
# HACKSERIES 01

# JAVASCRIPT

### What is JavaScript?

- JavaScript is a programming language that is commonly used in web development.

- It is a client-side language, which means that it is executed by the user's web browser rather than the web server. This allows JavaScript to interact with the user and create dynamic, interactive web pages.

- JavaScript is often used in combination with HTML and CSS to create web pages that are more interactive and engaging. It can be used to create all sorts of effects, such as drop-down menus, image sliders, and pop-up windows.

- JS **Frontend Frameworks:** React, Angular, Vue.
- JS **Backend Frameworks**: Express, Node.

## What are Variables?

- In JavaScript, variables are used to store data. They are an essential part of any programming language, as they allow you to store, retrieve, and manipulate data in your programs.
- There are a few different ways to declare variables in JavaScript, each with its own syntax and rules. In this blog post, we'll take a look at the different types of variables available in JavaScript and how to use them.
- **Declaring Variables**
- To declare a variable in JavaScript, you use the "var" keyword followed by the name of the variable.

**Data Types**

In JavaScript, there are several data types that you can use to store different types of data. Some common data types include:

- Numbers (e.g. 10, 3.14)
- Strings (e.g. "hello", 'world')
- Booleans (e.g. true, false)
- Arrays (e.g. [1, 2, 3])
- Objects (e.g. { name: "John", age: 30 })

**Primitives and Objects**

In JavaScript, there are two main types of data: primitives and objects.

**Primitives**

Primitives are the simplest and most basic data types in JavaScript. They include:

- Numbers (e.g. 10, 3.14)
- Strings (e.g. "hello", 'world')
- Booleans (e.g. true, false)
- Null (a special value that represents an absence of value)
- Undefined (a special value that represents an uninitialized variable)

Primitives are immutable, which means that once they are created, they cannot be changed. For example:

let x = 10;

x = 20; // x is now 20

In this example, the value of "x" is changed from 10 to 20. However, this does not change the value of the primitive itself, but rather creates a new primitive with the value of 20.

**Objects**

Objects are more complex data types in JavaScript and are used to represent real-world objects or abstract concepts. They are composed of key-value pairs, where the keys are strings and the values can be any data type (including primitives and other objects).

Objects are mutable, which means that they can be changed after they are created. For example:

let obj = { name: "John", age: 30 };

obj.age = 31; // the age property of obj is now 31

In this example, the "age" property of the "obj" object is changed from 30 to 31. This changes the value of the object itself, rather than creating a new object.

There are several other data types in JavaScript that are classified as objects, including arrays, functions, and dates. These data types behave similarly to objects in that they are mutable and can be modified after they are created.

**Operators and Expressions**

Operators in JavaScript are symbols that perform specific operations on one or more operands (values or variables). For example, the addition operator (+) adds two operands together and the assignment operator (=) assigns a value to a variable.

There are several types of operators in JavaScript, including:

- Arithmetic operators (e.g. +, -, *, /, %)
- Comparison operators (e.g. >, <, >=, <=, ==, !=)
- Logical operators (e.g. &&, ||, !)
- Assignment operators (e.g. =, +=, -=, *=, /=)
- Conditional (ternary) operator (e.g. ?:)
- # If else conditionals

- The "if" statement in JavaScript is used to execute a block of code if a certain condition is met. The "else" clause is used to execute a block of code if the condition is not met.
- Here is the basic syntax for an "if" statement:

```
if (condition) {
  // code to be executed if condition is true
}
```

- Here is the syntax for an "if" statement with an "else" clause:

```
if (condition) {
// code to be executed if condition is true
} else {
// code to be executed if condition is false
}
```

- The condition is a boolean expression that evaluates to either true or false. If the condition is true, the code in the "if" block is executed. If the condition is false, the code in the "else" block is executed (if present).
- For example:

```
let x = 10;
if (x > 5) {
  console.log("x is greater than 5");
} else {
  console.log("x is not greater than 5");
}
```

- In this example, the condition "x > 5" is true, so the code in the "if" block is executed and the message "x is greater than 5" is printed to the console.

# If else

- The "if-else ladder" is a control structure in JavaScript that allows you to execute a different block of code depending on multiple conditions. It is called a ladder because it consists of multiple "if" and "else" statements arranged in a ladder-like fashion.
- Here is the syntax for an "if-else ladder":

```
if (condition1) {
  // code to be executed if condition1 is true
} else if (condition2) {
  // code to be executed if condition1 is false and condition2 is true
} else if (condition3) {
  // code to be executed if condition1 and condition2 are false and condition3 is true
} ...
else {
  // code to be executed if all conditions are false
```

```
}
```

- In this structure, each "if" statement is followed by an optional "else" statement. If the first "if" condition is true, the code in the corresponding block is executed and the rest of the ladder is skipped. If the first "if" condition is false, the second "if" condition is evaluated, and so on. If none of the conditions are true, the code in the "else" block is executed.
- For example:

```
let x = 10;
if (x > 15) {
  console.log("x is greater than 15");
} else if (x > 10) {
  console.log("x is greater than 10 but less than or equal to 15");
} else if (x > 5) {
  console.log("x is greater than 5 but less than or equal to 10");
} else {
  console.log("x is less than or equal to 5");
}
```

- In this example, the first "if" condition "x > 15" is false, so the second "if" condition "x > 10" is evaluated. This condition is also false, so the third "if" condition "x > 5" is evaluated. This condition is true, so the code in the corresponding block is executed and the message "x is greater than 5 but less than or equal to 10" is printed to the console.
- The "if-else ladder" is a useful control structure for executing different blocks of code based on multiple conditions. It can help you write more concise and maintainable code in JavaScript.
- # Switch case

- The "switch" statement in JavaScript is another control structure that allows you to execute a different block of code depending on a specific value. It is often used as an alternative to the "if-else ladder" when you have multiple conditions to check against a single value.
- Here is the syntax for a "switch" statement:

```
switch (expression) {
  case value1:
    // code to be executed if expression == value1
    break;
  case value2:
    // code to be executed if expression == value2
    break;
  ...
  default:
    // code to be executed if expression does not match any of the values
}
```

- In this structure, the "expression" is evaluated and compared to each of the "case" values. If the "expression" matches a "case" value, the corresponding block of code is executed. The "break" statement is used to exit the "switch" statement and prevent the code in the following cases from

being executed. The "default" case is optional and is executed if the "expression" does not match any of the "case" values.

- For example:

```
let x = "apple";
switch (x) {
  case "apple":
    console.log("x is an apple");
    break;
  case "banana":
    console.log("x is a banana");
    break;
  case "orange":
    console.log("x is an orange");
    break;
  default:
    console.log("x is something else");
}
```

- In this example, the "expression" is the variable "x," which has the value "apple." The "expression" is compared to each of the "case" values, and when it matches the value "apple," the corresponding block of code is executed and the message "x is an apple" is printed to the console.
- The "switch" statement is a useful control structure for executing different blocks of code based on a specific value. It can help you write more concise and maintainable code in JavaScript.

# Ternary Operator

- The ternary operator is a shorthand way to write an if-else statement in JavaScript. It takes the form of condition ? value1 : value2, where condition is a boolean expression, and value1 and value2 are expressions of any type. If condition is true, the ternary operator returns value1; if condition is false, it returns value2.
- Here's an example of how you can use the ternary operator to assign a value to a variable based on a condition:

```
let x = 10;
let y = 20;
let max;

max = (x > y) ? x : y;

console.log(max); // Outputs: 20
```

- In this example, the ternary operator checks whether x is greater than y. If it is, max is assigned the value of x; otherwise, it is assigned the value of y.

- The ternary operator can be a useful and concise way to write simple `if-else` statements, but it can become difficult to read and understand when used for more complex statements or nested inside other expressions. In these cases, it may be better to use a regular `if-else` statement instead.

# For Loops

- For loops are a common control flow structure in programming that allows you to repeat a block of code a specific number of times. In JavaScript, there are three types of for loops: the standard for loop, the for-in loop, and the for-of loop.
- **Standard for loop**
- The standard for loop has the following syntax:

```
for (initialization; condition; increment/decrement) {
  // code to be executed
}
```

- The `initialization` the statement is executed before the loop starts and is typically used to initialize a counter variable. The `condition` is checked at the beginning of each iteration and if it is `true`, the loop continues. If it is `false`, the loop exits. The `increment/decrement` statement is executed at the end of each iteration and is used to update the counter variable.
- Here's an example of a standard for loop that counts from 1 to 10:

```
for (let i = 1; i <= 10; i++) {
  console.log(i);
}
```

- This loop will print the numbers 1 through 10 to the console.
- **For-in loop**
- The for-in loop is used to iterate over the properties of an object. It has the following syntax:

```
for (variable in object) {
  // code to be executed
}
```

- The `variable` is assigned the name of each property in the object as the loop iterates over them.
- Here's an example of a for-in loop that iterates over the properties of an object:

```
let person = {
  name: "John",
  age: 30,
  job: "developer"
};

for (let key in person) {
  console.log(key + ": " + person[key]);
}
```

- This loop will print the following to the console:

```
name: John
age: 30
job: developer
```

- **For-of loop**
- The for-of loop is used to iterate over the values of an iterable object, such as an array or a string. It has the following syntax:

```
for (variable of object) {
  // code to be executed
}
```

- The `variable` is assigned the value of each element in the object as the loop iterates over them.
- Here's an example of a for-of loop that iterates over the elements of an array:

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

for (let number of numbers) {
  console.log(number);
}
```

- This loop will print the numbers 1 through 5 to the console.
- For loops are a powerful tool in JavaScript and can be used to perform a variety of tasks, such as iterating over arrays and objects, repeating a block of code a specific number of times, and more. With the three types of for loops available in JavaScript, you can choose the one that best fits your needs and use it to write more efficient and effective code.

# While Loop

- While loops are a control flow structure in programming that allow you to repeat a block of code while a certain condition is true. In JavaScript, the syntax for a while loop is:

```
while (condition) {
  // code to be executed
}
```

- The `condition` is checked at the beginning of each iteration and if it is `true`, the code block is executed. If it is `false`, the loop exits.
- Here's an example of a while loop that counts from 1 to 10:

```
let i = 1;

while (i <= 10) {
  console.log(i);
  i++;
}
```

- This loop will print the numbers 1 through 10 to the console.
- It's important to include a way to update the `condition` within the loop, otherwise it will become an infinite loop and will run forever. In the example above, the `i++` statement increments the value

of i by 1 at the end of each iteration, which eventually causes the condition to be false and the loop to exit.

# Functions

- JavaScript functions are blocks of code that can be defined and executed whenever needed. They are a crucial part of JavaScript programming and are used to perform specific tasks or actions.
- Functions are often referred to as "first-class citizens" in JavaScript because they can be treated like any other value, such as a number or a string. This means that they can be assigned to variables, passed as arguments to other functions, and returned as values from functions.
- Here's the basic syntax for defining a function in JavaScript:

```
function functionName(parameters) {
  // code to be executed
}
```

- The functionName is a unique identifier for the function, and the parameters are the variables that are passed to the function when it is called. These parameters act as placeholders for the actual values that are passed to the function when it is executed.
- Here's an example of a simple function that takes a single parameter and returns the square of that number:

```
function square(x) {
  return x * x;
}
```

To call this function, you would simply use the function name followed by the arguments in parentheses:

```
let result = square(5);  // returns 25
```

Functions can also have multiple parameters, like this:

```
function add(x, y) {
  return x + y;
}
```

- In this case, the add function takes two parameters, x and y, and returns their sum.
- JavaScript also has a special type of function called an "arrow function," which uses a shorter syntax. Here's the same square a function defined using an arrow function:

```
const square = (x) => {
  return x * x;
};
```

# Strings

- One of the most important aspects of JavaScript is its ability to manipulate strings, which are sequences of characters. In this blog post, we will explore the basics of JavaScript strings and the various string methods that can be used to manipulate them.

- A string in JavaScript is a sequence of characters enclosed in either single or double quotes. For example, the following are valid strings in JavaScript:

```
"Hello World"
'Hello World'
```

- JavaScript provides a number of built-in methods for manipulating strings. Some of the most commonly used string methods are:
-
- **length** - This method returns the number of characters in a string. For example, the following code will return 11:

```
var str = "Hello World";
```

- **concat** - This method is used to concatenate (combine) two or more strings. For example, the following code will return "Hello World":

```
var str1 = "Hello";
var str2 = " World";
console.log(str1.concat(str2));
```

- **indexOf** - This method is used to find the index of a specific character or substring in a string. For example, the following code will return 6:

```
var str = "Hello World";
console.log(str.indexOf("W"));
```

- **slice** - This method is used to extract a portion of a string. For example, the following code will return "World":

```
var str = "Hello World";
console.log(str.slice(6));
```

- **replace** - This method is used to replace a specific character or substring in a string. For example, the following code will return "Hello Universe":

```
var str = "Hello World";
console.log(str.replace("World", "Universe"));
```

- **toUpperCase** and **toLowerCase** - These methods are used to convert a string to uppercase or lowercase letters. For example, the following code will return "HELLO WORLD" and "hello world" respectively:

```
var str = "Hello World";
console.log(str.toUpperCase());
console.log(str.toLowerCase())
```

# Arrays and Array Methods

One of the most important data structures in JavaScript is the array, which is a collection of elements. In this blog post, we will explore the basics of JavaScript arrays and the various array methods that can be used to manipulate them.

An array in JavaScript is a collection of elements enclosed in square brackets. Elements can be of any data type, including numbers, strings, and other arrays. For example, the following is a valid array in JavaScript:

```
var myArray = [1, "Hello", [2, 3]];
```

JavaScript provides a number of built-in methods for manipulating arrays. Some of the most commonly used array methods are:

**length** - This method returns the number of elements in an array. For example, the following code will return 3:

```
var myArray = [1, "Hello", [2, 3]];
console.log(myArray.length);
```

**push** - This method is used to add an element to the end of an array. For example, the following code will add the element "World" to the end of the array:

```
var myArray = [1, "Hello", [2, 3]];
myArray.push("World");
console.log(myArray); // [1, "Hello", [2, 3], "World"]
```

**pop** - This method is used to remove the last element of an array. For example, the following code will remove the last element ("World") from the array:

```
var myArray = [1, "Hello", [2, 3], "World"];
myArray.pop();
console.log(myArray); // [1, "Hello", [2, 3]]
```

**shift** - This method is used to remove the first element of an array. For example, the following code will remove the first element (1) from the array:

```
var myArray = [1, "Hello", [2, 3]];
myArray.shift();
console.log(myArray); // ["Hello", [2, 3]]
```

**unshift** - This method is used to add an element to the beginning of an array. For example, the following code will add the element 0 to the beginning of the array:

```
var myArray = [1, "Hello", [2, 3]];
myArray.unshift(0);
console.log(myArray); // [0, 1, "Hello", [2, 3]]
```

- **slice** - This method is used to extract a portion of an array. For example, the following code will extract the elements from index 1 to 2 (exclusive):

```
var myArray = [1, "Hello", [2, 3]];
console.log(myArray.slice(1, 2)); // ["Hello"]
```

- **splice** - This method is used to add or remove elements from an array. For example, the following code will remove the element at index 1 and add the elements "Hello World" and [4, 5] at index 1:

```
var myArray = [1, "Hello", [2, 3]];
myArray.splice(1, 1, "Hello World", [4, 5]);
console.
```

# Window Object

- The JavaScript window object represents the current browser window or tab that is open in a web browser. It is a global object that provides access to various properties and methods related to the browser window.
- One of the most commonly used properties of the window object is the document property, which represents the current web page. This property can be used to access the HTML elements on a page, as well as manipulate them. For example, the following code can be used to change the text of a specific element on a page:

```
document.getElementById("myElement").innerHTML = "This is my new text";
```

- Another important property of the window object is the location property, which provides information about the current URL. This property can be used to redirect the user to a different page, as well as retrieve the current URL. For example, the following code can be used to redirect the user to a different page:

```
window.location.href = "https://www.example.com";
```

- The window object also provides several methods for displaying dialog boxes, such as alert(), confirm(), and prompt(). These methods can be used to display messages to the user and receive input from the user. For example, the following code can be used to display an alert box with a message:

```
alert("This is an alert message");
```

- There are many other properties and methods provided by the window object, such as setTimeout() and setInterval() for scheduling tasks, open() and close() for opening and closing new windows, and scrollTo() for scrolling the window to a specific position.
- It's important to note that window object is also the parent object of the document object and history object.
- In conclusion, the JavaScript window object provides a powerful set of tools for interacting with the browser window and web page. It can be used to access and manipulate HTML elements, redirect the user to a different page, and display dialog boxes. Understanding how to use the window object is essential for creating interactive and dynamic web pages.

# Document Object

- The JavaScript `document` object is a part of the `window` object and represents the current web page. It allows developers to access and manipulate the elements of the web page, as well as the Document Object Model (DOM) tree that represents the structure of the web page.
- One of the most commonly used methods of the `document` object is the `getElementById()` method. This method allows developers to access a specific element on the web page by its unique ID. For example, the following code can be used to change the text of an element with the ID "myElement":

```
document.getElementById("myElement").innerHTML = "This is my new text";
```

- Another important method is `getElementsByClassName()` method which returns a live HTMLCollection of elements with the given class name. It can be used to access multiple elements with the same class name.

```
let elements = document.getElementsByClassName("myClass");
for (let i = 0; i < elements.length; i++) {
    elements[i].innerHTML = "This is my new text";
}
```

- The `document` object also provides several methods for creating new elements and adding them to the web page, such as `createElement()` and `createTextNode()`. For example, the following code can be used to create a new `div` element and add it to the web page:

```
let newDiv = document.createElement("div");
document.body.appendChild(newDiv);
```

- It's important to note that the `document` object is a part of the DOM API and it is supported by all modern browsers, but it's important to check for browser compatibility before using it.
- In conclusion, the JavaScript `document` object provides a powerful set of tools for accessing and manipulating the elements of the web page and the Document Object Model (DOM) tree. Understanding how to use the `document` object is essential for creating dynamic and interactive web pages, but it's important to keep in mind its limitations and check browser compatibility before using it.

# getElementbyId

- The `getElementById()` method is a part of the JavaScript `document` object and it allows developers to access a specific element on a web page by its unique ID. This method returns the first element that matches the specified ID, or null if no such element is found.
- For example, consider the following HTML code:

```
<div id="myDiv">This is my div</div>
```

- The following JavaScript code can be used to access the element with the ID "myDiv" and change its text:

```
let myDiv = document.getElementById("myDiv");
myDiv.innerHTML = "This is my new text";
```

- The getElementById() method is a convenient way to access a specific element on a web page, as it saves developers from having to traverse the entire DOM tree to find the element. This method is especially useful when working with large and complex web pages, as it allows developers to quickly and easily access the elements they need.
- It's important to note that the ID of an element must be unique within the web page, as the getElementById() method only returns the first element that matches the specified ID. Also, the getElementById() method is case-sensitive, meaning that "myDiv" and "mydiv" are considered to be different IDs.
- In conclusion, the getElementById() method is a powerful tool for accessing specific elements on a web page by their unique ID. Understanding how to use this method is essential for creating dynamic and interactive web pages, as it allows developers to quickly and easily access the elements they need. However, it's important to keep in mind that the ID of an element must be unique within the web page and the method is case-sensitive.

# getElementsByClassName

- The getElementsByClassName() method is a part of the JavaScript document object and it allows developers to access multiple elements on a web page by their class name. This method returns a live HTMLCollection of elements with the given class name, or an empty HTMLCollection if no such elements are found.
- For example, consider the following HTML code:

```
<div class="myClass">This is my div</div>
<div class="myClass">This is my div</div>
<div class="myClass">This is my div</div>
```

- The following JavaScript code can be used to access the elements with the class name "myClass" and change their text:

```
let elements = document.getElementsByClassName("myClass");
for (let i = 0; i < elements.length; i++) {
    elements[i].innerHTML = "This is my new text";
}
```

- The getElementsByClassName() method is a convenient way to access multiple elements on a web page, as it allows developers to select elements based on their class name rather than their unique ID. This method is especially useful when working with large and complex web pages, as it allows developers to quickly and easily access multiple elements that share the same class name.
- It's important to note that the getElementsByClassName() method returns a live HTMLCollection, which means that the collection is updated automatically as elements are added or removed from the web page. Also, this method is case-sensitive, meaning that "myClass" and "myclass" are considered to be different class names.
- In conclusion, the getElementsByClassName() method is a powerful tool for accessing multiple elements on a web page by their class name. Understanding how to use this method is essential for creating dynamic and interactive web pages, as it allows developers to quickly and easily access multiple elements that share the same class name. However, it's important to keep in mind that the method returns a live HTMLCollection and is case-sensitive.

# getElementsByTagName

- The getElementsByTagName() method is a part of the JavaScript document object and it allows developers to access multiple elements on a web page by their HTML tag name. This method

returns a live HTMLCollection of elements with the given tag name, or an empty HTMLCollection if no such elements are found.

- For example, consider the following HTML code:

```
<p>This is my paragraph</p>
<p>This is my paragraph</p>
<p>This is my paragraph</p>
```

- The following JavaScript code can be used to access the <p> elements on the web page and change their text:

```
let elements = document.getElementsByTagName("p");
for (let i = 0; i < elements.length; i++) {
    elements[i].innerHTML = "This is my new text";
}
```

- The getElementsByTagName() method is a convenient way to access multiple elements on a web page, as it allows developers to select elements based on their HTML tag name. This method is especially useful when working with large and complex web pages, as it allows developers to quickly and easily access multiple elements of the same type.
- It's important to note that the getElementsByTagName() method returns a live HTMLCollection, which means that the collection is updated automatically as elements are added or removed from the web page. Also, this method is case-sensitive, meaning that "p" and "P" are considered to be different tag names.
- In conclusion, the getElementsByTagName() method is a powerful tool for accessing multiple elements on a web page by their HTML tag name. Understanding how to use this method is essential for creating dynamic and interactive web pages, as it allows developers to quickly and easily access multiple elements of the same type. However, it's important to keep in mind that the method returns a live HTMLCollection and is case-sensitive. It's a good practice to use this method in combination with other methods like getElementById and getElementsByClassName to target specific elements on the web page.

# • innerHTML

- The innerHTML property is a part of the JavaScript HTMLElement object and it allows developers to access and manipulate the HTML content of an element. The innerHTML property returns the content between the opening and closing tags of an element, as a string of HTML.
- For example, consider the following HTML code:

```
<div id="myDiv">
    <p>This is my paragraph</p>
    <p>This is my paragraph</p>
</div>
```

- The following JavaScript code can be used to access the content of the  element with the ID "myDiv" and change its text:

```
let myDiv = document.getElementById("myDiv");
```

```
myDiv.innerHTML = "<p>This is my new text</p>";
```

- The innerHTML property is a powerful tool for manipulating the HTML content of an element. Developers can use it to add, remove, or replace elements, as well as change the text and attributes of existing elements.
- It's important to note that the innerHTML property can be used to insert any valid HTML code, including scripts and event handlers. This can be a powerful feature, but it can also pose a security risk if the HTML is not properly sanitized. Also, it's important to note that when you set the innerHTML property, the content is completely replaced, any previous content, event handlers, and other properties are lost.
- In conclusion, the innerHTML property is a powerful tool for manipulating the HTML content of an element in JavaScript. Understanding how to use this property is essential for creating dynamic and interactive web pages, as it allows developers to easily add, remove, or replace elements, as well as change the text and attributes of existing elements. However, it's important to keep in mind that this property can pose a security risk if the HTML is not properly sanitized and also, it completely replaces the content when set.

# outerHTML

- The outerHTML property is a part of the JavaScript HTMLElement object and it allows developers to access and manipulate the entire HTML of an element, including the element's own tags. The outerHTML property returns the entire HTML of an element, as a string of HTML.
- For example, consider the following HTML code:

```
<div id="myDiv">
    <p>This is my paragraph</p>
    <p>This is my paragraph</p>
</div>
```

- The following JavaScript code can be used to access the  element with the ID "myDiv" and change its content:

```
let myDiv = document.getElementById("myDiv");
myDiv.outerHTML = "<div><p>This is my new text</p></div>";
```

- The outerHTML property is a powerful tool for manipulating the entire HTML of an element. Developers can use it to add, remove, or replace elements, as well as change the text and attributes of existing elements. The main difference between outerHTML and innerHTML is that outerHTML allows you to change the entire element, including its own tags, whereas innerHTML only allows you to change the content within the element.
- It's important to note that the outerHTML property can be used to insert any valid HTML code, including scripts and event handlers. This can be a powerful feature, but it can also pose a security risk if the HTML is not properly sanitized. Also, it's important to note that when you set the outerHTML property, the entire element is replaced, any previous content, event handlers, and other properties are lost.
- In conclusion, the outerHTML property is a powerful tool for manipulating the entire HTML of an element in JavaScript. Understanding how to use this property is essential for creating dynamic and interactive web pages, as it allows developers to easily add, remove, or replace elements, as

well as change the text and attributes of existing elements. However, it's important to keep in mind that this property can pose a security risk if the HTML is not properly sanitized and also, it completely replaces the entire element when set.

# YOUTUBE CHANNELS

**APNA COLLEGE**

**CodeWithHarry**

**JavaScript Mastery**

**RoadsideCoder**

**Sheryians Coding School**

**Chai aur Code**

**Hitesh Choudhary**

**Code With Antonio**

**GreatStack**

**freeCodeCamp.org**
**Anuj Bhaiya**

**CodeHelp - by Babbar**

# YOUTUBE VIDEOS

1. https://youtube.com/playlist?list=PLGjplNEQ1it_oTvuLRNqXfz_v_0pq6unW&feature=shared
2. https://youtube.com/playlist?list=PLu71SKxNbfoBuX3f4EOACle2y-tRC5Q37&feature=shared
3. https://youtu.be/VlPiVmYuoqw?feature=shared
4. https://youtu.be/PkZNo7MFNFg?feature=shared
5. https://youtube.com/playlist?list=PLhzIaPMgkbxDK0XplEg2SdbZu-yz3B_T-&feature=shared