Using Masonry.js

- Head on over to https://masonry.desandro.com/
- You could either
    - 1. download the library file and keep in a local directory (like we do with P5).
    - 2. Use a link to the CDN (Content Delivery Network).
- Either way, you must link to one of these in a script tag in your html.

For example, If I download the file I would add this to my head in the html:

```
<head>
```

```
<script src="/path/to/masonry.pkgd.min.js"></script>
```

```
</head>
```

If I used the CDN, I would add:

```
<head>
```

```
<script src="https://unpkg.com/masonry-layout@4/dist/masonry.pkgd.min.js"></script>
```

```
</head>
```

Great, now we linked to the masonry.js library.

Now we need to create a .js file that we will write our javaSctipt in.

Create that file,, call it something like `myMasonryScript.js`. Perhaps you want to stick it in a folder called `scripts` in your main project folder.

Now we need to link THAT script to out html.
Add this to the end of your `<body>` :

```
<script src="scripts/myMasonryScript.js"></script>
```

Great.

Lets talk about how Masonry.js works:

- Masonry.js works on a container grid element with a group of child items (the grid items):

```
<div class="gridContainer">
    <div class="gridItem">...</div>
    <div class="gridItem gridItem--width2">...</div>
    <div class="gridItem">...</div>
    ...
</div>```

- The sizing of the grid items is handled by the CSS
```

css
.gridItem {
width: 200px;
}
.gridItem--width2 {
width: 400px;
}```

- We initalize our masonry grid by creating a new Masonry object, using the `Masonry()` constructor.

- First, select out grid container element, in our myMasonryScript.js:

```
var theGrid = document.querySelector('.gridContainer');
```

- Then create the Masonry object form the grid element:

```
var msnry = new Masonry( elem, {
// options
itemSelector: '.gridItem',
columnWidth: 200
});
```

The Masonry() constructor takes two arguments:

- 1 The container element,
- 2.an options argument that uses object notation to set which selector is our grid items, and the column width, and if we want it to be set by %.

```
itemSelector: '.gridItem',
columnWidth: '50px',
percentPosition: false
```

It is recommended that you always set the columnWidth and the itemSelector.
Look at https://masonry.desandro.com/options.html for a list of options.
Sweet. Now lets use it.

The most important method is the layout() method. When it is called, it lays out your grid.

```
msnry.layout();
```

Its useful whenever items have changed and you want to re-layout your grid.

Head over to https://masonry.desandro.com/methods.html to see list of the available methods and the functionality of Masonry.js

Animate the boxes:
Apparently, you cannot animate the (re)sizing of an item AND properly lay it out.
To get around this, you can animate the child element of a grid item.

```
<div class="gridContainer">
  <!-- items have grid-item-content child elements -->
  <div class="gridItem">
    <div class="grid-item-content"></div>
     <!--such as an <img> -->
  </div>
```

In the CSS we will add a transition: property to the grid-item-content:

```
.grid-item-content {
    transition: width 1s, height 1s;
}
```

Then we will write a another selector that will take the place of the .grid-item-content whenever we want to animate the size of the content and cll it on both the grid item and the content:

```
.gridItem.is-expanded, .gridItem.is-expanded .grid-item-content   {
  width: 400px;
  height: 400px;
}
```

In the JS, we create a variable to target the content.
In this way, whenever we want to resize an item, we tack on the .is-expanded class which will trigger the animation

```
var itemElem = event.target.parentNode;
itemElem.classList.toggle('is-expanded');
```

both the grid Item and the grid Item content change size, but only the gridItem content is animated.

and don't forget to call `msnry.layout()` after.

For a responsive sizing and layout set the Masonry constructor, add `percentPosition : true`. Then, set the `columnWidth` as a child element in the grid container that is set to the same size as the grid items. Then use `%` sizing in the css instead of `px`.
For example in your html, add a div class='grid-sizer',

```
<div class="grid">
  <!-- width of .grid-sizer used for columnWidth -->
  <div class="grid-sizer"></div>
  <div class="grid-item"></div>
  <div class="grid-item grid-item--width2"></div>
  ...
</div>
```

in your CSS, size grid-sizer with your grid-item:

```
.grid-sizer,
.grid-item {
    width: 20%;
}
```

Then in your js, give the masonry constructor this:

```
var msnry = new Masonry( elem, {
// options
itemSelector: '.grid-item',
// use element for option
columnWidth: '.grid-sizer',
```

```
  percentPosition: true
  });
```