CSS Can be hard

- Not a way of thinking you may be used to
- "Simple" concepts turn out to be hard
    - Ex: Centering
- Efforts often break things that were working
    - Ex: Fixed widths
- "Newer" options are **fantastic**
    - **Flexbox** (mid-2015)
    - **Grids** (mid-2017)
- Like **inline** vs **block**
    - Learn the bigger picture
    - Not a list of properties

Why and What is Flexbox?

Original CSS all based on how items (text) align in flow

- No grouping outside of containers with flow
- Everything based on the needs of content

Flexbox arranges child elements

- Into row(s) or column(s)
- Most often, 1 row or 1 column
- Children set their base size

Weird Flex, but....

- Apply `display: flex;` to parent container
- **Flexbox arranges space *for* and *around* children**
- You can set flex-related properties on
    - The container (parent)
        - Effects all children
        - Or space between them
    - On any child element
        - Effects that child

Key Flexbox Lessons and Guides

- `display: flex` on parent
    - Only arranges direct child elements
- Remember the difference between
    - Flex Properties on parent
    - Flex Properties on children
- A handy game-tutorial
    - **https://flexboxfroggy.com/**
- CSS Tricks has a famously good guide
    - **https://css-tricks.com/snippets/css/a-guide-to-flexbox/**

Common Flexbox properties

- `flex-direction` (column or row)
- `justify-contents` space along main axis
- `align-items` space along cross axis

Common Menu Example

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Internet Cats</title>
    <link rel="stylesheet" href="styles.css"/>
  </head>
  <body>
    <nav>
      <ul>
        <li><a href="/">Link</a></li>
        <li><a href="/">Link</a></li>
        <li><a href="/">Link</a></li>
      </ul>
    </nav>
  </body>
</html>
```

Notes about our Menu HTML

- Very basic example
- Menus are a common situation
- Semantic `<nav>` is good
    - No visual effect
    - Definite semantic effect
    - Labels our "navigation section"
        - Not all links are in a `<nav>` section
    - `<menu>` is for controls, not navigation
- We are lacking class names
    - You will later switch to using class names

Look at our default styling in DevTools

- Inspect the element
- Select (click!) the `<nav>` in Elements
    - See `display: block;` in Styles
- Hover over the `<nav>`
    - See the rendered highlight
- Select the `<ul>` in Elements
    - Can't see Styles until you SELECT it!
    - See default styling!
- Hover over the `<ul>`
    - See padding and margin highlights!

Overriding the defaults

- You will later use class name selectors!
    - Type selectors don't scale
        - Compound selectors = likely conflicts
        - Elements used for different purposes

```
ul {
    margin: 0;
    padding: 0;
}
```

Wait, where do the bullet points go?

- They are still there, just offscreen
- You can verify this:
    - In DevTools, SELECT (not just hover) the `<ul>`
    - Uncheck the `padding` in Styles
        - Checkbox visible again
    - Check the box again and now select the units
        - Press up/down to change the padding
        - See the bullet points move on/offscreen

Let's remove the bullet points from our menu

```
ul {
    margin: 0;
    padding: 0;

    list-style-type: none;
}
```

- You might WANT padding/margin/markers
    - Different situations have different decisions
- I'm just covering the process for when you decide
- See MDN for more options

Let's use flexbox

- `display: flex;` **arranges child elements**
- `<ul>` is parent of `<li>` elements

```
ul {
    display: flex;

    /* other css here */
}
```

Now we have a (squished) row!

Before anything else: `flex-direction`

`display: flex;` arranged our child elements in a **row**

- Flexbox arranges children in one dimension
- We can also arrange vertically instead of horizontally

```
ul {
    display: flex;
    flex-direction: column;

    /* other css here */
}
```

Let's set `flex-direction: row;` and keep going

Spacing with Flexbox

- Can set `margin`/`padding` on parent/children
- Can also set a `gap`
    - Space between rows/columns

```
ul {
    display: flex;
    gap: 2rem;
}
```

What if we want an even distribution of space?

- Try `justify-content: space-around;`
- Try `justify-content: space-between;`

What if we wanted one element off to the right?

- Common way to center without flexbox/grid
  - `margin: auto;`
  - Can also be used one just one side

```
ul { /* Better with class names */
  display: flex;
  gap: 2rem; /* preventing a squish */

  padding: 0;
  margin: 0;

  list-style-type: none;
}

li:last-child { /* MUCH better with class names */
    margin-left: auto;
}
```

Using DevTools with Flexbox

- Notice `<ul>` has a "flex" next to in Elements
- On hover, element highlights flexbox
    - Also shows gap
- Click the "flex" to see highlights w/o hover
    - Helps see last `<li>` taking up space
    - Hover over last `<li>` element to see margin

CSS Grids

Places child elements into cells

- Based on rows AND columns

Grids mimic the old table-based layouts

- Without their pain
- Because layout (mostly) separate from structure

Children can be told to span multiple "cells" of the grid

Grids put the emphasis on the layout over the content

How to Grid

Set parent container to `display: grid;`

- Define template columns and/or rows
- Can define areas

Later we'll examine using grids for **multi-column grids**

- For now, we will use **area labels**

Guides to CSS Grids

A game-tutorial

- **https://cssgridgarden.com/**

CSS Tricks

- **https://css-tricks.com/snippets/css/complete-guide-grid/**

Debugging with Chrome

- **https://developers.google.com/codelabs/devtools-debug-css-grid#0**

Grid areas (Other options later)

- Parent element:
  - `grid-template-areas`
    - Double quoted strings of space separated labels
      - No commas!
    - One string per row, label is column
  - `grid-template-rows`, `grid-template-columns`
    - Defines sizes of cells
    - `%` of available
    - `auto` matches needed space for content
    - `1fr` (or `2fr`, etc), divvy up remaining

Grid areas, Child Elements

- `grid-area`
    - Give label (no quotes)

Example HTML for a Grid

```
<header>
  <img src="http://placecats.com/50/50" alt="FIXME"/>
  <h1>Cats of the Internet</h1>
</header>
<nav>
  <ul>
    <li><a href="/">Link</a></li>
    <li><a href="/">Link</a></li>
    <li><a href="/">Link</a></li>
  </ul>
</nav>
<main>
  <p>lorem ipsum</p>
  <p>lorem ipsum</p>
</main>
<footer>
  Footer text here
</footer>
```

Notes about our HTML

- Good semantic HTML
  - **landmark** elements
    - **header**, **nav**, **main**, **footer**
- Needs class names!
  - Assignment will use element types
    - To exercise your **selector** understanding
  - Later you should use **semantic class names**
    - Element types will be used many ways
    - Compound element selectors invite **specificity issues**
- Alt on logo was inadequate

Using a Grid

`display: grid;` arranges **children elements**

- NOT descendants
    - We have 4 child elements for our grid
        - Children of `<body>` here
            - That's not always the parent

Examining Our Grid

```
body {
    display: grid;
}
```

Doesn't really show any visual change!

- "Inspect" the `<body>` in DevTools
    - See the `display: grid;` in "Styles"
    - See the "grid" next to element in "Elements"
    - Hover over element start tag in "Elements"
        - See the grid lines in rendered HTML!

Seeing the Grid in DevTools

- Click the word "grid" in "Elements"
    - Gridlines stay visible
    - Also Numbers (covered later)
        - Can change display in "Layouts"
- You can see the 4 cells of the grid
    - The four children of the element
        - The element given `display: grid;`

Planning our Grid

We want a grid of 2 columns and 3 rows

- header (across full width)
- nav menu on the left, main content on right
- footer (across full width)

What Dimensions?

- Start with column heights being "auto"
    - Based on height needed by content
- Start with row widths being "auto"
    - Based on width needed by content

Trying Multiple Columns

```
body {
    display: grid;
    grid-template-columns: auto auto; /* 2 cols */
    grid-template-rows: auto auto auto; /* 3 rows */
}
```

Now we have a mess!

- 2 rols, 2 cols, NOT 3 rows
- Parts are in the wrong cells

Cells were auto-filled

- We never told page what goes where
- Defaulted to first-come-first-serve

Explaining Named Grid Areas

- Multiple solutions to this!
- We will start with **grid areas**

```
body {
  display: grid;
  grid-template-columns: auto auto;
  grid-template-rows: auto auto auto;
  grid-template-areas:
    "header header"  /* Quoted lines */
    "menu main"       /* No commas between */
    "footer footer"
  ;
}
```

Remember: Newlines and indentation are for HUMANS

- Browser doesn't care
- But **Programming is Communication**

Applying Grid Area Names

- That didn't change the appearance!
- We created area names
    - The labels happen to match element names
    - (Mostly) Coincidence! Just labels!
- Did NOT tell browser which elements to use

```
footer {
    grid-area: footer; /* NOT quoted!*/
}
```

- Now `<footer>` spans the page!
- Now 6 cells!

Using Grid Areas

```
header {
    grid-area: header;
}
```

- Now looks correct (so far)!
- BUT we never labeled the other areas
    - Working by coincidence!
    - Always be complete, don't rely on luck
    - Someone will later change the page

```
nav {
    grid-area: menu; /* area names NOT the element name!*/
}
main {
    grid-area: main;
}
```

Changing Width

Navigation Menu is pretty wide

- Both columns are `auto` - evenly split

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

1fr means "1 part of all parts left over"

- 1 "fraction"
- 1 part out of 1 part - everything left after the `auto`

Now the Menu is too SQUISHED

- We can give the `<nav>` element some `padding`
- OR we can give our grid a **gap**
    - Distance between all cells
    - See impact in DevTools!

```
body {
  display: grid;
  grid-template-columns: auto 1fr;
  grid-template-rows: auto auto auto;
  grid-template-areas:
    "header header"
    "menu main"
    "footer footer"
  ;
  gap: 1rem;
}
```

I want the footer at the bottom of the page!

This is NOT a common desire!

- I'm using this to teach some concepts
- Most pages either have enough content
    - Or aren't bothered
- This is not a "best practice"
    - But it's not BAD either
    - We will use it to learn

The Footer IS at the bottom

...of the `<body>` element

Oh, you want it at the bottom of the viewport?

- Need to fill that with `<body>`

```
body {
    /* Other stuff like grid stuff here */

    min-height: 100vh;
}
```

`vh` is "percent of viewport height" units

- `100vh` is 100% of viewport height

body 100vh gives us a scroll bar!

This is because `body` defaults to `margin: 8px;`

- Commonly solved by a **CSS reset**
    - "Resets" browser CSS defaults
        - Before your specific changes
- Common meaning of "reset" has evolved
- No longer stripping inconsistent browser defaults
    - More "fixing" unhelpful browser defaults

```
body {
    /* Other stuff like grid stuff here */

    min-height: 100vh;
    margin: 0px; /* Perhaps move to a reset later */
}
```

Works, but mobile has complication

"Viewport height" gets a bit complicated on mobile

- Browser/OS may insert "virtual" controls
    - These will cover up part of the viewport!
    - Why some sites have buttons that overlap with device controls
- A new unit, `dvh` helps!
    - Recalculates height when controls appear
    - BUT it is relatively recent!
    - Not all users may have updated browsers!

Always check caniuse.com for feature support

http://caniuse.com/

This is not a yes/no answer!

- Supported in all major browsers?
- For how long?
- Can you fallback?

```
body {
  /* Other stuff like grid stuff here */

  min-height: 100vh; /* Normally overridden by next line */
  min-height: 100dvh; /* Browser ignores unless understood */

  margin: 0px; /* Perhaps move to a reset later */
}
```

Further complications

What happens if we add padding?

```
body {
  /* Other stuff like grid stuff here */

  min-height: 100vh; /* Normally overridden by next line */
  min-height: 100dvh; /* Browser ignores unless understood */

  margin: 0px; /* Perhaps move to a reset later */
  padding: 1rem;
}
```

Scrollbar is back!

- By default `padding` is NOT part of content height
- This is controlled by `box-sizing` property

Basic CSS Reset

You will build a list of changes you usually want by default

```
*, *::before, *::after {
  box-sizing: border-box;
  margin: 0;
}
```

Body is now full height, but everything stretched

Like when we the grid columns were evenly spaced

- Except with our row heights
- Want the header/footer only as tall as needed
- Want the menu/main row taking up extra space

```
body {
  display: grid;
  grid-template-columns: auto 1fr;
  grid-template-rows: auto 1fr auto;

  /* other css here */
}
```

Summary - Flow

Big common concepts in layout

- `inline` (flow of text)
- `block` (sections to organize)

Controlled by `display` property

Inline has limited sizing options

- Doesn't break text flowed

Block has width AND breaks flow

`inline-block` - inline-like flow, block-like sizing

Summary - Flexbox

Layout beyond inline and block

Arranges child elements based on one dimension

- Distributes space
- Distributes space between/around

Set by `display: flex;` (NOT `flexbox`)

- Other properties on parent and child

Summary - CSS Grid

Another option for layout beyond inline/block

Places child elements into cells formed by rows+cols

- Has "cells" to distribute space

Set by `display: grid;`

- Other properties on parent and child
- Set sizes for columns and/or rows

Can label cells for content

- Content can span many cells

Other options for assigning cells covered later