CSIS 3380
Advanced Web Programming with JavaScript & AJAX

Event Handling

Week 5

Review

- Array
- Loop
- DOM

Events

- Events are actions or occurrences that happen in the system you are programming, which the system tells you about so you can respond to them in some way if desired
- For example, if the user clicks a button on a webpage, you might want to respond to that action by displaying an information box
- events are fired inside the browser window, and tend to be attached to a specific item that resides in it

Examples

- Examples of events on a webpage:
  - The user clicking the mouse over a certain element or hovering the cursor over a certain element.
  - The user pressing a key on the keyboard.
  - The user resizing or closing the browser window.
  - A web page finishing loading.
  - A form being submitted.
  - A video being played, or paused, or finishing play.

Event Handler

- Each available event has an **event handler**, which is a block of code (usually a user-defined JavaScript function) that will be run when the event fires
- When such a block of code is defined to be run in response to an event firing, we say we are **registering an event handler**
- Comprehensive list of events in the browser:
  - https://developer.mozilla.org/en-US/docs/Web/Events

Commonly used events

- click
- dblclick
- focus
- blur
- select
- keydown/keyup
- keypress
- mouseover
- mousedown/mouseup

Ways of adding Event Handler

- Ways of adding Event Handlers to a webpage:
  - Event handler properties
  - Inline event handlers (not recommended)
  - Via eventListener
- Let’s try using the 3 different ways to apply the following to the click event of a button

```
function toBlue() {
    document.body.style.backgroundColor = 'blue';
}
```

Example

```
<!DOCTYPE html>
<html>
<body>
<div>
  <button id="property">Via event handler property</button>
  <button id="inline" onclick="toBlue()">Via inline event handler</button>
  <button id="listener">Via event listener</button>
</div>
<script>
  function toBlue() {
    document.body.style.backgroundColor = 'blue';
  }
  var propertyBtn = document.getElementById('property');
  var listenerBtn = document.getElementById('listener');

  propertyBtn.onclick = toBlue;
  listenerBtn.addEventListener('click', toBlue);
</script>
</body>
</html>
```

Continue example

- What if we want to change the behavior to change the **button that was clicked** to blue instead of the **background** to blue?

```
function toBlue(event) {
   event.target.style.backgroundColor = 'blue';
 }
```

- Notice that using the inline event handler does not work in this case
- The event handler function takes in an argument by default. This argument represents the **event that triggers this action** (the click event in this case)
- This event argument is an implementation of the Event interface https://developer.mozilla.org/en-US/docs/Web/API/Event
- **event.target** would give you the the object that trigger this event

Continue example

- Alternatively, you can change the event handler to the following

```
function toBlue(event) {
   this.style.backgroundColor = 'blue';
 }
```

- The **this** in the event handler will reference the HTML element object that the event handler is attached to

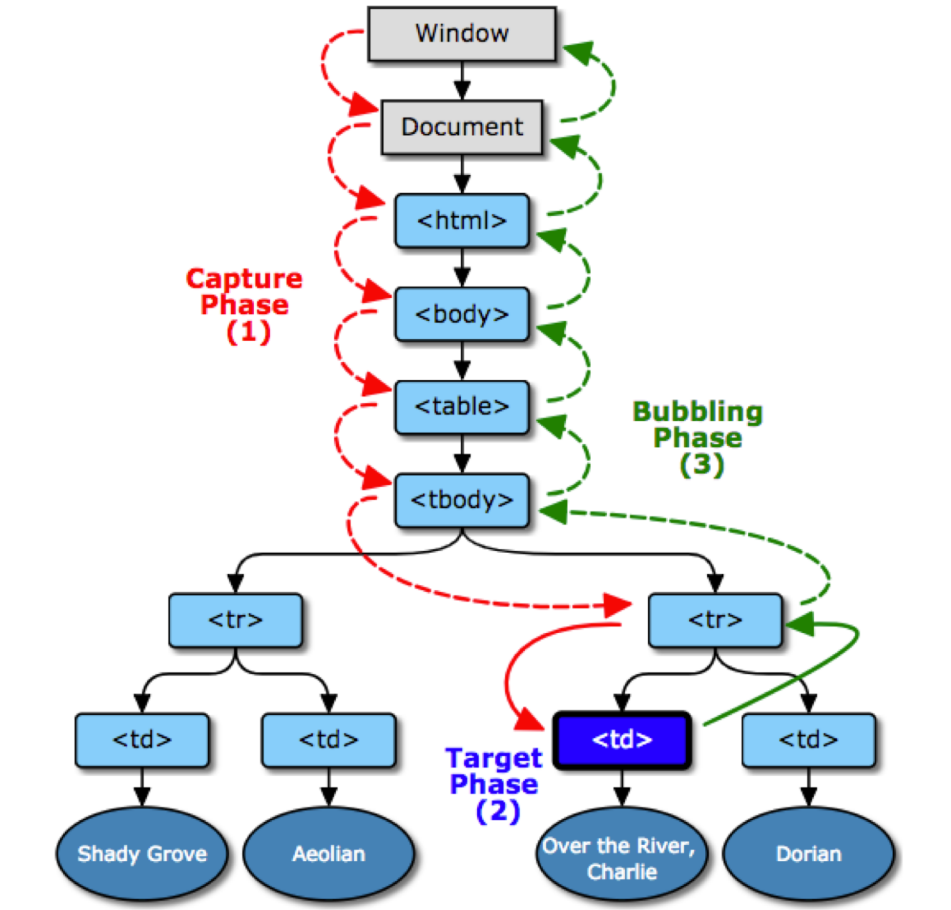
Preventing default behavior

- Sometimes, you'll come across a situation where you want to stop an event doing what it does by default
- The Event’s interface preventDefault() function can be used to prevent the default action of the element
- Refer to **preventDefault.html** on how we can perform a validation check before a form submission
- Note nowadays form validation can be done natively https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation

Bubbling and capturing

- When an event is fired on an element that has parent elements, modern browsers run two different phases — the **capturing** phase and the **bubbling** phase

Source: https://www.w3.org/TR/DOM-Level-3-Events/#event-flow



Graphical representation of an event dispatched in a DOM tree using the DOM event flow

Bubbling and capturing

- In the **capturing** phase:
  - The browser checks to see if the element's outer-most ancestor (<html>) has the event handler registered on it and runs it if so
  - Then it moves on to the next element inside <html> and does the same thing, then the next one, and so on until it reaches the element that was actually clicked on
- In the **bubbling** phase, the exact opposite occurs:
  - The browser checks to see if the element that was actually clicked on has an event handler registered on it in the bubbling phase, and runs it if so
  - Then it moves on to the next immediate ancestor element and does the same thing, then the next one, and so on until it reaches the <html> element
- In modern browsers, by default, all event handlers are registered in the bubbling phase
- the bubbling phase is all you need to concern about for this course

Bubbling and capturing

- Observe the console log as the click event bubbled up to the top

```
<!DOCTYPE html>
<html>
<body>
 <div>
  <button>Click Me</button>
 </div>
 <script>
  var elements = []
  elements.push(document.querySelector('html'));
  elements.push(document.querySelector('body'));
  elements.push(document.querySelector('div'));
  elements.push(document.querySelector('button'));
  for(var i=0; i<elements.length; i++) {
   elements[i].addEventListener('click', logClick)
  }
  function logClick() {
   console.log(this);
  }
 </script>
</body>
</html>
```

stopPropagation()

- The Event object has a function available on it called **stopPropagation()**, which when invoked on a handler's event object makes it so that handler is run, but the event doesn't bubble any further up the chain, so no more handlers will be run
- Observe how the change in the previous example if you change the event handler

```
function logClick(event) {
    if(this.tagName === 'BUTTON') {
      event.stopPropagation();
    }
    console.log(this);
}
```

Lab

- Create a page with an <input> that will disallow the letter ”q”, “w” and “x” to be typed into
- Whenever the letter ”q”, “w” and “x” is typed into the text field, an error message should display to inform the user that the entered letter is not allowed
- Hint: use the keypress event and preventDefault() function to implement this