

---

# ESERCIZIARIO JAVASCRIPT: EVENTI

*Missione Spaziale — Centro di Controllo*

*Prof. Mario Perna*

---

## Il tema

Sarai il tecnico software del Centro di Controllo Missione "Aurora". Ogni esercizio aggiunge una funzionalità al pannello di controllo che permette agli astronauti di lanciare, pilotare e comunicare durante la missione. Il pannello è una pagina HTML che, esercizio dopo esercizio, diventa interattiva grazie agli eventi JavaScript.

## Obiettivi didattici

- Comprendere il modello a eventi del DOM (event listener, oggetto `event`, propagazione)
- Collegare funzioni gestore a eventi del mouse, della tastiera, dei form e del ciclo di vita della pagina
- Progredire verso tecniche avanzate: delega degli eventi, drag and drop, eventi personalizzati, debounce/throttle

## Come leggere ogni esercizio

Ogni esercizio riporta Scenario, Consegna, API richieste e Criterio di verifica, oltre al livello di difficoltà indicato con le stelline (da ★☆☆☆☆ a ★★★★★). Ogni livello parte da un file HTML di base con gli elementi già presenti in pagina: lo studente scrive solo il file `script.js`.

## Prerequisiti

- HTML e CSS di base
- Sintassi JavaScript (variabili, funzioni, DOM: `document.querySelector`, `textContent`, `classList`)

# Livello 1 — Base: Sequenza di Avvio

## 1.1 — Accendi i sistemi

Difficoltà: ★☆☆☆☆

Scenario: il pannello ha tre pulsanti: "Motori", "Ossigeno", "Comunicazioni". All'avvio sono spenti (classe `spento`, sfondo grigio).

Consegna: al click su un pulsante, il sistema si accende: cambia colore (classe `acceso`, sfondo verde) e il testo diventa "ONLINE". Un secondo click lo spegne di nuovo.

API richieste: `addEventListener('click', ...)`, `classList.toggle`

Criterio di verifica: cliccando ogni pulsante più volte, lo stato si alterna correttamente tra acceso e spento.

**Codice HTML di partenza:**

```
<!DOCTYPE html>
<html lang="it">
<head>
  <meta charset="UTF-8">
  <title>Sequenza di Avvio</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>Sequenza di Avvio - Aurora</h1>
  <button id="btn-motori" class="spento">Motori</button>
  <button id="btn-ossigeno" class="spento">Ossigeno</button>
  <button id="btn-comunicazioni" class="spento">Comunicazioni</button>
  <script src="script.js"></script>
</body>
</html>
```

## 1.2 — Spia di sistema

Difficoltà: ★★☆☆☆

Scenario: sopra ogni pulsante c'è una scritta di stato, es. `<span id="stato-motori">In attesa</span>`.

Consegna: passando il mouse sopra (`mouseover`) un pulsante non ancora acceso, la spia mostra "Pronto al lancio"; togliendo il mouse (`mouseout`) torna a "In attesa". Se il sistema è già acceso, il passaggio del mouse non deve modificare il testo.

API richieste: `mouseover`, `mouseout`, `classList.contains`

Criterio di verifica: il testo cambia solo sui pulsanti spenti e torna sempre al valore originale quando il mouse esce.

**Codice HTML di partenza:**

```
<!DOCTYPE html>
<html lang="it">
<head>
  <meta charset="UTF-8">
  <title>Spia di Sistema</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="sistema">
    <button id="btn-motori" class="spento">Motori</button>
    <span id="stato-motori">In attesa</span>
  </div>
  <div class="sistema">
    <button id="btn-ossigeno" class="spento">Ossigeno</button>
    <span id="stato-ossigeno">In attesa</span>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

## 1.3 — Countdown manuale

Difficoltà: ★★☆☆☆

Scenario: un pulsante "T-1" decrementa un contatore visibile in pagina, che parte da 10.

Consegna: ogni click decrementa il contatore di 1. Quando arriva a 0, il testo diventa "LIFTOFF!" e il pulsante si disabilita.

API richieste: [click](#), [textContent](#), proprietà [disabled](#)

Criterio di verifica: non è possibile scendere sotto 0 e il pulsante resta disabilitato dopo il lancio.

**Codice HTML di partenza:**

```
<!DOCTYPE html>
<html lang="it">
<head>
  <meta charset="UTF-8">
  <title>Countdown</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1 id="countdown">10</h1>
  <button id="btn-t-meno-1">T-1</button>
  <script src="script.js"></script>
</body>
</html>
```

# Livello 2 — Intermedio: Configurazione dei Parametri di Volo

## 2.1 — Comandi da tastiera

Difficoltà: ★★☆☆☆

Scenario: un modulo di navigazione (un quadrato disegnato con CSS) deve muoversi sul pannello.

---

Consegna: alla pressione dei tasti freccia ([ArrowUp/Down/Left/Right](#)), sposta il modulo di 10px nella direzione corrispondente. Impedisci che il modulo esca dai bordi del pannello.

API richieste: `keydown`, `event.key`, `event.preventDefault()`

Criterio di verifica: il modulo si muove in tutte e 4 le direzioni e non esce mai dal contenitore.

**Codice HTML di partenza:**

```
<!DOCTYPE html>
<html lang="it">
<head>
  <meta charset="UTF-8">
  <title>Navigazione</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div id="pannello-nav">
    <div id="modulo"></div>
  </div>
  <script src="script.js"></script>
</body>
</html>

<!-- Nel style.css fornito: #pannello-nav ha position:relative,
      larghezza e bordo; #modulo ha position:absolute e dimensioni fisse. -->
```

## 2.2 — Validazione dei parametri di missione

Difficoltà: ★★★★★

Scenario: un form chiede altitudine (100–500 km) e velocità (7000–8000 m/s) prima del lancio.

Consegna: all'invio del form (`submit`), verifica che i valori siano nei range indicati. Se non lo sono, impedisce l'invio e mostra un messaggio d'errore accanto al campo non valido; altrimenti mostra "Parametri confermati".

API richieste: `submit`, `event.preventDefault()`, `event.target.elements`

Criterio di verifica: il form non si invia mai con dati fuori range; il messaggio corretto appare per ciascun campo.

**Codice HTML di partenza:**

```
<!DOCTYPE html>
<html lang="it">
<head>
  <meta charset="UTF-8">
  <title>Parametri di Missione</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <form id="form-parametri">
    <label for="altitudine">Altitudine (km):</label>
    <input type="number" id="altitudine" name="altitudine">
    <span class="errore" id="errore-altitudine"></span><br>
    <label for="velocita">Velocità (m/s):</label>
    <input type="number" id="velocita" name="velocita">
    <span class="errore" id="errore-velocita"></span><br>
    <button type="submit">Conferma parametri</button>
  </form>
  <p id="esito"></p>
  <script src="script.js"></script>
</body>
</html>
```

---

## 2.3 — Campo sotto osservazione

**Difficoltà:** ★★☆☆☆

Scenario: il campo "altitudine" deve essere evidenziato quando l'operatore ci sta lavorando.

Consegna: al focus, il bordo del campo diventa blu e appare un suggerimento ("Inserisci un valore tra 100 e 500"); al blur, il suggerimento scompare e, se il campo è vuoto, il bordo diventa rosso.

API richieste: `focus`, `blur`

Criterio di verifica: il comportamento del bordo e del suggerimento risponde correttamente a ogni cambio di focus.

**Codice HTML di partenza:**

```
<!DOCTYPE html>
<html lang="it">
<head>
  <meta charset="UTF-8">
  <title>Campo Osservato</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <label for="altitudine">Altitudine (km):</label>
  <input type="number" id="altitudine">
  <p id="suggerimento"></p>
  <script src="script.js"></script>
</body>
</html>
```

## Livello 3 — Avanzato: Carico Merci e Sistema di Allarme

### 3.1 — Gestione della stiva (delega degli eventi)

**Difficoltà:** ★★★☆☆

Scenario: una lista `<ul id="stiva">` contiene decine di `<li>`, uno per ogni cassa di rifornimento; nuove casse possono essere aggiunte dinamicamente.

Consegna: invece di aggiungere un listener a ogni `<li>`, collega UN SOLO listener `click` al contenitore `<ul>` e usa `event.target` per capire su quale cassa è stato cliccato, segnandola come "caricata" (classe `caricata`). Verifica che funzioni anche su casse aggiunte dopo il caricamento della pagina.

API richieste: delega degli eventi, `event.target`, `closest()`

Criterio di verifica: il click funziona su tutte le casse, comprese quelle aggiunte dinamicamente via JS dopo il caricamento.

**Codice HTML di partenza:**

```
<!DOCTYPE html>
<html lang="it">
<head>
  <meta charset="UTF-8">
  <title>Stiva</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <ul id="stiva">
    <li>Cassa 1</li>
    <li>Cassa 2</li>
    <li>Cassa 3</li>
  </ul>
  <button id="aggiungi-cassa">Aggiungi cassa</button>
  <script src="script.js"></script>
</body>
</html>
```

## 3.2 — Carico con trascinamento

Difficoltà: ★★★★★☆

Scenario: le casse (in una lista di scorta) devono essere trascinate dentro il modulo di carico (<div id="modulo-carico">).

Consegna: implementa il drag and drop: la cassa trascinata deve staccarsi dalla lista di scorta e apparire dentro il modulo di carico. Il modulo evidenzia il bordo mentre una cassa lo sovrasta.

API richieste: [dragstart](#), [dragover](#) (con [preventDefault](#)), [drop](#), [dataTransfer](#)

Criterio di verifica: ogni cassa può essere trascinata correttamente e il modulo mostra un feedback visivo durante il trascinamento.

**Codice HTML di partenza:**

```
<!DOCTYPE html>
<html lang="it">
<head>
  <meta charset="UTF-8">
  <title>Carico Merci</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="area">
    <h3>Lista di scorta</h3>
    <ul id="scorta">
      <li id="cassa-1" draggable="true">Cassa A</li>
      <li id="cassa-2" draggable="true">Cassa B</li>
      <li id="cassa-3" draggable="true">Cassa C</li>
    </ul>
  </div>
  <div id="modulo-carico" class="area">
    <h3>Modulo di carico</h3>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

## 3.3 — Allarme ossigeno

Difficoltà: ★★★★★☆

---

Scenario: il livello di ossigeno scende automaticamente nel tempo (simulato con `setInterval`).

Consegna: ogni secondo il livello scende dell'1%. Sotto il 20%, un allarme visivo lampeggia (alternando una classe `lampeggio`) finché l'operatore non clicca "Ripristina ossigeno", che ferma sia il countdown sia il lampeggio.

API richieste: `setInterval`, `clearInterval`

Criterio di verifica: l'allarme si attiva esattamente sotto il 20%, lampeggia in modo continuo e si ferma in modo pulito al click, senza intervalli residui attivi.

**Codice HTML di partenza:**

```
<!DOCTYPE html>
<html lang="it">
<head>
  <meta charset="UTF-8">
  <title>Allarme Ossigeno</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1 id="livello-ossigeno">100%</h1>
  <div id="allarme" style="display:none;">ALLARME: LIVELLO OSSIGENO CRITICO</div>
  <button id="btn-ripristina">Ripristina ossigeno</button>
  <script src="script.js"></script>
</body>
</html>
```

## Livello 4 — Esperto: Sistema di Comunicazione con la Base

### 4.1 — Evento personalizzato di guasto critico

Difficoltà: ★★★★★

Scenario: più moduli del pannello (ossigeno, motori, comunicazioni) devono reagire quando scatta un guasto critico in uno qualsiasi di essi.

Consegna: quando un modulo va in errore, genera un evento personalizzato `guasto-critico` con `CustomEvent`, includendo nel `detail` il nome del modulo guasto. Un gestore centrale, in ascolto sul `document`, riceve l'evento e mostra un banner rosso con il nome del modulo in errore.

API richieste: `new CustomEvent()`, `dispatchEvent()`, `addEventListener` su `document`

Criterio di verifica: il banner mostra sempre il modulo corretto, qualunque modulo generi il guasto.

**Codice HTML di partenza:**

---

```
<!DOCTYPE html>
<html lang="it">
<head>
  <meta charset="UTF-8">
  <title>Guasto Critico</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <button id="guasto-motori" data-modulo="Motori">Simula guasto Motori</button>
  <button id="guasto-ossigeno" data-modulo="Ossigeno">Simula guasto Ossigeno</button>
  <button id="guasto-comunicazioni" data-modulo="Comunicazioni">Simula guasto Comunicazioni</button>
  <div id="banner-allarme" style="display:none;"></div>
  <script src="script.js"></script>
</body>
</html>
```

---

## 4.2 — Telemetria senza sovraccarico (debounce)

Difficoltà: ★★★★★

Scenario: un campo di testo permette di cercare tra i log di missione mentre l'operatore digita; ogni ricerca simulata (una funzione `cercaNeiLog`) è "costosa".

Consegna: collega l'evento `input` del campo di ricerca a `cercaNeiLog`, ma con un debounce di 300ms: la funzione deve eseguirsi solo quando l'operatore smette di digitare per 300ms, non a ogni tasto premuto.

API richieste: `input`, `setTimeout/clearTimeout` per implementare il debounce

Criterio di verifica: digitando rapidamente una parola, `cercaNeiLog` viene chiamata una sola volta, non una volta per carattere.

**Codice HTML di partenza:**

```
<!DOCTYPE html>
<html lang="it">
<head>
  <meta charset="UTF-8">
  <title>Ricerca Log</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <input type="text" id="ricerca-log" placeholder="Cerca nei log...">
  <ul id="risultati-log"></ul>
  <script src="script.js"></script>
</body>
</html>
```

## 4.3 — Chi ascolta per primo? (bubbling e capturing)

Difficoltà: ★★★★★

Scenario: il pannello ha una struttura annidata: `pannello > modulo-comunicazioni > pulsante-invia`.

Consegna: aggiungi listener `click` su tutti e tre i livelli. Dimostra, stampando in console l'ordine di esecuzione, la differenza tra bubbling (default) e capturing (`{capture: true}`). Poi usa `event.stopPropagation()` sul pulsante per impedire che l'evento raggiunga il pannello esterno, e verifica il cambiamento nel log.

API richieste: terzo parametro di `addEventListener` (`capture`), `event.stopPropagation()`

Criterio di verifica: la console mostra l'ordine corretto in ciascuno dei tre scenari richiesti (bubbling, capturing, stopPropagation).

**Codice HTML di partenza:**

---

```
<!DOCTYPE html>
<html lang="it">
<head>
  <meta charset="UTF-8">
  <title>Comunicazioni</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div id="pannello">
    <div id="modulo-comunicazioni">
      <button id="pulsante-invia">Invia messaggio</button>
    </div>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

## Griglia di sintesi

| Es. | Titolo                   | Difficoltà | Evento/API                 | Competenza verificata                                    |
|-----|--------------------------|------------|----------------------------|----------------------------------------------------------|
| 1.1 | Accendi i sistemi        | ★☆☆☆☆      | click, classList           | Collegare un evento base e modificare lo stato del DOM   |
| 1.2 | Spia di sistema          | ★☆☆☆☆      | mouseover, mouseout        | Gestire eventi di hover in modo condizionato             |
| 1.3 | Countdown manuale        | ★☆☆☆☆      | click, textContent         | Gestire uno stato numerico e disabilitare un elemento    |
| 2.1 | Comandi da tastiera      | ★☆☆☆☆      | keydown, event.key         | Intercettare input da tastiera e limitare un movimento   |
| 2.2 | Validazione parametri    | ★☆☆☆☆      | submit, preventDefault     | Validare un form e bloccarne l'invio                     |
| 2.3 | Campo sotto osservazione | ★☆☆☆☆      | focus, blur                | Gestire lo stato di un campo in base al focus            |
| 3.1 | Gestione stiva           | ★☆☆☆☆      | delega, event.target       | Applicare la delega degli eventi su elementi dinamici    |
| 3.2 | Carico con trascinamento | ★☆☆☆☆      | dragstart, dragover, drop  | Implementare un'interazione drag and drop completa       |
| 3.3 | Allarme ossigeno         | ★☆☆☆☆      | setInterval, clearInterval | Gestire eventi periodici e la loro corretta interruzione |
| 4.1 | Evento guasto critico    | ★☆☆☆☆      | CustomEvent, dispatchEvent | Creare e propagare eventi personalizzati                 |
| 4.2 | Telemetria (debounce)    | ★★★★★      | input, setTimeout          | Ottimizzare la frequenza di esecuzione di un gestore     |
| 4.3 | Bubbling e capturing     | ★★★★★      | capture, stopPropagation   | Comprendere e controllare la propagazione degli eventi   |