Events and Interaction

CSC309

Mark Kazakevich

HTML Events

- All events occur on HTML elements in the browser

- JavaScript is used to define the **action** that needs to be taken when an event occurs

When **[HTML Event]** , do **[JS Action]**

Adding action listeners in HTML

- To show that actions originate from HTML elements, we can put attributes inside of elements

```
<div onclick="alert('Clicked!')">...</div>
```

Setting up listeners in JS

- An **event listener** in JS programmatically sets an event attribute on an HTML element

- Select an `element` in JS, and then:

```
element.addEventListener(event,
        functionToExecuteWhenEventOccurs)
```

Setting up listeners in JS

```
element.addEventListener(event,
        functionToExecuteWhenEventOccurs)
```

A ‘**callback**’ function

```
element.addEventListener(event, callback)
```

Callback functions

- A callback is a function that is designated to be ‘called back’ at an appropriate time
- In the case of events, it will be ‘called back’ when the event occurs
- Can be an anonymous function, or a function defined outside of the event listener

Callback functions

```
button.addEventListener(‘click’, function() {
      alert(‘Clicked’)
    });
```

Or

```
function alertClick() { alert(‘Clicked’) }

button.addEventListener(‘click’, alertClick);
```

Event Objects

- All events that occur create a JS Object with information about that event:
  - Event.target - event origin element
  - Event.type - type of event

- Passed to the callback function as argument

```
function myCallback(e) {
    // figure out where e came from
    // execute proper code
}
```

Common Events

- **onchange** An HTML element has been changed
- **onclick** The user clicks an HTML element
- **onmouseover** The user moves the mouse over an HTML element
- **onmouseout** The user moves the mouse away from an HTML element
- **onkeydown** The user pushes a keyboard key
- **onkeyup** The user releases a keyboard key
- **onload** The browser has finished loading the page

Events demo

Non-blocking JS

- For the most part, we've dealt with **blocking** code
  - Code that runs one instruction after another, and makes next instructions wait (block)

- **Non-blocking** code allows JS to continue executing instructions while we wait for some blocking code to complete

Non-blocking JS

- What's some non-blocking code we've seen so far?

```
setTimeout(function()
      { console.log('2 seconds') },
2000)

// We can execute instructions after
without waiting for setTimeout to finish
```

Non-blocking JS

- What's some non-blocking code we've seen so far?

A **callback** - only runs once 2 seconds pass

```
setTimeout(function()
      { console.log('2 seconds') },
2000)

  // We can execute instructions after
without waiting for setTimeout to finish
```

‘Asynchronous’

- This feature of JS that allows for non-blocking code is an example of **Asynchronous** programming
  - Takes some time to get used to when coming from a **synchronous** language (one-line-after-another)

- How does JS handle this under the hood?

JavaScript Event Loop

JS runtime engine

- JavaScript must be ‘compiled’ and interpreted
  - Needs a **‘runtime’** environment

- E.g., In Chrome, the JS runtime is the **V8 Engine**

- Javascript is an **event-driven** language
  - (as we’ve seen with the number of user interactions)
  - How does it keep track of all of these events?
    - **Event Loop**

How many threads?

- Important: **JavaScript is single-threaded!**
  - It still only runs one thing at a time
    - Doesn't seem very asynchronous...
  - So how does it do all of its asynchronous stuff?

- The **Event Loop**
  - A way of scheduling events one after the other
  - Often with the help of the platform the engine is running on (i.e., the browser)

JS Event Loop

- Interesting note:
  - setTimeout and other non-blocking functions aren't built in to the V8 runtime!
  - They live in the platform JS is running on
    - Chrome contains the instructions for setTimeout

- Let's see how this works when we run it in JS
  - Using a neat web app called Loupe