# WEB230: JavaScript 1

Module 6: Handling Events

## Events

- Events are interactions with our page
- Often initiated by the user
- We can't predict when they will happen

## Event Handlers

- JavaScript code that runs when an event occurs
- Written as a function that can be passed to a method

## Events and DOM Nodes

- Every DOM Element node can have events associated with it
- use `addEventListener()`
- first argument is the event name such as `click`
- second argument is a callback function (event handler)

```
const button = document.querySelector('button');
button.addEventListener('click', function() {
  console.log('Button clicked.');
});
```

### Deleting an Event Handler

- Use a named callback function
- This gives us a reference to the function that we can pass to `removeEventListener()`

```
const button = document.querySelector('button');
function once() {
  console.log('Done.');
  button.removeEventListener('click', once);
}
button.addEventListener('click', once);
```

## Event Objects

- callback functions can accept a parameter called the event object
- this object has information about the event
    - for example, which element was clicked on
- properties and methods vary depending on the type of event
- more later

## Propagation

- if an event occurs on a child element it will trigger the event handler on the parent element
- if both have handlers the more specific one runs first
- `stopPropogation()` method on the event object can stop this

### `target` Property

- most events have a `target` property
- this is the element that the event occurred on
- often used to delegate event handling to parent element

## Default Actions

- some element have default actions
    - such as a form being submitted to a server or a link being followed
- the event handler runs before the default action
- `preventDefault()` method can stop the default action

## Keyboard Events

- `keydown`, `keyup`, and `keypress` events
- `keydown` and `keypress` will repeat if held
- `event.key` holds a string with the value that the key would type
- `shiftKey`, `ctrlKey`, `altKey`, and `metaKey` properties for modifier keys
- event occurs on element that has focus (or document.body)
- if you want to capture all keystrokes, use `window.addEventListener()`
    - `window.` is optional since it is the global object

### Key Event Properties

- `event.key` (String) The key value of the key represented by the event. If the value has a printed representation, this attribute's value is the same as the char attribute. Otherwise, it describes the key.
- `event.code` (String) Holds a string that identifies the physical key being pressed. The value is not affected by the current keyboard layout or modifier state, so a particular key will always return the same value.

```
document.body.addEventListener('keydown', function(event) {
  console.log('Key pressed:', event.key);
});
```

- `event.repeat` (Boolean) true if the key is being held down such that it is automatically repeating
    - can be used to avoid repeatedly running the event handler

```
document.body.addEventListener('keydown', function(event) {
  if (event.repeat) return;
  console.log('Key pressed:', event.key);
});
```

### Key Event Order

1. When the key is first depressed, the keydown event is sent.
2. If the key is not a modifier key, the keypress event is sent.
3. When the user releases the key, the keyup event is sent.

- Note: Shift and other modifier keys do not activate keypress

## Mouse Clicks

- mousedown, mouseup, click, and dblclick events
- pageX and pageY properties give exact location
- event.button takes into account user customization
    - 0: Main button pressed, usually the left button or the un-initialized state
    - 1: Auxiliary button pressed, usually the wheel button or the middle button (if present)
    - 2: Secondary button pressed, usually the right button
    - 3: Fourth button, typically the Browser Back button
    - 4: Fifth button, typically the Browser Forward button

### Mouse Button Event Order

1. mousedown
2. mouseup
3. click
4. dblclick - if applicable
    - dblclick will repeat the previous three twice

## Mouse Motion

- mousemove event
- mouseover or mouseout event equivalent to CSS :hover

## Touch Events

- there are specific events for touch interfaces
    - touchstart
    - touchmove
    - touchend
- touch interfaces also produce click events

## Scroll Events

- scroll event when page scrolls
- fired every time the page is scrolled
- pageYOffset and pageXOffset for scroll position

## Focus Events

- focus and blur
- when an element is selected it has focus
- when it looses focus a blur event is fired

- often used with forms

## Load Event

- `load` event fires when the window finishes loading
- often used to schedule initialization actions that require the DOM
- element that load external files, such as images, also have a load event
- `beforeunload` fires when navigating away from a page

## Script Execution Timeline

- no two scripts can run at the same time
- each script (or function) will wait for others to finish
- web workers provide a way to do something while other things run

## Setting Timers

- `setTimeout` to run a function after an amount of time
- schedules a function to be called in a specified amount of time
- `clearTimeout` can be used to cancel it
- `setInterval` and `clearInterval` is similar but repeats every specified time interval

## Debouncing

- some events may fire many times in a row
    - eg. `mousemove` and `scroll`
- can use `setTimeout` to prevent this
- for key events use the `event.repeat` to detect repeating keys

## Summary

- event handlers make it possible to detect and react to external events
- each event has a type - eg. 'click'
- events *propagate* to their parent elements
    - `stopPropagation()`
- some elements have default actions
    - `event.preventDefault()`
- only one piece of JavaScript can run at once