

WEB-215 Project 12 – Modal Boxes

Modal Dialog

A modal dialog box is a 'window' that overlays the page. You've seen these with image lightboxes or confirmation messages. You can't interact with anything else on the page until the modal box is closed.

In this tutorial, you'll create the HTML, CSS, and JS required for a modal dialog. For its content, you'll just load the text of the element you clicked. In a future project, you'll populate the modal dialog's content from an external data source.

The Plan:

1. Code the HTML with containers for the modal dialog.
2. Code the CSS to style the modal dialog and hide it by default
3. Code the JS to display the modal dialog and populate its content when clicked. Also code a way to close it.

HTML

1. Create a new HTML file in the *modal* folder named **index.html**.
2. Code a standard HTML skeleton/structure with an empty BODY.
3. Thinking ahead, code `<link>` and `<script>` tags to link to our future CSS and JS files. Name them whatever you like.
4. Code an H1 and two interactive elements – a button and an anchor:

`<body>`

```
<h1>Modal Dialog Boxes</h1>
<button>This is a button</button>
<a href="http://google.com">Link to Google</a>
```

5. The modal dialog box requires three pieces. Because these pieces are needed for presentation only, it's appropriate to use DIVs. Code this after the existing code.
 - a. An outer box that will 'black out' the underlying web page
 - b. An inner box that will hold the content
 - c. A close button, which will be an HTML character code for the letter X

```
<div id="modalOuterBox">
  <div id="modalContent">
    <span id="close">&times;</span>
  </div>
</div>
```

CSS

Outer Box

1. Create a rule to style the outer box. I'll list the declarations in an order that makes sense for you to save and test the page after each one. When finished, you'll need to rearrange the declaration so they're in alphabetical order (as per our code formatting guidelines). First – target the outer box and give it a black background faded to 50%. Save and test the page after coding this so you can see what it does.

```
#modalOuterBox {  
  background-color: rgba(0, 0, 0, .5);  
}
```

2. Set its `position` to `fixed` (so it remains in place if the page is scrolled) and position it 0 px from the top and left corner of the viewport. Again, save and refresh after you code this to see what happens.

```
position: fixed;  
left: 0;  
top: 0;
```

3. Set both its `width` and `height` to 100% so that it fills the viewport. Save and refresh.

```
width: 100%;  
height: 100%;
```

4. When we're finished with the JS, we'll come back here and add `display: none` so that it's hidden by default. For now, we need to leave it visible so we can style it. Go ahead and alphabetize the declarations now.

Content Box

1. Create a new rule to target the inner content box and give it a background color of white. Save and refresh.

```
#modalContent {  
  background-color: #fff;  
}
```

2. Give it a `width` of 80% and center it on the page with auto margins. Move it down a little with a top margin. Save and refresh.

```
width: 80%;  
margin: 2rem auto auto;
```

3. Give it a border on all four sides and then override the top border to make it thicker. Save and refresh.

```
border: 1px solid #000;  
border-top: 10px solid #333;
```

4. Give it an outer glow and add some padding so that when we eventually add content, it won't be cramped. Save and refresh.

```
box-shadow: 0 0 5px #000;  
padding: 1rem;
```

5. Alphabetize the declarations.

Close Button

1. Create a new rule to target the close button and increase its font size. Save and refresh.

```
#close {  
  font-size: 2rem;  
}
```

2. Position it in the top-right corner. Save and refresh.

```
position: absolute;  
top: 0;  
right: 0;
```

3. Doh! It's in the top-right corner of the viewport. We want it in the top-right corner of the content box. Return to the #modalContent rule and add relative positioning to it. Save and refresh.

```
#modalContent {  
  background-color: #fff;  
  border: 1px solid #000;  
  
  padding: 1rem;  
  position: relative;  
  width: 80%;  
}
```

4. Tweak the close button's position to move it 10px away from the right side. Be sure to make this edit in the #close rule. Save and refresh.

```
right: 10px;
```

5. Alphabetize.

Close Button Interaction

1. Create a new rule to target the close button on hover to change its color to red, display 'the finger', and rotate it 15 degrees. Save and refresh.

```
#close:hover {  
  color: #f00;  
  cursor: pointer;  
  transform: rotate(15deg);  
}
```

2. Let's make the rotation animate. First, change it to 90 degrees.

```
transform: rotate(90deg);
```

3. Next, add a transition on the original #close rule so that any properties that change from then to the hover state animate. Be sure to add this to the #close rule and not the #close:hover rule. Save and refresh.

```
transition: all .25s;
```

4. If you increase the transition from .25 seconds to something ridiculous like 5.25 seconds, you'll see both the rotation *and* the color change animate. Once you're done playing, set it back to .25

5. Mid-way through our JS code, we'll come back and hide the modal box completely. For now, leave it visible.
6. Alphabetize.

JavaScript

Close the Modal

First, let's figure out how to close the modal dialog. Then we can fix the CSS to hide it by default. Finally, we can work on showing it when something is clicked.

1. Call a function and then declare it.

```
modal();  
  
function modal(){  
|  
}
```

2. In the function block, let's grab all the pieces we need:

- a. The outer box
- b. The inner content box
- c. The close button
- d. The button and link on the page

```
const CLOSE = document.querySelector('#close');  
const BOXOUT = document.querySelector('#modalOuterBox');  
const BOXIN = document.querySelector('#modalContent');  
const BTN = document.querySelector('button');  
const A = document.querySelector('a');
```

3. Attach an event listener to the close button. When clicked, change the `display` value of the outer box.

```
CLOSE.addEventListener('click', function(){  
|   modalOuterBox.style.display = 'none';  
});
```

4. Save, refresh, and click the X close button in the modal dialog. It should 'close'.
5. It works, but since the anonymous function has a single statement, we can convert it to a much shorter arrow function.

```
CLOSE.addEventListener('click', () => modalOuterBox.style.display = 'none');
```

6. Save and refresh and make sure the close button still works.
7. We also want to close the modal box if users click anywhere outside of the white content box. You'd think you can just add an event listener to the faded black outer box. We'll try that so you can see it won't work as expected. I'll revert back to an anonymous function because I know what's coming later.

```
BOXOUT.addEventListener('click', function (){  
|   modalOuterBox.style.display = 'none';  
});
```

8. Save, refresh, and test it. Click any of the faded black outer box and it *does* close. Great! Refresh the page to bring it back and this time click the white content box. The modal closes again. No good. We only want to close it if the faded back box is clicked or the close button is clicked. Edit the code so we can test again and see what's happening.

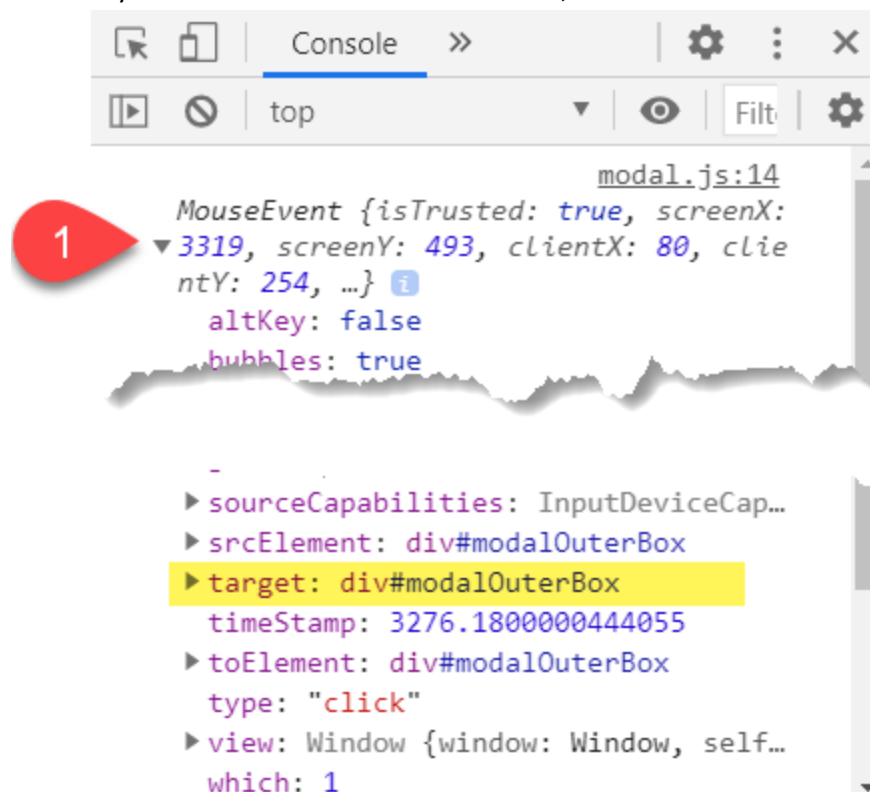
```
BOXOUT.addEventListener('click', function(){  
    //modalOuterBox.style.display = 'none';  
    console.log(this);  
});
```

9. Save and refresh. Because the display line is commented out, the modal won't close. But every time you click, a message is logged to the console showing you the value of this – meaning the element you clicked. Click around the page and examine the console messages. No matter where you click (faded black outer box, white inner box, or close button) it always registers as having clicked the faded black outer box. That's because of something in JS called bubbling. Basically, the inner box and close button live inside the outer box. So if an inner element is clicked, that click registers to the outer containing element. (The close button still closes the modal because it still has its own event listener.)

10. Pass the event itself to the anonymous function and log that to the console instead.

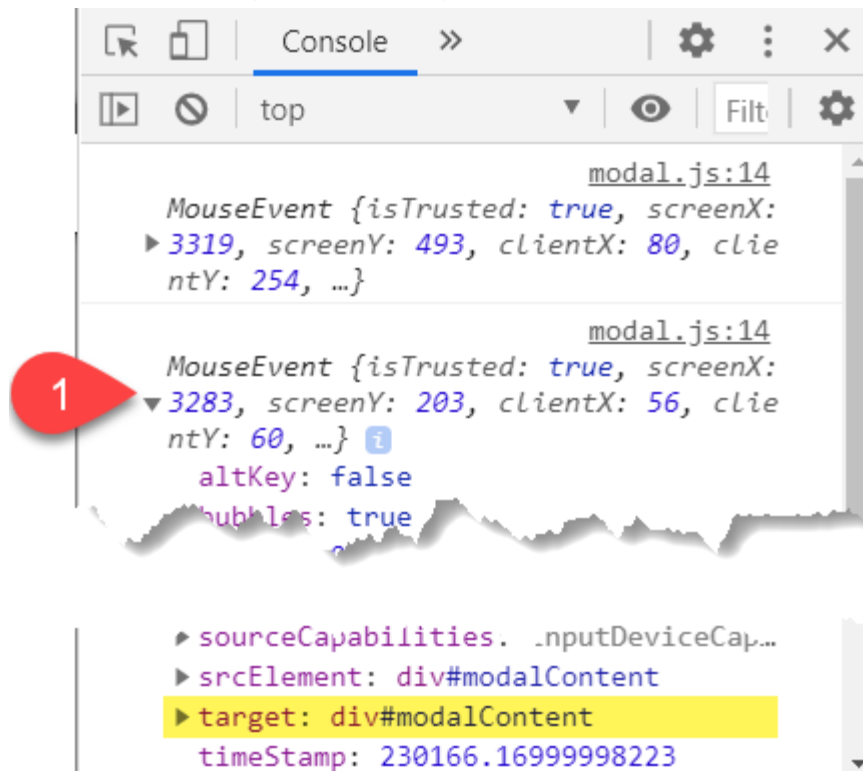
```
BOXOUT.addEventListener('click', function(e){  
    //modalOuterBox.style.display = 'none';  
    console.log(e);  
});
```

11. Click once on the faded black outer box and then stop! Don't click anything else. The console shows the single click event was logged. And it has a little triangle to its left. Click that triangle to expand the details and you'll see all the properties and methods available to the event. Scroll down that list and find the `target` property. Because you clicked the faded black outer box, that element shows as the target (`#modalOuterBox`.)



12. Collapse the arrow to hide the details.

13. Click the white inner box, look in the console, expand its details, and locate its target. This time the target shows the white inner box (`#modalContent`) was clicked.



14. Collapse the details.

15. Click the close button, expand its details, and notice this time the details show that `#close` was the target. (And the modal closed because that button still works.) So – the `target` property of the event tells us what we really clicked on. We can use that by testing to see if the target matches the outer box. Edit the code to test for that. If the event's target is the outer box, only then should the modal close.

```
BOXOUT.addEventListener('click', function(e){
  //modalOuterBox.style.display = 'none';
  console.log(e);
  if(e.target == document.querySelector('#modalOuterBox')){
    modalOuterBox.style.display = 'none';
  }
});
```

16. Save and refresh. Test it. Click the faded black area and the modal closes because the target was `#modalOuterBox`. Refresh and click the white inner area. The modal doesn't close because the target was not `#modalOuterBox`. (It was `#modalContent`.) Refresh and click the close button. The modal closes because the close button has its own event listener.

17. Clean up the code. Delete the old commented-out line and the console log.

```
BOXOUT.addEventListener('click', function(e){
  if(e.target == document.querySelector('#modalOuterBox')){
    modalOuterBox.style.display = 'none';
  }
});
```

18. Finally, we can optimize the code by using a new JavaScript method `matches()`. Currently, the IF statement is saying, "if the event's target matches the element `#modalOuterBox...`" We can rewrite that like this:

```
BOXOUT.addEventListener('click', function(e){
  if(e.target.matches('#modalOuterBox')){
    modalOuterBox.style.display = 'none';
  }
});
```

19. Save, refresh, and test. The modal closes if the faded black outer box or close button are clicked. It doesn't close if the white content area is clicked.

Show the Modal When the Button Is Clicked

1. Since our 'close the modal' code is working, let's hide the modal on page load by default. GO back to the CSS and edit the `#modalOuterBox` rule to hide it.

```
#modalOuterBox {
  background-color: rgba(0, 0, 0, .5);
  display: none;
  height: 100%;
```

2. Return to the JS and add an event listener to the button. Grab the button's text when clicked and log it to the console. Save, refresh, and test it.

```
BTN.addEventListener('click', function(){
  const btnText = this.textContent;
  console.log(btnText);
});
```

3. Now let's set the inner box's text to whatever the button's text is and then show the modal.

```
BTN.addEventListener('click', function(){
  const btnText = this.textContent;
  console.log(btnText);
  BOXIN.textContent = btnText;
  BOXOUT.style.display = 'block';
});
```


4. Save, refresh, and click the button. The modal displays and shows the text of the button. But oh no – the close button is gone! That's because we deleted all the content inside the inner box when we set its new `textContent`. We'll need a more robust way to create content and load it into that inner box. Let's create a new element and then append it as new child. That will maintain the existing content. I've commented out the `textContent` line – but you can simply delete it.

```
BTN.addEventListener('click', function(){
  const btnText = this.textContent;
  console.log(btnText);
  //BOXIN.textContent = btnText;
  const PARA = document.createElement('p');
  PARA.textContent = btnText;
  BOXIN.appendChild(PARA);
  BOXOUT.style.display = 'block';
});
```

5. Save, refresh and click the button. The modal appears with the new text content and the close button. Click the faded black background or the close button to close the modal. Then... click the button again. Doh! We get double text! That's because the old text was still there and we just appended another child. Every time we close the modal and click the button, an additional paragraph gets appended. The fix? Before we add the new text, we need to delete the existing (old) text. First, give the paragraph an attribute so we can easily identify it later.

```
//BOXIN.textContent = btnText;
const PARA = document.createElement('p');
PARA.setAttribute('data-desc', 'content');
PARA.textContent = btnText;
```

6. Save, refresh, and click the button to display the modal. Right-click the white content box and choose the **Inspect** option to inspect the generated HTML. You'll notice that paragraph has a `data-desc` attribute now with a value of `content`.
7. If a paragraph with `data-desc='content'` exists, let's remove it before adding the new content and displaying the modal.

```
const btnText = this.textContent;
console.log(btnText);
//BOXIN.textContent = btnText;
const oldData = document.querySelector('[data-desc=content]');
if(oldData){
  oldData.remove();
}
const PARA = document.createElement('p');
PARA.setAttribute('data-desc', 'content');
```


8. Save, refresh, and click the button/close the modal a few times. You'll notice the content doesn't get duplicated any more. Let's optimize our `if` statement to use a ternary operator. "If `oldData` exists, then remove it. Otherwise, do nothing."

```
//BOXIN.textContent = btnText;  
const oldData = document.querySelector('[data-desc=content]')  
oldData ? oldData.remove() : null;  
const PARA = document.createElement('p');  
PARA.setAttribute('data-desc', 'content');
```

9. Clean up the code and delete the comment-out and console log statements.

```
BTN.addEventListener('click', function(){  
  const btnText = this.textContent;  
  const oldData = document.querySelector('[data-desc=content]')  
  oldData ? oldData.remove() : null;  
  const PARA = document.createElement('p');  
  PARA.setAttribute('data-desc', 'content');  
  PARA.textContent = btnText;  
  BOXIN.appendChild(PARA);  
  BOXOUT.style.display = 'block';  
});
```

Show the Modal When the Link Is Clicked

Same thing – but for the anchor tag.

1. Duplicate the BTN code and just change the element that has the event listener.

```
A.addEventListener('click', function(){
  const btnText = this.textContent;
  const oldData = document.querySelector('[data-desc=content]');
  oldData ? oldData.remove() : null;
  const PARA = document.createElement('p');
  PARA.setAttribute('data-desc', 'content');
  PARA.textContent = btnText;
  BOXIN.appendChild(PARA);
  BOXOUT.style.display = 'block';
});
```

2. Save, refresh, and click the link to Google. It works. Kind of. The modal displays for a second but then the link is followed and Google displays. We need to cancel the default behavior of the link. Pass the event and prevent its default behavior.

```
A.addEventListener('click', function(e){
  e.preventDefault();
  const btnText = this.textContent;
  const oldData = document.querySelector('[data-desc=content]');
```

3. Save, refresh, and test everything – the button, the Google link, and closing the modal. Everything should work as expected. Yay. But the code is a little bloated. Both the BTN and A code duplicate each other.

```
BTN.addEventListener('click', function(){
  const btnText = this.textContent;
  const oldData = document.querySelector('[data-desc=content]');
  oldData ? oldData.remove() : null;
  const PARA = document.createElement('p');
  PARA.setAttribute('data-desc', 'content');
  PARA.textContent = btnText;
  BOXIN.appendChild(PARA);
  BOXOUT.style.display = 'block';
});
```


```
A.addEventListener('click', function(e){
  e.preventDefault();
  const btnText = this.textContent;
  const oldData = document.querySelector('[data-desc=content]');
  oldData ? oldData.remove() : null;
  const PARA = document.createElement('p');
  PARA.setAttribute('data-desc', 'content');
  PARA.textContent = btnText;
  BOXIN.appendChild(PARA);
  BOXOUT.style.display = 'block';
});
```

4. Let's move all that code to its own function and just call it from the BTN and A event listeners. We just need to pass along the text to display. We'll change some variable names while we're at it so they are more meaningful. Make sure your *showModal* function is not nested inside the *modal* function.

```
BTN.addEventListener('click', function(){  
  const btnText = this.textContent;  
  showModal(btnText);  
});
```

```
A.addEventListener('click', function(e){  
  e.preventDefault();  
  const linkText = this.textContent;  
  showModal(linkText);  
});
```

```
function showModal(modalText){  
  const oldData = document.querySelector('[data-desc=content]')  
  oldData ? oldData.remove() : null;  
  const PARA = document.createElement('p');  
  PARA.setAttribute('data-desc', 'content');  
  PARA.textContent = modalText;  
  BOXIN.appendChild(PARA);  
  BOXOUT.style.display = 'block';  
}
```



5. Test your page in the browser and you'll it no longer works! Explore the console and you'll see why. Two variables – BOXIN and BOXOUT – aren't defined. They were declared in *modal()*, but are also used in *showModal()* where they don't exist. You have two options:
- Redeclare them at the top of *showModal()*, or...
 - ...delete the keyword *const* from their declarations in *modal()* to make them global variables available to all functions.

Final Code

The final code, minus DocBlocks, is shown on the next page.

```

1  modal();
2
3  ▼ function modal(){
4      const CLOSE = document.querySelector('#close');
5      const BOXOUT = document.querySelector('#modalOuterBox');
6      const BOXIN = document.querySelector('#modalContent');
7      const BTN = document.querySelector('button');
8      const A = document.querySelector('a');
9
10     CLOSE.addEventListener('click', () => modalOuterBox.style.display = 'none');
11
12     ▼ BOXOUT.addEventListener('click', function(e){
13         ▼ if(e.target.matches('#modalOuterBox')){
14             modalOuterBox.style.display = 'none';
15         }
16     });
17
18     ▼ BTN.addEventListener('click', function(){
19         const btnText = this.textContent;
20         showModal(btnText);
21     });
22
23     ▼ A.addEventListener('click', function(e){
24         e.preventDefault();
25         const linkText = this.textContent;
26         showModal(linkText);
27     });
28 }
29
30 ▼ function showModal(modalText){
31     const BOXOUT = document.querySelector('#modalOuterBox');
32     const BOXIN = document.querySelector('#modalContent');
33     const oldData = document.querySelector('[data-desc=content]');
34     oldData ? oldData.remove() : null;
35     const PARA = document.createElement('p');
36     PARA.setAttribute('data-desc', 'content');
37     PARA.textContent = modalText;
38     BOXIN.appendChild(PARA);
39     BOXOUT.style.display = 'block';
40 }

```

Get Ready for Submission

- Have your **modal** folder ready to submit after you finish both tutorials.