WEB-215 Project 12 – Lazy Loading

Lazy Loading

Consider a page with a ton of content. Rather than taking all the energy and resources to load all the content at once, the browser gets lazy and loads only some of the content. If the user scrolls to the bottom of the page, the browser wakes up from its nap and loads a little more content. Each time the users scroll to the bottom, more content is loaded. This results in a seemingly endless page. More content is loaded until there's no more content.

In this tutorial, you'll create a lazy loading page and load content from a second JavaScript file. In a real web site (and in a future project) you'd load the additional content from an external source such as a database or JSON file. You'll learn about JSON files in a future project.

The Plan:

1. Explore height and offset properties so you can figure out when a user scrolls to the bottom of a page
2. Get net content from an external source and display it on the page as needed

HTML

1. Examine the HTML code. There are a number of long 'dummy paragraphs' hard-coded in the HTML to create a scrollable page. Notice there are two JS files linked. The first one is essentially a flat-file database. It contains one object with blocks of text as members. The second JS file is where you'll write your code. As your code relies on having external data, the data file (data.js) must be loaded first. As there's no executable code inside of data.js, it can load with ASYNC rather than DEFER.

2. Preview the page in a browser. Shrink your window if there's not enough content to make you scroll. Scroll to the bottom of the page and notice it ends at the 44444444 paragraph.

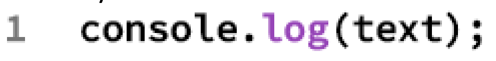
Examine JavaScript

1. Display the code for **data.js**. It contains a single variable, `text`, that stores an array with five text blocks as the members. `let` is used because maybe this array of content will be updated dynamically.

Code JavaScript

1. Display the code for **lazyloading.js**.

2. Code a single statement to display the content of the `text` variable in the console. Even though the variable `text` hasn't been defined in this file, it was defined in **data.js** which loaded first. There, the `text` variable exists by the time this statement runs.



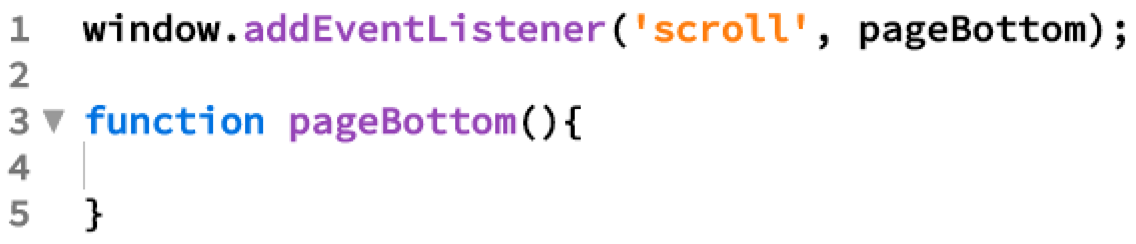
```
console.log(text);
```

3. Save, refresh, and examine the console log. The text blocks appear as array members. This means any code we write in **lazyloading.js** has access to that array.

4. Delete the `console.log` statement.

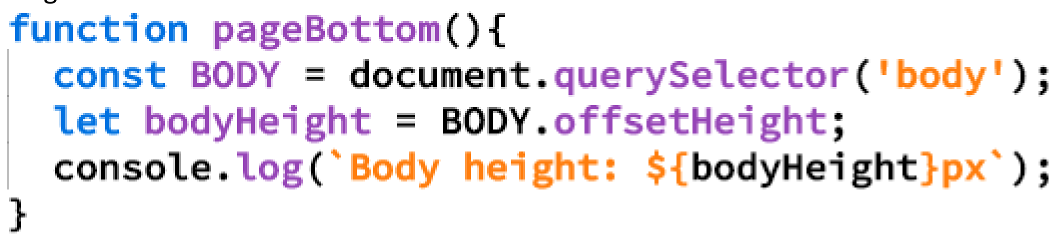
Explore Height and Offset Properties

1. Add an event listener to the entire window so that when a user scrolls, a function is called. Then declare the function.



```
window.addEventListener('scroll', pageBottom);

function pageBottom(){

}
```

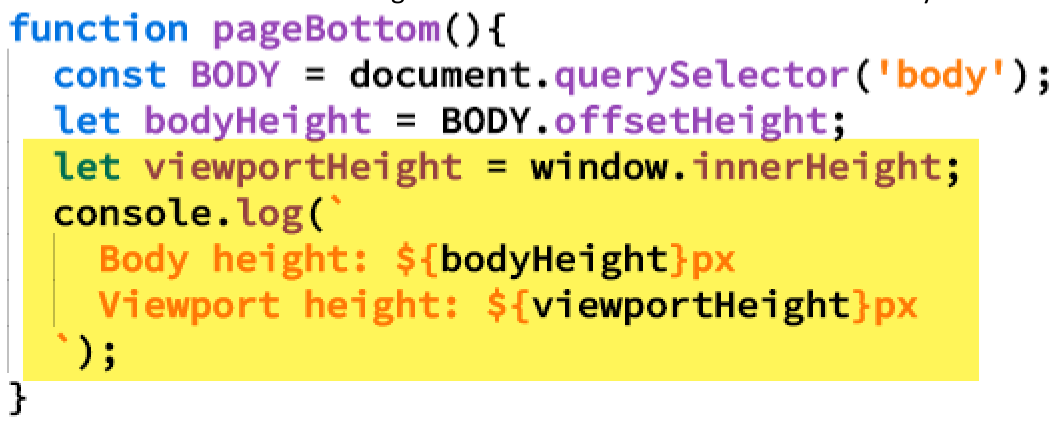
2. Within the function block, get the body element, then get its height using the `offsetHeight` property. The `offsetHeight` property returns the viewable height of an element in pixels, including padding, border and scrollbar, but not the margin. Note in this case, it's the height of the body – not the viewport! Finally log its height to the console.



```
function pageBottom(){
  const BODY = document.querySelector('body');
  let bodyHeight = BODY.offsetHeight;
  console.log(`Body height: ${bodyHeight}px`);
}
```

3. Save, refresh, display the developer console, and scroll down the page. The console shows the height of the body as you scroll.

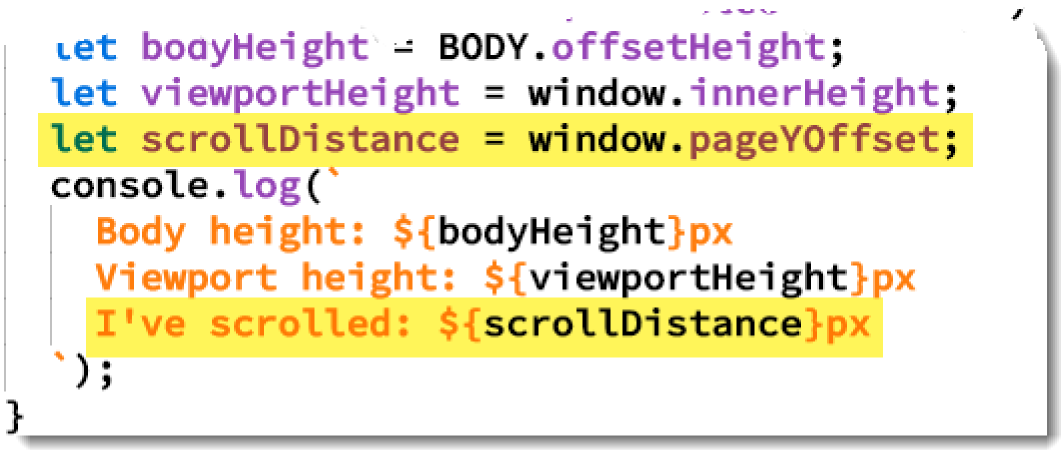
4. Next, get the height of the viewport using the `innerHeight` property of the `window` object. Display that in the console. Note the console.log statement has been tweaked for readability.



```
function pageBottom(){
  const BODY = document.querySelector('body');
  let bodyHeight = BODY.offsetHeight;
  let viewportHeight = window.innerHeight;
  console.log(`
    Body height: ${bodyHeight}px
    Viewport height: ${viewportHeight}px
  `);
}
```

5. Save, refresh, and scroll down the page. The console shows the height of the *body* and the *viewport* as you scroll. Notice that [1] neither of these values change as you scroll, and [2] the *body* is taller than the *viewport*. Makes sense. The *body* is really tall to hold all the content. But we see only a portion of that content in the *viewport* – which is why we have to scroll down.

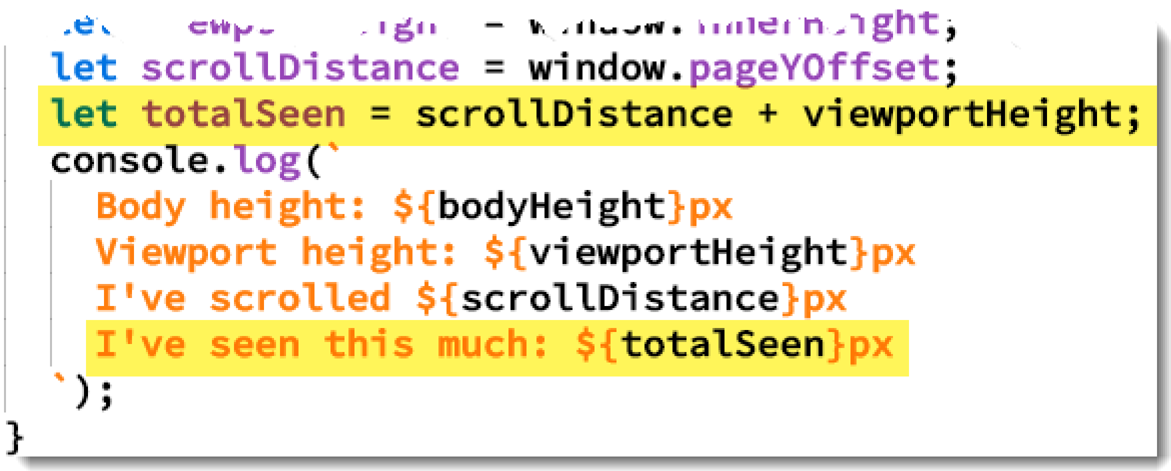
6. Next, let's capture the amount of pixels scrolled every time we scroll the page. Get that with the `pageYOffset` property of the `window` object.



```
  let bodyHeight = BODY.offsetHeight;
  let viewportHeight = window.innerHeight;
  let scrollDistance = window.pageYOffset;
  console.log(`
    Body height: ${bodyHeight}px
    Viewport height: ${viewportHeight}px
    I've scrolled: ${scrollDistance}px
  `);
}
```

7. Save, refresh, and scroll down the page. Now you get a new console message every time a different *scrollDistance* value shows up – which happens a lot!

8. Finally, think about the math. Let's use some easy numbers as an example. Assume the body is 1000px tall. If the viewport is 500px tall and you scrolled 100px, then you've seen 600px of the body. You haven't reached the end. If you scroll a total of 500px, then you've reached the end. (500px in the viewport at page load plus the 500px you scrolled equals 1000px.) So, if we add the viewport height to the number of pixels scrolled, we can compare that to the body height to see if we've reached the bottom. Let's add that as a final variable and console.log item.



```
  let viewportHeight = window.innerHeight;
  let scrollDistance = window.pageYOffset;
  let totalSeen = scrollDistance + viewportHeight;
  console.log(`
    Body height: ${bodyHeight}px
    Viewport height: ${viewportHeight}px
    I've scrolled ${scrollDistance}px
    I've seen this much: ${totalSeen}px
  `);
}
```

9. Save, refresh, and scroll down the page. Pause as you scroll and compare the *Body height* to the *I've seen this much*. As you get towards the bottom of the page, *I've seen this much* will be equal to or greater than the *Body height*.

10. Delete the console log (or comment it out if you want to keep it to examine later) and replace it with a conditional that compares the 'total seen' to the 'body height'. If we've seen the whole body, log a message that tells us it's time to load more content.

```
if(totalSeen >= bodyHeight){
  console.log('Time to load more content');
}
```

11. Save, refresh, and scroll. Once you get almost to the very bottom, the console log shows you it's time to load more content.

Load More Content

Loading content will be different for your application. Usually, you'd need to create a URL string as per an API's documentation, send that string to some server, parse through the data the server returns, and then load that as the new content. You'll do that in a future project. For now, we'll just grab extra content from that *data.js* file.

1. Replace the console.log statement with a variable that stores the value returned from another function. It'll be the job of the other function to get our new data. We'll pass that function the value of 0 for now so that it grabs the 1st paragraph in its array.

```
if(totalSeen >= bodyHeight){
  let newContent = getNewContent(0);
}
```

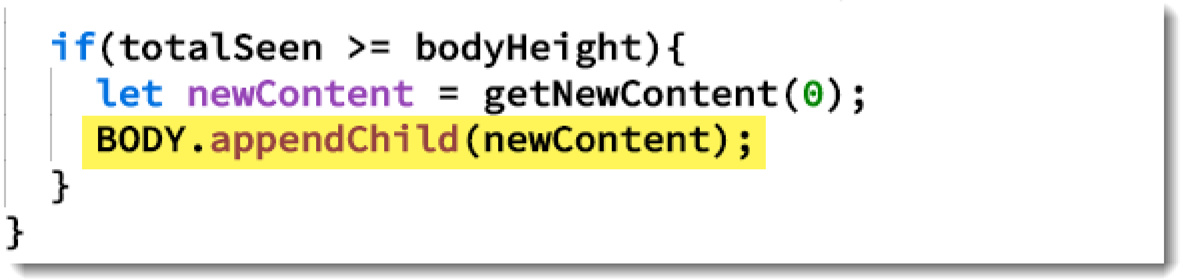
2. After the `pageBottom` function ends, create the new function, accepting one parameter.

```
function getNewContent(paragraphIndex){

}
```

3. Inside that new function, create a `P` element and load it up with the first paragraph from the **data.js** file. Remember – we can reference the array in that file with the `text` variable because *data.js* loaded before our code.

```
function getNewContent(paragraphIndex){
  const PARA = document.createElement('p');
  PARA.textContent = text[paragraphIndex];
  return PARA;
}
```

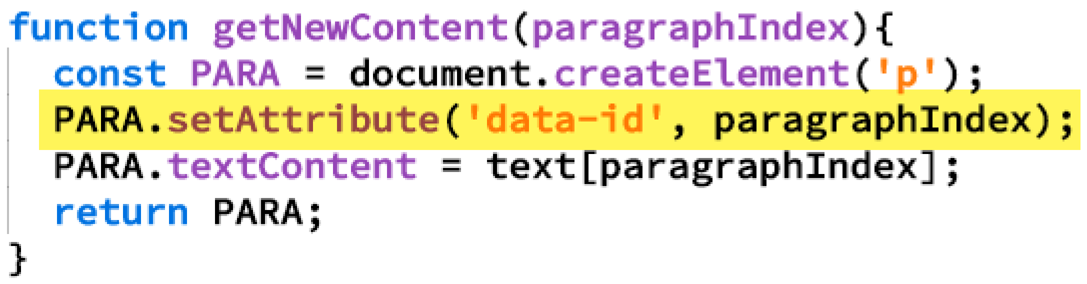
4. Return to the `pageBottom` function and append that returned paragraph element to the body.



```
  if(totalSeen >= bodyHeight){
    let newContent = getNewContent(0);
    BODY.appendChild(newContent);
  }
}
```

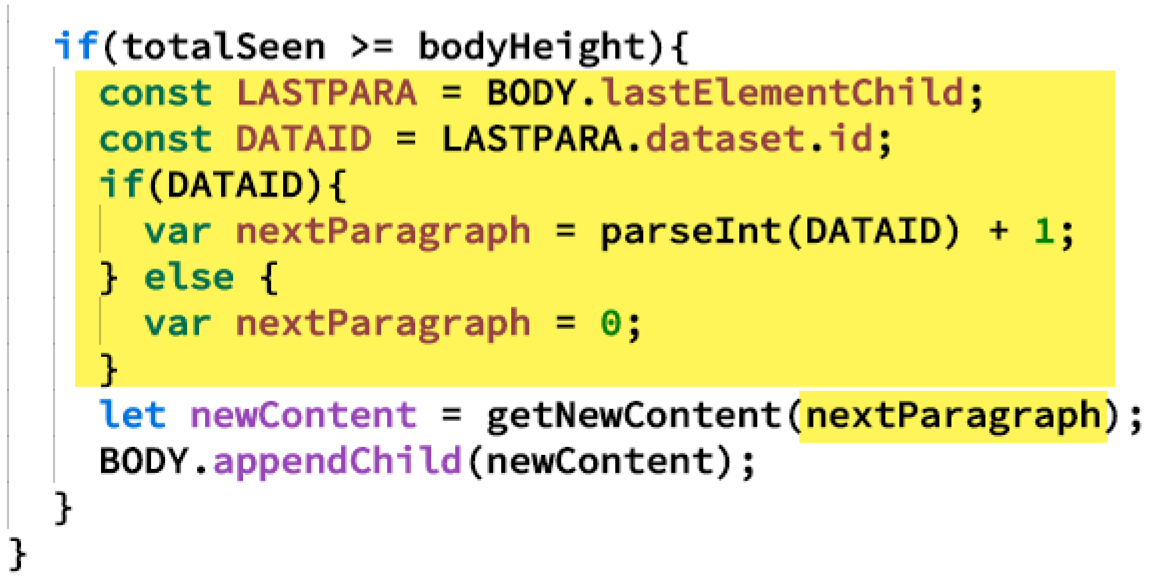
5. Save and refresh and scroll to the bottom. Watch the scrollbar button closely as you get close to the bottom and you'll see it jiggle to indicate the page just got a little taller with the new content added. As you continue to scroll down, the same paragraph continues to load because right now our function always loads paragraph index 0. We need to get creative to loading the paragraphs in order.

   First, we need to know which paragraph we last added. We can store that paragraph's index in a data-attribute. Add that to the `getNewContent` function.



```
function getNewContent(paragraphIndex){
  const PARA = document.createElement('p');
  PARA.setAttribute('data-id', paragraphIndex);
  PARA.textContent = text[paragraphIndex];
  return PARA;
}
```

6. Back in the `pageBottom` function, we need to grab the last paragraph on the page and examine its data-id attribute. If there isn't one, that means we haven't added any paragraphs yet and we need to ask for paragraph 0. If there is a paragraph with a data-id, then we need to ask for the next paragraph in the function call (rather than just asking for paragraph 0).

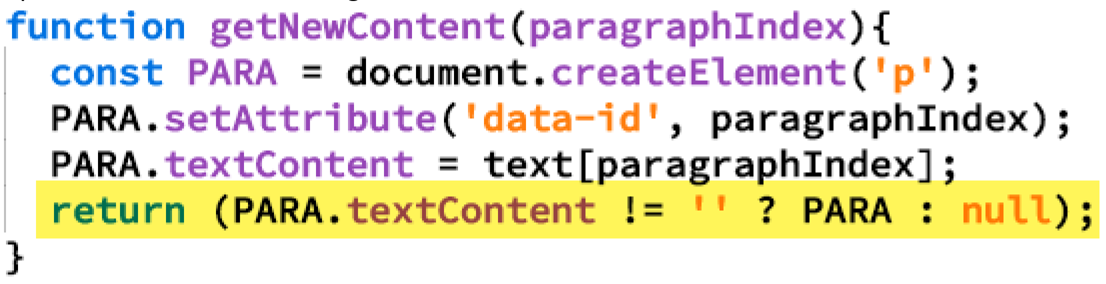


```
  if(totalSeen >= bodyHeight){
    const LASTPARA = BODY.lastElementChild;
    const DATAID = LASTPARA.dataset.id;
    if(DATAID){
      var nextParagraph = parseInt(DATAID) + 1;
    } else {
      var nextParagraph = 0;
    }
    let newContent = getNewContent(nextParagraph);
    BODY.appendChild(newContent);
  }
}
```

7. Save, refresh, and scroll. Now the paragraphs are added in order. When the last paragraph in the array is added, no more text gets added to the page. But if you switch to the *Elements* panel of the dev tools, you'll see that blank paragraphs continue to get added with an incrementing `data-id` value. That's because we don't have an error checking to make sure paragraphs only get added if text exists.

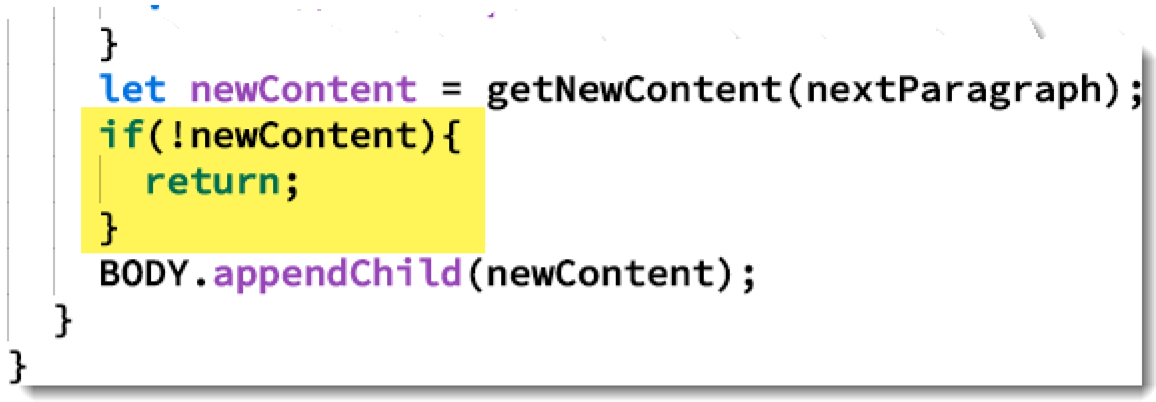
Fix the Blank Paragraph Bug

1. Rather than returning an empty paragraph if there was no more text, do a test. If text exists, return a paragraph with that text. If text doesn't exist, return `null`. Edit the `getNewContent` function and use a ternary operator to determine what gets returned.



```
function getNewContent(paragraphIndex){
  const PARA = document.createElement('p');
  PARA.setAttribute('data-id', paragraphIndex);
  PARA.textContent = text[paragraphIndex];
  return (PARA.textContent != '' ? PARA : null);
}
```

2. Finally, edit the `pageBottom` function to test what was returned. Remember it's the `newContent` variable that holds what was returned. So, it's value will either be the returned paragraph or the returned `null`. If it's null, then use the `return` keyword as that stops the function dead in its tracks. Run that test between getting the returned value from the helper function and appending the new content to the body.



```
    }
    let newContent = getNewContent(nextParagraph);
    if(!newContent){
      return;
    }
    BODY.appendChild(newContent);
  }
}
```

3. Save, refresh, and scroll. Keep your eye on the *Elements* console and you'll see new paragraphs being added one at a time as you continue to scroll to the bottom. Once the final paragraph loads and there no more text to return, blank paragraphs don't get added.

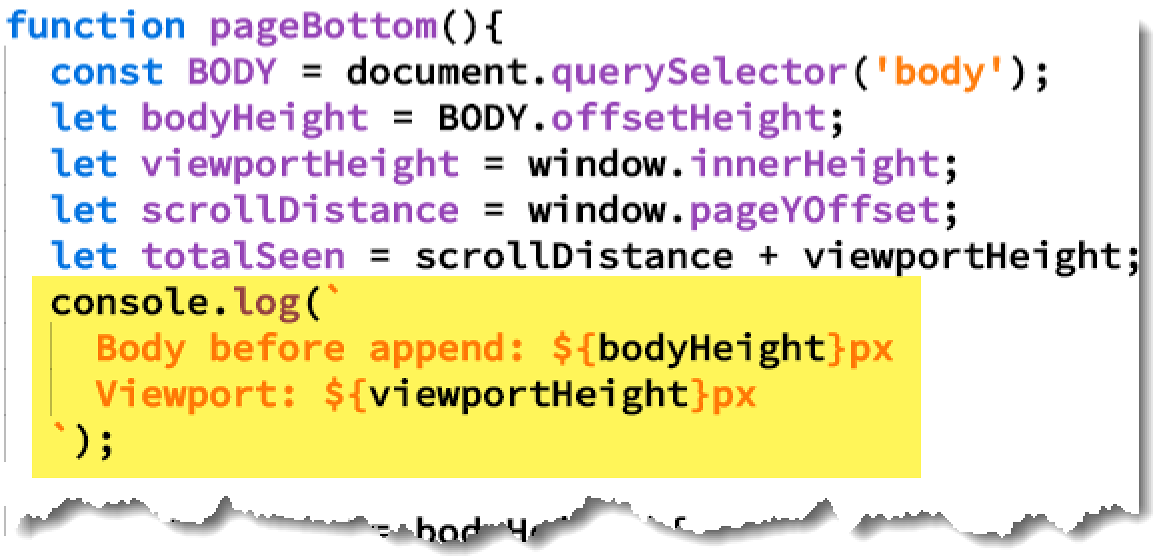
Short Pages

There's one final problem to address. What if the page, on page load, doesn't have a lot of content and there's no vertical scrollbar for users to scroll? If there's no scrollbar, users can't trigger the function that adds more content from the external source. You'll need to compare the body's height to the viewport's height when the page initially loads. If the body is shorter than the viewport, all of the initial content can display without scrolling and you'll need to force the additional content to load. If that's not clear, come back and read it again after completing the steps in this final section.

1. Display the code for the HTML and comment out lines 12-15 (the 1111, 2222, 3333, and 4444 paragraphs).

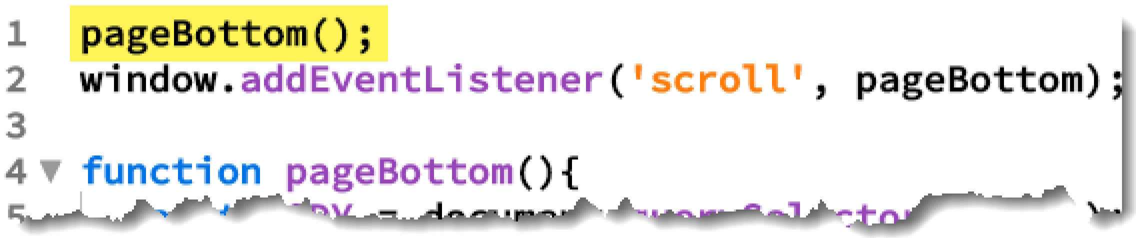
2. Save and refresh. Only one paragraph displays and there's no scrollbar for users to scroll down and trigger the loading of the additional content. When this happens, you'd want to automatically load content to fill the viewport.

3. Display the JavaScript code and add a `console.log` message to output the heights of the body and viewport before any new paragraphs get appended. Add this towards the top of the `pageBottom` function immediately after the variables are created.



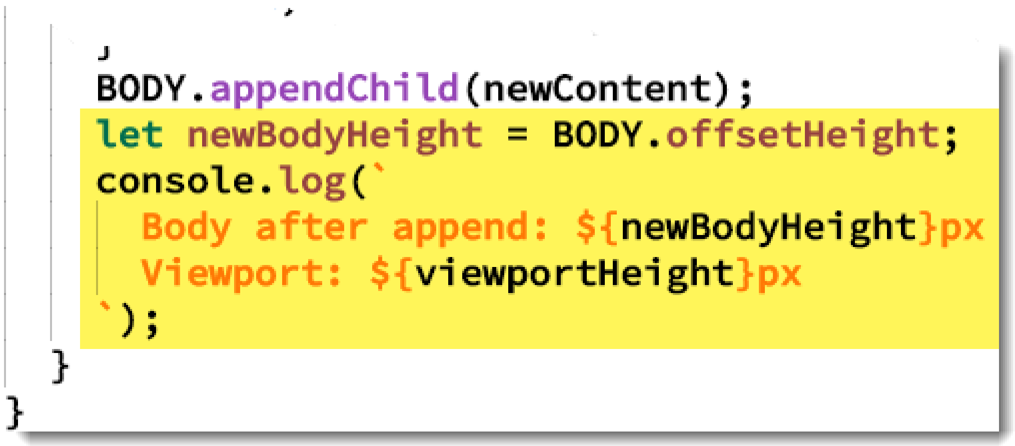
```
function pageBottom(){
  const BODY = document.querySelector('body');
  let bodyHeight = BODY.offsetHeight;
  let viewportHeight = window.innerHeight;
  let scrollDistance = window.pageYOffset;
  let totalSeen = scrollDistance + viewportHeight;
  console.log(`
    Body before append: ${bodyHeight}px
    Viewport: ${viewportHeight}px
  `);
```

4. Save and refresh. Nothing displays in the console because the `pageBottom` function only runs when the page is scrolled. And there's no scrollbar! Add a function call at the very top of the JS to run `pageBottom` on page load. **Do not** refresh yet.



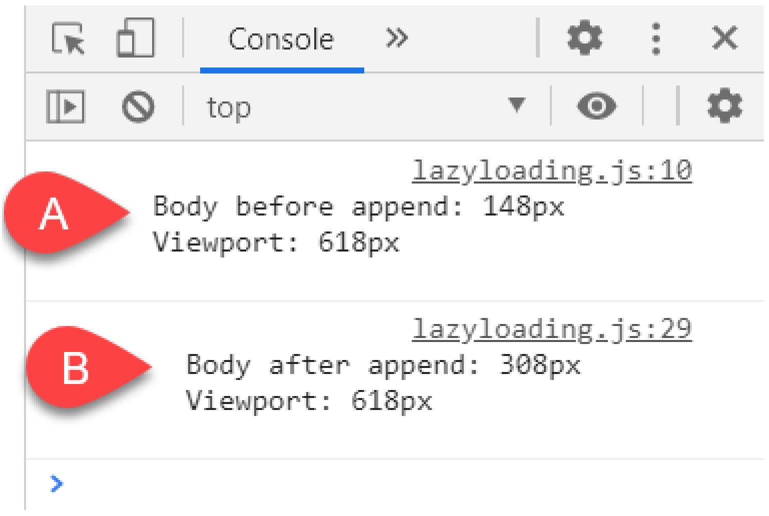
```
pageBottom();
window.addEventListener('scroll', pageBottom);

function pageBottom(){
```

5. Display the new body height (after a paragraph has been appended) in the log. You'll need to create a new variable to get the height after the `appendChild` statement and then log it. Add this new code immediately after the `appendChild` statement towards the end of the function.



```
    }
    BODY.appendChild(newContent);
    let newBodyHeight = BODY.offsetHeight;
    console.log(`
      Body after append: ${newBodyHeight}px
      Viewport: ${viewportHeight}px
    `);
  }
}
```

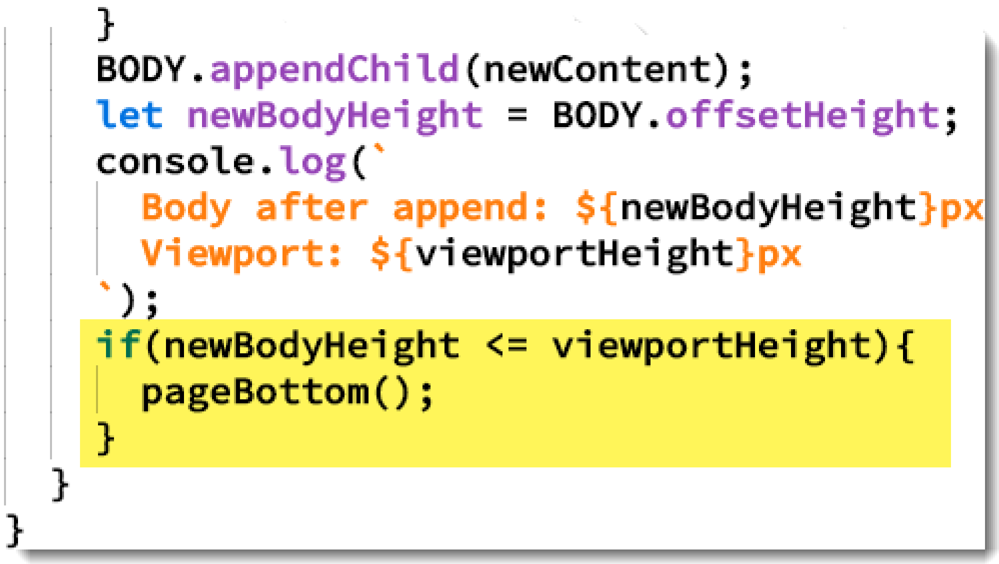
6. Save and refresh. The page shows one of the additional paragraphs loaded (because `pageBottom` ran when the page loaded). The console log shows you the body and viewport heights both before and after the `appendChild`. Your numbers may differ, but they should be similar. (The body should be shorter than the viewport both times.)



```
lazyloading.js:10
Body before append: 148px
Viewport: 618px

lazyloading.js:29
Body after append: 308px
Viewport: 618px
```

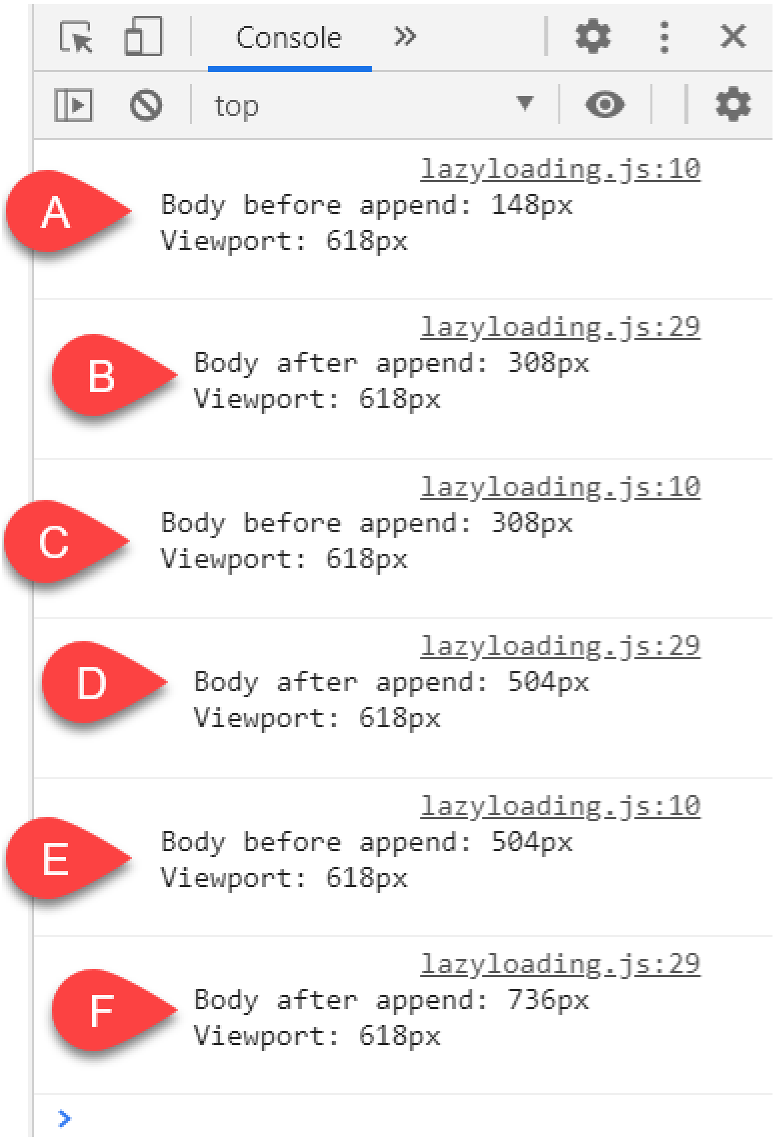
   a. Page loads, `pageBottom` runs, and the body is shorter than the viewport. A new paragraph is retrieved and appended and displays in the browser.
   b. After the append, the body is a little taller, but still shorter than the viewport.

7. Great. But only one paragraph was appended and body is still shorter. We need to keep appending new content until the body is taller than the viewport so the scrollbar will show. Add a test to see if the updated body is shorter. If it is, call `pageBottom`. Add this immediately after the bottom *console.log* block.



```
    }
    BODY.appendChild(newContent);
    let newBodyHeight = BODY.offsetHeight;
    console.log(`
      Body after append: ${newBodyHeight}px
      Viewport: ${viewportHeight}px
    `);
    if(newBodyHeight <= viewportHeight){
      pageBottom();
    }
  }
}
```

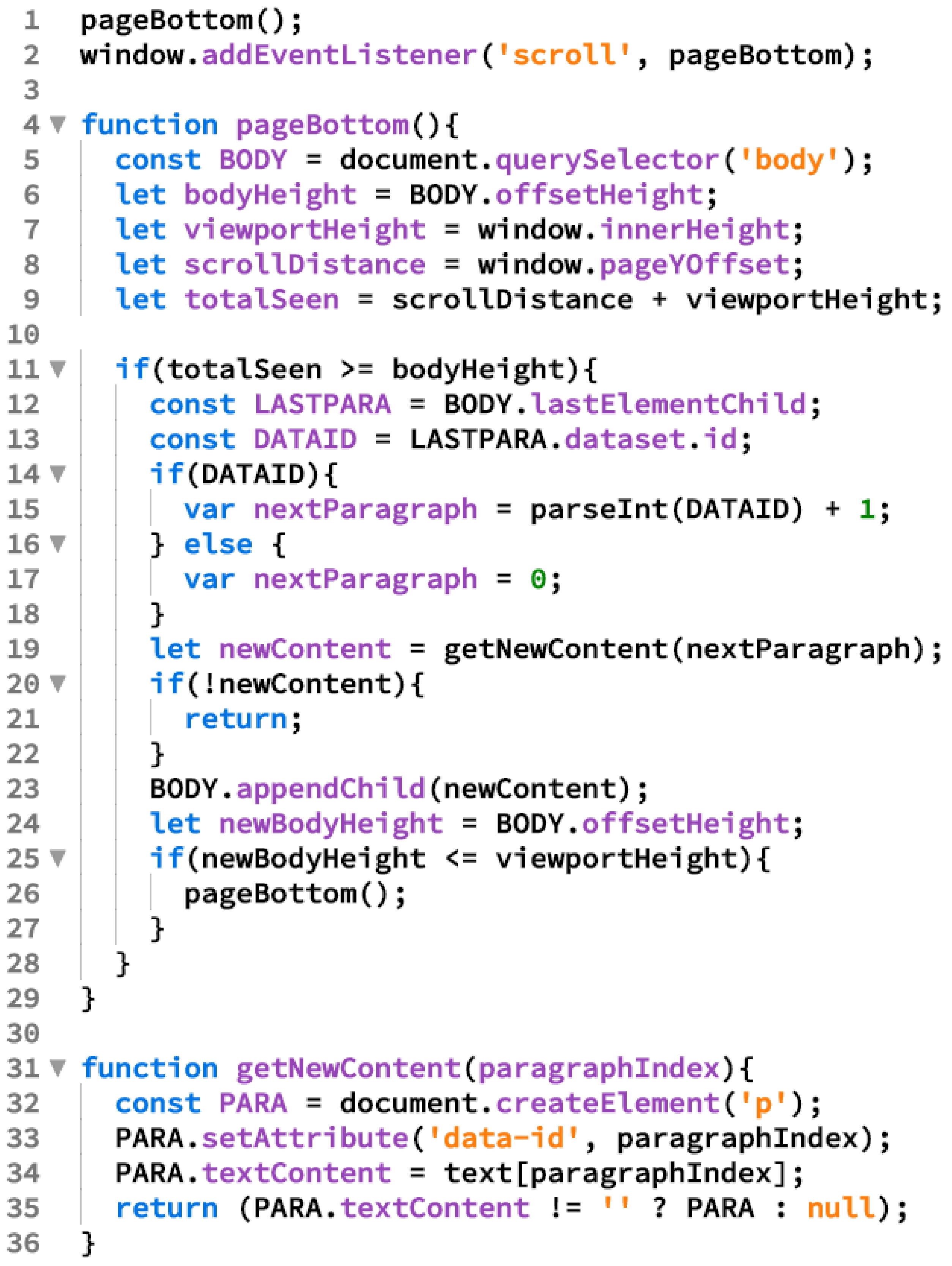
8. Save and refresh. Several new paragraphs have loaded. Here's what happened. Your numbers may differ, but you should get similar results in the console.



```
lazyloading.js:10
Body before append: 148px
Viewport: 618px

lazyloading.js:29
Body after append: 308px
Viewport: 618px

lazyloading.js:10
Body before append: 308px
Viewport: 618px

lazyloading.js:29
Body after append: 504px
Viewport: 618px

lazyloading.js:10
Body before append: 504px
Viewport: 618px

lazyloading.js:29
Body after append: 736px
Viewport: 618px
```

a. Page loads, `pageBottom` runs, and the body is shorter than the viewport. A new paragraph is retrieved and appended and displays in the browser.
b. After the append, the body is a little taller, but still shorter than the viewport. So, `pageBottom` runs again.
c. The current heights are logged, body is shorter, so a new paragraph is retrieved and appended.
d. After the append, the body is a little taller, but still shorter than the viewport. So, `pageBottom` runs again.
e. The current heights are logged, body is shorter, so a new paragraph is retrieved and appended.
f. After the append, the body is taller than the viewport. `pageBottom` isn't called again, but the viewport now displays a scrollbar. The user can scroll down and invoke the function again manually.

9. Delete (or comment out) the *console.log* statements. At this point, the page works as expected. If yours isn't working, compare you code to the final code on the next page.



```javascript
pageBottom();
window.addEventListener('scroll', pageBottom);

function pageBottom(){
  const BODY = document.querySelector('body');
  let bodyHeight = BODY.offsetHeight;
  let viewportHeight = window.innerHeight;
  let scrollDistance = window.pageYOffset;
  let totalSeen = scrollDistance + viewportHeight;

  if(totalSeen >= bodyHeight){
    const LASTPARA = BODY.lastElementChild;
    const DATAID = LASTPARA.dataset.id;
    if(DATAID){
      var nextParagraph = parseInt(DATAID) + 1;
    } else {
      var nextParagraph = 0;
    }
    let newContent = getNewContent(nextParagraph);
    if(!newContent){
      return;
    }
    BODY.appendChild(newContent);
    let newBodyHeight = BODY.offsetHeight;
    if(newBodyHeight <= viewportHeight){
      pageBottom();
    }
  }
}

function getNewContent(paragraphIndex){
  const PARA = document.createElement('p');
  PARA.setAttribute('data-id', paragraphIndex);
  PARA.textContent = text[paragraphIndex];
  return (PARA.textContent != '' ? PARA : null);
}
```

Get Ready for Submission

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