Notes - CSS - Flexbox - Common Use

- Dr Nick Hayward

A general guide to common CSS Flexbox use cases and examples.

Contents

Intro CSS Flexbox helps solve many issues that have continued to plague layout and positioning of HTML elements and components in both client-side and cross-platform apps.

For example,

- vertical and horizontal alignment
- defining a centred position for child elements relative to their parent
- equal spacing and proportions for child nodes regardless of available space
- equal heights and widths for varied content
- & lots more…

Initial benefits of Flexbox Flexbox was designed to help with *one dimensional* layout of elements, items rendered by a web browser.

We commonly choose Flexbox to allow us to control the layout of such items in one direction or another, control their individual dimensions, and control spacing between these items.

However, whilst this may be the described initial intention for Flexbox, we may also use this layout option for other common tasks often better suited for CSS Grid Layout, or simply as a fallback for Grids.

Common examples Flexbox usage, as noted, has benefits for *one dimensional* structure and layout.

For example, we might use Flexbox for a row of items in a banner area at the top of a HTML page.

We might also use this type of pattern to layout a row of items for a navigation menu, form fields, or information cards in a visualisation or content design.

Flexbox is also useful to help define a centred layout for an item, or group of items, relative to a single dimension within the overall design of a HTML page.

We may also use Flexbox patterns to help organise items such as images and asociated textual content. Such media content and groupings commonly require row or column based layouts, which fit well with Flexbox's properties.

Menu items and banners Banner areas may be added to the top of a HTML page, and the top of a content area to help define and delineate structure within a page.

Such structure for banners is commonly designed as rows of content, which may also include menus and navigation items.

Such rows of items may, of course, use a one dimensional layout with Flexbox.

For example, we might design an initial row of items for a navigation menu.

```
<!-- nav menu for site -->
<nav class="site-menu">
    <ul>
        <li><a href="">Home</a></li>
        <li><a href="">About this project</a></li>
        <li><a href="">News</a></li>
        <li><a href="">Search</a></li>
    </ul>
</nav>
```

We may then define a ruleset for the `ul` , which will be defined as the container for the flex items.

We remove the initial default margin and padding for the `ul` box model, and then add the flex properties.

```
nav ul {
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: space-around;
}
```

Initially, we may define the distribution of space in this container to the value `space-around` , which creates an even distribution of menu items, each `li a` elements, within the menu.

However, we may modify the use of space relative to the desired menu design and rendering. For example, we might prefer to use `space-between` , `space-evenly` , `flex-start` , `flex-end` or, perhaps, `center` .

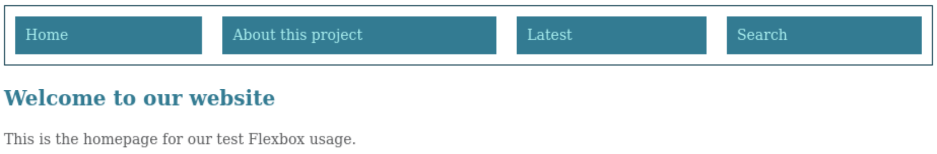
With the current example, we are defining how the available space may be used between the flex items. However, we may also consider organising the menu by defining how space is arranged within the flex items themselves.

For this organisation of space, we may use the `flex` property to enable each flex item to grow and shrink as necessary relative to its content requirements.

For example, we may define a ruleset as follows

```
nav ul li {
    list-style-type: none;
    flex: auto; /* shorthand for flex: 1 1 auto */
}
```

In this example ruleset, we remove the default `li` bullets, and then define each flex item to apportion their required space relative to content. Initially, this would be derived from `basis` set to auto. Each item would then be able to grow relative to content.



Welcome to our website

This is the homepage for our test Flexbox usage.

Figure 1: CSS Flexbox - example menu

Card layout Card layouts commonly include a combination of heading, content, and a footer to the design of an individual card.

As content is added to each card, it will render its height to fit the content, thereby pushing the footer down. However, as long as the defined container has sufficient space available, a card will stretch to the height of that container.

The content inside the card will, commonly, use a *block* layout, which means the footer may be rendered above the foot of the card design if there is insufficient content to fit the rendered height.

However, Flexbox offers a solution to this poorly placed footer design.

Initially, we may define a card as a Flex container, and set the property `flex-direction` with a `column` value. We may then set the content to the property `flex: 1` .

In effect, the flex item may grow and shrink, but it does so from a `basis` of `0` . This becomes the only item that may grow in the container, forcing the *footer* to the foot of the card.

For example, we may design a simple card

```
<div class="card-view">
    <header>card header...</header>
  <div class="card-content">
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</p>
  </div>
  <footer>card footer...</footer>
 </div>
```

and add each card to a `cards` grouping, which we may use to define

```
<div class="cards">
    ...
</div>
```

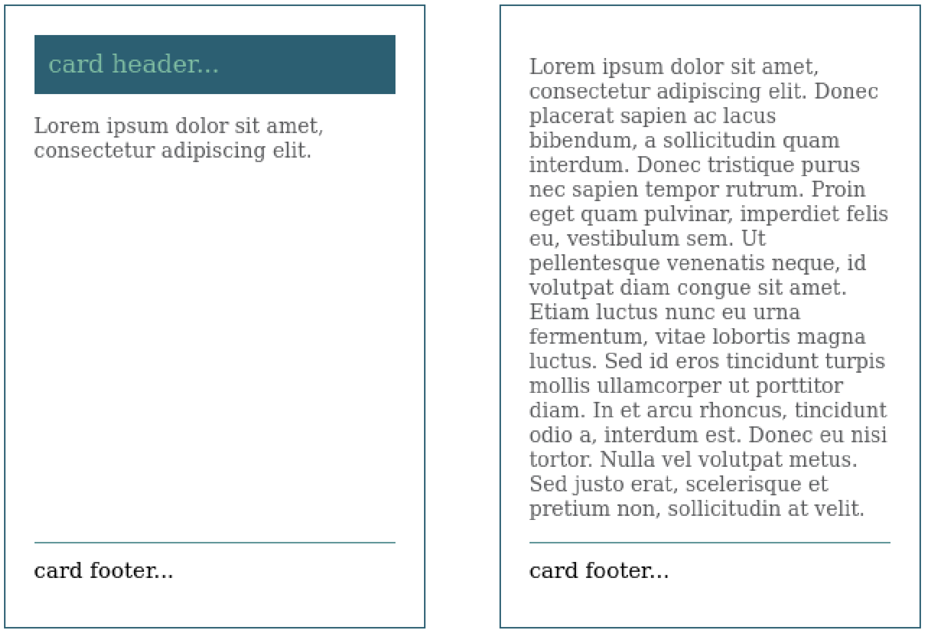
To implement the required Flex based view, including the footer at the foot of each card, we may define the following rulesets

```
.cards {
    display: flex;
    justify-content: space-around;
    flex-wrap: wrap;
}

.card-view {
  display: flex;
  flex-direction: column;
    flex: 0 0 250px;
}

.card-content {
  flex: 1;
}
```

We’ve included some extra properties to help style each card.


card header...

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

card footer...

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec placerat sapien ac lacus bibendum, a sollicitudin quam interdum. Donec tristique purus nec sapien tempor rutrum. Proin eget quam pulvinar, imperdiet felis eu, vestibulum sem. Ut pellentesque venenatis neque, id volutpat diam congue sit amet. Etiam luctus nunc eu urna fermentum, vitae lobortis magna luctus. Sed id eros tincidunt turpis mollis ullamcorper ut porttitor diam. In et arcu rhoncus, tincidunt odio a, interdum est. Donec eu nisi tortor. Nulla vel volutpat metus. Sed justo erat, scelerisque et pretium non, sollicitudin at velit.

card footer...

Figure 2: CSS Flexbox - cards

We may define a Flex container for the overall `cards` grouping, which organises the cards based on a default main axis. We may also add the option to allow the cards to wrap to multiple rows.

In the `card-view` ruleset, we define a flex container for each individual card, and set the content flow direction to ensure we get the desired stacked content rendering.

We also set the Flex *basis* for each card to 250px, and then prevent stretching or shrinking of the card’s design.

cards with media We may also use Flexbox to help organise text and media objects within a card. This includes, for example, effective rendering of portrait and landscape orientation images.

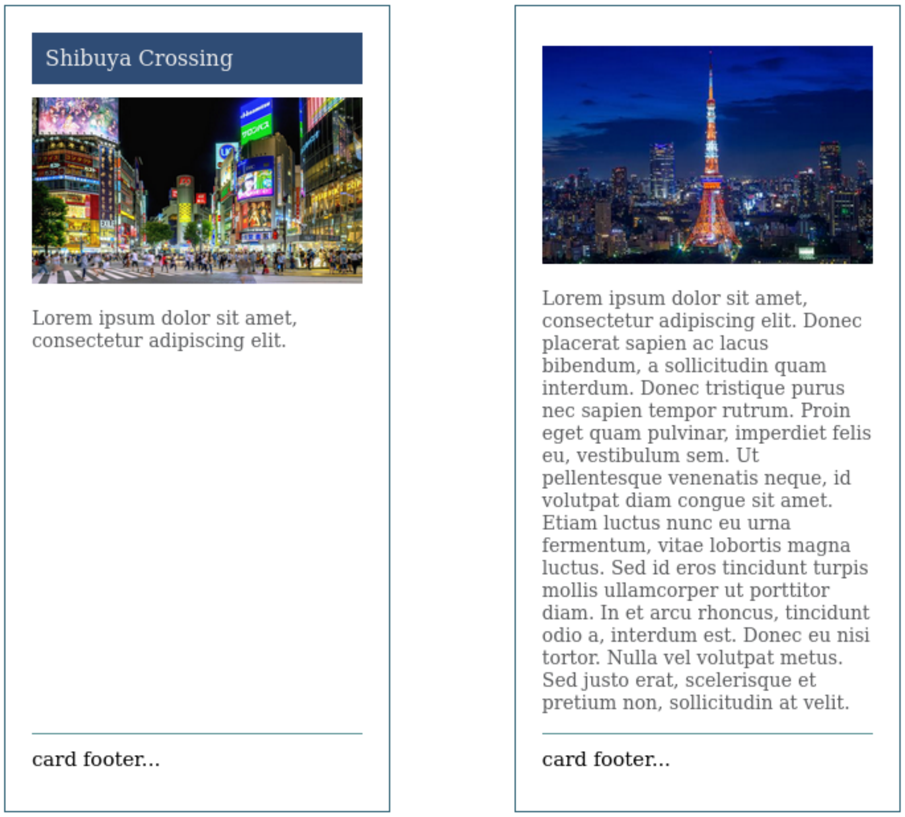
For example, we may update the HTML for a card

```
<div class="card-view">
    <header>card header...</header>
    <div class="card-media">
        <img src="./media/images/test-image.jpg" alt="Test Image...">
    </div>
  <div class="card-content">
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</p>
  </div>
  <footer>card footer...</footer>
 </div>
```

If we maintain the same layout pattern for a column-based card, we may add a ruleset for the image container, and then style the image.

For example, we may simply add the following ruleset for our current layout and design

```css
/* media objects */
.card-media img {
    max-width: 250px;
    padding: 10px 0 0 0;
}
```



Shibuya Crossing

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

card footer...

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec placerat sapien ac lacus bibendum, a sollicitudin quam interdum. Donec tristique purus nec sapien tempor rutrum. Proin eget quam pulvinar, imperdiet felis eu, vestibulum sem. Ut pellentesque venenatis neque, id volutpat diam congue sit amet. Etiam luctus nunc eu urna fermentum, vitae lobortis magna luctus. Sed id eros tincidunt turpis mollis ullamcorper ut porttitor diam. In et arcu rhoncus, tincidunt odio a, interdum est. Donec eu nisi tortor. Nulla vel volutpat metus. Sed justo erat, scelerisque et pretium non, sollicitudin at velit.

card footer...

Figure 3: CSS Flexbox - cards with media

However, we need to consider images that are also portrait orientation. This will require an updated Flexbox layout, and accompanying rulesets for the content and media objects.

cards with media - add portrait support We may slightly modify our current *cards* example to support horizontal layout with portrait and text content.

To maintain the column rendering for header, content, and footer, we may group the portrait image with its textual content in a separate container. We may then define this group as its own flex container, which we may style as required to achieve the required parallel display of image and text.

We may update the HTML for this type of card,

```
<div class="card">
    <div class="card-media">
        <img src="./media/images/kinkaku-ji-kyoto.jpg" alt="Kinkaku-ji, Kyoto">
    </div>
  <div class="card-content">
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec placerat sapien ac lacus bibendum
  </div>
</div>
```

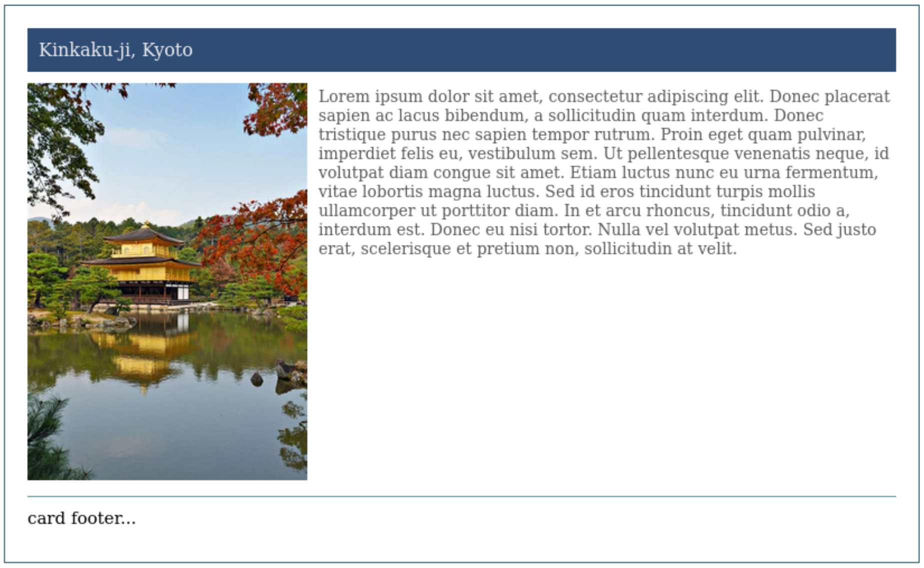
We may then style the `.card` class, creating a flex container for the card's content.

```
.card {
    display: flex;
    align-items: flex-start;
}
```

We'll also need to update the original `.card-view` ruleset to enable stretching for the card.

```
.card-view {
  display: flex;
  flex-direction: column;
    flex: 1 0 250px;
}
```

We've now updated the `flex` property to enable stretching, which allows the card to correctly render the image and text.



Kinkaku-ji, Kyoto

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec placerat sapien ac lacus bibendum, a sollicitudin quam interdum. Donec tristique purus nec sapien tempor rutrum. Proin eget quam pulvinar, imperdiet felis eu, vestibulum sem. Ut pellentesque venenatis neque, id volutpat diam congue sit amet. Etiam luctus nunc eu urna fermentum, vitae lobortis magna luctus. Sed id eros tincidunt turpis mollis ullamcorper ut porttitor diam. In et arcu rhoncus, tincidunt odio a, interdum est. Donec eu nisi tortor. Nulla vel volutpat metus. Sed justo erat, scelerisque et pretium non, sollicitudin at velit.

card footer...

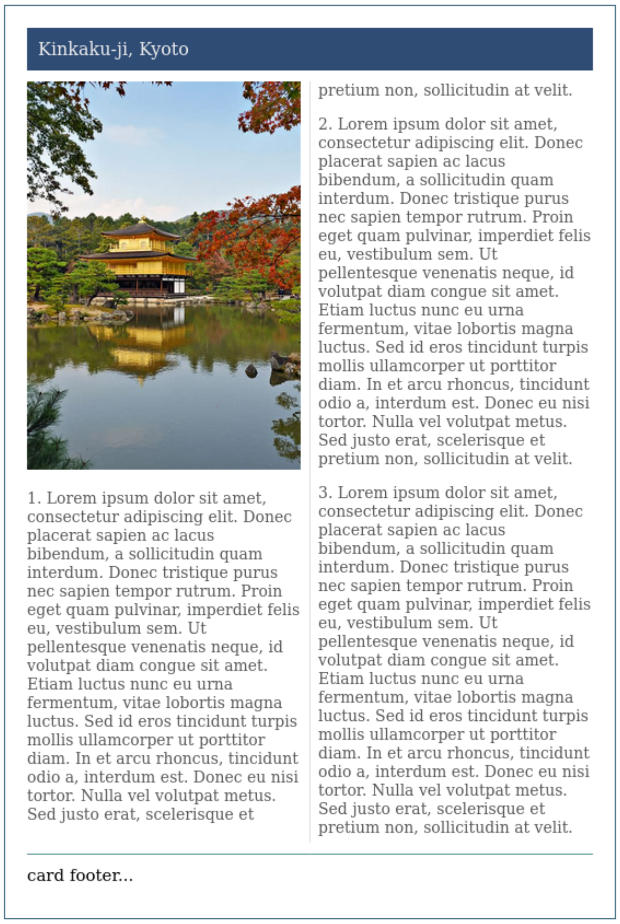
Figure 4: CSS Flexbox - cards with media and portrait image

cards with newspaper flow Whilst we may use Flexbox to align horizontal content in a row direction, as shown in the previous example, we still encounter an issue with wrapping.

For example, if the textual content wraps beyond the height of the image content, it does not currently render beneath the image. Instead, it continues the box for the current textual content. In effect, we have two boxes rendered in parallel on the main axis and wrapping from one row to another simply extends the box for the text.

However, we may modify the current CSS rulesets to create a *newspaper* style design, which allows the textual content to wrap and render from column to column.

For example, we may create the following rendered output using a *multi-ccolumn* layout option within the current Flexbox container.



Kinkaku-ji, Kyoto

1. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec placerat sapien ac lacus bibendum, a sollicitudin quam interdum. Donec tristique purus nec sapien tempor rutrum. Proin eget quam pulvinar, imperdiet felis eu, vestibulum sem. Ut pellentesque venenatis neque, id volutpat diam congue sit amet. Etiam luctus nunc eu urna fermentum, vitae lobortis magna luctus. Sed id eros tincidunt turpis mollis ullamcorper ut porttitor diam. In et arcu rhoncus, tincidunt odio a, interdum est. Donec eu nisi tortor. Nulla vel volutpat metus. Sed justo erat, scelerisque et pretium non, sollicitudin at velit.

2. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec placerat sapien ac lacus bibendum, a sollicitudin quam interdum. Donec tristique purus nec sapien tempor rutrum. Proin eget quam pulvinar, imperdiet felis eu, vestibulum sem. Ut pellentesque venenatis neque, id volutpat diam congue sit amet. Etiam luctus nunc eu urna fermentum, vitae lobortis magna luctus. Sed id eros tincidunt turpis mollis ullamcorper ut porttitor diam. In et arcu rhoncus, tincidunt odio a, interdum est. Donec eu nisi tortor. Nulla vel volutpat metus. Sed justo erat, scelerisque et pretium non, sollicitudin at velit.

3. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec placerat sapien ac lacus bibendum, a sollicitudin quam interdum. Donec tristique purus nec sapien tempor rutrum. Proin eget quam pulvinar, imperdiet felis eu, vestibulum sem. Ut pellentesque venenatis neque, id volutpat diam congue sit amet. Etiam luctus nunc eu urna fermentum, vitae lobortis magna luctus. Sed id eros tincidunt turpis mollis ullamcorper ut porttitor diam. In et arcu rhoncus, tincidunt odio a, interdum est. Donec eu nisi tortor. Nulla vel volutpat metus. Sed justo erat, scelerisque et pretium non, sollicitudin at velit.

card footer...

Figure 5: CSS Flexbox - cards with media in column layout

We may update the CSS for the current card design,

```css
.card {
    column-width: 250px;
    column-rule: 1px solid #dddddd;
    margin: 10px 0;
}
```

The card itself will now render the image and text content as a *multi-column* layout within the overall Flexbox layout for the remainder of the card design.

Form fields and layouts Another common layout we might adapt with Flexbox is form groups, and the individual form fields.

For example, we may need to dynamically adjust the width and space required for a form's label, a text input field, username or password field, &c.

We may also need to modify a form group's layout to fit both portrait and landscape screen orientation.

example form group We may initially define a form group using the following HTML,

```
<form class="photos-search">
    <div class="search-container">
        <input type="text" id="text" value="search for photos...">
        <input type="submit" value="Search">
    </div>
</form>
```

In this example, we create a form grouping, and then add a generic `div` , which we may style as a Flex container.

```
.search-container {
    display: flex;
}
```

Each `input` field becomes a flex item, which we may choose to style or use the default flex settings.

For example, we'll add a ruleset for the form's text input field,

```
.search-container input[type="text"] {
    flex: 1 1 auto;
}
```

We may also add various optional properties to this ruleset to help with aesthetic design and initial layout. We might update the ruleset, for example, as follows

```
.search-container input[type="text"] {
    flex: 1 1 auto;
    border: 1px solid #5C9396;
    line-height: 2;
    margin: 0 5px 0 0;
    padding: 10px;
    color: #5C9396;
    font-weight: 450;
}
```

Such aesthetic properties and values are a design choice, and may, of course, be modified to fit a preferred look and design scheme.

We may then choose to extend such input fields with a label,

```
<label for="text">Photos</label>
```

As with `input` fields, we may then add aesthetic styling and structure for a label in the form group.

For example, we may design an initial form group as follows

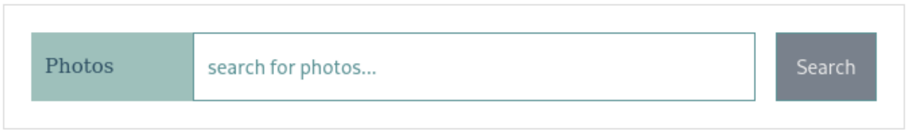


Figure 6: CSS Flexbox - form group with flex layout

References

- MDN - Flexbox
- W3 - CSS Units