```
console.clear();

/*

------------------------------------------------------------

  THROTTLE FUNCTION (with leading + trailing behavior)

------------------------------------------------------------

  PURPOSE:

  --------

  Ensures that a function runs at most once every `delay` ms.

  Behavior:

  ---------

  ✓ First call → runs immediately (leading)

  ✓ If more calls happen before delay finishes → store only the LAST one

  ✓ After the delay → run the stored call (trailing)

*/

const throttle = (fn, delay) => {

    // Timestamp of the last executed call

    let lastCall = 0;

    // Holds the timeout ID for the trailing call

    let timeoutId = null;
```

```
38
39      // Stores the arguments of the most recent call (to run later)
40
41      let lastArgs = null;
42
43      // Return the throttled wrapper function
44
45      return function (...args) {
46
47          // Get the current time in ms
48
49          const now = Date.now();
50
51          /*
52
53          ------------------------------------------------------------
54
55            CASE 1: TOO SOON TO RUN (within delay period)
56
57          ------------------------------------------------------------
58
59            If the time difference is less than delay, we do NOT execute fn immediately.
60
61          ------------------------------------------------------------
62
63          */
64
65          if (now - lastCall < delay) {
66
67              // Save the latest arguments (only last call matters)
68
69              lastArgs = args;
70
71              // If a trailing timeout is NOT already scheduled:
72
73              if (!timeoutId) {
74
75                  // Calculate how much time is left before we can run again
```

```
76
77                  const remaining = delay - (now - lastCall);
78
79                  // Schedule a trailing execution
80
81                  timeoutId = setTimeout(() => {
82
83                      fn(...lastArgs);        // Run the last skipped call
84
85                      lastCall = Date.now();  // Update last execution time
86
87                      timeoutId = null;       // Timeout finished → reset
88
89                      lastArgs = null;        // Clear stored arguments
90
91                  }, remaining);
92              }
93
94              return; // Exit — do not execute fn now
95          }
96
97          /*
98
99          ------------------------------------------------------------
100
101           CASE 2: ENOUGH TIME HAS PASSED → RUN IMMEDIATELY
102
103         ------------------------------------------------------------
104
105         */
106
107         lastCall = now;  // Update last time function was run
108
109         fn(...args);     // Execute the function immediately
110     };
111 };
112
113 /*
```

```
114
115 ------------------------------------------------------------
116
117   CHAT MESSAGE FUNCTION
118
119 ------------------------------------------------------------
120
121   Just prints:
122
123   - message text
124
125   - current date & time (human readable)
126
127 ------------------------------------------------------------
128
129 */
130
131 const sendChatMessage = (message) => {
132
133     console.log("Sending Message:", message, "at", new Date().toLocaleString());
134 };
135
136 // Wrap sendChatMessage inside a throttle
137
138 const sendChatMessageWithSlowMessage = throttle(sendChatMessage, 2000);
139
140 /*
141
142 ------------------------------------------------------------
143
144   TEST CALLS
145
146 ------------------------------------------------------------
147
148 These 3 calls happen almost instantly (no delay between them)
149
150   Expected Output:
151
```

```
  ----------------

  ✓ First call → runs immediately

  ✓ Next calls → ignored but LAST one runs after 2 seconds

------------------------------------------------------------

*/

sendChatMessageWithSlowMessage("Hello");

sendChatMessageWithSlowMessage("Hello");

sendChatMessageWithSlowMessage("Hello");
```