Flexbox Layout Activity: Understanding Flexbox Properties and Hands-On Practice

Objective:
This activity is designed to help learners solidify their understanding of flexbox properties, memorise key properties and their values, and then apply them in a practical implementation. The activity is divided into two parts: a conceptual part to reinforce understanding and a hands-on part for practising flexbox layout.

Additional Resources If Needed:

- HTML Display Property Explained: Block vs Inline vs Inline-Block with Examples
- CSS Flexbox Layout Explained: Easy Visual Examples and Flex Tutorial
- Flexbox Playground Codepen
- Flexbox Navbar Code Demo
- CSS Flexbox Layout Guide Written Lesson

Part 1: Conceptual Understanding and Memorisation

Task 1: Flexbox Property Definitions

Fill in the table below with definitions and explanations for each property related to flexbox. If you're unsure, refer to W3Schools for help.

Property	Definition/Explanation
`display: flex`	WIll use the flex display property so flex properties can be set
`justify-content: center`	Moves elements center along the x axis
`flex-direction: row-reverse`	Keeps all elements in a row but reversed
`flex-wrap: wrap`	Wraps elements to the next line when the screen is small enough for the elements to lose their space

`align-items: center`	Centers elements along the y axis
`flex-grow`	Will allow the element to take up the rest of the area with its elements or element
`flex-shrink`	Will shrink the element or elements depending on the number specified. Greater than zero is more aggressively shrunk than zero.
`align-content`	Aligns the content of the element with values that start it at the beginning, the end, stretched to take up remaining space, spaces evenly, or spaces the content between.

Task 2: Flexbox Property Matching

Match the flexbox properties with their correct function or description:

Flexbox Property	Function/Description
`justify-content: space-between` E	a) Defines how items are aligned along the main axis. Align-items:flex-end
`align-items: flex-end` A	b) Defines whether the flex container is a row or a column. Flex-direction
`flex-direction: column` B	c) Allows items to shrink or grow to fit the container. Flex-grow
`flex-wrap: nowrap` D	d) Items will not wrap, and overflow the container. Flex-wrap:nowrap
`flex-grow: 1` C	e) Space is distributed evenly between and around items in a flex container. Justify-content: Space-between

Task 3: Reflection Questions

Answer the following questions to reflect on your understanding:

1. **What is the primary advantage of using flexbox for layouts compared to traditional methods like floats?**
   - **Answer: It is automatically responsive so it guarantees users to get similar experiences.**
2. **What does the `flex-direction` property control in a flex container, and how does it affect item alignment?**
   - **Answer: Whether elements are stacked on the y axis or put on the same x axis as a row**
3. **What happens when you set `flex-wrap` to `wrap` in a flex container with many items?**
   - **Answer: When the screen runs out of space for the elements it will put the next element on the next row beneath the elements.**

Part 2: Hands-On Flexbox Implementation

Scenario:

You have been tasked to create a **responsive image gallery** for a client. The client wants the images to be aligned horizontally on larger screens and wrap to the next line on smaller screens. They also want the items to be centered on the page.

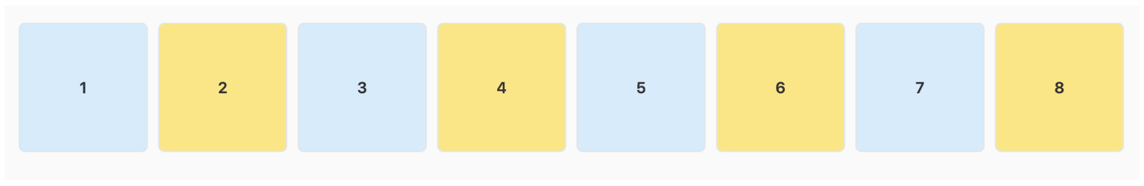
Instructions:

1. **Create a container:**
   - Create a `div` with a class name of `container`. Set its display property to `flex` to turn it into a flex container.
2. **Add multiple items:**
   - Inside the container, add multiple `div` elements with the class `box`. Each `box` will represent an image (for now, you can just use background colors to differentiate them).
3. **Apply flexbox properties:**
   - Use flexbox properties to achieve the following layout:
     - The boxes should be horizontally aligned on larger screens.
     - The boxes should wrap to the next line if the screen size is too small.

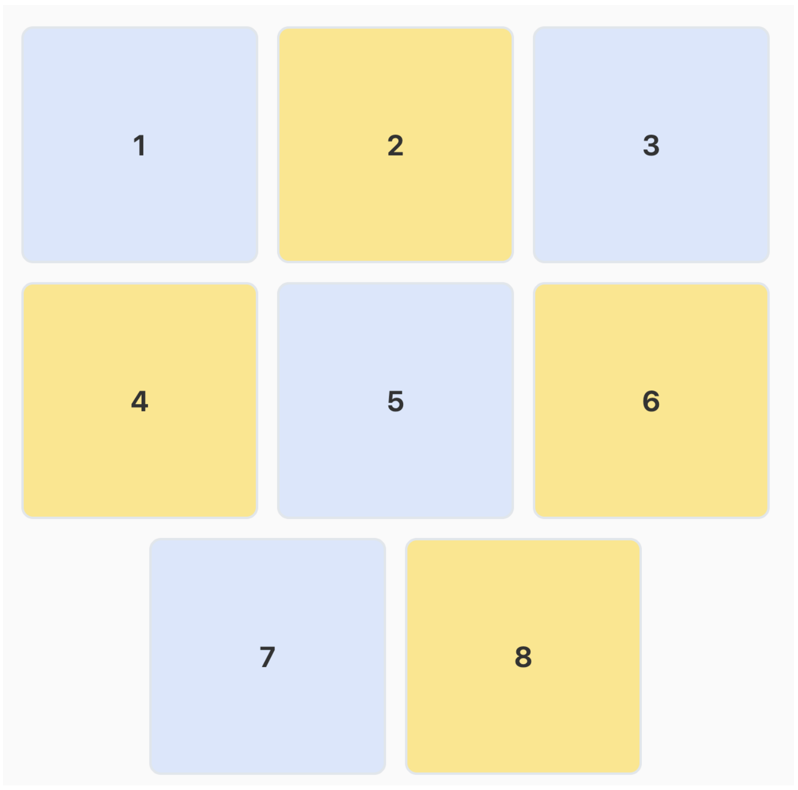
- The boxes should be centered within the container both horizontally and vertically.
- Add margins and padding for spacing.

4. **Customize the box properties:**
   - Set a width and height for each box, and add some styles like background color, padding, and a border.
5. **Test responsiveness:**
   - Resize the browser window to see the flexbox wrapping behavior in action.

Output in full screen view:



Output when browser window shrinks and there is no enough space to display the whole line of boxes on the same row



Hints: Example Structure (No Example Code):

- **Container:** A div with the `display: flex` property.
- **Boxes:** Several div elements representing images (use background colors for now).
- **Flex Properties:**
  - Use `justify-content`, `flex-wrap`, `flex-direction`, and `align-items` to achieve the desired layout.

Reflection:

1. **Check Your Work:**
   - Open the developer tools and inspect your container to see how the flexbox properties are applied. Use the **flexbox inspector** in the dev tools to visualize the alignment.
2. **Challenges:**

- Did you encounter any difficulties when items started wrapping? How did you resolve them?

3. **Final Layout:**
    - Ensure that the boxes align correctly when the screen is resized.

Completion:

Once you've completed the activity, review the layout, and experiment with different flexbox properties to see how they affect the layout. This activity will help solidify your understanding of flexbox before moving on to more complex layouts.