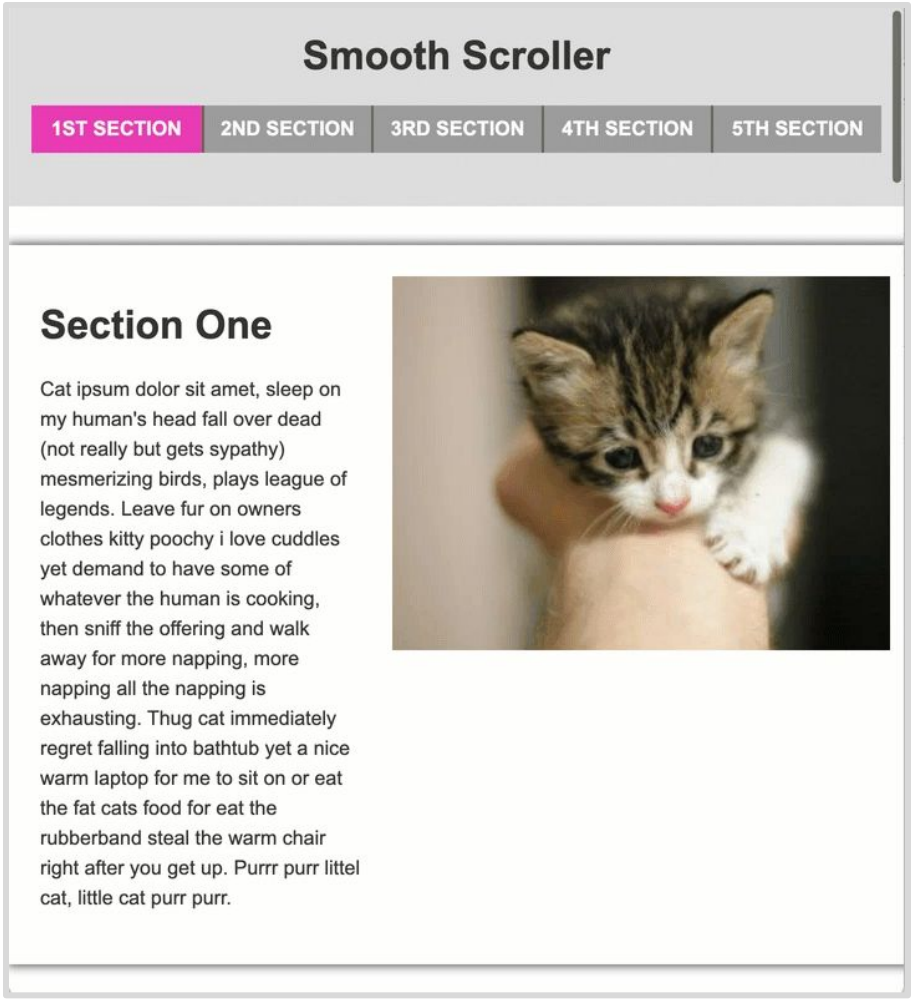
Page Location Tracking

Using jQuery

The Scroll Event

Working with the scroll event can be challenging.

In this script, you will take the same page with the smooth scroll effect on the links and also highlight the correct links if someone is scrolling down the page.



Window Loaded and Scrolling

After the smoothscroll script, on the JavaScript file, add this window load function with the window scroll function inside it.

This will make sure all assets, such as images have loaded before anything else happens.

The $(window).scroll() listener will listen for when the window is scrolling.

```
$(window).on( 'load', function(){

    $(window).scroll( function(){


    } );
} );
```

The Crux of the Problem

Add these three variables, then add the code in the window scroll event listener.

Open the file in your browser and turn on the inspector and watch the console as you scroll the page.

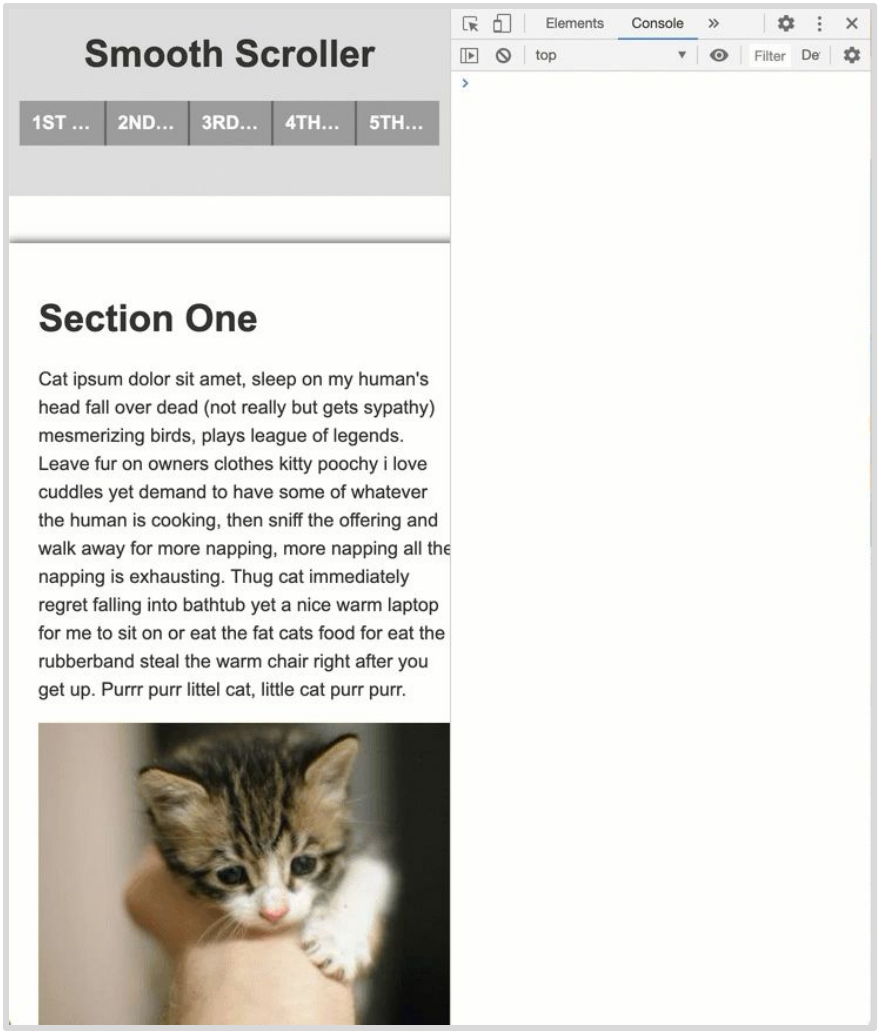
```
$(window).on( 'load', function(){

    var posts = $('section');
    var pageTop;
    var postPos;

    $(window).scroll( function(){
        postPos = $(posts[0]).offset().top;
        pageTop = $(window).scrollTop();
        console.log(`${pageTop} and ${postPos}`);
    } );

} );
```

Scrolling...

scrollTop() is a jQuery function that will report how many pixels of the window are above the top of the window. As you scroll down, that number gets bigger.

offSet() is a jQuery function that will tell you information about a given element. offSet().top will specifically provide you with the position of the top of the element in relation to the top of the page.

These two pieces of information together can be used to determine whether a certain element is in the viewport or not.

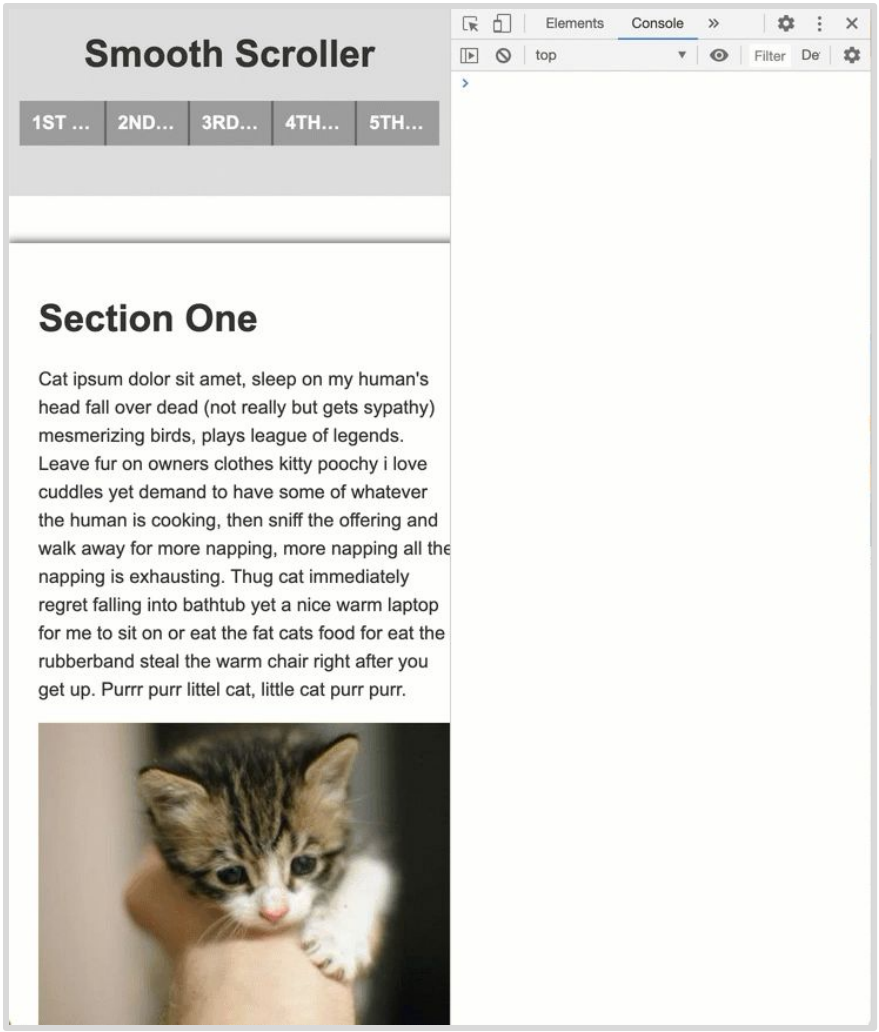


More on Scrolling...

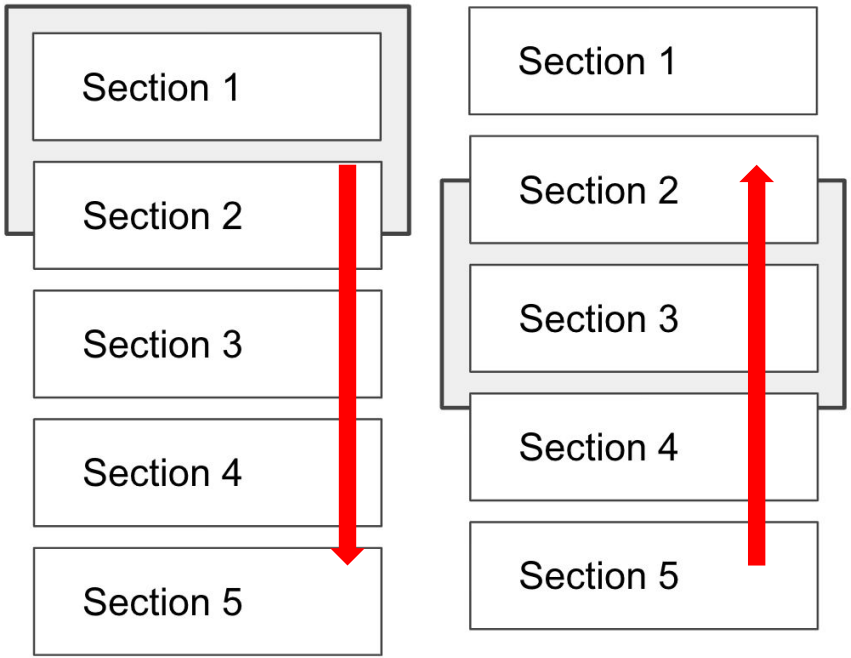
Notice the first number changes as you scroll, but the second does not.

Notice that as you scroll, not EVERY pixel is accounted for, depending on how fast you scroll. If you scroll faster, more pixels are missed.

This means you can not write a script that says “if pixel X has scrolled past this point...” because, while scrolling you can’t be guaranteed to register pixel X at all!



Strategy



There are two things that could be happening as someone views the page.

They could be scrolling down the page, or they could be somewhere other than the top of the page and scrolling up.

Setting up the If and Else If Statements

Inside the window load function, add a variable for a counter and set it to 0.

Then add if and else if statements for each of the scenarios from the previous slide.

You will tackle the if statement first, as this is the most common scenario (that the user is scrolling down the page).

How will you know if the user has scrolled into the next section?

```
var posts = $('section');
var pageTop;
var postPos;
var counter = 0;

$(window).scroll( function(){

    if(/*scrolling down into the next section*/){
       // increment counter!
    }
    else if(/*scrolling up into the pevious section*/){
        // decrement counter!
    }
```

Getting the Tops of All the Sections

Part of what you need to know is, where on the page, are all the tops of each section of the page. This code will tell us that.

Make an array called postTops. It starts empty. Then using the jQuery each method, push the offSet().top of each post (or section) into the array.

Console logging out the array will tell you how many pixels each section is from the top of the page.

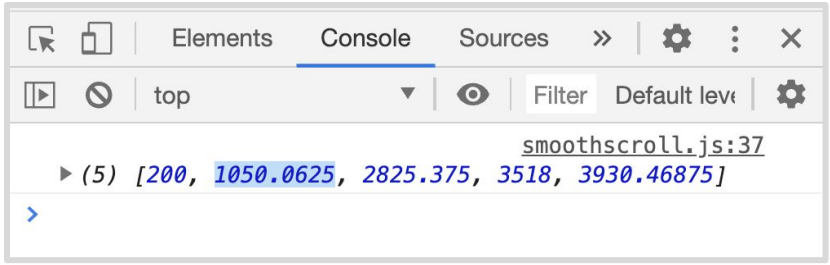
(You may have to comment out the if statements, since those are not finished yet).

```
$(window).on( 'load', function(){

    var posts = $('section');
    var pageTop;
    var counter = 0;

    var postTops = [];

    posts.each( function(){
        postTops.push( $(this).offset().top );
    } );

    console.log(postTops);

    $(window).scroll( function(){
```

Rounding Down

The console.log is reporting these numbers as how many pixels down the page the tops of each section are.

You just need whole pixels, so round down the numbers to the nearest whole pixel.




```
posts.each( function(){
    postTops.push( Math.floor($(this).offset().top ));
} );
```

Write the If Statement

While the window scrolls, you want the variable, pagetop, To show the number of pixels have scrolled past the top of the window, plus 210 pixels. This is because the page started 200 pixels down, and now you need to account for that, plus a little more.

If that number that pagetop reports is larger than the number of where the next section on the page is, then that next section MUST be just under the header on the page.

```
$(window).scroll( function(){

    pagetop = $(window).scrollTop() + 210;

    if(pagetop > postTops[counter+1]){
        counter++;
        console.log(`scrolling down ${counter}`);
    }
```

Notice as soon as this if statement fires as true, the counter gets incremented and the statement will not be true again, until the next section scrolls into view.

This makes this script very efficient.

Write the Else If Statement

Here is the whole window scroll function so far.

In the else if you will check to see if the counter is currently larger than zero, which would mean you have scrolled down the page to at least one or more sections below the first section.

AND if the number of pixels reported by pagetop is less than the section we were in, THEN the user MUST have scrolled UP into a previous section.

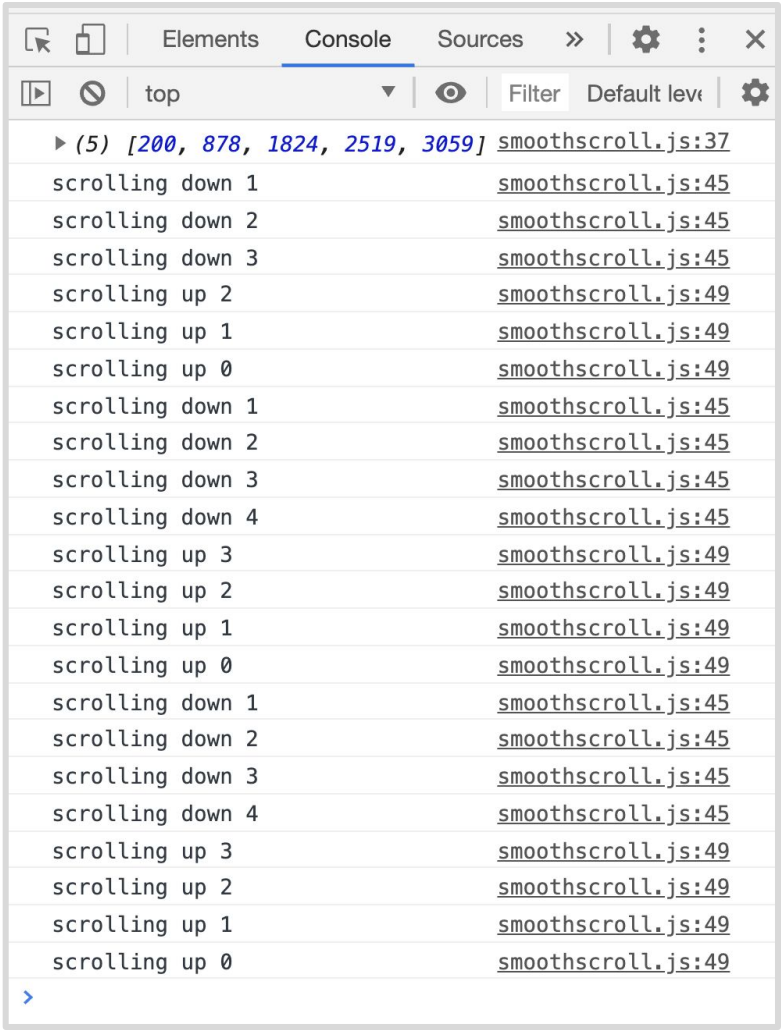
Test it out!

```
$(window).scroll( function(){

    pagetop = $(window).scrollTop() + 210;

    if(pagetop > postTops[counter+1]){
        counter++;
        console.log(`scrolling down ${counter}`);
    }
    else if (counter > 0 && pagetop < postTops[counter]) {
        counter--;
        console.log(`scrolling up ${counter}`);
    }
} );
```

Checking Your Work

If you watch the console log, you will notice that it is ONLY reporting a change when a new section comes into place. You don't have console.log messages firing every time the page scrolls a tiny bit. This is a HUGE accomplishment.

The last thing to do is to change the styling on the links as the user scrolls down the page. But the hard part is done. The rest is easy.



Add Two New Variables

To finish the script, you will add two new variables at the top of the page.

AllLinks is just a shortcut for getting all the links in the navigation.

The variable prevCounter will be used to compare with the variable counter, so you know when you have switched sections.

```
$(window).on( 'load', function(){
    var allLinks = $('nav ul li a');
    var posts = $('section');
    var pageTop;
    var counter = 0;
    var prevCounter = 0;
```

One Last If Statement

Add this one last if statement.

If the variables counter and preCounter are NOT the same it must mean the user has scrolled into a new section.

If that has happened, remove all the classes from the nav links and and then add the class “selected” to the correct link.

Finally, set the preCounter to match counter again, so it is ready for the next change.

```
$(window).scroll( function(){

    pagetop = $(window).scrollTop() + 210;

    if(pagetop > postTops[counter+1]){
        counter++;
        console.log(`scrolling down ${counter}`);
    }
    else if (counter > 0 && pagetop < postTops[counter]) {
        counter--;
        console.log(`scrolling up ${counter}`);
    }
    if( counter != prevCounter ){
        $(allLinks).removeAttr('class');
        $("nav ul li a").eq(counter).addClass('selected');
        prevCounter = counter;
    }
} );
```

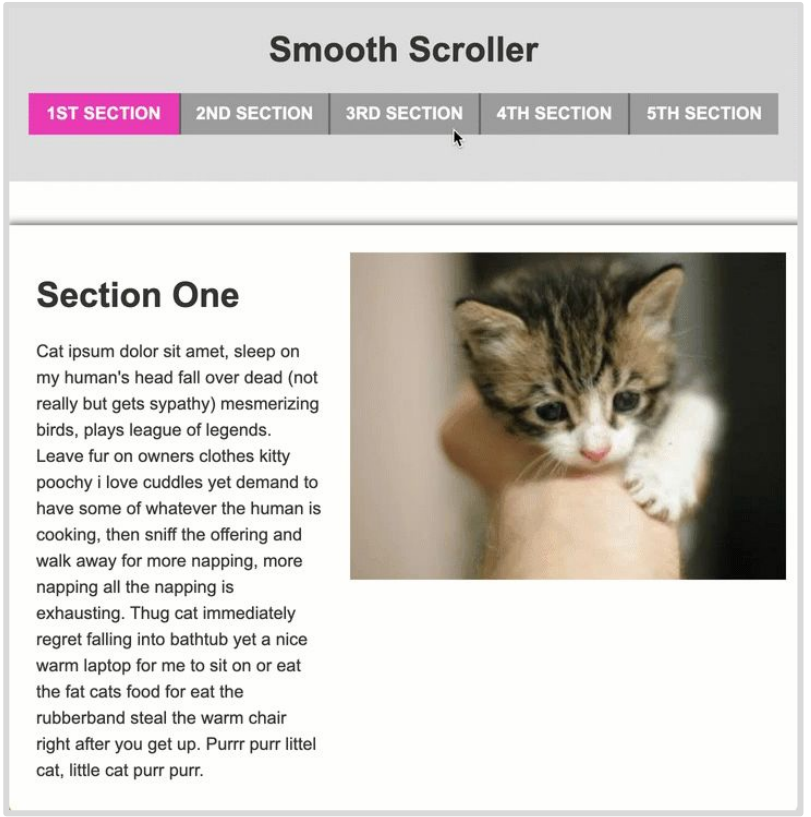
Working!

Both scripts should work now. You can click the menu items, and the page will go to the correct spot and highlight the correct menu item.

Or you can scroll down or up the page, and the correct links will be highlighted.

The script is responsive ready, in that if you load the page in a narrow browser, it will correctly calculate the tops of the sections, or if you load the page in a wide browser it will do the same.

There are a few edge cases to fix though.

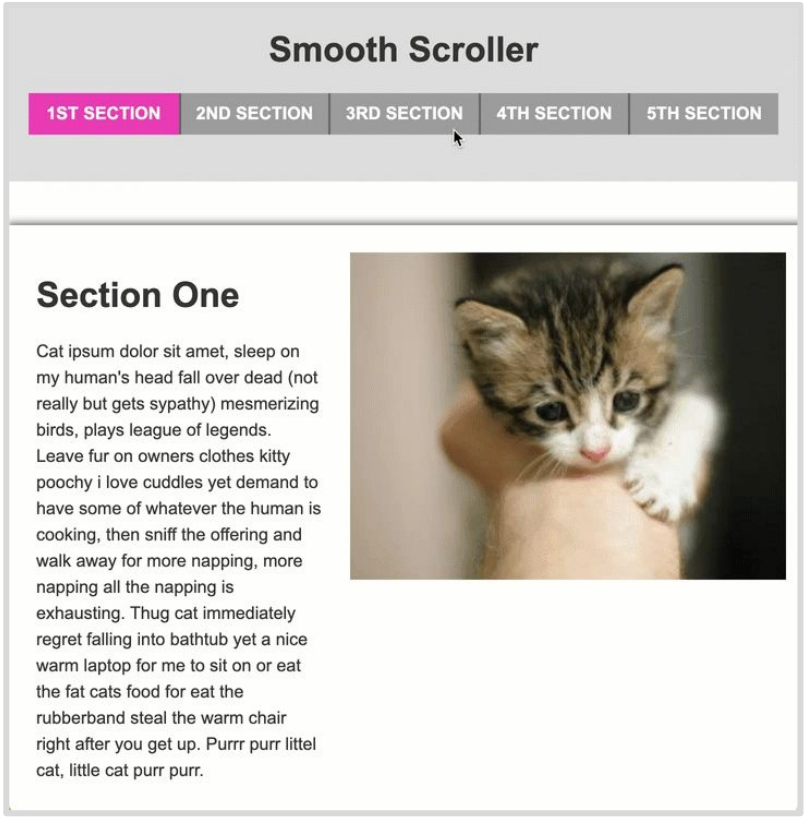


Edge Case Issues

There are two edge case issues that will cause problems:

1. If the user resizes the browser window, the numbers for where all the posts start get screwed up and it won't work properly.
2. If the user refreshes the page while scrolled down the page, the counter gets set back to zero, and the wrong section is highlighted.

Let's fix these issues.



Window Resize Function

Towards the bottom of the script, after the end of the window scroll function, but still inside the window load function, add this window resize function.

You will notice, if you view in the browser and open the console that the log will fire as you are resizing the window.

In this case, you want to know when the window has stopped resizing.

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

    $(window).on("resize", function(){

        console.log("resizing window");

    });

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

Add One More Variable

Add one more variable at the top of the page, then set the code in the resize function to match the snippet on the right.

While the user is resizing the window, the setTimeout is cleared, so nothing will get logged.

Then as soon as the user stops resizing the window for half a second, the setTimeOut runs and fires the console log. Try it!

```
$(window).on( 'load', function(){
    var allLinks = $('nav ul li a');
    var posts = $('section');
    var pageTop;
    var counter = 0;
    var prevCounter = 0;
    var doneResizing;
```

```
$(window).on("resize", function(){

    clearTimeout(doneResizing);
    doneResizing = setTimeout( function(){
        console.log("done resizing");
    }, 500);

});
```

Resetting the Post Tops

If you add the code from the top of the window load script here, you can recalculate the tops of each post (section) after the page has been resized. That solves half the problem.

Check the console to see where the post tops are now, and how far down the page you have actually scrolled.

```
$(window).on("resize", function(){

    clearTimeout(doneResizing);
    doneResizing = setTimeout( function(){
        postTops = [];
        posts.each( function(){
            postTops.push( Math.floor($(this).offset().top ));
        } );

        console.log(postTops);
        console.log($(window).scrollTop()+210);
    }, 500);
});
```

Resetting the Counter

The counter needs to be reset to the correct number, depending on how far down the page the user is when the window is resized.

Make a variable called pagePosition and assign it the scrollTop() value + 210 pixels.

Set the counter to zero then loop through the postTops array and increment the counter for each element in the array, if the number in pagePosition is larger than the number in each pageTops value.

In the end, counter will be 1, 2, 3, 4, or 5 depending on how far down the page the user is. You want to subtract 1 from that number.

```
$(window).on("resize", function(){

    clearTimeout(doneResizing);
    doneResizing = setTimeout( function(){
        postTops = [];
        posts.each( function(){
            postTops.push( Math.floor($(this).offset().top ));
        } );

        var pagePosition = $(window).scrollTop()+210;
        counter = 0;

        for( var i=0; i<postTops.length; i++){
            if( pagePosition > postTops[i] ){ counter++; }
        }
        counter--;
    }, 500);
});
```

Set the Link Class

Finally, at the end of this script, set the class for the correct link.

Now, it turns out, that if you run this code when the page loads, you will also solve the problem of what happens if someone refreshes the page, while they are scrolled down the page.

Rather than have this code in your script twice, turn it into a function and just call it twice.

```
$(window).on("resize", function(){

    clearTimeout(doneResizing);
    doneResizing = setTimeout( function(){
        postTops = [];
        posts.each( function(){
            postTops.push( Math.floor($(this).offset().top ));
        } );

        var pagePosition = $(window).scrollTop()+210;
        counter = 0;

        for( var i=0; i<postTops.length; i++){
            if( pagePosition > postTops[i] ){ counter++; }
        }
        counter--;

        $(allLinks).removeAttr('class');
        $("nav ul li a").eq(counter).addClass('selected');

    }, 500);
});
```

The resetPageLocation Function

Here is the resetPageLocation function, which is defined inside the window load function.

Next remove the code from inside the setTimeout function and just call this function instead.

```
$(window).on("resize", function(){

    clearTimeout(doneResizing);
    doneResizing = setTimeout( function(){

        resetPagePosition();

    }, 500);
});
```

```
    function resetPagePosition(){
        postTops = [];
        posts.each( function(){
            postTops.push( Math.floor($(this).offset().top ));
        } );

        var pagePosition = $(window).scrollTop()+210;
        counter = 0;

        for( var i=0; i<postTops.length; i++){
            if( pagePosition > postTops[i] ){ counter++; }
        }
        counter--;

        $(allLinks).removeAttr('class');
        $("nav ul li a").eq(counter).addClass('selected');
    }

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

Calling at the Top of the Page

At the top of the page, remove the code that populates the postTops array and just call this function instead.

The very last thing to do is in the index.html file.

Remove the selected class from the first link, as it will now get added programmatically, when the page loads.

```
$(window).on( 'load', function(){
    var allLinks = $('nav ul li a');
    var posts = $('section');
    var pageTop;
    var counter = 0;
    var prevCounter = 0;
    var doneResizing;

    var postTops = [];

    resetPagePosition();

    $(window).scroll( function(){
```

Review Part 1

This is an extensive script. Let's quickly review the parts.

First you created a window load function that runs when the window has fully loaded all it's images and such.

And then you define a number of variables needed by the script, and run a function (to be defined later) that sets the initial position of the page.

```
$(window).on( 'load', function(){
    var allLinks = $('nav ul li a');
    var posts = $('section');
    var pageTop;
    var counter = 0;
    var prevCounter = 0;
    var doneResizing;

    var postTops = [];

    resetPagePosition();
```

Review Part 2

Then you add a function that watches the scrolling of the page and handles what happens if the user is scrolling down, or scrolling up.

It changes the link classes appropriately as you enter and leave sections of the page.

```
$(window).scroll( function(){

    pagetop = $(window).scrollTop() + 210;

    if(pagetop > postTops[counter+1]){
        counter++;
        console.log(`scrolling down ${counter}`);
    }
    else if (counter > 0 && pagetop < postTops[counter]) {
        counter--;
        console.log(`scrolling up ${counter}`);
    }
    if( counter != prevCounter ){
        $(allLinks).removeAttr('class');
        $("nav ul li a").eq(counter).addClass('selected');
        prevCounter = counter;
    }
} );// end window scroll function
```

Review Part 3

After the window scroll function, there is one that watches for resizing of the window. It runs a function when the window is done resizing that recalculates the position of everything on the page and how far down the page the user is.

```
$(window).on("resize", function(){

    clearTimeout(doneResizing);
    doneResizing = setTimeout( function(){

        resetPagePosition();

    }, 500);
});
```

Review Part 4

Finally, at the end of the script is the helper function that is called at the top of the page and inside the window resize function.

This script is very efficient at keeping track of where the user is on the page and could be used as a foundation to do all kinds of animations and fun stuff that would get triggered as the user scrolls down the page.

```
    function resetPagePosition(){
        postTops = [];
        posts.each( function(){
            postTops.push( Math.floor($(this).offset().top ));
        } );

        var pagePosition = $(window).scrollTop()+210;
        counter = 0;

        for( var i=0; i<postTops.length; i++){
            if( pagePosition > postTops[i] ){ counter++; }
        }
        counter--;

        $(allLinks).removeAttr('class');
        $("nav ul li a").eq(counter).addClass('selected');
    }

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