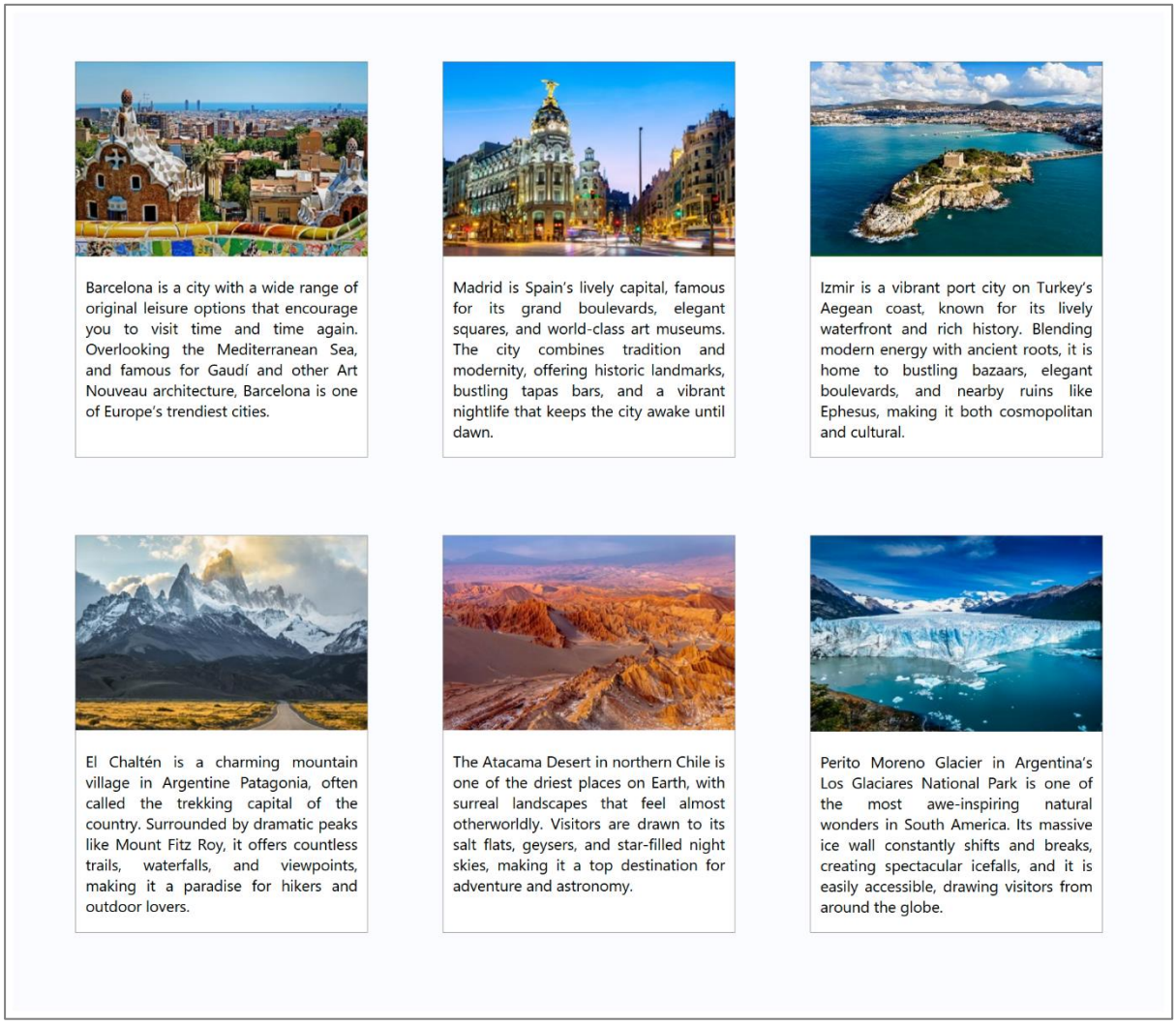
Author: Brian Webberley
Last updated: 2025.09.29

Interactive Web Design

Lab 3, Part 1: The CSS Grid

This lab is mostly a revision of creating CSS Grid, covered in Exploring Web Design. In Lab 3, Part 2, we will make our grid responsive so that it will work on smaller screens of devices like mobiles, and tablet computers.

Our final webpage for lab 3, Part 1 should look something like the below.



Recap: What is the CSS Grid Layout Module?

The CSS Grid Layout Module (more commonly referred to as *CSS Grid* or *Grid Layout*) is a grid-based layout system with rows and columns. It allows you to easily create complex web layouts. The CSS Grid makes it easier to design a responsive layout structure, without using

floats or positioning (which are more traditional and usually more difficult ways to create layouts). The CSS Grid properties are supported in all modern browsers.

A grid of any kind also makes it easier to quickly reference a location. Just like in Excel, if we say cell H22, we know its location is column H, row 22.

We can use a grid to determine the position of any HTML elements items such as text boxes, images, videos. Note that unlike Excel, a browser won't use grid lines. The CSS Grid is, by default, invisible.

Recap: When to Use a CSS Grid?

Before we know when to use a CSS Grid, we need to first understand how HTML elements are displayed by default on a webpage.

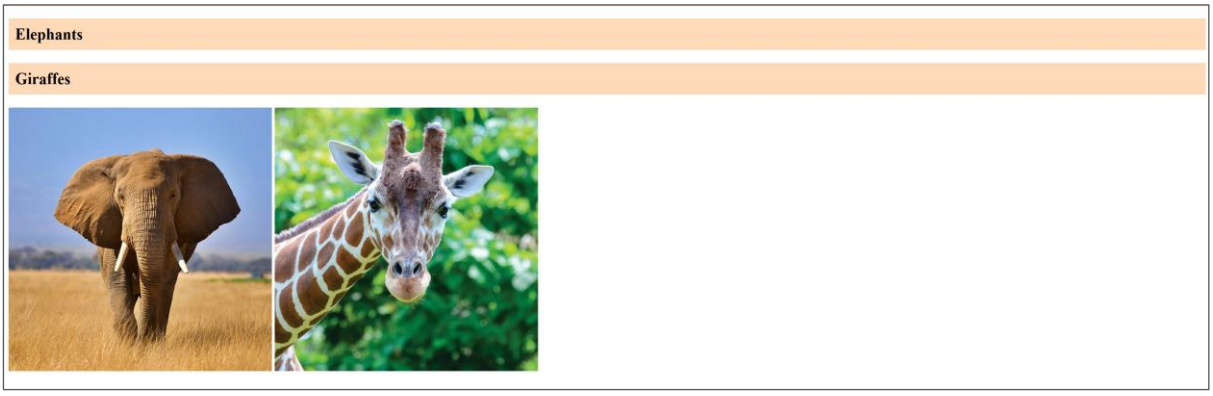
Inline vs Block Elements

HTML elements have default behaviour that controls their positioning. Look at the behaviour of the paragraph elements versus the image elements in this example:

```
<p>Paragraph 1</p>
<p>Paragraph 2</p>

<img src="images/elephant.jpg" alt="An image of an African elephant in the wild">
<img src="images/giraffe_close_up.webp" alt="A close up image of a giraffe">
```

In the browser:



The paragraph colours were added via CSS. Notice how each paragraph expands to take up the full width of the page, on its own line. However, the images all appear side by side, in the same line.

Every HTML element has a default block-level or inline behaviour. Paragraphs are block-level elements, which means that they 'block off' a whole line for themselves, and images are inline elements, which means they will automatically be placed next to one another on the same line.

Block-level elements	Inline elements
that you've seen so far include: ▪ Headings ▪ Paragraphs <p> ▪ Lists and list items ▪ Structural elements (header, nav, section, article, aside, figure, footer)	that you've seen so far include: ▪ Images <img> ▪ Links <a>

As you code more and more HTML elements, and notice their positioning in the browser, you'll memorise which HTML elements are block-level or inline.

Let's create some examples to demonstrate where we can use the CSS Grid to change the default inline and block element behaviours.

Task 1: File and folder set up

Step 1.1: Create a new folder called ***lab_3*** *(or lab3 if you perfer – just don't have any spaces)*

Step 1.2: Inside ***lab_3*** root folder, create a folder called ***images***, and the files: ***image_gallery.html*** and ***image_gallery.css***

Step 1.3: Copy all the HTML from ***lab_2.html*** into ***image_gallery.html***

Step 1.4: In the head section of ***image_gallery.html***, change the CSS file link to ***image_gallery.css***

Step 1.5: Copy all the CSS from ***lab_2.css*** into ***image_gallery.css***

Step 1.6: Copy the images folder from your lab_2 folder into your lab_3 folder.

Step 1.7: Open *image_gallery.html* in your browser and check that it looks the same as lab 2. If not, check back through the previous steps.

Why didn't we just continue with lab 2 instead of making a copy of it you may ask. We made a copy of lab 2 so that if things go wrong with this lab, we can always go back to the working lab 2 and compare the two labs to see what extra code was added that might have messed things up.

Task 2: Creating the Grid

Hopefully from lab 2, you should have gallery cards written with code something like the following, which you've copied into ***image_gallery.html***.

```
<article class="gallery_card_container">

    <div class="image_thumbnail">
        <img src="barcelona_thumbnail.jpg">
    </div>

    <div class="gallery_card_description">

        <p><b>Barcelona</b> is a city with a wide range of original leisure options that encourage you to
        visit time and time again. Overlooking the Mediterranean Sea, and famous for Gaudí and other Art
        Nouveau architecture, Barcelona is one of Europe's trendiest cities.
        </p>

    </div>

</article>
```

This code is just a sample of one gallery card.

Step 2.1: Adding the Grid Container

Encapsulate all the articles into one <section> tag, and give that <section> tag a class of grid-container, i.e.:

```
    <section class="grid-container">
```

Then add this to your CSS file:

```
.grid-container {
    display: grid;
}
```

What does this code do?

`display: grid;` - this turns `<section class="grid-container">` into a grid. (See lab 7 for details on what a container is)

This means all direct child elements of the container will become **grid items**. Grid items are boxes (squares or rectangles) on the grid. So, in our example, each <article> will become a grid item of the <section> container.

```
<section class="grid-container"> Grid container
    <article> Grid item
        ...
    </article>
    <article> Grid item
        ...
    </article>
</section>
```

That's all the code needed to create a grid. However, if we view the page in the browser, nothing will have changed. We need to tell the browser what type of grid we want.

In this example, we will create a grid of two columns.

Step 2.2: Specifying Columns

Update your CSS to the below:

```
.grid-container {
    display: grid;
    grid-template-columns: auto auto auto;
    gap: 50px;
    justify-content: center;
    background-color: #fafcff;
}
```

Save all files and view in the browser. Your webpage should now display two columns like the following image:

Each grid item has gone into a **column** but what did all that code do exactly?

`grid-template-columns` tells the browser how many columns you would like to display across the page (horizontally, from left to right) and what width they should be.

Here we use `auto auto auto`, which will tell the browser calculate the free space three ways. More examples:

`grid-template-columns: auto;` - this sets the amount of columns. In this example, it's one column.
`grid-template-columns: auto auto;` = 2 columns
`grid-template-columns: auto auto auto auto;` = 4 columns, and so on.

You can also use create three columns using percentages. For three columns you could have chosen 30% 30% 30%, on so on. Why not 33% 33% 33%? Using only 90% helps to leave some space for gaps, padding and / or margins.

Percentages give greater control, which can be helpful if things don't appear as they should using *auto*. However, using auto allows for the width to be more flexible which helps makes the page more responsive, as we will see later.

```
gap: 50px;
```
simply creates a 50 pixels gap (also called a gutter) between the two columns

```
justify-content: center;
```
Moves the columns into the centre of the webpage.
See more here: https://www.w3schools.com/cssref/css_pr_justify-items.php

```
background-color: #fafcff;
```

This is just to help us see where exactly the grid container is. It separates the background of the grid from the background of the page. It helps you to distinguish between the two

should you wish to style one or the other, you can see where your changes are having an effect. It's just temporary.

Step 2.3: Style the Grid Items

Just like with the grid container, it's often helpful when coding to give grid items a background colour so that you can see where it is. When working on a project you can always remove the styling later if it's not needed.

Add a background colour to the CSS *gallery_card_container* class to make the text stand out from the background.

```
.gallery_card_container
{
    border: 1px solid rgb(128, 128, 128);
    width: 300px;
    margin-bottom: 30px;
    background-color: white;
}
```

Your webpage should now look like the below in your browser:

Applying different colours to the grid container and the grid container's items different colours makes it easier to see where the container is and where the grid items are by their background colours.