Comp 324/424 - Client-side Web Design

Fall Semester 2019 - Week 4

Dr Nick Hayward

Project outline & mockup assessment

Course total = 15%

- begin outline and design of a web application
  - *built from scratch*
  - *HTML5, CSS...*
  - *builds upon examples, technology outlined during first part of semester*
  - *purpose, scope &c. is group's choice*
  - ***NO** blogs, to-do lists, note-taking...*
  - *chosen topic requires approval*
  - *presentation should include mockup designs and concepts*

Project mockup demo

Assessment will include the following:

- brief presentation or demonstration of current project work
  - *~ 5 to 10 minutes per group*
  - *analysis of work conducted so far*
  - *presentation and demonstration*
  - *outline current state of web app concept and design*
  - *show prototypes and designs*
  - *due Tuesday 24th September 2019 @ 7pm*

CSS3 Grid - intro

- gid layout with CSS is useful for structure and organisation
  - *applied to HTML page*
- usage similar to table for structuring data
- in its basic form
  - *enables developers to add columns and rows to a page*
- grid layout also permits more complex, interesting layout options
  - *e.g. overlap and layers...*
- further information on MDN website,
  - *MDN - CSS Grid Layout*

CSS3 Grid - general concepts & usage

- grid may be composed of rows and columns
  - *thereby forming an intersecting set of horizontal and vertical lines*
- elements may be added to the grid with reference to this structured layout

Grid layout in CSS includes the following general features,

- additional tracks for content
  - *option to create more columns and rows as needed to fit dynamic content*
- control of alignment
  - *align a grid area or overall grid*
- control of overlapping content
  - *permit partial overlap of content*
  - *an item may overlap a grid cell or area*
- placement of items - explicit and implicit
  - *precise location of elements &c.*
  - *use line numbers, names, grid areas &c.*
- variable track sizes - fixed and flexible, e.g.
  - *specify pixel size for track sizes*
  - *or use flexible sizes with percentages or new `fr` unit*

CSS3 Grid - grid container

- define an element as a grid container using
  - *`display: grid`* or *`display: inline-grid`*
- any children of this element become *grid items*
  - *e.g.*

```
.wrapper {
  display: grid;
}
```

- we may also define other, child nodes as a `grid` container
  - *any direct child nodes to a grid container are now defined as grid items*

CSS3 Grid - what is a grid track?

- rows and columns defined with
  - *`grid-template-rows` and `grid-template-columns` properties*
- in effect, these define *grid tracks*
- as MDN notes,
  - *"a grid track is the space between any two lines on the grid.""*
  - *(https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout)*
- so, we may create both row and column tracks, e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: 200px 200px 200px;
}
```

- `wrapper` class now includes three defined columns of width 200px
  - *thereby creating three tracks*
- *n.b.* a track may be defined using any valid length unit, not just `px`...

CSS3 Grid - `fr` unit for tracks - part I

- CSS Grid now introduces an additional length unit for tracks, `fr`
- `fr` unit represents fractions of the space available in the current grid container
  - e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
}
```

- we may also apportion various space to tracks, e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: 2fr 1fr 1fr;
}
```

- creates three tracks in the grid
  - *but overall space effectively now occupies four parts*
  - *two parts for `2fr`, and one part each for remaining two `1fr`*

CSS3 Grid - `fr` unit for tracks - part 2

- we may also be specific in this sub-division of parts in tracks, e.g.

```
.wrapper {
  display: grid;
  grid-template-columns: 200px 1fr 1fr;
}
```

- first track will occupy a width of `200px`
  - *remaining two tracks will each occupy 1 fraction unit*

CSS3 Grid - `repeat()` notation for `fr` - part I

- for larger, repetitive grids, easier to use `repeat()`
  - *helps define multiple instances of the same track*
  - *e.g.*

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(4, 1fr);
}
```

- this creates four separate tracks - each defined as `1fr` unit's width

CSS3 Grid - `repeat()` notation for `fr` - part 2

- `repeat()` notation may also be used as part of the track definition
  - *e.g.*

```
.wrapper {
  display: grid;
  grid-template-columns: 200px repeat(4, 1fr) 100px;
}
```

- this example will create
  - *one track of `200px` width*
  - *then four tracks of `1fr` width*
  - *and finally a single track of `100px` width*
- `repeat()` may also be used with multiple track definitions
  - *thereby repeating multiple times*
  - *e.g.*

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(4, 1fr 2fr);
}
```

- this will now create eight tracks
  - *the first four of width `1fr`*
  - *and the remaining four of `2fr`*

CSS3 Grid - implicit and explicit grid creation

- in the above examples
  - *we simply define tracks for the columns*
  - *and CSS grid will then apportion content to required rows*
- we may also define an explicit grid of columns and rows
  - *e.g.*

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(2 1fr);
  grid-auto-rows: 150px;
}
```

- this slightly modifies an implicit grid to ensure each row is `200px` tall

CSS3 Grid - track sizing

- a grid may require tracks with a minimum size
  - *and the option to expand to fit dynamic content*
- e.g. ensuring a track does not collapse below a certain height or width
  - *and that it has the option to expand as necessary for the content...*
- CSS Grid provides a `minmax()` function, which we may use with rows
  - *e.g.*

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(2 1fr);
  grid-auto-rows: minmax(150px, auto);
}
```

- ensures each row will occupy a minimum of `150px` in height
  - *still able to stretch to contain the tallest content*
  - *whole row will expand to meet the `auto` height requirements*
  - *thereby affecting each track in the row*

CSS3 Grid - grid lines

- a grid is defined using *tracks*
  - *and not lines in the grid*
- created grid also helps us with positioning by providing numbered lines
- e.g. in a three column, two row grid we have the following,
  - *four lines for the three vertical columns*
  - *three lines for the two horizontal rows*
- such lines start at the left for columns, and at the top for rows
- *n.b.* line numbers start relative to written script
  - *e.g left to right for western, right to left for arabic...*

CSS3 Grid - positioning against lines

- when we place an item in a grid
  - *we use these lines for positioning, and not the tracks*
- reflected in usage of
  - *`grid-column-start`, `grid-column-end`, `grid-row-start`, and `grid-row-end` properties.*
- items in the grid may be positioned from one line to another
  - *e.g. column line 1 to column line 3*
- *n.b.* default span for an item in a grid is one track,
  - *e.g. define column start and no end - default span will be one track...*
  - *e.g.*

```
.content1 {
  grid-column-start: 1;
  grid-column-end: 4;
  grid-row-start: 1;
  grid-row-end: 3;
}
```

CSS3 Grid - grid cell & grid area

grid cell

- a *cell* is the smallest unit on the defined grid layout
- it is conceptually the same as a cell in a standard table
- as content is added to the grid, it will be stored in one cell

grid area

- we may also store content in multiple cells
  - *thereby creating grid areas*
- grid areas must be rectangular in shape
- e.g. a grid area may span multiple row and column tracks for required content

CSS3 Grid - add some gutters

- gutters may be created using the *gap* property
  - *available for either column or row*
  - *`column-gap` and `row-gap`*
  - *e.g.*

```
.wrapper {
  display: grid;
  grid-template-columns: repeat(4, 1fr 2fr);
  column-gap: 5px;
  row-gap: 10px;
}
```

- *n.b.* any space used for gaps will be determined prior to assigned space for `fr` tracks

CSS3 Grid - working examples

- grid basic - page zones and groups
- grid basic - article style page
- grid layout - articles with scroll

CSS3 Grid - sample layouts

intro

- grid layout enables more complex and interesting layout options
  - *overlap, layers...*
- sample layouts using CSS grid structure
  - *common layout options and designs*
  - *useful repetition of design*
  - *modify base layouts for various site requirements*
- sample layouts
  - *responsive layouts*
  - *auto placement for dynamic content and media*
  - *platform agnostic designs*
  - *useful with SPA, SVG, async patterns &c.*

CSS3 Grid - responsive layout

intro

- display a layout with a variety of patterns and structures, e.g.
  - *single column for a phone*
  - *add a sidebar for a tablet of lower window resolution*
  - *full width view with dual sidebars &c.*
- use responsive designs and structures for various games, media playback...
- responsive works with variety of markup
  - *e.g. transform SVG designs*

CSS3 Grid - responsive layout

page structure

- start with a sample page structure for a HTML page

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>CSS Grid - Responsive Layout</title>
    <link rel="stylesheet" type="text/css" href="./assets/style.css">
  </head>
  <body>
    <div class="wrapper">
            ...
        </div>
    </body>
</html>
```

CSS3 Grid - responsive layout

page structure - HTML5

- add some HTML5 markup for a `header`, `navigation`, `footer`, and some `main` content

```
<div class="wrapper">
    <header class="site-header">
        <h3>Spire & the Signpost</h3>
        <h5>Shine through the gloom, and point to the stars...</h5>
    </header>
    <nav class="site-nav">
        <ul>
            <li><a href="">Home</a></li>
            <li><a href="">Charts</a></li>
            <li><a href="">Data</a></li>
            <li><a href="">Views</a></li>
        </ul>
    </nav>
    <!-- use aside for tangentially related content for parent section... -->
    <aside class="content-side">
        <header>
            <h5>sidebar...</h5>
        </header>
    </aside>
    <main class="content">
        <article class="content-article">
            <header class="article-header">
                <h5>Welcome</h5>
            </header>
            <p>...</p>
        </article>
    </main>
    <section class="site-links">
        <h6>social links...</h6>
    </section>
    <footer class="site-footer">
        <h6>footer...</h6>
    </footer>
</div>
```

- demo - basic responsive

CSS3 Grid - responsive layout

CSS and structure - part 1

- for the page structure
  - *need to define some template areas for our grid in the CSS*
  - *e.g.*

```
/* CONTENT */
.content {
    grid-area: content;
}
```

- use such template area names
  - *defined with the `grid-area` property*
  - *define a layout for the overall page or part of a page*

CSS3 Grid - responsive layout

CSS and structure - part 2

- template areas may then be used with the parent for the grid structure
  - *e.g.* `wrapper` *for the overall page*

```
.wrapper {
    display: grid;
    grid-gap: 10px;
    grid-template-areas:
        "site-header"
        "site-nav"
        "content-side"
        "content"
        "site-links"
        "site-footer"
}
```

- `wrapper` class will display as a grid
  - *with a gap between each area of the grid*
  - *has a single column in this example*
  - *includes the required order for the grid areas*

CSS3 Grid - responsive layout

define media query

- current example would be suitable for a collapsed phone view
  - *single column view*
  - *will also render for other resolutions and devices*
- then add a media query for alternative layouts and displays
  - *may be triggered using a check of current width for screen*
  - *check width of window...*

```
/* min 700 */
@media (min-width: 700px) {
    .wrapper {
        grid-template-columns: 1fr 3fr;
        grid-template-areas:
        "site-header site-header"
        "site-nav site-nav"
        "content-side content"
        "site-links site-footer"
    }
}
```

CSS3 Grid - responsive layout

specific media query

- add further media queries to handle various rendering requirements
  - *e.g. add `height` property to fix footer at bottom of page*

```
@media (min-width: 700px) {
    .wrapper {
        grid-template-columns: 1fr 3fr;
        grid-template-rows: 120px 60px calc(98vh - 240) 60px;
        grid-template-areas:
        "site-header site-header"
        "site-nav site-nav"
        "content-side content"
        "site-links site-footer";
        height: 98vh;
    }
}
```

- specify height of current *viewport* using a relative unit, `vh`
- add `grid-template-rows` to define known heights for three of the four rows
- add a variant height for the main content
  - *main content is only given a height corresponding to available space in viewer window*
  - *height achieved using the `calc()` function*
- demo - responsive with specific media query

CSS3 Grid - responsive layout

relative lengths

- use relative lengths and calculations for CSS property values
- for example,
  - *`vw` - variable width relative to 1% of width of current viewport*
  - *`vh` - variable height relative to 1% of height of current viewport*
  - *`vmin` - relative to 1% of viewport's smaller dimension*
  - *`vmax` - relative to 1% of viewport's larger dimension*

CSS3 Grid - responsive layout

sample updates - part 1

- after testing this type of responsive layout
  - *we might add various updates*
  - *e.g. create a parent banner area for a header, user login, site search, and site nav*

```
.banner {
    grid-area: site-banner;
    display: grid;
    grid-template-columns: auto 300px;
    grid-template-rows: 120px 60px;
    grid-template-areas:
        "site-header banner-extras"
        "site-nav site-nav";
}
```

- helps manage layout and relative sizes of banner content
  - *regardless of page width and height*

CSS3 Grid - responsive layout

sample updates - part 2

- `banner-extras` might be styled as follows,

```
.banner-extras {
    grid-area: banner-extras;
    display: grid;
    grid-template-areas:
        "site-user"
        "site-search";
    padding: 5%;
}
```

- use of a child grid helps us manage fixed places within the parent `banner` area

CSS3 Grid - responsive layout

sample updates - part 3

- update our current media query for a `min-width` of 900px

```
/* min 900 */
@media (min-width: 900px) {
    .wrapper {
        grid-template-areas:
        "site-banner site-banner"
        "content-side content"
        "site-links site-footer";
        height: 98vh;
        grid-template-columns: 250px 3fr;
        grid-template-rows: 180px auto 60px;
    }
}
```

- demo - responsive layout - part 1
- demo - responsive layout - part 2

CSS3 Grid - auto placement

dynamic content and media - part 1

- also use CSS grid with Flexbox to create content layouts
  - *e.g. similar to placing cards in the UI*
- we might create a layout to dynamically render images for a photo album
  - *or a series of products in a brochure &c.*
- start by defining a simple list with various list items

```
<ul class="items">
    <li>One</li>
    <li>Two</li>
    <li>Three</li>
    <li>Four</li>
    <li>Five</li>
    <li>Six</li>
    <li>Seven</li>
    <li>Eight</li>
    <li>Nine</li>
</ul>
```

CSS3 Grid - auto placement

dynamic content and media - part 2

- then render these list items in flexible columns within our grid layout
  - *define a minimum size*
  - *then ensure they expand to equally fill available space*
- ensures rendered layout includes equal width columns regardless of available content

```
/* content items */
.items {
    display: grid;
    grid-gap: 5px;
    grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
    list-style: none;
}
```

- and render individual items using flexbox

```
.items li {
    border: 1px solid #3b8eb4;
  display: flex;
  flex-direction: column;
}
```

- demo - dynamic content - part 1
- demo - dynamic content - part 2

CSS3 Grids - fun layout

a game board

- also use a grid layout for internal uses
  - *e.g. design a game board*
- basic HTML list use for 3x3 game board
  - *each list item as a square on the board*

```
<main class="content">
    <ul class="items">
        <li>
            <h5>One</h5>
        </li>
        <li>
            <h5>Two</h5>
        </li>
        <li>
            <h5>Three</h5>
        </li>
        <li>
            <h5>Four</h5>
        </li>
        <li>
            <h5>Five</h5>
        </li>
        <li>
            <h5>Six</h5>
        </li>
        <li>
            <h5>Seven</h5>
        </li>
        <li>
            <h5>Eight</h5>
        </li>
        <li>
            <h5>Nine</h5>
        </li>
    </ul>
</main>
```

CSS3 Grids - fun layout

a game board - part 2

- then create the grid for the `content` class

```
/* CONTENT */
.content {
    grid-area: content;
    display: grid;
    grid-template-areas:
        "items";
    grid-template-columns: 1fr;
    align-self: center;
    justify-self: center;
    align-items: center;
    padding: 50px;
    border: 1px solid #aaa;
}
```

- we can embed this content area within other grids
  - *then add child items for the grid content itself*
- `content` container will be aligned and justified to the centre of the parent
- each child column will occupy the same proportion of available space
  - *using `grid-template-columns: 1fr`*
- each item will also be aligned to the centre of the available space
- properties such as padding and border are optional
  - *e.g. dictated by aesthetic requirements...*

CSS3 Grids - fun layout

grid items for the board - part 1

- each square will be a child list item to the parent `ul`
  - *e.g. style `ul` as follows*

```
.items {
    grid-area: items;
    display: grid;
    grid-gap: 10px;
    grid-template-columns: repeat(3, 150px);
  grid-template-rows: 150px 150px 150px;
}
```

CSS3 Grids - fun layout

grid items for the board - part 2

- then style each item, which creates the squares on the game board

```
.items li {
    margin: 0;
    list-style-type: none;
    border: 1px solid #333;
    background-color: #333;
    color: #fff;
    padding: 10%;
}
```

- styling is for aesthetic purposes
  - *e.g. to render a list item as a square without the default list style*
- also define an alternating colour scheme for our squares, e.g.

```
.items li:nth-child(even) {
    border: 1px solid #ccc;
    background-color: #ccc;
    color: #333;
}
```

- demo - Fun with Squares

CSS3 Grid - structure and layout

fun exercise

Choose one of the following app examples,

- **sports** website for latest scores and updates
  - *e.g. scores for current matches, statistics, team data, player info &c.*
- **shopping** website
  - *product listings and adverts, cart, reviews, user account page &c.*
- **restaurant** website
  - *introductory info, menus, sample food images, user reviews &c.*

Then, consider the following

- use of a **grid** to layout your example pages
  - *where is it being used?*
  - *why is it being used for a given part of the UI?*
- how is the defined **grid** layout working with the **box model**?
- rendering of **box model** in the main content relative to **grid** usage
  - *i.e. box model updates due to changes in content*

CSS - Flexbox

intro

- helps solve many issues that have continued to plague layout and positioning
- used with HTML elements and components
  - *both client-side and cross-platform apps*
- a few issues it tries to solve
  - *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...*

CSS - Flexbox

basic usage

- for any app layout, we need to define specific elements as *flexible boxes*
- i.e. those allowed to use flexbox in a given app
  - *e.g.*

```
section {
  display: flex;
}
```

- ruleset will define a `section` element as a parent flex container
  - *child elements may now accept flex declarations*
- initial declaration, `display: flex`
  - *also includes default values for flexbox layout of child elements*
- e.g. `<div>` elements in a section
  - *by default now arranged as equal sized columns with the same initial height*

CSS - Flexbox

axes

- elements arranged using flexbox are laid out on two axes
- main axis
  - *axis running in the direction of the currently laid out flex items*
  - *e.g. rows or columns*
  - *start and end of axis = main start & main end*
- cross axis
  - *axis running perpendicular to the current main axis*
  - *start and end of axis = cross start & cross end*
- each child element laid out inside flex container called a *flex item*

CSS - Flexbox

flex direction

- set a property for the flex direction
  - *defines direction of flex items relative to main axis*
  - *i.e. layout direction for child elements*
- default setting is `row`
  - *direction will be relative to current browser language setting*
  - *e.g. for English language browsers = left to right*

```
section {
  flex-direction: column;
}
```

- override the default `row` setting
  - *arrange child items in a column*

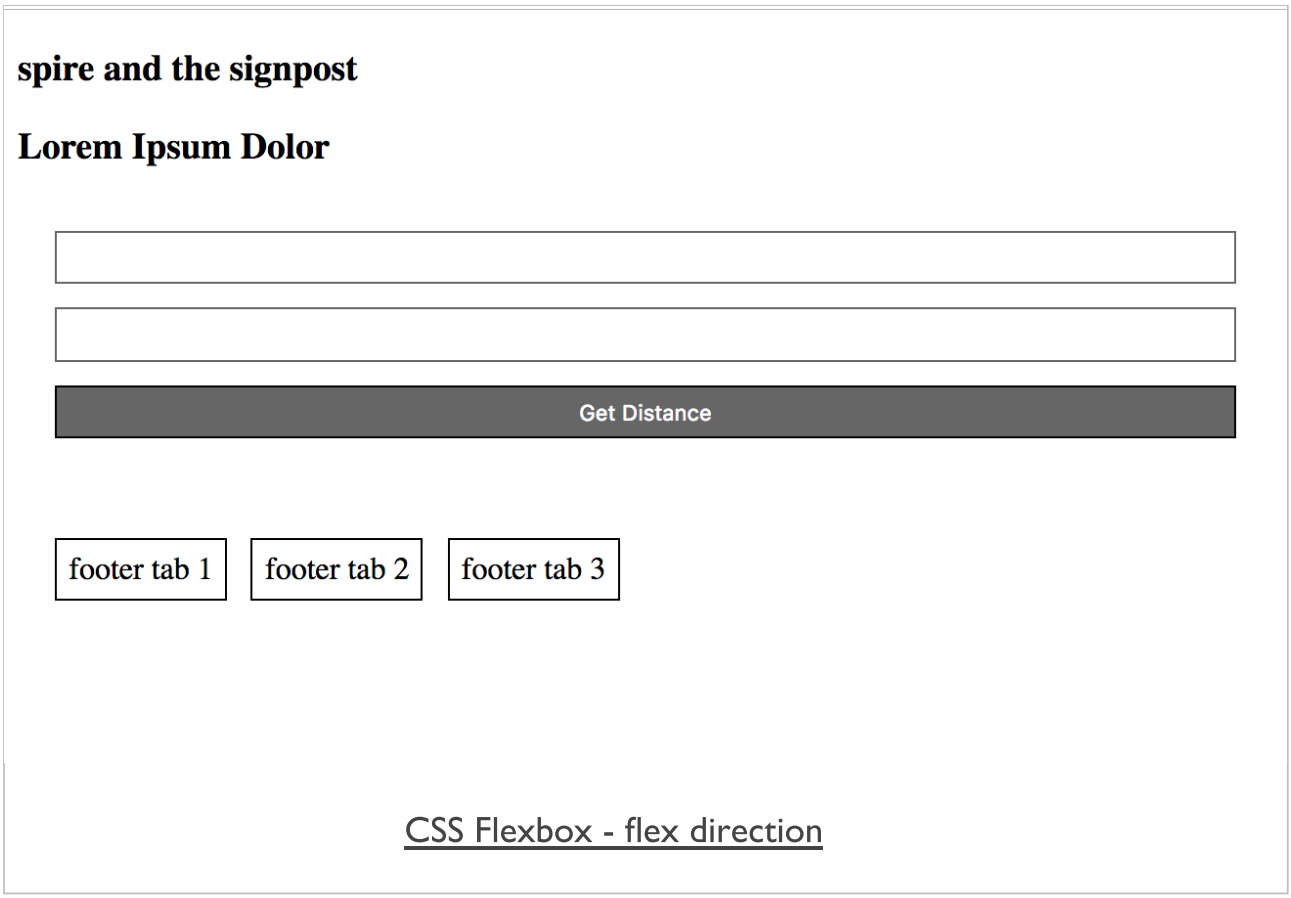
```
section {
  display: flex;
  flex-direction: column;
}
```

- ensures child flex items were laid out in a single column
- then override specific `section` elements
  - *allow child flex items in a `row` direction*

```
#tabs {
  flex-direction: row;
}
```

Image - CSS Flexbox

flex direction



CSS - Flexbox

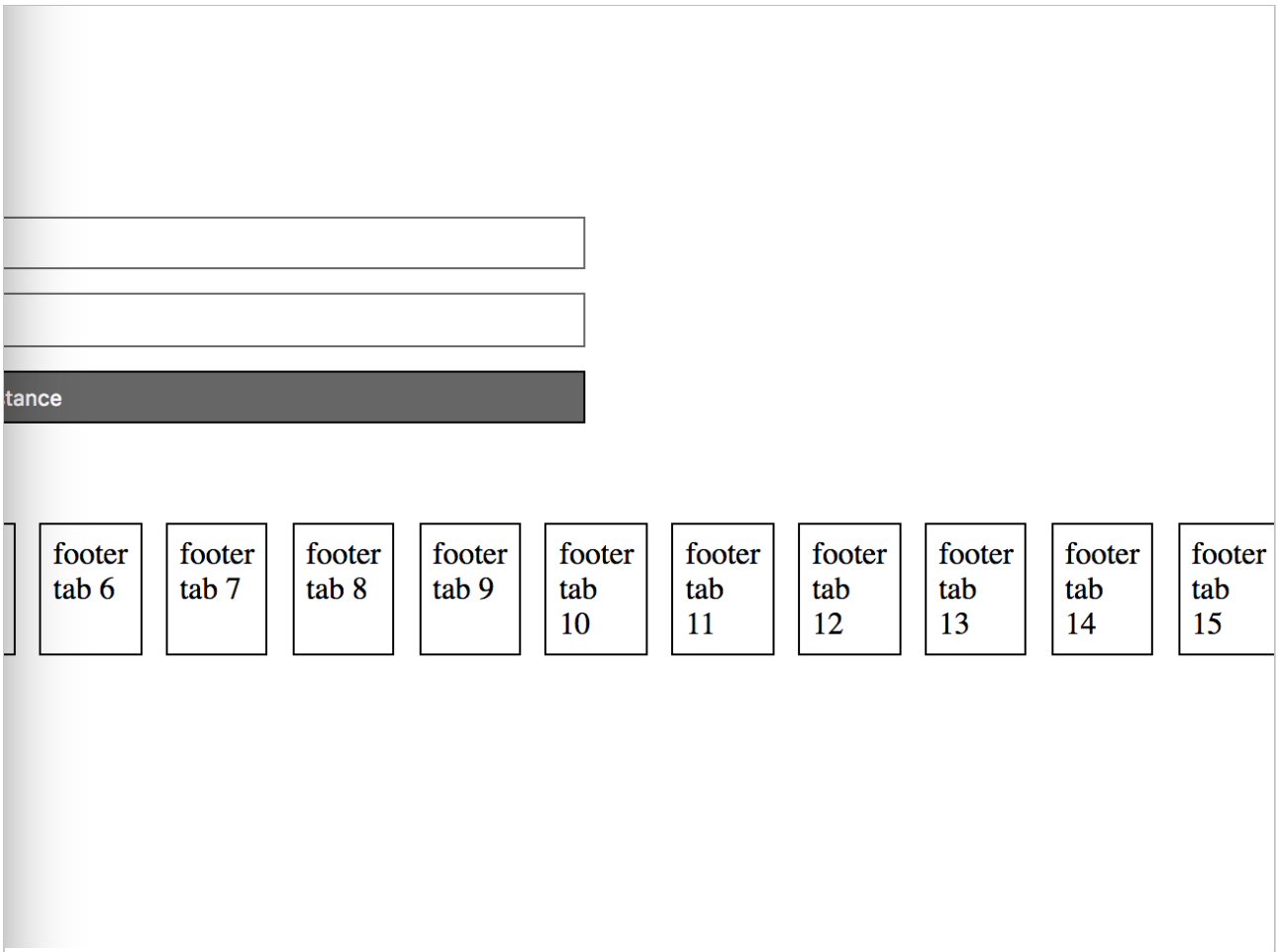
flex item wrapping

- ensure child items do not overlap their parent flex container
  - *add a declaration for `flex-wrap` to a required ruleset*
  - *e.g.*

```
#tabs {
  flex-direction: row;
  flex-wrap: wrap;
}
```

Image - CSS Flexbox

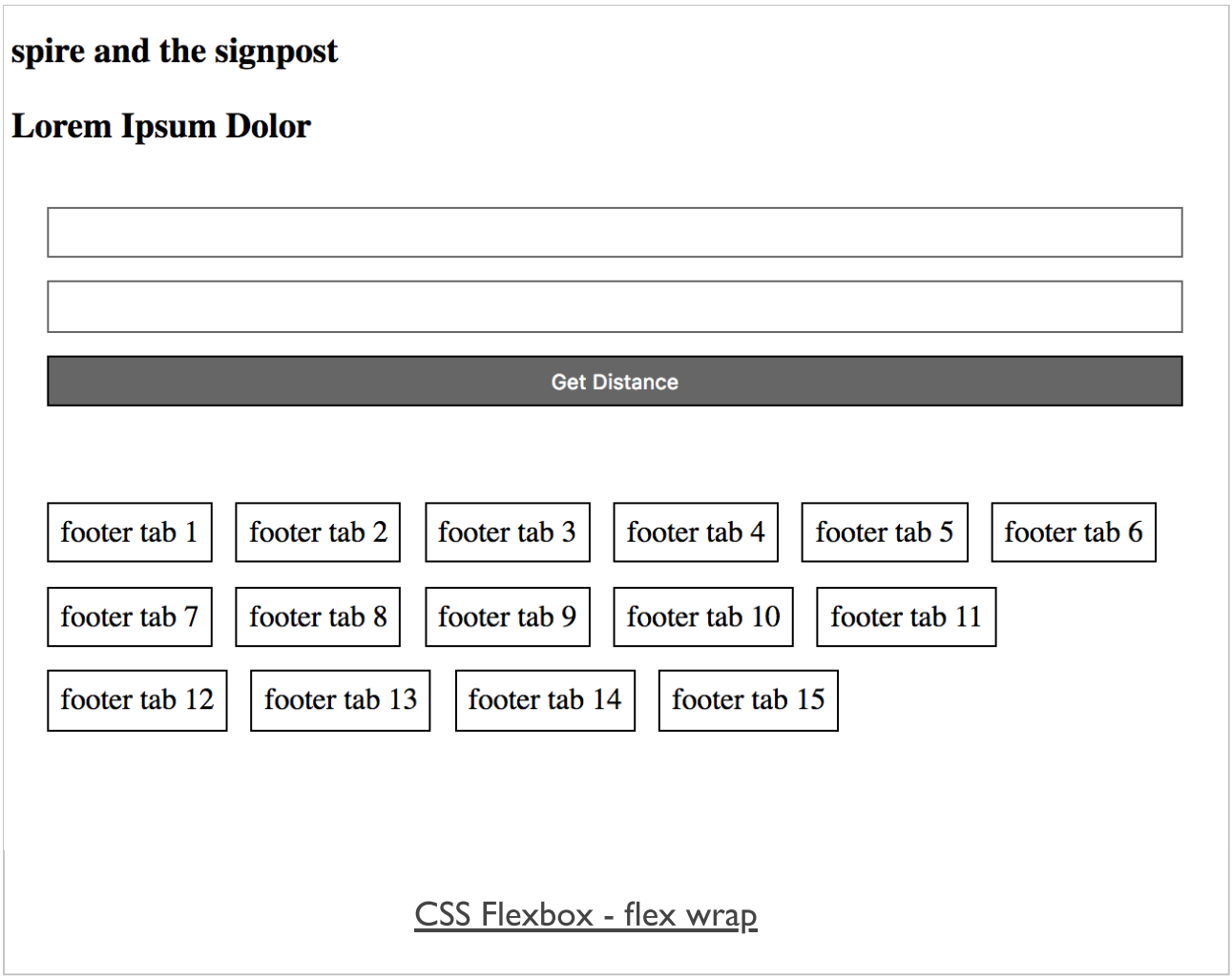
without wrap



CSS Flexbox - no flex wrap

Image - CSS Flexbox

with wrap



CSS - Flexbox

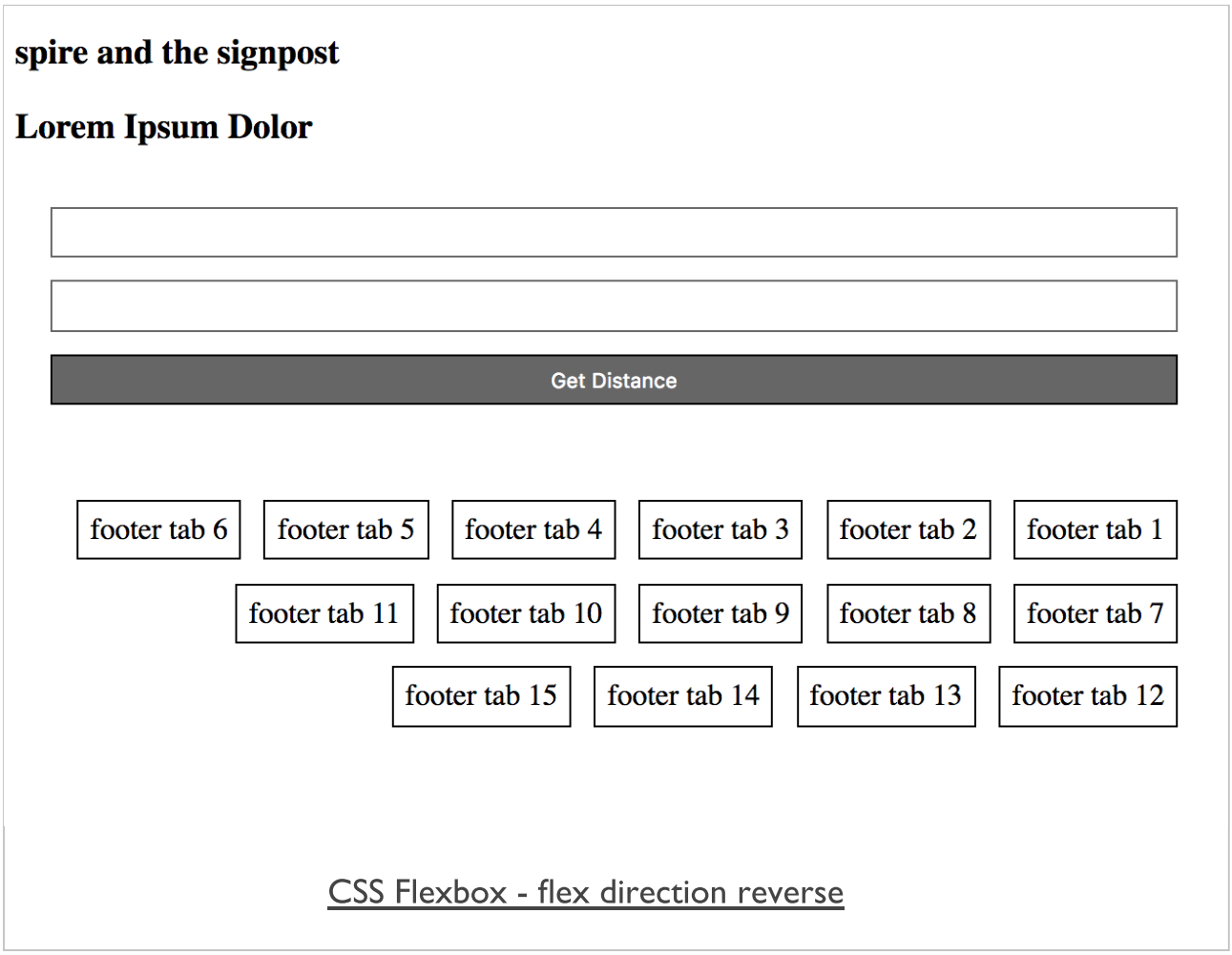
flex direction reverse

- also set flex direction to reverse
  - *starts flex items from the right on an English language browser*

```
#tabs {
  flex-direction: row-reverse;
  flex-wrap: wrap;
}
```

Image - CSS Flexbox

flex direction reverse



CSS - Flexbox

`flex-flow` shorthand

- also combine *direction* and *wrap* into a single declaration
  - *`flex-flow`*
  - *now contain values for both `row` and `wrap`*
  - *e.g.*

```
#tabs {
  flex-flow: row wrap;
}
```

CSS - Flexbox

sizing of flex items

- for each flex item, we may need to specify apportioned space in the layout
  - *e.g. set space as an equal proportion for each flex item*
  - *we may add the following to a child item ruleset*

```
div.fTab {
  flex: 1;
}
```

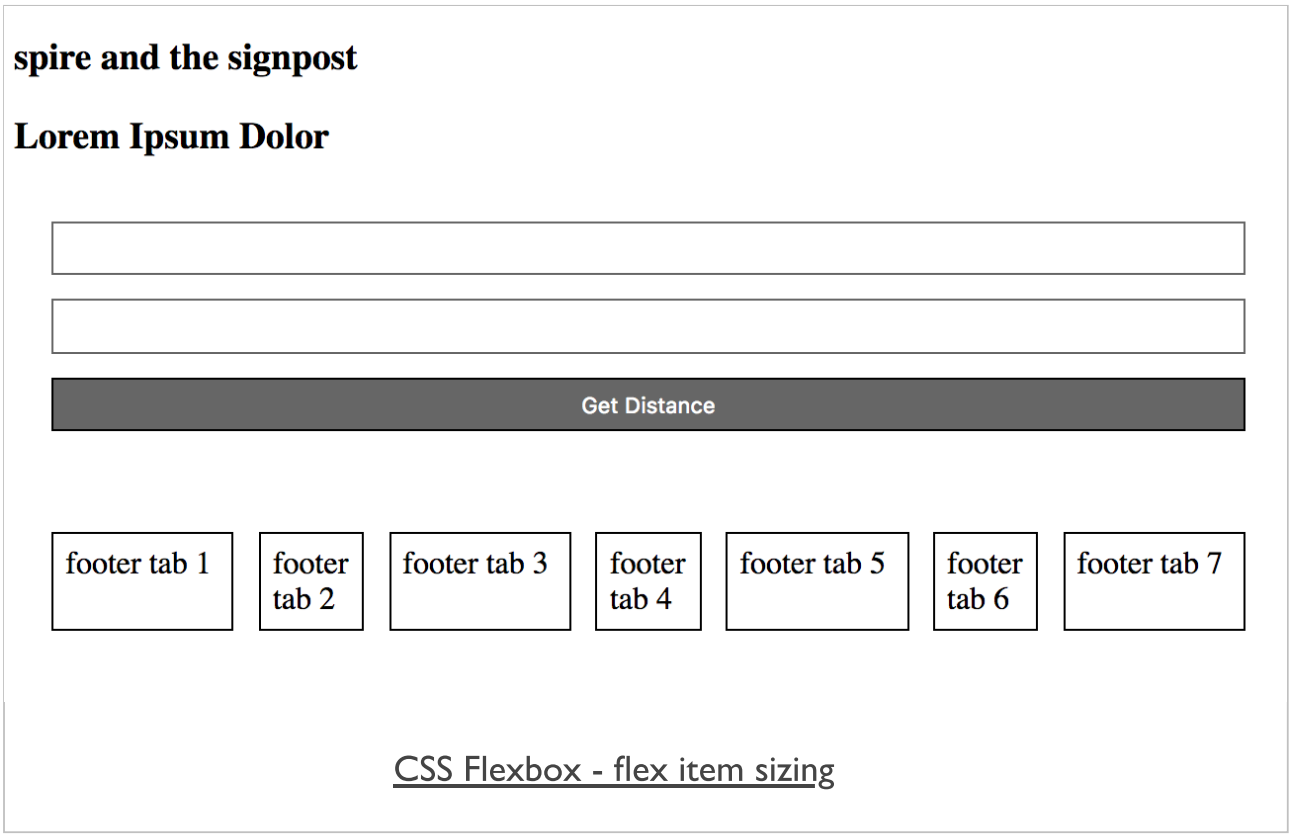
- defines each child flex item `<div class="fTab">`
  - *occupy an equal amount of space within the given row*
  - *after considering margin and padding*
- **n.b.** this value is proportional
  - *doesn't matter if the value is 1 or 100 &c.*
- define additional flex proportions for specific child items
  - *e.g.*

```
div.fTab:nth-child(odd) {
  flex: 2;
}
```

- each odd *flex-item* will now occupy twice available space
  - *space in the current direction*

Image - CSS Flexbox

flex item sizing



CSS - Flexbox

minimum size

- then set a minimum size for a flex item
  - e.g.

```
div.fTab {
  flex: 1 100px;
}
```

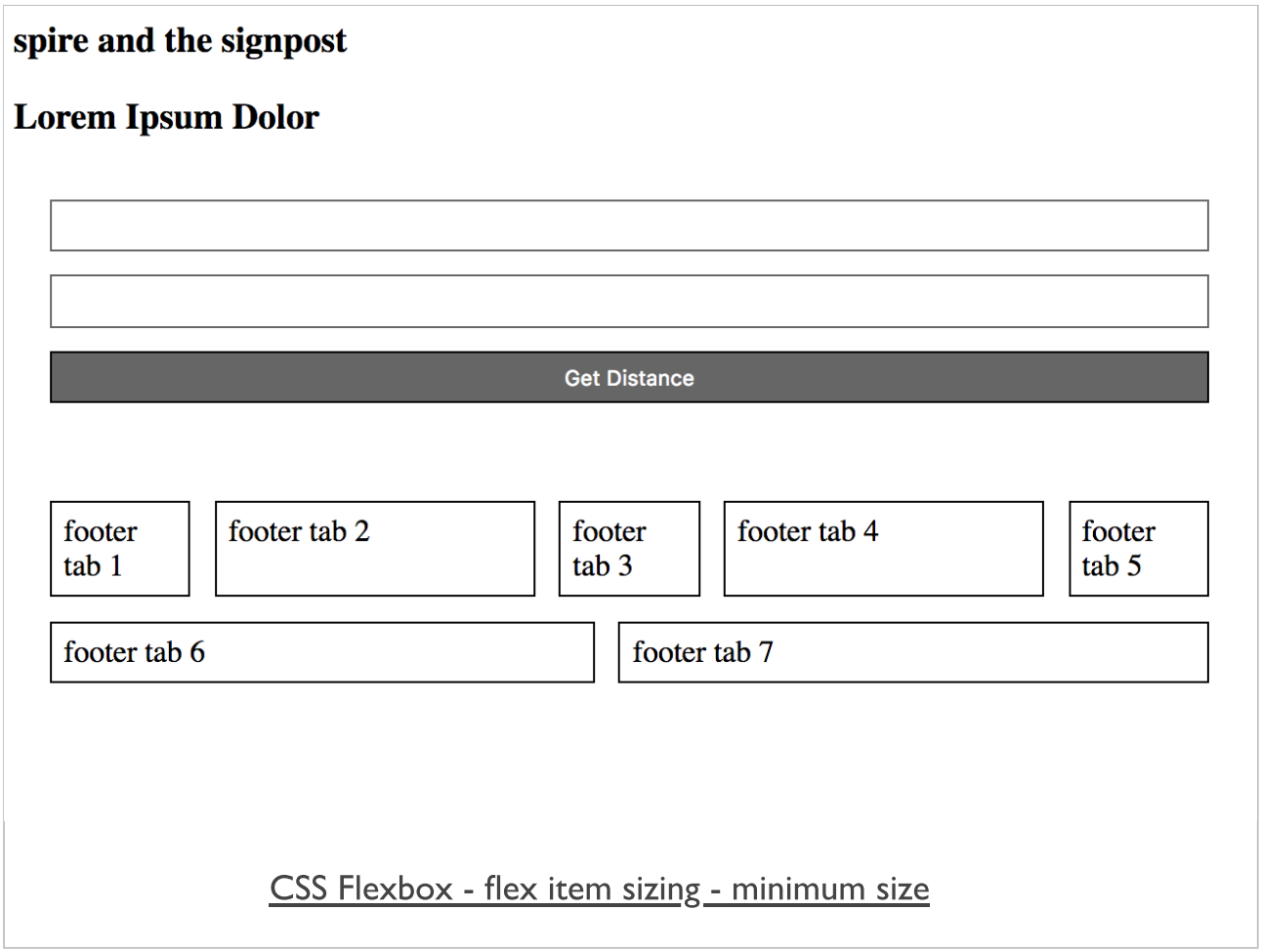
- or a relative unit for the size

```
div.fTab {
  flex: 1 20%;
}
```

- each flex item will initially be given a minimum
  - *e.g. 20% of the available space*
  - *the remaining space will be defined relative to proportion units*

Image - CSS Flexbox

flex item sizing



CSS - Flexbox

flex item alignment

- Flexbox allows us to define alignment for flex items in each flex container
  - *relative to the main and cross axes*
- e.g. we might want to specify a centred alignment for flex items

```
#tabs {
  flex-direction: row;
  flex-wrap: wrap;
  align-items: center;
}
```

- `align-items: center`
  - *causes flex item in flex container to be centred along the cross axis*
  - *however, they'll still maintain their basic dimensions*
- also modify value for `align-items` to either `flex-start` or `flex-end`
- such values will align flex items to either start or end of cross axis

CSS - Flexbox

override align per flex item

- as with `flex`
  - *also override alignment per flex item*
  - *using `align-self` property add a value for positioning*
- e.g. a sample declaration might be as follows

```
div.fTab:nth-child(even) {
  flex: 2;
  align-self: flex-end;
}
```

CSS - Flexbox

justify content for flex item

- also specify `justify-content` for flex items in a flex container
  - *property allows us to define position of a flex item relative to main axis*
- default value is `flex-start`
- then modify it relative to one of the following
  - *`flex-end`*
  - *`center`*
  - *`space-around`*
    - distributes each flex item evenly along main axis with space at either end
  - *`space-between`*
    - same as `space-around` without space at either end...

CSS - Flexbox

alignment and order - part I

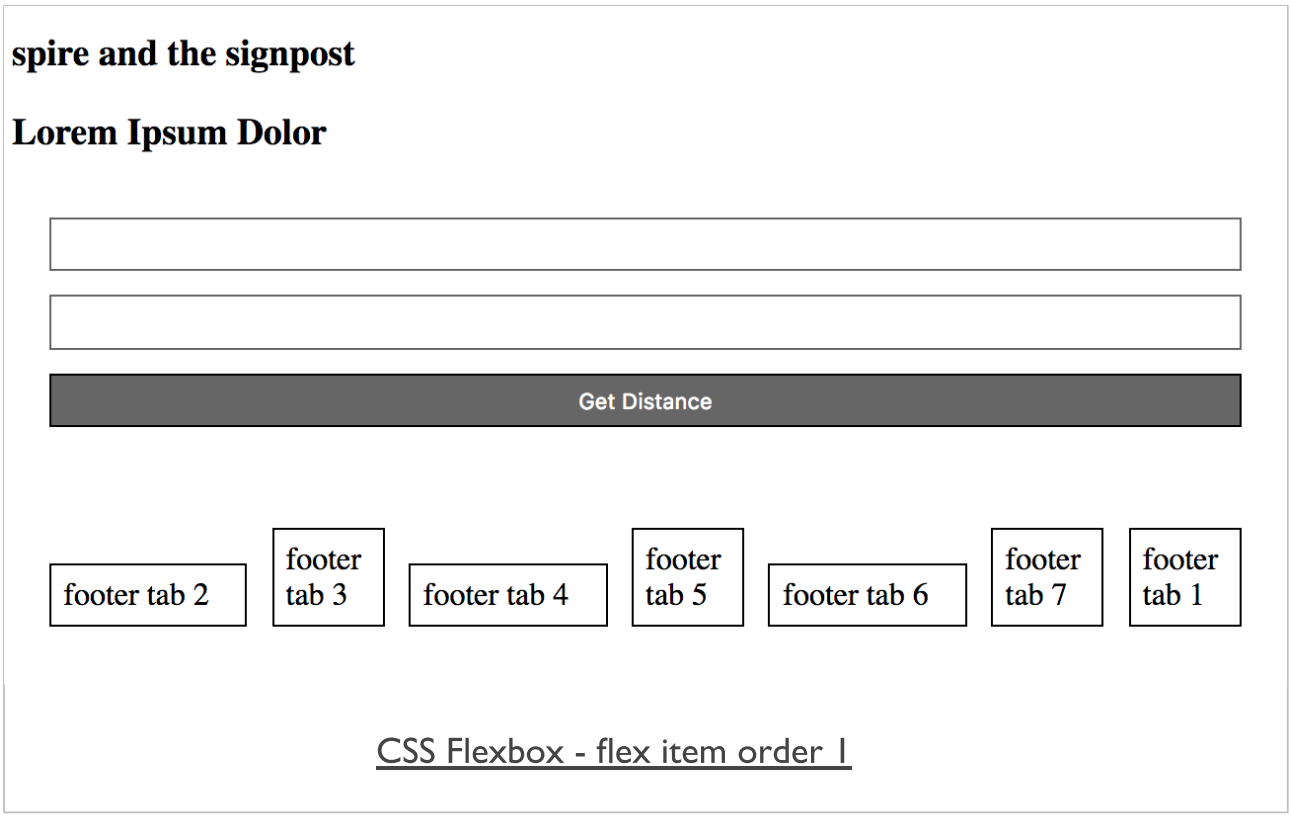
- define alignment relative to each axis using a specific declaration
  - *e.g. for the main we may use* `justify-content`
  - *for the cross axis we use* `align-items`
- also modify layout order of flex items
  - *without directly changing underlying source order*
- use the following pattern to specify order

```
div.fTab:first-child {
  order: 1;
}
```

- first flex item will now move to the end of the tab list

Image - CSS Flexbox

flex item order



CSS Flexbox - flex item order 1

CSS - Flexbox

alignment and order - part 2

- due to default order for flex items
  - *by default, all flex items have an `order` value set to `0`*
- higher the `order` value, later the item will appear in the list &c.
- items with the same order will revert to the order in the source code
- also possible to ensure certain items will always appear first
  - *or at least before default `order` values*
  - *by using a negative value for the `order` declaration*
  - *e.g.*

```
div.fTab:last-child {
  order: -1;
}
```

CSS - Flexbox

nesting flex containers and items - part 1

- Flexbox can also be used to create nested patterns and structures
  - *e.g. we may set a flex item as a flex container for its child nodes*
- we might add a banner to the top of a page

```
<section id="banner">
  <header id="page-header">
    <h3>spire and the signpost</h3>
    <h5>point to the stars...</h5>
  </header>
  <section id="search">
    <input type="text" id="searchBox"/>
    <button id="searchBtn">Search</button>
  </section>
</section>
```

CSS - Flexbox

nesting flex containers and items - part 2

- set `#banner`, `#page-header`, and `#search` as flex containers
  - *e.g.*

```
#search {
  display: flex;
}
```

- then specify various declarations for `#search`
  - *e.g.*

```
#search {
  display: flex;
  flex-direction: row;
  flex: 2;
  align-self: flex-start;
}
```

- includes values for itself and any child elements
  - *if we then add some rulesets for the nested flex items*
  - *e.g.*

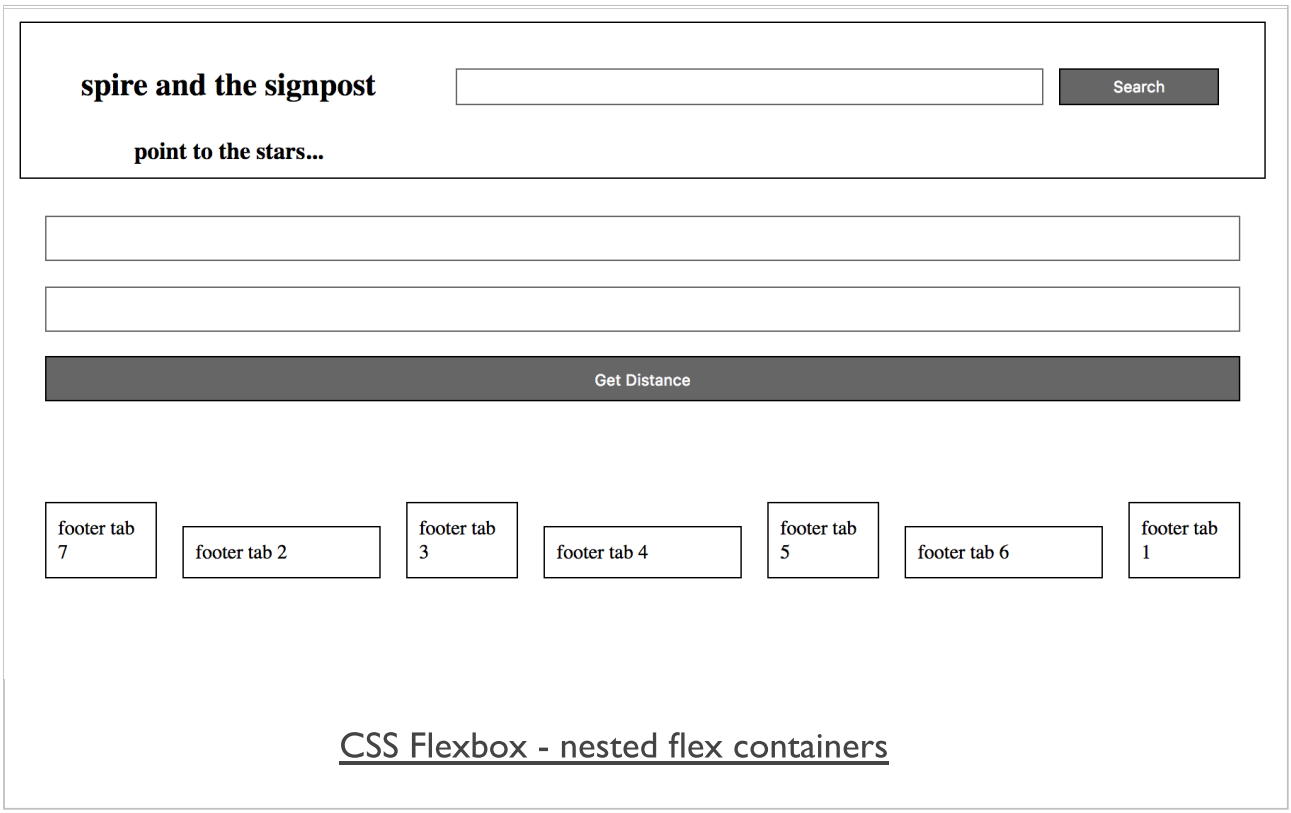
```
#searchBox {
  flex: 4;
}

#searchBtn {
  flex: 1;
}
```

- we get a simple proportional split of `4:1` for the input field to the button

Image - CSS Flexbox

nested flex containers



Demos

CSS - Grid

- grid basic - page zones and groups
- grid basic - article style page
- grid layout - articles with scroll
- grid layout - basic responsive
- grid layout - responsive with specific media query
- grid layout - responsive layout - part 1
- grid layout - responsive layout - part 2
- grid layout - dynamic content - part 1
- grid layout - dynamic content - part 2
- grid layout - Fun with Squares

Resources

- MDN - CSS3 Grid
- MDN - CSS Flexbox
- W3 Schools - CSS Grid View
- W3 Schools - CSS Flexbox