Notes - CSS - Flexbox

- Dr Nick Hayward

A general intro and outline for using flexbox with HTML5 compatible apps.

Contents

- intro
- basic usage
- axes
- flex direction
  - flex item wrapping
  - flex-flow shorthand
- sizing of flex items
  - minimum size
- flex item alignment
  - override align per flex item
  - justify content for flex item
- order flex items
- nesting flex containers and items

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...

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;
}
```

This CSS ruleset will define a `section` element as a parent flex container. Any child elements may now accept flex declarations.

This initial declaration, `display: flex`, also includes default values for flexbox layout of child elements.

e.g. `<div>` elements in a section will, by default, be arranged as equal sized columns with the same initial height.

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 being laid out inside the flex container is called a *flex item*.

flex direction

We can set a property for the flex direction, which defines direction of the flex items relative to the main axis. i.e. the layout direction for the child elements.

Default setting is `row`, and the direction will be relative to the current browser language setting. e.g. for English language browsers this will be left to right.

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

This will override the default `row` setting, and arrange the child items in a column.

So, we might define a default `section` element ruleset as follows,

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

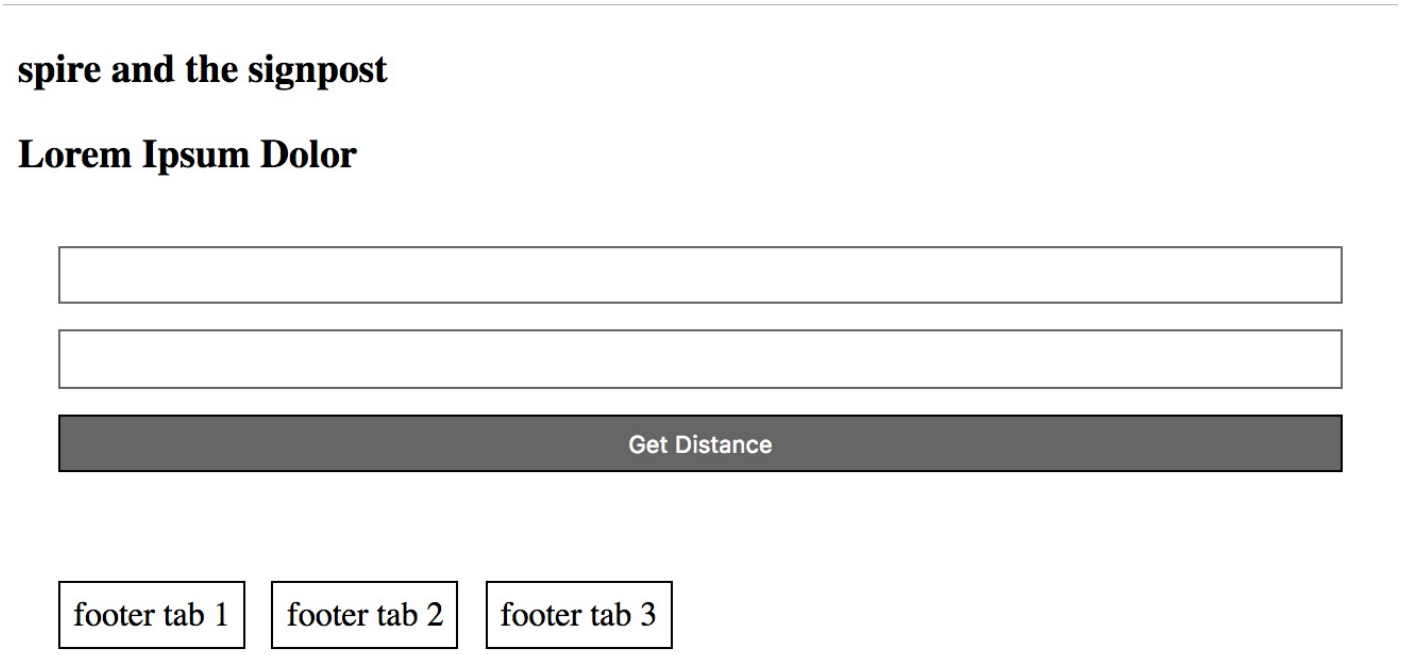
This would ensure that child flex items were 1aid out in a single column.

However, we might override specific `section` elements to allow child flex items in a `row` direction.

e.g.

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

Image - Flex direction



flex item wrapping

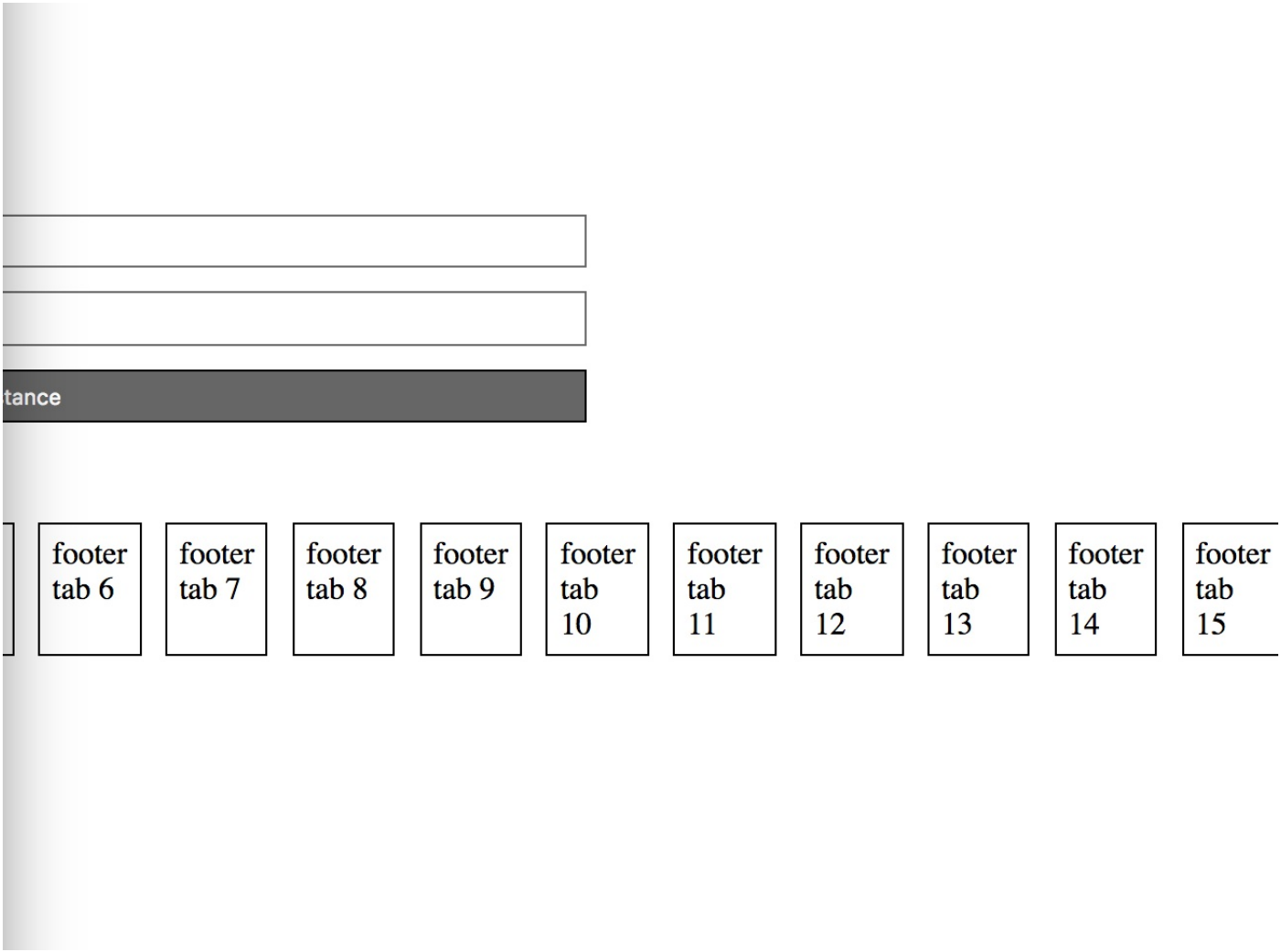
To ensure that child items do not overlap their parent flex container, we may add a declaration for `flex-wrap` to a required ruleset.

e.g.

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

So, without wrap

Image - No flex wrap



and, with wrap

Image - Flex wrap

spire and the signpost

Lorem Ipsum Dolor

Get Distance

footer tab 1 footer tab 2 footer tab 3 footer tab 4 footer tab 5 footer tab 6

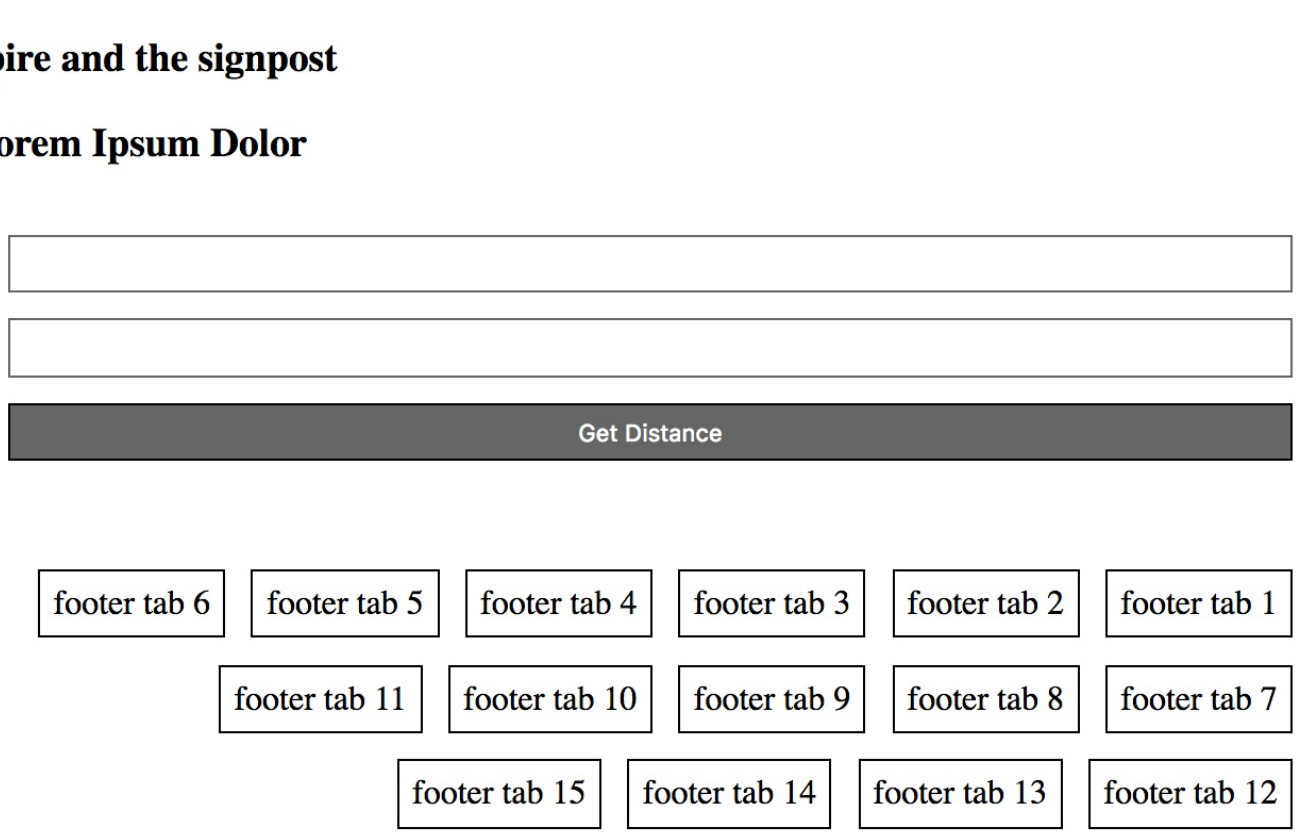
footer tab 7 footer tab 8 footer tab 9 footer tab 10 footer tab 11

footer tab 12 footer tab 13 footer tab 14 footer tab 15

We might also set the flex direction to reverse, which will start the flex items from the right on an English language browser.

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

Image - Flex direction reverse



`flex-flow` shorthand

We may also combine *direction* and *wrap* into a single declaration, `flex-flow`, which will now contain values for both `row` and `wrap`.

e.g.

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

sizing of flex items

For each flex item, we may need to specify apportioned space in the layout.

If we wanted to set space as an equal proportion for each flex item, we may add the following to a child item ruleset,

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

This defines each child flex item `<div class="fTab">` to occupy an equal amount of space, after considering margin and padding, within the given row.

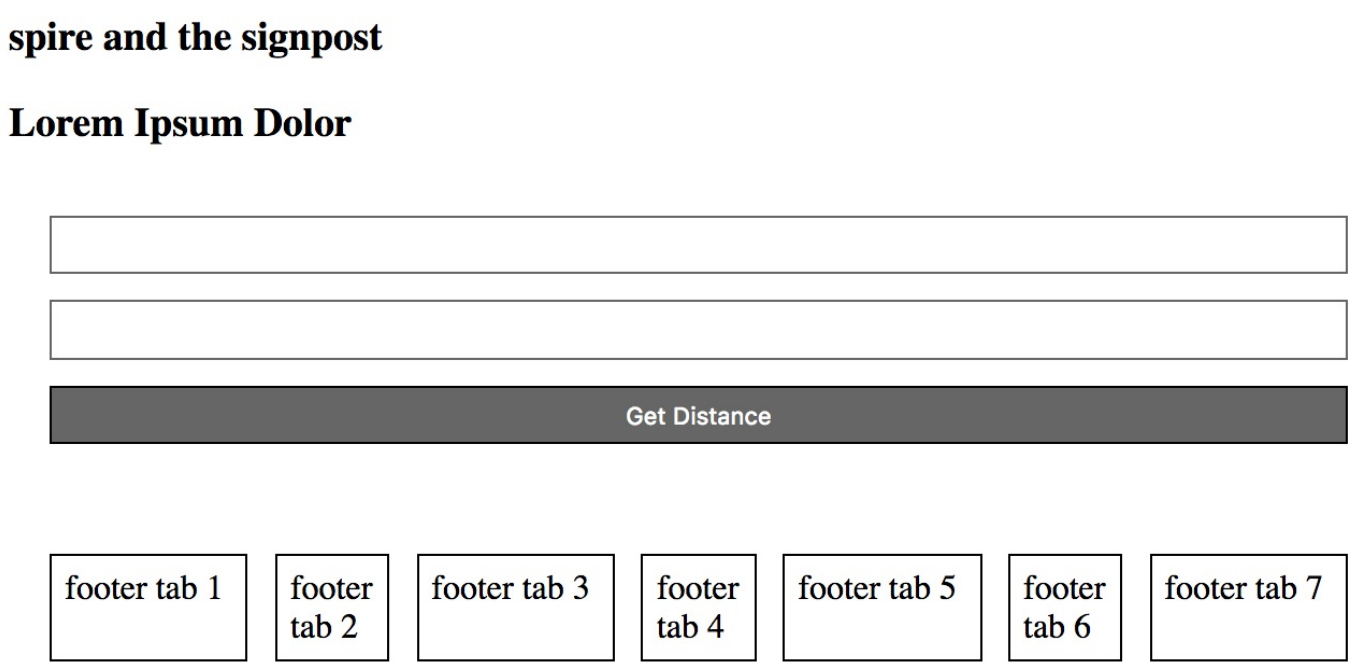
n.b. this value is proportional, so it doesn't matter if the value is 1 or 100 &c.

However, we may define additional flex proportions for specific child items. e.g.

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

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

Image - Flex item sizing



minimum size

We may then set a minimum size for a flex item, e.g.

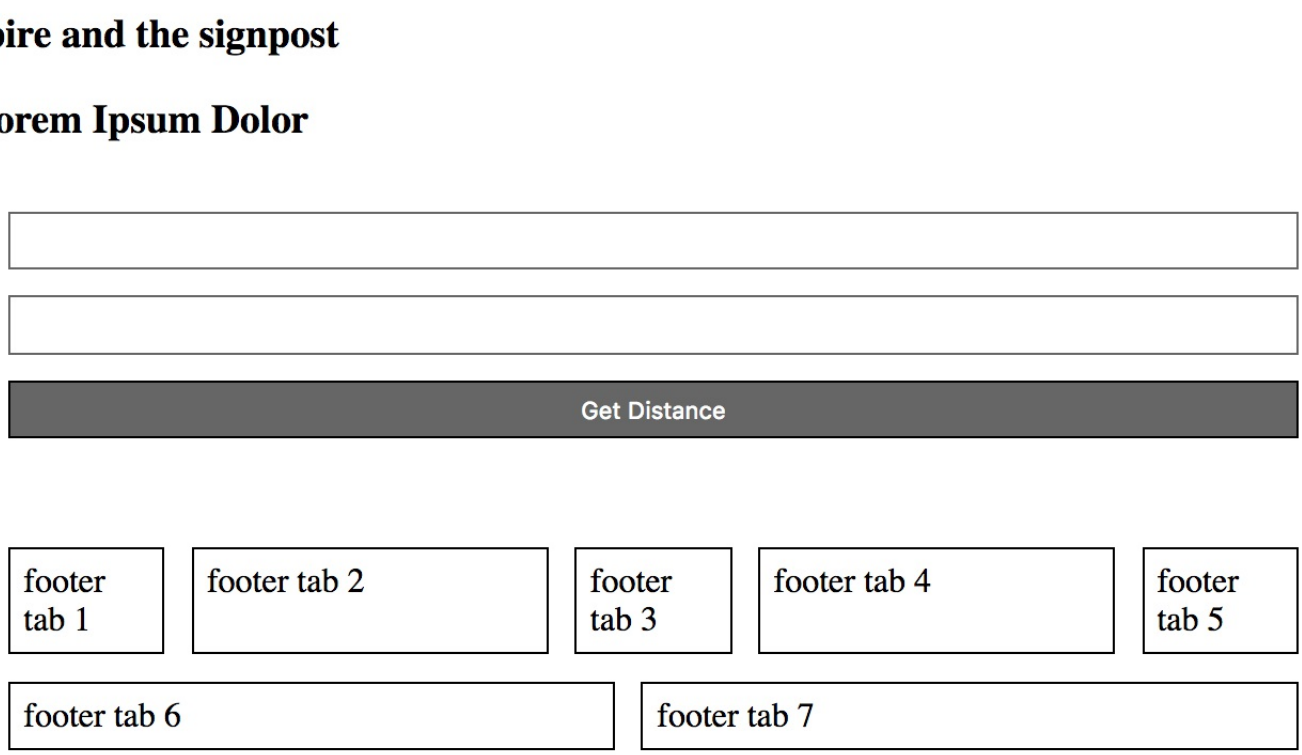
```
div.fTab {
  flex: 1 100px;
}
```

or a relative unit for the size,

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

This means each flex item will initially be given a minimum of `20%` of the available space. Then, the remaining space will be defined relative to proportion units.

Image - Flex item sizing



flex item alignment

Flexbox also allows us to define alignment for flex items in each flex container, relative to the main and cross axes.

For example, we might want to specify a centred alignment for flex items,

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

So, `align-items: center` will cause the flex item in the flex container to be centred along the cross axis. However, they'll still maintain their basic dimensions.

We can also modify the value for `align-items` to either `flex-start` or `flex-end`. As expected, such values will align flex items to either the start or end of the cross axis.

override align per flex item

As with `flex`, we can also override alignment per flex item. Using the `align-self` property, we may 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;
}
```

justify content for flex item

We may also specify `justify-content` for flex items in a flex container.

This property allows us to define the position of a flex item relative to the main axis.

The default value is `flex-start`, and we 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...

alignment per axis

In effect, we may define alignment relative to each axis using a specific declaration.

For example, for the main we may use `justify-content` and for the cross axis we use `align-items`.

order flex items

We may also modify the layout order of flex items without directly changing the underlying source order.

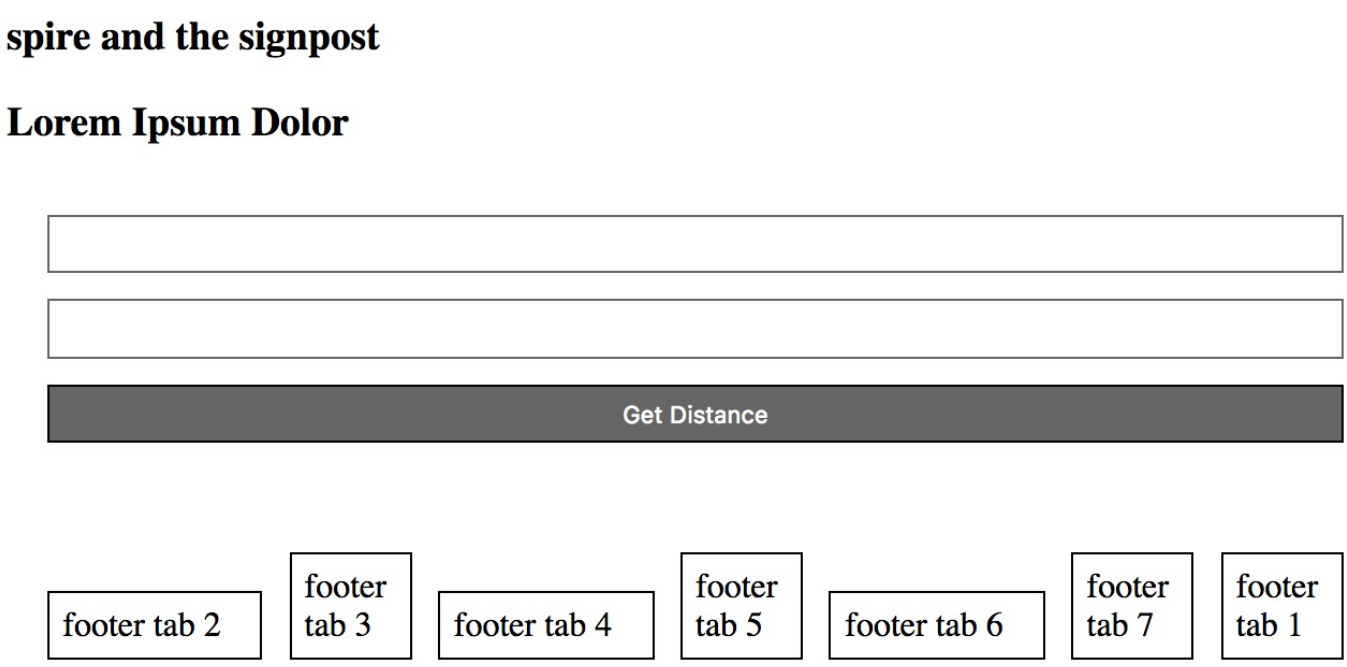
n.b. yet another modification we can't do with traditional CSS layout options...

We use the following pattern to specify order,

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

So, the first flex item will now move to the end of the tab list.

Image - Flex item order



This is due to the default order for flex items. By default, all flex items have an `order` value set to `0`.

So, the higher the `order` value, the later the item will appear in the list &c.

Items with the same order will revert to the order in the source code.

It's also possible to ensure that 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;
}
```

nesting flex containers and items

Flexbox can also be used to create nested patterns and structures.

For example, we may set a flex item as a flex container for its child nodes.

e.g. 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>
```

For this HTML, we may set `#banner`, `#page-header`, and `#search` as flex containers. e.g.

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

We may then specify various declarations for `#search`, e.g.

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

which will include values for itself and any child elements.

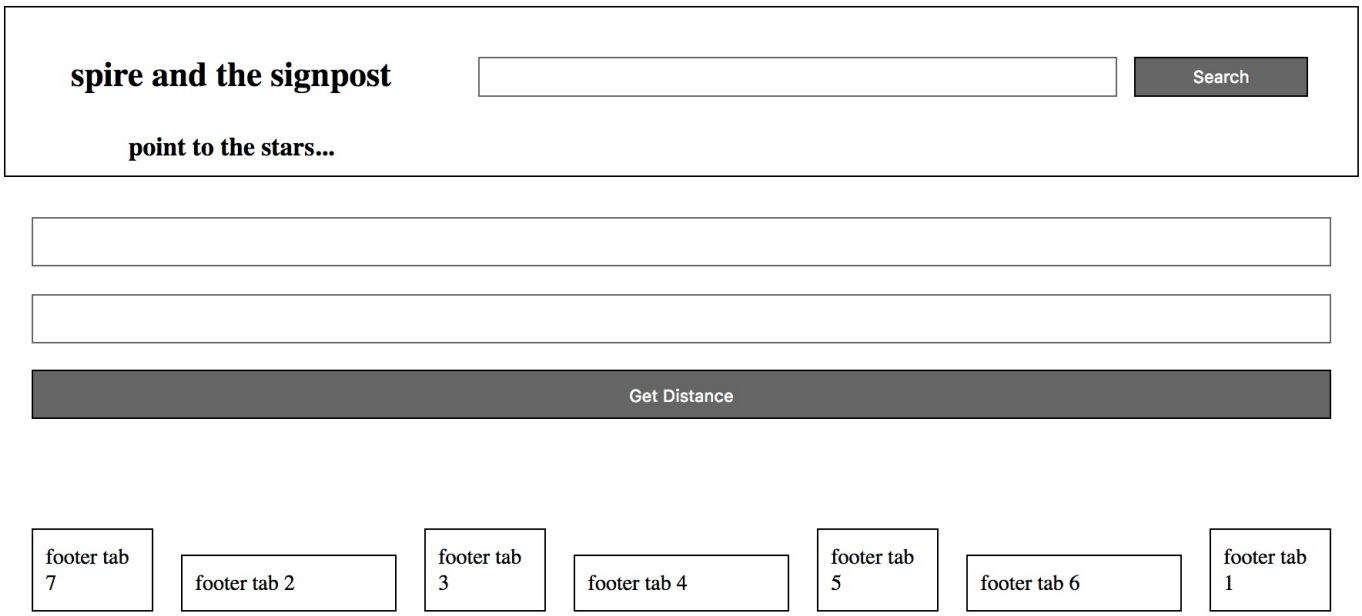
So, if we then add some rulesets for the nested flex items, e.g.

```
#searchBox {
  flex: 4;
}

#searchBtn {
  flex: 1;
}
```

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

Image - Nested flex containers



Reference

- MDN - CSS FLexbox