EVENTS AND EVENT HANDLERS

IT'S HANDLED

EVENTS

Most JavaScript is written to react to events.

The web browser keeps track of how the user interacts with the web page through what are known as events.

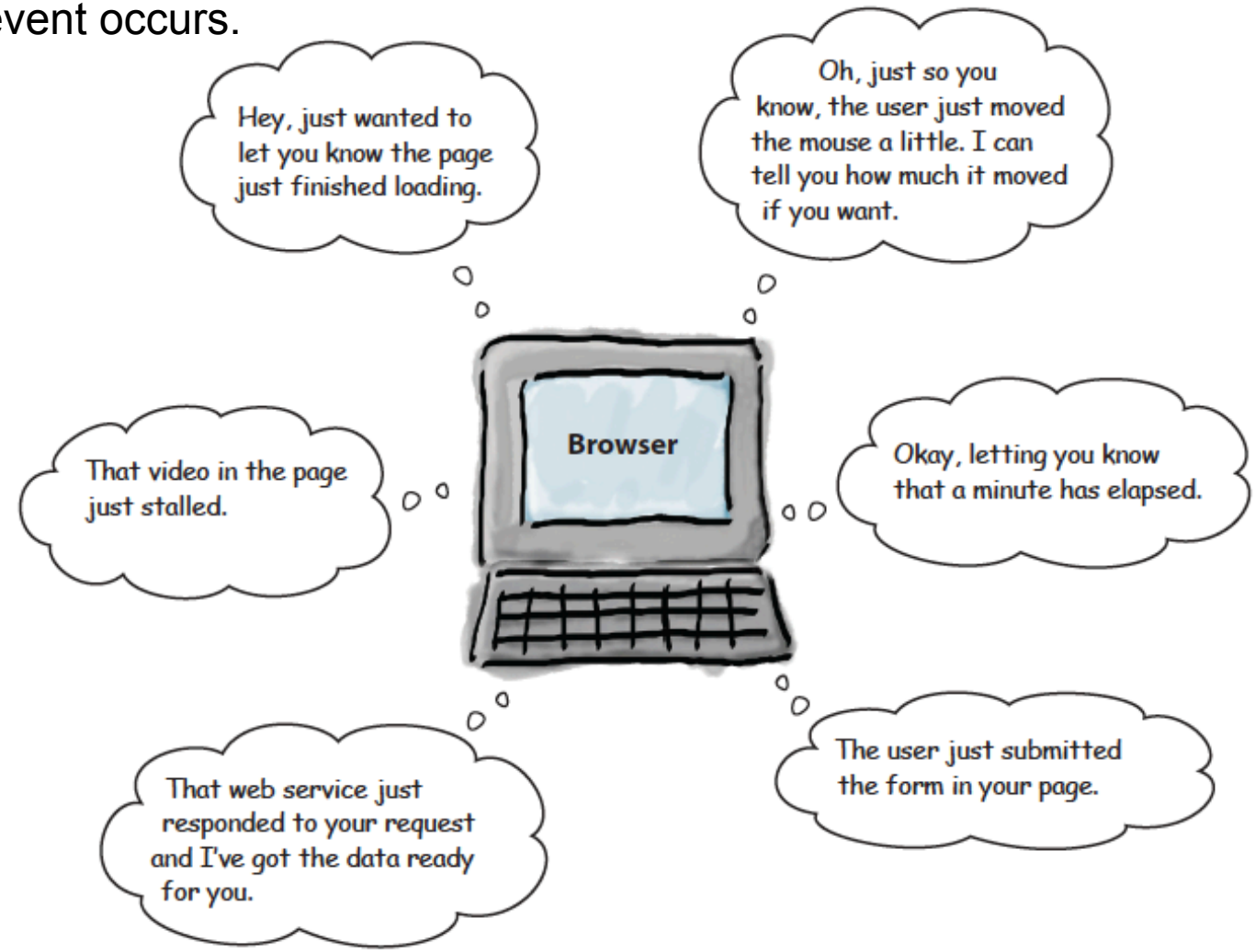
An **event** is a user initiated activity.

- For example: loading or reloading a page
- clicking on a button or hyperlink
- select an item in a pull-down menu
- placing the mouse cursor over an image, or removing the cursor from an image.

JavaScript provides a way to respond to each of these types of user interactions with the web browser (as well as many others) through what are known as **event handlers**.

EVENTS ARE OPPORTUNITIES

So basically, whenever there's an **event**, there is an opportunity for your code to **handle** it. That is, to supply some code that will be invoked when the event occurs.



WHAT'S AN EVENT HANDLER?

We write handlers to handle events.

- Handlers are typically small pieces of code that know what to do when an event occurs.
- In terms of code, **a handler is just a function**. When an event occurs, its handler function is called.
- You might also hear developers use the name callback or listener instead of handler.
- On the following pages, Let's step through what it takes to write the handler and to make sure a function gets invoked when the page load event is triggered.

1. First we need to write a function that can handle the page load event when it occurs. In this case, the function is going to announce to the world "I'm alive!" when it knows the page is fully loaded.

A handler is just an ordinary function.

```
function pageLoadedHandler() {
    alert("I'm alive!");
}
```

Here's our function, we'll name it pageLoadedHandler, but you can call it anything you like.

Remember we often refer to this as a handler or a callback.

This event handler doesn't do much. It just creates an alert

2. Now that we have a handler written and ready to go, we need to wire things up so the browser knows there's a function it should invoke when the load event occurs. To do that we use the onload property of the window object, like this:

```
window.onload = pageLoadedHandler;
```

In the case of the load event, we assign the name of the handler to the window's onload property.

Now when the page load event is generated, the pageLoadedHandler function is going to be called.

We're going to see that different kinds of events are assigned handlers in different ways.

3. That's it! Now, with this code written, we can sit back and know that the browser will invoke the function assigned to the window.onload property when the page is loaded.

Go ahead and create a new file, “event.html”, and add the code to test your load event handler. Load the page into the browser and make sure you see the alert.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title> I'm alive! </title>
  <script>
    window.onload = pageLoadedHandler;
    function pageLoadedHandler() {
        alert("I'm alive!");
    }
  </script>
</head>
<body>
</body>
</html>
```

First the browser loads your page, and starts parsing the HTML and building up the DOM.

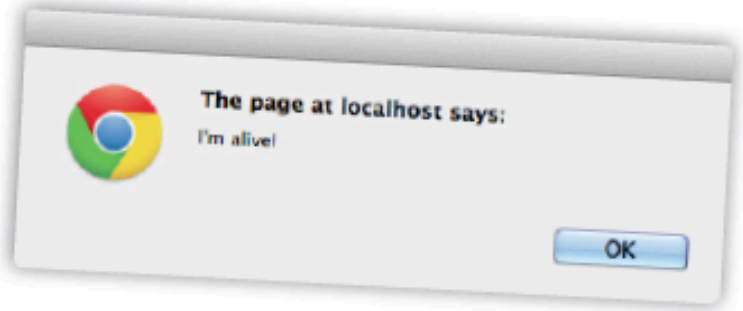
When it gets to your script the browser starts executing the code.

For now, the script just defines a function, and assigns that function to the window.onload property. Remember this function will be invoked when the page is fully loaded.

Then the browser continues parsing the HTML.

When the browser is done parsing the HTML, and the DOM is ready, the browser calls the page load handler.

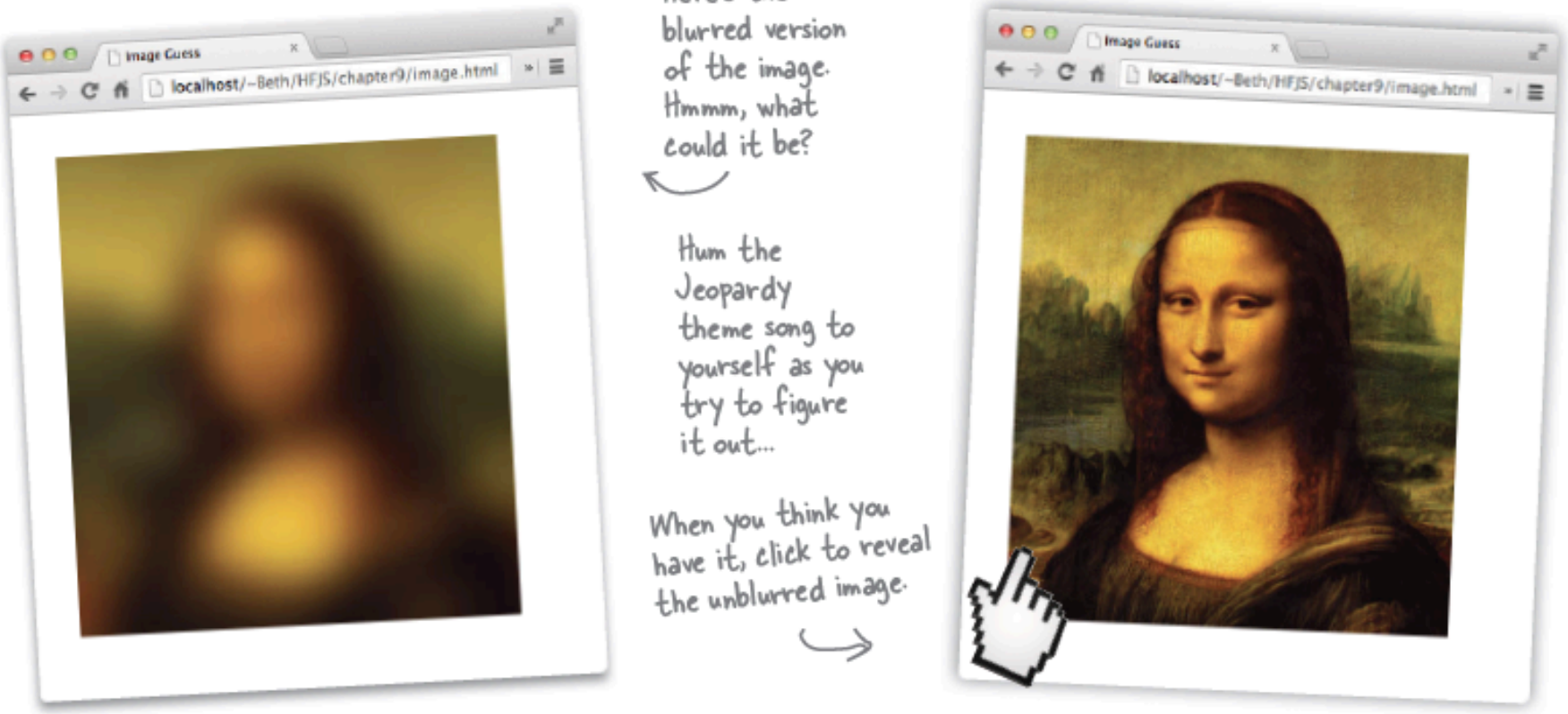
Which in this case creates the “I’m alive” alert



ONCLICK EVENT

The best way to understand events is with experience, so let's get some more by writing a simple game. The game works like this: you load a page and are presented with an image. Not just any image, but a really blurred image.

Your job is to guess what the image is. And, to check your answer, you click on the image to unblur it.



MARKUP

We'll use two JPG images. One is blurred and the other isn't. We've named them "zeroblur.jpg" and "zero.jpg" respectively. Here's the markup:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title> Image Guess </title>
  <style> body { margin: 20px; } </style>
  <script>  </script>
</head>
<body>
    <img id="zero" src="zeroblur.jpg">
</body>
</html>
```

Just some basic HTML, with a <script> element all ready for our code. Rather than use a separate file for the JavaScript, we'll keep it simple and add the script here. As you'll see, there is very little code needed to implement this.

And here's the blurred image, placed in the page. We'll give it an id of "zero". You'll see how we use the id in a sec...

CREATE A HANDLER FOR THE CLICK EVENT

Go ahead and load this markup in your browser and you'll see the blurred image.

To implement the game, we need to react to a click on the image in order to display the unblurred version of the image.

Lucky for us, every time an HTML element in the page is clicked (or touched on a mobile device), an event is generated. Your job is to create a handler for that event, and in it write the code to display the unblurred version of the image. Here's how you're going to do that:

1. **Access the image object in the DOM and assign a handler to its onclick property.**
2. **In your handler, write the code to change the image src attribute from the blurred image to the unblurred one.**

STEP 1: ACCESS THE IMAGE IN THE DOM

Getting access to the image is old hat for you; we just need to use our old friend, the `getElementById` method, to get a reference to it.

```
var image = document.getElementById("zero");
```

Here we're grabbing a reference to the image element and assigning it to the image variable.

Oh, but we also need this code to run only *after* the DOM for the page has been created, so let's use the window's `onload` property to ensure that. We'll place our code into a function, `init`, that we'll assign to the `onload` property.

Remember, we can't get the image from the DOM until the page has finished loading.

```
window.onload = init;
function init() {
    var image = document.getElementById("zero");
}
```

We create a function init, and assign it to the onload handler to make sure this code doesn't run until the page is fully loaded.

In the code of init, we'll grab a reference to the image with id="zero".

Remember in JavaScript the order in which you define your functions doesn't matter. So we can define init after we assign it to the onload property.

STEP 2: ADD THE HANDLER, AND UPDATE THE IMAGE

To add a handler to deal with clicks on the image, we simply assign a function to the image's `onclick` property. Let's call that function `showAnswer`, and we'll define it next.

```
window.onload = init;
function init() {
    var image = document.getElementById("zero");
    image.onclick = showAnswer;
}
```

Using the image object from the DOM, we're assigning a handler to its onclick property.

Now we need to write the `showAnswer` function, which unblurs the image by resetting the image element's `src` property to the unblurred image:

First, we have to get the image from the DOM again.

```
function showAnswer() {
    var image = document.getElementById("zero");
    image.src = "zero.jpg";
}
```

Remember the blurred version is named "zeroblur.jpg" and the unblurred is named "zero.jpg".

Once we have the image, we can change it by setting its src property to the unblurred image.

THE FULL CODE

Make sure the showAnswer function is outside the init function.

```
window.onload = init;
function init() {
    var image = document.getElementById("zero");
    image.onclick = showAnswer;
}

function showAnswer() {
    var image = document.getElementById("zero");
    image.src = "zero.jpg";
}
```

Here's the code you're executing...

HMMM

Why do we need to use getElementById again in showAnswer?

1. The init function is called when the page is loaded. But the showAnswer function isn't called until later, when you click the image. So these two event handlers get called at two different times.
2. In addition, in the init function we're putting the object returned by getElementById into a local variable image, which means when that function completes, the variable falls out of scope and is destroyed. So later, when the showAnswer function is called, we have to get the image object again from the DOM. Sure, we could have put this in a global variable, but over use of globals can lead to confusing and buggy code, which we'd like to avoid.

ASSIGNMENT

1. Please complete and submit you events.html page.

(You will receive a second events pdf to pick up where this one left off.)

Work in Groups:

2. Add a clickable button beneath the image so that it advances the image to the unblurred version.