EVENT HANDLING:

SPONTANEOUS RIGHT ACTION

Main Point Preview

Event handlers take callback functions that are executed later when the event occurs.

Science of Consciousness: Callbacks are a form of memory for an action that is automatically executed when an event happens. When we act from deep levels of awareness, we are more likely to activate appropriate memories and reactions (event handlers).

Reacting to Events

- Examples of HTML events:
- When a user clicks the mouse
- When a web page has loaded
- When an image has been loaded
- When the mouse moves over an element
- When an input field is changed
- When an HTML form is submitted
- When a user strokes a key

HTML DOM Events

- HTML DOM events allow JavaScript to register different event handlers on elements in an HTML document.
  - `click, touch, load, drag, change, input, error, resize` — there are more…
- Events can be triggered on any part of a document, whether by a user’s interaction or by the browser.
- Events are normally used in combination with functions, and the function will not be executed before the event occurs (such as when a user clicks a button).

Listen for an event

- Two ways to listen for an event:
  - `element.onevent = function1;`
  - `element.addEventListener('event', myFunction1);`
- What's the difference between them?
  - The main difference is that `onclick` is just a property. If you write more than once, it will be overwritten.
  - `addEventListener()` on the other hand, can have multiple event handlers applied to the same element. It doesn't overwrite other present event handlers.

```
const btn1 = document.getElementById("btn1");
btn1.onclick = function () {
    console.log('Button 1 clicked.....1');
}
btn1.onclick = function () {
    console.log('Button 1 clicked.....2');
}
```

```
const btn2 = document.getElementById("btn2");
btn2.addEventListener('click', function () {
    console.log('Button 2 clicked.....1');
});
btn2.addEventListener('click', function () {
    console.log('Button 2 clicked.....2');
});
```

Mouse Events

Events that occur when the mouse interacts with the HTML document belongs to the MouseEvent Object.

`click`	user presses/releases mouse button on the element
`dblclick`	user presses/releases mouse button twice on the element
`mousedown`	user presses down mouse button on the element
`mouseup`	user releases mouse button on the element movement
`mouseover`	mouse cursor enters the element's box
`mouseout`	mouse cursor exits the element's box*
`mousemove`	mouse cursor moves around within the element's box

* or exits any descendent.

Page/window events

Events triggered for the window object (applies to the <body> tag):

Attribute	Description
onload	Fires after the page is finished loading
onresize	Fires when the browser window is resized
onunload	Fires once a page has unloaded (or the browser window has been closed)

Form events

`submit`	form is being submitted
`Reset`	form is being reset
`change`	the text or state of a form control has changed

Document

`DOMContentLoaded` – when the HTML is loaded and processed, DOM is fully built.

Recall `window.onload` event

- We want to attach our event handlers right after the page is done loading (Why?)
  - There is a global **event** called `window.onload` event that occurs at that moment

```
// this will run once the page has finished loading
function functionName() {
        element.event = functionName;
        element.event = functionName;
        ...
}

window.onload = functionName; // DOM version
```

```
window.onload = function() {
    alert('hi');
}
```

Main Point

Event handlers take callback functions that are executed later when the event occurs.

Science of Consciousness: Callbacks are a form of memory for an action that is automatically executed when an event happens. When we act from deep levels of awareness we are more likely to activate appropriate memories and reactions (event handlers).

Main Point Preview

JavaScript code runs inside of an object and the ‘this’ keyword refers to that object. Event handlers that are attached unobtrusively are bound to that element and inside the handler ‘this’ references the bound DOM element. Usage of ‘this’ in event handlers is a common JavaScript programming idiom that enables handlers to be reused across different kinds of elements.

Science of Consciousness: We can think of the TM Technique as an event handler that gives the result of transcending and can be used by any person (element).

Event handler binding

Event handlers attached unobtrusively are bound to the element. Inside the handler, that element becomes `this` (rather than the window)

```
function sayHi() {
 // sayHi knows what object it was called on
 this.value = "sayHi" + this.id;
}
document.getElementById("submitBn").addEventListener("click", sayHi);
```

```
<div class="exampleoutput">
 <input id="textbox" />
 <input type="submit" id="submitBtn" value="Save">
</div>
```

Main Point

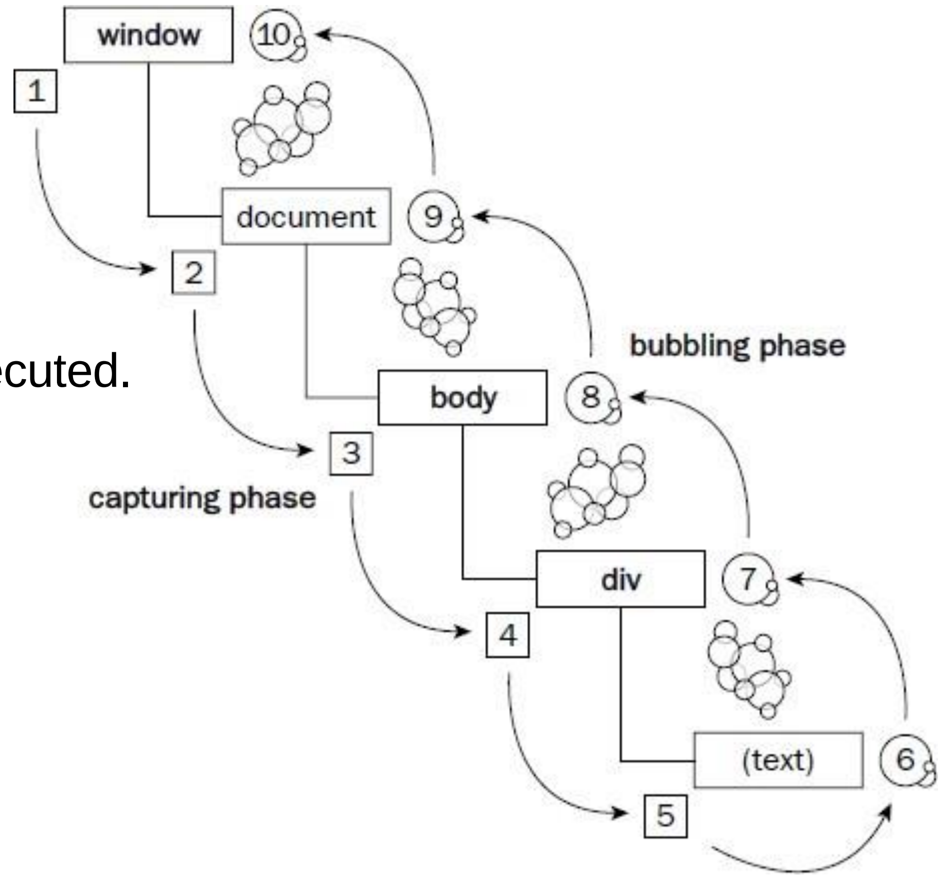
JavaScript code runs inside of an object and the ‘this’ keyword refers to that object. Event handlers that are attached unobtrusively are bound to that element and inside the handler ‘this’ references the bound DOM element. Usage of ‘this’ in event handlers is a common JavaScript programming idiom that enables handlers to be reused across different kinds of elements.

Science of Consciousness: We can think of the TM Technique as an event handler that gives the result of transcending and can be used by any person (element).

Event Bubbling

```
<div><p> Events are <em>crazy</em></p></div>
```

- Clicking the `em` is actually a click on every element in this page.
- Therefore it was decided that all of the handlers should be executed.
- The events **bubble from the bottom of the DOM tree to the top.**
- The opposite model (top to bottom) is called **capturing and is not widely used.**



addEventListener(*event*, *function*, *useCapture*);

In ***bubbling*** the inner most element's event is handled first and then the outer: the <p> element's click event is handled first, then the <div> element's click event.

In ***capturing*** the outer most element's event is handled first and then the inner: the <div> element's click event will be handled first, then the <p> element's click event.

stopPropagation

The stopPropagation() method prevents further propagation of the current event in the capturing and bubbling phases.

It does not, however, prevent any default behaviors from occurring; for instance, clicks on links are still processed.

It also does not prevent propagation to other event-handlers of the current element.

```
function setSpan(evt) {
    evt.stopPropagation();
    console.log("span");
}
```

stopImmediatePropagation

- stopImmediatePropagation will prevent any parent handlers **and also** any **other** handlers of the current element.

```
function setSpan(evt) {
    evt.stopImmediatePropagation();
    console.log("span");
}
```

preventDefault() Event Method

The preventDefault() method cancels the event if it is cancelable, meaning that the default action that belongs to the event will not occur.

For example, this can be useful when:

- Clicking on a "Submit" button, prevent it from submitting a form
- Clicking on a link, prevent the link from following the URL

Note: Not all events are cancelable. Use the cancelable property to find out if an event is cancelable.

Note: The preventDefault() method does not prevent further propagation of an event through the DOM.

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

Spontaneous Right Action

1. Event handling is a fundamental aspect of JavaScript programming.

2. Some subtle aspects of JavaScript event handlers include the use of event arguments passed to event handlers depending on the type of element, the use of the keyword ‘this’ that can refer to different objects since functions are first class in JavaScript, and the need to sometimes control event propagation.

---

3. Transcendental consciousness. The home of all the laws of nature

4. Impulses within the transcendental field: Thoughts arising from this level will be able to spontaneously respond with right actions to events because they are supported by all the laws of nature.

5. Wholeness moving within itself: In unity consciousness one appreciates the interconnectedness of everything at the underlying basis of the unified field.