Pixel Art Editor

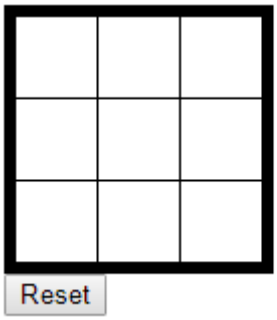
Extra Challenges

1. Adding a Reset button

Add a reset button to your HTML, below the #art div.

```
<div id="art">
        Pixels go here
</div>
<button>Reset</button>
```

The result should look something like this:



(try adding using css to add some space between the bottom of the art div and the reset button, this can be done using Margins)

Add a Javascript function to reset all of the pixels to have a white background. In your script.js file add the following:

```
function clearPixels(){
        var pixels = document.getElementsByClassName('pixel');
        for(var i = 0; i < pixels.length; i++){
                pixels[i].style.backgroundColor = 'white';
        }
}
```

Finally we need to trigger this function from the reset button. So we add the onclick attribute to our HTML like so:

```
<button onclick=”clearPixels()”>Reset</button>
```

We can now test this by adding some colour to the grid and then clicking the reset button.

Explaining the Code

Let's take a look at what this code is actually doing. First of all we have this line:

```
var pixels = document.getElementsByClassName('pixel');
```

"document" refers to the current html page.

The "getElementsByClassName" function looks through the document for any elements which have a specified class.

In this case we want to get all of the pixels on our page, these all contain the "pixel" css class, so the document.getElementsByClassName('pixel') will return an **Array** containing all of the pixel elements.

We are then storing that Array into a new variable which we have decided to call "pixels". This is so we can reuse it in future without having to type "document.getElementsByClassName" every time.

Now the next step is to do something with our list of pixels. In order to change each pixel we need to **Loop** through each item in the list and change each pixel in it individually. In this case we will be using a **For Loop** to do this.

```
for(var i = 0; i < pixels.length; i++){
        Code goes inside the loop
}
```

Loops can initially look a bit complicated, so let's see what's actually going on here.

To begin with we create a variable called "i" (this could be called anything, but i is a common convention used when writing a loop, it stands for index). In order to loop i requires a **starting point**, so we set it to be equal to 0.

The next step is then to set an **end point** for the loop, when it will finish. In this case we want to go through the entire "pixels" array. Using "pixels.length" will tell us how many items are in the array. So we can say "continue to loop while i is less than the number of pixels".

Finally we can change the value of i at the end of each loop. We only want i to increase by 1 at the end of each loop so that the loop moves on to the next item. Writing i++ is just means "add 1 to i".

So essentially i will start at 0, then increase to 1 and then 2 etc etc until the loop finishes.

Finally, we update the backgroundColor style on each pixel to set it back to white.

```
pixels[i].style.backgroundColor = white;
```

This selects an item from the pixels array which has an index equal to i. So when i = 0 it will select the first pixel in the array. On the next loop i will equal 1, so it will select the next item in the array. Again the next loop i will equal 2 and so the third pixel will be selected, and so on until all the pixels in the array have had their backgroundColor reset.

Extra info

For extra reading on how Arrays and Loops work see the following pages:

Arrays - https://www.w3schools.com/js/js_arrays.asp

Loops - https://www.w3schools.com/js/js_loop_for.asp

Populating the grid with loops

As you may have noticed typing out the HTML for each row and pixel in the grid is slow and boring, and you can end up with accidental mistakes in the code. For these reasons **Code Reuse** is a huge part of programming, being able to write a small amount of code which can be used to do common tasks over and over without you as the coder having to change anything.

In this example we will be looking at how we can reuse the same code to make the pixel art grid any size we want, while changing as little as possible.

In order to do this we will be using more loops, just like we did in the previous challenge.

Adding the variables

First off, we want to tell our program how many rows we want, and how many pixels we want in each row. To do this we just add two variables in to our script.js file.

```
var numRows = 5;
var numPixels = 5;
```

These can contain any number that you want, and they don't have to be the same! But in order to keep things simple to begin with we'll start with a 5x5 grid.

Adding Rows

Now we need a way to build up a list of all the rows we want. Ideally good code should be split up in to blocks, with each block trying to achieve one goal. In Javascript we call these blocks **Functions**.

Functions are used to encourage Code Reuse, rather than typing out the same code again and again we can just **call** a Function and it will run all of the code that it contains.

In this case we can create a Function which has the goal of adding all of the rows we need to the grid.

```
function addRows(){
        var rows = "";
        for(var i = 0; i < numRows; i++){
                rows += "<div class='row'></div>";
        }
        return rows;
}
```

In this function we have three main things going on.

- Firstly, we create a "rows" variable to store the HTML we're generating.
- Secondly, we add a loop which will add some HTML to the "rows" variable until it reaches the end point (the end point being when i equals numRows, this means that we can change how many times the loop will run by changing the value of numRows).
- Finally, we **return** "rows". Returning a value from a function means that we can use that value in other places outside of that function (hopefully this will become clearer in the next section).

Adding the grid to the art div

In order to test the code that we have so far we need to add the rows to our "art" div in the HTML code. We can do this by adding another short function.

```
function addGrid(){
        var art = document.getElementById('art');
        art.innerHTML = addRows();
}
```

We create a variable called art, we use getElementById to find the "art" div in our HTML and store it in the art variable.

We can then use art.innerHTML to set what HTML goes inside the art div.

```
<div id="art">
        Inner HTML will go here
<div>
```

In this case we are **calling** the addRows() function we created earlier, running any code inside that function and then setting the innerHTML to equal the **returned** value from that function.

Finally all we need to do is call the addGrid function from inside our code. That will cause the grid to be created when our page starts up. We can do this by just adding addGrid(); at the bottom of our code.

So far the full code should look something like this:

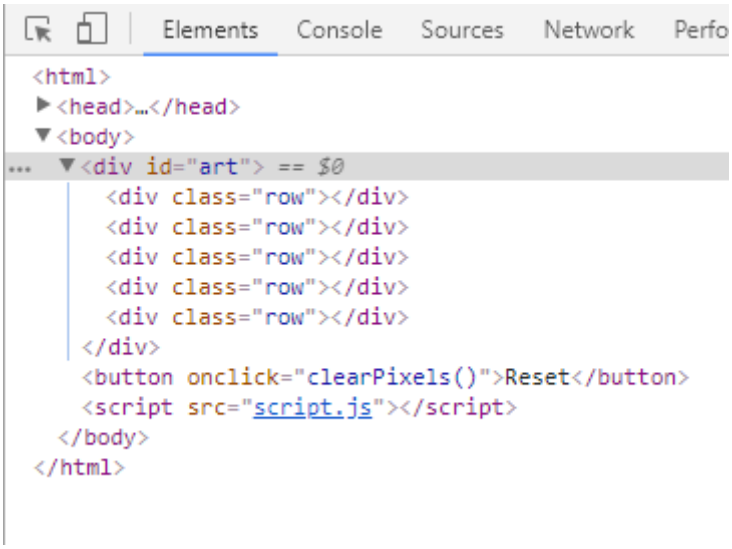
```
var numRows = 5;
var numPixels = 5;

function addRows(){
        var rows = "";
        for(var i =0; i < numRows; i++){
                rows += "<div class='row'></div>";
        }
        return rows;
}

function addGrid(){
        var art = document.getElementById('art');
        art.innerHTML = addRows();
}

addGrid();
```

As you will see when you run this, the grid will currently be empty as we haven't added any pixels yet. However, using the developer tools built in to most browsers (right click -> inspect in Chrome) we can find the art div and see that it does actually contain the rows that we've added.



Adding Pixels

In order to add Pixels to the grid we can essentially copy the addRows function from earlier, replacing the pixel html rather than the row. And making sure to use numPixels for the loop end point rather than numRows.

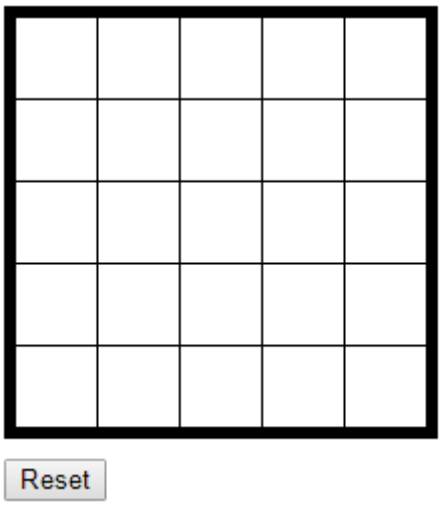
```
function addPixels(){
        var pixels = "";
        for(var i =0; i < numPixels; i++){
                rows += "<div class='pixel' onclick='setPixelColour(this)'></div>";
        }
        return pixels;
}
```

The important part about this function is how we call it. We need to update the addRows function to call addPixels inside the row HTML.

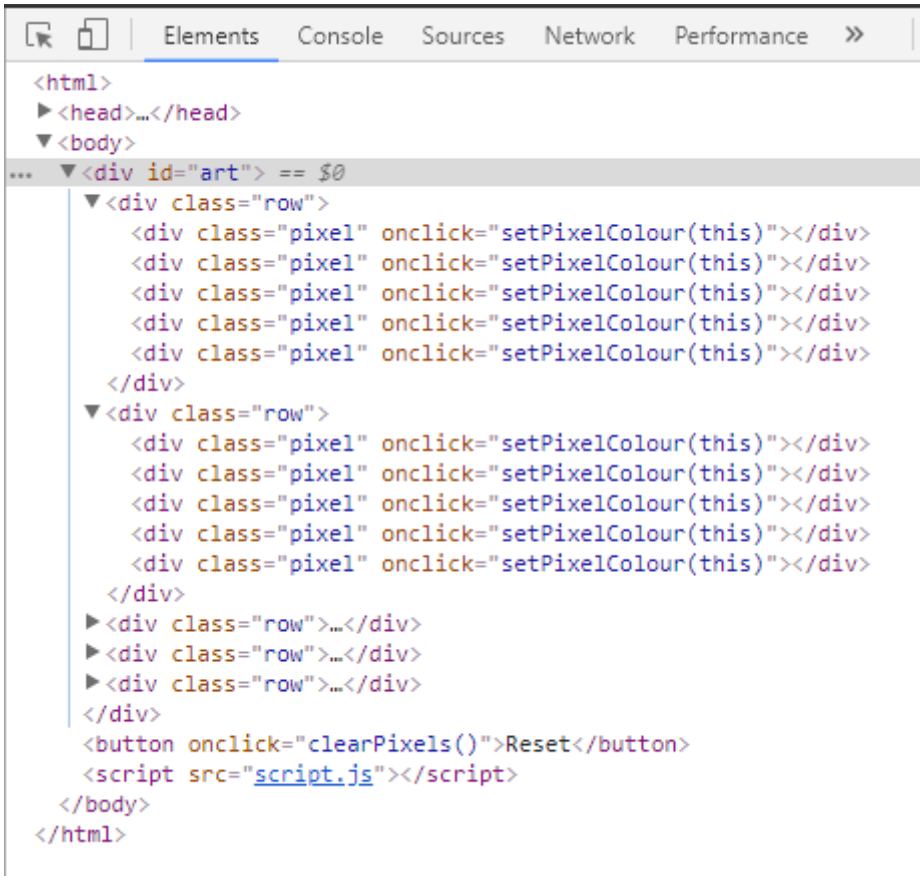
```
function addRows(){
        var rows = "";
        for(var i = 0; i < numRows; i++){
                rows += "<div class='row'>" +  addPixels() + "</div>";
        }
        return rows;
}
```

Notice how we've added a call to the addPixels function inside the row HTML. This means that every time a row is added addPixels will be called which will add 5 pixels inside the row. Essentially this is using a loop inside another loop.

Refresh the page once these changes have been added and you should be able to see the full grid has now been created.



Now if we inspect the code in the browser we can see that each row now contains 5 pixels.



Try adjusting the values of numRows and numPixels and seeing what effect this has on the grid, why does this change happen?

Updating the grid from the Web Page

At the moment we still need to change the values of numRows and numPixels in our code in order to change the size of the grid, but ideally we would like to be able to change this from our html page, without needing to touch the code. This is one of the cool things about Javascript, it allows you to change elements on the page and instantly see those changes on your site.

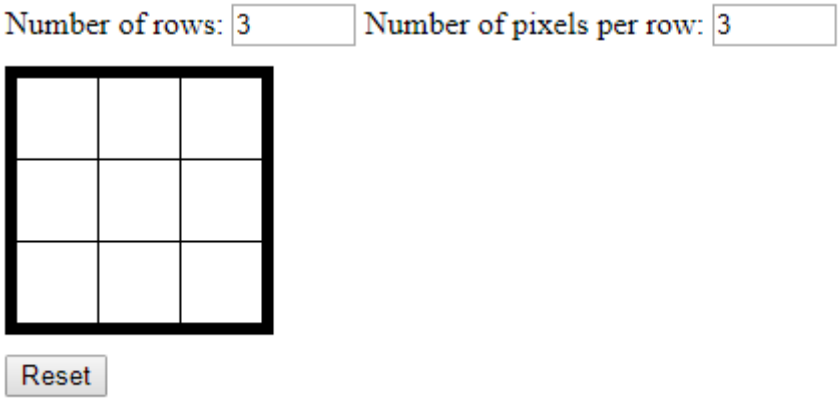
Adding some number inputs

The first step will be to add some number inputs to our html so that we can change the number of rows and pixels in each row. Try adding the following HTML code above the "art" div.

```
Number of rows: <input id="numRows" type="number" value="3">

Number of pixels per row: <input id="numRows" type="number" value="3">
```

These won't do anything yet but if you visit your page you should see something like this:



Remember that you can add your own styling via css if you want to change how the page looks.

Updating the Javascript

Rather than "hard coding" the values of numRows and numPixels in our code we can now set them to use the values from the number inputs we just added. We want this to happen any time the grid is changed so we can add this code to the addGrid function, and remove the existing numRows and numPixels variables.

```
function addGrid(){
        var numRows = document.getElementById('numRows').value;
        var numPixels = document.getElementsById('numPixels').value;

        var art = document.getElementById('art');
        art.innerHTML = addRows();
}
```

Now that we have these values stored we need to actually use them in our addRows and addPixels functions. Unfortunately this leads us to a problem**, variables that are created inside a function can not be used in other functions**. This is called the **scope** of the variable. So our addRows function has no idea that the numRows variable exists, and can't use the value from it at the moment.

In order to get around this we can use something called a **parameter**, which lets us pass an existing variable from one function to another.

Adding parameters to the addRows function

You've probably noticed by now that all of our functions have a pair of brackets after the name, which don't seem to do anything. This is where we tell the function what parameters it can expect. Having a pair of empty brackets () after a function means that it doesn't need any paramters.

So we need to change our addRows function to take two parameters, numRows and numPixels, so that we can use those values within that function. You can add these parameters inside the brackets, like this:

```
function addRows(numRows, numPixels){
        var rows = "";
        for(var i = 0; i < numRows; i++){
                rows += "<div class='row'>" + addPixels() + "</div>";
        }
        return rows;
}
```

The numRows variable in the loop now uses whatever value is passed in to the function in the numRows parameter.

However this means that addRows **must be given two parameters any time it is called** so we need to update the addRows() call in the addGrid function.

```
function addGrid(){
        var numRows = document.getElementById('numRows').value;
        var numPixels = document.getElementsById('numPixels').value;

        var art = document.getElementById('art');
        art.innerHTML = addRows(numRows, numPixels);
}
```

Adding a parameter to the addPixels function

Notice how despite giving numPixels to the addRows function we aren't actually using it anywhere, so we also need to add that to the addPixels function, like so:

```
function addPixels(numPixels){
        var pixels = "";
        for(var i =0; i < numPixels; i++){
                rows += "<div class='pixel' onclick='setPixelColour(this)'></div>";
        }
        return pixels;
}
```

And we also need to update the addPixels call in the addRows function:

```
function addRows(numRows, numPixels){
        var rows = "";
        for(var i = 0; i < numRows; i++){
                rows += "<div class='row'>" + addPixels(numPixels) + "</div>";
        }
        return rows;
}
```

So now the numPixels parameter is being passed from addRows to the addPixels function.

Getting the grid to change

The last thing we need to do is call the addGrid function from our html, to make the grid change when the numbers are changed. Earlier we were using the "onclick" event to call our Javacript, but this time we want to use the "onchange" event so that any time the numbers are changed the grid will update.

Add the following to your html:

```
Number of rows: <input id="numRows" type="number" value="3" onchange="addGrid()">
Number of pixels per row: <input id="numRows" type="number" value="3" onchange="addGrid()">
```

If you now open up your html file you should find that adjusting the numbers will change the size of the grid.