Scroll Effects

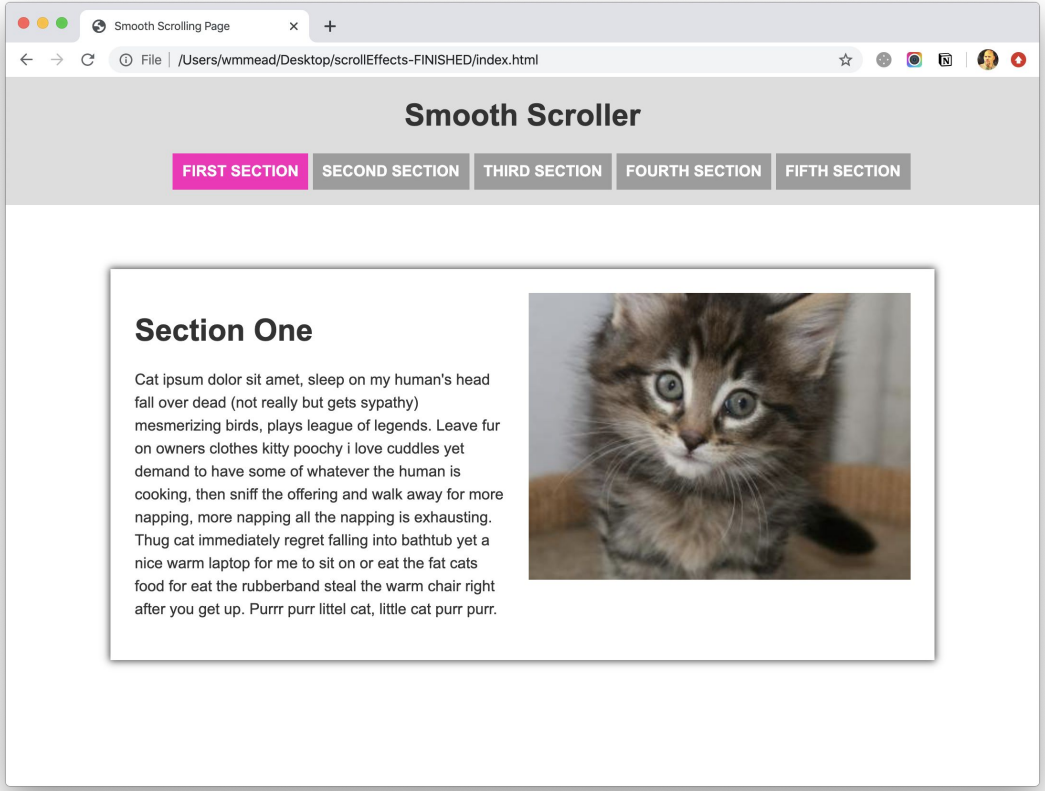
with JavaScript

Two Scripts

This project will contain two scripts. One will provide a smooth scroll effect, when you click the links.

The other will take effect when you manually scroll the page.

This second script will highlight the correct nav element that is currently in view.



HTML and CSS

The HTML and CSS for the page are fairly simple. One thing to note, is that the header is fixed, so it stays on the page as the content scrolls.

The body is given padding of 200px so that the first section on the page is below the header.

Also, the elements we will be working with most are the links in the navigation, and the section elements that make up each area of the page.

```
1  body {
2      font-family: Arial;
3      color: #333;
4      padding-top: 200px;
5  }
6
7  header {
8      background: rgba(217, 217, 217, 0.9);
9      width: 100%;
10     position: fixed;
11     top: 0;
12 }
```

Starting the Smooth Scroll

To get the smooth scroll working, you need to add a click handler to each of the links in the navigation, so that when one of those links is clicked, the scroll event to the corresponding section will be handled.

```
const navLinks = document.querySelectorAll('nav ul li a');

navLinks.forEach(function(eachLink) {
    eachLink.addEventListener('click', smoothScroll);
});
```

Typing from the Outside to Inside

Be sure you type from the outside to the inside, rather than from left to right, as you normally would. This will help you understand the code you are typing.

```
navLinks.forEach();
```

navLinks is an array, and forEach() is an array method that takes a callback function. Add the callback function next.

```
navLinks.forEach( function(){} );
```

The callback function takes an input variable that stands if for each element in the array...

```
navLinks.forEach( function(eachLink){} );
```

Add the event listener for each link in the array.

```
navLinks.forEach( function(eachLink){
    eachLink.addEventListener();
} );
```

eachLink is listening for a click, then it will run a function called smoothScroll

```
eachLink.addEventListener('click', smoothScroll);
```

Add the smoothScroll Function

Next add the smoothScroll() function, and pass in the event, so that you can prevent the default behavior, which is just to pop the browser to that section of the page.

```
const navLinks = document.querySelectorAll('nav ul li a');

navLinks.forEach(function(eachLink) {
    eachLink.addEventListener('click', smoothScroll);
});

function smoothScroll(event) {
    event.preventDefault();

}
```

Getting the Target

If you look at the html, the links in the anchor tags match the IDs for the sections below. Use two variables to get these two items.

```
function smoothScroll(event) {
    event.preventDefault();

    const targetID = event.target.getAttribute('href');
    const targetAnchor = document.querySelector(targetID);

}
```

How Many Pixels from the Top?

There are two pieces of JavaScript that make this script work. The first is this method called getBoundingClientRect().

This method has properties which will tell you where the top, bottom, left, or right of any given element are in relation to the window.

Here you are concerned with the top property, because that is what you want to be scrolling to.

Put this in a console.log to see what you get as shown on the next slide...

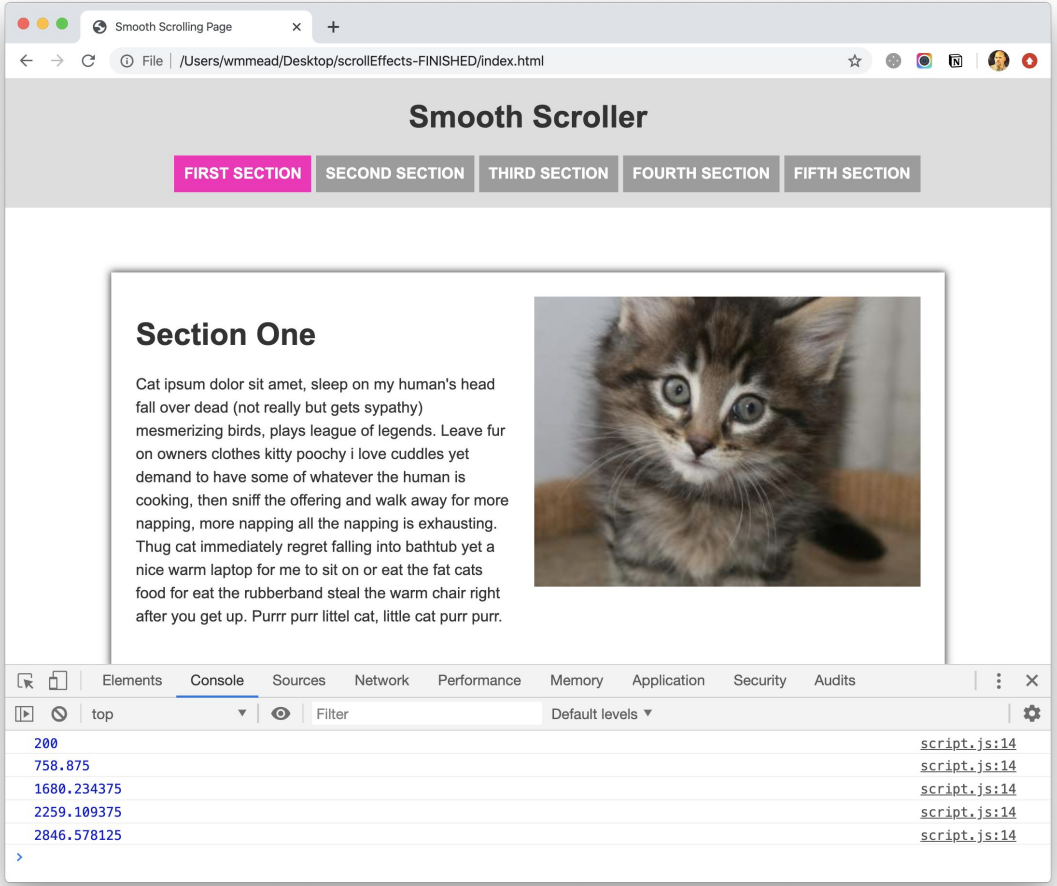
In jQuery, you have the offSet() helper method to do the same thing.

```
const targetID = event.target.getAttribute('href');
const targetAnchor = document.querySelector(targetID);

console.log(targetAnchor.getBoundingClientRect().top);
```

Tops of sections

You can see, when you click on the links that the distance to the tops of those related sections show up in the console.log.

Next, you will take that value, round it down (don't need the decimal places), subtract 200px from it (to leave room for the header at the top of the page), and assign it to a variable.



Assigning the Distance

Rounding gets rid of the decimals, leaving only whole numbers. Subtracting 200 will put the element below the header, rather than at the top of the window, behind the header.

Next comes the second piece of JavaScript magic, the actual smooth scrolling.

```
const targetID = event.target.getAttribute('href');
const targetAnchor = document.querySelector(targetID);

const originalTop = Math.floor(targetAnchor.getBoundingClientRect().top) - 200;
```

The scrollBy Method

The scrollBy() method is what provides the smooth scroll magic. This method takes an object as an argument, and that object has three properties: top, left and behavior.

Add a console.log for the originalTop variable and watch it change as you click the links in either Chrome or Firefox.

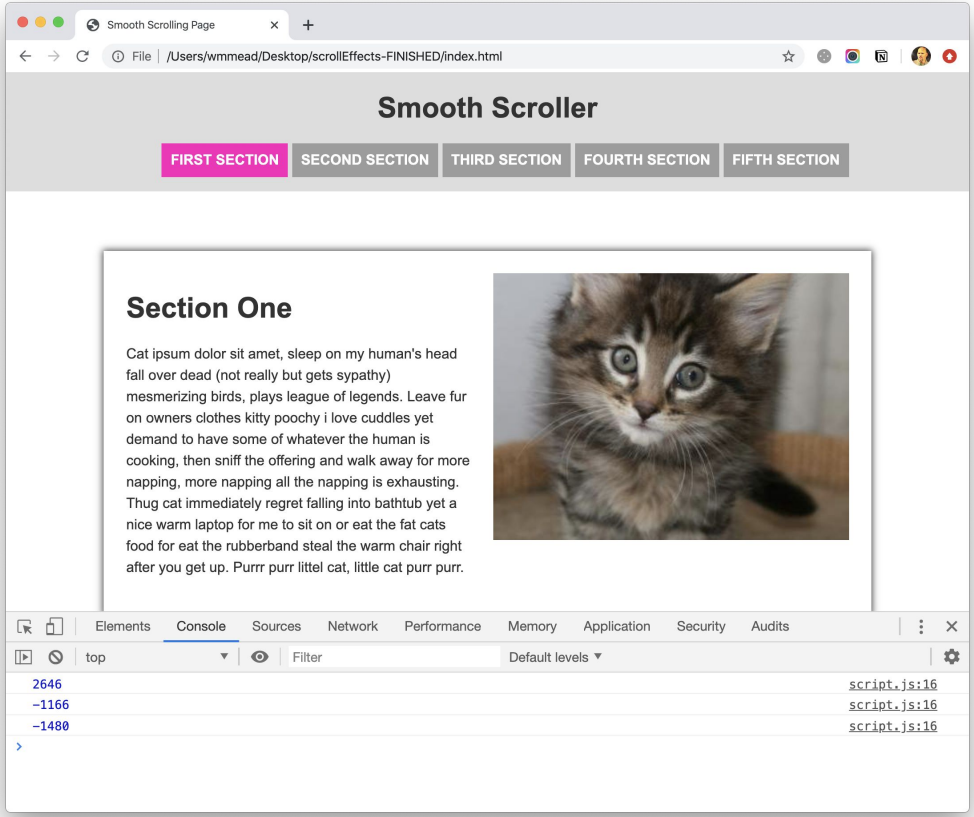
```
const originalTop = Math.floor(targetAnchor.getBoundingClientRect().top) - 200;
window.scrollBy({ top: originalTop, left: 0, behavior: 'smooth' });
console.log(originalTop);
```

Testing in Chrome

You can see in the console, the amount of pixels that the page is traveling is either a positive number, when you go down the page, or a negative number when you go up the page.

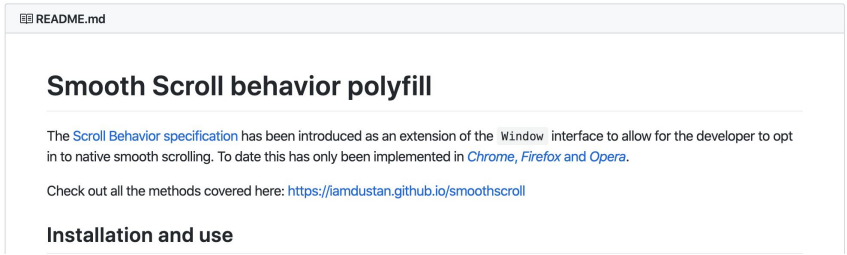
You might notice that this is not working properly in Safari. Safari does not support the behavior property yet.

However there is a fix.



Polyfill for Safari

Polyfill is a term coined by a developer named Remy Sharp to describe code written to patch a browser that doesn't support a particular feature.


README.md

Smooth Scroll behavior polyfill

The Scroll Behavior specification has been introduced as an extension of the `Window` interface to allow for the developer to opt in to native smooth scrolling. To date this has only been implemented in *Chrome*, *Firefox* and *Opera*.

Check out all the methods covered here: https://iamdustan.github.io/smoothscroll

Installation and use

```
    <script src="smoothscroll.js"></script>
    <script src="script.js"></script>
</body>
</html>
```

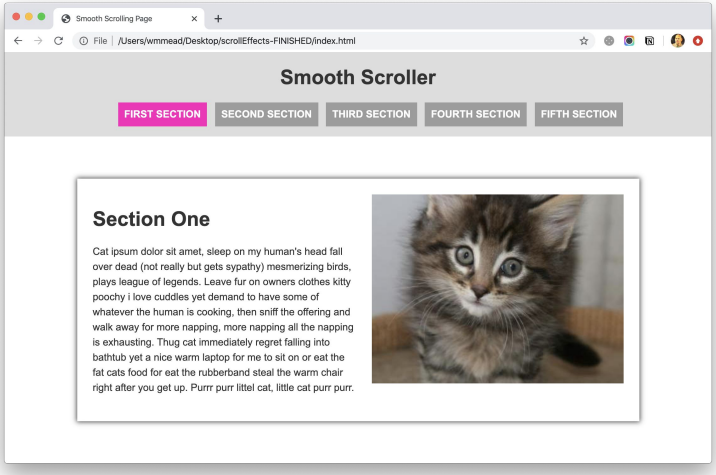
We used to use a lot of polyfills, back when JavaScript implementations differed a lot from browser to browser. They are needed less often now.

A developer with the username iamdustin on GitHub wrote this one for Safari.

All you have to do is link the file. I already put it in the start folder. Link it before your script and it will magically work in Safari too.

The Smooth Scroll Script is Finished

That is it for the smooth scroll script. This script does not have any features for timing the scrolling or providing specific easing, which is a little unfortunate, but to that, you would need more work.



The next part of this project will accomplish the same task you accomplished using jQuery.

I think you will find it interesting that the script is just a little different in places, but the strategy is the same. The syntax is not really that much harder.

In the end, not needing jQuery to do this is a big bonus and you could use this part of this script for lots of different things.

Checking for Load

As with the jQuery version, the entire script is going to run inside a window load function. The syntax for that is a little different in plain JS, but not that different.

Notice in this version, the counter and prevCounter start at 1 instead of 0.

This has to do with the method you will use to change the styling for the links in the navigation.

In this version the articles will be numbered 1, 2, 3, 4 & 5. In the jQuery version they were numbered 0, 1, 2, 3 & 4.

```
window.addEventListener('load', function() {
    const posts = document.querySelectorAll('section');
    let postTops = [];
    let pagetop;
    let counter = 1;
    let prevCounter = 1;
    let doneResizing;
} );
```

The getBoundingClientRect Again

The getBoundingClientRect() method is back for round two, in order to find the top of each section. Put it into a console.log for the first post in the posts array and see what you get.

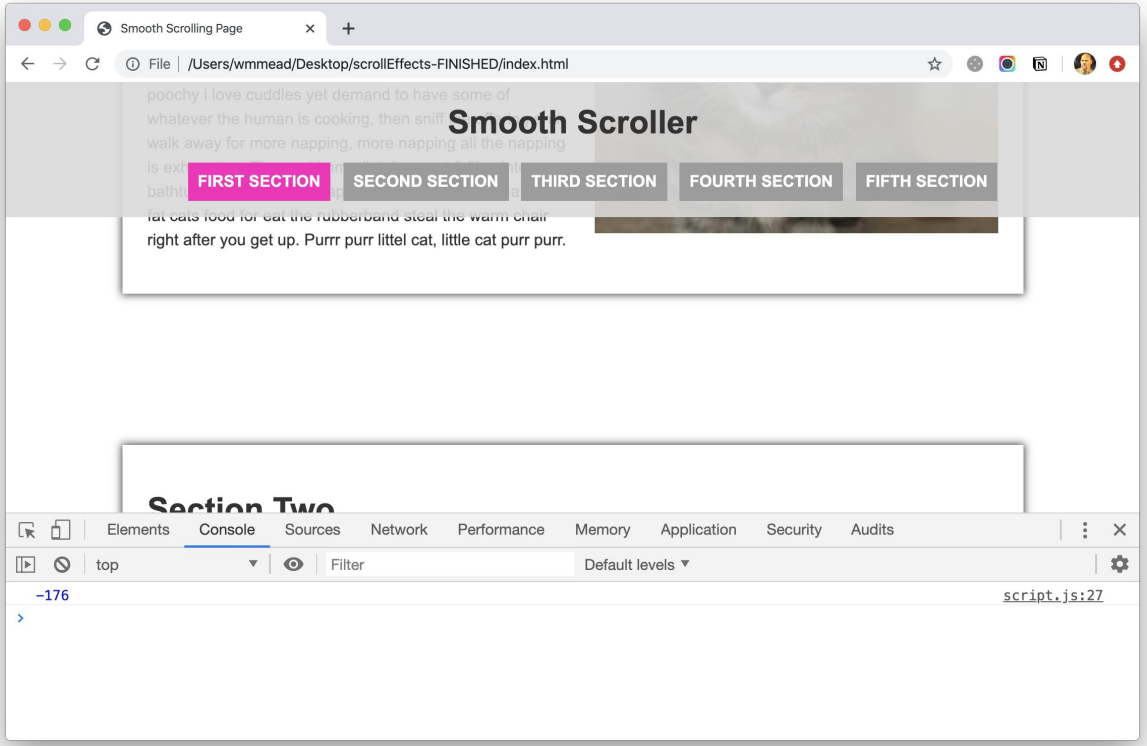
Again, this is instead of the jQuery function offSet().top, and will report how many pixels down the page the first section is.

```
window.addEventListener('load', function() {
    const posts = document.querySelectorAll('section');
    let postTops = [];
    let pagetop;
    let counter = 1;
    let prevCounter = 1;
    let doneResizing;

    console.log( posts[0].getBoundingClientRect().top );
} );
```

Scroll Down and Click Refresh

If you do a fresh load of the page with the page scrolled to the top, you should get 200. However, if you scroll down the page, and refresh, you might get a different number.

We want it to always be 200, because that post is always 200px from the top of the page, if not 200px from the top of the window.



Add window.pageYOffset

Adding window.pageYOffset to the value will give us a consistent value. pageYOffset is the amount of pixels the window has scrolled past the viewport, and will therefore give us a consistent number each time.

Test it in the browser, you will see no matter where the page is scrolled to, you get 200 for the first post.

Next, round the distance, and put all of them in into the postTops array.

```
console.log(posts[0].getBoundingClientRect().top) + window.pageYOffset;
```

Use forEach() and push()

Use the forEach() and push() array methods to add the top of each post into the postTops array and then add a console.log call to see what is inside postTops

Don't forget to write this from the outside to the inside, like you did before, to make sure you understand each step.

This step is very similar to what you did with the jQuery version. The JS syntax is a little different.

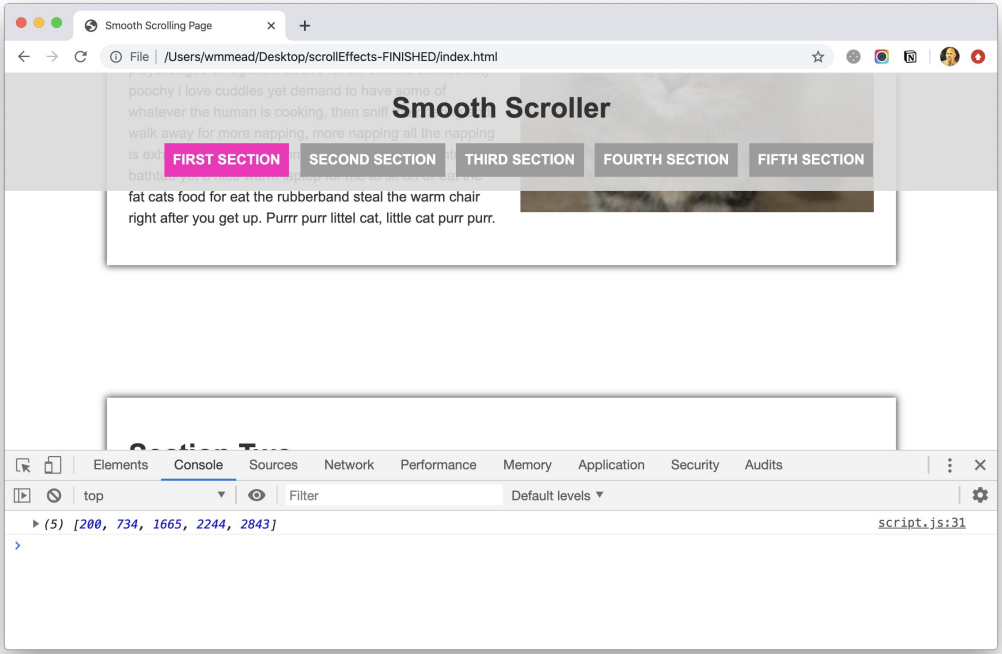
```
posts.forEach(function(post) {
    postTops.push(Math.floor(post.getBoundingClientRect().top) + window.pageYOffset);
});

console.log(postTops);
```

The Tops of Each Section (Post)

The console.log shows the array with the number of pixels down the page that the tops of each post are.

Reminder: The reason why you need the page load event handler is that without it, you might get different numbers for the tops of the posts, which would really mess things up.

You might get different numbers because the tops of the posts might get calculated BEFORE the images load, and the images change the height of some of the sections.



Add the Scroll Event Listener

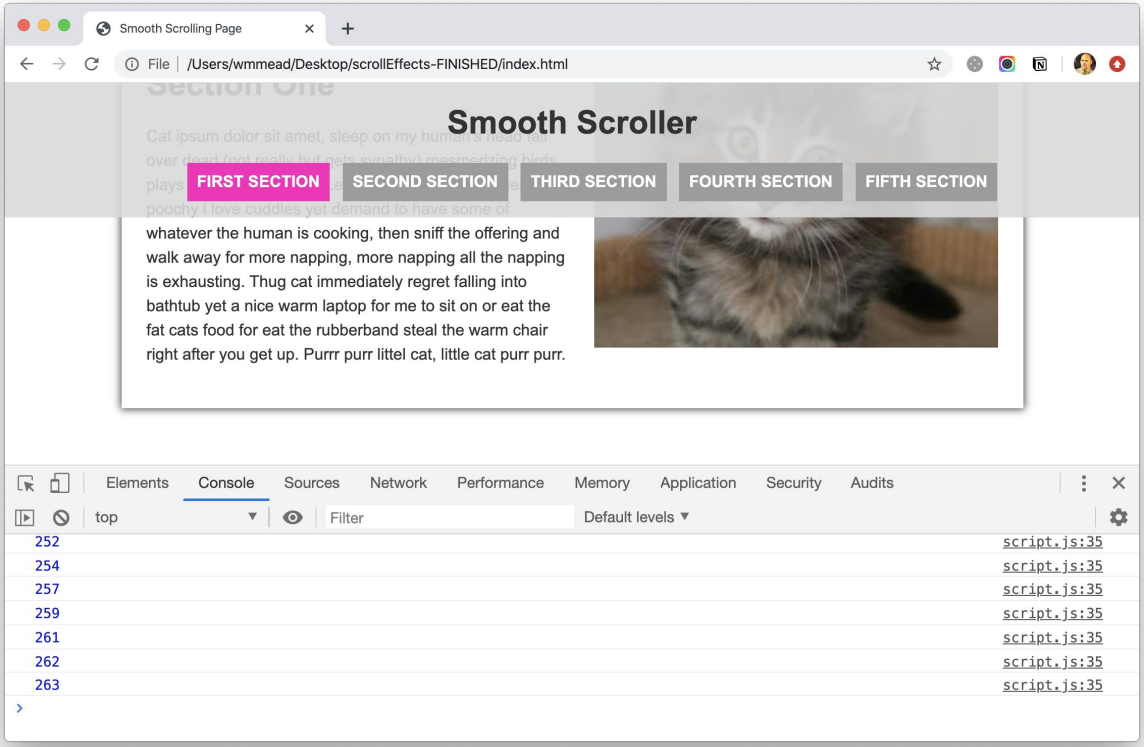
```
window.addEventListener('load', function() {
    const posts = document.querySelectorAll('section');
    let postTops = [];
    let pagetop;
    let counter = 1;
    let prevCounter = 1;
    let doneResizing;

    posts.forEach(function(post) {
        postTops.push(Math.floor(post.getBoundingClientRect().top) + window.pageYOffset);
    });
    console.log(postTops);

    window.addEventListener( 'scroll', function() {
        pagetop = window.pageYOffset;
        console.log(pagetop);
    } );
} );
```

Check the Console for pagetop

In the console log, notice that as you scroll down the page, pagetop is reporting how far down the page you have gone.

Also notice, depending on how fast you scroll, pixels might be skipped. The fact that you can't be sure a particular pixel will be hit on scroll makes scroll capture scripts a little challenging.

This part is the same as in the jQuery version.



Add 250 Pixels

Add 250 to the scrolling number because you want to actually watch for a bit down the page, after the header, not the very top of the window.

```
window.addEventListener('scroll', function() {
    pagetop = window.pageYOffset + 250;
    console.log(pagetop);
});
```

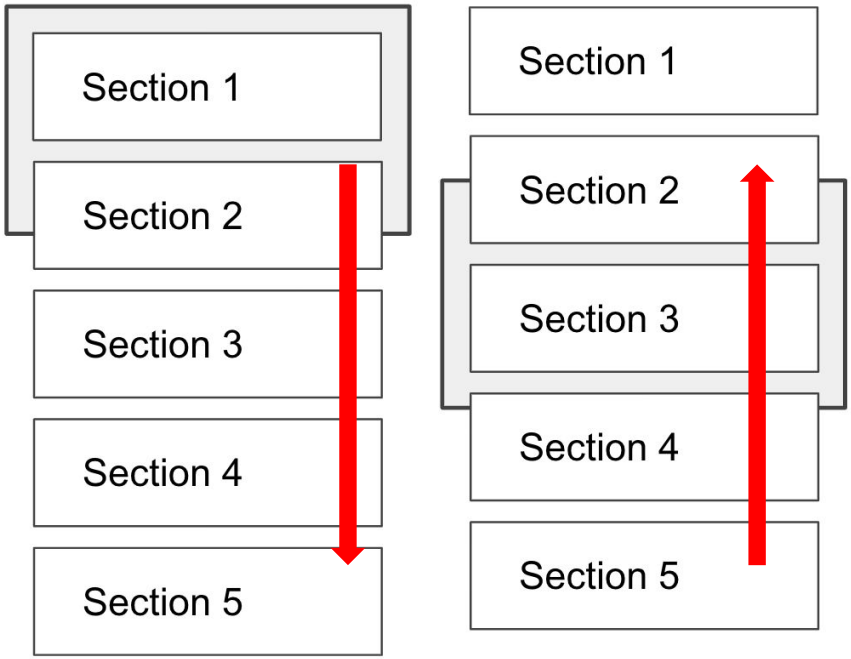
The basic strategy for the script is the same as before, but it bears repeating.

JS will be watching this pagetop number.

When the pagetop number is greater than the number for the top of the second section, it must mean the second section is on the screen.

When the pagetop number is greater than the number for the top of the third section, it must mean the top of the third section is on the screen, etc..

Understanding the Options



The options you are keeping track of have not changed.

Either you are at the top of the page scrolling down, or you are somewhere further down the page scrolling up.

The Magic Conditionals

The if / if else statements shown here handle the two conditions from the previous page, and are the real magic of this script.

On the next slides each one will be explained.

```
window.addEventListener( 'scroll', function() {
    pagetop = window.pageYOffset + 250;
    //console.log(pagetop);

    if (pagetop > postTops[counter]) {
        counter++;
        console.log(`scrolling down ${counter}`);

    } else if (counter > 1 && pagetop < postTops[counter-1]) {
        counter--;
        console.log(`scrolling up ${counter}`);
    }
} ); // end window scroll function
```

The first if statement

The first if statement handles the most common situation. If the user is scrolling down the page, and the pagetop number becomes higher than the number representing the top of the next post, increment the counter.

This way, the counter is 1 while scrolling through the first section, then 2 while scrolling through the next section, then 3 while scrolling through the third section, etc.

It is important to note that the console.log here fires just ONE TIME for each section. As soon as you scroll into that section, the counter gets incremented, and now it is looking for the next section.

This makes this script efficient.

```
if (pagetop > postTops[counter]) {
    counter++;
    console.log(`scrolling down ${counter}`);
```

The Else If Statement

This second statement is for when the user is in the middle of the page, past the first post, and can scroll back up the page. In other words, the counter is greater than 1.

If that is the case, and pagetop becomes a number lower than the number that represents the top of the current post, decrement the counter.

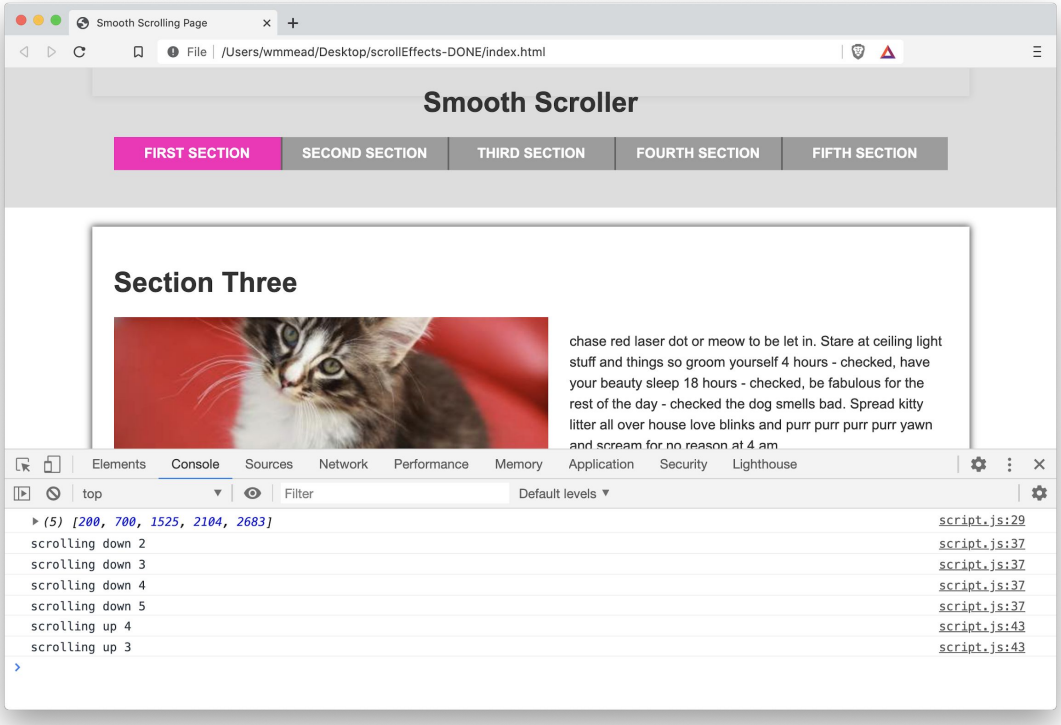
Again, this fires just ONE TIME, as you scroll into the previous section.

```
else if (counter > 1 && pagetop < postTops[counter-1]) {
    counter--;
    console.log(`scrolling up ${counter}`);
}
```

Watching the Console

Watch the console for when these calls to the console.log fire. Notice in this case the number are 1, 2, 3, 4 or 5.

In the jQuery version they were 0, 1, 2, 3 or 4.



Changing the Nav Links

As you did in the jQuery version of the script, you will check the value of counter against the prevCounter value and if they are NOT the same, remove all the classes from the links at the top of the page.

The syntax in plain JavaScript is a little different, but you can use this nice little forEach() method to loop through the array and remove the class attribute from all the links.

```
if (counter != prevCounter) {

    navLinks.forEach(function(eachLink) {
        eachLink.removeAttribute('class');
    });

}
```

Changing the Next Link

Then of course add the class to the correct link. This is finally where you get to see why the counter changed from starting at 0 to starting at 1. The nth-child() selector is indexed to start at 1 so nth-child(1) gets the first list item, etc.

In the jQuery version of the script, you used the eq() method to get the anchor tag and that method is zero indexed, so .eq(0) gets the first link and eq(1) gets the second link, etc.

```
if (counter != prevCounter) {

    navLinks.forEach(function(eachLink) {
        eachLink.removeAttribute('class');
    });

    const thisLink = document.querySelector(`nav ul li:nth-child(${counter}) a`);

    thisLink.className = 'selected';
    prevCounter = counter;
}
```

Responsive Concerns

Just like with the jQuery version of the script, you are down to fixing the edge case scenarios of if someone resizes the window, or refreshes the window or some combination of these two events.

To deal with this, you will follow the exact same procedure as before, the syntax is just a little different.

Add the window resize event listener function and once again use the clearTimout() method to make sure it fires only when the user is done resizing the window. Then run the resetPagePosition() function.

```
    } ); // end window scroll function

    window.addEventListener('resize', function() {
        clearTimeout(doneResizing);
        doneResizing = setTimeout( function(){

            resetPagePosition();

        }, 500);
    });

} ); // end window load function
```

Reset Page Position Function

Here is the function. It is almost the same as the jQuery version, just using plain JS syntax.

```
function resetPagePosition(){
    postTops = [];
    posts.forEach(function(post) {
        postTops.push(Math.floor(post.getBoundingClientRect().top) + window.pageYOffset);
    });

    const pagePosition = window.pageYOffset + 250;
    counter = 0;

    postTops.forEach( function(post){ if( pagePosition > post ){ counter++; } } );

    navLinks.forEach(function(eachLink) { eachLink.removeAttribute('class'); });

    const thisLink = document.querySelector(`nav ul li:nth-child(${counter}) a`);
    thisLink.className = 'selected';
}
```

Add the Function at the Top

As with the jQuery version, comment out, or just remove the forEach() method putting the number for the tops of each section into the postTops array, and just add a call to the fancy new resetPagePosition() function again.

```
window.addEventListener('load', function() {
    const posts = document.querySelectorAll('section');
    let postTops = [];
    let pagetop;
    let counter = 1;
    let prevCounter = 1;
    let doneResizing;

    /*posts.forEach(function(post) {
        postTops.push(Math.floor(post.getBoundingClientR
    });
    console.log(postTops);*/

    resetPagePosition();

    window.addEventListener( 'scroll', function() {
```

Summary

This is a GREAT script, especially the second part. With it you could watch as sections scroll into view and apply or remove classes from ANYTHING on the page.

It is very flexible and you could use it for a lot of ideas.