Fundamentals of Web Design

Third New Edition • Unit 4

BY LUIS POZA

ART280 • Fundamentals of Web Design • Lakeland University

CLASS TEXTBOOK **UNIT 4**, Chapters 13 – 15, Extras

Chapter 13: Forms

TECHNICAL

13a. Form Basics

HTML and CSS are not **programming languages**. A programming language is capable of solving problems, carrying out sets of instructions to achieve specific tasks.

HTML builds the basic structure of a web page for the purposes of display, but it does not directly interact with the user. CSS modifies HTML to display a greater variety of styles, but again, it does not carry out tasks.

We have learned so far that one may use other languages, like CSS, within an HTML document. We also learned that programming languages, like PHP and Javascript, may also be integrated directly into a web page.

However, sometimes it is more useful and organized to put these languages *outside* the HTML document, and create a new document dedicated only to that language. We did this with CSS, in creating a **stylesheet**.

The same is done with programming languages; external documents, called "scripts," are created; these scripts have only the programming language, carrying out a specific task or tasks. Just like the HTML `<link>` tag calls out for the stylesheet to be used, program scripts can be called for as well.

Using HTML, we will create a form on a web page in which a visitor can type the data. Our form will send the data to a PHP script, which will process the data and give us a result—in this case, it will send an email with the form's data.

CLIENT AND SERVER

Once again, let's note the difference between a "client" and a "server."

A **client** is software which runs on a normal computer or device used by an ordinary person. A web browser, for example, is a client. An email program is also a client.

A **server** is software that stores information that people want on a network, and has the job of delivering that data to any client which makes a request for the data.

Actions carried out on networks are usually called "**client-server**," in that they consist of a program on a computer requesting data from servers.

For example, when you use a web browser, you enter a URL for a web site. When you do so, your browser—a **client**—sends out a request for a web page to the address of the web page. The **server** program is at that address. It receives the request, and in response, it collects the HTML, CSS, image, and other files, and sends them back to the client. The client browser then takes those files to create ("render") the web page display on your browser.

Similarly, when you use an email program—a client—it contacts your email server (for example, your LUJ Mail account) and requests information on any new email which has

arrived. Your email server sends the data to your email program. Again, a client-server relationship.

PHP

PHP is a **programming language** of a very specific type. It is a **server-side** language. In other words, PHP is a language which is run on the server, not on the client. Your browser cannot run PHP. It can send a request to the server to run PHP; the server will run the program, then send the result back to your browser, the client.

HTML forms can send requests to various types of programs, not just PHP, but in this class, we will use a PHP script.

Because the PHP script is server-side, it must "live" on the server. Your computer is not a server because it does not have server software installed. The server software *is* installed on the web host, the location which has the web site "on" the Internet.

PHP scripts will not work on your computer. In fact, for security reasons, a PHP form script will only accept input from a form which is on the same server; therefore, although you can *make* and *view* a form anywhere, **you cannot *use* a form until you upload it to the server** using FTP (in our case, Filezilla).

CODE

13b. The Form Tag

We begin with a very basic HTML tag:

```
<form>
</form>
```

Of course, it's not that simple. The form tag must include these **attributes**:

```
<form action="http://lujweb.com/thisone.php" method="POST">

</form>
```

The **action** attribute identifies the location of the PHP script which will be used.

The **method** attribute identifies what will happen. There are two basic ways of interacting: `POST` and `GET`. *(These can be typed in uppercase or lowercase.)*

The GET method makes a **request** for data on the server using the URL string. For example, on the Lakeland "Open College" site, when you request information about a class, it asks the PHP script to send data on a class using a GET code:

```
http://www.lcjoc.com/class.php?cc=0313
```

In this case, "cc" is the "class code" variable, and "0313" is the course number. Notice the question mark right after the basic URL—you have probably seen this before, right? That's the GET method being used to send a data request to a server-side script.

The POST method is used to **deliver** data to the server. In this case, the form data, which will then be processed by the PHP script.

You can add a few more things in the `form` tag, including a request regarding autocomplete:

```
<form action="http://lujweb.com/thisone.php" method="POST"
autocomplete="off">
```

The "autocomplete" is when a form field automatically recalls previous data that was typed into the same field. For example, when you fill in a form, and then later return to the same form, the data you typed in previously will return.

When I give you a web page test, I turn autocomplete off so that answers previous students gave do not appear on your test form!

Browsers can and usually do override the autocomplete-off setting when it comes to login and password fields. Each browser allows the user to turn off this feature.

Script Data

When we use the `form` tag, we are making a request to a script on a server. That script may require special information in order to do its job. For example, in our case, we want our form script to (a) send the form data to you in an email, and then (b) direct the visitor to a "Thank you" page informing them that the message was sent. We are using an older script called "PHPFormMail," which requires specific information from the form in order to work right.

To send data using a form, we usually use the `<input>` tag. "Input" is data sent to a program:

```
<form action="http://lujweb.com/thisone.php" method="post">
   <input type="hidden" name="recipient" value="poza@japan.lakeland.edu">
   <input type="hidden" name="subject" value="Here Is the Form Data">
   <input type="hidden" name="redirect" value="http://lujweb.com/thanks/">

</form>
```

Notice that the input tag has certain attributes. In this case, they are:

`type`	*which type of input; in this case, "hidden" means not visible on the page*
`name`	*the name is an identifier; the script knows what data is by the name*
`value`	*this is the data that the script will use*

In this case, you are giving the script information it needs:

1. Which address should the email be sent to?
2. What should the subject line on the email be?
3. What page should the visitor be sent to after the email is successfully sent?

These are the formal requirements to use our particular script. The script specifically requires these input names to be "recipient," "subject," and "redirect"; if they are not named correctly, the script will not work correctly.

If you use a different PHP script, it may require different data, with different names.

13c. Form Controls

A "form control" is a way of asking the visitor for information in a form. There are many types. The following are used specifically with the `<input>` tag:

`text`	*the standard one-line text box, used for names and other short data*
`password`	*similar to text, but this box hides what you type as bullets*
`email`	*an HTML5 control; it requires the user to input an email address*
`url`	*an HTML5 control; it requires the user to input a valid URL*
`checkbox`	*creates a small box which can be checked or unchecked*
`radio`	*creates a round "button" belonging to a group; only one can be "on"*
`hidden`	*used by the page creator to send needed information to the script*
`reset`	*creates a button that will erase all data in the form; rarely used*
`submit`	*creates a button that will send the form data to the script*

Note that the HTML5 controls are handled differently by different browsers. Some browsers, for example, will not allow a form to be submitted if there is not a valid email address in the "email" type of input control. The input must have text, an @ mark, and then a domain name.

In addition to form types, you must also give the inputs a **name** and an **id** attribute. For example:

```
<input type="text" name="First Name" id="firstname">
```

The **name** and **id** attributes are usually similar, but serve different purposes.

The **name** is used by the script on the server. It is the name that will appear in the email you get when a visitor fills out the form. You can use capital letters and spaces in the name.

The **id** is used by CSS and programming languages to identify the specific object in the HTML code. It can also be used to identify a location on a page for links or other purposes.

The text for name and id probably should not carry spaces; if you want to use multiple words as a name or id, use "CamelCase," such as "`MyFavoriteColor`".

You can also use other attributes:

`size="50"`	*changes the width of the text box, in pixels*
`value="something"`	*adds editable text in the text box*
`placeholder="something"`	*shows gray text as a suggestion; disappears*
`autocomplete="off"`	*turns off autocomplete for one textbox only*
`required="required"`	*in some browsers, requires input for the form to be sent*

The **value** attribute will add text which can be sent in the form. For example, when I give a test, I ask students to fill in their LUJ email address. To help, I enter the value **@japan.lakeland.edu** in the text field; all the student must do is add the user name. However, "value" can also be used for different purposes, as in checkboxes and radios.

In contrast, the **placeholder** attribute shows only gray text which disappears when the visitor begins to type anything. The placeholder text is used to explain the nature of the data, or to give an example of data which should be input.

TEXT INPUT

```
First name: <input type="text" name="First Name" id="firstname">
```

This simply asks for a bit of text to be entered into the form. It can be any kind of text, usually one sentence or less.

PASSWORD INPUT

```
Password: <input type="password" name="Password" id=" password ">
```

This asks for a password to be input. Our form will not need one, but you should know about this. Any text entered into this field will be hidden, shown only as bullets.

EMAIL INPUT

```
Email address: <input type="email" name="Email" id="email">
```

This asks for a valid email address to be input. In some (but not all) browsers, the form cannot be sent until a valid email address (`xxx@xxx.com`) is input.

URL INPUT

```
Your home page: <input type="url" name="Homepage" id="homepage">
```

This asks for a valid web page address (URL) to be input. In some (but not all) browsers, the form cannot be sent until a valid address (`http://xxx.com`) is input.

CHECKBOX INPUT

```
<p>
   Do you have: <br><br>
   <input type="checkbox" name="I have" id="dog" value="a dog"> A dog <br>
   <input type="checkbox" name="I have" id="cat" value="a cat"> A cat <br>
   <input type="checkbox" name="I have" id="hamster" value="a hamster"> A
   hamster
</p>
```

Note: the **name** will appear in the resulting email as the input **title**;

the **value** will appear in the email as the input **response** (the answer given).

In checkboxes and radio buttons, "value" is used to make the form results more clear.

The form control above will create what you can see below: a list of boxes which can be checked. In checkboxes, none, one, some, or all can be checked at the same time.

Do you have:
- ☐ A dog
- ☐ A cat
- ☐ A hamster

RADIO BUTTON INPUT

```
<p>
   Which is your favorite pet? <br><br>
   <input type="radio" name="I like" id="dogs" value="dogs best">Dog<br>
   <input type="radio" name="I like" id="cats" value="cats best">Cat<br>
   <input type="radio" name="I like" id="birds" value="birds best">Bird
</p>
```

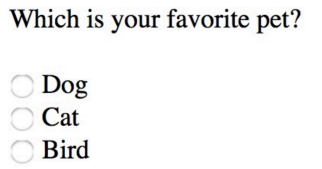
This is similar to the checkbox, except (1) the buttons are round, and (2) **only one button in a group can be selected**. Notice that in this case, the **name** and **id** attributes are different:

The **name** identifies a group of buttons, and appears as the title in the email;

The **id** identifies the specific choice for the script;

The **value** appears in the email as the answer.

Only one button among several with the same "name" can be selected at one time. If you click on Dog, that button will be selected; if you then click on Cat, then Dog will be *unselected* and Cat will become the choice.



Here is what the email result would look like for the above checkbox & radio code:

RESET INPUT

```
<input type="reset" value="Reset the form!">
```

This button will clear (erase) all data that was filled in on the form. Note that there is no name or id, but you can give a **value**, which will appear as the text name of the button.

This button is rarely used. It was included because it was believed that people might want to change all the data in a form, but in practice, it rarely happens. Instead, this button was a nuisance because it often was placed right next to the "Submit" button, and many people would accidentally click it and erase all their carefully typed information!

SUBMIT INPUT

```
<input type="submit" value="Send the Data!">
```

This button will execute the form, sending the data that was filled in. Note that there is no name or id, but you can give a **value**, which will appear as the text name of the button.

STYLING INPUT CONTROLS

You can add CSS styles to the form controls, including text boxes, buttons, and more, so long as they use the `<input>` tag. You can change the font and text properties in text boxes, and you can style the "Submit" button to be whatever you like.

In order to do this, just create a rule with the selector `input[type=oooo]` as seen here:

```
input[type=submit] {
    border: 1px solid black;
    color: white;
    background: red;
}
```

There are other form controls which do not use the `<input>` tag:

TEXTAREA

```
<textarea name="comments" id="comments" cols="60" rows="4"> </textarea>
```

This tag creates a form control which is a multi-line text input area.

Note that you can define the size with the attributes `cols` and `rows`. The "`cols`" attribute sets the width, and the "`rows`" attribute sets the height. One "col" is a standard-width character, but since web fonts are usually proportional, fewer or more than that number of characters may fit. One "row" is a line of text, so `rows="4"` would accommodate four lines of text.

You can enter more than four lines, but the text will begin to scroll. Most browsers create a resize handle at the bottom right of textareas, allowing the user to freely resize them.

Alternately, you can set the width and height in CSS, using any measurement unit you wish.

An important note about the `<textarea>` tag: **any characters, including whitespace characters, which are between the start and end tags will be recreated inside the textarea on the web page**.

Normally, when we have a block tag, we put a space between the start and end tags. With textarea, you would not do this anyway, as it is an inline tag. However, we now have another reason: if you put an empty line between the tags, then one or two lines of empty space will be added inside the textarea box.

Alternately, any text put between the two tags will also appear inside the box on the web page. As a result, there is no need to use the `value` attribute for this form control—just type the text between the tags, and it will appear.

SELECT & OPTION

```
<p>
   What class would you like to take?
   <select name="I want to take" id="takeclass">
      <option value="Mass Media" selected>Mass Media</option>
      <option value="Anthropology of Japan">Anthropology of Japan</option>
      <option value="Microeconomics">Principles of Microeconomics</option>
      <option value="Web Design">Web Design</option>
      <option value="US-Japan Relations">US-Japan Relations</option>
   </select>
</p>
```

This set of tags will create a **drop-down menu** from which one option can be selected. The `name` attribute in the `select` tag will be used as the title in the result, and the value of whatever option is selected will be the value in the result.

Notice that the "value" attribute does not have to exactly match the value shown in the menu, which is the text between the option tags.

You can add the size attribute in the "select" tag allows you to change how the menu works. If you set the size to "1" then the menu will be a standard drop-down menu.

However, you can also set the list to have a multiple size, in which case a selection box appears (see right). The selection box will not drop down.

There are various ways to set this:

```
<select size="6" ...
<select multiple="multiple" ...
<select multiple ...
```

If you use the `multiple` attribute (the "multiple" value is optional), then the default size of the box will be 4 lines; if there are more than 4 choices, then the box will scroll.

If you set the `size` attribute to more than 4, then the height of the box will increase.

Note that in some browsers, if the `size` attribute is set to 2 or 3, the default value of 4 will be used instead.

In all types of select menus, you can set width and height using CSS; this will override all defaults.

Note also that you can pre-select a favored option by using the `selected` attribute:

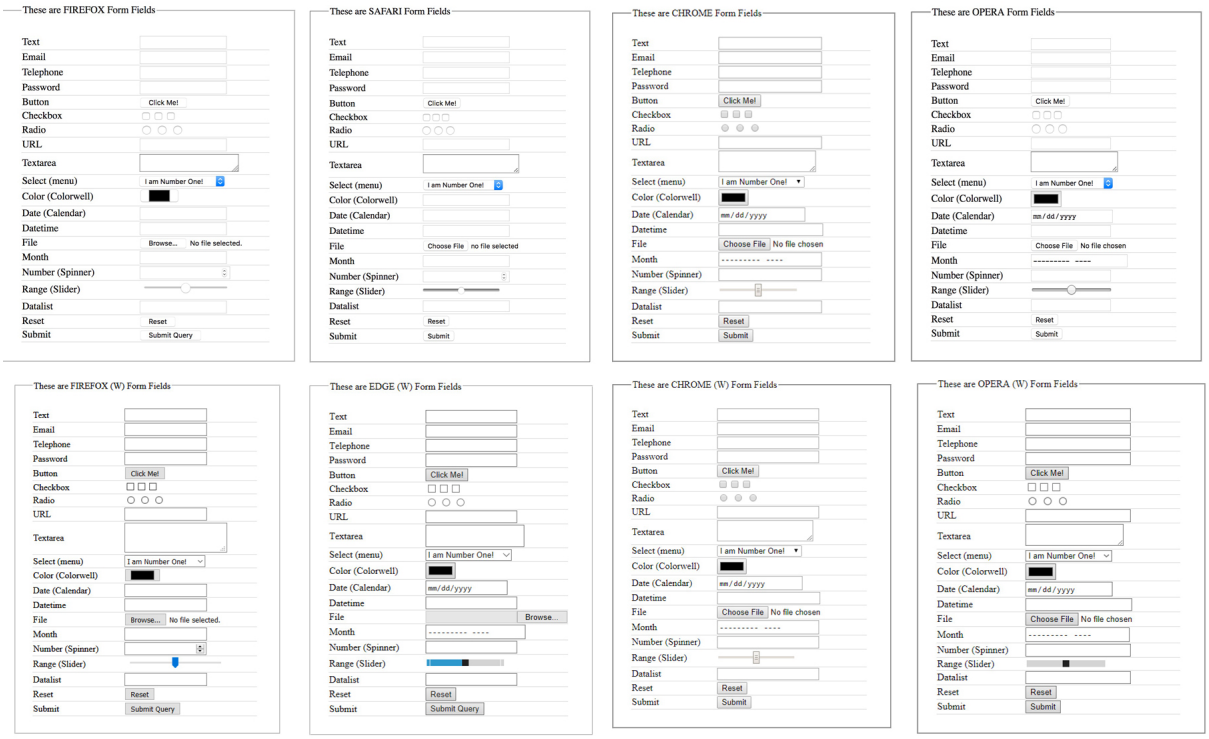
```
<option value="Mass Media" selected>Mass Media</option>
```

This can be expressed as `selected`, or `selected="selected"`.

If you want to be friendly to much older browsers, then `selected="selected"` and `multiple="multiple"` is better, but the attributes without values should work anyway.

13d. Other HTML5 Controls

In HTML5, new form controls were introduced. However, not all browsers support these controls, and even the ones that do will display them differently:



Notice that Safari on a Mac in particular is "blind" to the new form types.

New controls include:

- color
- date
- datetime
- month
- number
- range
- datalist

Because these are (1) specialized, and (2) not yet universal, we will not be studying them in this class. However, if you want to explore these, just do a search for them on the web. W3 Schools usually has very good information.

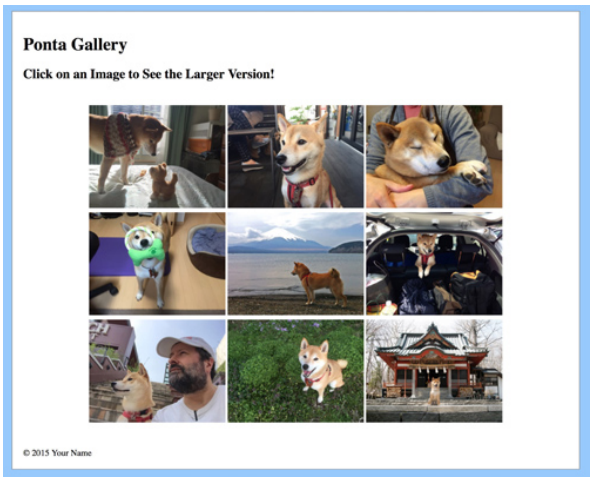
Chapter 14: Image Galleries

14a. Older Image Galleries

There are various ways of doing this, of course.

Thumbnail Table Gallery

The simplest is the most plain: have a grid of thumbnail images (smaller versions of the images, no more than 250px wide) in a table. People click on a link, they go to a page with the larger image. On that page, there are buttons for previous, gallery, and next. I am giving you the site within the gallery folder.



This is the type of gallery that was popular in the 90's. It only requires a table and some links, and then a number of individual pages with the larger images.

First, create a main gallery page with a table wide enough to hold the thumbnails (in this example, 3x3), and then place the thumbnails in the table data cells (the `<td>` tags). Make each thumbnail image into a link to a web page that will hold the larger version of the image.

Next, create an image page. This is also a table, with three rows. The first two rows, the title and image rows, have one cell, a `colspan`. The last row will hold the links. It can be three or five cells across, and the links point to the previous image's page, the main gallery page, and the next image's page.

Then copy the image page as many times as there are images, and on each page, change the addresses for the images and links. For example, I made the full-size pages have names "pon01.html", "pon02.html", etc. Inside each one, I would change the `img src` to the correct image, and adjust the "previous" and "next" links to work correctly.

The upside to this is that it works well. The downside to this is that it looks old-fashioned, and it requires you to do more work, creating all the individual image display pages.

14b. Simple JavaScript Image Gallery

There is a way to make image galleries with CSS, but there is a drawback to them: the images do not stay. You can hover over a thumbnail image and a larger image appears—just like with drop-down menus—but the larger image only remains visible as long as you hover.

Therefore, to make the kind of gallery most people are used to, we have to use JavaScript—which is why we learned a little about JavaScript before this chapter. JavaScript allows us to change the properties of objects that are separate from the trigger, or the "button" that a user clicks on.

We are used to galleries where a number of thumbnails appear in a row, and when you click on one, a larger image appears. This is what we will build.

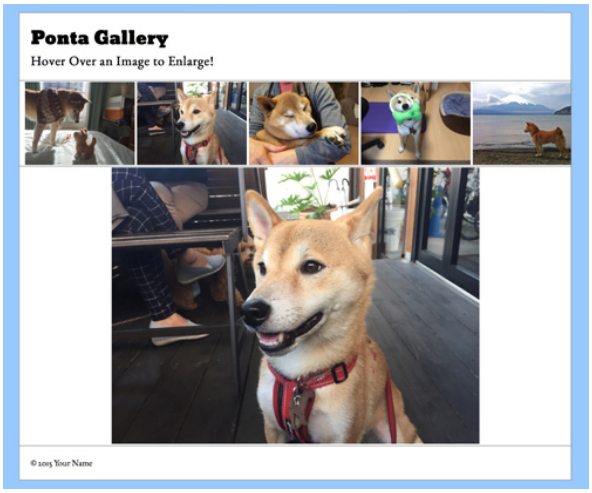
Step 1

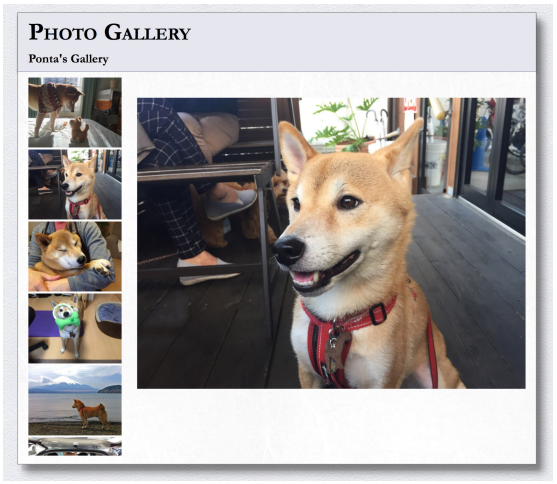
Create a page that will hold the gallery. Depending on your site, it may simply be an empty page with the same layout as other pages on that level of that site. Take special note of the available space, in particular the width of the area the gallery will occupy.

In our current exercise, we will be working with an available space 1000px wide.

Step 2

Decide your gallery layout. You should decide where the thumbnails will go relative to the location in which the larger image appear. Here are two example layouts:

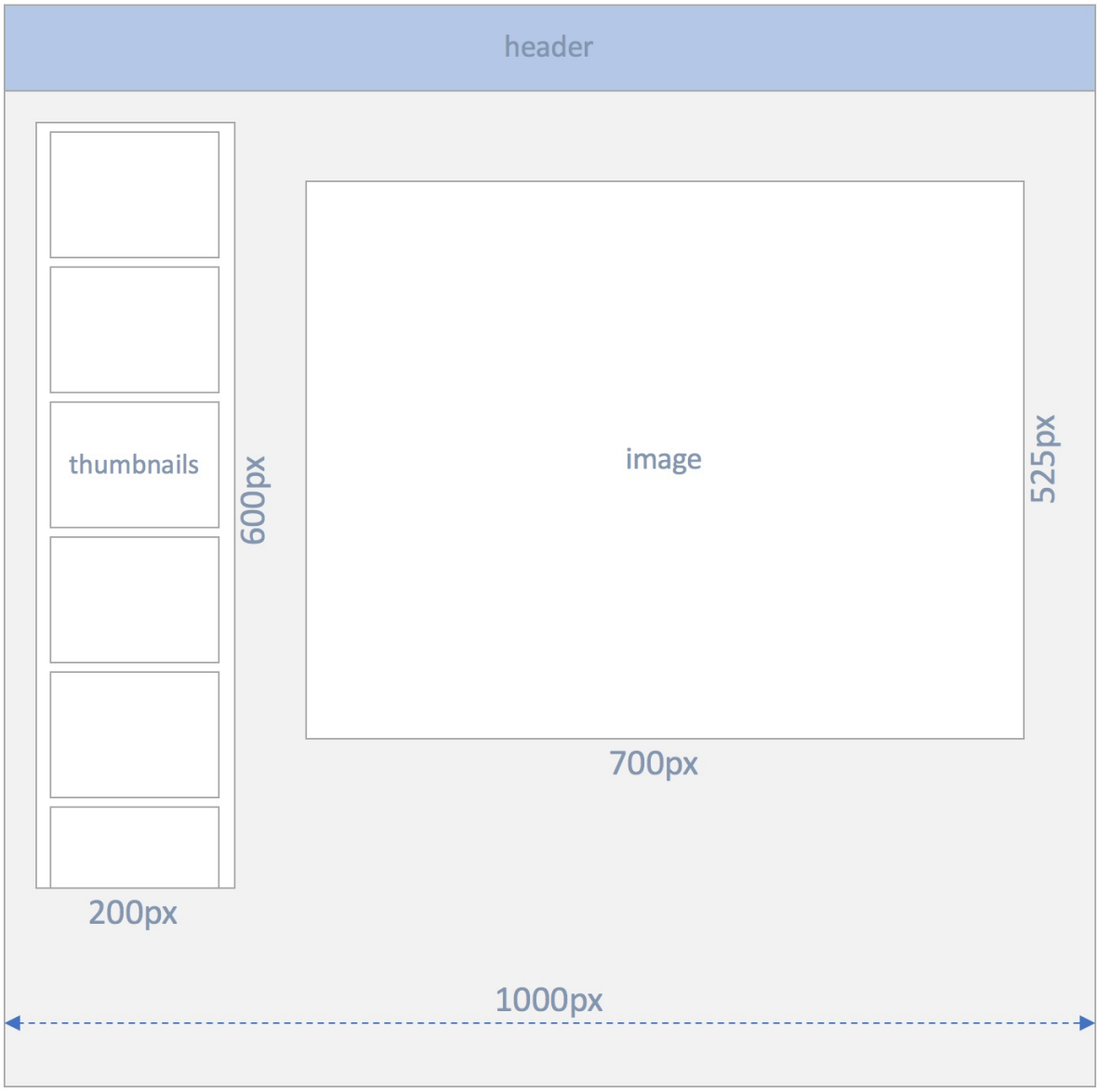




ou can have the thumbnails along the top, the bottom, the left, or the right. My preference, and the layout we will do in this exercise, is to have the thumbnails appear along the left side. If the thumbnails appear along the top, then the larger image may be down far enough that it is not fully visible on a smaller screen; if the thumbnails are on the bottom, they might get cut off. You can decide whatever you like when you create your own gallery, but always keep in mind how the gallery will appear on different sized displays.

Step 3

Calculate your sizes. Determine how tall and wide your images will be, and how tall and wide the larger image will be. Here is a basic plan for our page, 1000px wide:



In this plan, the thumbnails will be 175px wide, inside a container 200px wide by 600px tall. The container will be set to **overflow: scroll;** so the thumbnails will scroll up and down. The larger image will be 700px wide, so there will be 50px left for spacing between the elements.

Step 4

Collect your images. The images should be all **the same size and shape**. If possible, all images should be either portrait, or all should be landscape. It is possible to have a mix, but it makes the coding and styling somewhat more difficult. In this exercise, we are using all landscape.

We want the thumbnails to be 175px wide, and the large images to be 700px wide. This means that first, we must collect all the images, and make sure they are at least 700px wide.

They can be a little larger, but not too much. In our case, we are using images 1000 x 750 pixels, or a size ratio of 4:3.

If your images are different sizes, they must still have the same ratio. If some are wider and some are taller, it will create an imbalance in the image sizes which will make the gallery look worse. It is best to use an image editing program like Photoshop, Preview, or Pixlr to resize and perhaps crop the images so they are all the exact same size and ratio.

In addition, the images should all be named similarly; in our case, pon01, pon02, etc.

Step 5

Create thumbnails from the larger images. You should *not* use the large images as thumbnails. Some people just use HTML or CSS to resize the main images to thumbnail size. The reason this is not good is that the larger images will be too big in file size, and that will make the thumbnails load too slowly.

Therefore, you must open all the images in an image editing program and resize them (there are programs you can download which can batch-resize all images quickly). Again, they should all be the same size. In my case, the 700 x 525 images resize down to 175 x 130.

Again, the images should all be named similarly; it is best that the thumbnails have the exact same names as the larger images.

Create two new folders in your project page: *small* and *large*. Put the larger images in *large*, and the thumbnails is *small*. When you open the folders, the files should look identical—same names, same images. The only difference will be the size.

Step 6

Create the layout. The gallery will appear in a single block, probably the **main** or the **article**, depending on which you use. In this case, we are using a `<main>` tag.

Within the main tag, we will have two `<div>` tags, one for the thumbnails and one for the larger image. The basic code will look like this:

```
<main>
    <div class="slider">

    </div>
    <div id="imgdisplay" class="default">

    </div>
</main>
```

The "slider" will hold the thumbnail images; the "imgdisplay" will show the image full size. The `class="default"` will load the first image of the series automatically.

STEP 7

Create CSS rules for the layout. First, since the larger image will be placed with absolute positioning, but we want the `<main>` tag to be the boundaries, we must use the "absolute capture" method. To do this, the main must be set to `position: relative;` which will contain the absolute object.

Second, we must name the two `<div>` tags. I called the thumbnail holder the *slider* (because the thumbnails will appear to slide up and down when they scroll), and the larger image holder I called *imgdisplay*. Note that "slider" is a class, and "imgdisplay" is an id.

```
main {
    position: relative;
}
.slider {
    width: 200px;
    height: 600px;
    padding-top: 10px;
    overflow-y: scroll;
    overflow-x: hidden;
    border-right: 1px solid gray;
}
.slider img {
    width: 175px;
    padding-left: 10px;
}
#imgdisplay {
    width: 700px;
    height: 525px;
    position: absolute;
    top: 30px;
    right: 50px;
    border: 1px solid gray;
    background-size: 700px 525px;
}
```

Note that I set the slider to be 200px wide and 600px tall. The padding-top is to create a small space at the top of the thumbnail list so it does not "hit" the top of the box.

Notice that I created two different overflows: an *overflow-y* (vertical) which will scroll, and an *overflow-x* (horizontal) which will not. If you have scrolling left & right, then a horizontal scrollbar may appear and you may see movement left and right, which we do not want.

There is a right border only because in our layout, the slider will be the tallest element, so that right border will hit the borders of the main at the top and bottom.

In the class *.slider img* we simply set the width of the thumbnails (the height will bet set automatically), and create padding along the left. (If you wish the padding could be in the slider div instead.)

For the div to display the larger image, **in an id** we set the width and height (700 x 525), make the position absolute with 30px space from the top and 50px from the right. You can adjust these later if you wish. The border is optional, but you must set the background-size to 700px by 525px so that the image will display exactly the right size no matter what the original image resolution is (though the ratio must be the same!).

At this point, I should explain how this gallery is going to work.

As you could see, we set up a list of thumbnails, and a div which will hold the larger picture. The trick will be to make the larger image appear in the div when we click on a thumbnail image. That larger image should stay visible until a different thumbnail is clicked. With just HTML & CSS, this is not possible.

Therefore, we will use JavaScript. We will give each thumbnail a JavaScript Mouse Event (`onMouseClick`) which will use the `SetAttribute` method. This will change the class of the larger image div to have a background image of the image we wish.

Notice that when we styled the larger image div, we used an id (`imgdisplay`); that means that there is no class, so we can use a class to just set the background image. By changing the "class" attribute every time we click on a thumbnail, a different background image will appear. If you do not understand this yet, it's OK; it should become clear as we do this.

STEP 8

Create different background images using classes. Here, we will create the classes that will be used to decide which image will be the background for the larger image div.

```
.default {
    background-image: url(images/large/pon01.jpg);
}
.pic01 {
    background-image: url(images/large/pon01.jpg);
}
```

The *.default* class will be the normal background, which will be the image shown in the first thumbnail. We will have the same background image in a class called *.pic01* which will be activated when the first thumbnail is clicked.

Next, copy and paste the *.pic01* class rule as many times as you have images, and then change the numbers of the selectors and the image names to match the image numbers.

```
.pic01 {
    background-image: url(images/large/pon01.jpg);
}
.pic02 {
    background-image: url(images/large/pon02.jpg);
}
.pic03 {
    background-image: url(images/large/pon03.jpg);
}
.pic04 {
    background-image: url(images/large/pon04.jpg);
}
.pic05 {
    background-image: url(images/large/pon05.jpg);
}
.pic06 {
    background-image: url(images/large/pon06.jpg);
}
.pic07 {
    background-image: url(images/large/pon07.jpg);
}
.pic08 {
    background-image: url(images/large/pon08.jpg);
}
.pic09 {
    background-image: url(images/large/pon09.jpg);
}
```

Step 9

Create the thumbnail images. You can create one and then copy and adjust. The image tags for the thumbnails should be placed inside the "slider" div.

```
<img src="images/small/pon01.jpg" alt="first Ponta picture">
```

To this, we will add the JavaScript `onClick="imgdisplay.setAttribute('class','pic01')"`:

```
<img src="images/small/pon01.jpg" onClick="imgdisplay.setAttribute('class','pic01')" alt="first Ponta picture">
```

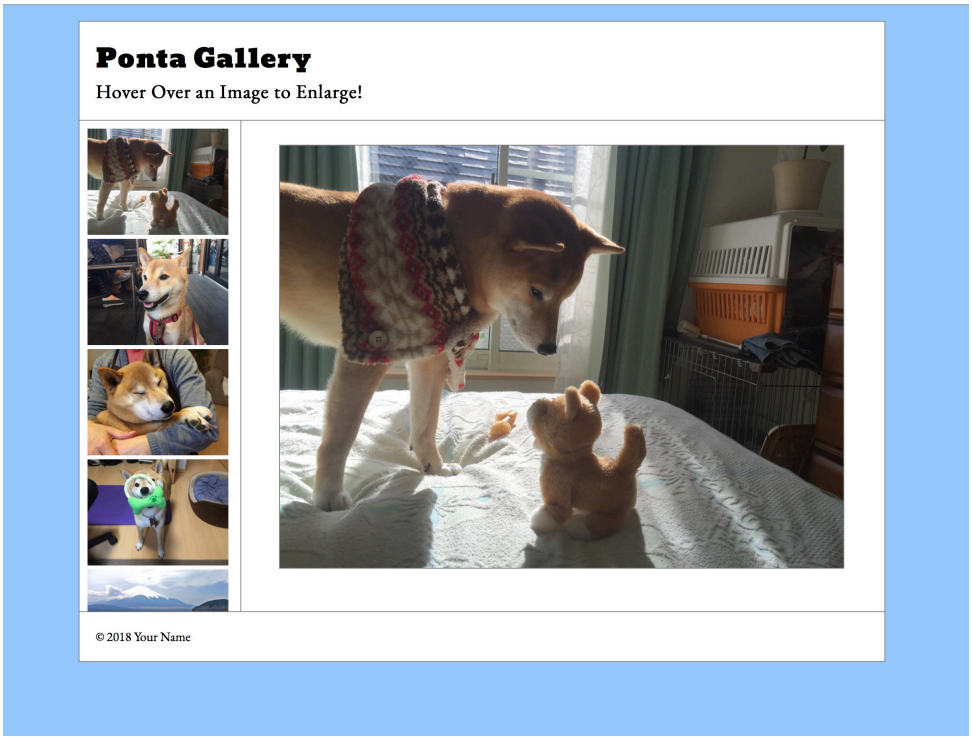
This JavaScript means that when you click the image, the tag with the id "imgdisplay" will have a new class attribute set. The new class will have a background image matching the thumbnail that is clicked on.

After that, just copy, paste, and change the numbers:

```
<img src="images/small/pon01.jpg" onClick="imgdisplay.setAttribute('class','pic01')" alt="first Ponta picture">
<img src="images/small/pon02.jpg" onClick="imgdisplay.setAttribute('class','pic02')" alt="second Ponta picture">
<img src="images/small/pon03.jpg" onClick="imgdisplay.setAttribute('class','pic03')" alt="third Ponta picture">
<img src="images/small/pon04.jpg" onClick="imgdisplay.setAttribute('class','pic04')" alt="fourth Ponta picture">
<img src="images/small/pon05.jpg" onClick="imgdisplay.setAttribute('class','pic05')" alt="fifth Ponta picture">
<img src="images/small/pon06.jpg" onClick="imgdisplay.setAttribute('class','pic06')" alt="sixth Ponta picture">
<img src="images/small/pon07.jpg" onClick="imgdisplay.setAttribute('class','pic07')" alt="seventh Ponta picture">
<img src="images/small/pon08.jpg" onClick="imgdisplay.setAttribute('class','pic08')" alt="eighth Ponta picture">
<img src="images/small/pon09.jpg" onClick="imgdisplay.setAttribute('class','pic09')" alt="ninth Ponta picture">
```

Step 10

Save and view the page! You should be done!



Chapter 15: Animation

15a. Transform

One trick we haven't learned yet is the **transform** property in CSS. This allows you to change the appearance of objects you place in your web page.

The different effects that you can get with transform are:

```
transform: translate(100px,50px);
```

"Translate" means "move." This is similar to `position: relative;` and the use of the `top`, `bottom`, `left`, and `right` properties. With `translate()`, you can move the object in two dimensions: the first number (100px above) is to the right, while a negative number goes to the left. The second number (50px above) moves the object down, with a negative number moving it up.

Note that, like position: relative, the translate function keeps the object in the normal flow, and just moves the visible object while the original location of the object is preserved.

```
transform: translate(100px,50px);      object goes right and down
transform: translate(-100px,50px);     object goes left and down
transform: translate(100px,-50px);     object goes right and up
transform: translate(-100px,-50px);    object goes left and up
```

```
transform: scale(2,1.5);
```

"Scale" means "resize." This is similar to changing the width and height. The numbers are for the horizontal ("x") and then vertical ("y"). In the example above, the object will become 200% (2x) the width, and 150% (1.5x) the height.

In the case of transform: scale, the expansion goes from the center outwards in all directions, and not to the right and down as is expected in normal flow.

Note that, like position: relative, the translate function keeps the object in the normal flow, and just moves the visible object while the original location of the object is preserved.

```
transform: rotate(45deg);
```

The rotate function allows you to do exactly that: rotate the object. If you begin with an image of an arrow point to the right, it is at the default of 0deg. If you set the rotate for 90deg, then the arrow will point up. 180deg is rotated to be reverse pointing left, 270deg is pointing down, and 360deg takes you back to the start.

If you set the rotate for more than 360deg, and add an animation, the object will spin 1x for every 360deg increment. For example, if a rule says that an object is `transform: rotate(0deg);` and then a hover says `transform: rotate(720deg);` then the object will spin counter-clockwise two full times. You can speed up or slow down the spin by changing the animation duration.

```
transform: skew(20deg, 0deg);
```

The skew function will tilt the object along the horizontal and vertical.

If you skew the first value (the "x" value, 20deg above), it will skew the image horizontally, so that the top of the object will move left, while the bottom moves to the right; it looks a little like the object may be falling over, but as you get up to 90deg, the width of the object becomes extreme.

In the image at right, the skew is (20deg, 0deg). If you skew the x more, the top keeps moving left and the bottom keeps moving right. At 90deg, the image disappears, and then from 91deg to 180deg the image skews back to the original size, this time the top coming from the right, and the bottom from the left.

Skewing the y-axis (the second value) moves the left side up and the right side down; the transform show just below will produce the image seen at right:

```
transform: skew(0deg, 20deg);
```

The same effect happens with 90deg making the object disappear and then up to 180deg, the image comes back from the other direction.

You can combine the x and y skews for various rotational effects. It is a bit hard to use.

```
transform: rotate(20deg) skew(0deg,20deg) scale(1.4,2);
```

You can combine more than one transform, the same way you can combine more than one image filter (see Chapter 9f).

Note: these transforms can work both with the `:hover` pseudo-element, and with the `@keyframes` animation. The keyframes may work better, as these effects cause objects to move and distort, and the exact hover location may be hard to keep. The `@keyframes` will animate more smoothly.

15b. Hover and Transition

There are two relatively easy forms of animation in CSS: using either the **transition** property, or using **@keyframes**.

The transition property is simpler: using the **:hover** pseudo-class, you can make the change between the normal state and the hover state smoothly animated.

```
#idname {
    position: relative;
    transition: all 0.5s ease;
}
#idname:hover {
    left: 200px;
    transition: all 1s ease 1s;
}
```

In the example above, an object (like an image or a div) has a relative position, which by itself does not change anything. In the hover, you make it move by 200px from the left to the right. Normally, this would make the object suddenly jump 200 pixels to the right if you hover the mouse over it.

By adding the **transition** property, you can make the change into a smooth animation.

To make the transition work both for mouse-in and mouse-out, all you need is to add the transition property to the original (not the hover) rule. This will animate both sides of the hover. If you add the transition declaration only to the hover rule, then the animation will only work on mouse-in, and not mouse-out. If you wish to have different animations for mouse-in and mouse-out, then place the mouse-in effect in the hover rule, and the mouse-out effect in the original rul.

Note that some changes do not work well with transition, for example **font-size** (it will appear jumpy), or **background image** (Firefox won't do it smoothly)

In older versions of browsers, it was necessary to add the transition declaration to both sides, the original rule and the hover; with current browsers, the transition only needs to be in the original rule. However, you may want to put the transition in both (as shown above) so people with older browsers will still see the animation OK. The shorthand for transitions is:

```
transition: <property> <duration> <timing-function> <delay>;
```

Property means which CSS properties will be affected. Usually, you just use **all**;
Duration means how long the whole animation takes; this is in seconds, e.g. **10s**.
Timing function means how the animation will flow; the most-used properties:

ease	(slow start, fast middle, slow end)
ease-in / *ease-out*	(ease-in only starts slowly, ease-out only ends slowly)
linear	(exactly same speed from start to end; looks less natural)

Delay means that the change will wait a number of seconds before happening, e.g. **3s**.

15c. @keyframes Animation

Keyframes are a bit more tricky, but they are more powerful. They do not need a trigger; you can set them up to happen independently, on a schedule you decide. Also, they are not just on-and-off like transitions; instead, you can program them to appear over multiple steps.

Keyframes require two parts:
1. The animation property
2. The @keyframes definition

ANIMATION PROPERTY

The **animation** property has these parts:

```
animation: <name> <duration> <timing-function> <delay> <iteration>
```

There are a few more, but these are enough for now. What they mean is:

name	the name of the animation keyframe (it can be anything)
duration	how long it lasts in seconds (e.g., 10s)
timing-function	what speed the animation has at the start, middle, and end
delay	how long a delay will be added before the beginning (e.g., 3s)
iteration	how many times will the animation play (a number or "infinite")
direction	will it *alternate* or *reverse* direction? If neither, leave this out.

In use, the animation looks like:

```
selector {
       animation: myanimation 10s ease 2s infinite alternate;
}
```

This is half of the setup (the easier half). The other half is the **@keyframes**, which uses the name that you set in the animation declaration.

@KEYFRAMES

The @keyframes part will tell the browser (1) what happens in the animation and (2) when it happens.

To set what happens in an animation, the keyframe sets **steps** which the animation takes. These are specific directions. They say what to do and when to do it.

If your animation has only two parts (start one way, end another way), then you can use **from** and **to**:

```
@keyframes myanimation {
    from {background-color: yellow;}
    to {background-color: red;}
}
```

1. The **@keyframes** defines it as a keyframe;
2. The **myanimation** is the name set in the **animation** declaration;
3. **From** shows what the style is at the start;
4. **To** shows what the style is at the end.

In this case, the background color will change smoothly from yellow to red (with orange in between). You can, of course, use any CSS declaration—and multiple declarations:

```
@keyframes myanimation {
    from {width: 200px; background-color: yellow;}
    to {width: 300px; background-color: red;}
}
```

MULTI-STEP KEYFRAMES

Alternately, you can use percentages to show multiple steps:

```
@keyframes myanimation {
    0%   { width: 200px;}
    15%  { width: 300px;}
    25%  { width: 400px;}
    40%  { width: 500px;}
    60%  { width: 350px;}
    100% { width: 2px;}
}
```

The percentages set what the state of the object will be at x% of the duration of the animation. Each percentage statement is a "selector."

For example, if you have an animation with a duration of 10 seconds (10s), and then you use this animation:

```
@keyframes tester {
    0%   { margin-left: 0px;}
    50%   { margin-left: 500px;}
    100%   { margin-left: 0px;}
}
```

The object will begin with no margin, then over 5 seconds, it will increase the left margin to 500px, making it seem like the object is slowly moving to the right. At 5 seconds, it will stop, and then take another 5 seconds to go back to the original state.

If you want to animation to pause, just make two selectors which are the same; in the animation below, the object will seem to move for 4 seconds, stop for 2 seconds, and then take another 4 seconds to move back:

```
@keyframes tester {
    0%   { margin-left: 0px;}
    40%   { margin-left: 500px;}
    60%   { margin-left: 500px;}
    100%   { margin-left: 0px;}
}
```

Example of Keyframe Animation

Here is an id with an animation called "jumpingout":

```
#ani1 {
      position: relative;
      height: 200px;
      width: 300px;
      background-color: green;
      animation: jumpingout 10s ease infinite alternate;
      animation-fill-mode: forwards;
}

@keyframes jumpingout {
    0%   { left: 0px;}
    10%  { left: 20px;}
    20%  { left: 20px;}
    40%  { left: 500px;}
    60%  { left: 500px;}
    100% { left: 0px;}
}
```

What does this mean? It means that over a 10-second period of time, the following will happen:

1. Between the start and 1 seconds, the object will move 20 pixels to the right;
2. Between seconds 1 and 2, the object will pause;
3. Between seconds 2 and 4, the object will move from 20px to 500px rightwards;
4. Between seconds 4 and 6, the object will pause again;
5. Between seconds 6 and 10, the object will move back to the starting place at left.

Note that I used "animation-fill-mode: forwards;" for this animation. Normally, after an animation plays, it resets to the beginning. Adding this declaration keeps the end state.

Uses of Keyframe Animation: Fading Image Box

You can use keyframe animations for almost everything except background images. One common use it to change the **opacity** of an object. At `opacity: 0;` the object will be invisible; at `opacity: 1;` the object will be visible; using animation, it become a fade-in/out.

Therefore this following 2-second keyframe sequence will start with an image being invisible; it will slowly become visible over 3 seconds; it will stay visible for 6 seconds; then it will slowly fade out over another 3 seconds; then it will stay invisible for 8 seconds:

```
@keyframes image01 {
    0%   { opacity: 0;}
    15%  { opacity: 1;}
    45%  { opacity: 1;}
    60%  { opacity: 0;}
    100% { opacity: 0;}
}
```

You can use this to make a rotating/fading image box:

- you can have several images inside a block container (`div` or `figure`);
- the block can be set to `position: relative;` so it will 'capture' the images inside of it;
- all images can be set to `position: absolute;` so they all will appear in the same position;
- each image will have a different id with its own animation;
- each animation will "activate" the image for a certain period of time.

For example:

```
0sec          5sec          10sec         15sec         20sec         25sec
|             |             |             |             |             |
image 1   FADEOUT                                            FADEIN image 1
           FADEIN image 2   FADEOUT
                            FADEIN image 3   FADEOUT
                                               FADEIN image 4 FADEOUT
```

```
#fader {
    position: relative;
    width: 800px;
    height: 571px;
    border: 1px solid black
}
#fader img {
    position: absolute;
    top: 0px;
    left: 0px;
}
```

The images in the fader are absolute so they will stack on top of each other. The "fader" box (made to be the size of the images, 800 x 571) is relative so that when you make the images have an absolute position (top and left are 0px), they will stay inside the fader box.

```
#img1 {
    animation: image01 25s ease infinite;
}
#img2 {
    animation: image02 25s ease infinite;
}
#img3 {
    animation: image03 25s ease infinite;
}
#img4 {
    animation: image04 25s ease infinite;
}
```

Each of the above ids are for each image. the "image01" names are the names of the keyframe animations (see next page) Each is set to a 25-second cycle.

Below are the keyframe settings. Note that as each one fades to opacity: 0; the next one is fading in. There is a little bit of overlap so that the background doesn't peek through. Note that each animation gives the name (e.g., image01) which was set in the ids (e.g., #img1) on the previous page.

```
@keyframes image01 {
    0%   { opacity: 1;}
    20%  { opacity: 1;}
    30%  { opacity: 0;}
    85%  { opacity: 0;}
    100% { opacity: 1;}
}
@keyframes image02 {
    0%   { opacity: 0;}
    15%  { opacity: 0;}
    25%  { opacity: 1;}
    45%  { opacity: 1;}
    55%  { opacity: 0;}
    100% { opacity: 0;}
}
@keyframes image03 {
    0%   { opacity: 0;}
    40%  { opacity: 0;}
    50%  { opacity: 1;}
    65%  { opacity: 1;}
    75%  { opacity: 0;}
    100% { opacity: 0;}
}
@keyframes image04 {
    0%   { opacity: 0;}
    60%  { opacity: 0;}
    70%  { opacity: 1;}
    90%  { opacity: 1;}
    100% { opacity: 0;}
}
```

Another Example: Image Sliding

In this example, keyframe animations are used to make images slide over each other. This requires a few more settings:

`z-index: 10;`	This places objects above or below each other; items with higher numbers are above ones with lower numbers
`overflow: hidden;`	This hides images as they slide out of the box

The code for this is exactly the same as the fader box, except that you add the `overflow: hidden;` to the main div. The declarations in #img1, #img2, etc. are exactly the same.

In this case, each image must be on top (z-index: 100;), then it must move down (z-index: 0;) so the next image can slide in on top of it (left: 0;); after that happens, the now-hidden image slides out to the right and waits its turn before coming back in again. You have to carefully think out the positions and timings to make it work right, and sometimes you need to correct and adjust when you write the code.

The keyframe code looks like this:

```
@keyframes image01 {
    0%   { left: 0px; z-index: 100; }    /* Starts in & on top */
    20%  { left: 0px; z-index: 100; }    /* Stays on top for 20% */
    25%  { left: 0px; z-index: 0; }      /* Move below the next image */
    35%  { left: 850px; z-index: 0; }    /* Slide the image out */
    95%  { left: 850px; z-index: 100; }  /* Wait outside */
    100% { left: 0px; z-index: 100; }    /* Slide the image back in */
}
@keyframes image02 {
    0%   { left: 850px; z-index: 0; }    /* Starts out of the box */
    20%  { left: 850px; z-index: 100; }  /* Waits its turn */
    25%  { left: 0px; z-index: 100; }    /* Slide the image in */
    45%  { left: 0px; z-index: 100; }    /* Stays on top for 20% */
    50%  { left: 0px; z-index: 0; }      /* Move below the next image */
    60%  { left: 850px; z-index: 0; }    /* Slide the image back out */
    100% { left: 850px; z-index: 0; }    /* Waits again */
}
@keyframes image03 {
    0%   { left: 850px; z-index: 0; }    /* Starts out of the box */
    45%  { left: 850px; z-index: 100; }  /* Waits its turn */
    50%  { left: 0px; z-index: 100; }    /* Slide the image in */
    70%  { left: 0px; z-index: 100; }    /* Stays on top for 20% */
    75%  { left: 0px; z-index: 0; }      /* Move below the next image */
    85%  { left: 850px; z-index: 0; }    /* Slide the image back out */
    100% { left: 850px; z-index: 0; }    /* Waits again */
}
@keyframes image04 {
    0%   { left: 0px; z-index: 0; }      /* Starts the box below img1 */
    10%  { left: 850px; z-index: 0; }    /* Slide the image  out */
    70%  { left: 850px; z-index: 100; }  /* Waits its turn */
    75%  { left: 0px; z-index: 100; }    /* Slide the image in */
    100% { left: 0px; z-index: 0; }      /* Stays on top for 20% */
}
```

Domain Registration, cpanel, & FTP

1. Go to **namecheap.com**
2. Click "Sign up"
3. Choose a username (can be anything); write it in the space below
4. Choose a password (use letters, numbers, capital letter, symbol);
   write it down in the space below so you won't forget!
5. Fill out the rest of the form, un-check the "sign me up" option
6. Click "Create Account and Continue."
7. Keep that tab open; go to a new tab, and go to **nc.me**
8. Type in the name of the domain you want, click "Search."
9. When you find a name that is available, click "ADD"
10. Click "Complete order"
11. On the next page, near the bottom, enter your LUJ email address; click "Finish up"
12. In a third tab, open Gmail, sign in to your LUJ Mail account
13. Open the message which is titled, "Namecheap – Please confirm your email"
14. Click on the "Verify your email" button
15. On the page saying your email has been verified, enter the username & password you chose in steps #3 and #4, then click "Login"
16. On the confirm page, enter an address & phone number (it can be fake, or you could use the LUJ address & phone number: **1-10-5 Yokoami, Sumida-ku, Tokyo, 130-0015 Japan, +81-3-6240-4243**, then click "Confirm Order"
17. After you get the "Thanks" page, you can go back to the first tab at namecheap.com and check your account; if you reload, you should see the new domain appear.

namecheap.com Sign-up

Username: _____ Password: _____

Username: _____ Password: _____

Username: _____ Password: _____

nc.me Sign-up

Domain name: _____.me_____

lujweb.com Sign-up

Cpanel address: **lujweb.com/cpanel**
Host: **lujweb.com**
User name: **lujweb**
Password: (see email)

Setting Up Your Domain

In order for a web site to work, you need two things:

1. A domain name
2. A web server

The domain name will be the "shortcut" to the IP Address. The IP address is the address of the web server. The web server computer has your web site files. The domain and the web server connect through the DNS system.

The web server requires many things in order to function:

1. A computer which is on all the time
2. A high-speed Internet connection (including high upload speeds)
3. A static (unchanging) and dedicated (private) IP address
4. Web server software—for example, Apache
5. Someone to install and maintain the software and server

All of these can be expensive to buy and difficult to maintain. Therefore, it is much easier to use a web host. A web host will provide all of these services for a monthly fee. For reliable service, plans costing at least $10 a month are usually required. Lakeland College Japan has purchased a web hosting account, and everyone is free to use it for one year.

The Process

These are the steps you need to take:

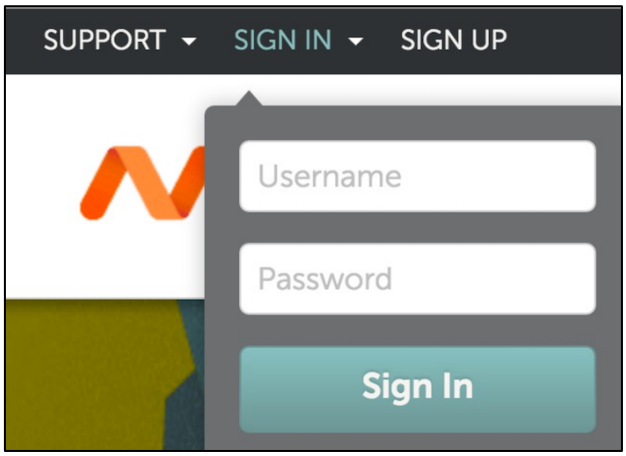
1. Change the DNS address at your domain registrar (in our case, NameCheap)
2. Add your domain using cPanel at the web host (in our case, lujweb.com on BigScoots)

1. Change Your Domain's DNS

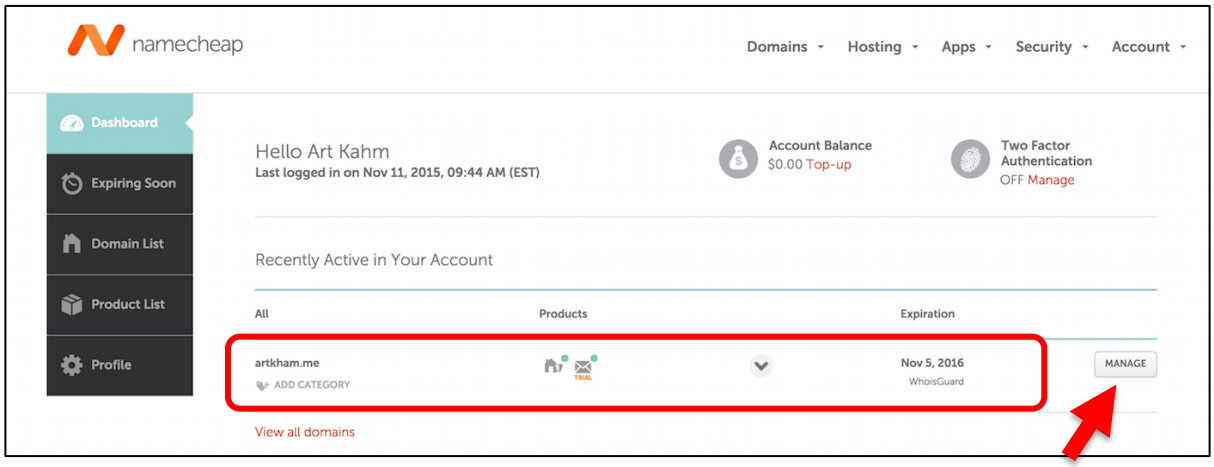
Begin by logging in to your account at NameCheap. For this, you must have the **username** and **password** you used for the account when you created your domain name.

Go to **http://namecheap.com**

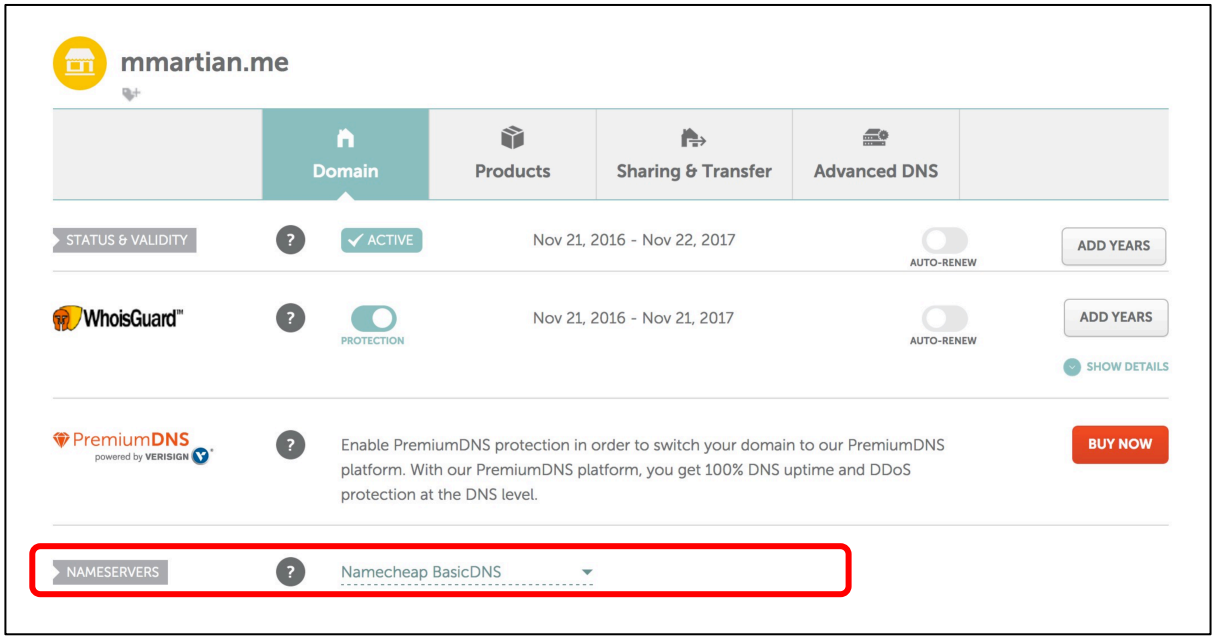
In the top-left corner, there is a "Sign in" menu item. Enter your username and password, and then click the "Sign In" button.



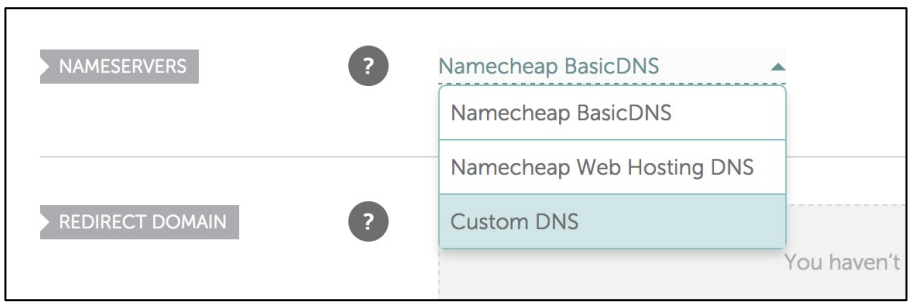
Your account page should appear, with your domain name listed.



Click on the "Manage" button. You will see a new page:



In the list, it will say "Nameservers." Normally, this is set to "Namecheap BasicDNS." You need to change it to "Custom DNS."



Use the nameservers:

NS1.BIGSCOOTS.COM

NS2.BIGSCOOTS.COM

Once you change them, you have to save the changes: click the green check mark. 

That's it! You're finished!

2. Add Your Domain to the lujweb.com Web Host Account

Next, you have to go to the cPanel at lujweb.com Go to:

lujweb.com/cpanel

I will send you an email with a PDF giving you the instructions on how to do this part. Since it contains sensitive information such as the site password, I will not put that in the textbook.

Appendix 1: HTML Cheat Sheet

TAG	HTML5 ATTRIBUTES	NOTES
<!DOCTYPE html>		Comes at beginning of every web page, with "html" (HTML5 rules).
*	title="[tooltip text]" style="[css decl.]"	Universal attributes; the asterisk represents all tags.
<html>	lang="en"	Begins normal HTML code. Should carry the language attribute to set editing/proofing language.
<head>		Contains document info, including title, meta, style, link, and scripts.
<title>		Contains the title for the web page, seen in the browser title bar or tab
<style>		Contains embedded CSS. Within the style tag, CSS rules apply, including different comment tags /* */
<meta>	name="author" name="description" content="value" charset="UTF-8"	Contains information about the document. The name attribute sets the type of information and is used with the content attribute.
<link>	rel="stylesheet" href="*filepath*"	Is used to link to an external stylesheet
<body>		Designates the area of the code where web page content is located. The body & all tags inside can use the style attribute to set CSS.
<header>		Structural tag which holds the title and subtitle of the page
<nav>		Structural tag which holds the main menu for the site
<main>		Structural tag which holds the primary content of the page
<footer>		Structural tag which holds publishing and site information
<article>		Structural tag which holds primary content specific to one page
<section>		Structural tag which holds one of many parts of segmented content for one page
<aside>		Structural tag which holds related but not primary content for a page or site
		A blank block tag
		A blank inline tag
<h1> ~ <h6>		Headers, used for titles and subtitles. Block, semantic.
<p>		Paragraph tag. Basic block element.
 		Line break. A helpful variation is: <br style="clear: both;">
<blockquote>		Creates an area with 0.5 inch indents on left and right. Can repeat. Block, semantic (used for quotes).
		Defines a bullet (unordered) list. Block.
	type="A" (not universal)	Defines an ordered list. Block.

		List item; used in both ul and ol tags.
<hr>		Creates a horizontal line. Block
<img>	src="*filepath*" alt="Alternate Text" width="400" height="300"	Places an image on the page. This is one of the few tags with many attributes which are still allowed instead of CSS.
<figure>		Block container for an image.
<figcaption>		Block container for image caption; used within <figure> tag. Semantic.
<q>		For inline quotes; adds quote marks
<b>		Makes text bold.
		Makes text bold (semantic).
<i>		Makes text italic.
		Makes text italic (semantic).
<sub>		Creates superscript text (x^2).
<sup>		Creates subscript text (H_2O).
<table>		Creates a table
<tr>		Creates a table row
<td>		Creates a table cell / column
<thead> <tbody> <tfoot>		Creates sections of a table so each can be formatted separately
<caption>		Defines a table caption
<form>	action="*global-filepath*" method="post" autocomplete="off"	Creates a form. The "action" sets the page or script that the form data will be set to; "method" sets the way the data will be sent
<input>	type="text" name="yourchoice" id="yourchoice" value="yourtext" placeholder="yourtext" autocomplete="off" maxlength="40" autofocus required	Creates one of various input fields in a form where the visitor can input data or choices. The **name** attribute will become the "holder" or variable name which is related to the data; the **id** can be used for styling or for certain purposes in scripts. **Value** will insert editable text into the form field, while **placeholder** will insert greyed text which cannot be edited and will vanish when text is typed.
<textarea>	size="40"	Creates a multi-line text input field. Any text, including whitespace, within the start and end tags, will be placed into the field.
<select> <option>		Create a drop-down menu; select creates the menu, each item in the menu is set by an option.
<fieldset>		Creates a bordered rectangle around parts or the whole of a form.
<!-- -->		Adds comment to code; will not be displayed in browser. Can be used to temporarily "disable" or hide existing tags.
<iframe>	scr="*filepath*"	Creates a window in which another HTML file can be displayed.

Appendix 2: CSS Cheat Sheet

FONT & TEXT PROPERTIES

Property	Values	NOTES
font	italic bold 18pt/30px 'Georgia', serif;	Allows for multiple values for font properties in one line. The example value gives style, weight, size, line-height, and font-family.

color	red;
	rgb(255,0,0);
	rgba(255,0,0,0.8);
		Sets the font color. Can use any of the methods shown at left.

font-size	14pt;	Sets font size using any measure (pt, px, cm, mm, in, %, em, etc.). Default font size is 12pt.
font-family	'Arial Narrow', 'Franklin Condensed', 'Open Sans Condensed', sans-serif;	Default font is Times New Roman. This sets fonts in order of availability. It should be ended with a font category matching the listed families.
font-style	italic; oblique;	Sets style, primarily italic.
font-weight	bold; none; 700;	Sets font weight, usually bold or none. If number weights (100-900) are available, they can be used.
font-variant	small-caps;	Creates small caps style.
text-align	center; left; right; justify;	Aligns paragraph text within a block area.
text-decoration	underline; none;	Allows you to add or subtract an underline. Most commonly used to remove underlines from links.
text-indent	0.5in;	Sets left indent using any measure (pt, px, cm, mm, in, %, em, etc.).
text-transform	capitalize; uppercase; lowercase;	
text-shadow	2px 2px 3px rgba(0,0,0,0.5);	Creates a drop shadow for text. The values are, in order, [move to right], [move down], [blur], and color.
line-height	2em;	Sets line spacing using any measure (pt, px, cm, mm, in, %, em, etc.). For example, 2em is double spacing.
letter-spacing	5px; 2mm;	Creates extra spacing between letters in text. Any measure will work.
word-spacing	10px; 1cm;	Creates extra spacing between words in text. Any measure will work.

BOX MODEL BLOCK PROPERTIES

Property	Values	NOTES
margin margin-top margin-right margin-bottom margin-left	20px; 20px 30px; 20px 30px 10px; 20px 10px 15px 20px;	Sets the margin for the outside of a block element. Margin will be ADDED to the element size.
border	1px solid black;	Allows for all border qualities to be set in one line (width, style, color).
border-width border-top-width border-right-width border-bottom-width border-left-width	2px; 2px 3px; 2px 3px 1px; 2px 1px 1px 2px;	Sets border thickness. For no border, set to 0px.

`border-style`
`border-top-style`
`border-right-style`
`border-bottom-style`
`border-left-style`
	`solid;`
	`dashed;`
	`dotted;`
	`double;`
	`groove;`
	`inset;`
	`outset;`
	`ridge;`
		Sets the border style. Some styles only appear when the border width is set to 3px or greater.

`border-color`
`border-top-color`
`border-right-color`
`border-bottom-color`
`border-left-color`
	`red;`
	`red blue green brown;`
	`rgb(255,0,0);`
	`rgba(255,0,0,0.8);`
		Sets the color of the borders.

`outline`	`2px solid red;`	Creates extra border.
`outline-offset`	`4px;`	Sets distance from border to outline.

`border-radius`	`20px;`
	`20px 30px;`
	`20px 30px 10px;`
	`20px 10px 15px 20px;`
		Sets the rounding of corners. Instead of top-right-left-bottom, the values are top-left, top-right, bottom-right, and bottom-left.

`padding`
`padding-top`
`padding-right`
`padding-bottom`
`padding-left`
	`20px;`
	`20px 30px;`
	`20px 30px 10px;`
	`20px 10px 15px 20px;`
		Sets the distance between the border and the inside content of a block element. Usually, the background will show beneath padding.

`background`	`pink url(bg01.jpg) no-repeat;`	Allows for multiple background styles. Order is: background-color, -image, -repeat, -position.

`background-color`	`red;`
	`rgb(255,0,0);`
	`rgba(255,0,0,0.8);`
		Sets the background color for the body or a block element. Can be used for inline elements, but not recommended.

`background-image`	`url(bg01.jpg);`	Sets image as background for the body or a block element.
`background-attachment`	`fixed;`	Background will not scroll with page

`background-size`	`300px;`
	`300px 200px;`
	`50%;`
	`cover;`
	`contain;`
		Sets the size of the background image. The first number value is the width, the second is height. If there is only one value, height is set to auto. **Cover** will auto-resize the image to always cover 100% of the background; **contain** resizes to always show the full image, even if there will be a blank margin.

`background-repeat`	`no-repeat;`
	`repeat-x;`
	`repeat-y`
		Sets how a background repeats.

`background-position`	`50px 20px;`
	`top center;`
	`bottom right;`
		Sets the starting position for a background. If numbers are used, the first sets the distance from left, the second the position from top. If words are used, it sets the alignment of the beginning image. Default is top left.

`background-origin`
`background-clip`
	`border-box;`
	`padding-box;`
	`content-box;`
		Determines where and how much of a background will appear. **Border-box** makes backgrounds extend to appear under the border; **padding-box** has a background fill the padding but not extend under the border; **content-box** will make the background cover only the content, and not the padding or border.

`box-shadow`	`5px 5px 10px rgba(0,0,0,0.5);`	Creates a shadow for a block element. Values are [distance right], [distance down], [blur], and color.

CSS Cheat Sheet

OBJECT PROPERTIES

Property	Values	NOTES
width	400px;	Sets the width of an object using any measure (px, cm, mm, in, %, etc.).
height	300px;	Sets the height of an object using any measure.
min-width	600px;	Sets the minimum width for a block element that may become larger.
min-height	600px;	Sets the minimum height for a block element that may become larger.
max-width	600px;	Sets the maximum width and stops the object from becoming larger.
max-height	600px;	Sets the maximum height and stops the object from becoming larger.
overflow	visible; hidden; scroll;	If a box's content is bigger than the box, this sets how the extra content will appear. **Visible** means the content will spill outside the box's borders. **Hidden** means that the extra content will disappear. **Scroll** will allow the box to become like a scrollable window.
opacity	0.5;	Sets the transparency of an object and everything inside it (text, image, background, etc.). 1 means normally visible; 0 means invisible. 0.5 will make the object half-transparent.
position	static; absolute; fixed; relative;	Sets the position of an object. **Static** is default. **Relative** will move the image relative to its normal position. **Absolute** will set the image relative to the sides of the document (top, right, bottom, left). **Fixed** will set the position of an object relative to the window, and the rest of the page will scroll past it.
top	40px;	Sets the position of an object relative to the top of the page or containing block element (if absolute) or to its normal position (relative).
right	20px;	Sets the position of an object relative to the right.
bottom	30px;	Sets the position of an object relative to the bottom.
left	25px;	Sets the position of an object relative to the left.
z-index	20;	Sets the layer for the object if objects overlap. A higher number places the object "on top" of others. Any number value can be used.
display	block; inline; none;	Sets the object as a block or inline element, or makes it disappear (none). Used to hide objects until an action is taken to reveal it.
float	left; right;	Sets a block element on the left or right side of a container, all else wrapping around it.
clear	left; right; both;	Finds the bottom of floating objects to the left side, right side, or both sides, and begins below that point.

ANIMATION PROPERTIES		
Property	**Values**	**NOTES**

`transform`	`translate(10px 20px);`
	`scale(2,3);`
	`rotate(10deg);`
	`scale(2,3) rotate(20deg);`
		changes an object's position in a 2-D or 3-D manner. **Translate** simply moves the object [to the right] and [down] by the number of pixels; **scale** changes the size [horiontally] and [vertically] by x times the size; and **rotate** changes the angle of the object clockwise by degrees. They can be used together.

`transition`	`all 2s ease-in 1s;`	This is an all-in-one property for several animation features.

`transition-property`	`width;`
	`display;`
	`all;`
		Specifies which property or properties will be animated.

`transition-duration`	`2s;`	Specifies how long the animation will take, in seconds.

`transition-timing-function`	`cubic-bezier(1,0,0,1);`
	`ease;`
	`ease-in;`
	`ease-in-out;`
	`ease-out;`
	`linear;`
		Specifies how the animation starts and stops: how fast it starts, how fast it accelerates, how fast ir slows down, and how fast it stops. The cubic-bezier numbers (0-1) set the speeds in a complex pattern.

`transition-delay`	`2s;`	Specifies how long before the animation begins.

OTHERS		
Property	**Values**	**NOTES**
`list-style`		Allows the type, position, and/or image to be listed in one line.

`list-style-type`	`none;`
	`circle;`
	`disc;`
	`square;`
	`lower/upper-latin;`
	`lower/upper-roman;`
	`lower/upper-alpha;`
	`hiragana(-iroha);`
	`katakana(-iroha);`
	`cjk-ideographic;`
	`decimal(-leading-zero);`
		Specifies the type of ordered list. Use this **instead of** the type attribute!

`list-style-image`	`url(filepath);`	Sets an image as the bullet in an unordered list.
`list-style-position`	`inside;` `outside;`	**Outside** means the number or bullet in a list stands out to the left of the list; **inside** means it does not.
`clip`	`rect(0px,400px,300px, 50px);`	Crops an image. The values are: [cut from top], [cut from right starting x pixels from the left side], [cut from bottom starting x pixels from the top side], [cut from left].

`cursor`	`crosshair;`
	`help;`
	`move;`
	`pointer;`
	`text;`
		Sets the cursor type when hovering over the affected area. Additional cursors include url and wait.

`visibility`	`hidden;` `visible;`	Makes something invisible, but it still takes the same space.