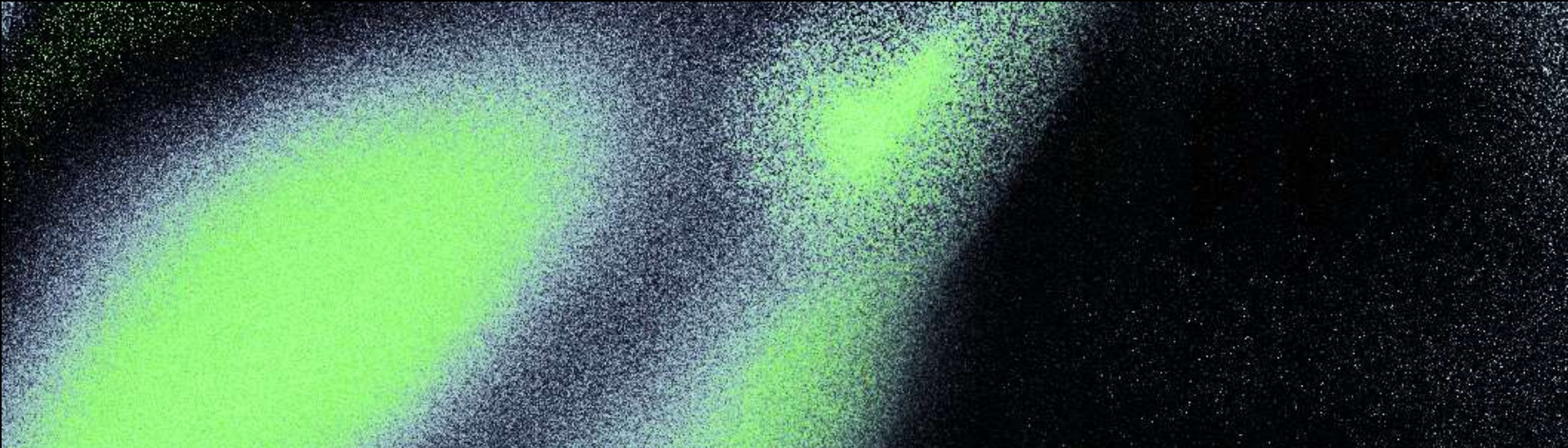
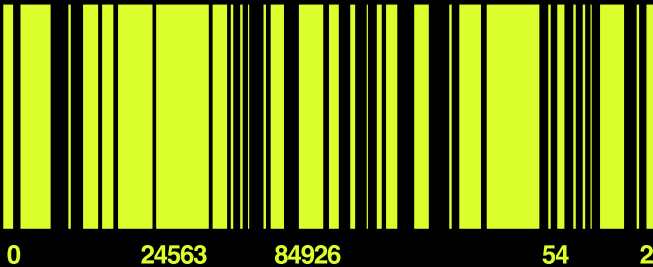
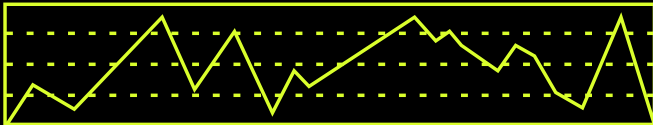
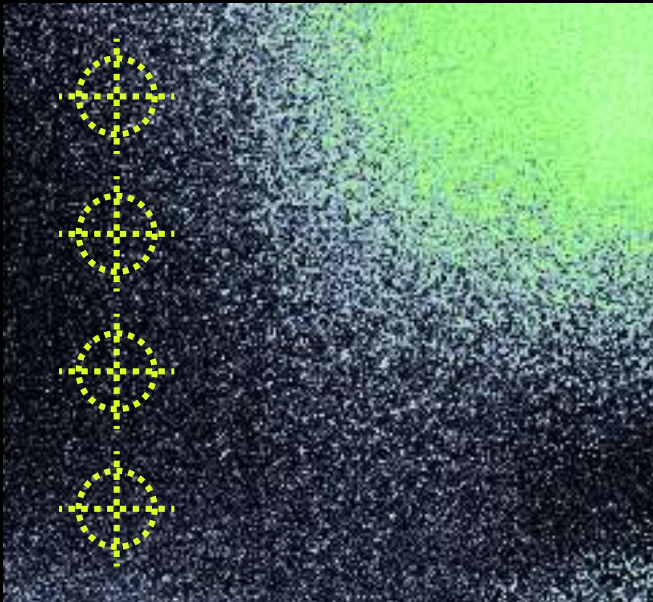
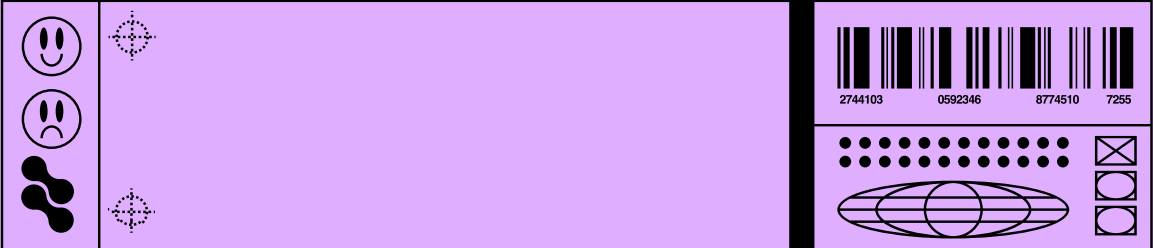


MANIPOLAZIONE DEL DOM



SIMONE DI MEGLIO
FULL STACK WEB DEV

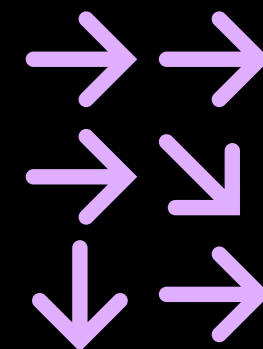
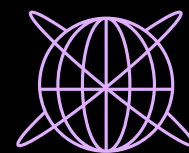


COSA FACCIAMO STASERA PROF?

QUELLO CHE FACCIAMO TUTTE LE SERE,
MIGNOLO.

TENTARE DI CONQUISTARE
IL DOM!

INTRO BOOM



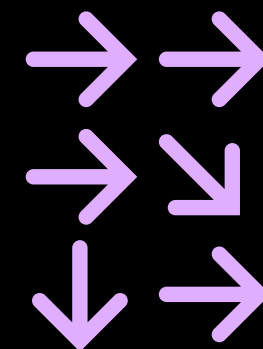
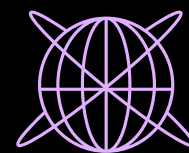


INTRO BOOM

OBIETTIVO DELLA LEZIONE

L'obiettivo principale di questa lezione è fornire una comprensione approfondita della manipolazione del Document Object Model (DOM) in JavaScript.

Impareremo come interagire con gli elementi HTML, modificarli dinamicamente e rispondere agli eventi utente.



COSA È IL DOM

Il **Document Object Model (DOM)** è una rappresentazione gerarchica e strutturata di un documento HTML che consente agli sviluppatori di accedere, manipolare e aggiornare dinamicamente il contenuto di una pagina web utilizzando JavaScript.

Il DOM è essenzialmente una struttura ad albero in cui ogni elemento HTML è rappresentato come un **nodo**, e ogni nodo può avere **figli** (elementi contenuti al suo interno) e **genitori** (elementi che lo contengono). Questa struttura ad albero riflette la struttura del documento HTML, consentendo una manipolazione agevole e dinamica del contenuto della pagina.

DIFFERENZA TRA DOM E CODICE HTML STATICO

È importante comprendere la differenza fondamentale tra il DOM e il codice HTML statico.

Il codice HTML statico rappresenta la struttura iniziale della pagina web come definita nel file sorgente.

D'altra parte, il DOM è la rappresentazione dinamica di questa struttura durante l'esecuzione del codice JavaScript.

HTML STATICO

```
HTML

<!DOCTYPE html>
<html>
<head>
  <title>La mia Pagina</title>
</head>
<body>
  <h1>Ciao Mondo!</h1>
  <p>Benvenuti alla mia pagina web.</p>
</body>
</html>
```

DOM DINAMICO (DOPO L'ESECUZIONE DI JAVASCRIPT):

```
DOM dinamico

- Document
  - html
    - head
      - title
        - "La mia Pagina"
    - body
      - h1
        - "Ciao Mondo!"
      - p
        - "Benvenuti alla mia pagina web."
```

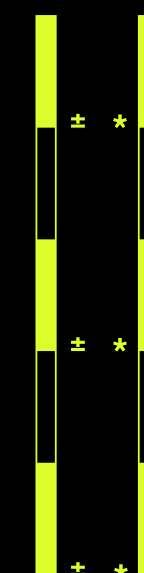
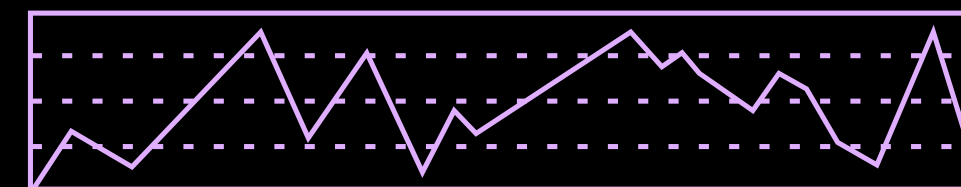
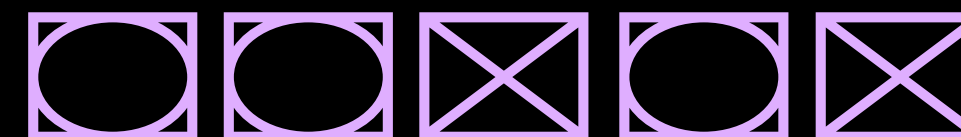
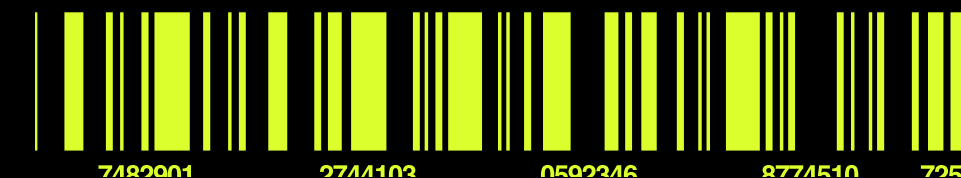
DIFFERENZA TRA DOM E CODICE HTML STATICO

Nel DOM dinamico, possiamo manipolare, aggiungere o rimuovere nodi, cambiare testi e attributi in risposta a interazioni dell'utente o ad eventi specifici.

Questa capacità dinamica è ciò che rende il DOM così potente e versatile nella creazione di esperienze interattive sul web.

Durante questo focus, esploreremo approfonditamente le varie tecniche per accedere, modificare e sfruttare appieno il potenziale del Document Object Model utilizzando JavaScript.

ACCESSO AD ELEMENTI DOM



INTRODUZIONE

Accedere agli elementi del DOM è un passo fondamentale nella manipolazione dinamica delle pagine web.

JavaScript fornisce diversi metodi per recuperare elementi dal DOM, ciascuno adatto a casi specifici.

document.getElementById

```
<!DOCTYPE html>
<html>
<head>
  <title>Accesso agli Elementi DOM</title>
</head>
<body>
  <h1 id="titolo">Titolo della Pagina</h1>
  <p id="paragrafo">Questo è un paragrafo di esempio.</p>

  <script>
    // Accesso all'elemento con ID "titolo"
    var titoloElement = document.getElementById("titolo");
    console.log(titoloElement.textContent);
  </script>
</body>
</html>
```

Il metodo **getElementById** consente di recuperare un elemento tramite il suo ID univoco.

Questo è particolarmente utile quando si lavora con elementi specifici conosciuti.

textContent ? --> vediamo subito!

textContent

```
<!DOCTYPE html>
<html>
<head>
  <title>Accesso agli Elementi DOM</title>
</head>
<body>
  <h1 id="titolo">Titolo della Pagina</h1>
  <p id="paragrafo">Questo è un paragrafo di esempio.</p>

  <script>
    // Accesso all'elemento con ID "titolo"
    var titoloElement = document.getElementById("titolo");
    console.log(titoloElement.textContent);
  </script>
</body>
</html>
```

textContent è una proprietà dell'interfaccia DOM che rappresenta il testo contenuto in un elemento e nei suoi discendenti.

Questa proprietà restituisce il testo di tutti i nodi di testo figli concatenati all'interno dell'elemento.

Se l'elemento non contiene alcun nodo di testo, **textContent** restituirà una stringa vuota.

document.getElementsByClassName

```
<!DOCTYPE html>
<html>
<head>
  <title>Accesso agli Elementi DOM</title>
</head>
<body>
  <p class="paragrafo">Primo paragrafo</p>
  <p class="paragrafo">Secondo paragrafo</p>

  <script>
    // Accesso agli elementi con la classe "paragrafo"
    var paragrafi = document.getElementsByClassName("paragrafo");
    for (var i = 0; i < paragrafi.length; i++) {
      console.log(paragrafi[i].textContent);
    }
  </script>
</body>
</html>
```

Il metodo **getElementsByClassName** restituisce una collezione di elementi che hanno la stessa classe.

Può essere utile quando si desidera lavorare con più elementi dello stesso tipo.

document.getElementsByTagName

```

<!DOCTYPE html>
<html>
<head>
  <title>Accesso agli Elementi DOM</title>
</head>
<body>
  <ul>
    <li>Elemento 1</li>
    <li>Elemento 2</li>
    <li>Elemento 3</li>
  </ul>

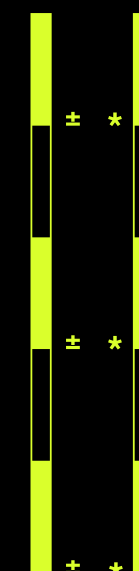
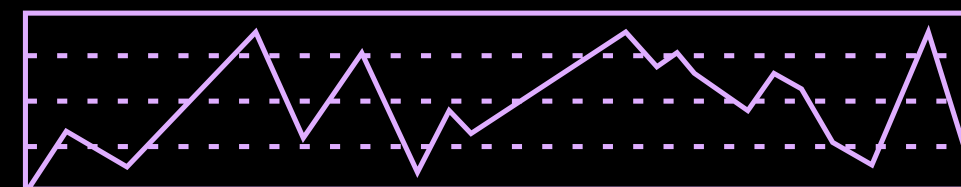
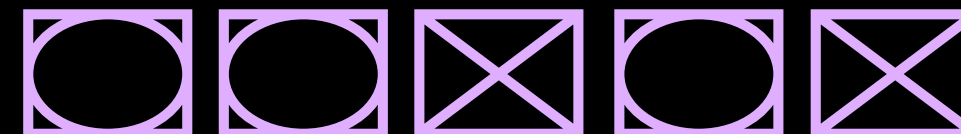
  <script>
    // Accesso a tutti gli elementi "li" nella lista
    var listaElementi = document.getElementsByTagName("li");
    for (var i = 0; i < listaElementi.length; i++) {
      console.log(listaElementi[i].textContent);
    }
  </script>
</body>
</html>

```

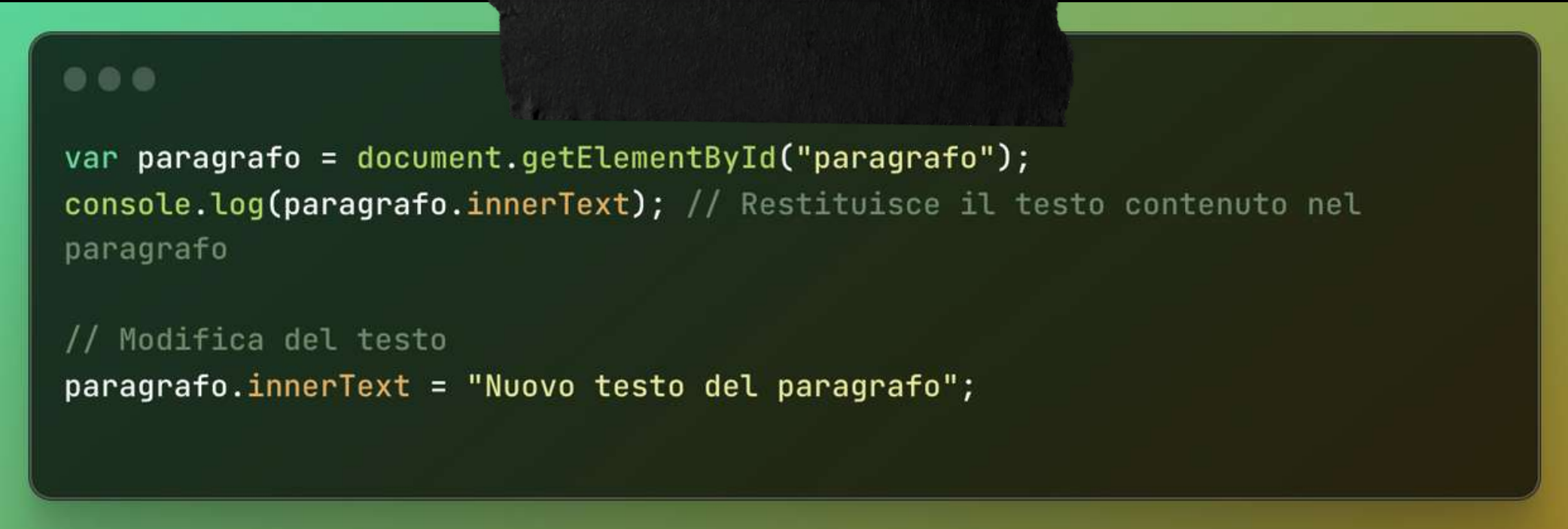
Il metodo **getElementsByTagName** restituisce una collezione di tutti gli elementi con un determinato nome di tag.

Questo è utile quando si desidera lavorare con tutti gli elementi di un certo tipo.

MODIFICA DEL TESTO



innerText



```
var paragrafo = document.getElementById("paragrafo");  
console.log(paragrafo.innerText); // Restituisce il testo contenuto nel  
paragrafo  
  
// Modifica del testo  
paragrafo.innerText = "Nuovo testo del paragrafo";
```

La proprietà **innerText** consente di recuperare o modificare il testo di un elemento

innerHTML



```
var divContenitore = document.getElementById("divContenitore");
console.log(divContenitore.innerHTML); // Restituisce il contenuto HTML del
div

// Modifica del contenuto HTML
divContenitore.innerHTML = "<p>Contenuto aggiornato</p>";
```

La proprietà innerHTML consente di recuperare o modificare il contenuto HTML di un elemento.

Questo significa che può interpretare e manipolare il testo e gli elementi HTML all'interno dell'elemento di destinazione.

La differenza tra i due

la differenza principale tra `innerText` e `innerHTML` riguarda la gestione del contenuto all'interno di un elemento HTML.

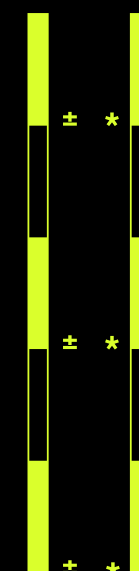
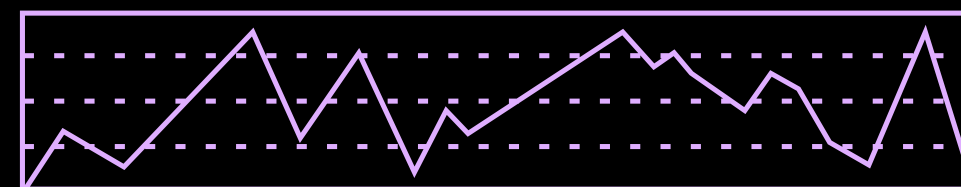
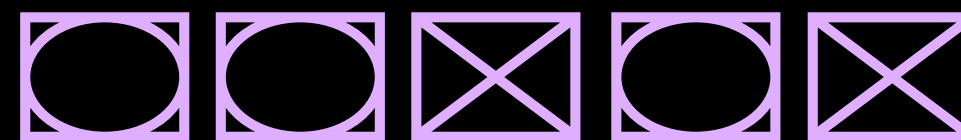
1. `innerText`:

- Restituisce solo il testo visibile all'interno di un elemento, ignorando completamente la struttura HTML.
- Quando si imposta `innerText`, il testo fornito viene considerato come testo puro e non interpreta eventuali tag HTML.
- È utile quando si desidera lavorare solo con il testo visibile e si vuole evitare l'interpretazione di tag HTML all'interno del contenuto.

2. `innerHTML`:

- Restituisce o imposta il contenuto HTML completo di un elemento, compresi tutti i tag HTML al suo interno.
- Quando si utilizza `innerHTML` per impostare il contenuto, è possibile inserire HTML valido e, di conseguenza, creare o modificare la struttura HTML all'interno dell'elemento.
- Può essere più potente, ma è importante utilizzarlo con cautela, poiché l'inserimento di HTML dinamico può aprire la porta a problemi di sicurezza, come gli attacchi XSS (Cross-Site Scripting) se i dati non sono opportunamente sanificati.

MODIFICA DEGLI ATTRIBUTI DEGLI ELEMENTI



Modifica degli Attributi degli Elementi

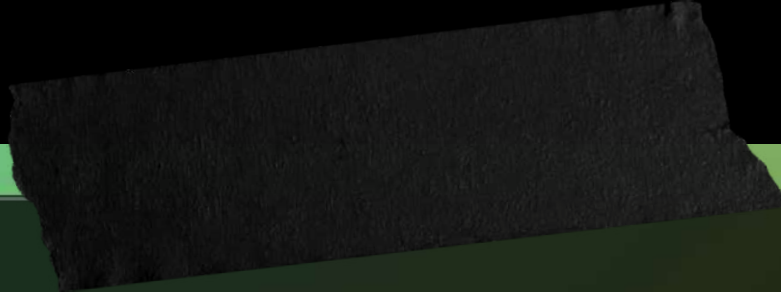
La modifica degli attributi degli elementi è fondamentale per adattare dinamicamente la presentazione di una pagina web. Utilizziamo principalmente il metodo `setAttribute` per impostare o modificare gli attributi di un elemento. Esempio:



```
var immagine = document.getElementById("miaImmagine");  
  
// Impostazione dell'attributo src  
immagine.setAttribute("src", "nuova-immagine.jpg");  
  
// Modifica di altri attributi  
immagine.setAttribute("alt", "Descrizione dell'immagine");
```

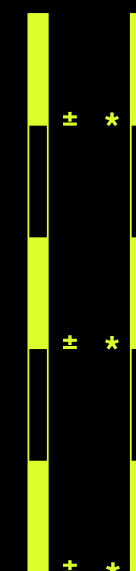
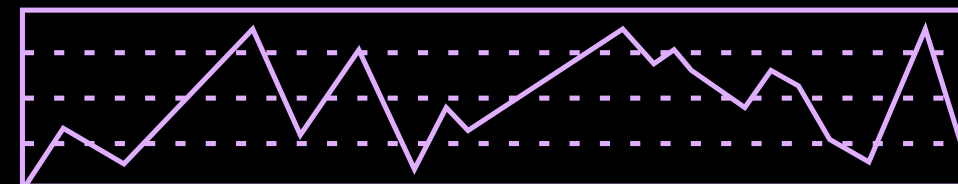
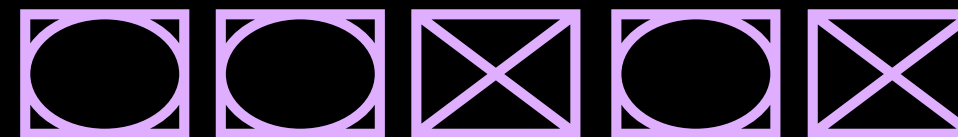

Modifica degli Attributi degli Elementi

Inoltre, possiamo accedere direttamente agli attributi di un elemento come se fossero proprietà dell'oggetto. Ad esempio:



```
console.log(image.src); // Restituisce il valore dell'attributo src
console.log(image.alt); // Restituisce il valore dell'attributo alt
```

CREAZIONE DI NUOVI ELEMENTI



document.createElement(tagName)

Il metodo createElement consente di creare un nuovo elemento con il nome specificato. Ad esempio:



```
var nuovoParagrafo = document.createElement("p");
```

document.createTextNode(text)


Il metodo createTextNode consente di creare un nuovo nodo di testo con il testo specificato. Ad esempio:

A code editor snippet with a dark background and a light blue border. It features three small circular window control buttons in the top-left corner. The code is written in a light blue monospace font.

```
var testoNodo = document.createTextNode("Questo è un nuovo paragrafo.");
```


parentNode.appendChild(childNode)

Il metodo appendChild viene utilizzato per aggiungere un nodo figlio all'interno di un nodo padre.
Ad esempio:



```
var mioDiv = document.getElementById("mioDiv");  
mioDiv.appendChild(nuovoParagrafo);
```

Esempio Completo

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Creazione di Nuovi Elementi</title>
</head>
<body>
  <div id="mioDiv">
    <!-- Il nuovo paragrafo verrà aggiunto qui -->
  </div>

  <script>
    // Creazione di un nuovo paragrafo
    var nuovoParagrafo = document.createElement("p");

    // Creazione di un nodo di testo
    var testoNodo = document.createTextNode("Questo è un nuovo paragrafo.");

    // Aggiunta del nodo di testo al paragrafo
    nuovoParagrafo.appendChild(testoNodo);

    // Recupero dell'elemento padre
    var mioDiv = document.getElementById("mioDiv");

    // Aggiunta del nuovo paragrafo all'elemento padre
    mioDiv.appendChild(nuovoParagrafo);
  </script>
</body>
</html>
```

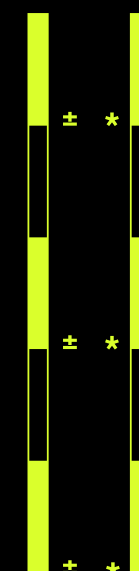
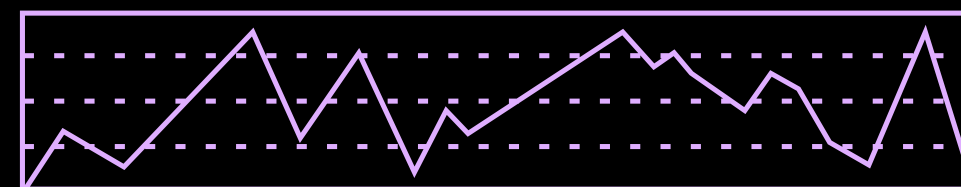
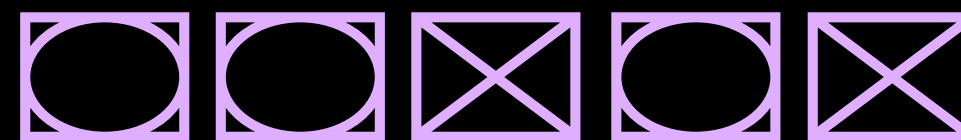
differenza tra innerText e createTextNode?

La differenza principale tra `createTextNode` e `innerText` riguarda cosa manipolano:

`createTextNode` crea un nuovo nodo di testo

`innerText` È una proprietà che consente di recuperare o impostare il testo visibile all'interno di un elemento, ignorando completamente la struttura HTML.

MANIPOLAZIONE DEGLI STILI E DELLE CLASSI



Modifica dello stile tramite js

La manipolazione degli stili CSS attraverso JavaScript consente di apportare modifiche dinamiche all'aspetto di un elemento sulla pagina.

In questo esempio, la dimensione e il colore di fondo dell'elemento con ID "mioDiv" vengono modificati dinamicamente utilizzando la proprietà **style** di JavaScript.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Manipolazione degli Stili CSS</title>
  <style>
    .mioElemento {
      width: 100px;
      height: 100px;
      background-color: lightblue;
    }
  </style>
</head>
<body>
  <div id="mioDiv" class="mioElemento"></div>

  <script>
    // Manipolazione degli stili
    var mioDiv = document.getElementById("mioDiv");
    mioDiv.style.width = "150px";
    mioDiv.style.height = "150px";
    mioDiv.style.backgroundColor = "lightgreen";
  </script>
</body>
</html>
```


Gestione delle Classi

La gestione delle classi consente di applicare stili predefiniti a un elemento o di modificarli dinamicamente.

In questo esempio, la classe "evidenziato" viene aggiunta al paragrafo e successivamente rimossa dopo 2 secondi, creando un effetto di evidenziazione.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Gestione delle Classi</title>
  <style>
    .evidenziato {
      color: red;
    }
  </style>
</head>
<body>
  <p id="mioParagrafo">Questo è un paragrafo.</p>

  <script>
    // Gestione delle classi
    var mioParagrafo = document.getElementById("mioParagrafo");

    // Aggiunta di una classe
    mioParagrafo.classList.add("evidenziato");

    // Rimozione di una classe
    setTimeout(function() {
      mioParagrafo.classList.remove("evidenziato");
    }, 2000);
  </script>
</body>
</html>
```

Gestione delle animazioni

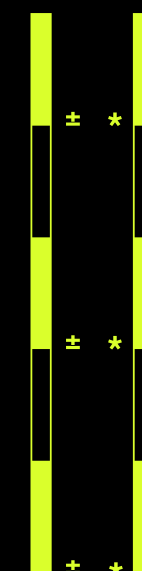
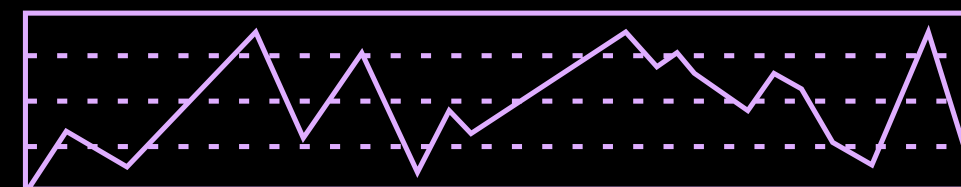
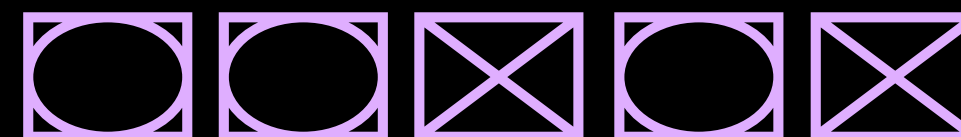
In questo esempio, l'animazione viene avviata cambiando dinamicamente le proprietà stile dell'elemento, sfruttando le transizioni CSS definite nel foglio di stile.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Animazioni CSS controllate</title>
  <style>
    .animazione {
      width: 100px;
      height: 100px;
      background-color: lightblue;
      transition: width 1s, height 1s, background-color 1s;
    }
  </style>
</head>
<body>
  <div id="mioDiv" class="animazione"></div>

  <script>
    // Animazioni CSS controllate tramite JavaScript
    var mioDiv = document.getElementById("mioDiv");

    // Modifica degli stili per avviare l'animazione
    mioDiv.style.width = "150px";
    mioDiv.style.height = "150px";
    mioDiv.style.backgroundColor = "lightgreen";
  </script>
</body>
</html>
```


EVENTI DEL DOM



Concetto di Eventi e Gestori di Eventi

Gli eventi DOM rappresentano interazioni dell'utente o modifiche nello stato del documento che possono essere rilevate e gestite tramite JavaScript.

Evento: Un'azione che si verifica, come un clic del mouse, un tocco su uno schermo, una pressione di un tasto, ecc.

Gestore di Eventi: Una funzione JavaScript che viene eseguita in risposta a un determinato evento.

Tipi Comuni di Eventi

Eventi di Mouse: click, mouseover, mouseout, ecc.

Eventi di Tastiera: keydown, keyup, ecc.

Eventi di Form: submit, change, ecc.

Eventi di Caricamento della Pagina: load, DOMContentLoaded, ecc.

Utilizzo di addEventListener per Ascoltare Eventi

Il metodo `addEventListener` consente di ascoltare eventi su un elemento HTML e associare una funzione di gestione degli eventi ad essi.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Ascolto e Gestione degli Eventi</title>
</head>
<body>
  <button id="mioBottone">Clicca Qui</button>

  <script>
    var mioBottone = document.getElementById("mioBottone");

    mioBottone.addEventListener("click", function() {
      alert("Hai cliccato sul bottone!");
    });
  </script>
</body>
</html>
```

Esempio pratico

In questo esempio, il colore di sfondo del div cambierà quando il mouse passa sopra di esso.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Esempio Pratico 1</title>
  <style>
    #mioElemento {
      width: 100px;
      height: 100px;
      background-color: lightblue;
    }
  </style>
</head>
<body>
  <div id="mioElemento"></div>

  <script>
    var mioElemento = document.getElementById("mioElemento");

    mioElemento.addEventListener("mouseover", function() {
      mioElemento.style.backgroundColor = "lightgreen";
    });

    mioElemento.addEventListener("mouseout", function() {
      mioElemento.style.backgroundColor = "lightblue";
    });
  </script>
</body>
</html>
```

In questo esempio invece, cosa succede?

```
● ● ●  
  
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>Esempio Pratico 2</title>  
</head>  
<body>  
  <input type="text" id="mioInput">  
  
  <script>  
    var mioInput = document.getElementById("mioInput");  
  
    mioInput.addEventListener("keydown", function(event) {  
      console.log("Tasto premuto:", event.key);  
    });  
  </script>  
</body>  
</html>
```