P.PORTO

POLITÉCNICO
DO PORTO
ESMAD

COMPUTAÇÃO GRÁFICA
TSIW

Syllabus

- User interactions:
  - Event listeners
  - Event handlers
- Keyboard events
- Mouse events

Interaction

- So far, our animations had no user interaction
- In HTML, user interactions are detect by **events**:

  `click`   *`mouseover`*   *`mousein`*   *`mousemove`*   …

  `keypress`   *`keydown`*   *`keyup`*

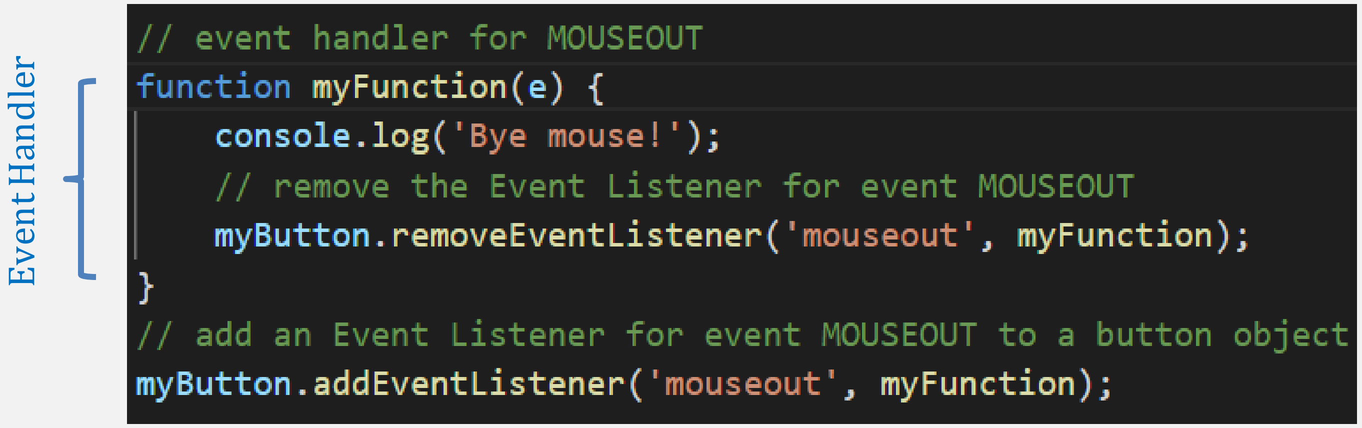
  `touchstart`   *`touchmove`*   …

  `wheel`

  …
- Events can be assigned to HTML elements in many ways; *event listeners* are used if more control over the events is required

Interaction

- Add and remove *Event listeners* to a button object



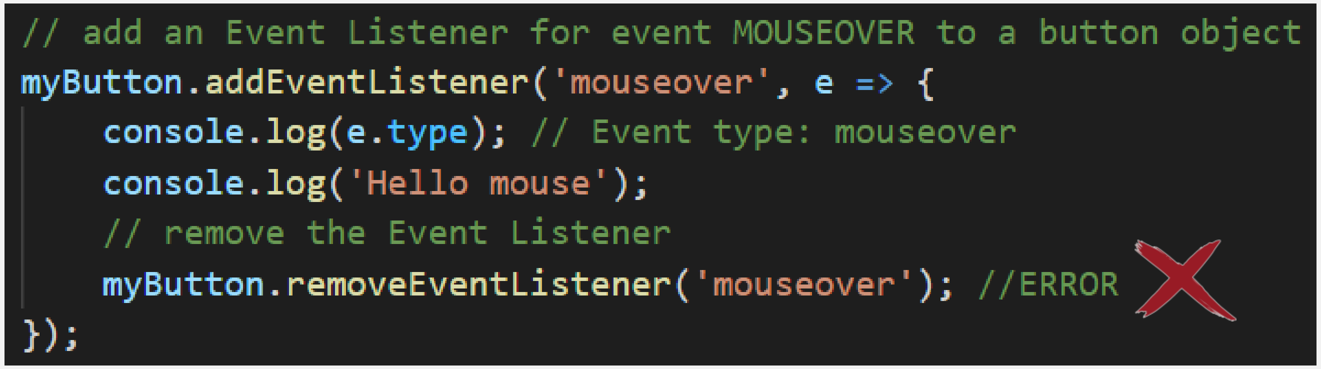
```
// event handler for MOUSEOUT
function myFunction(e) {
    console.log('Bye mouse!');
    // remove the Event Listener for event MOUSEOUT
    myButton.removeEventListener('mouseout', myFunction);
}
// add an Event Listener for event MOUSEOUT to a button object
myButton.addEventListener('mouseout', myFunction);
```

The event handler will be **called only once**, since the event listener is removed when the handler is executed

Interaction

- Add and remove *Event listeners* to a button object

Event Handler



```
// add an Event Listener for event MOUSEOVER to a button object
myButton.addEventListener('mouseover', e => {
    console.log(e.type); // Event type: mouseover
    console.log('Hello mouse');
    // remove the Event Listener
    myButton.removeEventListener('mouseover'); //ERROR
});
```

Removing the event listener throws an error, since it does not work when the event handler is an **anonymous function**

Interaction

KEYBOARD EVENTS

- When a key is pressed, the browser creates a keyboard event, detected by the HTML element with an active focus
- Since Canvas **does not have property focus**, all keyboard events must be redirected to objects window or document
- Keyboard events:
  - keydown: event sent when the key is depressed
  - keypress: event sent when the key is depressed (only if the key is not a modifier key: like arrows, TAB, CAPS LOCK, CTRL, ALT,…)
  - keyup: event sent when the user releases the key

Interaction

KEYBOARD EVENTS

- Useful properties/methods of keyboard events:
  - `key`: string representing the key

Pressed Key(s)	Event.key
Z	z
Shift + Z	Z

  - `repeat`: boolean set to `true` if the key is being held down such that it is automatically repeating
  - `altKey ctrlKey shiftKey`: booleans set to `true` if the keys ALT, CTRL or SHIFT are active when the event was triggered
  - `preventDefault()`: cancels the event (if possible); usefull to prevent default actions of some events, like scrollling up or down a page using the arrow keys

Interaction

KEYBOARD EVENTS – detect arrow keys

```
document.onkeydown = function (e) {
        switch (e.key) {
            case 'ArrowUp': /*UP arrow*/ doSomething; break;
            case 'ArrowDown': /*DOWN arrow*/ doSomething; break;
            case 'ArrowLeft': /*LEFT arrow*/ doSomething; break;
            case 'ArrowRight': /*RIGHT arrow*/ doSomething; break;
      }
      e.preventDefault();
};
```

Interaction

KEYBOARD EVENTS – handle multiple keys

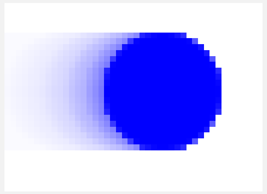
```
window.addEventListener('keydown',keyPressed);
window.addEventListener('keyup',keyReleased);

let keys = []; //array of entries for every key pressed

function keyPressed(e) {
        keys[e.key] = true; //store an entry
        if (e.key == something)
                doSomething;
        e.preventDefault();
}
function keyReleased(e) {
        keys[e.key] = false; //mark released keys
}
```

Interaction

KEYBOARD EVENTS

- An event response is usually **slower** than an animation framerate
- Animated objects should be updated on the animation loop, instead of the event handler

- Download and open file `Example_keyBoard.html` and compare the implemented animations



Interaction

MOUSE EVENTS

- Mouse events can be detected by the Canvas element
- Type of mouse events:
  - `click`: event sent when the mouse primary button is both pressed and released while the pointer is inside the HTML element
  - `dblclick`: event sent when the mouse primary button is clicked twice on a single HTML element within a very short span of time
  - `mousedown/mouseup`: events fired when the user presses/releases the mouse button while the pointer is inside the HTML element
  - `mouseenter/mouseleave`: events fired when the pointer enters/leaves the HTML element
  - `mousemove`: events fired when the pointer is moved inside the HTML element

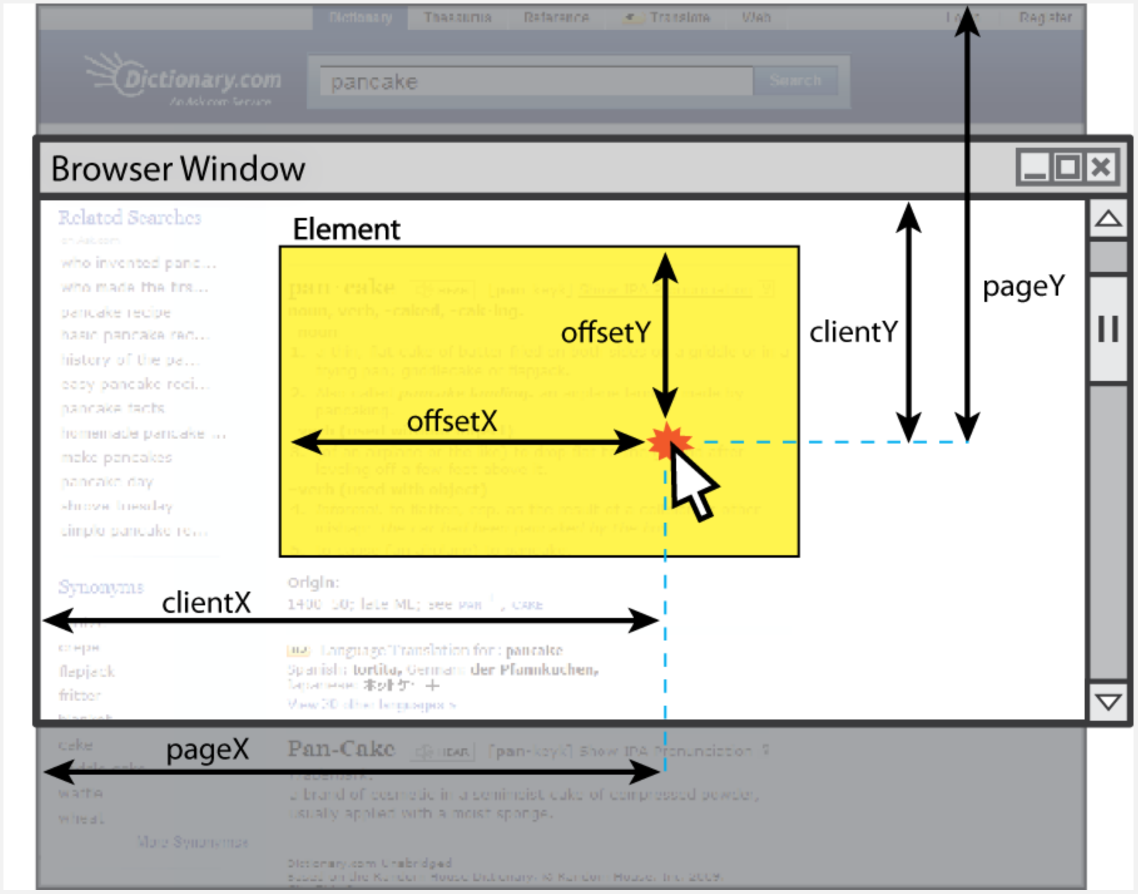
Interaction

Mouse EVENTS

- Useful properties/methods of mouse events:
  - `pageX/pageY`: mouse pointer coordinates in atual DOM
  - `clientX/clientY`: mouse pointer coordinates in browser window (equal to page coordinates, if document is not scrolled out)
  - `offsetX/offsetY`: mouse pointer coordinates in element
  - `button`: mouse button number that was pressed when the event fired (left button = 0; middle button = 1 – if any; right button = 2)

Interaction

Mouse EVENTS



Interaction

Mouse EVENTS – where is the cursor in Canvas coordinates?

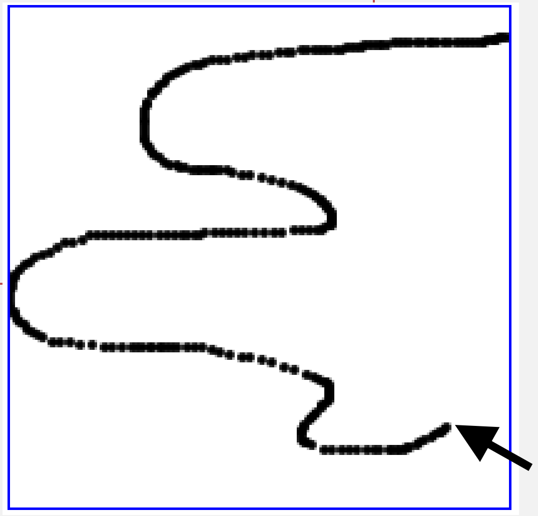
```
let x = 0; let y = 0;

// Add the event listener for mousemove
canvas.addEventListener('mousemove', e => {
     x = e.offsetX;
     y = e.offsetY;

     console.log('x: ' + x + ' y: ' + y);
});
```

Interaction

Mouse EVENTS – try this out!

Example_mouse.html

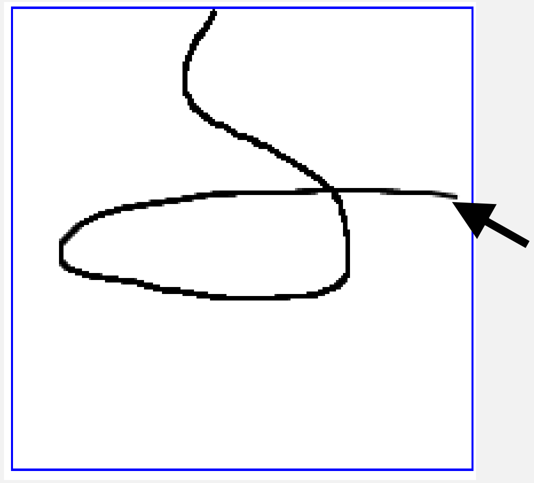
```
let x = 0; let y = 0;

// Add the event listener for mousemove
canvas.addEventListener('mousemove', e => {
     x = e.offsetX;
     y = e.offsetY;

     ctx.beginPath();
     ctx.arc(x, y, 2, 0 , Math.PI*2);
     ctx.fill();
});
```



Try yourself...

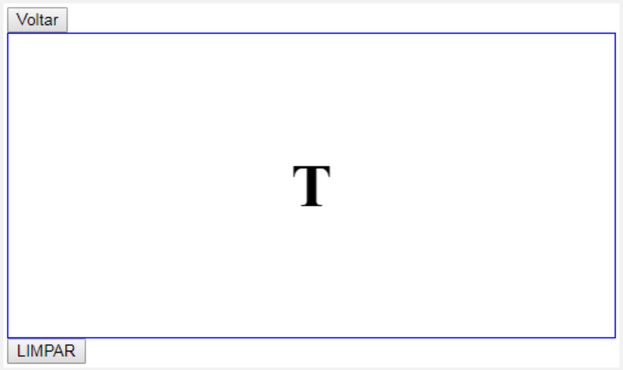
1. Alter previous example and make a continuous line drawing, using the mouse cursor. Each time the mouse pointer leaves the Canvas, a new line drawing is started (starts a new path).

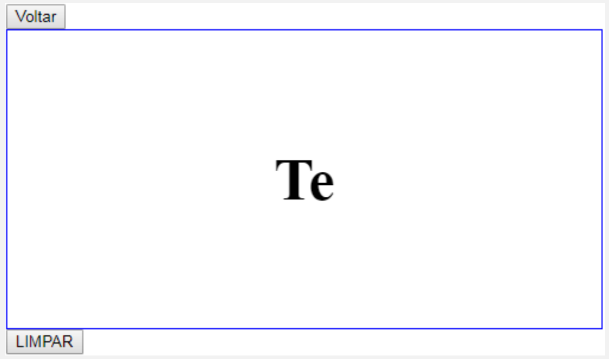
HINT: Instead of points, make a <u>path of connected lines</u>

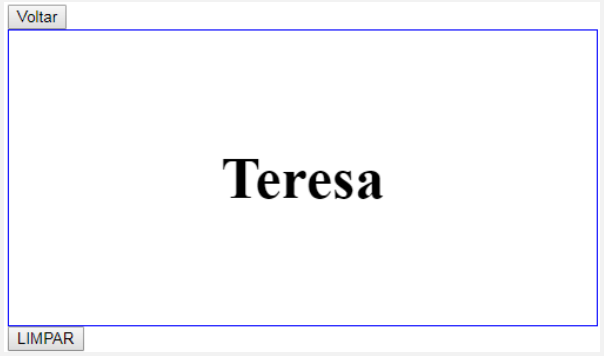


Try yourself...

2. Make the Canvas works like a simple notepad: each time user presses a key, the corresponding letter is added to a text, that is written at the Canvas center.
A button should clear the text and re-start the exercise.

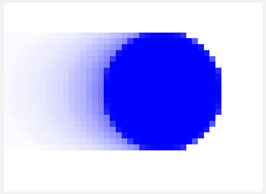






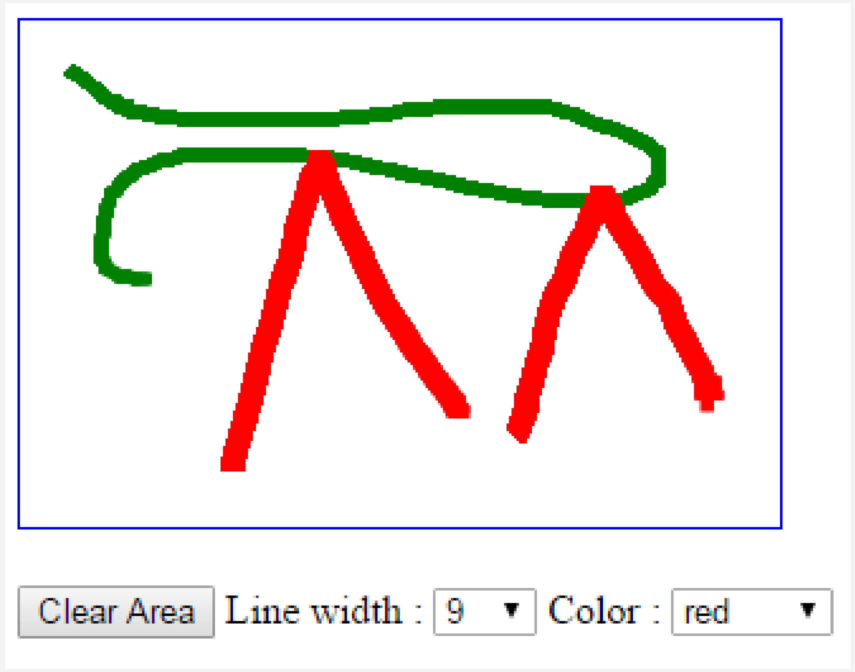
Try yourself...

3. Starting at the Canvas center, move a circle (1 pixel at each time) within the Canvas limits, using the four arrow keys.
   Allow for multiple keys detection, meaning that if user presses simultaneously arrows UP and LEFT, the circle should move diagonally.



Try yourself…

4. Alter exercise 1, so that the drawing only starts when user presses the mouse button. Add two drop-down lists to customize color and line width.
   - `mousedown`: starts drawing
   - `mousemove`: continues drawing
   - `mouseup`: ends drawing


Clear Area
Line width : 9
Color : red