Lab Exercise 4: CSS Grid Layout

The purpose of this assignment is to advance our CSS layout abilities, as well as try out some other CSS design techniques and tools. In particular, we'll use background images, grid layouts, embedded icons, and special fonts.

Additional resources are at the end of this document and **screenshots** are in the starter files.

Task 1: Setup

This time, I've provided a starter HTML file for you. Import this into a new project and open it in VS Code.

Task 2: HTML

Most of the HTML code has been written for you. However, there are a few things you'll have to do:

- Semantic tags**:**
- Some semantic tags are missing. Add necessary semantic tags.

- In the <header> section:
- As you can see from the screenshot, the text **"Lab Exercise 4:"** in the **<h1>** is a different color from the text **"Space Probes".** You'll need to somehow mark up this text so that it can be selected and recolored in the CSS code. Figure out how to do this. (hint: you'll need to do some research to find a generic container tag that behaves as an inline element. It may have been used in the instruction this week!)

- The tagline **"Probes that explored planets, asteroids, comets, and moons in the solar system and beyond"** is not marked up with any tag. Find an appropriate tag to do this, using this resource: https://www.w3.org/TR/2014/REC-html5-20141028/common-idioms.html
  Hint: this is a bit of a trick question.
  Also give this tag a **class** with the value "**tagline"**.

- In the <main> section:
- As you can see, there are **three sections** within the **<main>** element. One section has **class="articles".** Within this section, there are several article previews separated by their headings. (Each article preview contains a short section of text with a link to the full article.)

  In HTML, while we use the <section> tag to contain a subsection of the parent document, we use the <article> tag to contain content that could potentially stand on its

own outside the context of the parent document. It's also conventional to use the <article> tag to mark up short previews of articles. (Such as the article teasers we see on the front page of a magazine or news website.)

Your job is to use the **<article>** tag to complete the mark up in the articles section. Each article preview should be contained inside an <article> element.

Hint: In this exercise, each article starts with an image and ends with a paragraph that contains text “Read More”.

- The other two sections within the <main> element are **"Article Archive"** and **"Sci-fi Book Reviews"**. These two sections represent content that is peripherally related to the site's main content, but not the primary focus. You can also see that, in the screenshots, this content has been placed in its own sidebar column on the right side of the page. To mark up content like this, we can use the **<aside>** element. Go ahead and implement this element.

- In the <footer>:
- In the **"Contact"** section, the links for the email and telephone number don’t currently have **href**s. Figure out how to add **href** attributes for **email** and **telephone** links. When clicked/tapped, these links should launch the default email client or invoke the mobile device's telephone functionality, respectively.

Task 3: Icons

In the footer, social media icons are used in the screenshots. Let’s add these icons to your page. There are a few services that provide icons for use on the web; we're going to use one called **Font Awesome**. Sign up for a free account here: https://fontawesome.com/start

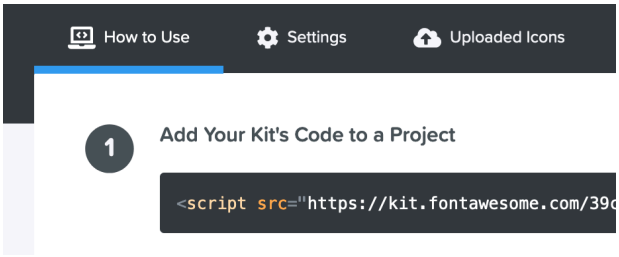
(As always, you may use fake information in the interest of maintaining your privacy.)

Once you have your free account, you'll need to set up a so-called **"kit"** that you'll use to apply icons to your site. Create a kit here (click the blue **New Kit** button): https://fontawesome.com/kits

You will then be directed to a page where you will see a "kit code". It will look something like this (but with a unique alphanumeric component):

<script src="https://kit.fontawesome.com/39xxx2681x.js" crossorigin="anonymous"></script> (Don’t copy this code! This is an example.)

If you don't see your kit code, try clicking on the kit and then navigating to the "How to Use" tab:



Copy your kit code and paste it into the <head> of your document. Now you can start using icons!
To find the icons you need, visit this page: https://fontawesome.com/icons and use the search functionality. Icons are placed in your layout by copying the appropriate **** tag and pasting it into your HTML. For example, the Pinterest icon is embedded using this tag:

****

If you are having a trouble loading some of the icons, try using “fab fa-facebook-square”, “fab fa-instagram-square”. This is due to some icons being Font-Awesome 5.

note:
The tag is actually meant to do something else and doesn't officially have anything to do with icons. Font Awesome has appropriated this tag for use with their system. *Why do you think they did this? What are the pros and cons of using the tag? What was it originally meant for? What should you do if your CSS code is directing special styles to elements?*

In your site, the icons should be implemented as hyperlinks so that one can click on them to be directed to the appropriate social media page.

The use of icons in our design presents special challenges for accessibility. We need to make sure that users with visual disabilities are able to perceive the purpose of the hyperlink when using a screen reader. Therefore, we need to make sure that even though users navigating visually will perceive the purpose by seeing the icon, the HTML also contains some text description of the link that can be converted to audio by a screen reader program. Figure out how to do that here: https://fontawesome.com/how-to-use/on-the-web/other-topics/accessibility#web-fonts-semantic

They provide a few different choices for making your icons accessible; my opinion is that the best option is the one that involves using the **aria-label** and **aria-hidden** attributes.

Optional task: figure out what ARIA stands for, and what the ARIA specification is meant to do.

Task 4: Google Fonts:

So far, we've been using boring fonts that are guaranteed to work in every browser. This has been necessary because there's no way to know which fonts will be available on a user's system.

However, there's another way: **Google Fonts** (and other similar services) which allow us to embed fonts that can be automatically downloaded onto the user's system when they view the page.

Spend a bit of time exploring the fonts available at https://fonts.google.com/. Be sure to experiment with the search and filter options. They update the UI frequently. Make your own decision if the following instructions are not applicable to the most recent UI.

Figure out how to select font families and add them to your shopping cart of fonts. (They actually call it **Selected Families,** not shopping cart.) Figure out how to view your selected fonts and add and remove different styles.

- Add the **Exo 2** font family
- Add the **Open Sans** font family

From the **Selected Family** menu, figure out how to get the line of CSS code that you will paste in your CSS stylesheet to embed these fonts in your site.

Set the default font family for the whole page to **Open Sans** with sans-serif as the fallback. Have the font family for all headings set to **Exo 2** with **sans-serif** as the fallback**.** Have the tagline also set to **Exo 2**, but using the **italic** style. (figure out how to do this.)

Task 5: Basic Styles

Using CSS, implement the following:

- Set the width of all images to 100% so that they fill their containers.
- In the starter files, I've given you an image called **pexels-philippe-donn-1257860.jpg**. Using the **background-image** property, apply this image to the **<body>:** https://www.w3schools.com/cssref/pr_background-image.asp
- Since the image is too large for the page, we need to make it smaller using the **background-size** property. Figure out how to do this.
- Set the background color of the wrapper ** ** to **white**. Set an appropriate **padding**, **box shadow** and **maximum width** (**1000px**). Center the element in the window. The wrapper should not get larger than 1000px, but should shrink appropriately when the browser window is made narrower than 1000px.
- Regarding the **box-shadow**: we'd like to use a color for the shadow that is partially transparent, since we don't want the shadow to be too intense. Figure out how to do this using the **rgba()** CSS function: https://www.w3schools.com/cssref/tryit.asp?filename=trycss_func_rgba

- What does the "a" stand for in rgba()?
- Change the color of **"Lab Exercise 4:"** in the **<h1>** to **royalblue**.

Task 6: Grid Layout For Navigation Menu

By default, our HTML container elements (such as footer, main, nav, section, article, etc.) will stack on top of each other vertically. (As block elements do.) However, we're often going to need to create layouts that are a bit more complex than that. For example, we may need to create multi-column layouts, sidebars, grid layouts, etc.

To do this, we're going to use a new display property called **grid.** Let's start by using this property to align the elements of our navigation menu horizontally. While we previously did this by using **display:inline-block**, the **CSS grid** system gives us the advantage of creating a responsive system in which the nav elements resize themselves to fit the window, regardless of the window or device size.

Properties that affect the whole grid layout are always applied to the **parent** element in which the grid cells are contained as children. In the case of the navigation menu on this page, *figure out which element is the parent, and which elements are the children that will become the grid cells.*

In the CSS code, select the **parent** element and add the following property:

display: grid;

The parent is now a grid container. We now need to set up a column template for the grid. In the parent style declaration block, set the following CSS property:

grid-template-columns: 1fr 1fr 1fr 1fr;

The above code is setting up **four** columns (since our navigation menu has four items). The **fr** unit is used for the **flexible length** value. What we're doing here is dividing the free space inside the grid parent into four equal parts. We know they're equal because each column has the same value: **1fr**. If we wanted one column to be larger than the others, we could give it a larger value, like 2fr, for example.

In order to add some space between the grid columns, use the **grid-gap** property:

gap: 1rem;

Optional challenge: try reducing the number of columns in the grid. What happens? What happens if you add additional columns?

A few other things you should do:

- Remove the bullet points from the list;
- Figure out how to remove the empty space on the left side of the grid. *Where is this coming from?* Hint: the inspector tool will be helpful for this.

Task 7: Other Grid Layouts

Using CSS grid, do the following:

- Figure out how to split the <main> element into two columns with the left column being three times wider than the right column. The left column should contain the "articles" section (with "Features Probes") and the right column should contain the <aside> content (with "Article Archive" and "Sci-fi book Reviews").

  Make sure the <main> element has *only two direct children*, which will be arranged as the left and right column. If you have more than two children in the <main> element, you may need to revisit the code you wrote in **Task 2**.
- Figure out how to split the <footer> into three columns of equal width.
- Figure out how to split the "articles" section into a grid with two columns and multiple rows. This section should have, in the HTML mark up, five children: one heading and four articles. (You may need to revisit your solution for Task 2 if this is not currently the case.)
- To do this, you'll need to have the heading stretch across two columns. Figure out how to do this using the **grid-column** property: https://developer.mozilla.org/en-US/docs/Web/CSS/grid-column

  This property allows us to have an element stretch across multiple columns by specifying a start column and end column. (or a start column and the number of columns to stretch across.)
- Note that you don't have to do anything special to set up multiple rows; the grid cells will automatically flow into new rows once the first row is filled.

- You do not need to align items inside each article such as image, heading, and paragraphs to other articles at this point although the screenshot has them aligned.

Task 8: Remaining Styles

The hard part is done; now we just need to add a few additional styles:

- The two theme colors for this page are **tomato** and **royalblue**; apply these colors to the appropriate elements.
- Match the colors and approximate size of the headings to the screenshots.
- Add **borders**, **padding** and **margin** to match the screenshots.
- Remember that the **gap** property can be used to add space between grid cells.
- Set the background color of the right column to be a partially transparent version of **royalblue** using the **rgba()** function. To do this, get the RGB value of **royalblue** from here: https://www.rapidtables.com/web/color/html-color-codes.html
- Implement any other styles necessary to approximately match the screenshots.

Check Your Work

- Validate your HTML and CSS code. Don't forget to save the screenshots to hand in.
- Embedding the Google fonts in your HTML code may cause validation errors; if it happens, try embedding them using CSS instead.

Checklist

- HTML markup
- Accessible icons
- Google fonts applied
- Grids
- Misc. styles (colors, borders, box model, etc.)
-

Hand In

I believe you are now familiar with the process of showing the instructor your work before the end of the class. So, I will stop mentioning it in every lab exercise instruction!

Rename your working folder to **a4-firstname-lastname.** (with your own first and last name, of course!) Create a new folder called **validation** that contains your validation screenshots and add it to this folder. Make sure the work you're submitting does not contain any unnecessary files, such as the screenshots from the starter files. Create a Zip archive from the **a4-firstname-lastname** folder and hand it in to D2L. (Do not use some other archive format like Rar or 7z. If you're having trouble creating a Zip archive, please let me know.)

Resources:

- Span element:
- https://www.w3schools.com/tags/tag_span.asp
- Semantic tag flowchart:
- http://html5doctor.com/downloads/h5d-sectioning-flowchart.pdf
- Article element:
- https://developer.mozilla.org/en-US/docs/Web/HTML/Element/article
- Aside element:
- https://developer.mozilla.org/en-US/docs/Web/HTML/Element/aside
- Intermediate-level how-to for Google Fonts:
- https://developers.google.com/fonts/docs/getting_started
- Font-style property:

- https://www.w3schools.com/cssref/pr_font_font-style.asp
- Background images:
- https://www.w3schools.com/cssref/pr_background-image.asp
- CSS Grid basic tutorial
- https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout
- Grid-column property:
- https://developer.mozilla.org/en-US/docs/Web/CSS/grid-column
- Grid Garden: an adorable browser game for learning CSS Grid. Note that it is a bit more advanced than necessary for the assignment, but you may find it useful if you want to become really good at using CSS grid:
- https://cssgridgarden.com/