

Esercizio MouseMove

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="description" content="mouse move">
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title></title>
    <style id="jsbin-css">
      #canvas {
        margin: 10px;
        border: 1px solid black;
        width: 300px;
        height: 300px;
      }

      #box {
        position: absolute;
        border: 1px solid black;
        width: 30px;
        height: 30px;
        top: 50px;
        left: 50px;
        background-color: red;
      }
    </style>
  </head>
  <body>
    <div id="canvas">
      <div id="box"></div>
    </div>
    <div id="output"></div>

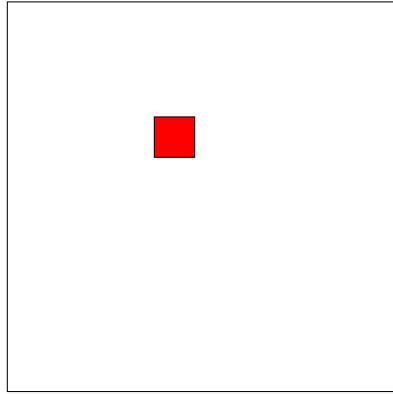
    <script id="jsbin-javascript">
      var box = document.getElementById("box");
      var output = document.getElementById("output");

      document.getElementById("canvas").addEventListener("mousemove", function(event) {
        var x = event.clientX;
        var y = event.clientY;
        var coords = "X coords: " + x + ", Y coords: " + y;
        output.innerHTML = coords;
        box.style.top = (y - 15) + "px";
        box.style.left = (x - 15) + "px";
      });
    </script>
  </body>
</html>
```

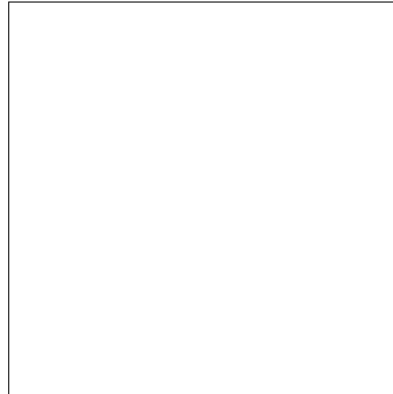
- Vogliamo creare un'area all'interno del quale un elemento quadrato di colore rosso dovrà seguire il nostro cursore. Vogliamo anche registrare e stampare le coordinate del cursore.
- Nell'HTML troviamo
 1. Un elemento div con id `canvas`, che consiste nell'area dove muoveremo il cursore.
 2. Un elemento div con id `box`, contenuto all'interno del primo div, che si muoverà col cursore.
 3. Un elemento div con id `output` dove stamperemo le coordinate del cursore.
- Creiamo l'oggetto `box` con cui ci collegheremo al secondo div.
- Creiamo l'oggetto `output` con cui ci collegheremo al terzo div
- Creiamo un *listener* che verifica se si manifesta un evento `mousemove`¹ relativamente al secondo div.
- Attraverso le proprietà `clientX` e `clientY` dell'oggetto `event` otteniamo le coordinate del cursore.

¹ L'evento `mousemove` consiste nel movimento del cursore sopra un particolare elemento div.

- Con `innerHTML` aggiorniamo il contenuto del terzo div (poniamo come contenuto la proprietà `coords`).
- A questo punto l'unica cosa che ci manca da gestire è il movimento del secondo div: con le ultime due righe della funzione impostiamo le proprietà CSS `top` e `left` relativamente a `box`. Entrambi i valori numerici vengono decrementati di 15 in modo tale che il cursore si posizioni al centro dell'elemento (che ha lunghezza e larghezza pari a 30px)
- **Osservazione:** non si è posto un controllo vero e proprio relativamente ai movimenti del div `box` (può essere portato fuori dal div `canvas`). Ci occuperemo di questa cosa nell'esercizio successivo.
- **Output:**



X coords: 147, Y coords: 114

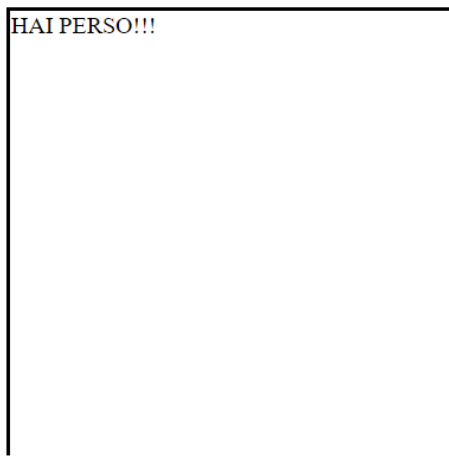
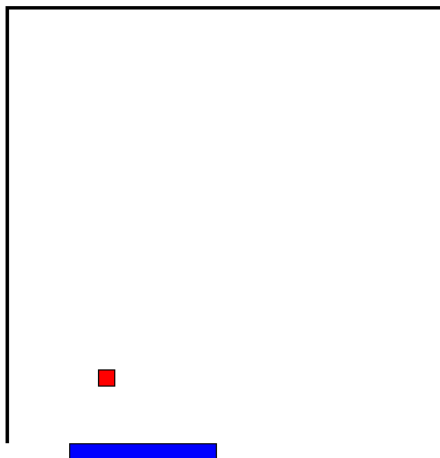


X coords: 363, Y coords: 177

Arkanoid

Arkanoid è uno storico gioco degli anni 80, fonte di ispirazione per questo esercizio svolto durante il laboratorio.

- **Output:**



Prendiamo il codice:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Arkanoid</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <style id="jsbin-css">
      #canvas {
        position: absolute;
        margin: 0px;
        border: 3px solid black;
        width: 300px;
        height: 300px;
        border-bottom: none;
      }
    </style>
  </head>
  <body>
    <div id="canvas">
      <div id="ball">
        <div id="paddle">
          <div id="text">
            HAI PERSO!!!
          </div>
        </div>
      </div>
    </div>
  </body>
</html>
```

L'area di gioco è posizionata in alto a sinistra della pagina, ha un bordo di 3px continuo, lunghezza e larghezza di 300px, bordo inferiore assente.

```
#ball {
position: absolute;
border: 1px solid black;
width: 10px;
height: 10px;
top: 50px;
left: 50px;
background-color: red;
}
```

La palla presenta `position: absolute` (per potersi muovere), ha un bordo di 1px continuo e nero, lunghezza e larghezza di 10px, sfondo di colore rosso. Il suo punto di partenza nell'area di gioco ha coordinate (50, 50), cioè 50px dall'alto e 50px da sinistra.

```
#bar {
position: absolute;
border: 1px solid black;
width: 100px;
height: 10px;
top: 300px;
left: 50px;
background-color: blue;
}
</style>
```

La barra presenta `position: absolute` (per potersi muovere orizzontalmente), ha un bordo di 1px continuo e nero, sfondo di colore blu, lunghezza di 10px e larghezza di 100px.

La barra è posizionata in fondo all'area di gioco (top:300px, esattamente in fondo) a 50px da sinistra.

```
</head>
```

```
<body>    Area di gioco
```

```
  <div id="canvas">
```

```
    <div id="ball"></div> ← Pallina che si muove
```

```
    <div id="bar"></div> ← Barra che dobbiamo muovere per far rimbalzare la pallina
```

```
  </div>
```

```
  <script id="jsbin-javascript">
```

```
    // Recupero i tre elementi HTML necessari per dare forma al gioco
```

```
    var canvas = document.getElementById("canvas");
```

```
    var ball = document.getElementById("ball");
```

```
    var bar = document.getElementById("bar");
```

```
    var x = 50;
```

```
    var y = 20;
```

```
    var inc_x = 1;
```

```
    var inc_y = 1;
```

```
    var bar_x = 0;
```

```
    // Imposto con questo handler il movimento orizzontale della bar in funzione del mio mouse
```

```
    // (la differenza mi permette di avere il cursore del mouse nel centro della barra).
```

```
    // Il movimento del mouse avviene all'interno dell'area di gioco, non solo sulla barra.
```

```
    canvas.addEventListener("mousemove", function(event) {
```

```
        bar.style.left = (event.clientX - 50) + "px";
```

```
        // Mi salvo la posizione per verificare se la barra è riuscita a far rimbalzare la palla
```

```
        bar_x = event.clientX;
```

```
    });
```



Mi salvo l'identificativo restituito dalla setInterval per usarlo nella clearInterval()

```
    var interval = setInterval(function() {
```

```
        x += inc_x;
```

Ogni volta altero la posizione (top,left) della ball.

```
        y += inc_y;
```

```
        ball.style.left = x + "px";
```

Gli incrementi sono determinati dai rimbalzi e possono avere valore negativo.

```
        ball.style.top = y + "px";
```

```
    // In caso di rimbalzo laterale altero la direzione della ball.
```

```
    if (x<=0 || x>=290) inc_x *= -1;
```

```

// In caso di rimbalzo in alto altero la direzione della ball
if (y<=0) inc_y *= -1;

// Controllo il rimbalzo in basso in prossimità del fondo
if (y>=290) {
    // Se la barra non ha fermato lo spostamento svuoto l'area di gioco scrivendo "HAI PERSO!!!"
    // Inoltre eseguo la clearInterval visto che non c'è più una ball da muovere.
    // In bar_x salvo dove si trova il centro della bar (motivo del -50 e del +50 nelle condizioni)
    if (x+5 < bar_x-50 || x+5 > bar_x+50 ) {
        canvas.innerHTML = "HAI PERSO!!!";
        clearInterval(interval);
    }
    else {
        // In base al punto della bar dove c'è stato contatto con la ball stabilisco l'inclinazione.
        // Più sono vicino agli estremi, maggiore sarà l'inclinazione della ball
        inc_x = ((x+5) - bar_x)/50;

        // In caso di rimbalzo in alto altero l'inclinazione della ball
        inc_y *= -1;
    }
}
}, 5);
</script>
</body>
</html>

```

Questionario

- Anteprima della pagina e codice completo posto alla fine dell'esercitazione.
- Questo esercizio consiste in un questionario con diversi controlli. Il documento prevede anche la presenza di un orologio (che sarà aggiornato costantemente) e di una textarea che segnala gli ultimi eventi che si sono manifestati.
- Le pagina da analizzare sono:
 - o index.html, che contiene la struttura della pagina (codice completo qualche pagina più avanti)
 - o css/forms.css, grafica del form (inclusa nell'head dell'index.html, codice completo qualche pagina più avanti)
 - o js/orologio.js, codice relativo all'input testuale contenente l'orologio
 - o js/gestoreForm.js, gestione degli errori del form
 - o js/feedback.js, gestione della textarea per la domanda "Cosa ne pensi del questionario?"
- Prendiamo come punto di partenza la index. Quali contenuti troviamo?
 - o Un input testuale di sola lettura (attributo readonly) contenente una descrizione della data e dell'ora corrente. Questa descrizione viene aggiornata ogni secondo usando Javascript.

Oggi è lunedì 30 novembre 2020, ore 11:30:29

```
<p><input name="orologio" type="text" value=" " size="100" readonly></p>
```

Con gli attributi dell'elemento body impostiamo l'esecuzione di una funzione clock() ogni 1000 millisecondi (cioè ogni secondo)

```
<body onLoad="setInterval('clock()',1000)"> [...]
```

Il javascript relativo si trova nel file orologio.js, incluso subito dopo l'orologio.

```
<script type="text/javascript" src="./js/orologio.js"></script>
```

Analizziamolo:

```
// Utilizzeremo questi due array per rendere più gradevole la lettura dell'orologio
```

```
var MONTH = ["gennaio", "febbraio", "marzo", "aprile", "maggio",  
"giugno", "luglio", "agosto", "settembre", "ottobre", "novembre",  
"dicembre"];
```

```
var DAY = ["domenica", "luned\u00EC", "marted\u00EC",  
"mercoled\u00EC", "gioved\u00EC", "venerd\u00EC", "sabato"];
```

```
// Recuperiamo le informazioni grafiche relative all'input testuale
```

```
// l'input è identificato dall'attributo name="orologio" e si trova in un form
```

```
// a sua volta identificato da name="mio_form"
```

```
var s = document.mio_form.orologio.style;  
s.borderStyle = "none"; // Rimuoviamo il bordo presente di default negli input  
s.fontFamily = "monospace"; // Impostiamo come font "monospace"  
s.fontWeight = "bolder"; // Indichiamo che il font deve essere più scuro rispetto alla  
font-weight ereditata  
s.fontSize = "x-large"; // Indichiamo la grandezza del font come "Extra large"
```

```
// Funzione eseguita ogni secondo per aggiornare il valore dell'input. Utilizziamo gli array all'inizio per  
stampare i giorni della settimana e i mesi dell'anno in formato testuale
```

```
function clock() {  
    var now = new Date(); // Utilizzo l'oggetto built-in Date per ottenere le informazioni  
  
    var m = now.getMonth(); // 0 = gennaio, 1 = febbraio, ...  
    var d = now.getDate(); // 1 = primo del mese, ...  
    var g = now.getDay(); // 0 = domenica, 1 = lunedì, ...  
    var a = now.getFullYear(); // YYYY
```

```

var time = now.toLocaleTimeString(); //HH:MM:SS

var dateValue = "Oggi \u00e8 " + DAY[g] + ' ' + d + ' ' +
MONTH[m] + ' ' + a + ", ore " + time;

// Aggiorno il valore dell'input
document.mio_form.orologio.value = dateValue;
}

```

- Un **fieldset** (cioè un insieme di campi) etichettato come “Questionario” (elemento `legend`) con al suo interno tre differenti `fieldsets`:

```

<fieldset name="questionario">
  <legend>Questionario:</legend>
  <div id=form_left>
    <fieldset name="dati_personali">
      <legend>Dati personali</legend>
      <div>

```

Start tag del fieldset ed elemento legend

- **Dati personali:** nome, cognome, password, e-mail, data di nascita, sito web, curriculum (file da caricare)
- **Domande:** sport praticato, sport divertente, passatempo, OS utilizzato e colore preferito.
- **Feedback:** giudizio in un range da 1 a 5 e parere testuale sul questionario

Non scriveremo tutto il codice, vista la sua lunghezza.

Quali sono le cose interessanti da notare nei vari controlli?

```

<input name="nome" size="15" type="text" placeholder="Es: Mario"
pattern="[a-zA-Z\s]+" required>

```

- L'attributo `name` che identifica il valore del controllo permettendo di recuperarlo sia nel Javascript che nel PHP (ne parleremo più avanti del PHP)
- L'attributo `size`, con cui stabiliamo un numero massimo di caratteri inseribili.
- L'attributo `type` con cui indichiamo il tipo del controllo. La selezione del tipo permette di svolgere alcuni controlli lato client senza scrivere codice Javascript (grazie al browser) oltre a impostare un aspetto grafico più consono al dato. Nel codice sono inseriti molti esempi (`text`, `email`, `password`, `file`, `checkbox`, `radio`, `range`)
- L'attributo `placeholder`, utile per mostrare all'utente (all'interno del controllo) un esempio di compilazione.
- L'attributo `required`, che rende il campo obbligatorio (e impedisce la sottomissione del form se questo non è stato compilato)
- L'attributo `pattern`, che permette di stabilire controlli di tipo sintattico ai valori inseriti usando le RegExp.
- L'attributo `readonly`, che impedisce la compilazione del controllo (già visto nell'orologio).
- I vari elementi e sottoelementi per gestire alcuni controlli (la `select` con gli elementi `option` e un esempio di selezione multipla, la `datalist` per stabilire una lista di suggerimenti in un input, i vari input per esprimere le opzioni di tipo `checkbox` o `radio`, la `textarea`...)

```

<label>Scegli dei passatempi:<br>
  <select multiple="multiple" name="passatempi" size="6">
    <option value="Musica">Musica</option>
    <option value="Cinema">Cinema</option>
    <option value="Sport">Sport</option>
    <option value="Viaggi">Viaggi</option>
    <option value="Lecture">Lecture</option>
    <option value="Altro">Altro</option>
  </select>
</label>

```

Select con selezione multipla

```

</label>
<label>Quale OS utilizzi?<br>
<input list="sistemi_operativi" name="sistemi_operativi">
<datalist id="sistemi_operativi">
  <option value="Windows">
  <option value="Mac OS X">
  <option value="Linux">
</datalist>
</label>

```

Input con datalist (valori suggeriti)

```

<div>
  Quale sport pratici?<br>
  <input name="sport" value="Calcio" type="checkbox" >Calcio<br>
  <input name="sport" value="Pallavolo" type="checkbox" >Pallavolo<br>
  <input name="sport" value="Danza" type="checkbox" >Danza<br>
  <input name="sport" value="Altro" type="checkbox" >Altro<br>
</div>
<div>Quale sport reputi piú divertente?<br>
  <input name="divertente" value="Calcio" type="radio" >Calcio<br>
  <input name="divertente" value="Pallavolo" type="radio" >Pallavolo<br>
  <input name="divertente" value="Danza" type="radio" >Danza<br>
  <input name="divertente" value="Altro" type="radio" >Altro<br>
</div>

```

checkbox e radio

```

<label>
  Giudizio (da 1 a 5):<br>
  <input type="range" name="voto" min="1" max="5" step="1" value="1" onInput="showValue(this.value)">
</label>

```

Range con valore minimo 1, valore massimo 5 e step 1 (cioè muovendo il controllo posso avere come valori solo 1,2,3,4 e 5)

```

<label>
  Cosa ne pensi del questionario?<br>
  <textarea name="messaggio_testo" rows="7" cols="30" onKeyDown="update(document.mio_form)" id="feedback" ></textarea><br>
</label>

```

Textarea impostata per avere al massimo 7 righe e al più 30 caratteri per riga
(informazioni di utilità esclusivamente grafica, determinano width ed height della textarea)

- Al di là del tipo di controllo vi invito a rivolgere la vostra attenzione a quest'area

Pulsanti:

svuota eventi INVIA INVIA SENZA VALIDARE RESET



Eventi in ingresso:

```

<input name="bottone_sottometti_no_validazione" value=
"INVIA SENZA VALIDARE" type="submit" formnovalidate>

```

```

Blur: bottone_azzerare (RESET)
Focus: bottone_azzerare (RESET)
Blur: bottone_azzerare (RESET)
Focus: bottone_azzerare (RESET)

```

I pulsanti ci permettono di gestire la sottomissione della form. L'area "Eventi in ingresso" consiste in una textarea aggiornata ogni volta che compiamo una certa azione sugli input (cliccare l'input – focus, abbandonarlo dopo aver cliccato – blur...). Il pulsante "svuota eventi" elimina il contenuto della textarea.

- Concludiamo con l'inclusione di altri due file js:

```

<script type="text/javascript" src="./js/gestoreForm.js"></script>
<script type="text/javascript" src="./js/feedback.js"></script>

```

```
- Vediamo gestoreForm.js
// Contiene i valore dell'attributo class
// in modo da indicare in quale stato si trova lo specifico elemento
STYLE_TYPE = ["error", "warning", ""];

function showValue(newVal) {
    document.mio_form.votoDisplay.value = newVal;
}
```

Funzione utilizzata nel range del giudizio. Vediamo il codice:

Guidizio (da 1 a 5): **2**

```
<div>
<label>
Giudizio (da 1 a 5):<br>
<input type="range" name="voto" min="1" max="5" step="1" value="1"
onInput="showValue(this.value)">
</label>
<input type="text" size="5" name="votoDisplay" readonly value="1" style="border-
style: none; font-size: 10pt;">
</div>
```

Ogni volta che aggiorniamo il valore del controllo `voto` rendiamo esplicito il numero appena selezionato ponendolo come valore dell'input `votoDisplay`. Questo input non ha bordi, ed è in sola lettura.

// La funzione 'detail' aggiunge i dettagli di un evento all'elemento area di testo "area_testo" presente nel form "mio_form". Viene invocata da vari gestori evento.

```
function detail(field, eventName) {
    var eventTextArea = document.mio_form.area_testo;
    var fieldName = field.name;
    var newValue = " ";
    if ((field.type == "select-one") || (field.type == "select-multiple")){
        for(var i = 0; i < field.options.length; i++){
            if (field.options[i].selected)
                newValue += field.options[i].value + " ";
        }
    }
    else
        if (field.type == "textarea")
            newValue = "...";
        else
            if (field.type == "fieldset")
                newValue = "!";
            else
                newValue = field.value;

    var message = eventName + ": " + fieldName + ' (' + newValue + ')\n';
    eventTextArea.value += message;
}
```

Funzione associata in `addHandlers()` a certe azioni sugli input. Permette di aggiornare il contenuto della textarea contenente le operazioni svolte (`area_testo`). Creo la nuova riga (e la salvo in `message`) per poi concatenarla al contenuto già presente. Il messaggio presenta:

- Il nome dell'evento (`eventName`, parametro in ingresso)
- Il nome del campo (`fieldName`)
- Il nuovo valore (`newValue`). Il codice gestisce i seguenti casi:
 - o Selezione su controlli di tipo `select` (gestisco singole e multiple selezioni)
 - o Passaggio su `textarea` (non metto il valore della textarea, mi limito a puntini di sospensione)
 - o Click su un `fieldset` (stampo un punto esclamativo come valore)
 - o In tutti gli altri casi stampo semplicemente il valore


```
function setStyle(field, styTypeIndex){
    var parent = field.parentNode;
    parent.className = STYLE_TYPE[styTypeIndex];
    field.className = STYLE_TYPE[styTypeIndex];
}
```

Funzione eseguita nella `refreshStyle()`: dato un controllo e uno stato identificato numericamente si modifica la classe dell'input e dell'elemento padre tenendo conto dell'array `STYLE_TYPE`.

Ricordiamo l'array introdotto all'inizio:
`STYLE_TYPE = ["error", "warning", ""];`

E-mail: *

dfghfd



Esempio: con `setStyle(input, 2)` modifico la classe dell'input (per rimuovere il colore rosso dello sfondo e del bordo) e dell'elemento padre (per far sparire l'iconcina di errore). Vedere il CSS per avere le idee più chiare.

```
function invalidHandler(evt){
    // evt.preventDefault();
    var field = evt.target;
    var validity = field.validity;
    // field.setCustomValidity("");
    if (validity.valueMissing) {
        setStyle(field, 1);
        return;
    }

    if (validity.patternMismatch || validity.typeMismatch){
        setStyle(field, 0);
        return;
    }
}
```

Password: *



E-mail: *

dfghfd



Funzione eseguita per ogni input in caso di evento "invalid" (quando si trovano valori di input non consistenti). Il parametro in ingresso è l'oggetto event relativo. Prendo il target dell'evento (l'input, in questo caso) e verifico la validità: se l'input non ha valore imposto lo style *warning*, se l'input è inconsistente pongo lo stato *error*.

```
function checkConstraint(evt){
    var field = evt.target;
    // field.setCustomValidity("");
    if (!field.checkValidity()){
        invalidHandler(evt);
        return;
    }

    setStyle(field, 2);
}
```

Funzione eseguita se rendo eseguibile la riga commentata in `addHandlers()` relativa all'evento blur. Praticamente verifico non appena abbandono l'input (quando perdo il focus) se il valore posto è valido. In caso contrario chiamo la `invalidHandler`.

```
function refreshStyle(form){
    for(var i = 0; i < form.elements.length; i++) {
        var field = form.elements[i];
        setStyle(field, 2);
    }
}
```

```
function send(form){
    var formOk = true;
```

Eseguo quando resetto il form con l'apposito bottone. Serve per rimuovere tutti gli avvisi di errore dei vari input (l'input di tipo reset reimposta al valore di default i vari input ma non rimuove le classi *error* e *warning*)

```
    alert("Form Inviata");
    return formOk;
```

Ci limitiamo a stampare un alert all'utente e a restituire *true*.

```
}
```

// Creo una serie di funzioni attraverso il costruttore (necessario perché la funzione detail presenta due parametri) e le associo, per ogni elemento input, ai vari elementi. Non controllo se l'elemento supporta l'evento associato (in quel caso non succederà niente). Utilizzo un for per associare gli eventi a tutti gli elementi presenti nel form (un risparmio di tempo considerando il numero di questi nel codice).

```
function addHandlers(form) {
    var clickHandler      = new Function("detail(this, 'Click')");
    var changeHandler     = new Function("detail(this, 'Change')");
    var focusHandler      = new Function("detail(this, 'Focus')");
    var blurHandler       = new Function("detail(this, 'Blur')");
    var selectHandler     = new Function("detail(this, 'Select')");
    var dblclickHandler    = new Function("detail(this, 'dblClick')");
    var invalidHandlerDetail = new Function("detail(this, 'Invalid')");
    // e.addEventListener("blur", checkConstraint, false);

    for(var i = 0; i < form.elements.length; i++) {
        var e = form.elements[i];
        e.onclick        = clickHandler;
        e.onchange        = changeHandler;
        e.onfocus        = focusHandler;
        e.onblur          = blurHandler;
        e.onselect        = selectHandler;
        e.ondblclick       = dblclickHandler;
        e.addEventListener("invalid", invalidHandler, false);
    }
}
```

// Se uso il bottone “svuota eventi” resetto il contenuto della textarea e aggiungo subito un nuovo evento nella textarea stessa (il click del bottone)

```
form.bottone_svuota.onclick = new Function("this.form.area_testo.value='';
detail(this, 'Click');");
```

// Se uso il bottone “RESET” resetto tutti i controlli del form, inserisco nella textarea che ho fatto click sul bottone di reset e rimuovo tutte le classi di error e warning usate negli input per segnalare i problemi.

```
form.bottone_azzerare.onclick = new Function("this.form.reset(); detail(this,
'Click');refreshStyle(document.mio_form);");
```

// Eseguo la funzione send() quando sottometto la form (sia quando valido che quando non valido).

```
form.onsubmit = new Function("return send(document.mio_form)");
}
```

// Attiviamo il form aggiungendo i possibili gestori

```
addHandlers(document.mio_form);
```

- Vediamo feedback.js

// Recupero l'input readonly contatore avente per valore il numero di caratteri rimanenti per riempire la textarea.

```
var MAX_CHAR = document.mio_form.contatore.value;
```

// Se vado col mouse sopra la textarea imposto uno sfondo blu e un colore bianco per i caratteri.

```
document.mio_form.messaggio_testo.onmouseover = function() {
    var feedbackAreaTesto = document.mio_form.messaggio_testo;
    feedbackAreaTesto.style.backgroundColor = "blue";
    feedbackAreaTesto.style.color = "white";
}
```

// Se mi muovo fuori dalla textarea ripristino lo sfondo bianco e il colore nero per i caratteri.

```
document.mio_form.messaggio_testo.onmouseout =
function() {
    var feedbackAreaTesto = document.mio_form.messaggio_testo;
```


```

feedbackAreaTesto.style.backgroundColor = "white";
feedbackAreaTesto.style.color = "black";
}

function update(form) {
    if (form.messaggio_testo.value.length > MAX_CHAR) {
        form.messaggio_testo.value = form.messaggio_testo.value.substring(0, MAX_CHAR);
        form.contatore.value = 0;
    }
    else {
        form.contatore.value = MAX_CHAR - form.messaggio_testo.value.length;
    }
}

```

Name della textarea



Eseguo la funzione `update()` ogni volta che inserisco o rimuovo un carattere dal valore della textarea:

```

<textarea name="messaggio_testo" rows="7" cols="30"
onkeyup="update (document.mio_form)" id="feedback" style="background-color: white;
color: black;"></textarea>

```

La funzione mi permette di intervenire quando supero la lunghezza massima: se si supera la lunghezza massima rimuovo i caratteri in eccesso con la `substring()` e setto il valore del contatore a 0, altrimenti aggiorno il valore del contatore con la differenza tra il numero massimo di caratteri e il numero di caratteri utilizzati.

Esercizio: Orologio grafico

- Realizziamo un orologio grafico con lancette in movimento.
- **Codice HTML:**

```
<div id="orologio">  
  <div id="secondi" class="lancetta"></div>  
  <div id="minuti" class="lancetta"></div>  
  <div id="ore" class="lancetta"></div>  
</div>
```

 - o L'elemento con id `orologio` consiste nel contenitore dell'orologio.
 - o Gli elementi appartenenti alla classe `lancetta` consistono nelle lancette dell'orologio. L'id distingue le varie lancette (`ore`, `minuti`, `secondi`).
 - o I numeri delle ore saranno aggiunti successivamente con Javascript.

- **Codice CSS:**

```
#orologio { // Quadrante dell'orologio  
  position: absolute;  
  top: 0px;  
  left: 0px;  
  border: 1px solid black;  
  
  // Lunghezza e larghezza del quadrante  
  width: 500px;  
  height: 500px;  
}
```

```
.lancetta { // Lancetta dell'orologio  
  position: absolute;  
  width: 0px;
```

*// La trasformazione ha origine di default dal centro dell'elemento.
Se non pongo questa proprietà la lancetta non si muoverà rispetto
al centro del quadrante ----->*

transform-origin: bottom left;

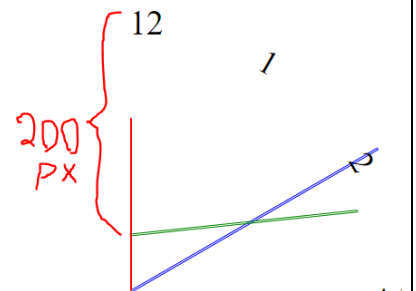
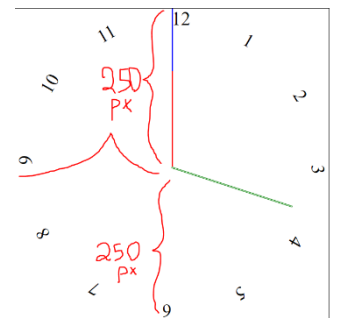
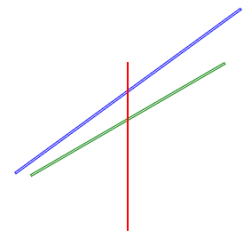
```
}
```

```
#secondi { // Proprietà specifiche della lancetta dei secondi  
  top: 0px;  
  left: 250px; // Metà della dimensione del quadrante  
  border: 1px solid blue;  
  height: 250px; // Metà della dimensione del quadrante  
}
```

```
#minuti { // Proprietà specifiche della lancetta dei minuti  
  // Minore della dimensione del quadrante. Ricollego la lancetta al  
  // centro del quadrante ponendo un valore top diverso da zero.  
  height: 200px;
```

```
  top: 50px;  
  left: 250px;  
  border: 1px solid green;  
}
```

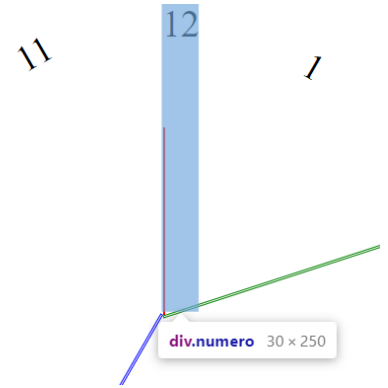
```
#ore { // Proprietà specifiche della lancetta delle ore  
  top: 100px;  
  left: 250px;  
  border: 1px solid red;  
  height: 150px; // Lancetta ancora più corta rispetto alle altre. Il valore di top aumenta  
}
```



Lancetta verde dei minuti con top a 0. Il problema si risolve se spingo la lancetta in basso di 50px.

```
.numero { // Numeri dell'orologio
  position: absolute;
  top: 0px;
  left: 250px;
  height: 250px;
  transform-origin: bottom left;
  font-size: 30px;
}
```

// L'elemento presenta le stesse proprietà della lancetta dei secondi (stessa altezza e posizione all'interno del quadrante). A differenza della lancetta presenta del contenuto: il numero. Sottolineo che le dimensioni degli elementi di classe numero non si fermano a quelle del contenuto. Per capire meglio aprite ispeziona elemento e verificate quale area coprono questi elementi.



- Codice Javascript:

```
<script type="text/javascript">
var secondi = document.getElementById("secondi");
var minuti = document.getElementById("minuti");
var ore = document.getElementById("ore");

setInterval(function() {
  var d = new Date();
  var s = d.getSeconds();
  var m = d.getMinutes();
  var h = d.getHours();
  deg_s = (360/60 * s);
  deg_m = (360/60 * m);
  deg_h = (360/24 * h);

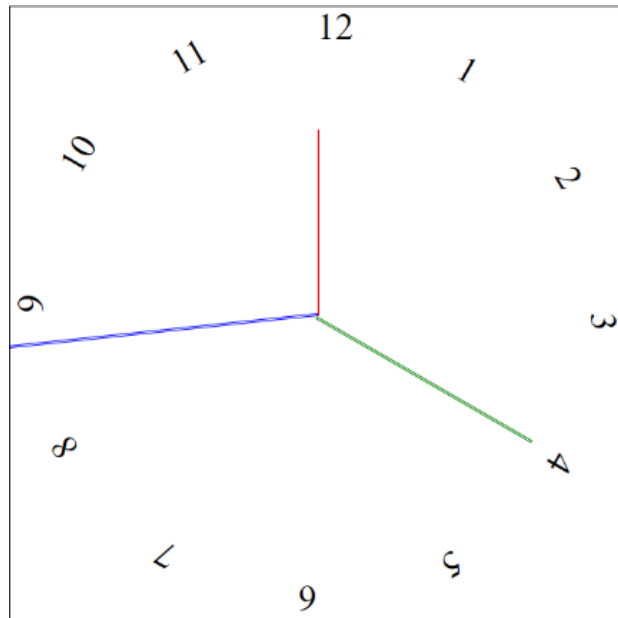
  secondi.style.transform = "rotate("+deg_s+"deg)";
  minuti.style.transform = "rotate("+deg_m+"deg)";
  ore.style.transform = "rotate("+deg_h+"deg)";
}, 1000);

for (i=1; i<=12; i++) {
  var num = document.createElement("div");
  num.setAttribute("class", "numero");
  num.innerHTML = i;
  document.body.appendChild(num);
  deg_n = (360/12) * i;
  num.style.transform = "rotate("+deg_n+"deg)";
}
</script>
```

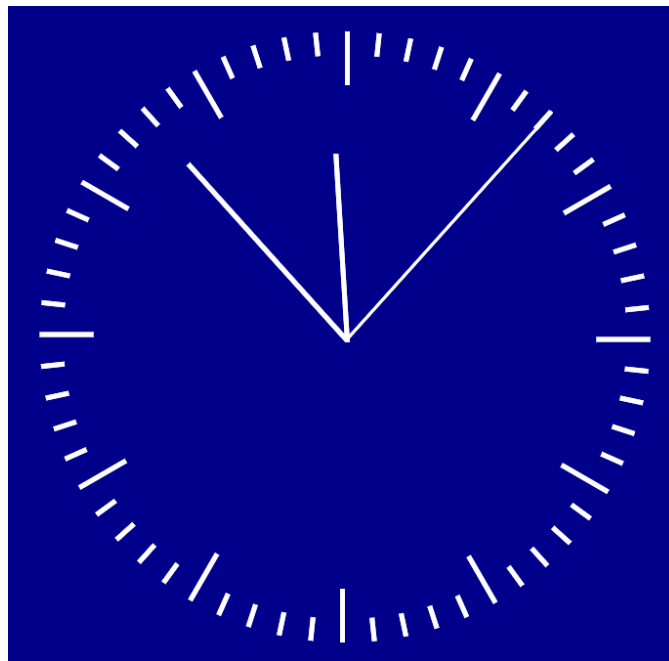
- o Recupero gli elementi da modificare con la `getElementById`
- o Con la `setInterval` definisco una funzione da eseguire ogni 1000 millisecondi, cioè ogni secondo:
 - Con l'oggetto built-in `Date` ottengo le informazioni sull'attuale istante temporale
 - Con la funzione `getSeconds()` ottengo i secondi
 - Con la funzione `getMinutes()` ottengo i minuti
 - Con la funzione `getHours()` ottengo l'ora
 - Calcolo l'inclinazione di ogni lancetta sfruttando i valori restituiti dalle funzioni citate. Nelle formule presenti mi immagino di avere una torta e di spartirla in parti uguali: è ovvio che se parlo di minuti e secondi questa torta sarà divisa in 60 parti uguali, mentre relativamente all'ora avrò una torta divisa in 12 parti uguali. Divido 360 gradi per il numero di fette della torta e moltiplico con il dato relativo trovato con una delle funzioni.
 - Aggiorno la proprietà CSS `transform` modificando `X.style.transform`.

- Con il for finale creo dodici elementi:
 - Creo un elemento `div`
 - Associa l'elemento alla classe `numero`
 - Imposto come contenuto dell'elemento un numero (`innerHTML`)
 - Pongo il numero all'interno del documento (`appendChild`)
 - Calcolo l'inclinazione del numero con lo stesso metodo di prima: avendo 12 ore dividerò la mia torta in 12 parti uguali.
 - Aggiorno la proprietà CSS `transform` modificando `X.style.transform`.

- **Output:**



- **Esercizio ulteriore:** realizzare l'orologio con uno stile differente. Io l'ho fatto ispirandomi al segnale orario RAI degli anni 90



- **Livello avanzato (proposta mia):** immagina l'orologio come il cruscotto di una macchina del tempo. La macchina è in funzione e le lancette si muovono a velocità elevatissima (**Suggerimento:** il movimento non è più legato alla data attuale).