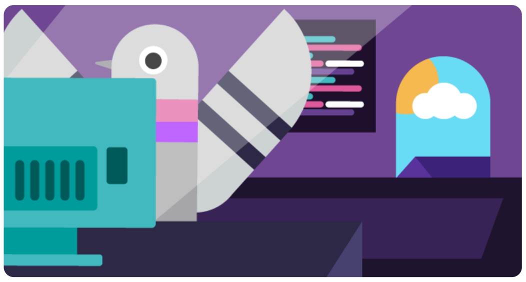
Projects

Bird watch website 3.0

Add nice layouts and cool effects to your website!



Step 1 Introduction

These Sushi Cards will show you how to make a website look polished and professional using cool effects and layouts.

What you will make

Move your mouse cursor around in the window below to try out some of the effects you'll learn how to use on your website:

What you will learn

In these Sushi Cards, you will find out how to:

- Create any website layout you want
- Make your layout adjust itself automatically to different screen sizes
- Make a collage of overlapping pictures and text
- Apply some neat hover and click effects that you might have seen on other websites

What you will need

Hardware

- A computer capable of accessing **trinket.io** (https://trinket.io)

Software

This project can be completed in a web browser using **trinket.io** (https://trinket.io) or offline using a plain text editor.

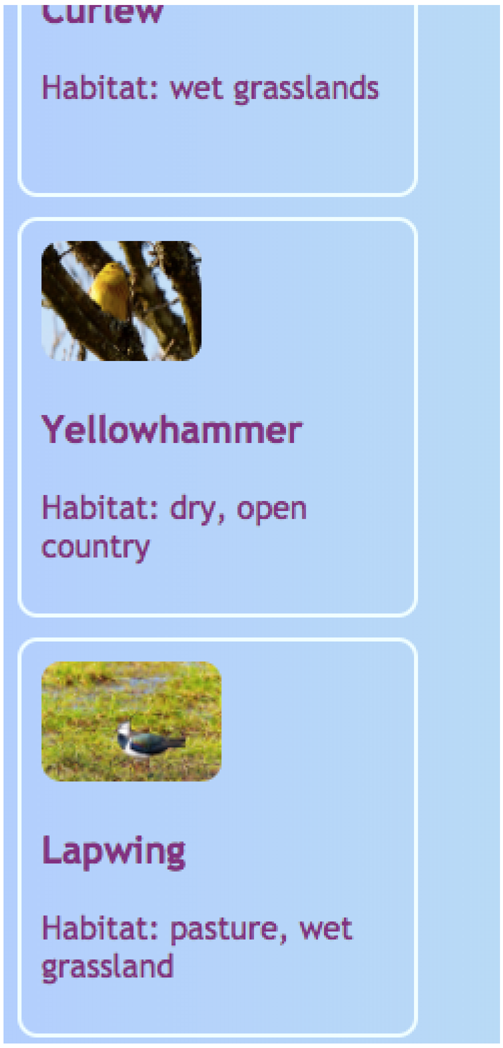
Step 2 Getting set up

To work through these Sushi Cards, you'll need to start with a website that you already made. For example, you can use the one that you made in the **HTML/CSS Intermediate Sushi Card series** (https://projects.raspberrypi.org/en/projects/cd-intermediate-html-css-sushi). If you prefer, you can work with **this trinket** (http://dojo.soy/se-html3-start), which is ready to use.

As you'll be working with lots of code, you may find it easier to use an offline text editor on your computer instead of Trinket. You can download code from Trinket by selecting **Download** from the **Share** menu.

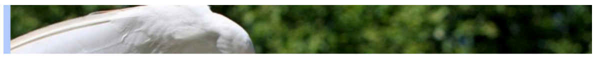
Step 3 Clickable cards

Here’s a technique you could use to make a photo gallery, or a portfolio page showing off your projects: little **preview cards**.



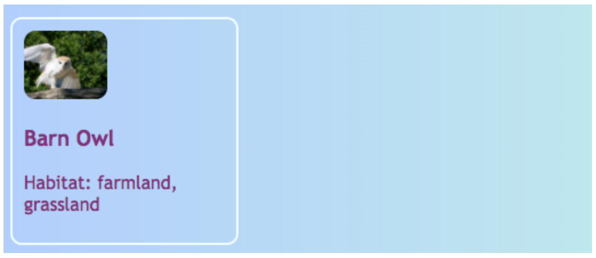
- Add the following HTML code to your website, anywhere you like. I’m doing mine on `index.html`. You can change the picture and text to suit your own preview cards. I’m going to do a bunch of highlights of the tourist attractions in Ireland.

```html
<article class="card">
    <img src="barn-owl-landing.jpg" class="tinyPicture">
    <h3>Barn Owl</h3>
    <p>Habitat: farmland, grassland</p>
</article>
```



- Add the following CSS code to create the classes `card` and `tinyPicture`:

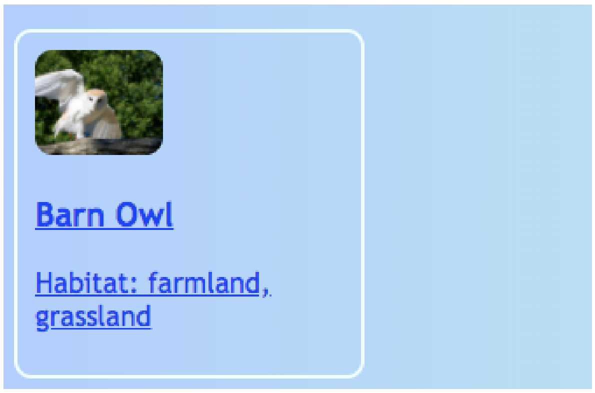
```
.tinyPicture {
    height: 60px;
    border-radius: 10px;
}
.card {
    width: 200px;
    height: 200px;
    border: 2px solid #F0FFFF;
    border-radius: 10px;
    box-sizing: border-box;
    padding: 10px;
    margin-top: 10px;
    font-family: "Trebuchet MS", sans-serif;
}
.card:hover {
    border-color: #1E90FF;
}
```



Let’s turn the whole preview card into a link so people can click to see more information.

- Place the whole `article` element inside a link element. Make sure the closing `</a>` tag is after the closing `</article>` tag! Feel free to change the link **URL** to whatever you want to link to. That could be another page on your website, or it could be another website entirely.

```
<a href="birds.html#scBarnowl">
    <article class="card ">
        <img src="barn-owl-landing.jpg" class="tinyPicture">
        <h3>Barn Owl</h3>
        <p>Habitat: farmland, grassland</p>
    </article>
</a>
```


Barn Owl

Habitat: farmland, grassland

Linking to a specific part of a page

Notice how the value of `href` in my link ends in `#scBarnowl`? This is a neat trick you can use to jump to a particular part of a page.

- First, type the URL of the page to link to, followed by `#`.
- In the code file for the page you are linking to, find the part you want to jump to and give that element an `id`, for example, `<section id="scBarnowl">`. The value of the `id` is what you type after the `#` in your link.

Resetting styles

Now that the whole preview card is a link, the text font may have changed.

- If so, you can fix it by adding a **CSS class** to the link: `class="cardLink"`. Here's the CSS code to put in your style sheet:

```css
.cardLink {
    color: inherit;
    text-decoration: none;
}
```

Setting the value of any property to `inherit` makes it use the value that the **parent** element has. So in this case, the text colour will match the rest of the text on the homepage.

- Make at least four or five of these cards. If you are working from my example website, you could do one for each of the sections on the Protected Birds page. On the next Sushi Card, you'll learn how to arrange the cards with a cool trick!

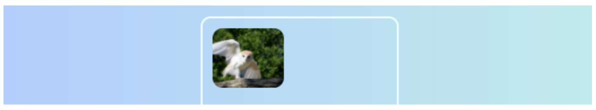
Step 4 All in a row

On this card you will learn some tricks for arranging things **horizontally** on a page. First, you'll see how to get stuff centered. Then you'll arrange elements side by side in a row.

- Add the following CSS properties to the `.card` class:

```
margin-left: auto;
margin-right: auto;
```

You should see the cards move to the center of the page. By setting the left and right margins to `auto`, you can make any element be in the middle instead of over to the left.



- Drag the edge of the browser window to make the page narrower and wider — notice that the cards stay centered.
- Put all of the card links you just made into a new container element. It's not going to be an `article` or a `section`, but one called `div`. This is a general-purpose container you can use for grouping things and making nice layouts.

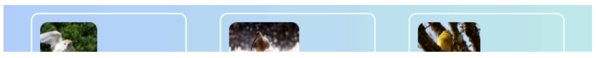
```
<div class="cardContainer">
```

- Add the following CSS code in your style sheet:

```
.cardContainer {
    display: flex;
    flex-wrap: wrap;
    justify-content: space-around;
    padding: 10px;
}
```

Voilà! Thanks to **Flex**, your cards are now displayed side by side!

- Drag the edge of your window to make the website wider and narrower, and watch how the cards move around to fit the window size, sometimes wrapping to the next line.



- Try deleting the `width` and `height` properties from the `.card` class and see what happens: `flex` cleverly fits the cards together like a jigsaw puzzle, keeping an even height across everything that's in the same row.

If you have a navigation menu at the top of your page, that's another place you can use this trick. Your menu needs to be composed of list elements((`li`) for this next bit. If you prefer, you can try it out with my website.

- Find the CSS rules for the menu. In my website, that's the blocks `nav ul`, `nav ul li`, and `nav ul li a`.
- Delete the property `display: inline;` from the list items. Then, in the list `nav ul`, add in:

```
display: flex;
justify-content: flex-start;
```

You end up with pretty much the same menu, right? The cool thing about `flex` is you can control the layout with the property `justify-content`.

- Change the value of `justify-content` to `flex-end` and see what happens. Or change it to `space-around` to make the menu items evenly spaced, just like you did for the cards.

`flex` is a pretty powerful layout tool that could fill a whole Sushi Card series of its own — you can learn more about it at **dojo.soy/se-flex** (http://dojo.soy/se-flex).

Step 5 Make your menu responsive

A **responsive** website is one that adjusts itself to the screen size so it always looks great, whether you're looking at it on a computer, mobile phone, or tablet. Let's make your menu responsive!

You'll start with the regular styles: this will be your **default** behaviour.

What does 'default' mean?

The default styles are your normal set of style rules. They are applied no matter what, before checking any special conditions.

You can add code that then checks the size of the screen and makes some adjustments if necessary.

- Add the following CSS rules to your menu. You probably have colours and borders defined as well; I've left them out to save space here! If you already have CSS rules defined for your menu, just add in or change the properties and values below that you are missing.

```
nav ul {
    padding: 0.5em;
    display: flex;
    flex-direction: column;
}
nav ul li {
    text-align: center;
    list-style-type: none;
    margin-right: 0.5em;
    margin-left: 0.5em;
}
```

With the CSS code above, your menu will be best suited to small screens. This is called **mobile-first** development.

What does 'mobile-first' mean?

Quite often when coding a website, you will be using a computer screen, and you'll probably define your styles based on how it looks on that screen.

When you code for mobile first, you instead choose default styles that are suitable for small screens such as smartphones. You then add extra code to make adjustments for bigger screens.

Since more and more people browse the internet on their smartphones or tablets rather than on a computer, it's good practise to develop your website with this in mind.

- Now add the following code to your style sheet:

```
@media all and (min-width: 1000px) {
    nav ul {
        flex-direction: row;
        justify-content: space-around;
    }
}
```

The first line of code above checks what size the browser window is. If the window is **1000 pixels** wide or more, it will apply all the style rules inside the block.

How does it work?

The block contains new values for only some properties of the `nav ul` menu.

Whenever the window is wider than 1000 pixels, these new values will be applied instead of the ones you already defined for `nav ul`.

The rest of the properties you defined previously for `nav ul` will stay the same.

- If you are using Trinket to write code, it might be helpful to download the project so you can test it out on a full-size screen.

Challenge: make your menu adjust itself for big screens

- Can you add another block for screens bigger than **1600 pixels**, with `flex-end` instead of `space-around`?

I need a hint

The following code defines flex properties for menu items when the screen is bigger than 1600 pixels:

```
@media all and (min-width: 1600px) {
    nav ul {
        flex-direction: row;
        justify-content: flex-end;
    }
}
```

You can put any CSS rules you like into blocks like these to define different styles for different screen sizes. It'll be especially useful when you do CSS grid layouts later!

Step 6 Captions and side notes

On this card you'll learn about two more types of **container** element: one that you can use to add a caption (some text like a title or short description) to a picture, and another for when you have extra stuff that doesn't really belong with the main information on a page.

Pictures with captions

- Find an `img` element where you have text above or below that goes with the picture. I'm working with the Tito picture on `index.html`, but you can go with whatever is on your website.

```
<img id="owly" src="barn-owl.jpg" class="topDivider someSpacing mediumPictures" alt="A barn owl" />the dog" />
<p>
  Owly the barn owl
</p>
```

- On the line above the code, add the opening tag `<figure>`. On a new line below the code, place the closing tag `</figure>`.

- Next, remove the `p` tags, or whatever tags you have around the text (maybe it's a heading, like `h2`?), and put the text in between `<figcaption> </figcaption>` tags instead. The whole thing should look something like this:

```
<figure>
    <img id="owly" src="barn-owl.jpg" class="topDivider someSpacing mediumPictures" alt="A barn owl" />the dog" />
    <figcaption>
    Owly the barn owl
    </figcaption>
</figure>
```

The `figcaption` element is your **caption**. It can go either above the `img` element or below it.

Why is this useful?

The `figure` element acts as a sort of **container** for your picture and its caption. This allows you to treat them as one unit when defining styles.

Grouping them together logically also helps to maintain good structure in your website code.

You can use CSS code to style `figure` and `figcaption` as you would any other element using classes, IDs, or element selectors. I'm adding the following rules to remove the extra spacing that was added by the new container:

```
figure {
    margin-top: 0px;
    margin-bottom: 0px;
    margin-left: 0px;
    margin-right: 0px;
}
```

Side notes

The Protected Birds page on my website is a list of bird species that are in decline in Ireland. I want to add some notes about common reasons for declining bird numbers, as well as some useful links. That information doesn't really belong in the `article` element with all the birds. This is an example of when you might use the `aside` element.

- Go to a page of your website that has an `article` element on it — I'm using `birds.html`.
- **Outside** of the `article` element, add one or more pairs of `<aside> </aside>` tags containing your extra stuff.

```
<aside class="sideNoteStyle">
    <h3>Threats to birds</h3>
    <p>
      Some of the main reasons for declining numbers of birds are:
    </p>
    <ol>
      <li>Habitat destruction</li>
      <li>Pollution</li>
      <li>Climate change</li>
    </ol>
  </aside>
  <aside class="sideNoteStyle">
    <h3>Useful links</h3>
    <p>See the complete published <span class="warnOrange">amber</span> and <span class="warnRed">red</span> lists
    <a href="https://www.birdwatchireland.ie/LinkClick.aspx?fileticket=VcYOTGOjNbA%3d&tabid=178">here</a>.</p>

    <p>Check out the Wikipedia <a href="https://en.wikipedia.org/wiki/Bird_conservation">article</a>.</p>
  </aside>
```

Why is this useful?

The `aside`, `article`, and other containers are all similar. The only real difference is in the **meaning**, that is, what you use them for.

It's important to use meaningful HTML elements whenever you can. It gives your website better structure and is especially helpful for people using **screen readers**.

Did you spot the other element in there, `span`? This is a special tag you can use just for adding extra CSS code! You can put anything in between a pair of `span` tags. It's useful for things like styling a **part** of the text in a paragraph.

- Add the following CSS code to your style sheet to complete the styling for the HTML code above.

```
.sideNoteStyle {
  border: dotted 1px purple;
  background-color: #cddffe;
  padding: 0.5em;
  margin: 0.5em;
}
.warnOrange {
  background-color: #ffa500;
}
.warnRed {
  color: #FF4500;
  font-size: larger;
}
```

On the next card, you're going to learn how to make your website's layout more interesting!

- To get ready, make a page that has one `article` and two `aside` elements inside the `<main> ↙main>` tags. Or if you prefer, you can work with the Protected Birds page on my website.

Step 7 Design cool page layouts

- For this card you should work with a page that contains a `main` element with three elements inside: one `article` and two `aside`s. Go ahead and create these first if you need to. If you want to work with my website, add the `aside` code from the previous Sushi Card to the Protected Birds page.

Here are three different page layouts you'll be applying:

- Add new CSS classes to `main` and each of three elements inside it.

```
<main class="myPageLayoutGrid">
    <article class="myGridArticle">
        <!--other stuff here-->
    </article>
    <aside class="myGridAside1">
        <!--other stuff here-->
    </aside>
    <aside class="myGridAside2">
        <!--other stuff here-->
    </aside>
</main>
```

The container you'll change the layout of is `main`, but you could do this with any kind of container, like a `div` or `article`, or even the whole page `body`. The technique you're going to use is called **CSS grid**.

In this example, the `header` and `footer` will be left out of the design, but it's quite common to include them in the grid too.

- Set the `display` property to `grid` on the overall container:

```
.myPageLayoutGrid {
    display: grid;
    grid-column-gap: 0.5em;
    grid-row-gap: 1em;
}
```

What do you think the `grid-column-gap` and `grid-row-gap` properties do?

- Next, you name a `grid-area` for each element:

```
.myGridArticle {
    grid-area: egArticle;
}
.myGridAside1 {
    grid-area: egAside1;
}
.myGridAside2 {
    grid-area: egAside2;
}
```

Then you design your layout! Let's put the two `aside` elements side by side at the bottom of the page. For this you need two **columns** of equal width. You can keep the **row** height automatic.

- Put the following code inside the `.myPageLayoutGrid` CSS rules:

```
grid-template-rows: auto;
grid-template-columns: 1fr 1fr;
grid-template-areas:
    "egArticle egArticle"
    "egAside1 egAside2";
```

`fr` stands for **fraction**. Notice how you make the `article` take up all the space over the two columns.

Help! I got errors and warnings!

If you are using Trinket, you may notice some errors and warnings appear, even if you typed the code exactly as above. This is because Trinket does not yet recognise the CSS grid properties. However, the code will still work.

If the CSS grid code gives you 'unknown property' warnings or an error like 'unexpected token 1fr', you can simply ignore these.

Let's put the `aside` elements over on the right and make them half the width of the `article`.

- Change the values of `grid-template-columns` and `grid-template-areas` to:

```
grid-template-columns: 2fr 1fr;
grid-template-areas:
    "egArticle egAside1"
    "egArticle egAside2";
```

- If you don't want the `aside` elements to stretch all the way to the bottom, you can add a blank space using a dot:

```
grid-template-areas:
    "egArticle egAside1"
    "egArticle egAside2"
    "egArticle . ";
```

Challenge: make different layouts for different screen sizes

- Can you use the screen size checks you added earlier to make the layout change depending on how wide the screen is? Note: if you already created CSS blocks for each screen size, you can add the new CSS code to those blocks instead of creating new ones.

I need a hint

The following code defines a layout for the CSS class above when the screen is bigger than 1600 pixels:

```
@media all and (min-width: 1600px) {
    .myPageLayoutGrid {
        grid-template-columns: 1fr 1fr;
        grid-template-areas:
            "egArticle egAside1"
            "egArticle egAside2"
            "egArticle .";
    }
}
```

With **CSS grid**, you can make almost any layout you like. If you want to learn more, go to **dojo.soy/se-css-grid** (http://dojo.soy/se-css-grid)

Step 8 Photo collage

On this card you will learn to use CSS to exactly position HTML elements and make a photo collage.

- Add a `div` to your page and put as many images in it as you like. Give the `div` and the `img` elements `id` values.

```
<div id="photoBox" class="relPos">
    <img id="imgYoungKestrel" class="absPos" src="young-kestrel.jpg" alt="A young kestrel" />
    <img id="imgYoungKestrelTree" class="absPos" src="baby-kestrel.jpg" alt="A young kestrel on a branch" />
</div>
```

The photos will appear one after the other on the web page, in the order they appear in your code.

- In your CSS file, add the following CSS class for the elements inside the `div`:

```
.absPos {
    position: absolute;
}
```

- Next, you need to add the property `position: relative;` to the container itself and define a size for it. This makes it so that the positions of the other elements are defined **relative to** (that is, within) the container.

```
.relPos {
    position: relative;
}

#photoBox {
    width: 800px;
    height: 400px;
}
```

- Then create a set of style rules for each of the elements using **id selectors** to set their sizes (`width` and/or `height` properties) as well as their exact positions.

To define the position of an element, there are four properties you can use: `left`, `right`, `top`, and `bottom`. They represent how far each of the edges should be from the parent's edge. Use either `top` or `bottom` for the vertical position, and either `left` or `right` for the horizontal position.

- Choose exact positions for each of your pictures, and use any of the properties `left`, `right`, `top`, and `bottom` to define those positions in your CSS rules. For example, this code places the cat picture 100 pixels from the top and 60 pixels from the left:

```
#imgYoungKestrel {
    width: 230px;
    top: 100px;
    left: 20px;
}
```

Note: The position values can also be negative! If you use a negative value, it will push the element off outside the container, over whichever edge you've specified.

Making things overlap

You might want to have some of the pictures overlapping. But how do you choose which one goes on top?

- Choose two images and give them positions that cause them to overlap.
- Add an extra property, `z-index: 10;` to one of them, and then add `z-index: 6;` to the other.
- Take a look at the result on your webpage.

- Now swap the `z-index` values, so that the `6` and the `10` are the other way around. Do you see any difference on your web page?

How does z-index work?

The `z-index` property lets you decide how two or more elements should overlap. The value can be any whole number.

The element with the **highest** number ends up on **top** of the pile, or in other words at the very **front**. The element with the next highest number is behind that, and in front of the others, and so on, until you get to the element with the lowest number, which appears at the back behind all of the other elements.

You can position any HTML elements in this way, not just images. For example, you could use a `p` element to add some text over a photo.

Challenge: make a photo collage

- Try creating your own collage of photos like the one shown below! Use exact positioning together with different `z-index` values to get the overlap effect the way you want it.

I need a hint

Here are the CSS classes I've used:

```
.collagePhoto {
    border: 1px solid white;
}
.relPos {
    position: relative;
}
.absPos {
    position: absolute;
}
```

Step 9 Special effects

On this card you'll learn a few more nice effects that you can achieve with CSS.

Shadows and movement

Let's add a little movement when you hover your cursor over the cards you made earlier.

- Find the `.card:hover` CSS class from earlier and change it to the following:

```
.card:hover {
    box-shadow: 0px 2px 2px rgba(0,0,0,0.2);
    transform: translateY(-2px);
}
```

- Try out different values in the `translate` function!

The `transform` property

If you completed the Intermediate HTML/CSS Sushi Cards, you may remember using the `transform` property in some `@keyframes` animations. Here you see that you can also use the property on its own within a regular CSS block.

One kind of value you can set it to is `rotate`, to make an element turn. Others are `translateY`, which moves something up or down, and `translateX`, for movement from side to side.

- Play about with different pixel values in the `box-shadow` property to see what they do.

What's `rgba`?

`rgba(0,0,0,0.2)` is another way of defining a colour.

It's got the usual three numbers (from `0` up to `255`) for red, green, and blue.

The fourth number, called the **alpha** value, defines how **transparent** (or see-through) something is. It is a decimal number between `0` and `1`, with `1` being not see-through at all, and `0` being completely invisible. This means the lower the alpha value of an element, the more see-through it is.

- Finally, make the movement smooth by adding the following property to the `.card` class from earlier:

```
transition: all 0.2s ease-out;
```

A duration of `0.2s` means the `transition` lasts for 0.2 seconds.

Lightbox

Another effect you've probably seen on loads of websites is **lightbox**: you click on something and the website dims while something else, like a bigger picture or a popup box, appears in front of everything.

To get this effect you will make two links: one for the actual lightbox (the bit that pops up), and one for the thing that you click to make the lightbox appear. I'm going to do mine on the Attractions page of my website. You go with whatever page you have pictures on!

- Decide what things you want to appear when you click, and add them all to your page in between a set of `a` tags to make a link. Make sure you give the link an `id`. The code can go anywhere on the page: you will be making the elements invisible in the next step!

```
<a href="#_" class="lightbox" id="boxBarnowl">
    <h3>Hi there!</h3>
    <img src="barn-owl-landing.jpg" alt="Picture of a barn owl" />
    <p>Owly the barn owl dropping in for lunch</p>
</a>
```

You can put anything you like in between the link tags. I've got a big picture, a heading, and some text. Maybe you just want a picture and no text!

- Add the following CSS code for the lightbox. Can you work out what some of it does?

```
.lightbox{
    background: rgba(0,0,0,0.8);
    color: #ffffff;
    text-align: center;
    text-decoration: none;
    width: 100%;
    height: 100%;
    top: 0;
    left: 0;
    position: fixed;
    visibility: hidden;
    z-index: 999;
}
```

Note: Setting the `position` property to `fixed` means the position you set will be relative to the browser window, so it will stay put when you scroll.

- Next, decide what thing you want to click to make the lightbox appear, and add add a pair of `a` tags around that element (in my case it's a smaller picture of a lemur). The **target** of the link will be the lightbox, which you set using the `id`. You might recognise this technique from earlier!

```
<a href="#boxBarnowl">
    <img src="barn-owl-landing.jpg" class="mediumPics">
</a>
```

- Finally add the following CSS code. Note that this is a **pseudo-class**; it should go after the code for the `.lightbox` class and not inside it!

```
.lightbox:target {
    visibility: visible;
}
```

The `:target` pseudo-class gets applied whenever the lightbox was the target of the last link clicked. So when you click anywhere, the `visibility` will be set back to `hidden`.

- Try clicking your new link to see the lightbox appear! To make it go away, just click anywhere on the page.

You can add as many lightboxes as you want to a page. They can all use the same CSS class — just make sure each one has a different `id`! For each one, you need to make something on your webpage into a link that you can click to make the lightbox appear, and then use the `id` as the `href` value in that link, just as you've done above!