All events inherit from the Event interface

Recap

Most DOM elements can be set up to accept (aka: "listen" for) these events, and execute code in response to the event happening (aka: it can "handle" the event).

Event-handlers are usually connected (developers refer to this as being "attached") to various DOM element (such as <p>, <button>, , etc.) using element.addEventListener(), and this generally replaces using the old HTML methods we've looked at (the inline event listener and inline properties).

It can get complicated

Right now, this may all seem very simple to you.

But when there are many nested elements, each with its own handler(s), event processing can become very complicated. Take the instance where a parent element receives the same event as the child because technically the event occurs on both. In this scenario, the processing order of the event(s) depends on the event bubbling settings of each handler triggered (something we'll get onto later).

Back up, what are Events again?

All events are represented by an object which is based on the **Event interface**. FYI, An interface is just a description of the actions that an object can do. Don't let the word intimidate or scare you.
But it doesn't stop here.

Some events (such as *keydown*) may have additional custom fields and/or functions that allow us to get additional information about what happened. Events can represent everything from basic user interactions to automated notifications of things happening in the rendering model.

Don't panic

Learning about events is a massive topic, and that's why this course is quite lengthy. We are going to look at a lot of events in this course, and will constantly look at examples of how we can apply what we've learnt to real web applications.

I hope you are enjoying this course so far.

Keep going, and see you in the next lecture!