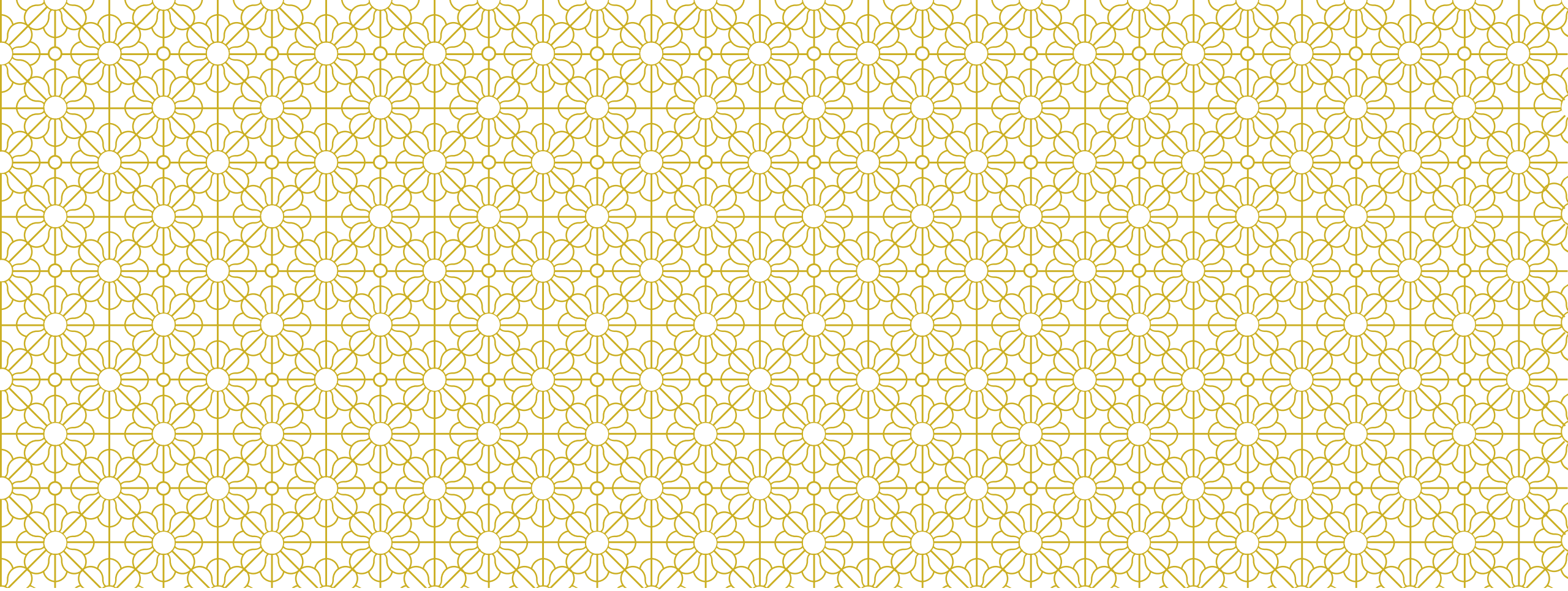


EVENT HANDLING

Prof. Andrew Sheehan

Metropolitan College
Boston University/CS Dept

HANDLING EVENTS: PURPOSE

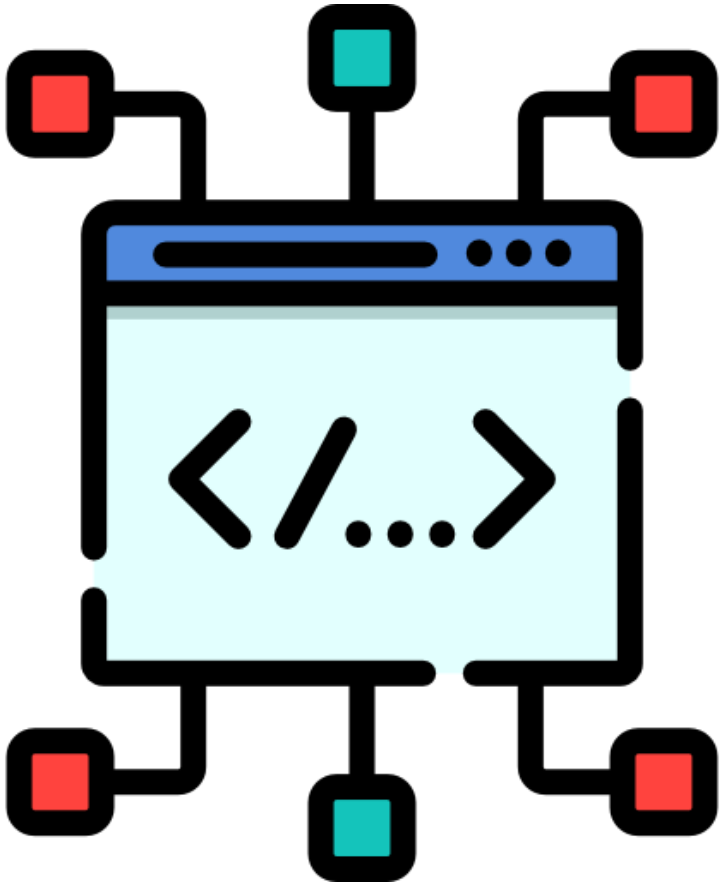
Something happened.

You want to know about.

ELEMENTS

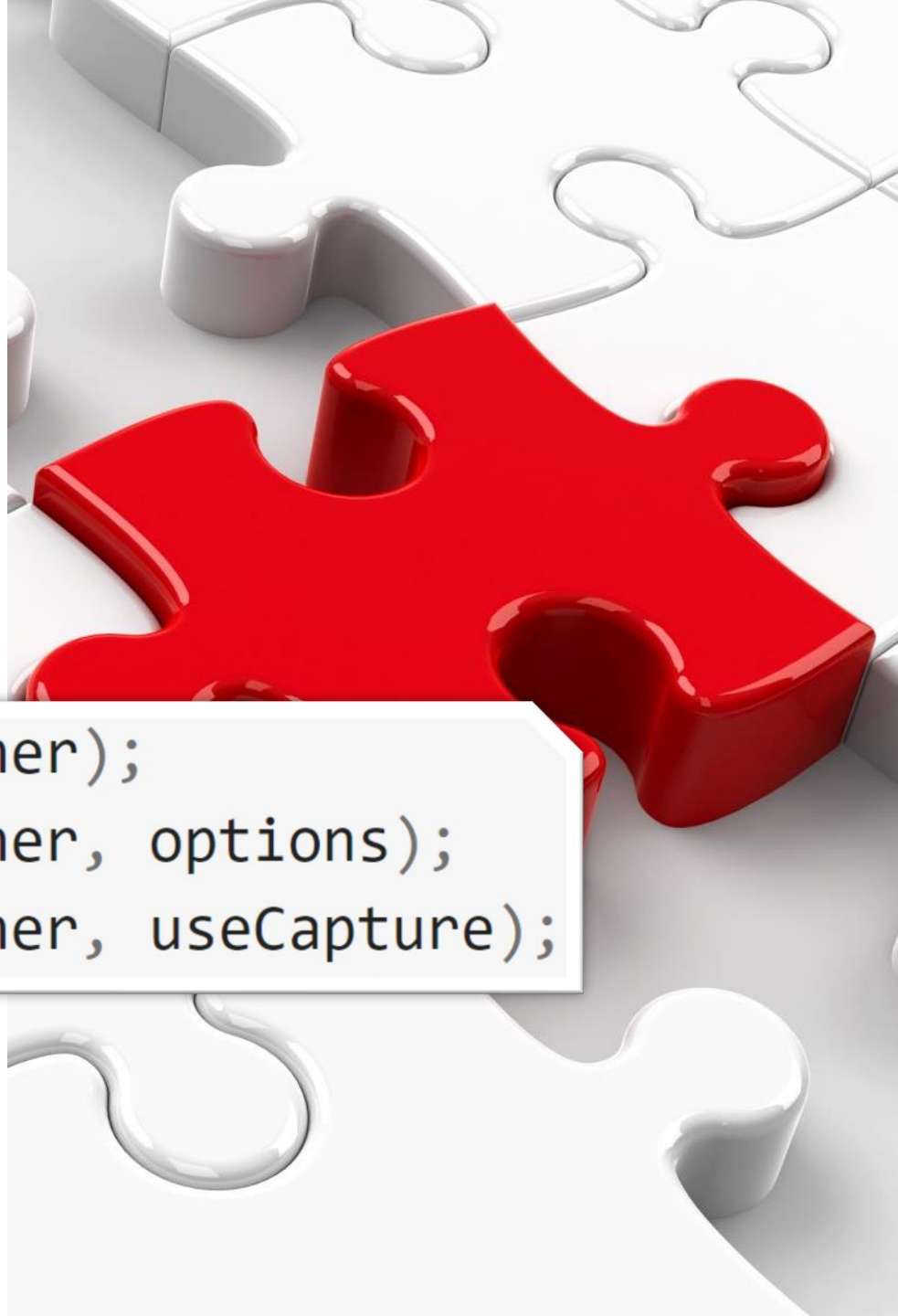
All elements can emit events.

Events can be dynamically added and removed



SYNTAX

This is the standard approach to bind events to an element.



```
target.addEventListener(type, listener);
target.addEventListener(type, listener, options);
target.addEventListener(type, listener, useCapture);
```

`type`

A case-sensitive string representing the event type to listen for.

`listener`

The object that receives a notification (an object that implements the `Event` interface) when an event of the specified type occurs. This must be an object implementing the `EventListener` interface, or a JavaScript function. See The event listener callback for details on the callback itself.

PARAMETERS: TYPE AND LISTENER

options **Optional**

An object that specifies characteristics about the event listener. The available options are:

capture

A boolean value indicating that events of this type will be dispatched to the registered `listener` *before being dispatched to any* `EventTarget` *beneath it in the DOM tree.*

once

A boolean value indicating that the `listener` *should be invoked at most once after being added. If* `true`*, the* `listener` *would be automatically removed when invoked.*

passive

A boolean value that, if `true`*, indicates that the function specified by* `listener` *will never call* `preventDefault()`*. If a passive listener does call* `preventDefault()`*, the user agent will do nothing other than generate a console warning. See Improving scrolling performance with passive listeners to learn more.*

signal

An `AbortSignal`*. The listener will be removed when the given* `AbortSignal` *object's* `abort()` *method is called.*

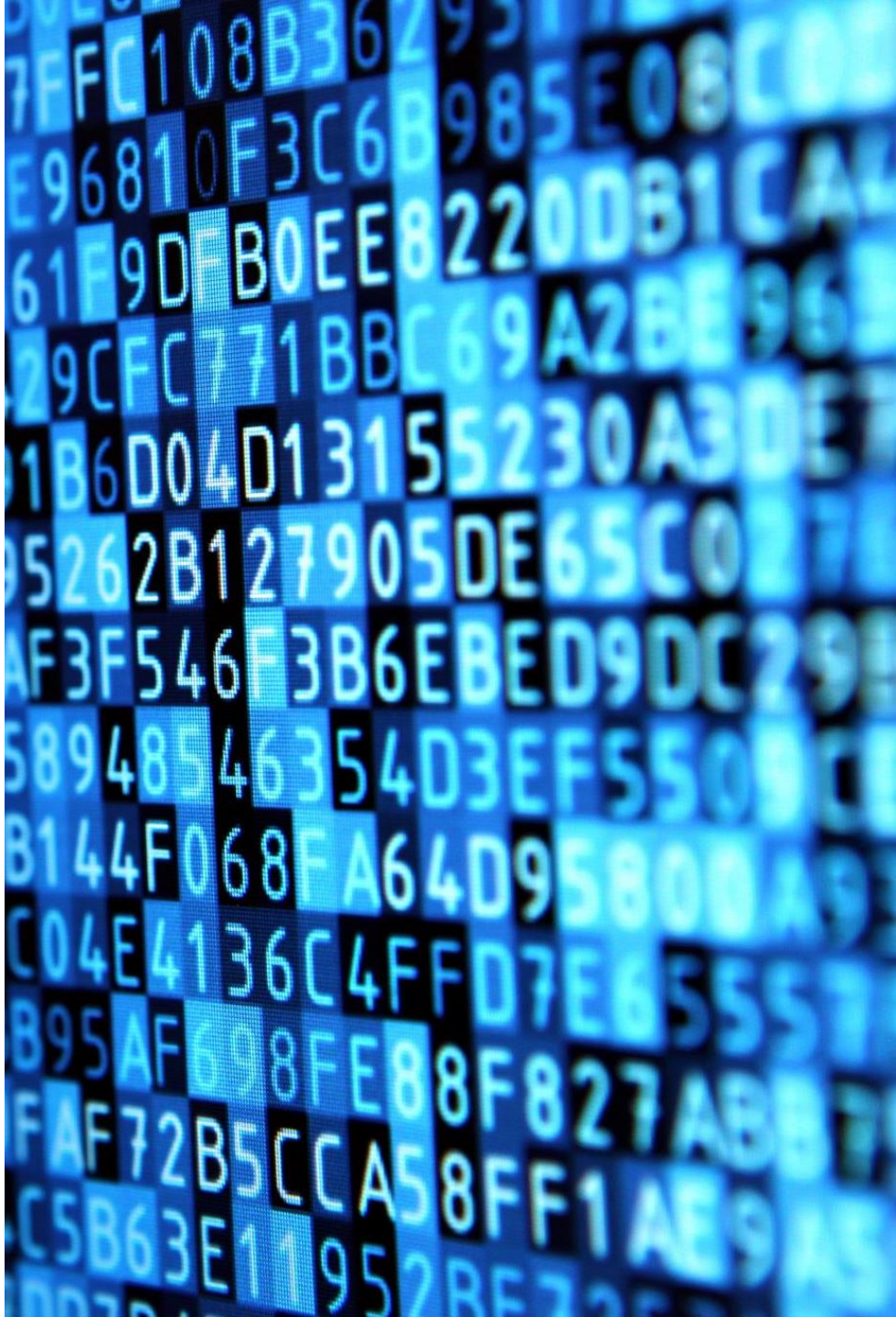
`useCapture` **Optional**

A boolean value indicating whether events of this type will be dispatched to the registered `listener` *before* being dispatched to any `EventTarget` beneath it in the DOM tree. Events that are bubbling upward through the tree will not trigger a listener designated to use capture. Event bubbling and capturing are two ways of propagating events that occur in an element that is nested within another element, when both elements have registered a handle for that event. The event propagation mode determines the order in which elements receive the event. See DOM Level 3 Events and JavaScript Event order for a detailed explanation. If not specified, `useCapture` defaults to `false`.

PARAMETERS: USECAPTURE

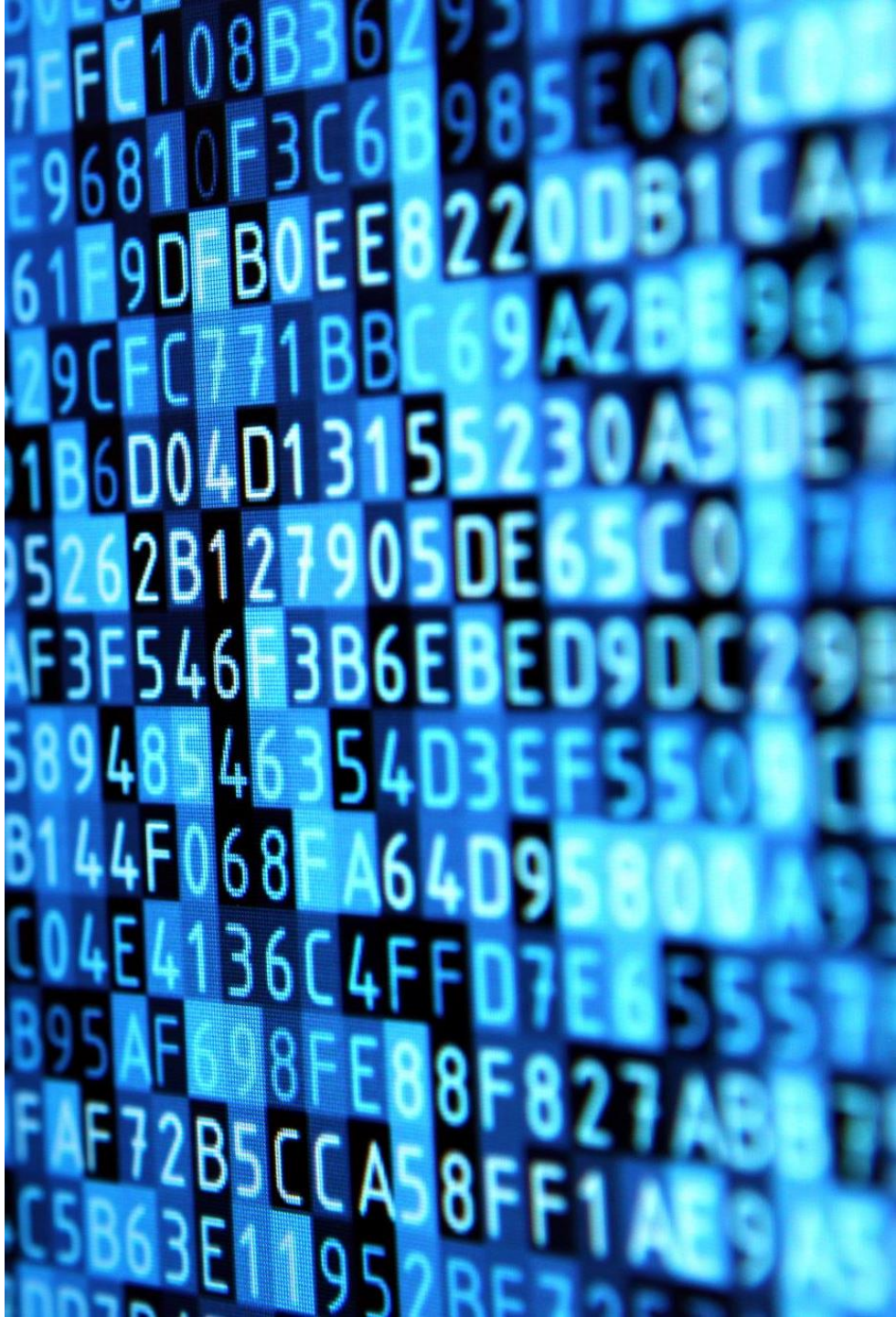
ADDING A LISTENER

```
const el = document.querySelector("#id");
el.addEventListener("click", handle);

function handle(event) {...}
```



REMOVING A LISTENER

```
const el =
    document.querySelector("#id");

el.removeEventListener("click", handle);
```



ES6 CLASS

```
class LipStick {
  register() {
      const tar = document.getElementById(“target”);
      tar.addEventListener(‘click’, (evt) => {
          this.run(evt) });
  }
}
```

STOP PROPAGATION

Prevents other objects from receiving the same event that was triggered.

The following example registers "mousedown" handlers on both a button and the paragraph around it. When clicked with the right mouse button, the handler for the button calls stopPropagation, which will prevent the handler on the paragraph from running. When the button is clicked with another mouse button, both handlers will run.

```
<p>A paragraph with a <button>button</button>.</p>
<script>
  let para = document.querySelector("p");
  let button = document.querySelector("button");
  para.addEventListener("mousedown", () => {
    console.log("Handler for paragraph.");
  });
  button.addEventListener("mousedown", event => {
    console.log("Handler for button.");
    if (event.button == 2) event.stopPropagation();
  });
</script>
```

`Event.bubbles` Read only

A boolean value indicating whether or not the event bubbles up through the DOM.

`Event.cancelable` Read only

A boolean value indicating whether the event is cancelable.

`Event.composed` Read only

A boolean indicating whether or not the event can bubble across the boundary between the shadow DOM and the regular DOM.

`Event.currentTarget` Read only

A reference to the currently registered target for the event. This is the object to which the event is currently slated to be sent. It's possible this has been changed along the way through *retargeting*.

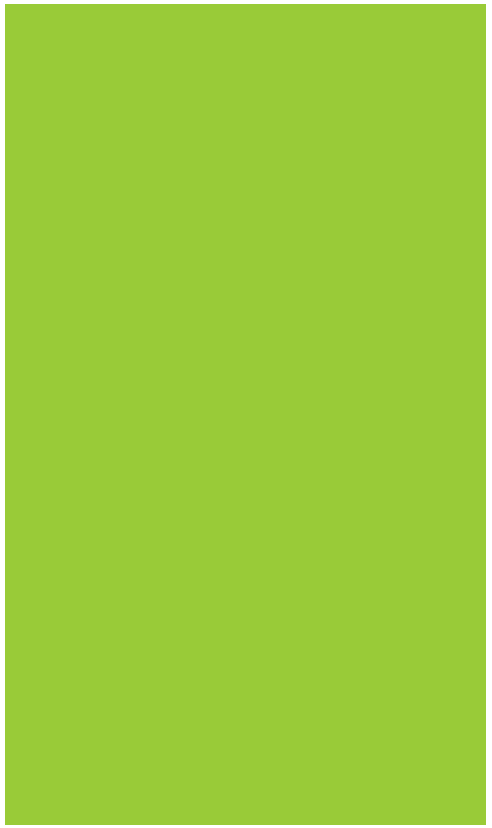
`Event.defaultPrevented` Read only

Indicates whether or not the call to `event.preventDefault()` canceled the event.

THE EVENT OBJECT

READ MORE: THE SHADOW DOM

developer.mozilla.org/en-US/docs/Web/API/Web_components/Using_shadow_DOM



THE EVENT OBJECT

`Event.eventPhase` Read only

Indicates which phase of the event flow is being processed. It is one of the following numbers: `NONE`, `CAPTURING_PHASE`, `AT_TARGET`, `BUBBLING_PHASE`.

`Event.isTrusted` Read only

Indicates whether or not the event was initiated by the browser (after a user click, for instance) or by a script (using an event creation method, for example).

`Event.target` Read only

A reference to the object to which the event was originally dispatched.

`Event.timeStamp` Read only

The time at which the event was created (in milliseconds). By specification, this value is time since epoch—but in reality, browsers' definitions vary. In addition, work is underway to change this to be a `DOMHighResTimeStamp` instead.

`Event.type` Read only

The case-insensitive name identifying the type of the event.

THE EVENT OBJECT

On Event

Another approach to event handling:

```
on[Event Name Goes Here]
= [your function];
```

```
function isDevelopment(evt) {
  return
    document.location.href.contains(“localhost”);
}

const target =
    document.querySelector(‘#target’);

target.onhover = isDevelopment
```

ON EVENT EXAMPLE