Fluid Page Layouts with Flexbox

with **Laura Cooper**

Senior Web Developer, VERB Interactive

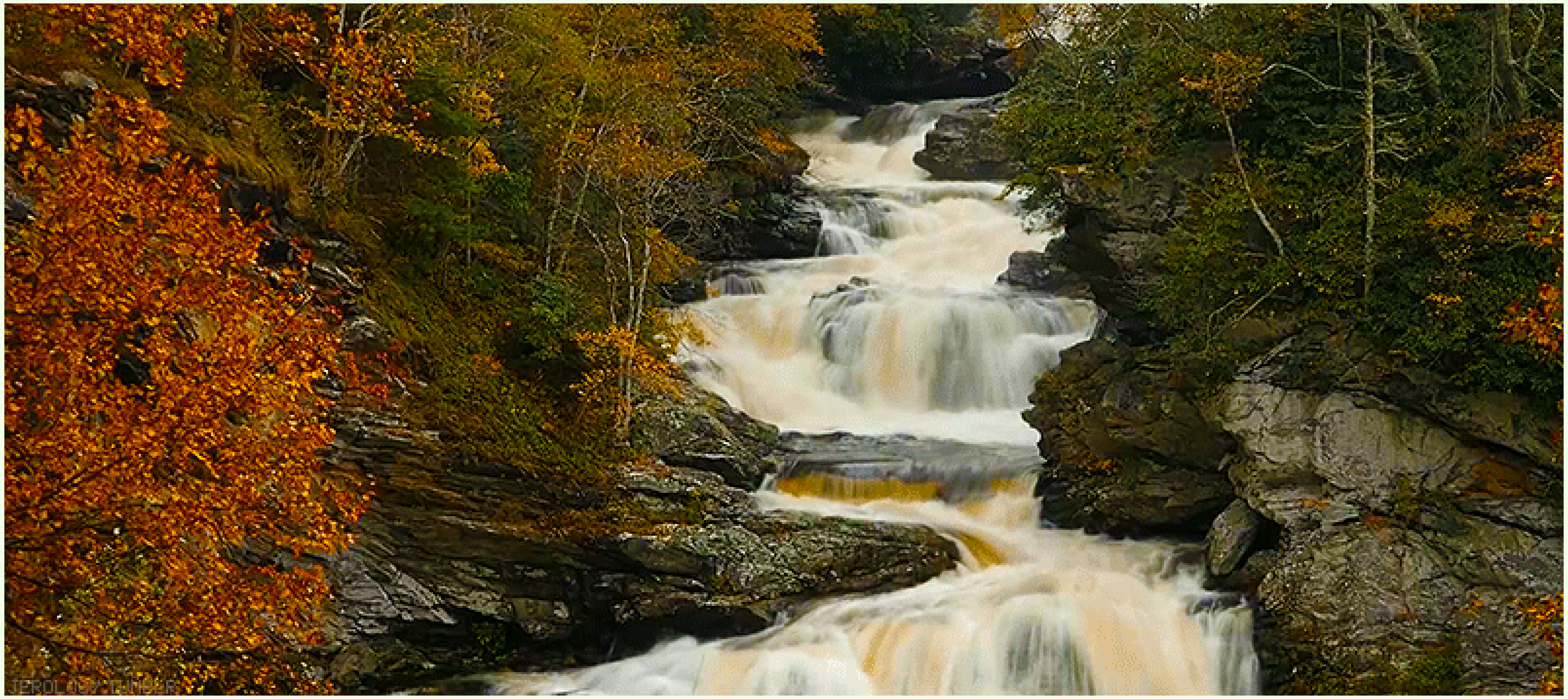
Strategic Volunteer & Instructor, Canada Learning Code

Board of Directors, YYJ Tech Ladies

Software & Workshop Files

- Download Sublime Editor: https://www.sublimetext.com/ (or use your text editor of choice)
- Chrome Browser: https://www.google.ca/chrome
- Download workshop files (zip file): https://goo.gl/Xv7G5n
- Unzip the zip file (*extract all* if you're on a PC)
- Open **slides.html** in Chrome to view the slides

CSS: Cascading Style Sheets



CSS Review

- CSS is a different language from HTML, with its own syntax rules
- HTML defines the content and structure of a page, whereas CSS defines how it looks

CSS Review

CSS requires these symbols:

curly brackets **{ }**   colon **:**   semi-colon **;**

CSS Syntax

- **Selectors** determine which HTML element(s) the styling rules apply to
- **Properties** determine what property we want to make a rule about, such as color or font size
- **Values** determine what that property will be set to, such as red or 20px

```
selector {
    property: value;
}
```

```
.my-class {
    font-size: 20px;
}
```

CSS Flexbox

So what is flexbox and why would we use it?

Flexbox is a collection of CSS properties that allow you to create a vast number of different layouts quickly and easily, such as vertical and horizontal centring, equal height columns, image galleries, split screen sections, equal height columns, and fluid grids. It is also supported by all modern web browsers.

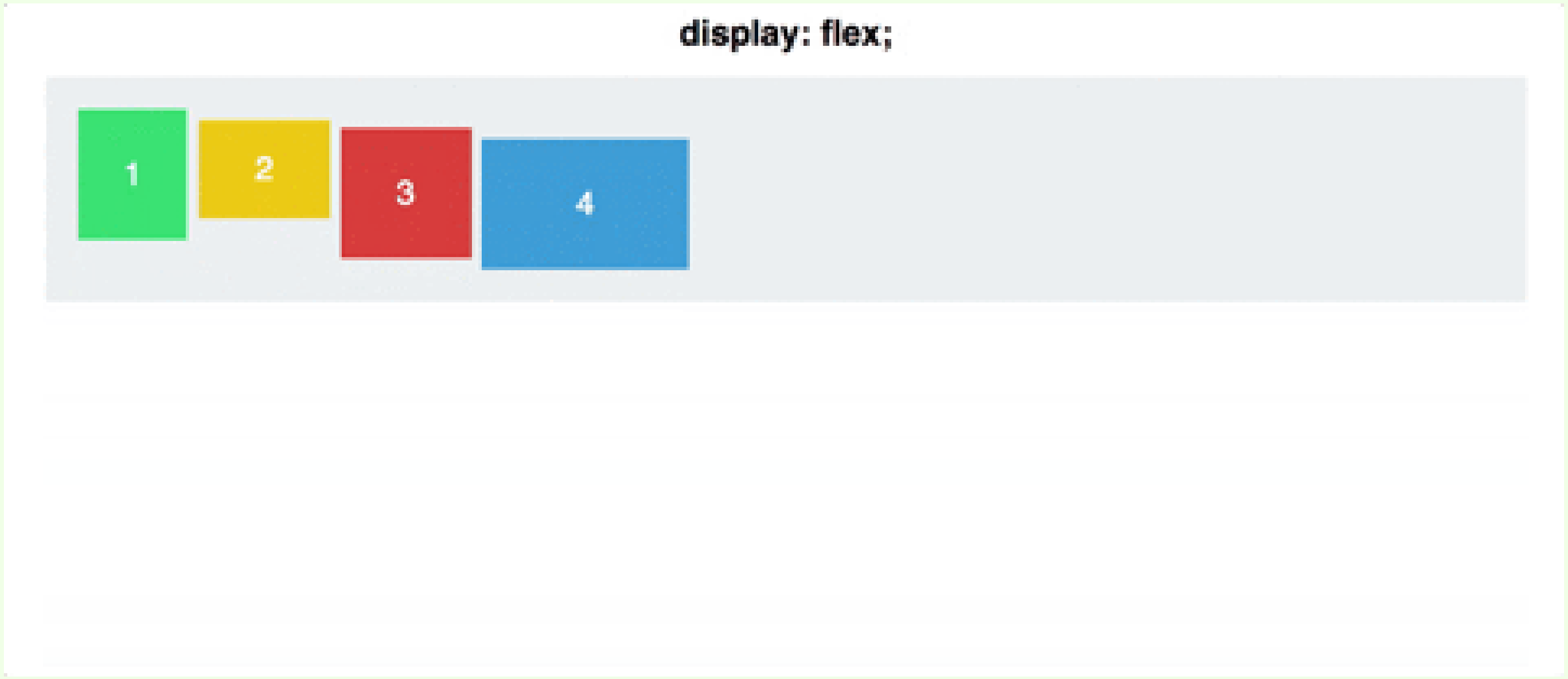
Display: Flex

Similar to **display: block;** or **display: inline**, **display: flex** gives the browser instructions on how to treat the container with respect to how it behaves with items around it.

Unlike block or inline however, **display: flex** also tells the browser how to lay out the items inside of itself.

You can also specify **display: inline-flex** which will allow the container to behave like an inline element and yet still allow the items inside it to be flexible and fluid.

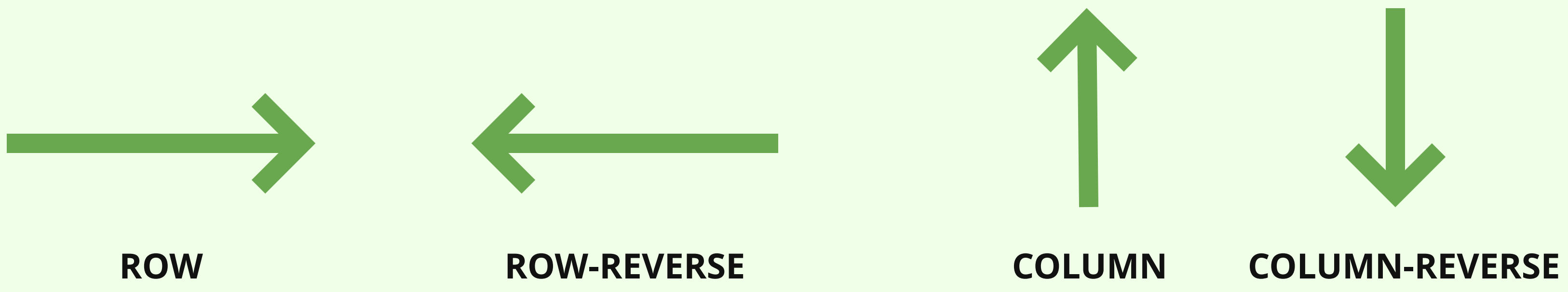
Display: Flex



Flex-Direction

This determines which direction the flexbox items will flow within the container.

- **row** (default): items flow horizontally (x-axis) from left to right
- **row-reverse**: items flow vertically (x-axis) from right to left
- **column**: items flow vertically (y-axis) from top to bottom
- **column-reverse**: items flow vertically (y-axis) from bottom to top



Flex-Wrap

The default flexbox behaviour is to try to fit all the items in its container on one line, but you can change this and make them wrap onto the next line using the flex-wrap property.

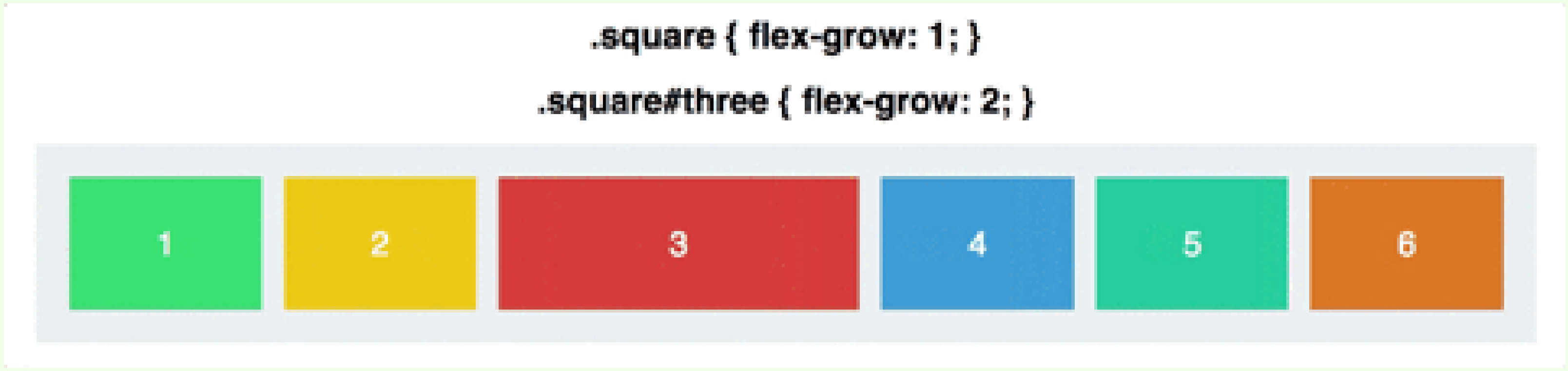
- **nowrap** (default): all items will be on one line
- **wrap**: items will wrap onto multiple lines from top to bottom
- **wrap-reverse**: items will wrap onto multiple lines from bottom to top

Flex Grow, Shrink, and Basis

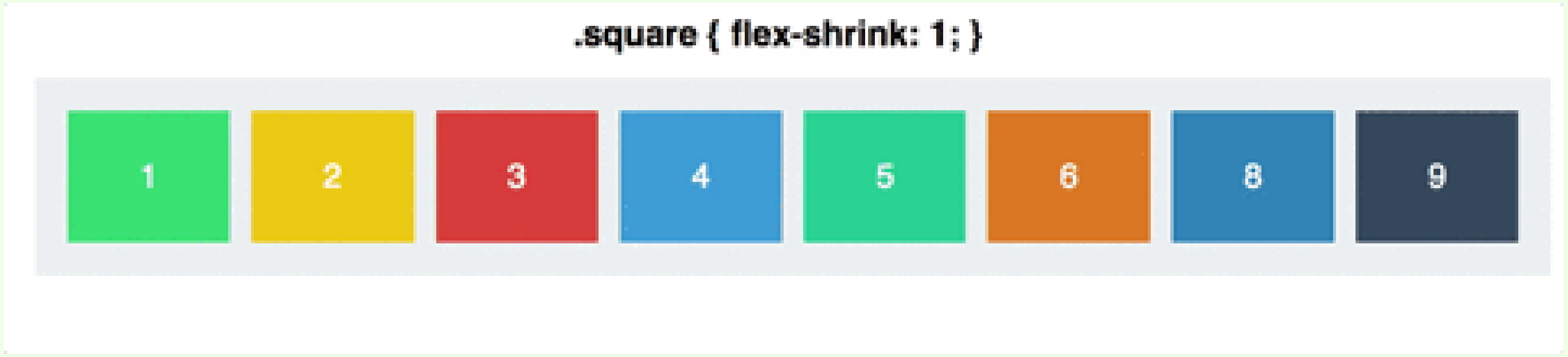
Flex-shrink and flex-grow dictate how much the items will shrink/grow to occupy space if the default size of the items is bigger or smaller than the size of the container. Flex-basis specifies the initial size of a flex item.

- **flex-shrink** & **flex-grow**: how much items will shrink or grow to fill the container, according to a specified unitless number relative to the rest of the flexible items in the container (default value is 1)
- **flex-basis**: the default size items should be, and accepts a pixel value, percentage value, or the keyword 'auto' (default value)

Flex Grow, Shrink, and Basis



```
.square { flex-grow: 1; }
.square#three { flex-grow: 2; }
```



```
.square { flex-shrink: 1; }
```

Flex (shorthand)

The **flex** property is the shorthand for **flex-grow**, **flex-shrink** and **flex-basis** combined.

The second and third property, flex shrink and flex-basis, are optional.

It is recommended to use this property rather than setting the three values separately, as this sets the other two values intelligently.

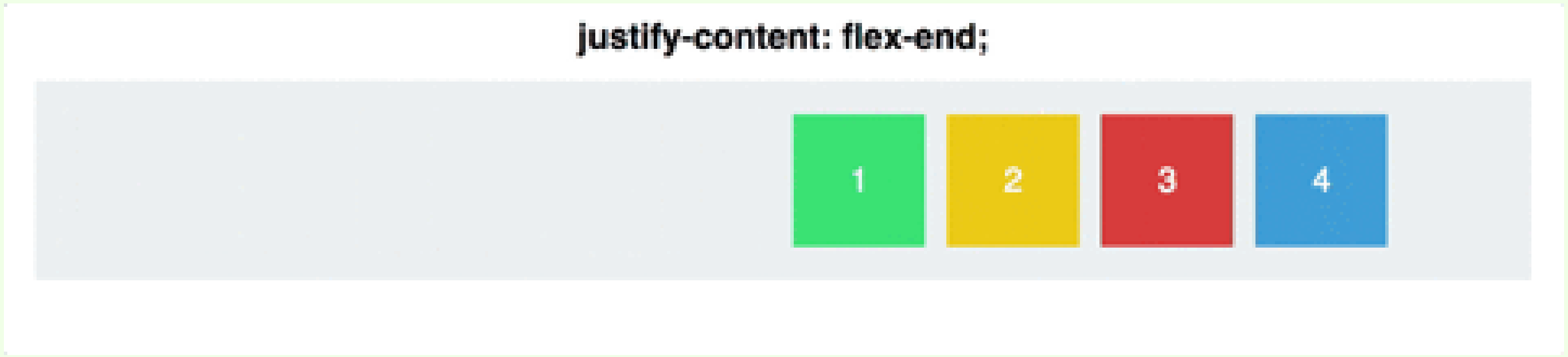
The default flex value is **1 0 auto**.

Justify-Content

Specifies how to spread out the content within a flexbox container. If flex-direction is set to row, this determines how the items are spread out on the x-axis. If flex-direction is column, it determines how the items are spread out on the Y-axis.

- **center**: items will all be in the center of the box
- **flex-start**: items will be aligned to the left or right side (row or row-reverse) or top or bottom of the box (column or column-reverse)
- **flex-end**: items will be aligned to the right side (row) or bottom of the box (column)
- **space-around**: items will have equal spacing around and between them
- **space-between**: the first and last items will be aligned to the beginning and end of the container, with equal spacing between all items
- **space-evenly**: items are distributed so that spacing between any two items and the spacing to the edges of the container are equal

Justify-Content

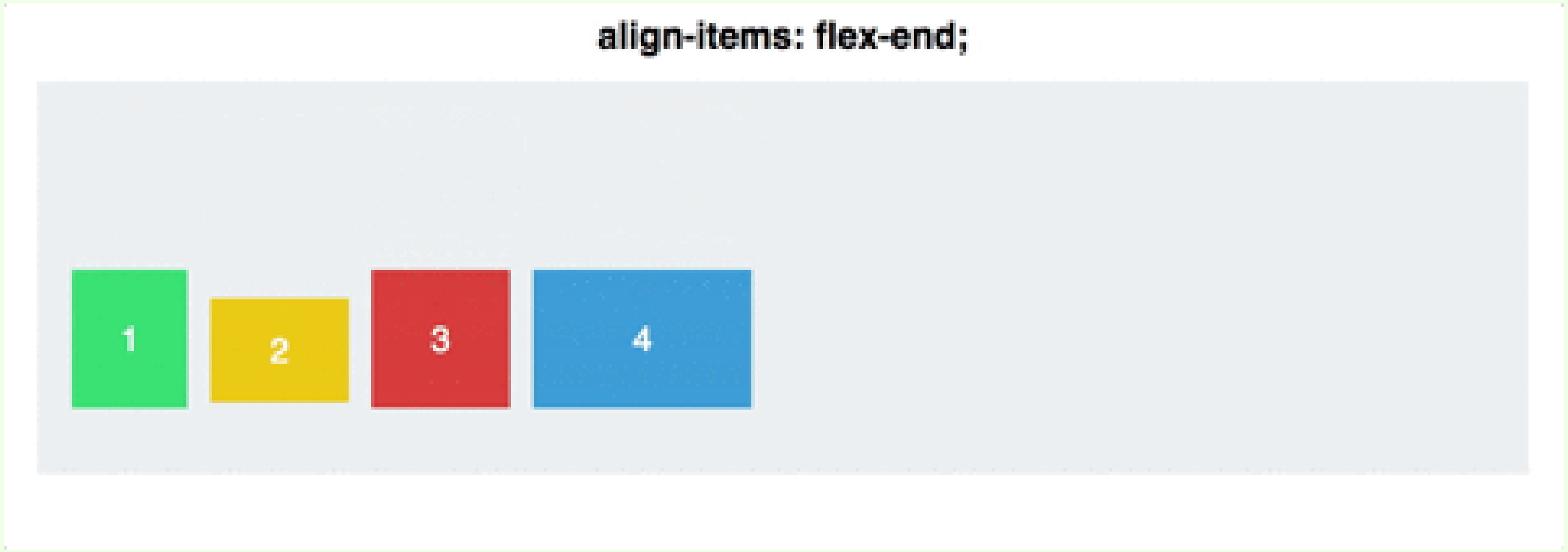


Align-Items

Specifies how to align the items within a flexbox container. This affects the opposite axis as justify-content. So, if flex-direction is row, this will determine the vertical alignment of its items. If flex-direction is column, this will determine the horizontal alignment.

- **flex-start**: items will be aligned so that the top of each item is at the top of the container
- **flex-end**: items will be aligned so that the bottom of each item is at the bottom of the container
- **center**: Items will be centered along the y-axis (or x-axis if flex-direction is column)
- **stretch**: Items will be stretched to all reach bot the top and bottom of the container (this is useful for when the desired effect is containers with equal height)
- **baseline**: items will be aligned based on the baseline of text content inside each item

Align-Items



Chrome Inspector Tool

Chrome Inspector Tool

The Inspector tool in Google Chrome is very useful when working with CSS.

To access the Inspector, right-click on a webpage while using Chrome and select **Inspect Element.**

Click the **Styles** tab to see all style rules being applied to an element from CSS stylesheets.

Project Files

For today's examples, all the HTML has been created for you already, with classes we can target using CSS, as well as some base styles for the page.

We'll mainly just be working with the **index.html** and **flexbox.css** files.

Open your **index.html** file in Chrome. You can do this by opening Chrome and going to **File > Open File** and navigating to the folder you downloaded earlier.

Sublime Text

Open Sublime or your preferred text editor and then go to **File > Open** and navigate to the folder you downloaded earlier. Select the **project** folder and click **Open**.

Double-click the **flexbox.css** file to open it.

Once you've made changes to your file, you can go to **File > Save** or simply hit **Command** (Mac) or **CTRL** (PC) **+ S**.

Refresh your **index.html** file in Chrome to see the changes.

Example 1: Hero

A **hero** section is a common layout used in web design. It is found at the top of the homepage underneath the header, and is a full-width, sometimes full-height image (or video) with text overlaid on top and centred. The hero section often contains the company tagline or website's main message/call-to-action.

```css
.hero {
  display: flex;
  align-items: center;
  justify-content: center;
}
```

Example 2: Intro

One of the most basic layouts is an **intro** section or basic text block, centred horizontally on the page. This is similar to what we just did on the hero section, but in this case, the height of the section is determined by the length of text + padding so no vertical centring is required.

```
.intro {
  display: flex;
  justify-content: center;
}
```

Example 3: Icons

Another use case for flexbox is when you have **icons, images** or **logos** of different sizes and you simply want to align them. We want our icons to be vertically centred in the section regardless of their size, and spaced out horizontally in a consistent way.

```
.icons {
  display: flex;
  align-items: center;
  justify-content: space-evenly;
}
```

Example 4: Split-Screen

A **split-screen** layout features two sections next to each other that each take up half the screen. It is common to see these with an image on one side and text on the other. Each side, or tile, needs to have equal height in order to achieve this layout.

```css
.tiles {
    display: flex;
}
```

```css
.tile {
  flex: 1 0 50%;
  display: flex;
  align-items: center;
  justify-content: center;
  height: 50vw;
}
```

Example 5: List

Another common content block is a **list** section - it is similar to the intro section as its height is dictated by the content and we want our text to be horizontally centred, so we can apply the same styles here by adding **list** to the existing **intro** declaration. We also want our list bullet icon and the list item text to be centred vertically.

```css
.intro, .list {
  display: flex;
  justify-content: center;
}
```

```css
.list li {
  display: flex;
  align-items: center;
}
```

Example 6: Image Grid

Our last layout is an **image grid**. We can easily use flexbox to create a fluid grid with images that resize consistently. We'll have to set flex-wrap to **wrap** so the images don't try to fit onto one line. We'll also add some styles to our gallery items to specify their width and height.

```css
.gallery {
  display: flex;
  flex-wrap: wrap;
}
```

```css
.gallery-item {
  flex: 0 0 25%;
  height: 25vw;
}
```

Additional Resources

CSS Flexbox (W3 Schools)

A Complete Guide to Flexbox (CSS Tricks)

Flexbox Cheatsheet

Flexbox Froggy (Game)

Flexbox Defense (Game)

THANK YOU!

Laura Cooper

Senior Web Developer, VERB Interactive

Strategic Volunteer & Instructor, Canada Learning Code

Board of Directors, YYJ Tech Ladies