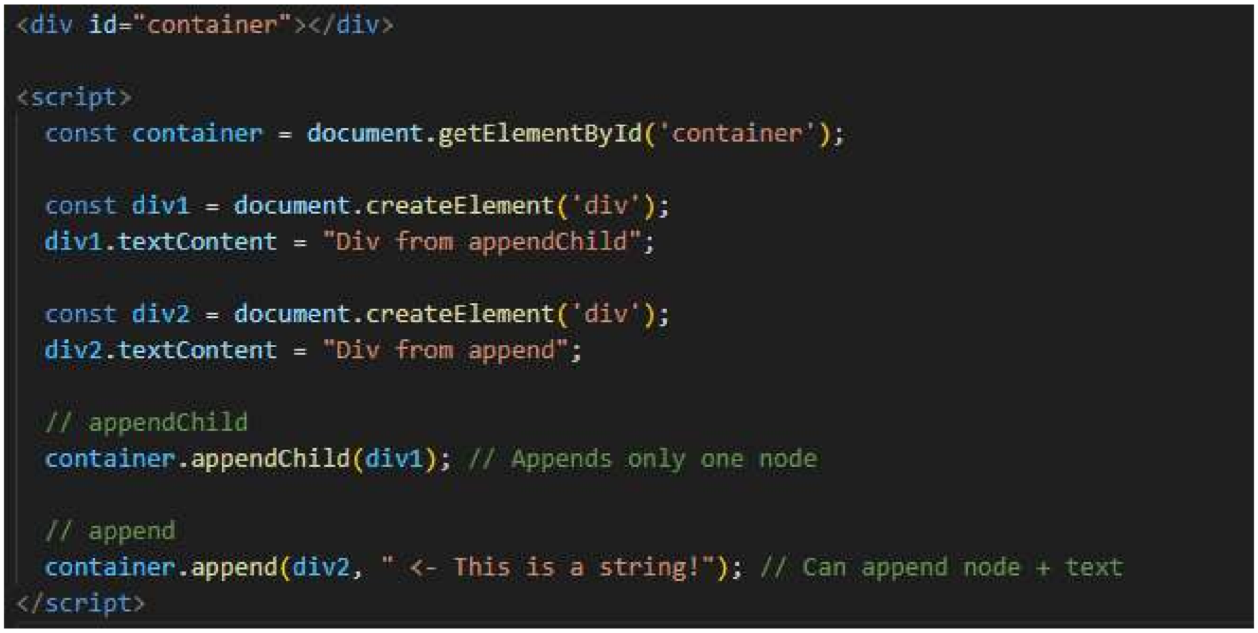
ASSIGNMENT 5

Q.1 - Find the difference between append and append child with code observation.

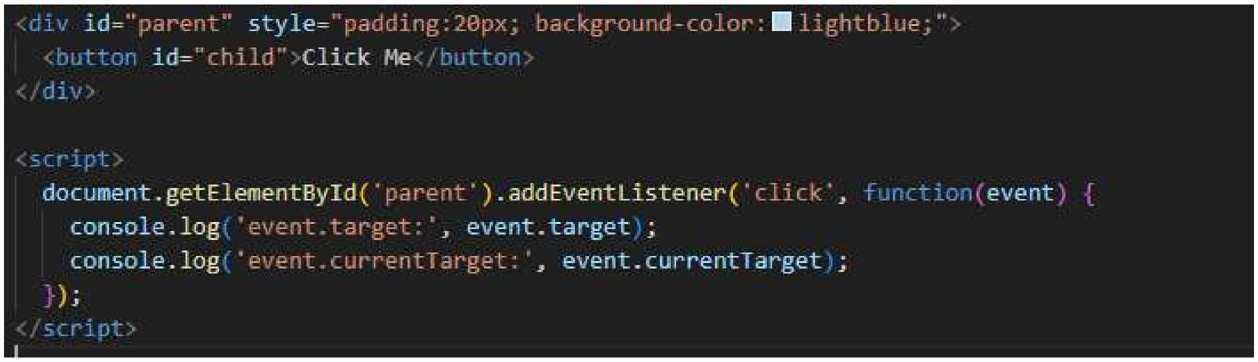
Ans - The append method allows adding multiple nodes or text to a parent element and supports both strings and DOM nodes. In contrast, appendChild only accepts a single node and returns the appended node. Unlike appendChild, append does not return a value and can handle plain text, whereas appendChild throws an error if given a string.



```
<div id="container"></div>

<script>
  const container = document.getElementById('container');

  const div1 = document.createElement('div');
  div1.textContent = "Div from appendChild";

  const div2 = document.createElement('div');
  div2.textContent = "Div from append";

  // appendChild
  container.appendChild(div1); // Appends only one node

  // append
  container.append(div2, " <- This is a string!"); // Can append node + text
</script>
```

<u>Q.2</u> - Find the difference between event.target and event.currenttarget with code observation.

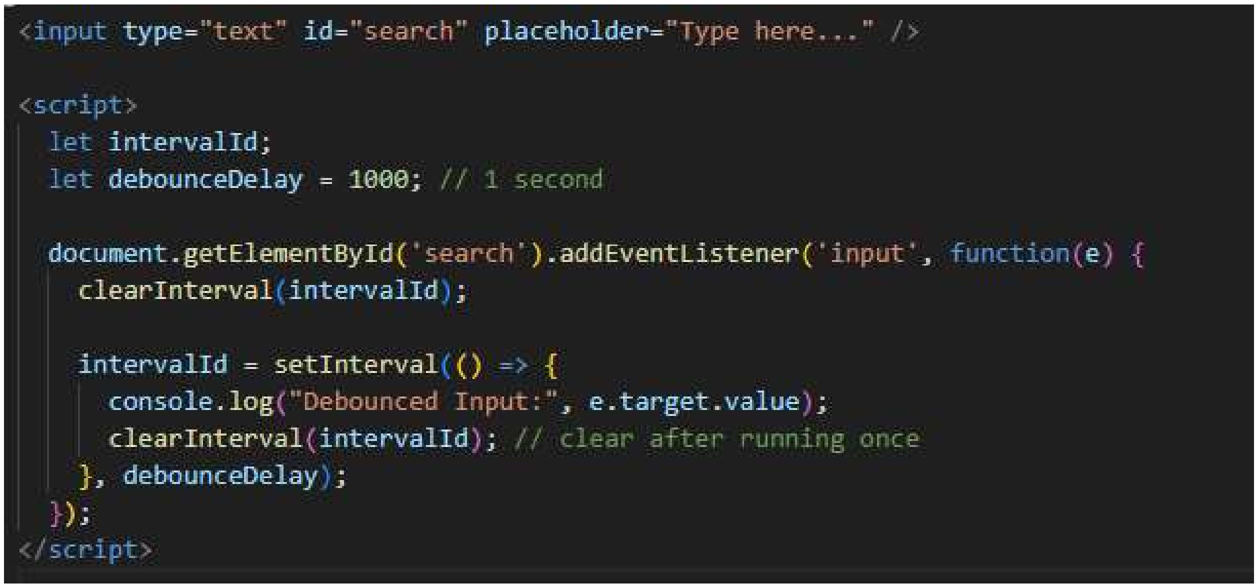
<u>Ans</u> - event.target is the exact element that fired the event, whereas event.currentTarget is the element on which the listener is attached. This distinction is key for event delegation, where a single listener on a parent checks event.target to see which child was clicked, enabling flexible and reusable handlers.



```html
<div id="parent" style="padding:20px; background-color:■lightblue;">
  <button id="child">Click Me</button>
</div>

<script>
  document.getElementById('parent').addEventListener('click', function(event) {
    console.log('event.target:', event.target);
    console.log('event.currentTarget:', event.currentTarget);
  });
</script>
```

<u>Q.3</u> - How to perform debouncing by Set Interval?

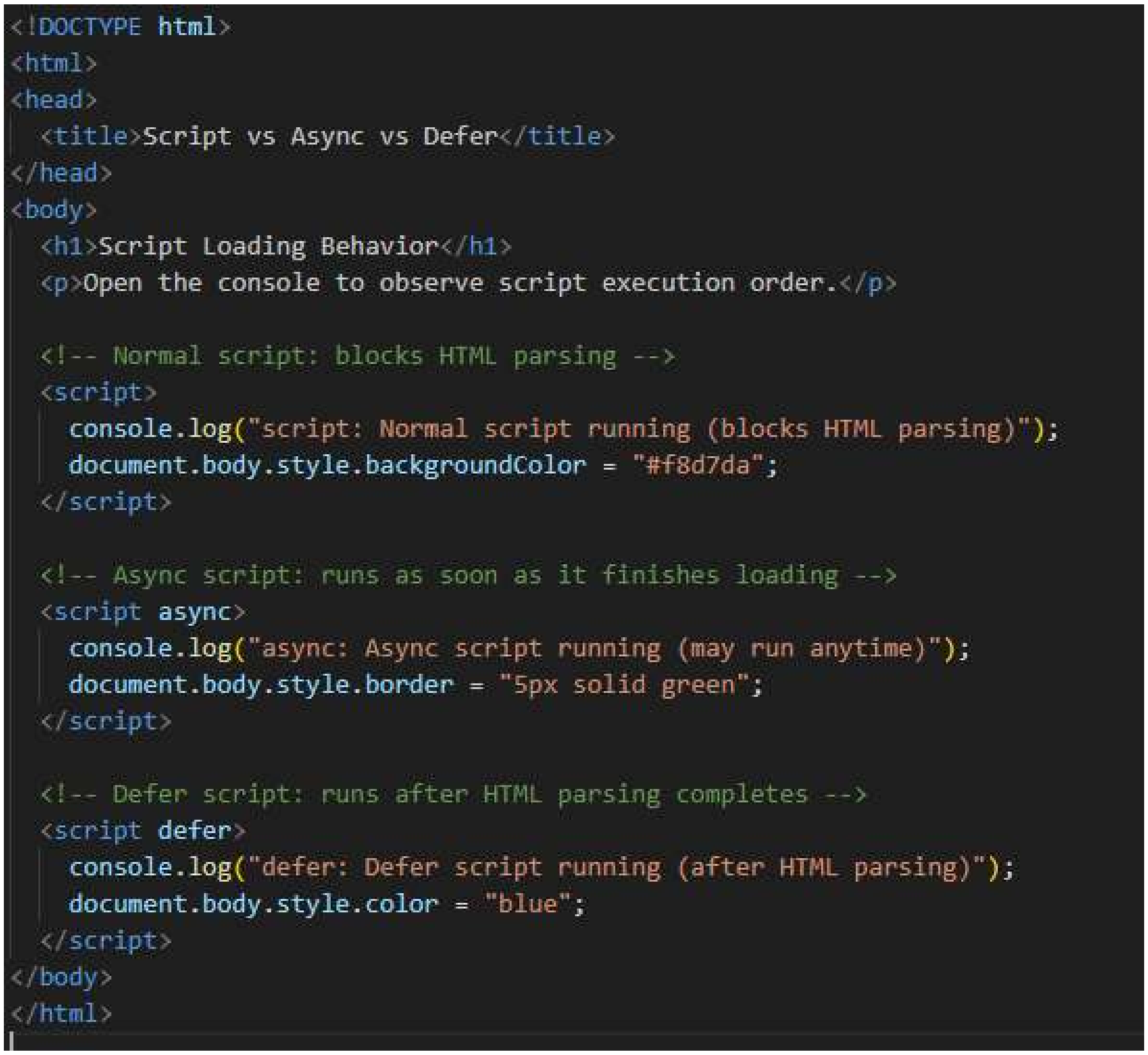
<u>Ans</u> - Debouncing limits how often a function runs, making it useful for events like typing or resizing. While setTimeout is commonly used, setInterval can mimic it by clearing and resetting the interval with each event. This ensures the function runs only after the user stops triggering events, reducing unnecessary calls and improving performance.



```html
<input type="text" id="search" placeholder="Type here..." />

<script>
  let intervalId;
  let debounceDelay = 1000; // 1 second

  document.getElementById('search').addEventListener('input', function(e) {
    clearInterval(intervalId);

    intervalId = setInterval(() => {
      console.log("Debounced Input:", e.target.value);
      clearInterval(intervalId); // clear after running once
    }, debounceDelay);
  });
</script>
```

Q.4 - Analyze and show the difference between script, async and defer practically.

Ans - In HTML, script tags can use async or defer to control when scripts load and run. A normal script tag blocks HTML parsing until the script finishes. With async, the script loads alongside the HTML but runs as soon as it’s ready, possibly interrupting parsing. Defer also loads in parallel but waits to run until the HTML is fully parsed, keeping the script order and preventing blocking.



```html
<!DOCTYPE html>
<html>
<head>
  <title>Script vs Async vs Defer</title>
</head>
<body>
  <h1>Script Loading Behavior</h1>
  <p>Open the console to observe script execution order.</p>

  <!-- Normal script: blocks HTML parsing -->
  <script>
    console.log("script: Normal script running (blocks HTML parsing)");
    document.body.style.backgroundColor = "#f8d7da";
  </script>

  <!-- Async script: runs as soon as it finishes loading -->
  <script async>
    console.log("async: Async script running (may run anytime)");
    document.body.style.border = "5px solid green";
  </script>

  <!-- Defer script: runs after HTML parsing completes -->
  <script defer>
    console.log("defer: Defer script running (after HTML parsing)");
    document.body.style.color = "blue";
  </script>
</body>
</html>
```