaN                    // Returns "number"
typeof false                  // Returns "boolean"
typeof [1,2,3,4]              // Returns "object"
typeof {name:'John', age:34}  // Returns "object"
typeof new Date()             // Returns "object"
typeof function () {}         // Returns "function"
typeof myCar                  // Returns "undefined" *
typeof null                   // Returns "object"
```



parseInt() and **parseFloat()**: Convert strings to numbers:

javascript

```javascript
const str = "42";
const num = parseInt(str);
console.log(num); // 42
```

- **String()**: Convert values to strings:

javascript

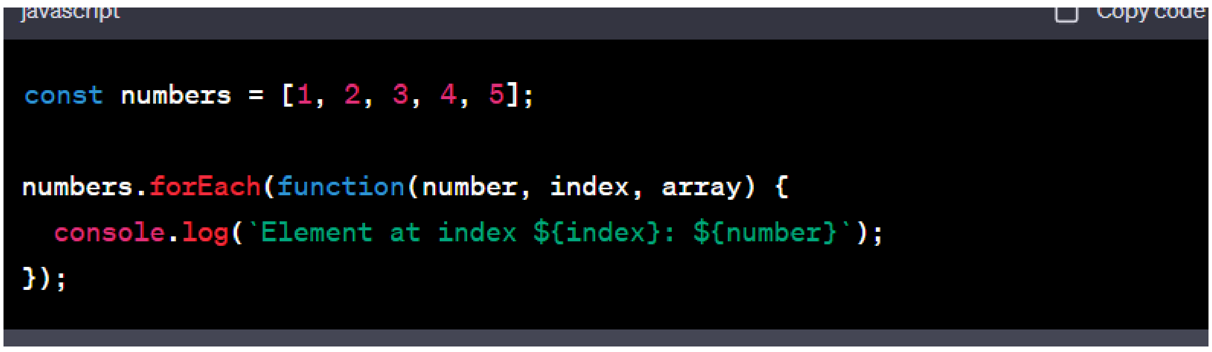
```javascript
const num = 42;
const str = String(num);
console.log(str); // "42"
```

- **Number()**: Convert values to numbers:

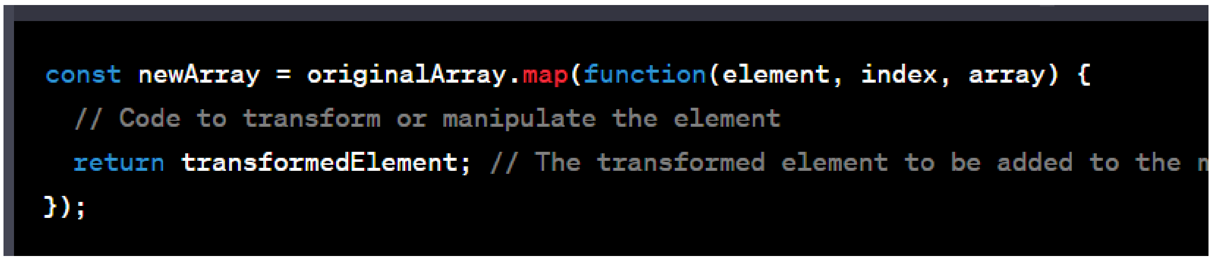
javascript

```javascript
const str = "3.14";
const num = Number(str);
console.log(num); // 3.14
```

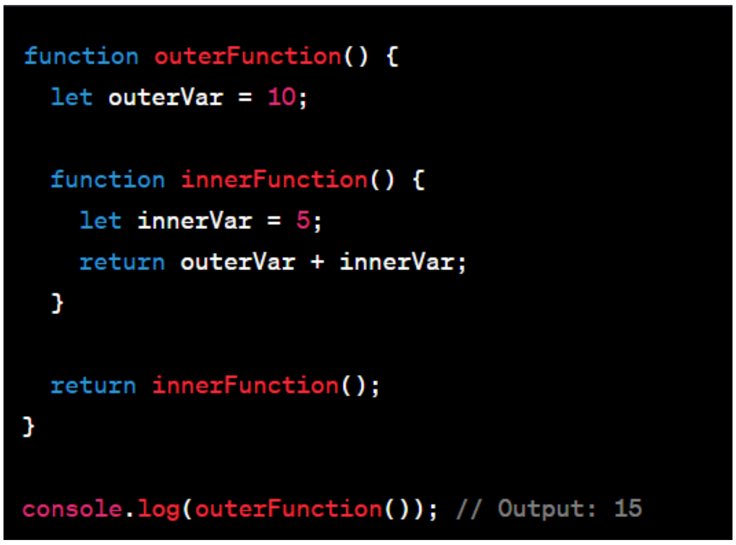
- **Boolean()**: Convert values to booleans:



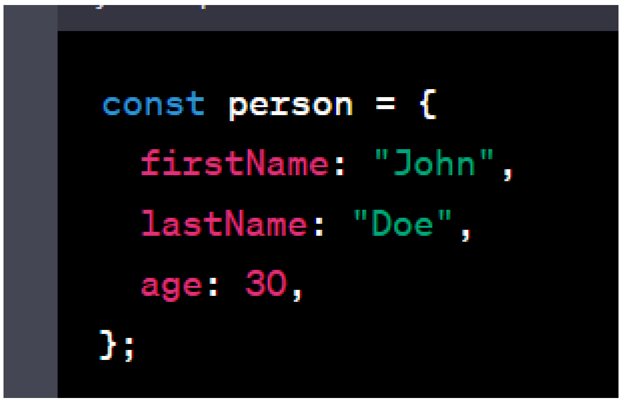
```javascript
const numbers = [1, 2, 3, 4, 5];

numbers.forEach(function(number, index, array) {
  console.log(`Element at index ${index}: ${number}`);
});
```



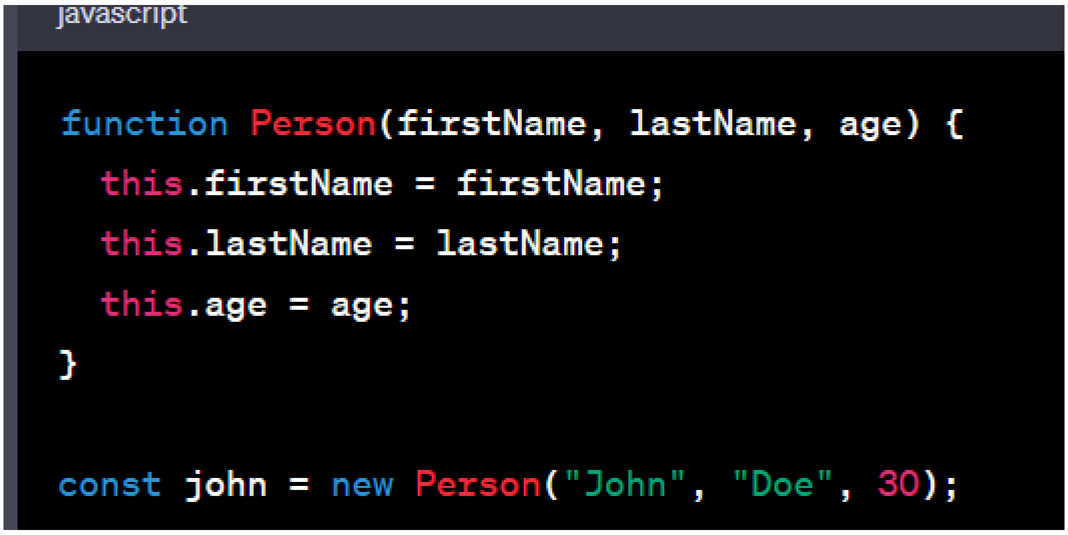
```
const newArray = originalArray.map(function(element, index, array) {
  // Code to transform or manipulate the element
  return transformedElement; // The transformed element to be added to the n
});
```



```
function outerFunction() {
  let outerVar = 10;

  function innerFunction() {
    let innerVar = 5;
    return outerVar + innerVar;
  }

  return innerFunction();
}

console.log(outerFunction()); // Output: 15
```

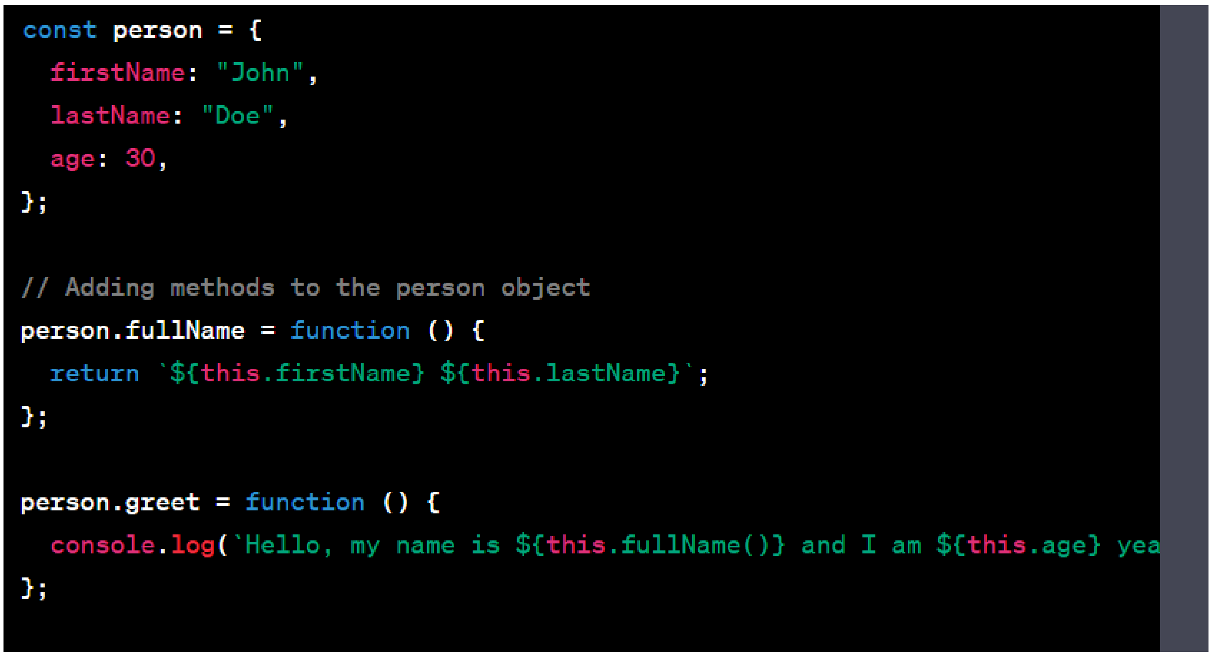


```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 30,
};
```

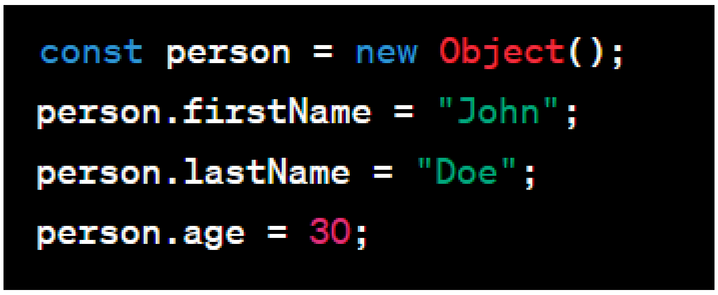


javascript

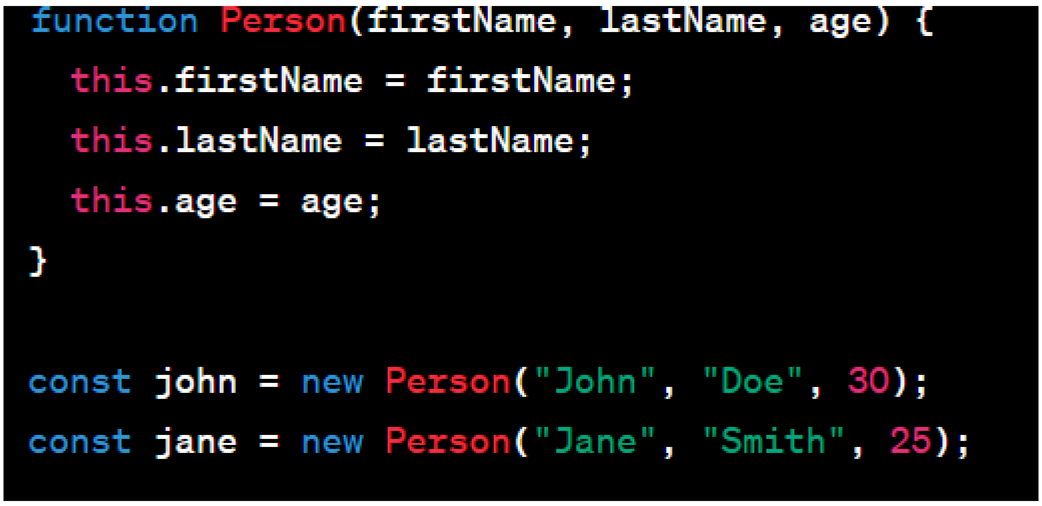
```
function Person(firstName, lastName, age) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.age = age;
}

const john = new Person("John", "Doe", 30);
```



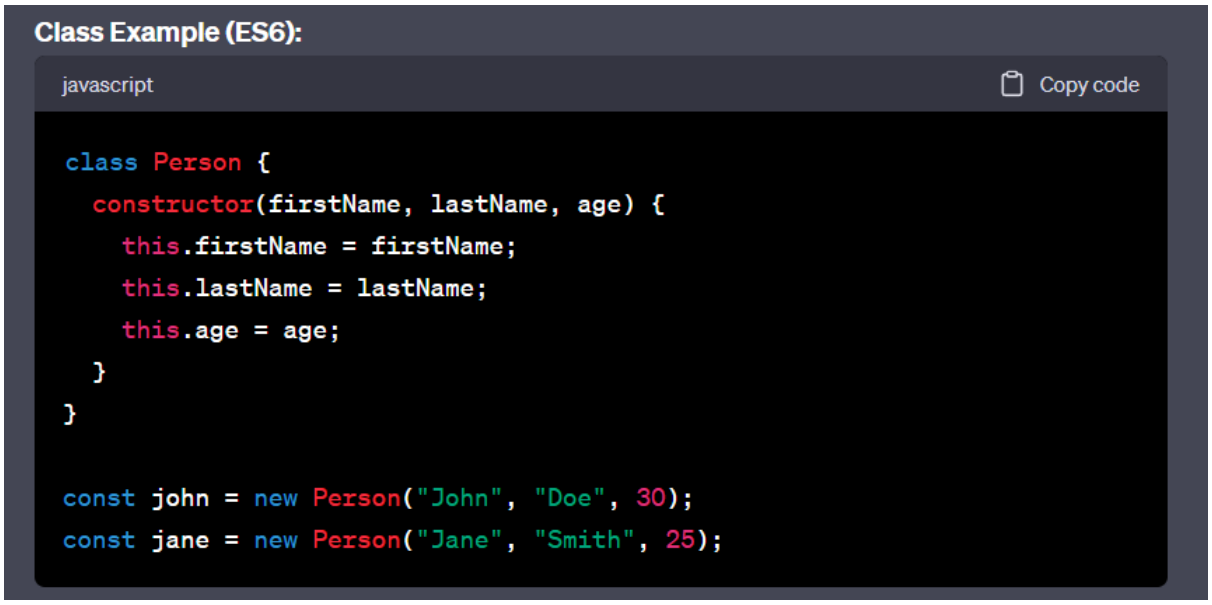
```javascript
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 30,
};

// Adding methods to the person object
person.fullName = function () {
  return `${this.firstName} ${this.lastName}`;
};

person.greet = function () {
  console.log(`Hello, my name is ${this.fullName()} and I am ${this.age} yea
};
```



```javascript
const person = new Object();
person.firstName = "John";
person.lastName = "Doe";
person.age = 30;
```

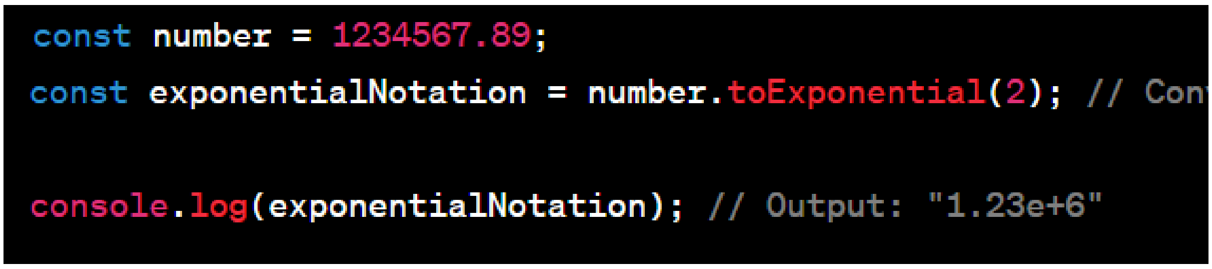


```javascript
function Person(firstName, lastName, age) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.age = age;
}

const john = new Person("John", "Doe", 30);
const jane = new Person("Jane", "Smith", 25);
```

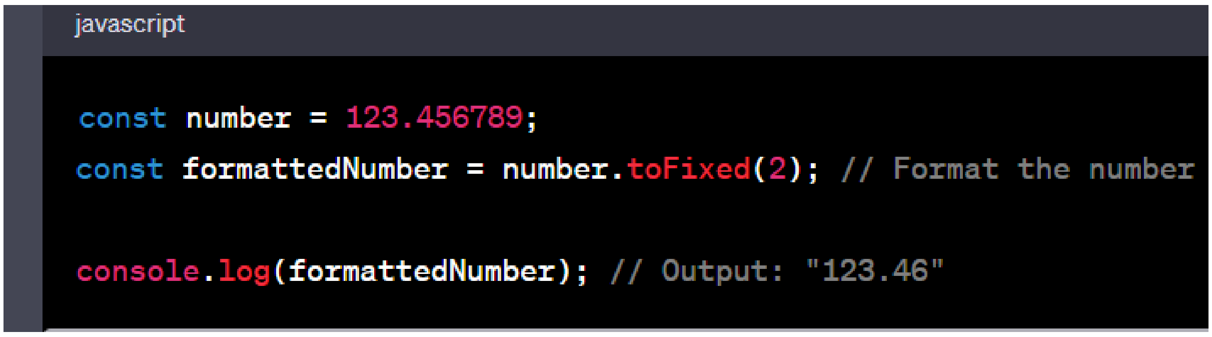

Class Example (ES6):

javascript Copy code

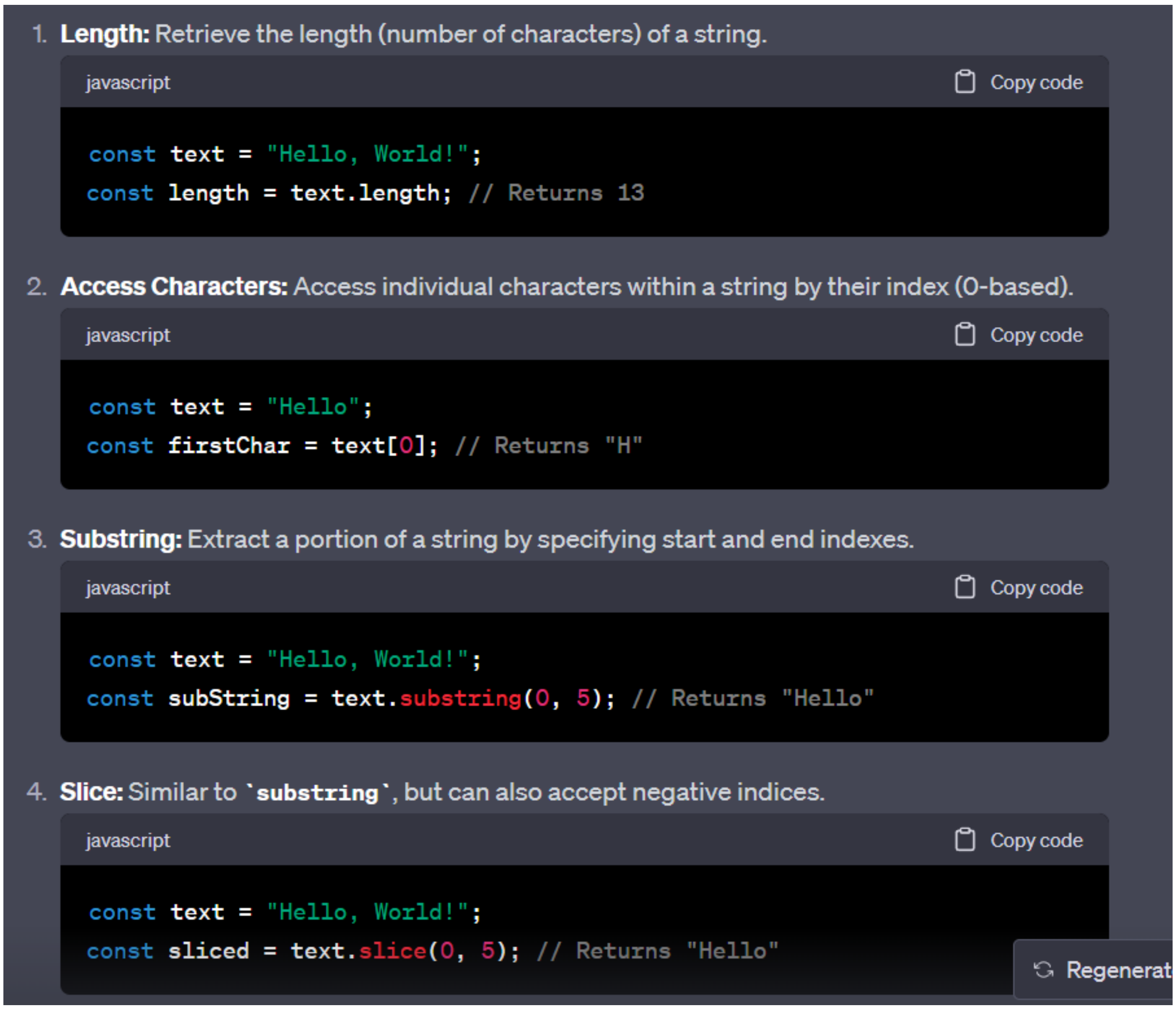
```javascript
class Person {
  constructor(firstName, lastName, age) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.age = age;
  }
}

const john = new Person("John", "Doe", 30);
const jane = new Person("Jane", "Smith", 25);
```



```javascript
const number = 1234567.89;
const exponentialNotation = number.toExponential(2); // Con

console.log(exponentialNotation); // Output: "1.23e+6"
```


javascript

```javascript
const number = 123.456789;
const formattedNumber = number.toFixed(2); // Format the number

console.log(formattedNumber); // Output: "123.46"
```


1. **Length:** Retrieve the length (number of characters) of a string.

```javascript
const text = "Hello, World!";
const length = text.length; // Returns 13
```

2. **Access Characters:** Access individual characters within a string by their index (0-based).

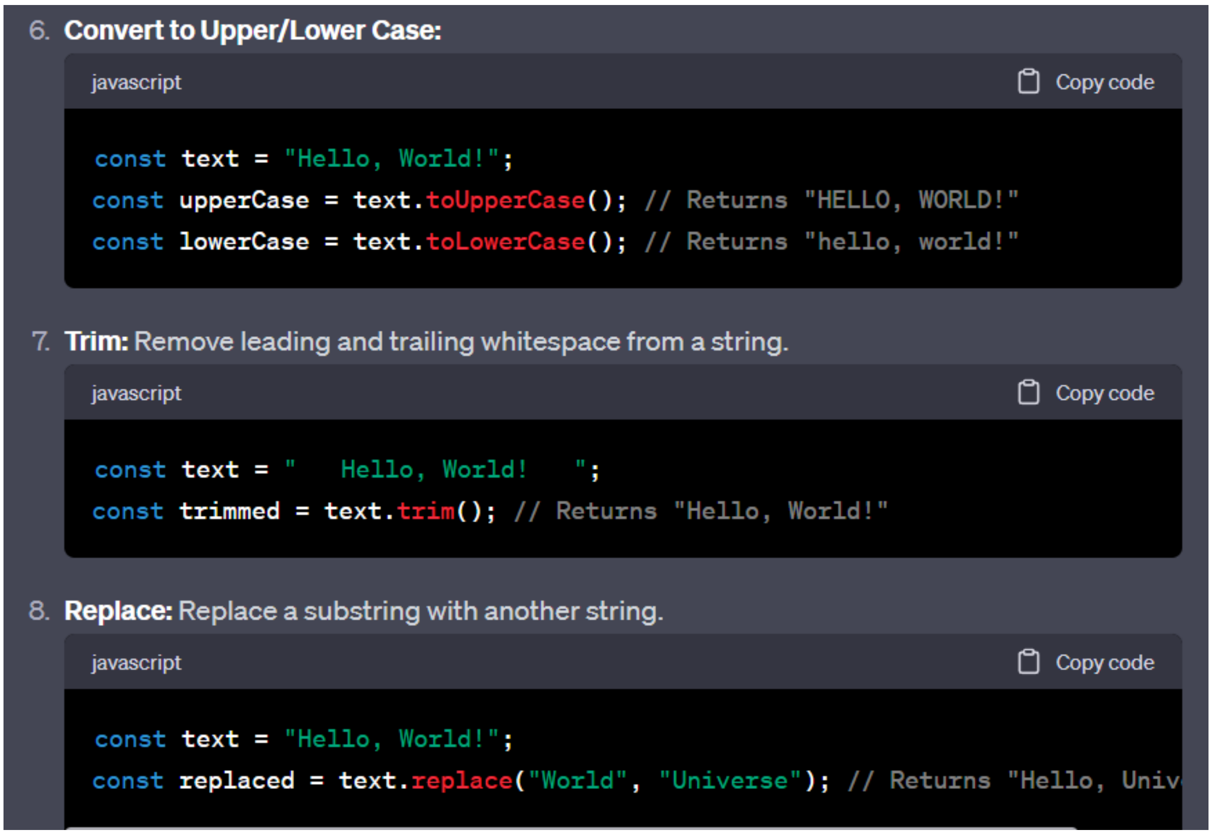
```javascript
const text = "Hello";
const firstChar = text[0]; // Returns "H"
```

3. **Substring:** Extract a portion of a string by specifying start and end indexes.

```javascript
const text = "Hello, World!";
const subString = text.substring(0, 5); // Returns "Hello"
```

4. **Slice:** Similar to `substring`, but can also accept negative indices.

```javascript
const text = "Hello, World!";
const sliced = text.slice(0, 5); // Returns "Hello"
```



6. **Convert to Upper/Lower Case:**

javascript — Copy code

```javascript
const text = "Hello, World!";
const upperCase = text.toUpperCase(); // Returns "HELLO, WORLD!"
const lowerCase = text.toLowerCase(); // Returns "hello, world!"
```

7. **Trim:** Remove leading and trailing whitespace from a string.

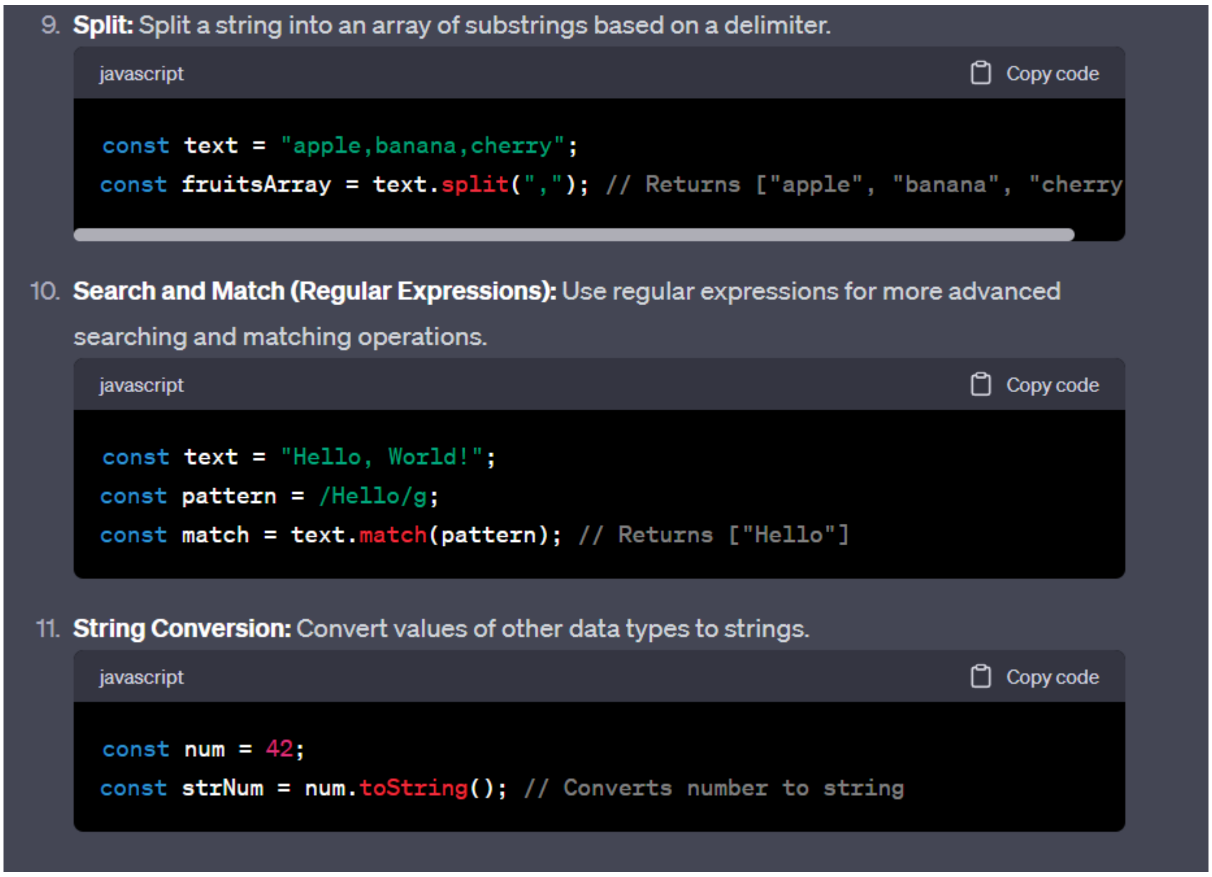
javascript — Copy code

```javascript
const text = "   Hello, World!   ";
const trimmed = text.trim(); // Returns "Hello, World!"
```

8. **Replace:** Replace a substring with another string.

javascript — Copy code

```javascript
const text = "Hello, World!";
const replaced = text.replace("World", "Universe"); // Returns "Hello, Univ
```



9. **Split:** Split a string into an array of substrings based on a delimiter.

javascript — Copy code

```javascript
const text = "apple,banana,cherry";
const fruitsArray = text.split(","); // Returns ["apple", "banana", "cherry
```

10. **Search and Match (Regular Expressions):** Use regular expressions for more advanced searching and matching operations.

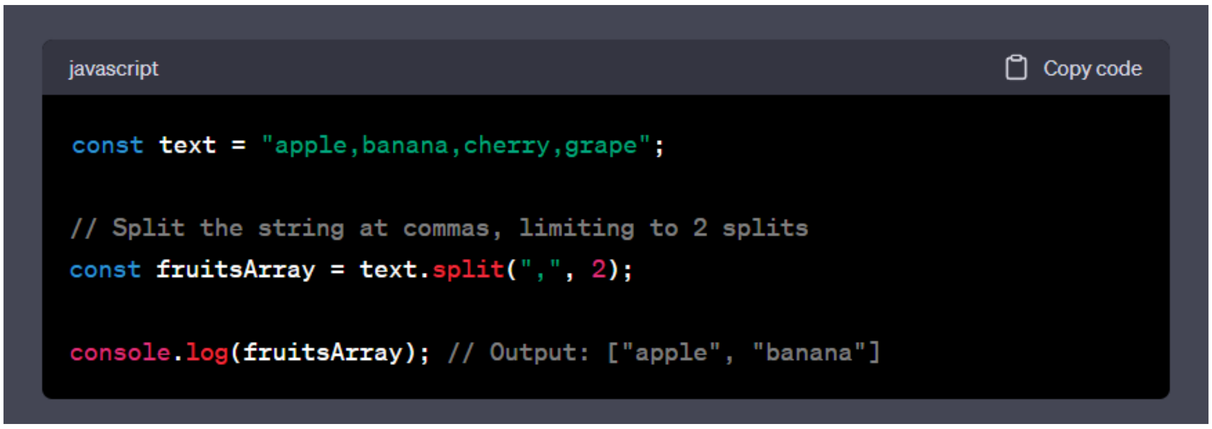
javascript — Copy code

```javascript
const text = "Hello, World!";
const pattern = /Hello/g;
const match = text.match(pattern); // Returns ["Hello"]
```

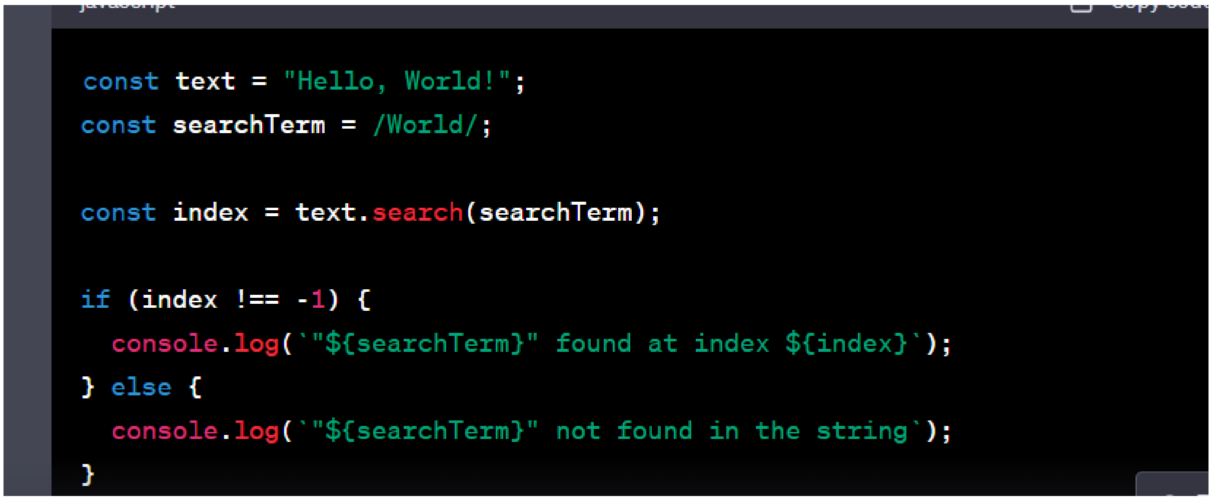
11. **String Conversion:** Convert values of other data types to strings.

javascript — Copy code

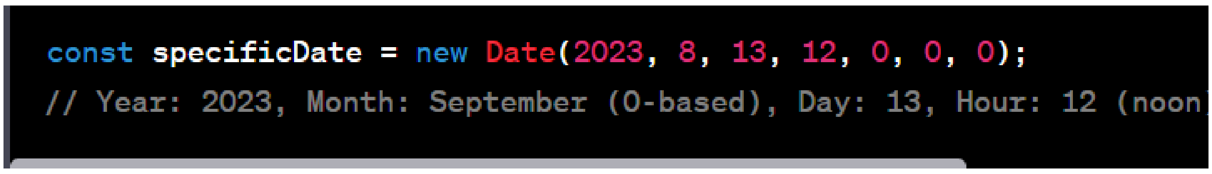
```javascript
const num = 42;
const strNum = num.toString(); // Converts number to string
```



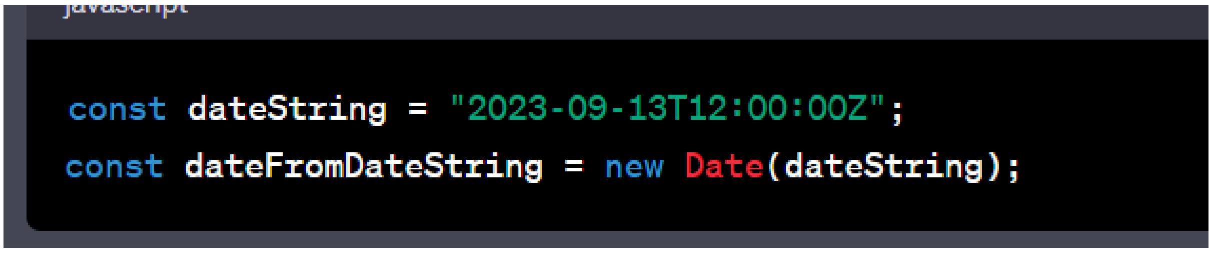
```javascript
const text = "apple,banana,cherry,grape";

// Split the string at commas, limiting to 2 splits
const fruitsArray = text.split(",", 2);

console.log(fruitsArray); // Output: ["apple", "banana"]
```



```javascript
const text = "Hello, World!";
const searchTerm = /World/;

const index = text.search(searchTerm);

if (index !== -1) {
  console.log(`"${searchTerm}" found at index ${index}`);
} else {
  console.log(`"${searchTerm}" not found in the string`);
}
```

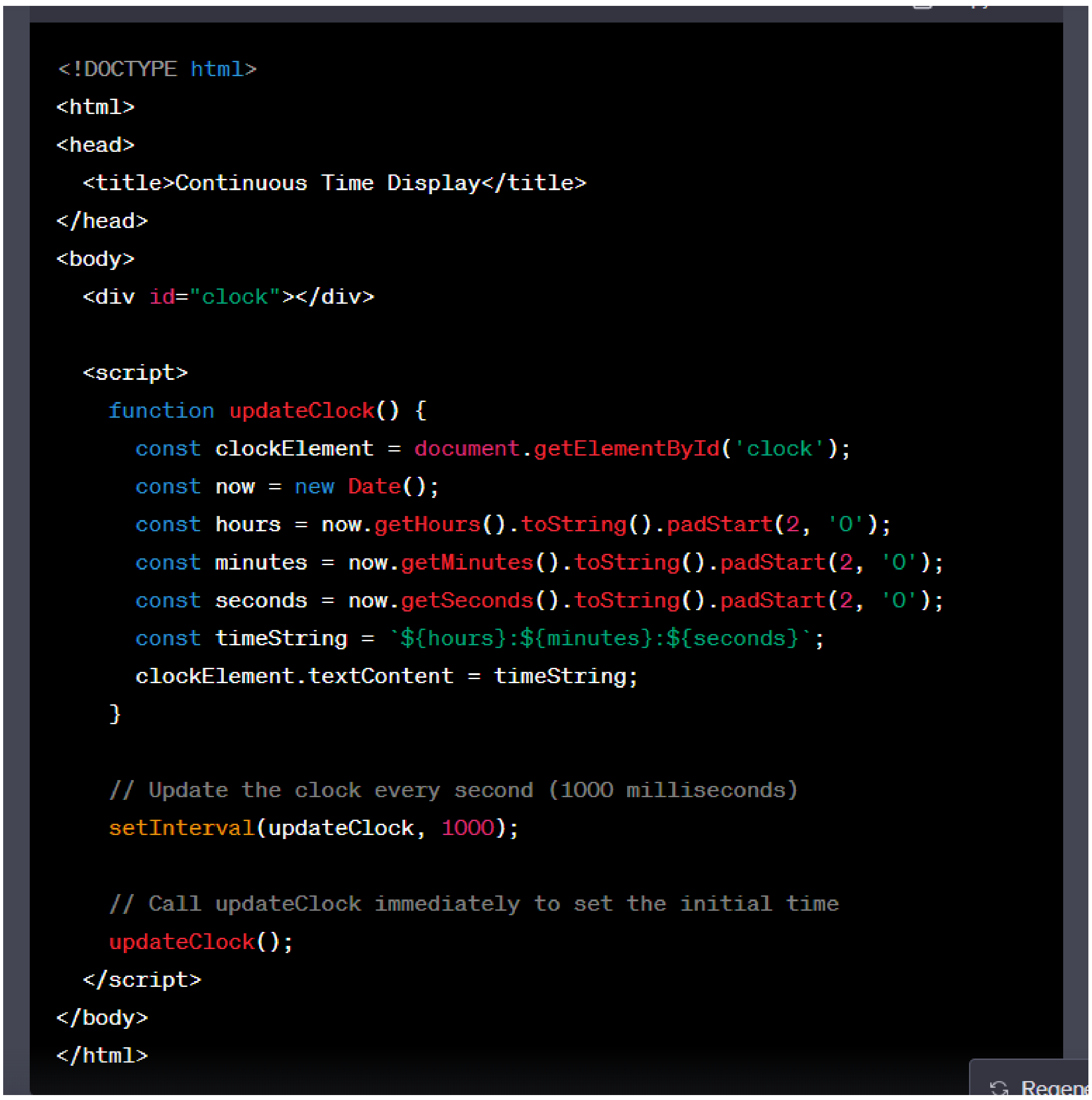


```javascript
const specificDate = new Date(2023, 8, 13, 12, 0, 0, 0);
// Year: 2023, Month: September (0-based), Day: 13, Hour: 12 (noon
```

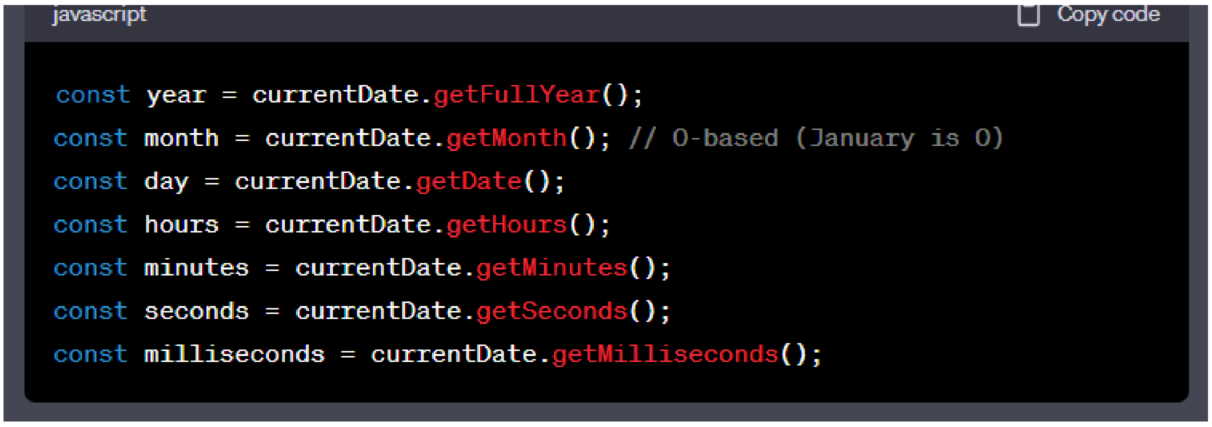


```javascript
const dateString = "2023-09-13T12:00:00Z";
const dateFromDateString = new Date(dateString);
```

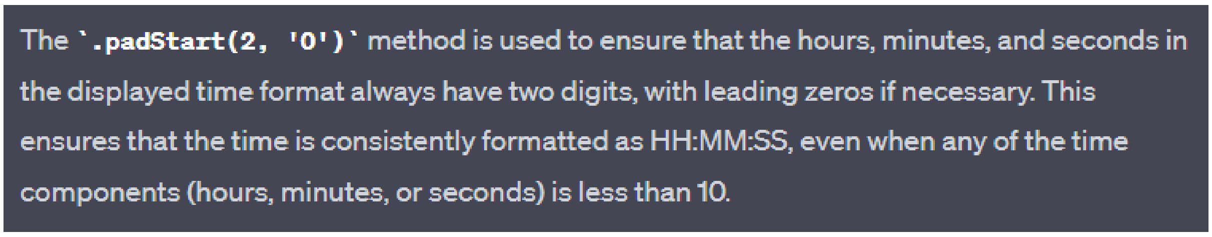
Date and Time: YYYY-MM-DDTHH:MM:SS (e.g., "2023-09-13T12:00:00" for September 13, 2023, at 12:00:00)



```html
<!DOCTYPE html>
<html>
<head>
  <title>Continuous Time Display</title>
</head>
<body>
  <div id="clock"></div>

  <script>
    function updateClock() {
      const clockElement = document.getElementById('clock');
      const now = new Date();
      const hours = now.getHours().toString().padStart(2, '0');
      const minutes = now.getMinutes().toString().padStart(2, '0');
      const seconds = now.getSeconds().toString().padStart(2, '0');
      const timeString = `${hours}:${minutes}:${seconds}`;
      clockElement.textContent = timeString;
    }

    // Update the clock every second (1000 milliseconds)
    setInterval(updateClock, 1000);

    // Call updateClock immediately to set the initial time
    updateClock();
  </script>
</body>
</html>
```

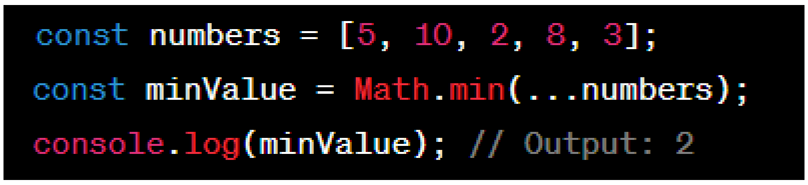


```javascript
const year = currentDate.getFullYear();
const month = currentDate.getMonth(); // 0-based (January is 0)
const day = currentDate.getDate();
const hours = currentDate.getHours();
const minutes = currentDate.getMinutes();
const seconds = currentDate.getSeconds();
const milliseconds = currentDate.getMilliseconds();
```

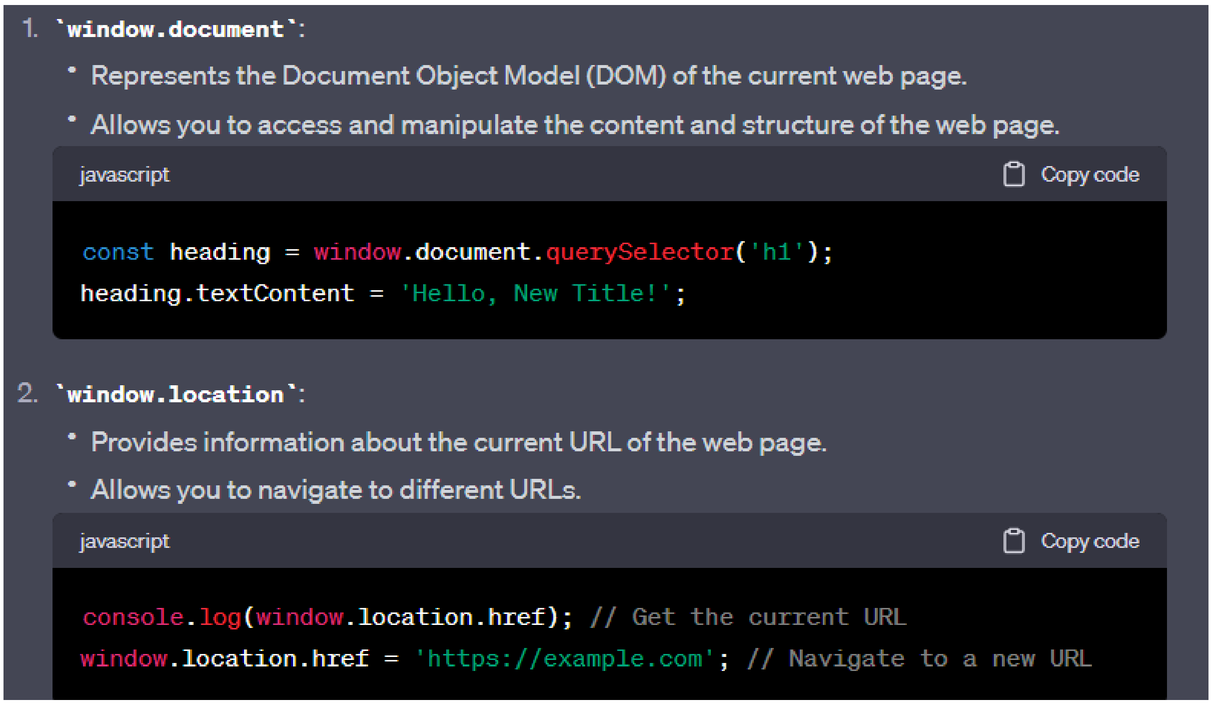


The `.padStart(2, '0')` method is used to ensure that the hours, minutes, and seconds in the displayed time format always have two digits, with leading zeros if necessary. This ensures that the time is consistently formatted as HH:MM:SS, even when any of the time components (hours, minutes, or seconds) is less than 10.

MATH ko lagi book



```javascript
const numbers = [5, 10, 2, 8, 3];
const minValue = Math.min(...numbers);
console.log(minValue); // Output: 2
```

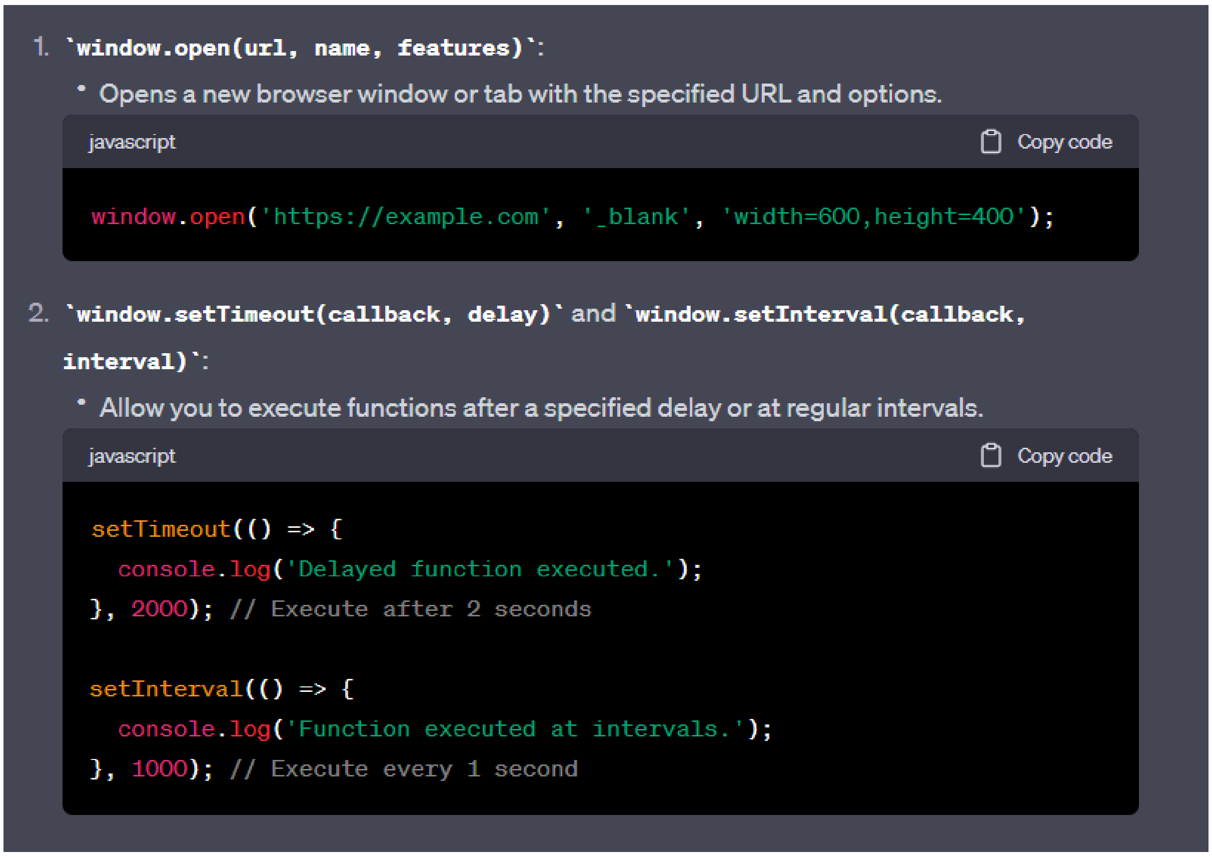


1. `window.document`:
   - Represents the Document Object Model (DOM) of the current web page.
   - Allows you to access and manipulate the content and structure of the web page.

   ```javascript
   const heading = window.document.querySelector('h1');
   heading.textContent = 'Hello, New Title!';
   ```

2. `window.location`:
   - Provides information about the current URL of the web page.
   - Allows you to navigate to different URLs.

   ```javascript
   console.log(window.location.href); // Get the current URL
   window.location.href = 'https://example.com'; // Navigate to a new URL
   ```

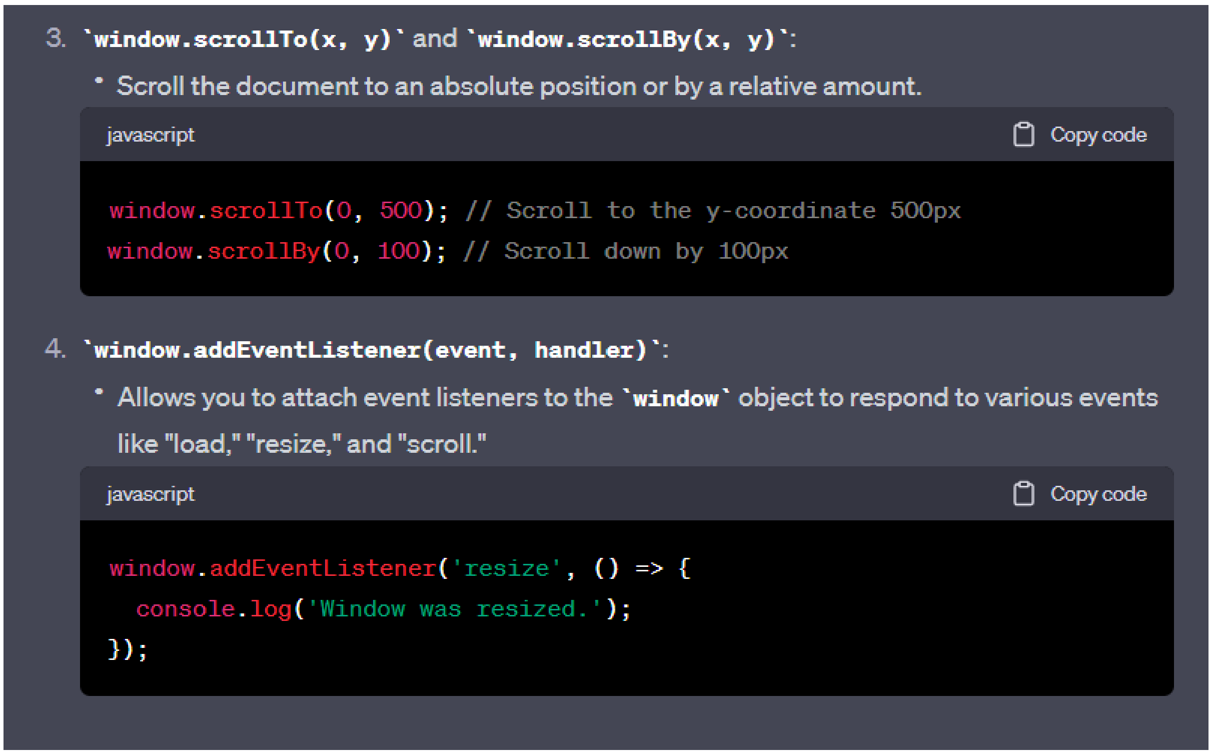


1. `window.open(url, name, features)`:
   - Opens a new browser window or tab with the specified URL and options.

```javascript
window.open('https://example.com', '_blank', 'width=600,height=400');
```

2. `window.setTimeout(callback, delay)` and `window.setInterval(callback, interval)`:
   - Allow you to execute functions after a specified delay or at regular intervals.

```javascript
setTimeout(() => {
  console.log('Delayed function executed.');
}, 2000); // Execute after 2 seconds

setInterval(() => {
  console.log('Function executed at intervals.');
}, 1000); // Execute every 1 second
```



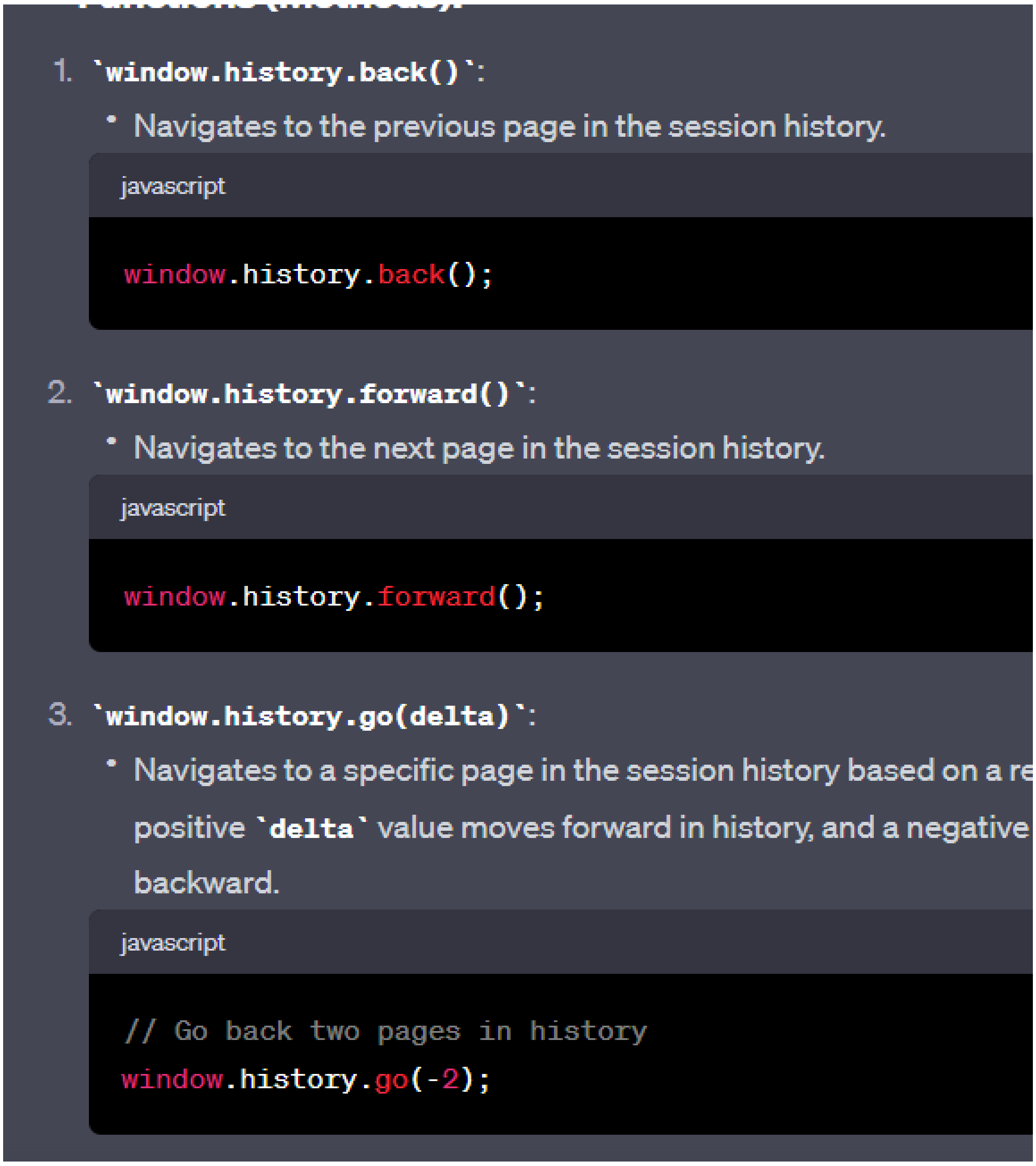
3. `window.scrollTo(x, y)` and `window.scrollBy(x, y)`:
   - Scroll the document to an absolute position or by a relative amount.

```javascript
window.scrollTo(0, 500); // Scroll to the y-coordinate 500px
window.scrollBy(0, 100); // Scroll down by 100px
```

4. `window.addEventListener(event, handler)`:
   - Allows you to attach event listeners to the `window` object to respond to various events like "load," "resize," and "scroll."

```javascript
window.addEventListener('resize', () => {
  console.log('Window was resized.');
});
```

LOOK BOOK



1. **`window.history.back()`**:
   - Navigates to the previous page in the session history.

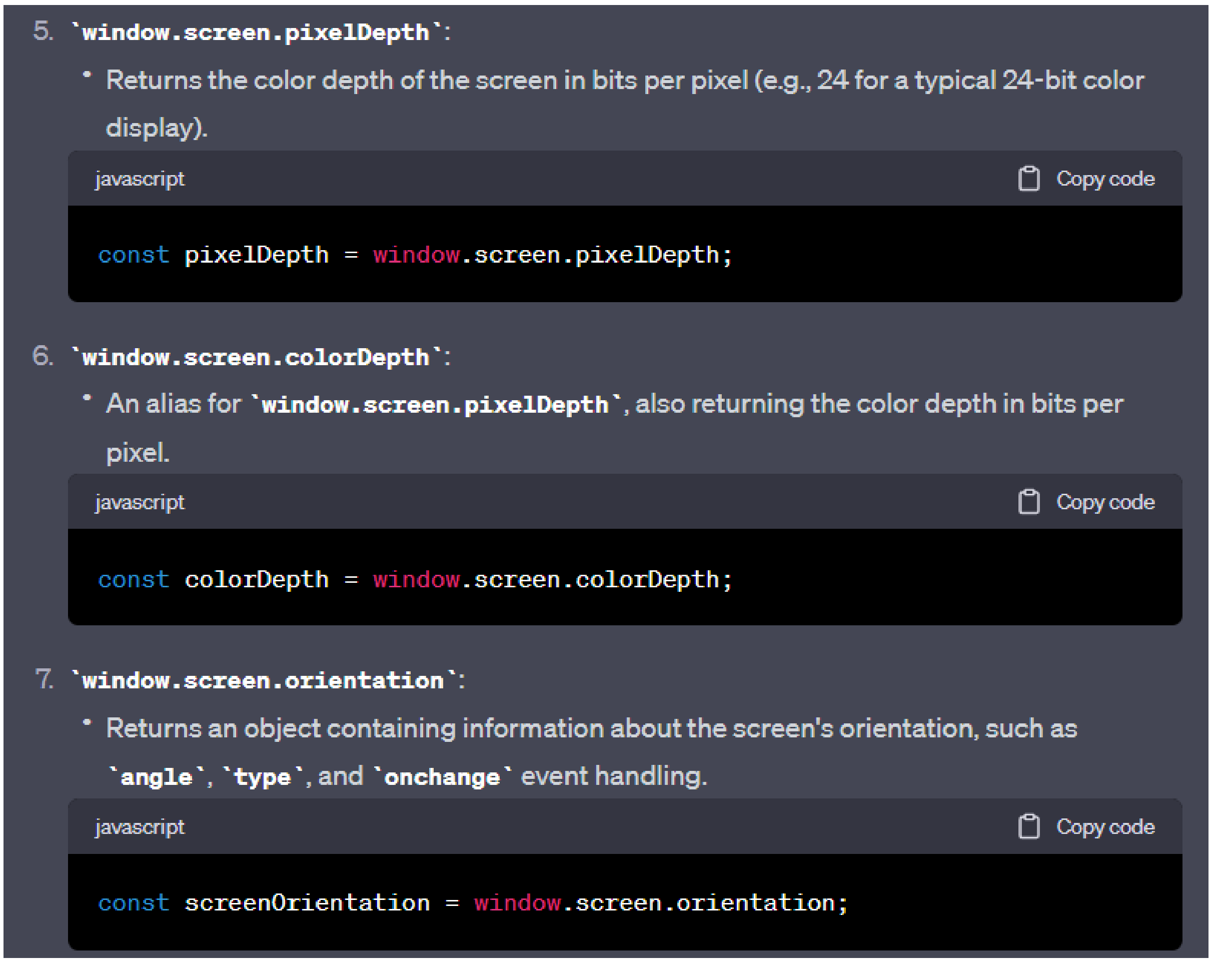
   javascript
   ```javascript
   window.history.back();
   ```

2. **`window.history.forward()`**:
   - Navigates to the next page in the session history.

   javascript
   ```javascript
   window.history.forward();
   ```

3. **`window.history.go(delta)`**:
   - Navigates to a specific page in the session history based on a re
     positive **`delta`** value moves forward in history, and a negative
     backward.

   javascript
   ```javascript
   // Go back two pages in history
   window.history.go(-2);
   ```



5. **`window.screen.pixelDepth`**:
   - Returns the color depth of the screen in bits per pixel (e.g., 24 for a typical 24-bit color display).

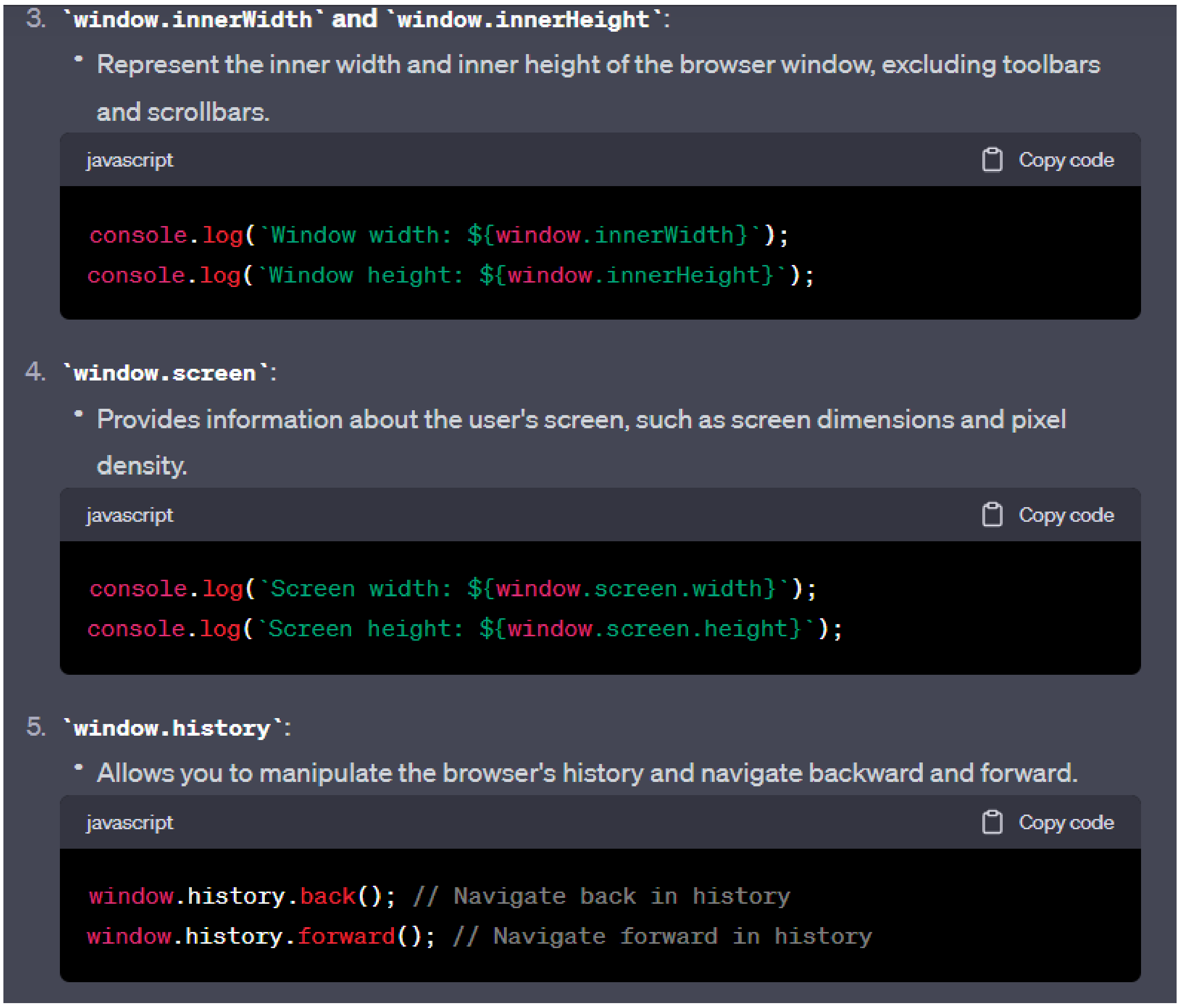
javascript Copy code
```javascript
const pixelDepth = window.screen.pixelDepth;
```

6. **`window.screen.colorDepth`**:
   - An alias for **`window.screen.pixelDepth`**, also returning the color depth in bits per pixel.

javascript Copy code
```javascript
const colorDepth = window.screen.colorDepth;
```

7. **`window.screen.orientation`**:
   - Returns an object containing information about the screen's orientation, such as **`angle`**, **`type`**, and **`onchange`** event handling.

javascript Copy code
```javascript
const screenOrientation = window.screen.orientation;
```



3. `` `window.innerWidth` `` **and** `` `window.innerHeight` ``:
   - Represent the inner width and inner height of the browser window, excluding toolbars and scrollbars.

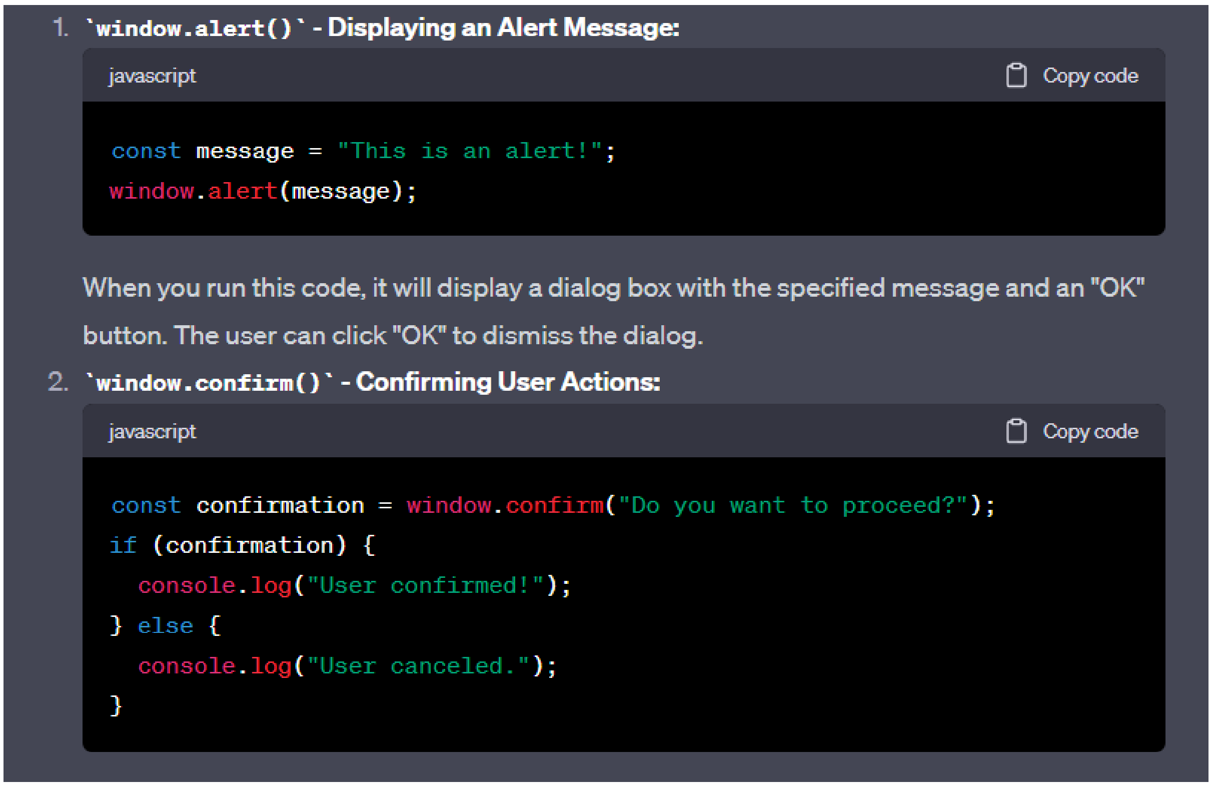
```javascript
console.log(`Window width: ${window.innerWidth}`);
console.log(`Window height: ${window.innerHeight}`);
```

4. `` `window.screen` ``:
   - Provides information about the user's screen, such as screen dimensions and pixel density.

```javascript
console.log(`Screen width: ${window.screen.width}`);
console.log(`Screen height: ${window.screen.height}`);
```

5. `` `window.history` ``:
   - Allows you to manipulate the browser's history and navigate backward and forward.

```javascript
window.history.back(); // Navigate back in history
window.history.forward(); // Navigate forward in history
```



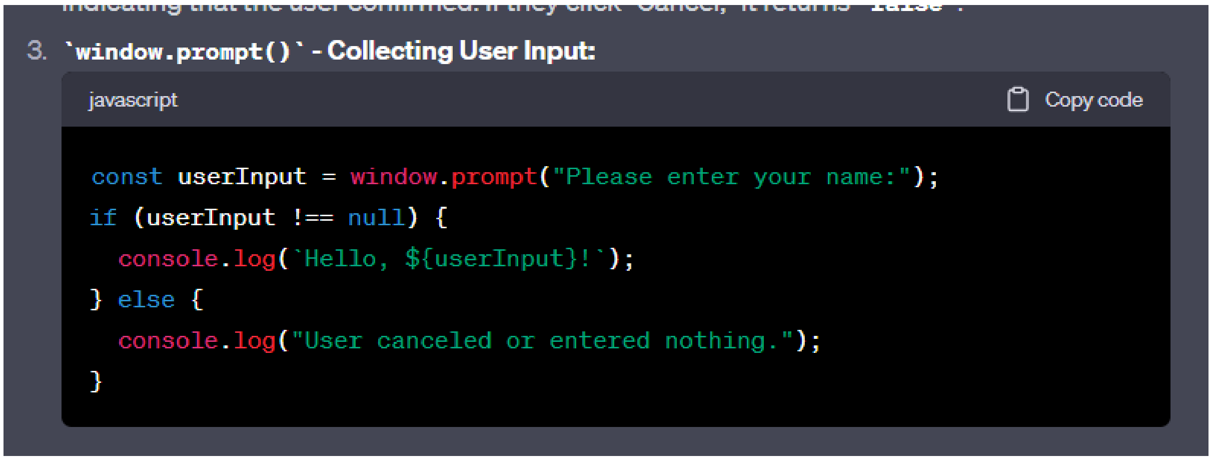
1. `` `window.alert()` `` **- Displaying an Alert Message:**

```javascript
const message = "This is an alert!";
window.alert(message);
```

When you run this code, it will display a dialog box with the specified message and an "OK" button. The user can click "OK" to dismiss the dialog.

2. `` `window.confirm()` `` **- Confirming User Actions:**

```javascript
const confirmation = window.confirm("Do you want to proceed?");
if (confirmation) {
  console.log("User confirmed!");
} else {
  console.log("User canceled.");
}
```

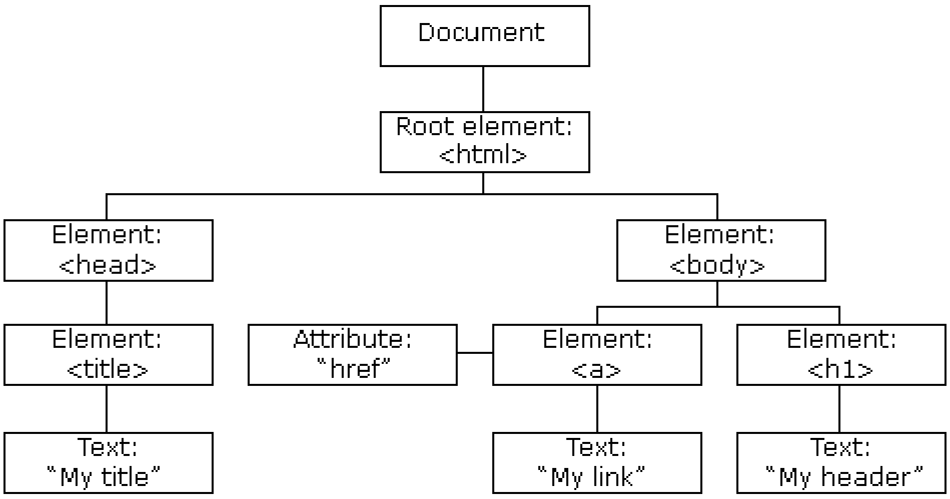


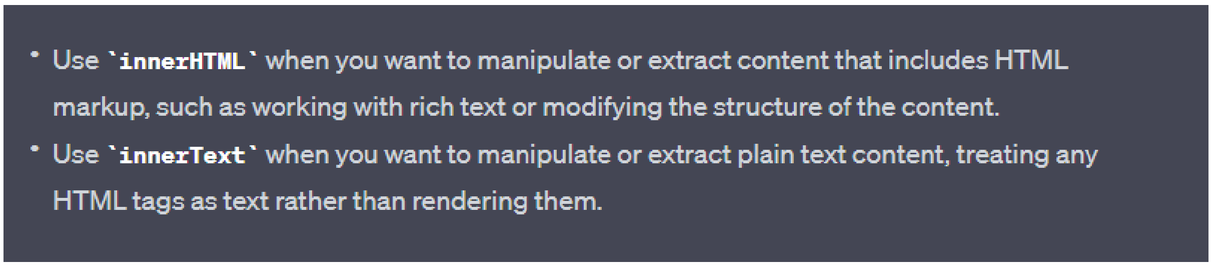
indicating that the user confirmed. If they click "Cancel," it returns `false`.

3. **`window.prompt()` - Collecting User Input:**

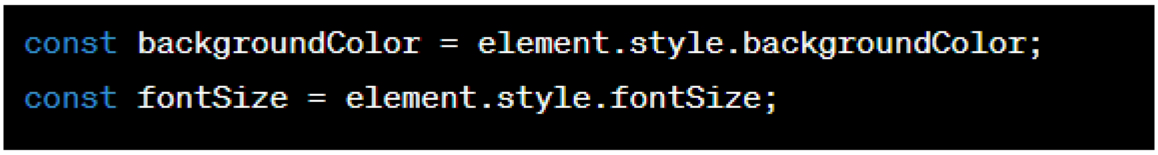
javascript Copy code

```javascript
const userInput = window.prompt("Please enter your name:");
if (userInput !== null) {
  console.log(`Hello, ${userInput}!`);
} else {
  console.log("User canceled or entered nothing.");
}
```

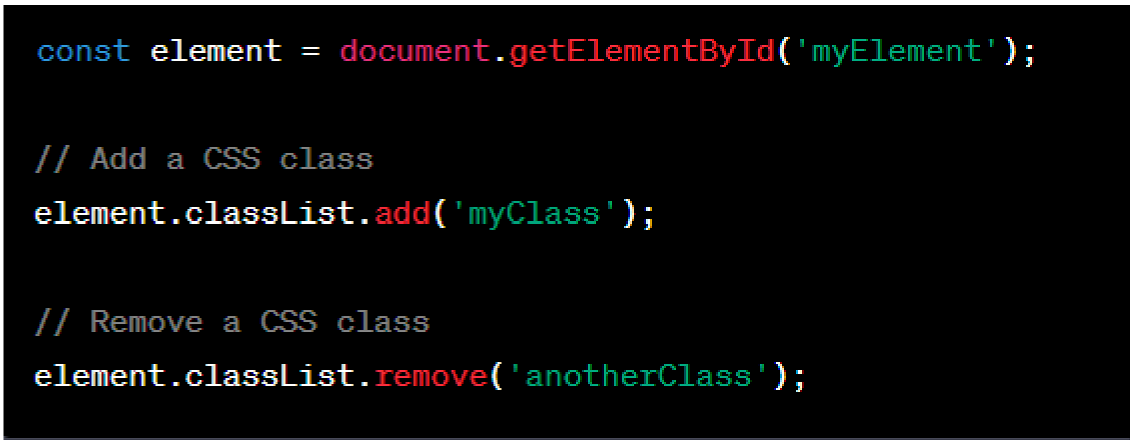




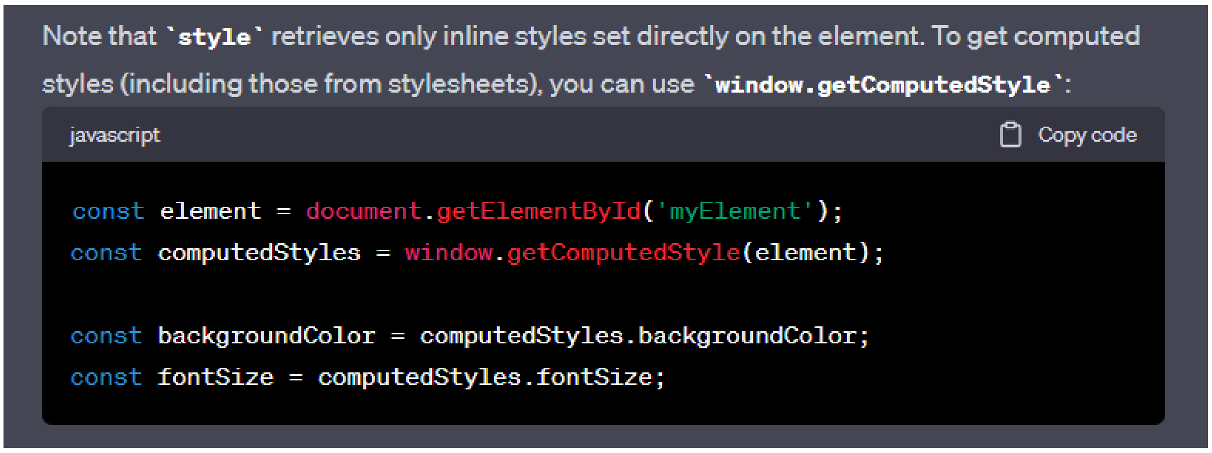
- Use **`innerHTML`** when you want to manipulate or extract content that includes HTML markup, such as working with rich text or modifying the structure of the content.
- Use **`innerText`** when you want to manipulate or extract plain text content, treating any HTML tags as text rather than rendering them.



```javascript
const backgroundColor = element.style.backgroundColor;
const fontSize = element.style.fontSize;
```



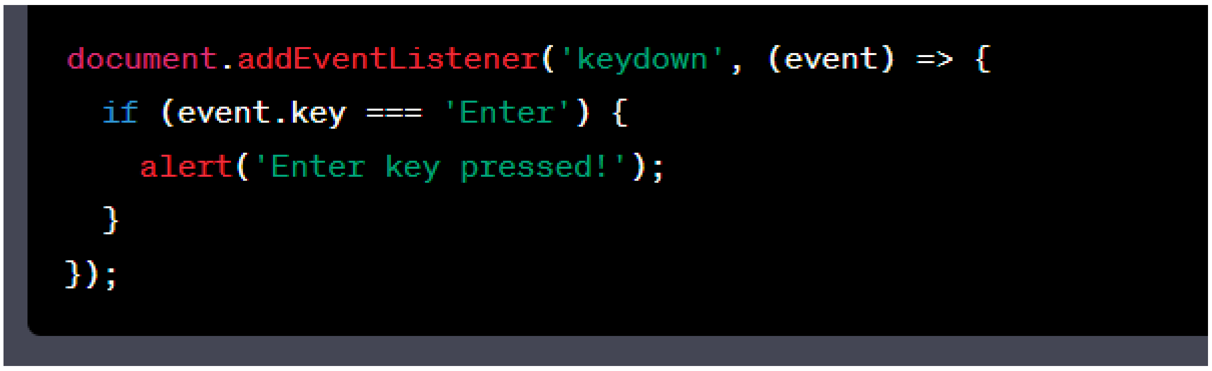
```javascript
const element = document.getElementById('myElement');

// Add a CSS class
element.classList.add('myClass');

// Remove a CSS class
element.classList.remove('anotherClass');
```



Note that `style` retrieves only inline styles set directly on the element. To get computed styles (including those from stylesheets), you can use `window.getComputedStyle`:

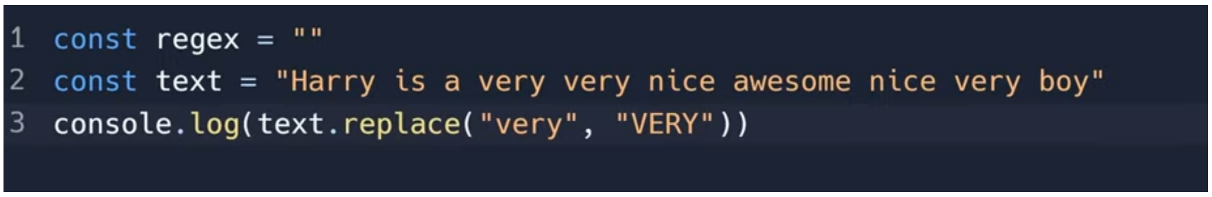
javascript Copy code

```javascript
const element = document.getElementById('myElement');
const computedStyles = window.getComputedStyle(element);

const backgroundColor = computedStyles.backgroundColor;
const fontSize = computedStyles.fontSize;
```



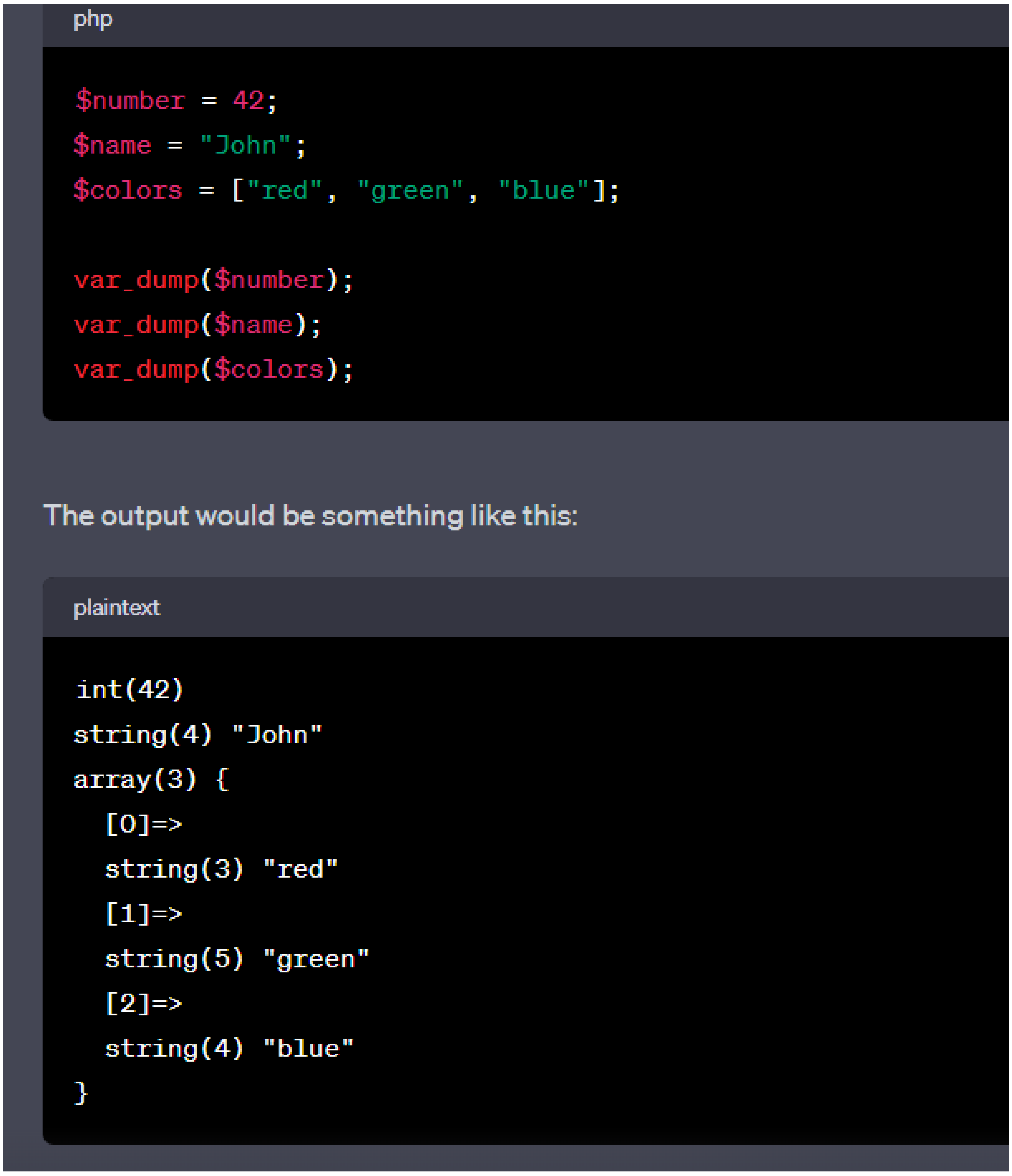
```javascript
document.addEventListener('keydown', (event) => {
  if (event.key === 'Enter') {
    alert('Enter key pressed!');
  }
});
```

LOOK at BOOK PG 94
SEE PG 100



```javascript
const regex = ""
const text = "Harry is a very very nice awesome nice very boy"
console.log(text.replace("very", "VERY"))
```

PHP

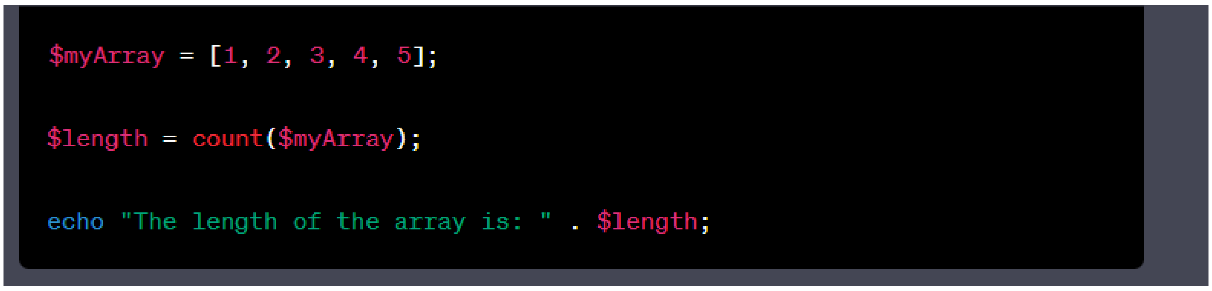

php

```php
$number = 42;
$name = "John";
$colors = ["red", "green", "blue"];

var_dump($number);
var_dump($name);
var_dump($colors);
```

The output would be something like this:

plaintext

```
int(42)
string(4) "John"
array(3) {
  [0]=>
  string(3) "red"
  [1]=>
  string(5) "green"
  [2]=>
  string(4) "blue"
}
```

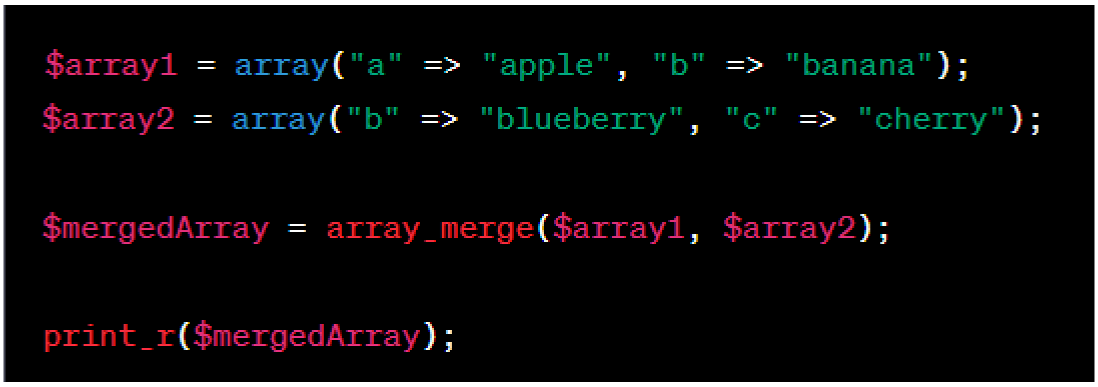


```php
$myArray = [1, 2, 3, 4, 5];

$length = count($myArray);

echo "The length of the array is: " . $length;
```

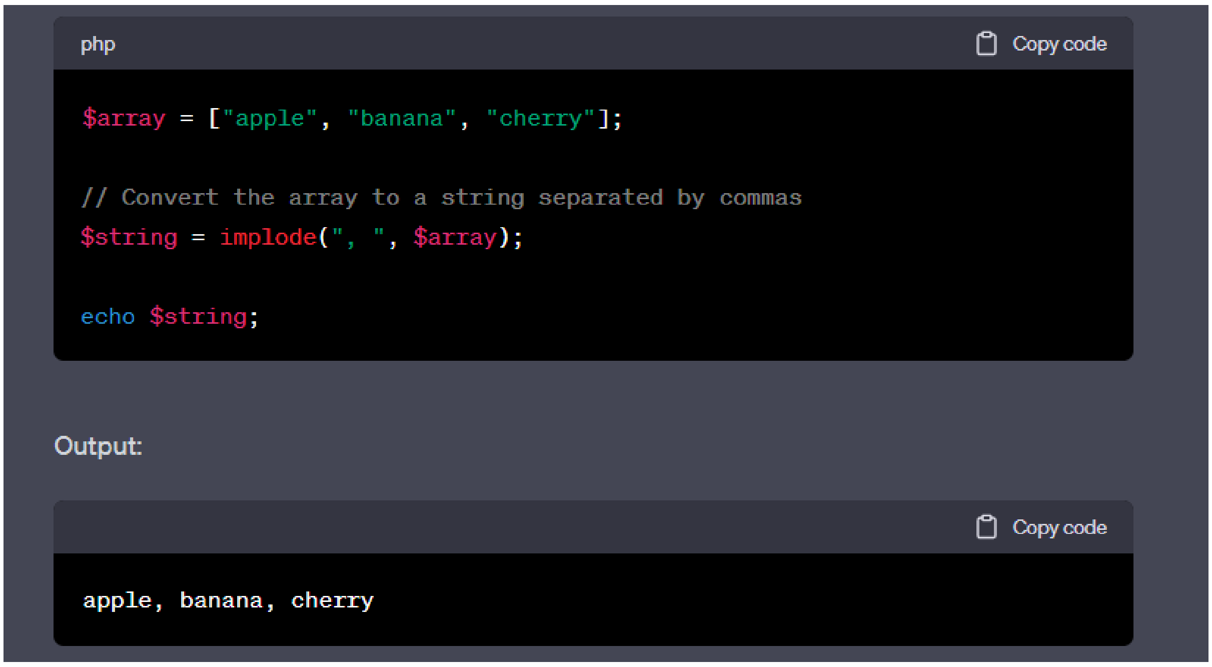
```php
function add() {
    $numArgs = func_num_args();

    if ($numArgs == 2) {
        $args = func_get_args();
        return $args[0] + $args[1];
    } elseif ($numArgs == 3) {
        $args = func_get_args();
        return $args[0] + $args[1] + $args[2];
    } else {
        return "Unsupported number of arguments";
    }
}

// Example usage:
$result1 = add(1, 2);
echo $result1;  // Output: 3

$result2 = add(1, 2, 3);
echo $result2;  // Output: 6

$result3 = add(1, 2, 3, 4);
echo $result3;  // Output: Unsupported number of arguments
```

fun_get_arg(arg_index)

```php
$globalVar = 42; // Declare a global variable

function accessGlobal() {
    global $globalVar; // Use the GLOBAL keyword to access the
    echo "The global variable is: " . $globalVar;
}

accessGlobal(); // Output: The global variable is: 42
```



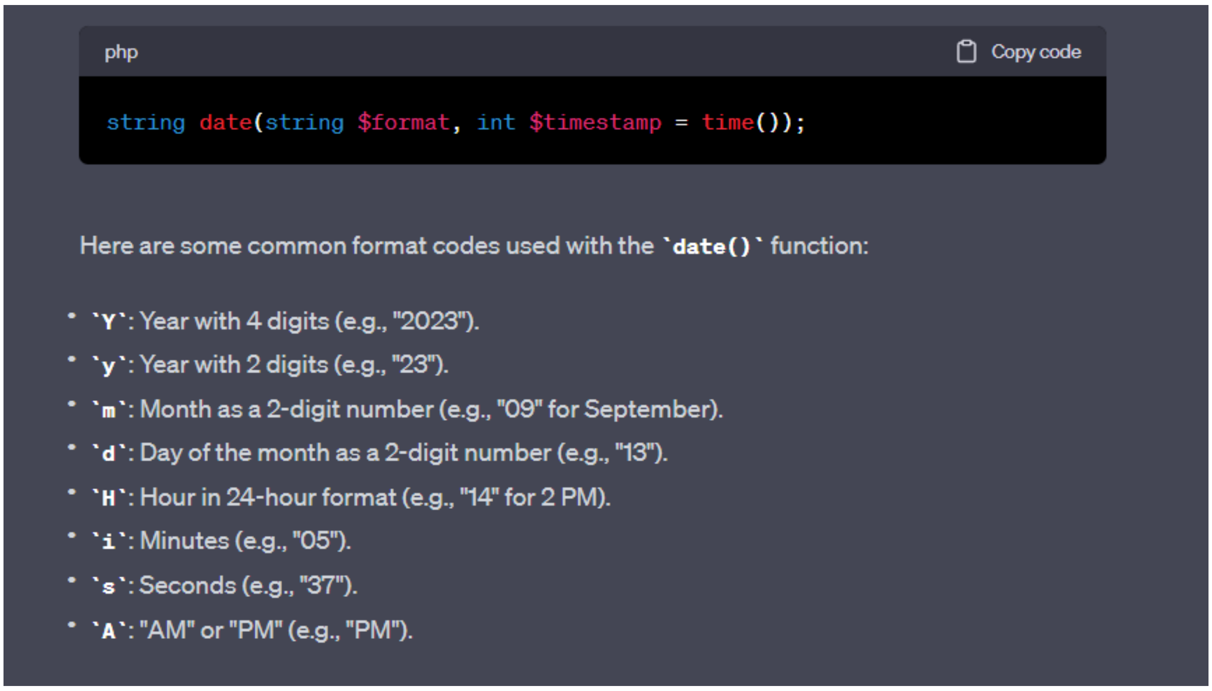
```php
$array1 = array("a" => "apple", "b" => "banana");
$array2 = array("b" => "blueberry", "c" => "cherry");

$mergedArray = array_merge($array1, $array2);

print_r($mergedArray);
```



```php
$array = ["apple", "banana", "cherry"];

// Convert the array to a string separated by commas
$string = implode(", ", $array);

echo $string;
```

Output:

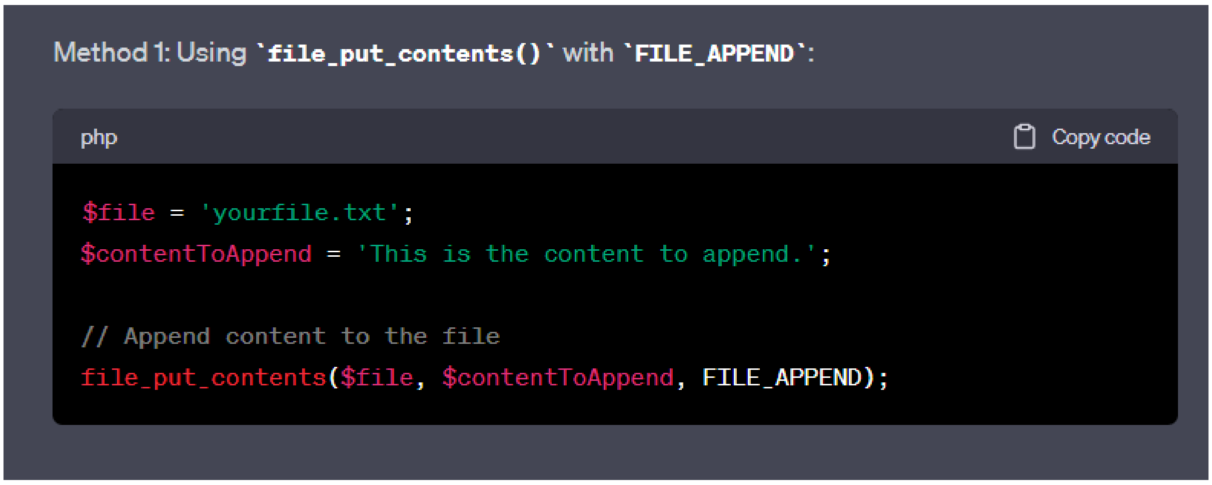
```
apple, banana, cherry
```



```php
string date(string $format, int $timestamp = time());
```

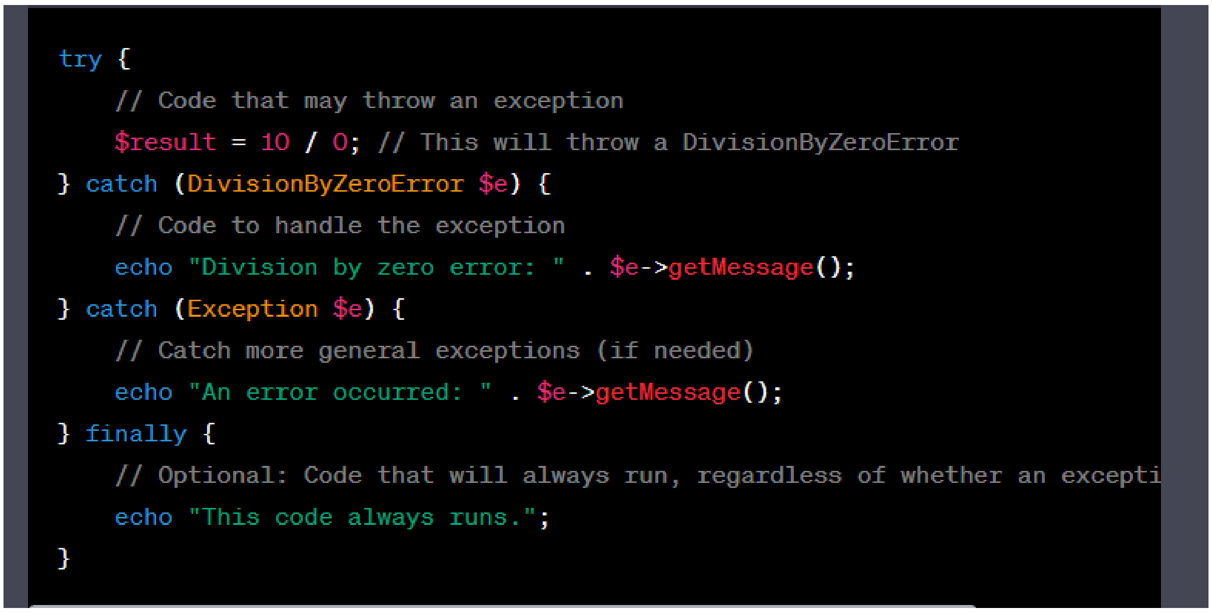
Here are some common format codes used with the `date()` function:

- `Y`: Year with 4 digits (e.g., "2023").
- `y`: Year with 2 digits (e.g., "23").
- `m`: Month as a 2-digit number (e.g., "09" for September).
- `d`: Day of the month as a 2-digit number (e.g., "13").
- `H`: Hour in 24-hour format (e.g., "14" for 2 PM).
- `i`: Minutes (e.g., "05").
- `s`: Seconds (e.g., "37").
- `A`: "AM" or "PM" (e.g., "PM").

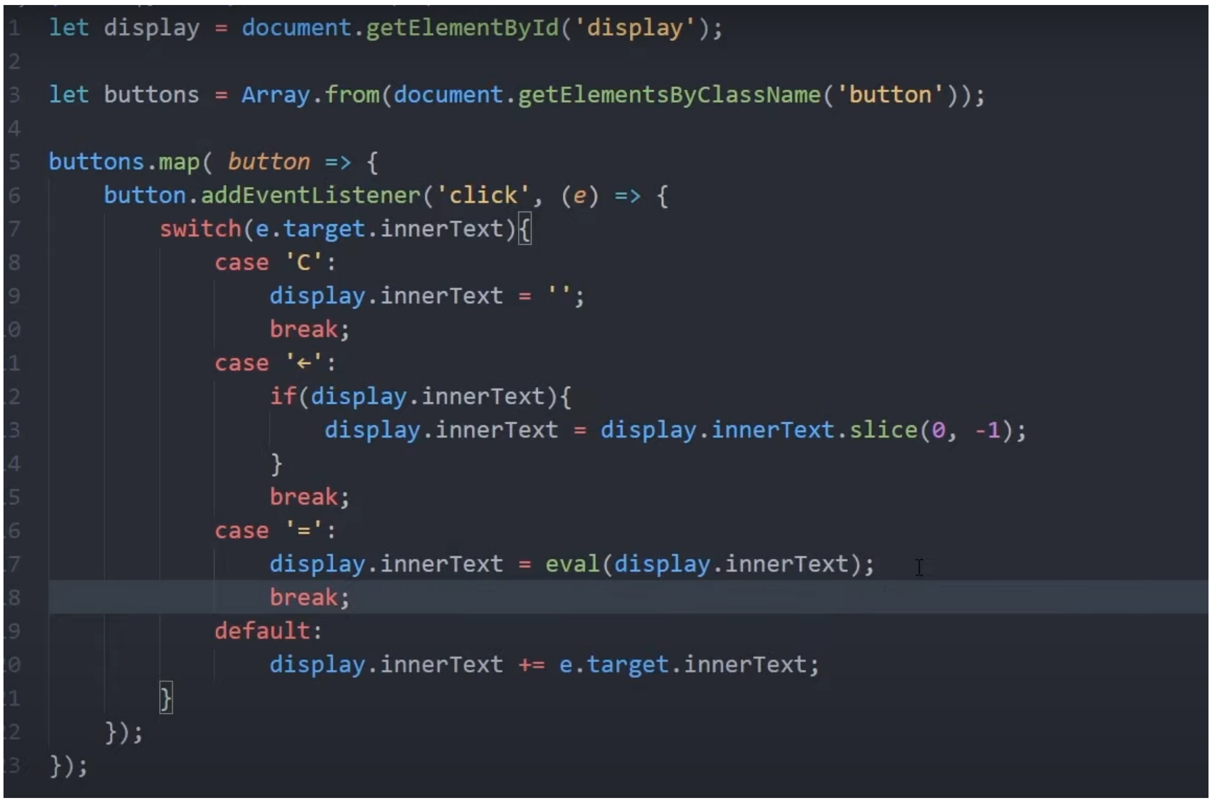

Method 1: Using `file_put_contents()` with `FILE_APPEND`:

php Copy code

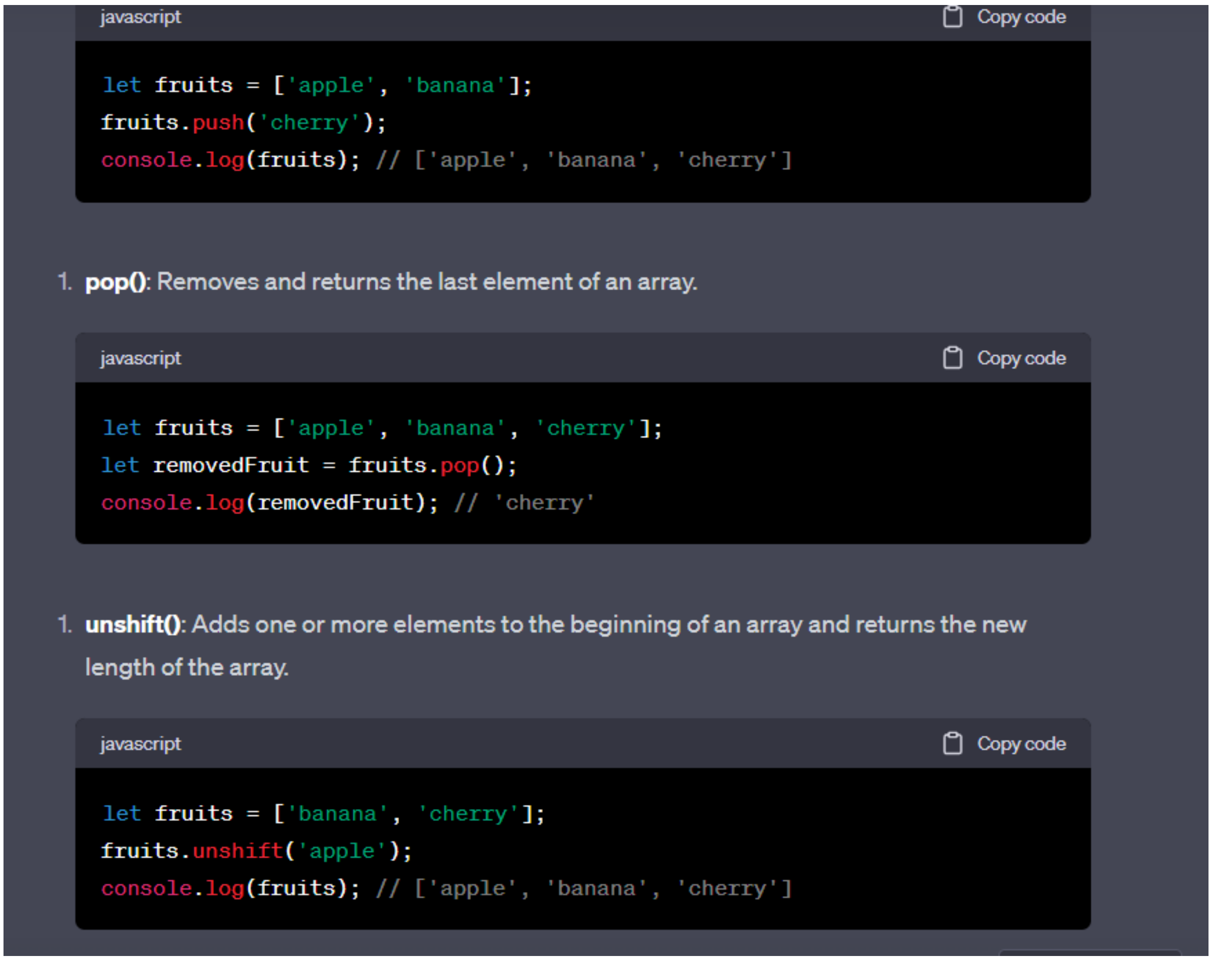
```php
$file = 'yourfile.txt';
$contentToAppend = 'This is the content to append.';

// Append content to the file
file_put_contents($file, $contentToAppend, FILE_APPEND);
```



```php
try {
    // Code that may throw an exception
    $result = 10 / 0; // This will throw a DivisionByZeroError
} catch (DivisionByZeroError $e) {
    // Code to handle the exception
    echo "Division by zero error: " . $e->getMessage();
} catch (Exception $e) {
    // Catch more general exceptions (if needed)
    echo "An error occurred: " . $e->getMessage();
} finally {
    // Optional: Code that will always run, regardless of whether an excepti
    echo "This code always runs.";
}
```



```javascript
let display = document.getElementById('display');

let buttons = Array.from(document.getElementsByClassName('button'));

buttons.map( button => {
    button.addEventListener('click', (e) => {
        switch(e.target.innerText){
            case 'C':
                display.innerText = '';
                break;
            case '←':
                if(display.innerText){
                    display.innerText = display.innerText.slice(0, -1);
                }
                break;
            case '=':
                display.innerText = eval(display.innerText);
                break;
            default:
                display.innerText += e.target.innerText;
        }
    });
});
```



javascript Copy code

```javascript
let fruits = ['apple', 'banana'];
fruits.push('cherry');
console.log(fruits); // ['apple', 'banana', 'cherry']
```

1. **pop()**: Removes and returns the last element of an array.

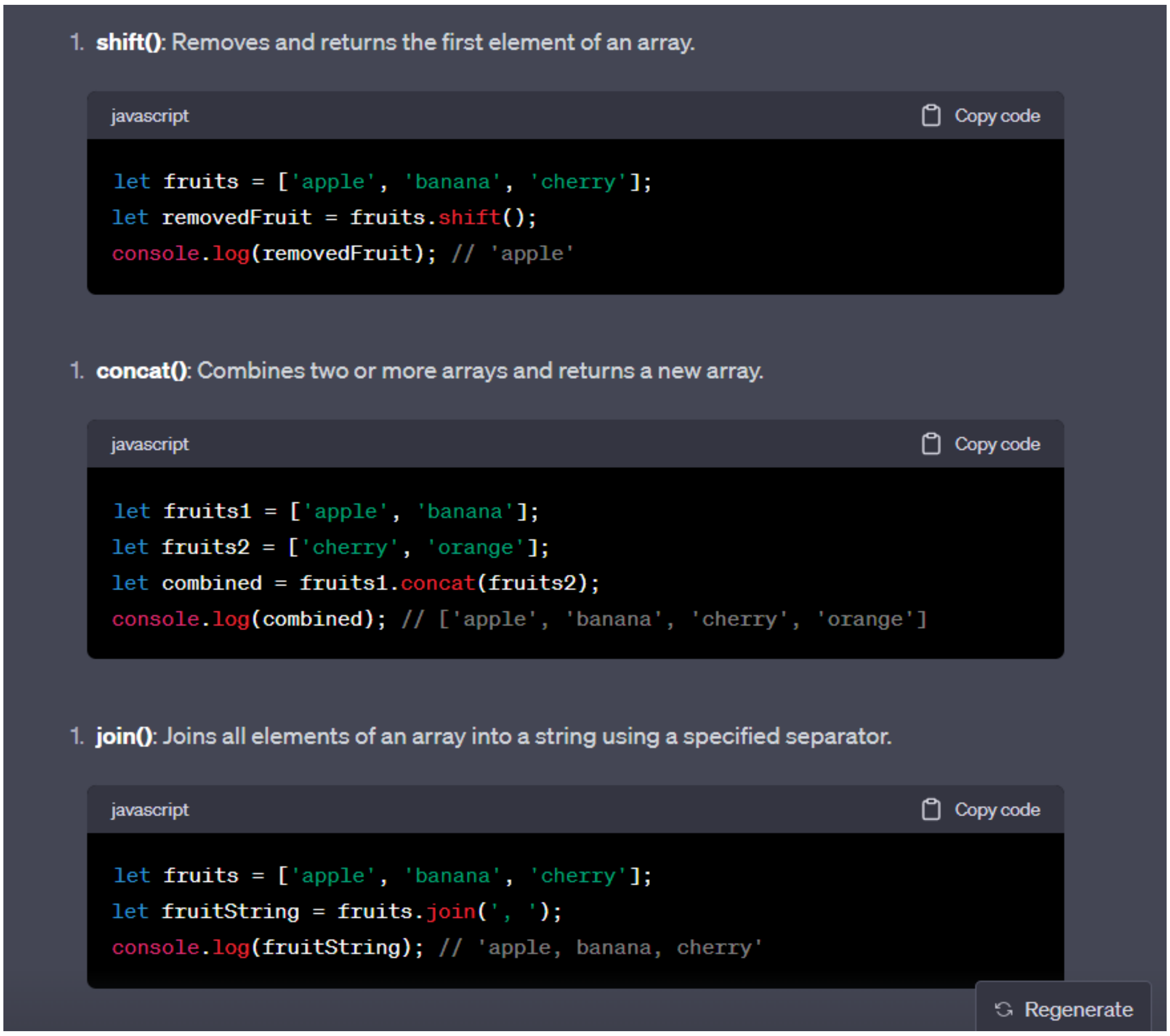
javascript Copy code

```javascript
let fruits = ['apple', 'banana', 'cherry'];
let removedFruit = fruits.pop();
console.log(removedFruit); // 'cherry'
```

1. **unshift()**: Adds one or more elements to the beginning of an array and returns the new length of the array.

javascript Copy code

```javascript
let fruits = ['banana', 'cherry'];
fruits.unshift('apple');
console.log(fruits); // ['apple', 'banana', 'cherry']
```


1. **shift()**: Removes and returns the first element of an array.

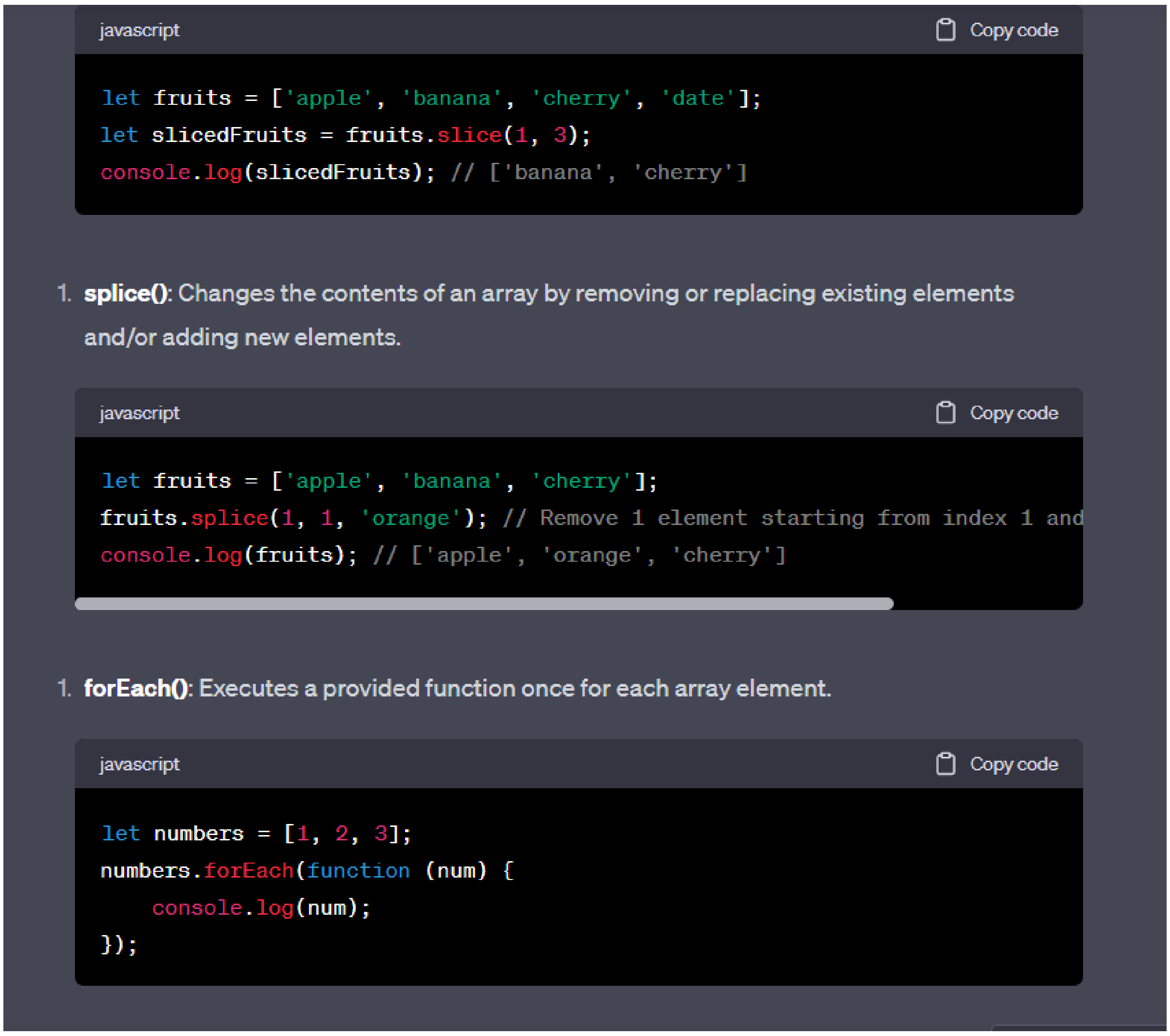
```javascript
let fruits = ['apple', 'banana', 'cherry'];
let removedFruit = fruits.shift();
console.log(removedFruit); // 'apple'
```

1. **concat()**: Combines two or more arrays and returns a new array.

```javascript
let fruits1 = ['apple', 'banana'];
let fruits2 = ['cherry', 'orange'];
let combined = fruits1.concat(fruits2);
console.log(combined); // ['apple', 'banana', 'cherry', 'orange']
```

1. **join()**: Joins all elements of an array into a string using a specified separator.

```javascript
let fruits = ['apple', 'banana', 'cherry'];
let fruitString = fruits.join(', ');
console.log(fruitString); // 'apple, banana, cherry'
```



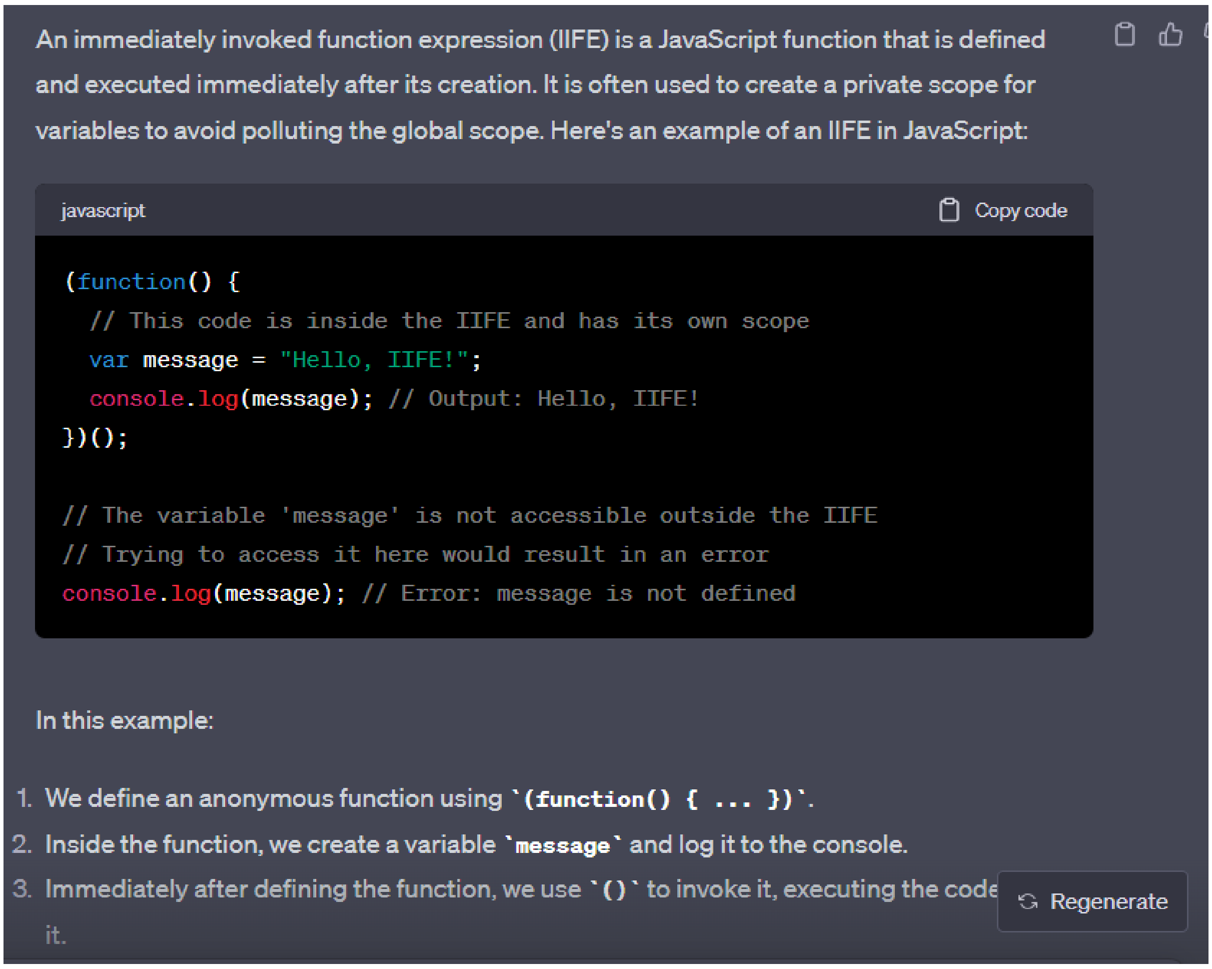
```javascript
let fruits = ['apple', 'banana', 'cherry', 'date'];
let slicedFruits = fruits.slice(1, 3);
console.log(slicedFruits); // ['banana', 'cherry']
```

1. **splice()**: Changes the contents of an array by removing or replacing existing elements and/or adding new elements.

```javascript
let fruits = ['apple', 'banana', 'cherry'];
fruits.splice(1, 1, 'orange'); // Remove 1 element starting from index 1 and
console.log(fruits); // ['apple', 'orange', 'cherry']
```

1. **forEach()**: Executes a provided function once for each array element.

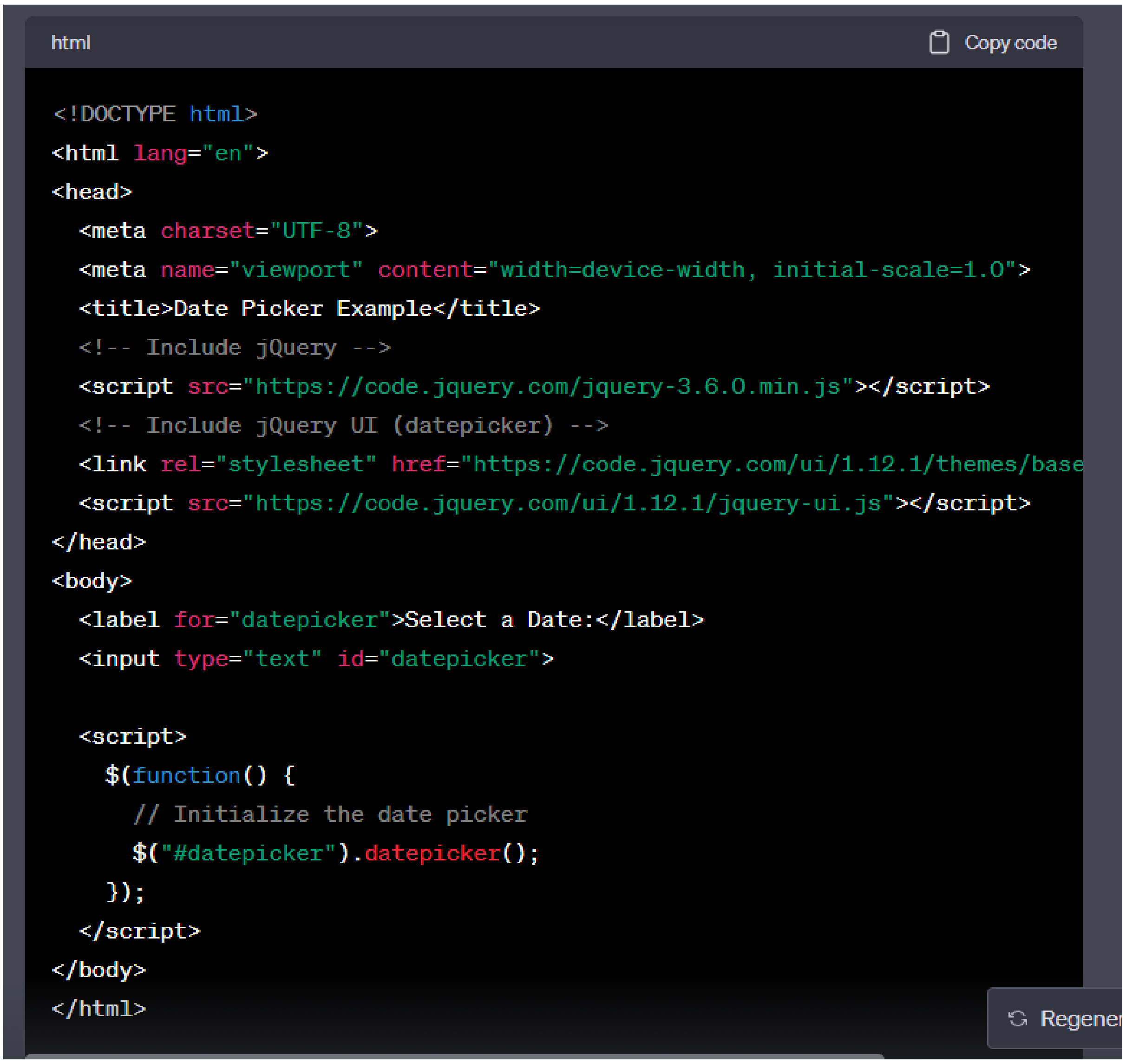
```javascript
let numbers = [1, 2, 3];
numbers.forEach(function (num) {
    console.log(num);
});
```


An immediately invoked function expression (IIFE) is a JavaScript function that is defined and executed immediately after its creation. It is often used to create a private scope for variables to avoid polluting the global scope. Here's an example of an IIFE in JavaScript:

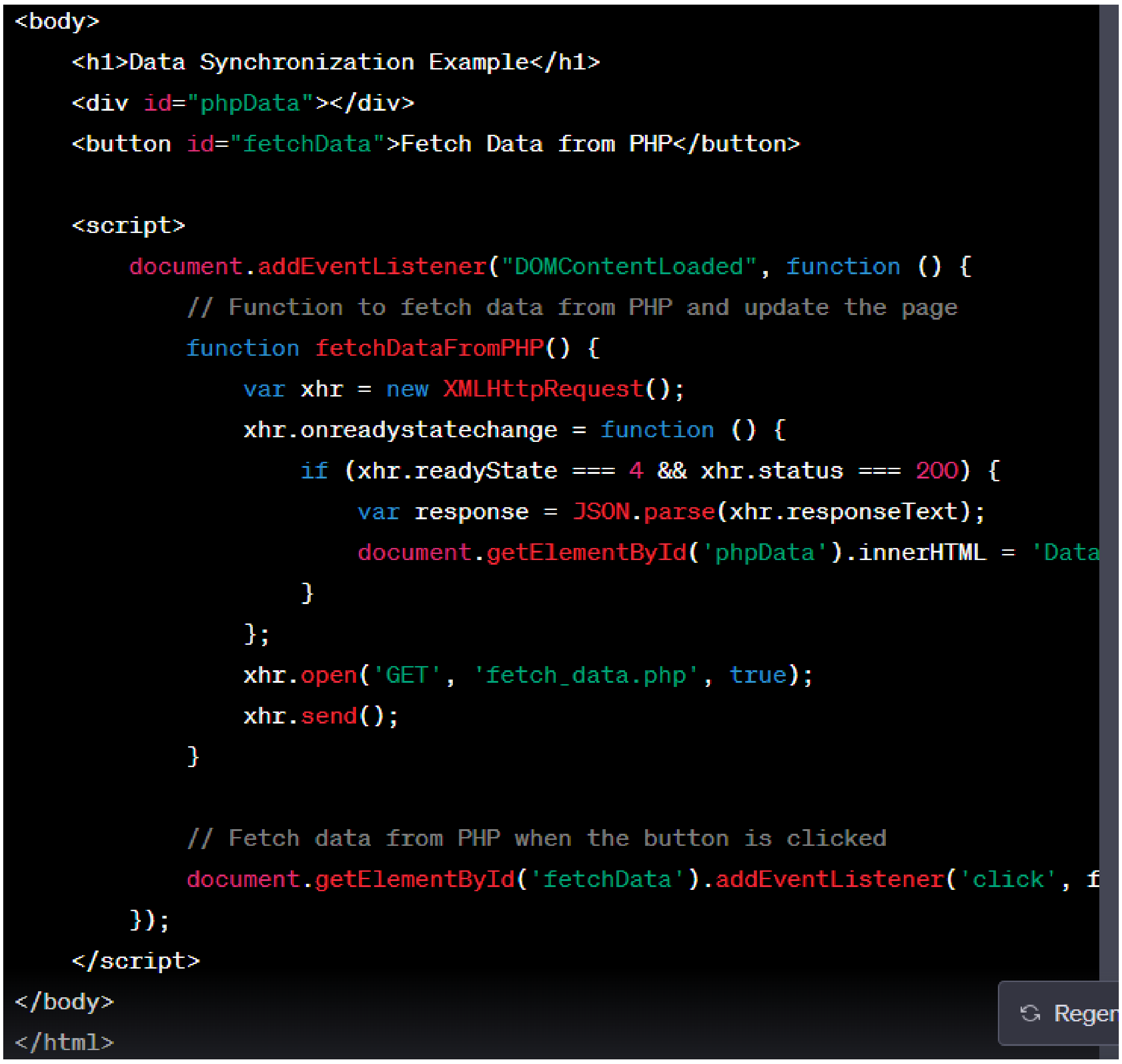
```javascript
(function() {
  // This code is inside the IIFE and has its own scope
  var message = "Hello, IIFE!";
  console.log(message); // Output: Hello, IIFE!
})();

// The variable 'message' is not accessible outside the IIFE
// Trying to access it here would result in an error
console.log(message); // Error: message is not defined
```

In this example:

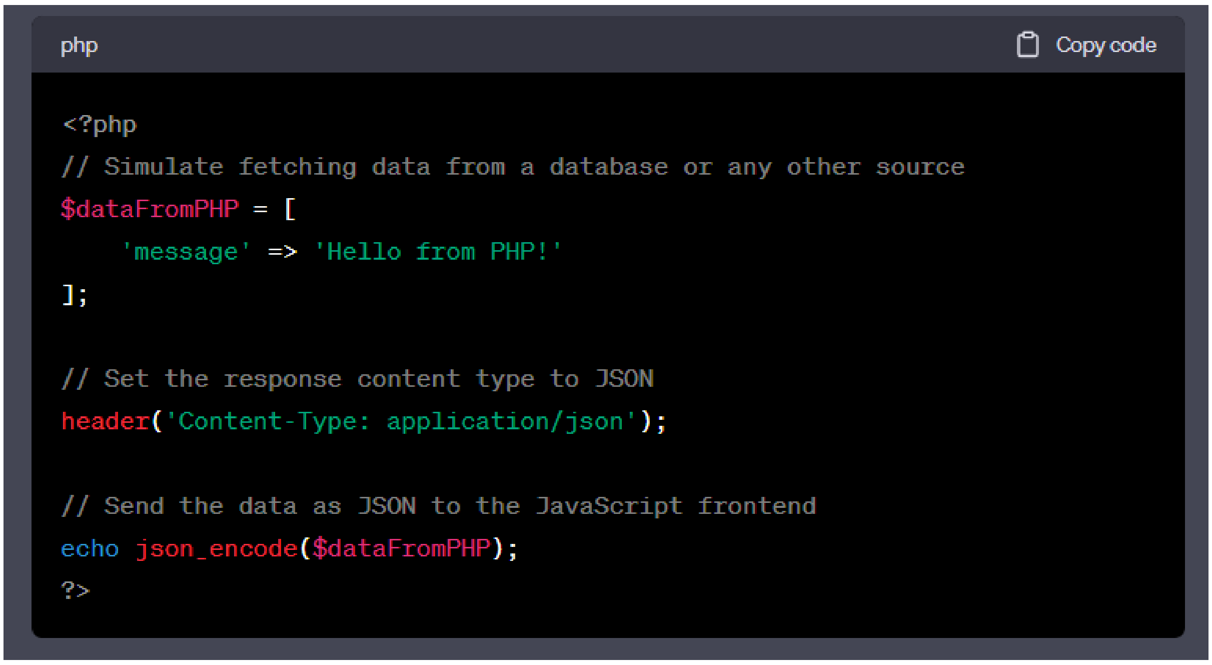
1. We define an anonymous function using `(function() { ... })`.
2. Inside the function, we create a variable `message` and log it to the console.
3. Immediately after defining the function, we use `()` to invoke it, executing the code it.



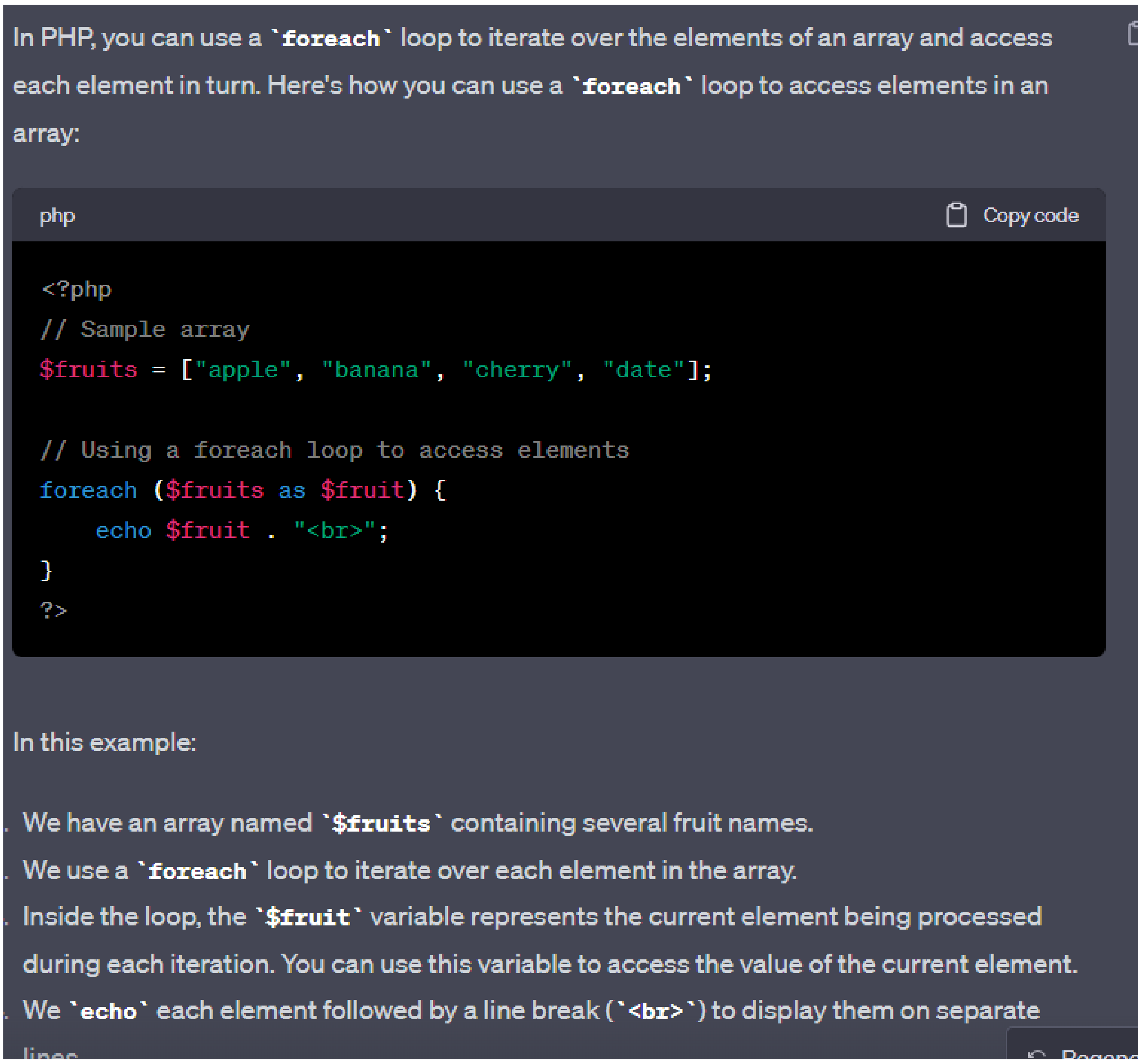
```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Date Picker Example</title>
  <!-- Include jQuery -->
  <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
  <!-- Include jQuery UI (datepicker) -->
  <link rel="stylesheet" href="https://code.jquery.com/ui/1.12.1/themes/base
  <script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>
</head>
<body>
  <label for="datepicker">Select a Date:</label>
  <input type="text" id="datepicker">

  <script>
    $(function() {
      // Initialize the date picker
      $("#datepicker").datepicker();
    });
  </script>
</body>
</html>
```



```html
<body>
    <h1>Data Synchronization Example</h1>
    <div id="phpData"></div>
    <button id="fetchData">Fetch Data from PHP</button>

    <script>
        document.addEventListener("DOMContentLoaded", function () {
            // Function to fetch data from PHP and update the page
            function fetchDataFromPHP() {
                var xhr = new XMLHttpRequest();
                xhr.onreadystatechange = function () {
                    if (xhr.readyState === 4 && xhr.status === 200) {
                        var response = JSON.parse(xhr.responseText);
                        document.getElementById('phpData').innerHTML = 'Data
                    }
                };
                xhr.open('GET', 'fetch_data.php', true);
                xhr.send();
            }

            // Fetch data from PHP when the button is clicked
            document.getElementById('fetchData').addEventListener('click', f
        });
    </script>
</body>
</html>
```



```php
<?php
// Simulate fetching data from a database or any other source
$dataFromPHP = [
    'message' => 'Hello from PHP!'
];

// Set the response content type to JSON
header('Content-Type: application/json');

// Send the data as JSON to the JavaScript frontend
echo json_encode($dataFromPHP);
?>
```

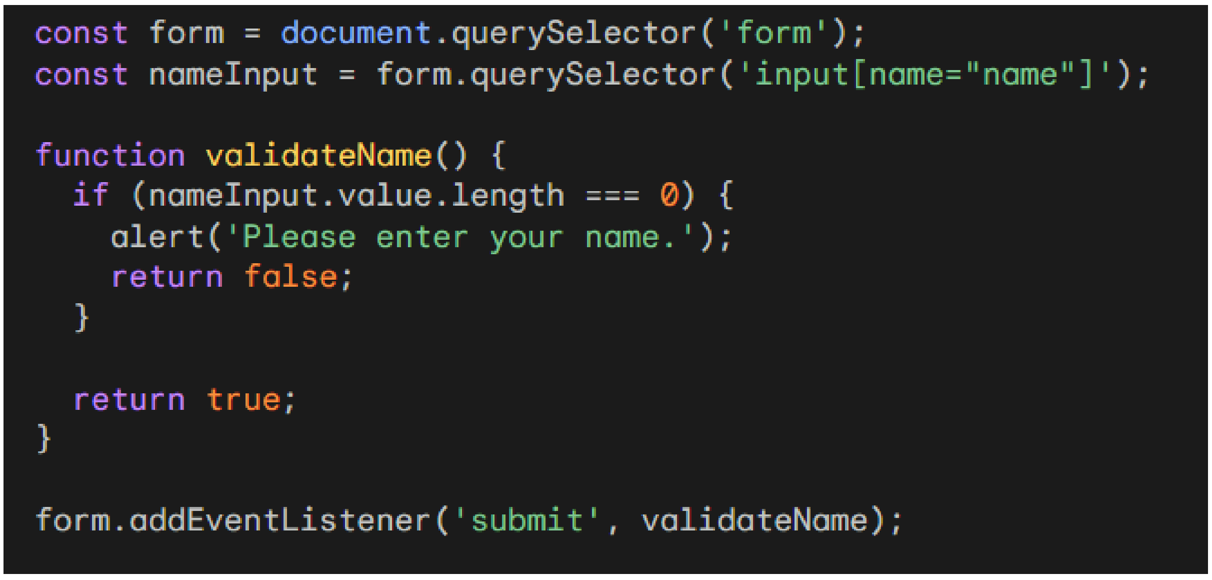

In PHP, you can use a `foreach` loop to iterate over the elements of an array and access each element in turn. Here's how you can use a `foreach` loop to access elements in an array:

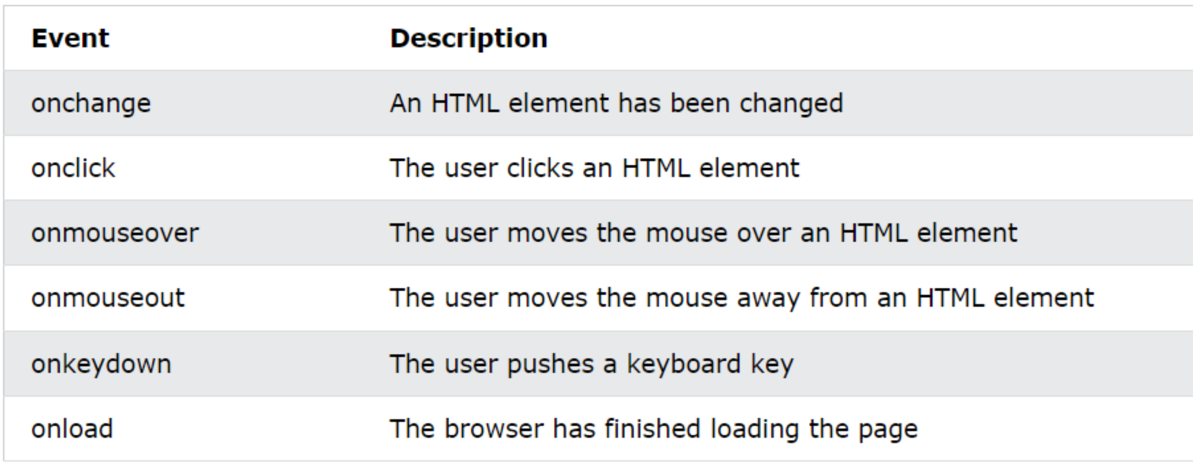
php Copy code

```php
<?php
// Sample array
$fruits = ["apple", "banana", "cherry", "date"];

// Using a foreach loop to access elements
foreach ($fruits as $fruit) {
    echo $fruit . "<br>";
}
?>
```

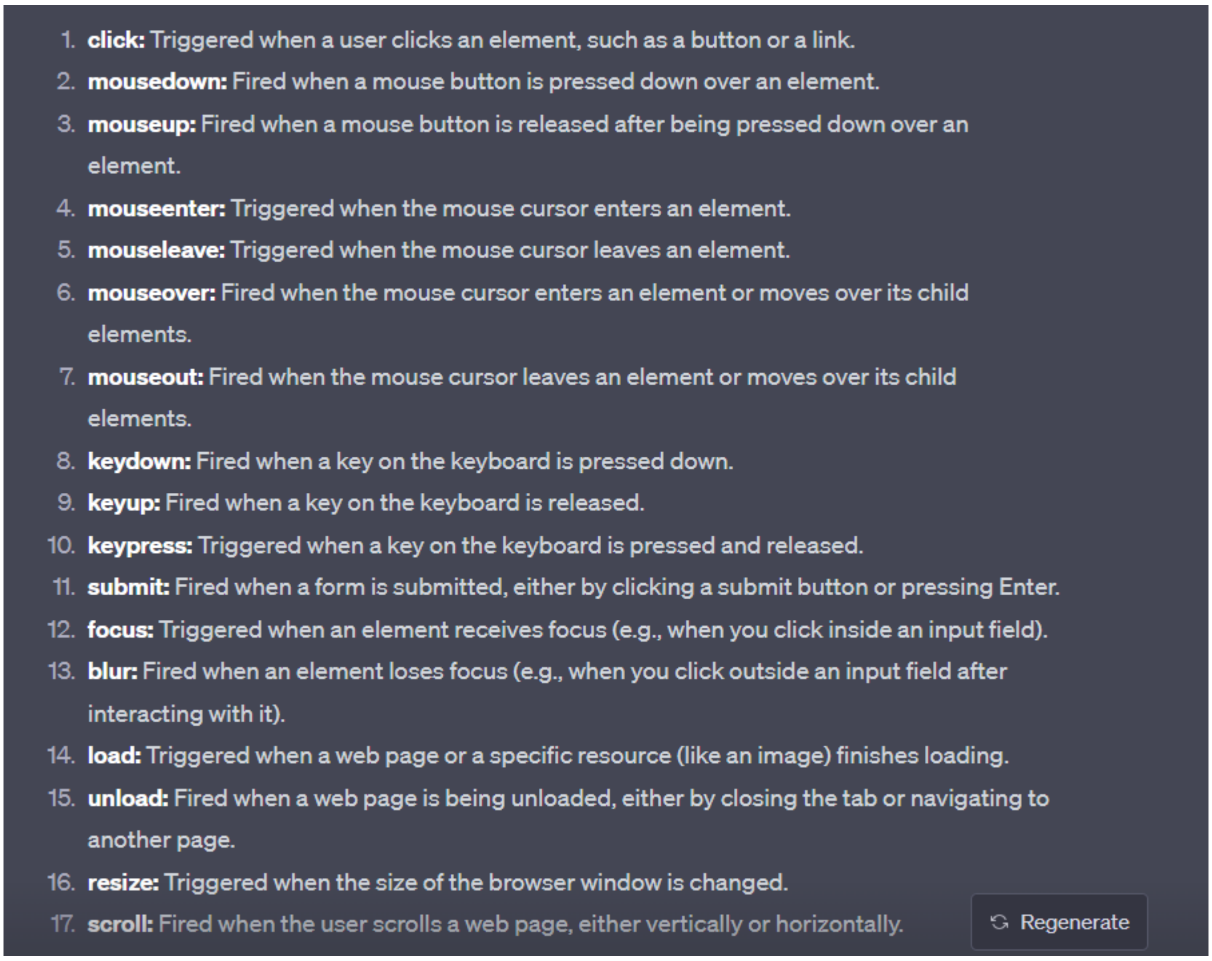
In this example:

. We have an array named `$fruits` containing several fruit names.
. We use a `foreach` loop to iterate over each element in the array.
. Inside the loop, the `$fruit` variable represents the current element being processed during each iteration. You can use this variable to access the value of the current element.
. We `echo` each element followed by a line break (`<br>`) to display them on separate lines

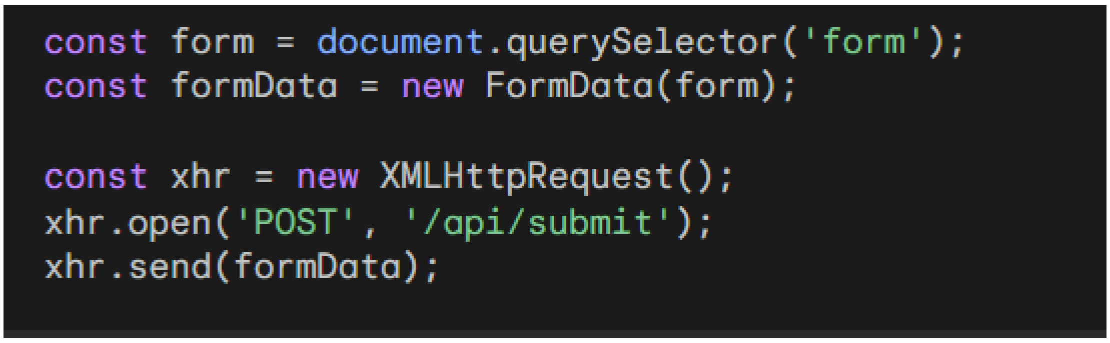

```javascript
const form = document.querySelector('form');
const nameInput = form.querySelector('input[name="name"]');

function validateName() {
  if (nameInput.value.length === 0) {
    alert('Please enter your name.');
    return false;
  }

  return true;
}

form.addEventListener('submit', validateName);
```



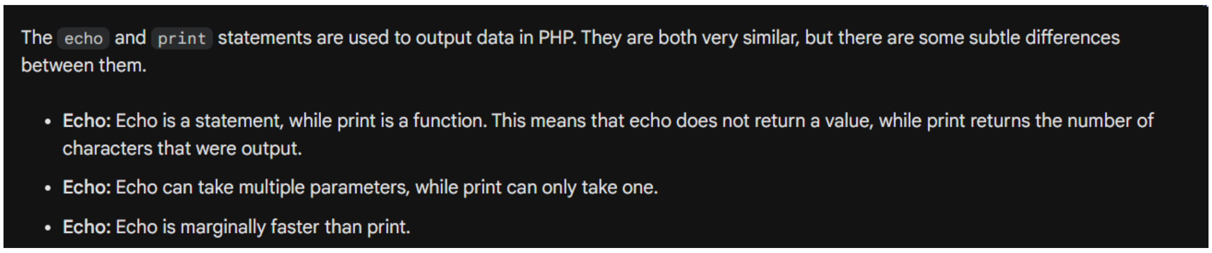
Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page



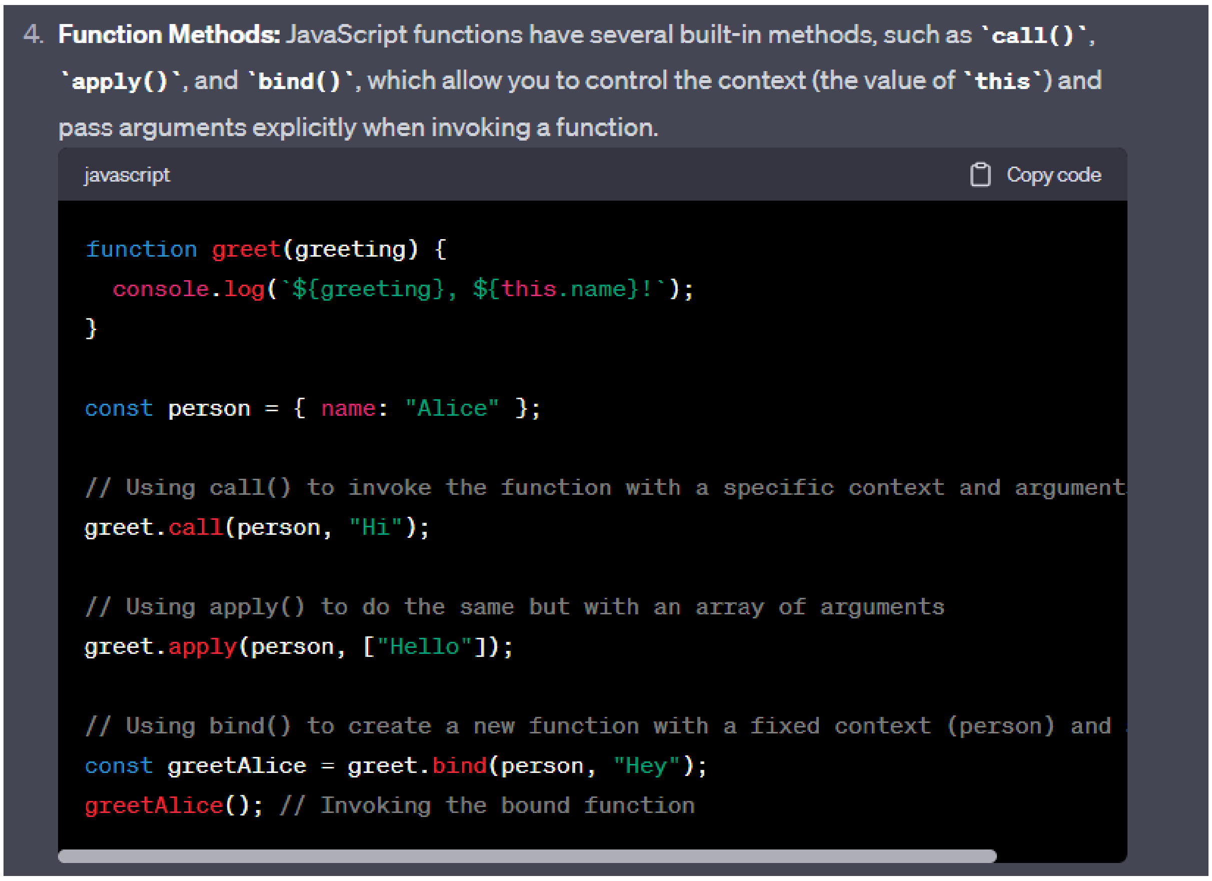
1. **click:** Triggered when a user clicks an element, such as a button or a link.
2. **mousedown:** Fired when a mouse button is pressed down over an element.
3. **mouseup:** Fired when a mouse button is released after being pressed down over an element.
4. **mouseenter:** Triggered when the mouse cursor enters an element.
5. **mouseleave:** Triggered when the mouse cursor leaves an element.
6. **mouseover:** Fired when the mouse cursor enters an element or moves over its child elements.
7. **mouseout:** Fired when the mouse cursor leaves an element or moves over its child elements.
8. **keydown:** Fired when a key on the keyboard is pressed down.
9. **keyup:** Fired when a key on the keyboard is released.
10. **keypress:** Triggered when a key on the keyboard is pressed and released.
11. **submit:** Fired when a form is submitted, either by clicking a submit button or pressing Enter.
12. **focus:** Triggered when an element receives focus (e.g., when you click inside an input field).
13. **blur:** Fired when an element loses focus (e.g., when you click outside an input field after interacting with it).
14. **load:** Triggered when a web page or a specific resource (like an image) finishes loading.
15. **unload:** Fired when a web page is being unloaded, either by closing the tab or navigating to another page.
16. **resize:** Triggered when the size of the browser window is changed.
17. **scroll:** Fired when the user scrolls a web page, either vertically or horizontally.



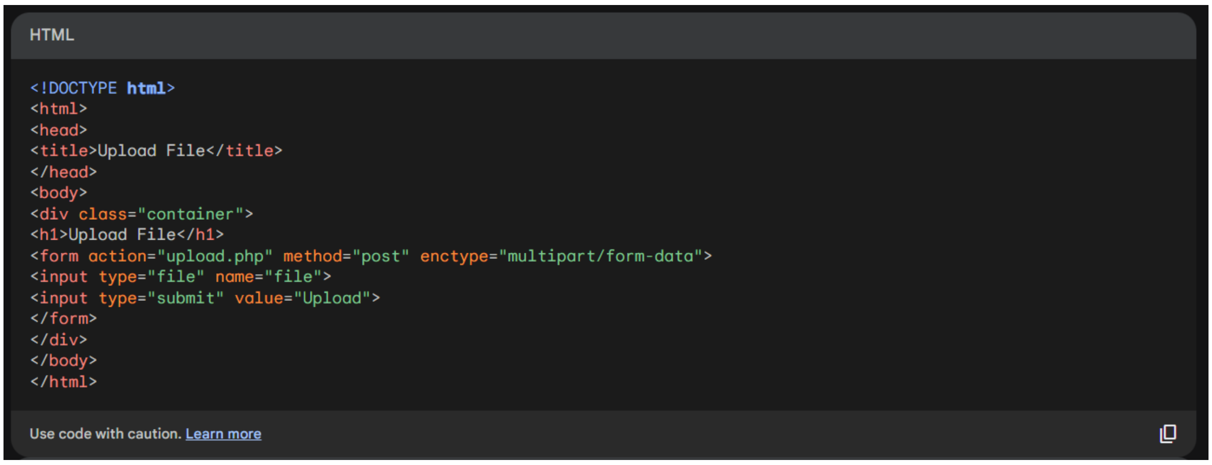
```
const form = document.querySelector('form');
const formData = new FormData(form);

const xhr = new XMLHttpRequest();
xhr.open('POST', '/api/submit');
xhr.send(formData);
```


The `echo` and `print` statements are used to output data in PHP. They are both very similar, but there are some subtle differences between them.

- **Echo:** Echo is a statement, while print is a function. This means that echo does not return a value, while print returns the number of characters that were output.
- **Echo:** Echo can take multiple parameters, while print can only take one.
- **Echo:** Echo is marginally faster than print.

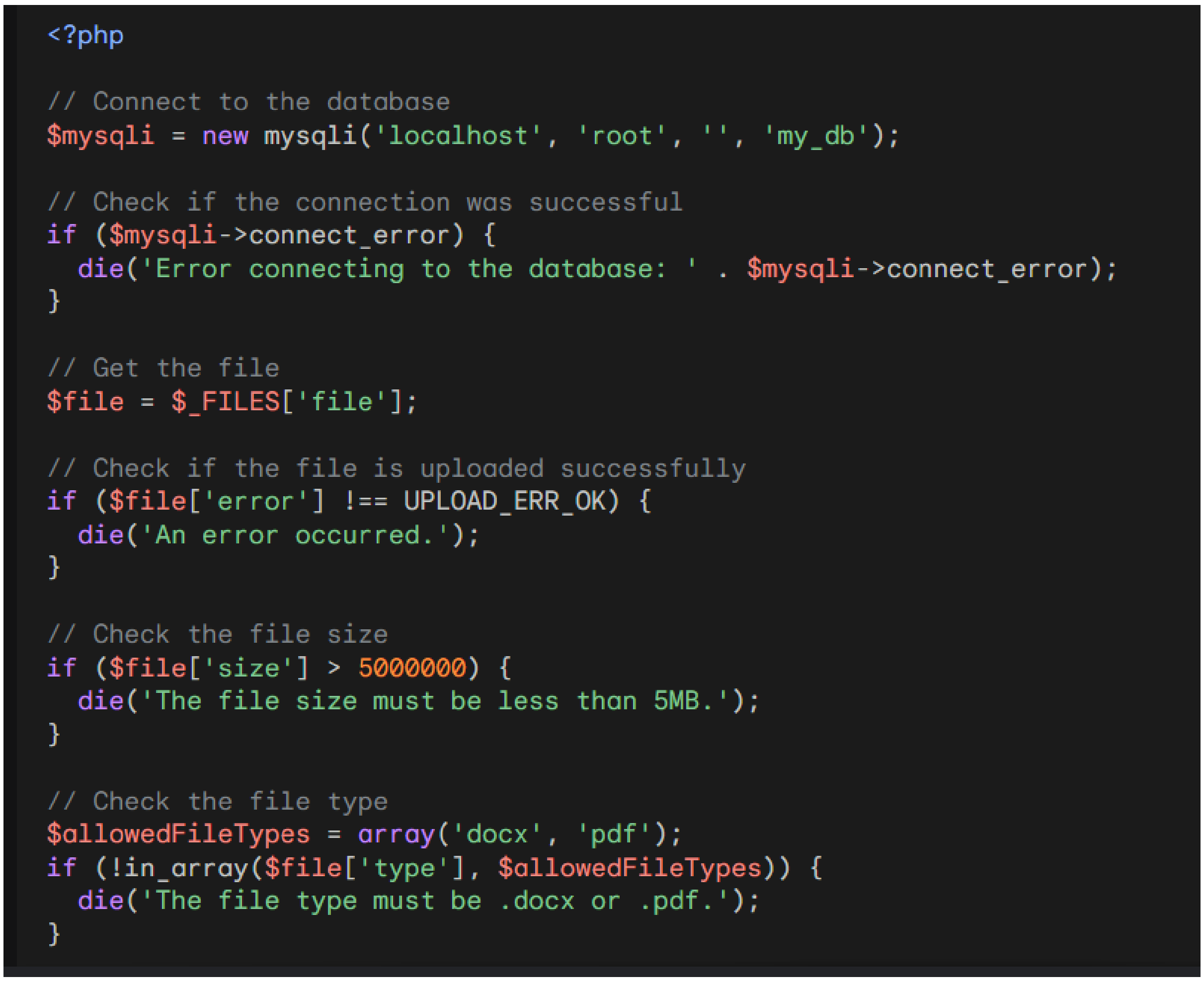

4. **Function Methods:** JavaScript functions have several built-in methods, such as **`` `call()` ``**, **`` `apply()` ``**, and **`` `bind()` ``**, which allow you to control the context (the value of **`` `this` ``**) and pass arguments explicitly when invoking a function.

```javascript
function greet(greeting) {
  console.log(`${greeting}, ${this.name}!`);
}

const person = { name: "Alice" };

// Using call() to invoke the function with a specific context and argument
greet.call(person, "Hi");

// Using apply() to do the same but with an array of arguments
greet.apply(person, ["Hello"]);

// Using bind() to create a new function with a fixed context (person) and
const greetAlice = greet.bind(person, "Hey");
greetAlice(); // Invoking the bound function
```

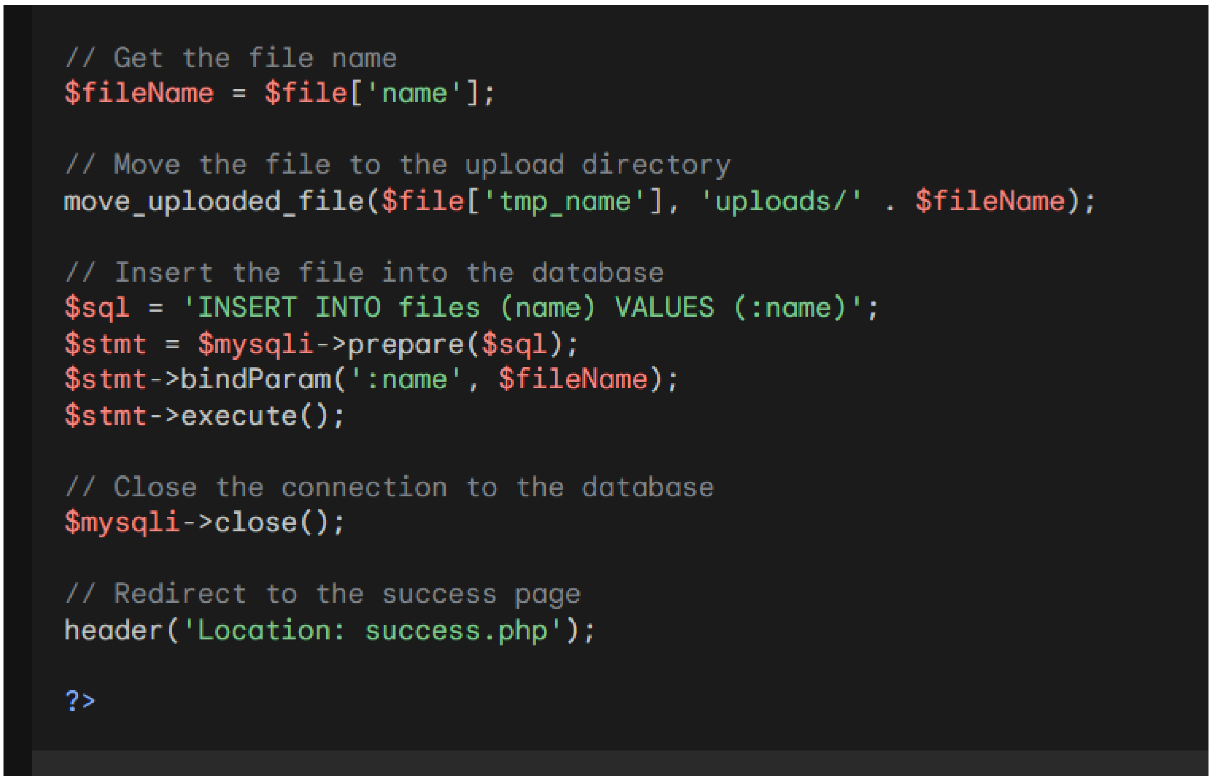


```html
<!DOCTYPE html>
<html>
<head>
<title>Upload File</title>
</head>
<body>
<div class="container">
<h1>Upload File</h1>
<form action="upload.php" method="post" enctype="multipart/form-data">
<input type="file" name="file">
<input type="submit" value="Upload">
</form>
</div>
</body>
</html>
```

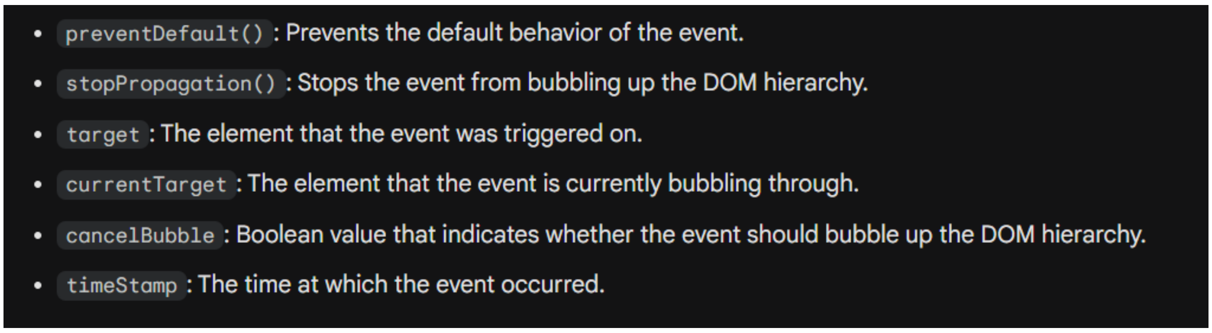
Use code with caution. Learn more



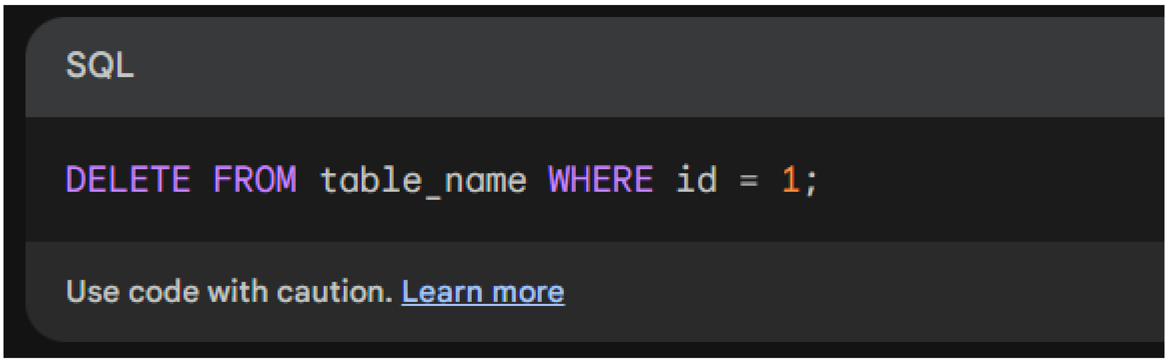
```php
<?php

// Connect to the database
$mysqli = new mysqli('localhost', 'root', '', 'my_db');

// Check if the connection was successful
if ($mysqli->connect_error) {
  die('Error connecting to the database: ' . $mysqli->connect_error);
}

// Get the file
$file = $_FILES['file'];

// Check if the file is uploaded successfully
if ($file['error'] !== UPLOAD_ERR_OK) {
  die('An error occurred.');
}

// Check the file size
if ($file['size'] > 5000000) {
  die('The file size must be less than 5MB.');
}

// Check the file type
$allowedFileTypes = array('docx', 'pdf');
if (!in_array($file['type'], $allowedFileTypes)) {
  die('The file type must be .docx or .pdf.');
}
```



```php
// Get the file name
$fileName = $file['name'];

// Move the file to the upload directory
move_uploaded_file($file['tmp_name'], 'uploads/' . $fileName);

// Insert the file into the database
$sql = 'INSERT INTO files (name) VALUES (:name)';
$stmt = $mysqli->prepare($sql);
$stmt->bindParam(':name', $fileName);
$stmt->execute();

// Close the connection to the database
$mysqli->close();

// Redirect to the success page
header('Location: success.php');

?>
```



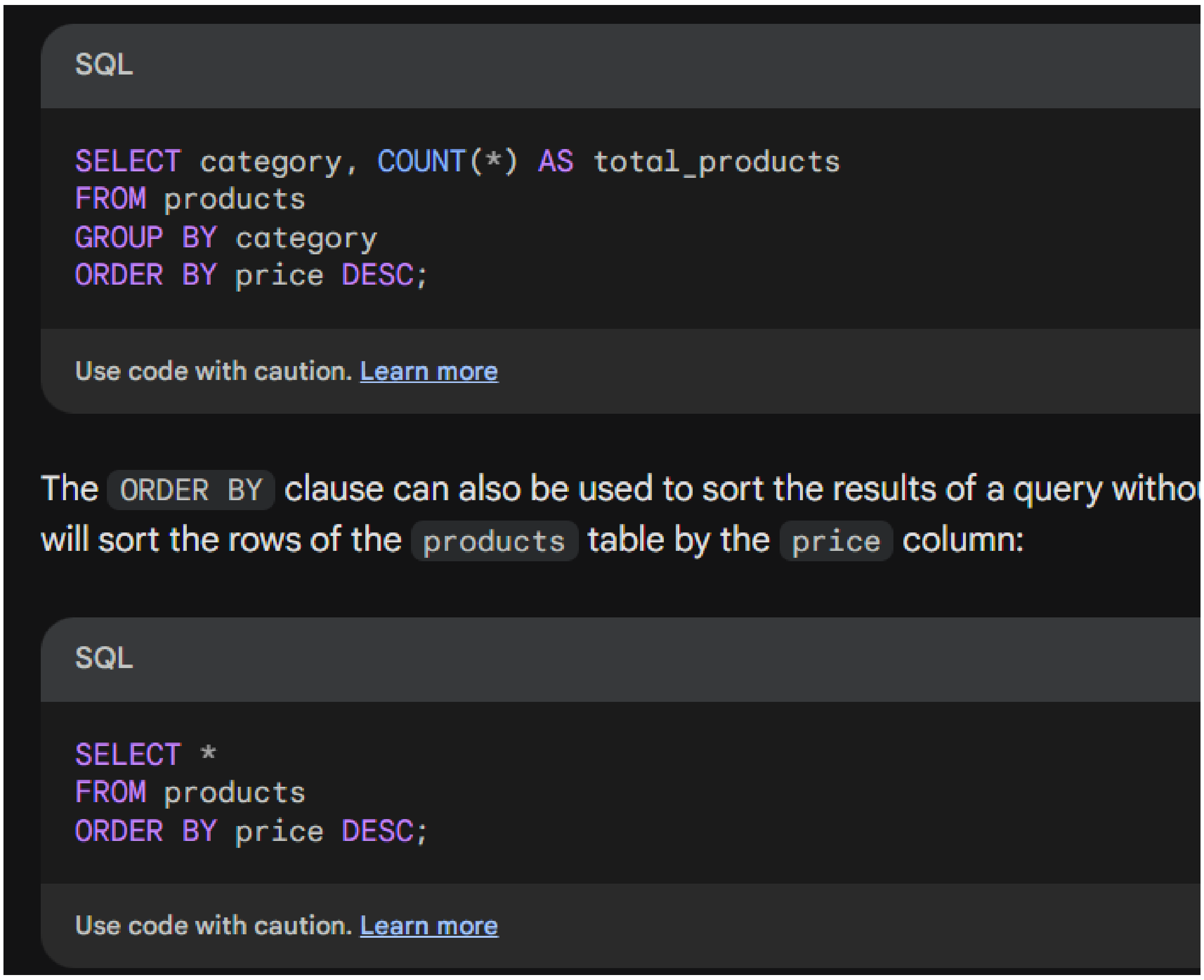
- `preventDefault()` : Prevents the default behavior of the event.
- `stopPropagation()` : Stops the event from bubbling up the DOM hierarchy.
- `target` : The element that the event was triggered on.
- `currentTarget` : The element that the event is currently bubbling through.
- `cancelBubble` : Boolean value that indicates whether the event should bubble up the DOM hierarchy.
- `timeStamp` : The time at which the event occurred.



SQL

```sql
DELETE FROM table_name WHERE id = 1;
```

Use code with caution. Learn more



SQL

```sql
SELECT category, COUNT(*) AS total_products
FROM products
GROUP BY category
ORDER BY price DESC;
```

Use code with caution. Learn more

The `ORDER BY` clause can also be used to sort the results of a query witho
will sort the rows of the `products` table by the `price` column:

SQL

```sql
SELECT *
FROM products
ORDER BY price DESC;
```

Use code with caution. Learn more