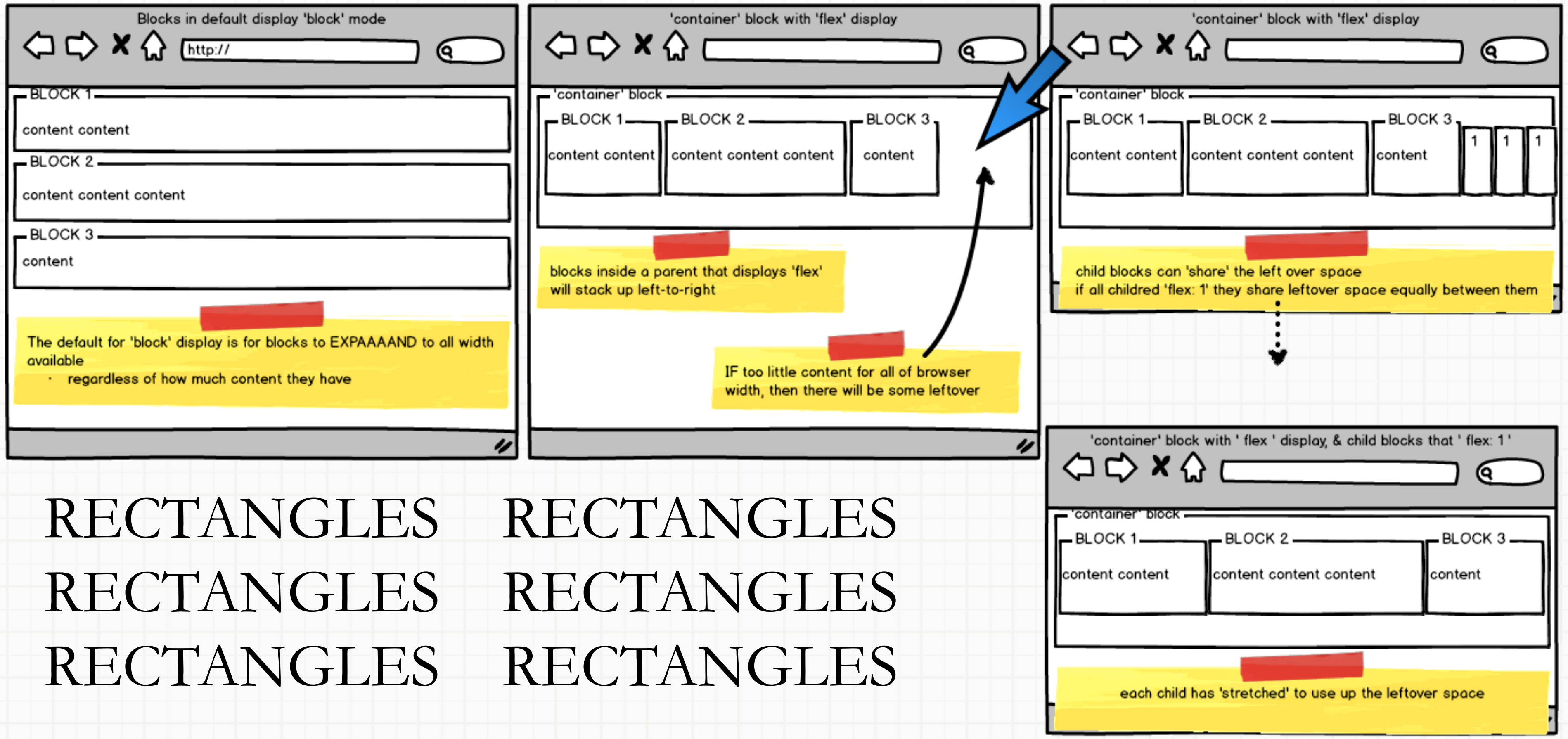
Flex Box Cont

Marie Brennan

Web Development Introduction



The default for 'block' display is for blocks to EXPAAAAND to all width available
- regardless of how much content they have

blocks inside a parent that displays 'flex' will stack up left-to-right

IF too little content for all of browser width, then there will be some leftover

child blocks can 'share' the left over space
if all childred 'flex: 1' they share leftover space equally between them

each child has 'stretched' to use up the leftover space

RECTANGLES RECTANGLES
RECTANGLES RECTANGLES
RECTANGLES RECTANGLES

(a.k.a. layouts with CSS Flexible boxes)

```
display: flex
flex: 1 / 1 0 200px
flex-wrap: wrap | nowrap | wrap-reverse
flex-direction: row / column
align-items: center;
```

- For navbars, multi-column layouts, ‘gallery’ pages
  - Straightforward & consistent approach to layouts

Flexible boxes are W3C ‘candidate recommendation’

- the topics in this lecture are highly likely to become part of the next version of CSS
  - But there is a small chance the odd term may change
  - So (as with any part of computing) you need to keep up with web new standards as they are published

- Sources of flexible box information:

w3c candidate recommendation (2012)

dev.w3.org/csswg/css-flexbox/

w3c EDITOR’S draft (2013)

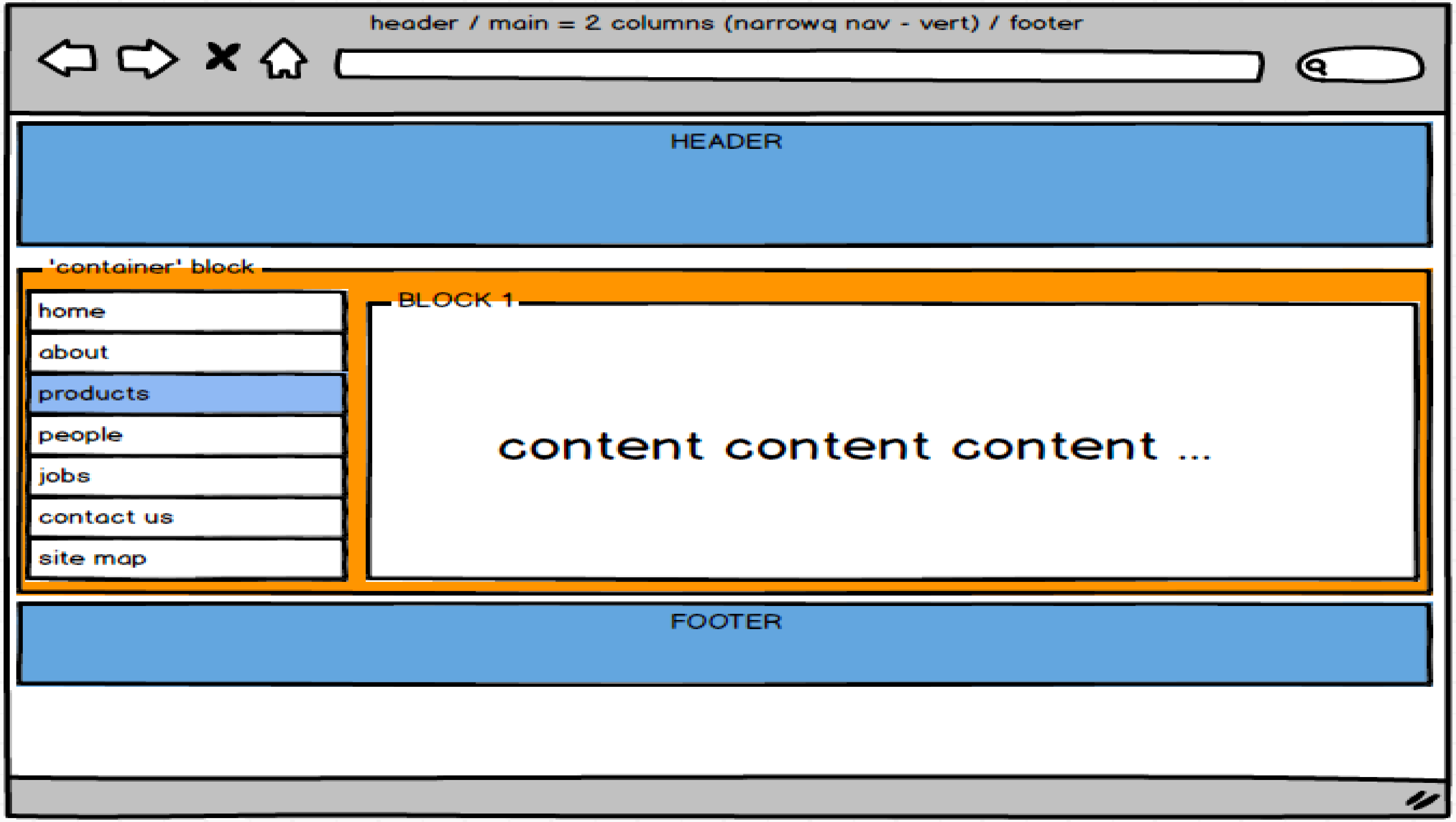
dev.w3.org/csswg/css-flexbox/

CSS tricks website (2013)

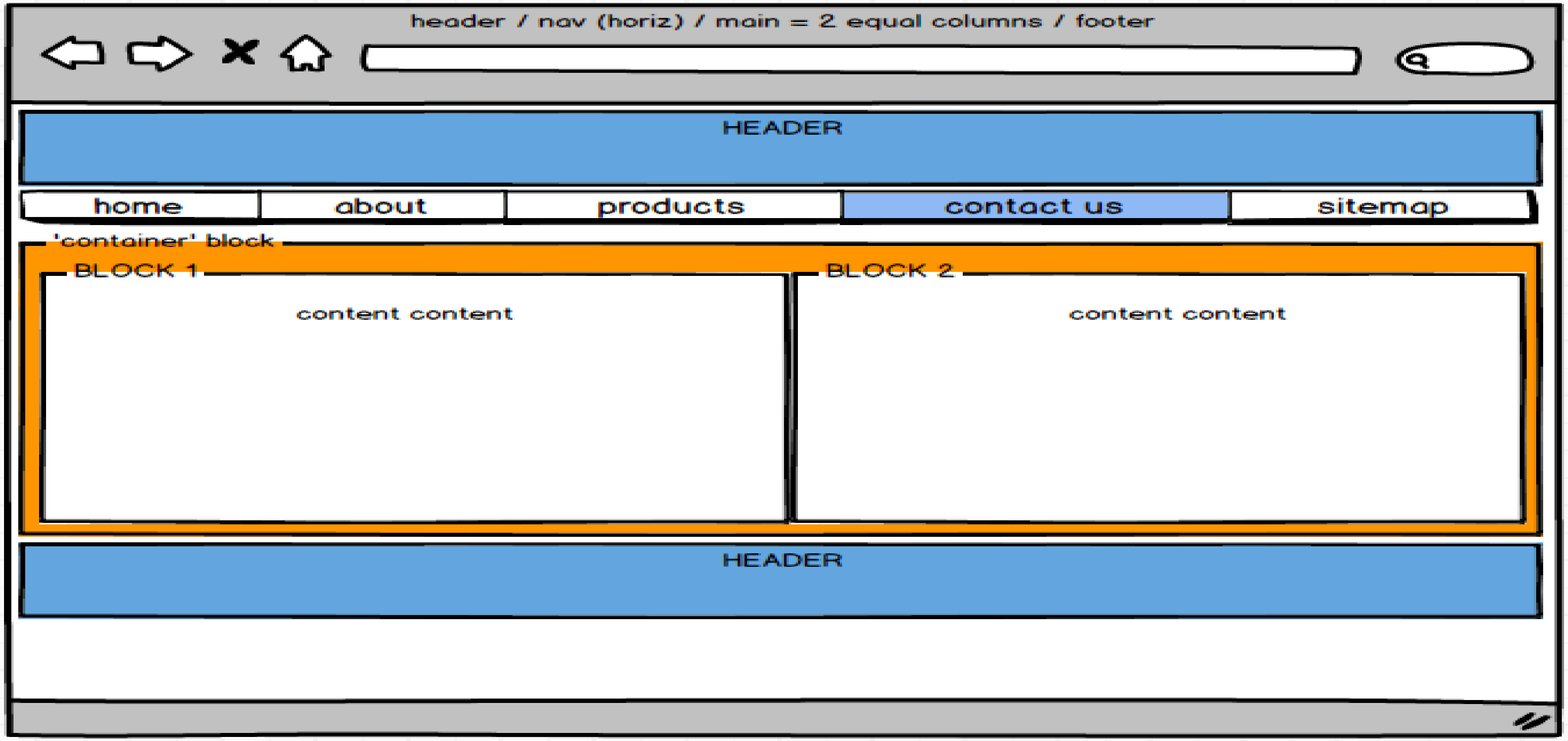
css-tricks.com/snippets/css/a-guide-to-flexbox/

Wouldn’t it be nice if …

- We could keep all the advantages of blocks
  - Margins and padding all 4 sides
  - They ‘fill’ available space
  - E.g. so background is for block, not just letters or words
- But we control whether blocks line up:
  - Like a row (left to right)
  - Like a column (top to bottom)
  - Or a mixture for complex page layouts …
- Flexible boxes allow us to CHANGE default layouts to allow control over page layouts …


header / main = 2 columns (narrowq nav - vert) / footer
HEADER
'container' block
home
about
products
people
jobs
contact us
site map
BLOCK 1
content content content ...
FOOTER

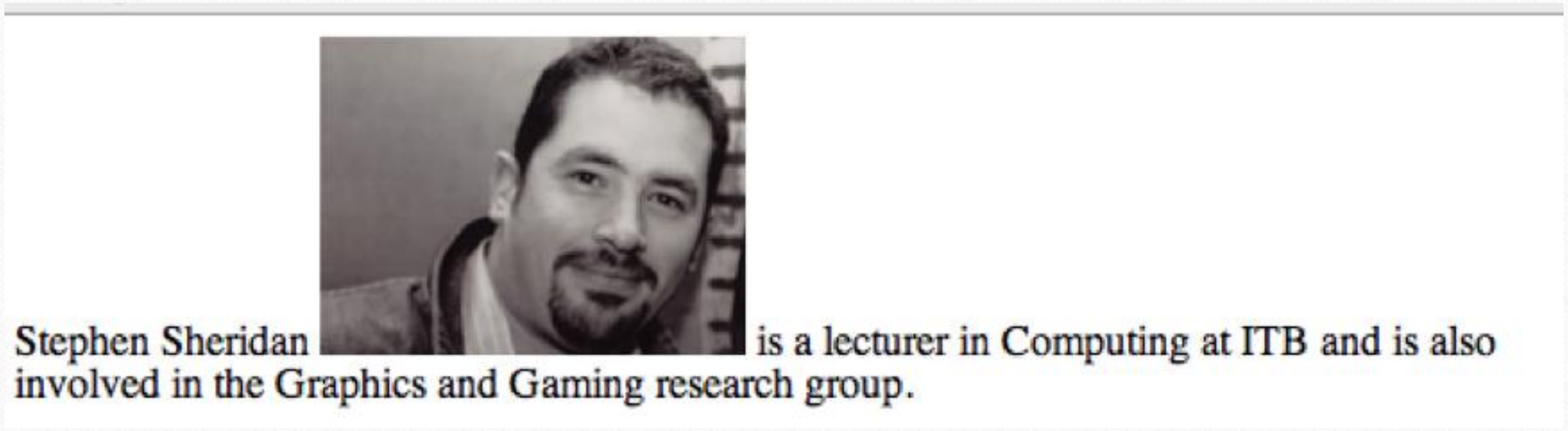
So will this …


header / nav (horiz) / main = 2 equal columns / footer
HEADER
home
about
products
contact us
sitemap
'container' block
BLOCK 1
content content
BLOCK 2
content content
HEADER

Recap:
default block/inline layout

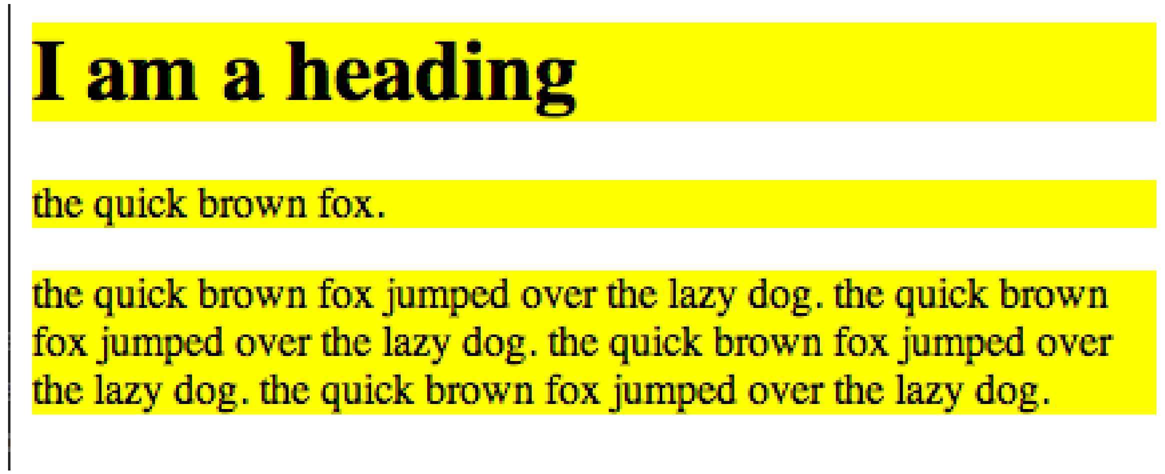
- Letters and images are 'inline' elements
  - They line up **left-to-right**
  - And wrap onto next line when right side reached

'inline' elements and their <u>default</u> layout


Stephen Sheridan is a lecturer in Computing at ITB and is also involved in the Graphics and Gaming research group.

‘block’ elements and their default layout

- Headings, paragraphs, list items etc. are ‘block’ elements
  - Every block starts on a NEW LINE
  - Blocks ‘expand’ to the full WIDTH available
    - Even if little content in block …



I am a heading

the quick brown fox.

the quick brown fox jumped over the lazy dog. the quick brown fox jumped over the lazy dog. the quick brown fox jumped over the lazy dog. the quick brown fox jumped over the lazy dog.

So blocks stack **top-to-bottom** in browser window

Before we worry about the CODE

Understand the RECTANGLES …

KISS (Keep It Simple Silly …)

- Make use of these DEFAULT layouts
  - Don’t add CSS style or extra HTML DIVs etc. if not needed !
  - So we’ll analyse rows then columns then rows etc.

- i.e. most typical page layouts naturally have several self-contained horizontal blocks
  - These fit within default block-level layout
  - And may not need any special layout CSS …
- I will demo examples at the end of the lecture

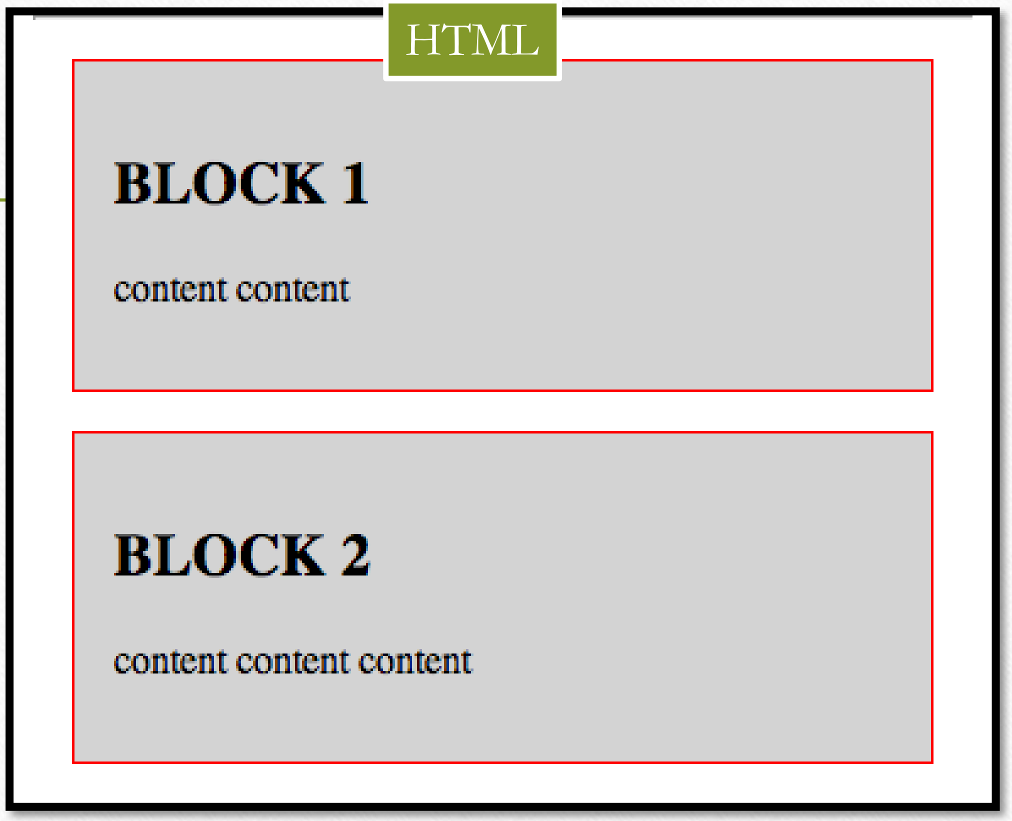
divide into simple horizontal sections

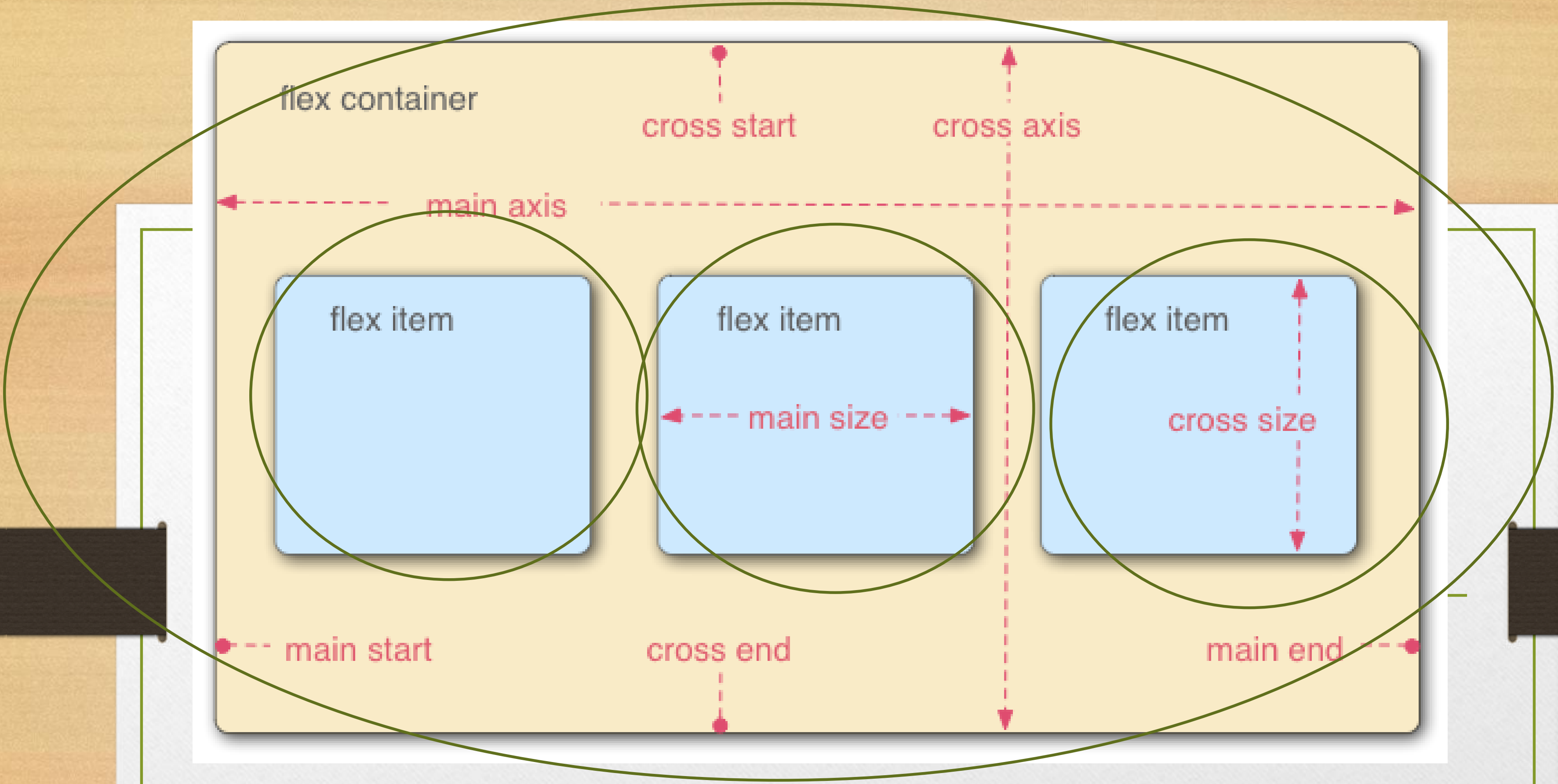
- each self-contained horizontal section should be a block
  - See how many horizontal lines you can draw
  - That run full width of page
- Usually we'll have some / all of the following
  - Header
  - Nav
  - Main, aside, section, article, section
  - Footer
  - NOTE – there may be MORE THAN one of each …

Write the code blocks_default_display

- Each section has a bit of style (red border etc.)
- So we can see its boundaries

```
<section>
    <h2>BLOCK 1</h2>
    <p>
        content content
    </p>
</section>
<main>
    <h2>BLOCK 2</h2>
    <p>
        content content content
    </p>
</main>
```

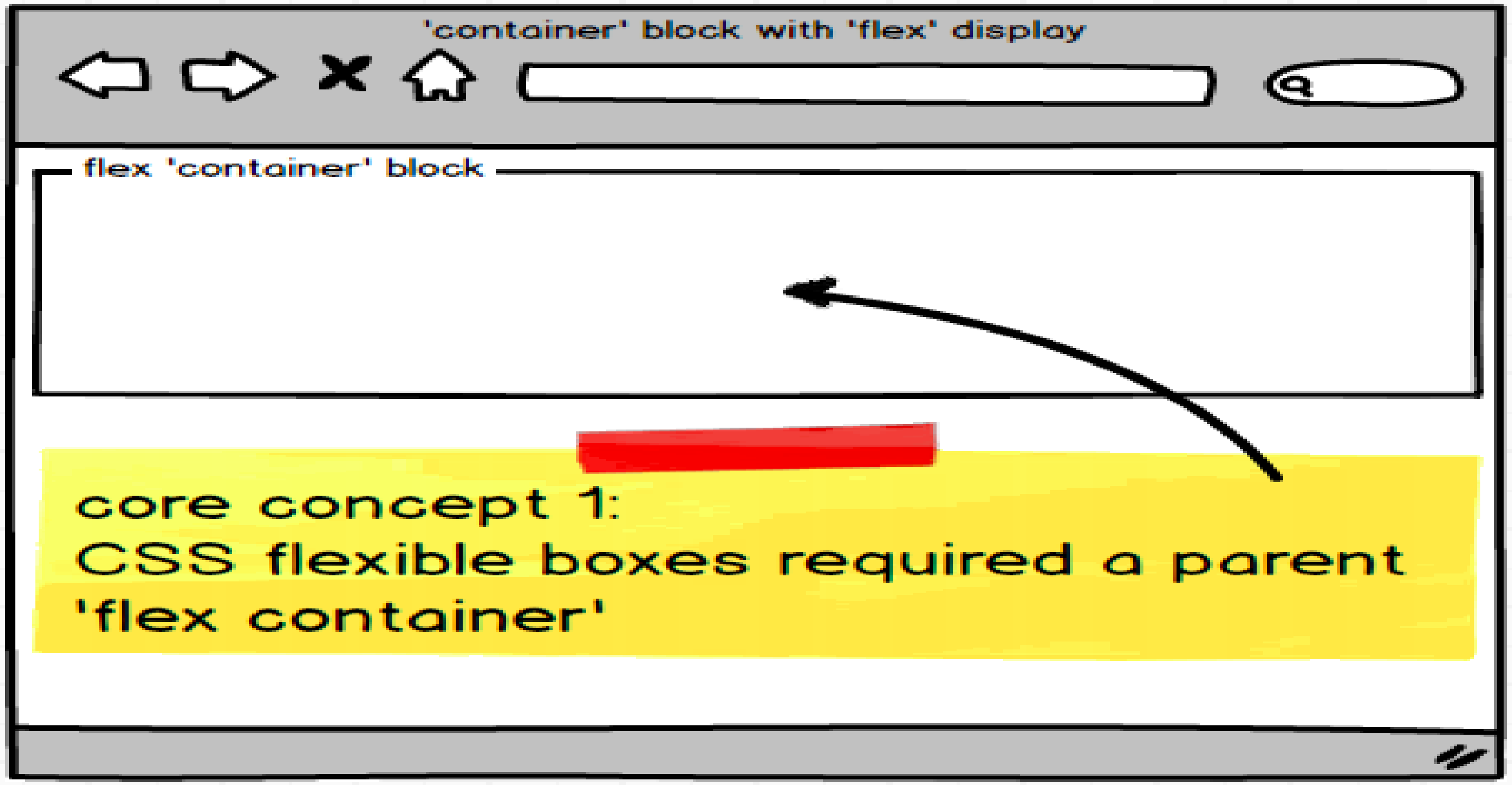




2 important core concepts:

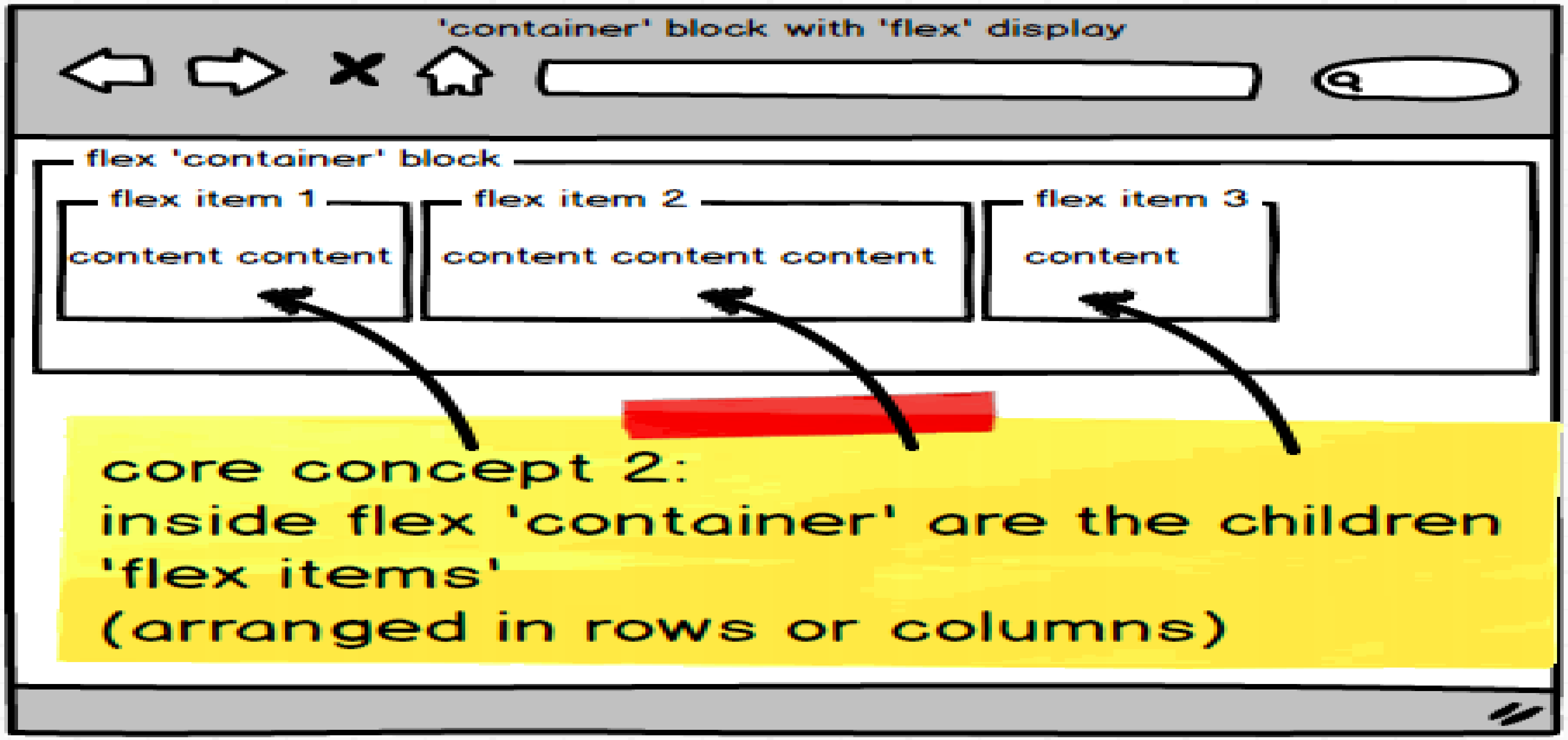
- Flex ‘container’ (parent block)
- Flex ‘item’ (child blocks)

The flex ‘container’ block
whose children can be arranged in rows or columns



core concept 1:
CSS flexible boxes required a parent 'flex container'

The flex 'items' - the children being laid out in the 'container'

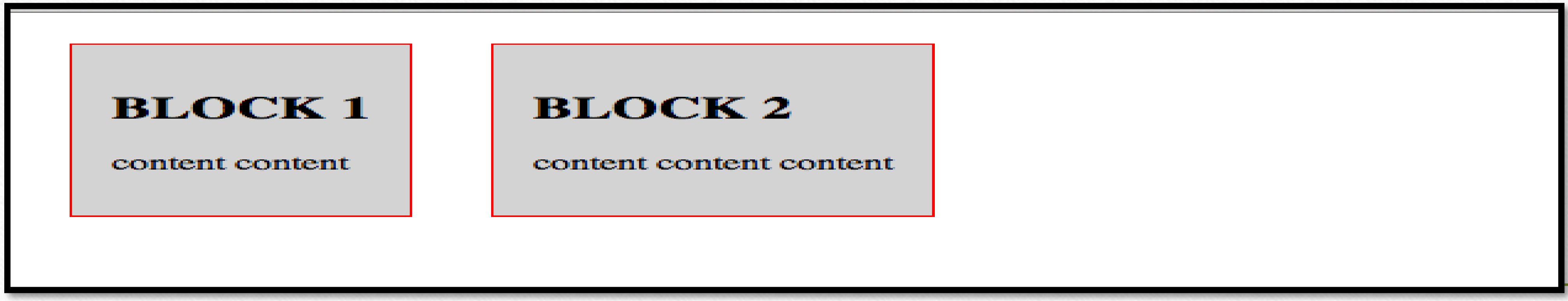


core concept 2:
inside flex 'container' are the children 'flex items'
(arranged in rows or columns)

We place the 2 SECTIONS inside a DIV parent given the ID = **`column_container`**

HTML

```
<div id="column_container">
   <section>
       <h2>BLOCK 1</h2>
       <p>content content
       </p>
   </section>
   <section>
       <h2>BLOCK 2</h2>
       <p>      content content content
       </p>
   </section>
</div>
```



BLOCK 1

content content

BLOCK 2

content content content

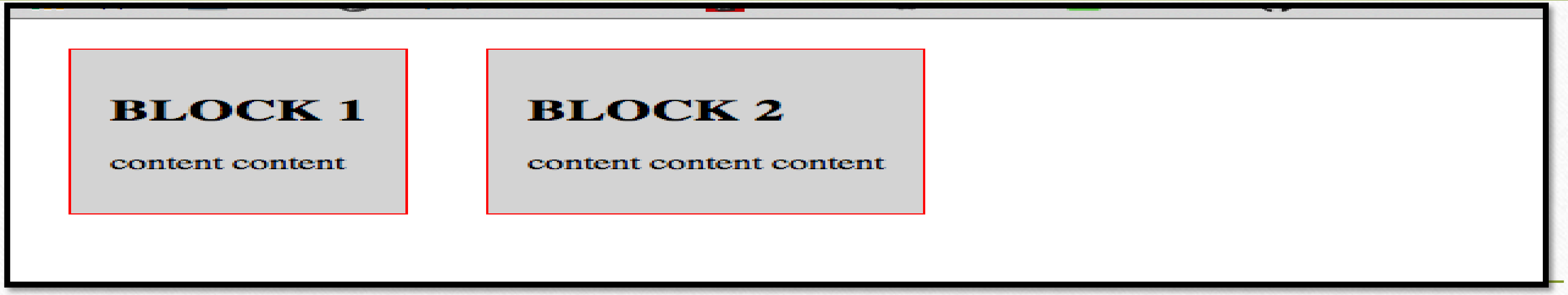
- The DIV has been styled as a ‘flex container’

```
display: flex;
```

- So the 2 SECTION children inside this DIV line up left-to-right (row layout is default for flex containers)

```
#column_container {
    -webkit-display: flex;
    display: flex;
}
```

CSS



BLOCK 1

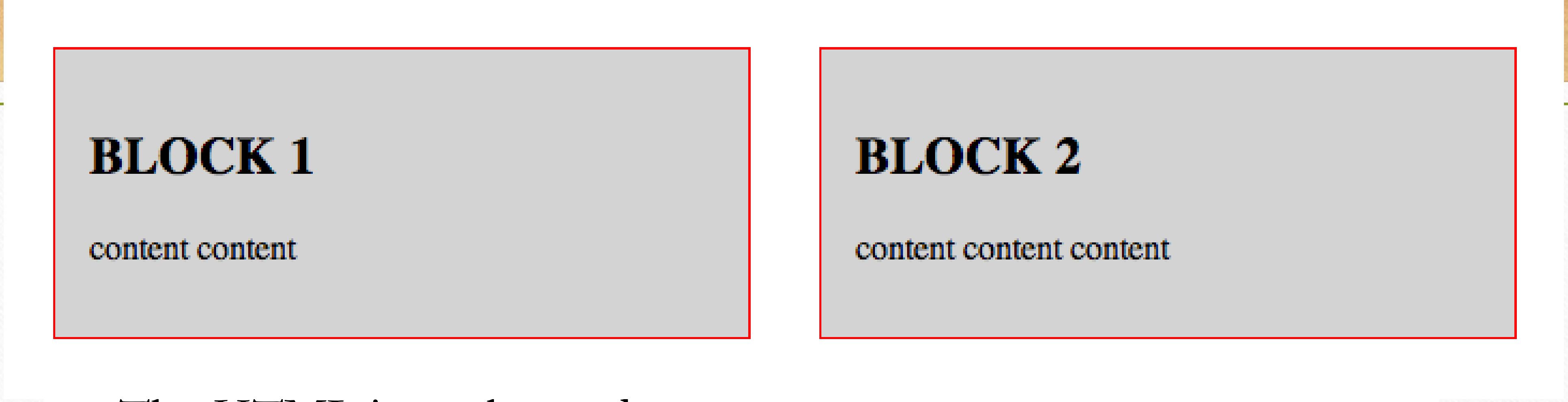
content content

BLOCK 2

content content content

- Note
  - There is spare space on the right
  - We can tell the children to ‘stretch’ to use up the spare space …

blocks_stretch_no_wrap



BLOCK 1

content content

BLOCK 2

content content content

- The HTML is unchanged
- Each section has been styled to equally stretch with any spare width

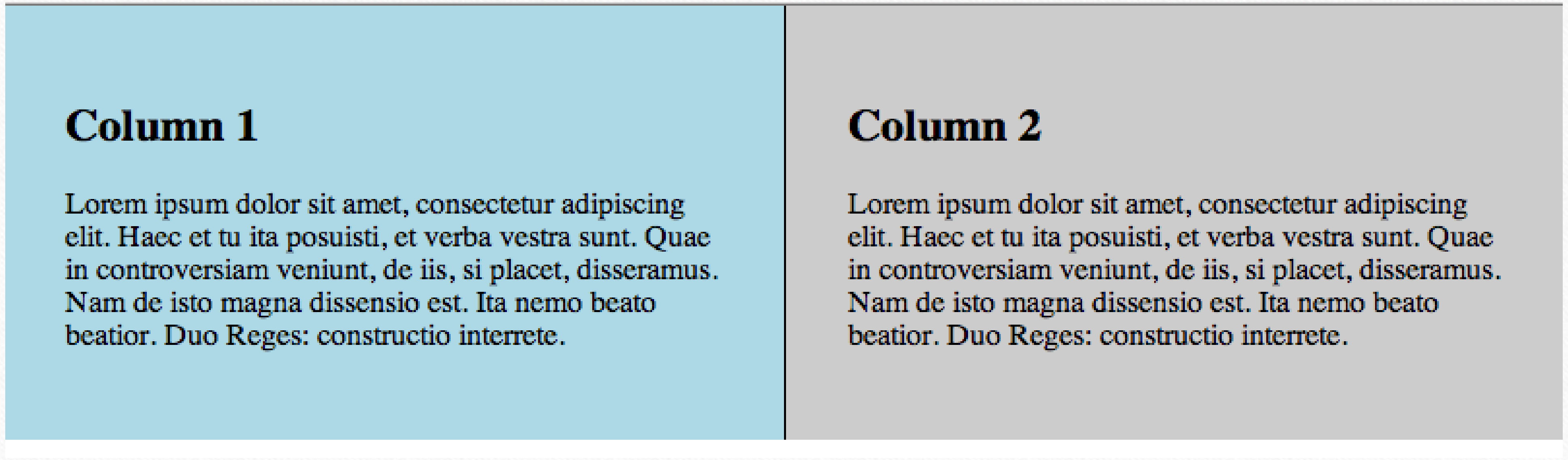
```
flex: 1;
```

- So the 2 SECTION children expand to be 2 equal columns

```
#column_container section {
    flex: 1;
    -webkit-flex: 1;
}
```

CSS

Here is same – but no margins on body or the sections – clean and simple 2 columns …



Column 1

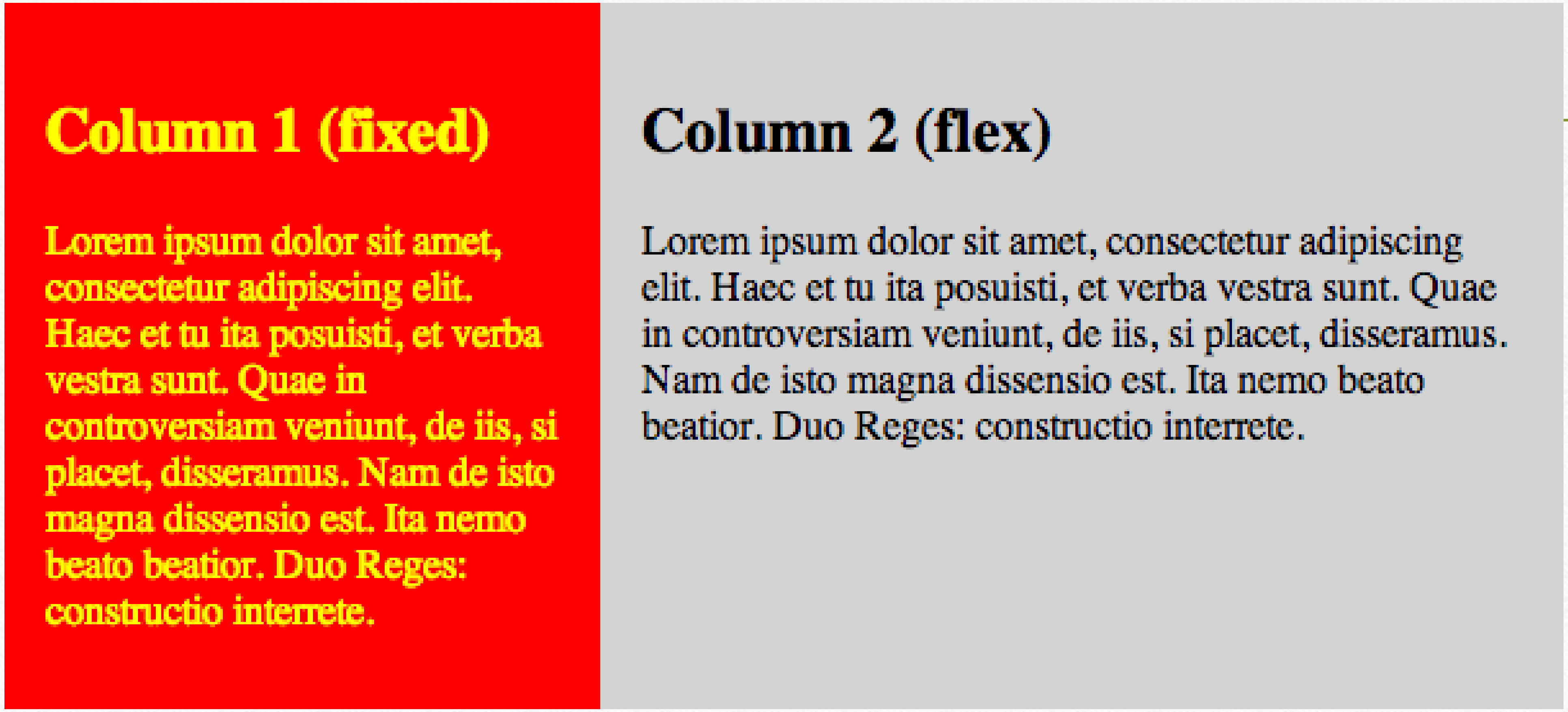
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Haec et tu ita posuisti, et verba vestra sunt. Quae in controversiam veniunt, de iis, si placet, disseramus. Nam de isto magna dissensio est. Ita nemo beato beatior. Duo Reges: constructio interrete.

Column 2

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Haec et tu ita posuisti, et verba vestra sunt. Quae in controversiam veniunt, de iis, si placet, disseramus. Nam de isto magna dissensio est. Ita nemo beato beatior. Duo Reges: constructio interrete.

one_fixed_one_flex

Left (red) column is FIXED width (width: 200px)
Right (grey) column flexes (flex: 1)



Column 1 (fixed)

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Haec et tu ita posuisti, et verba vestra sunt. Quae in controversiam veniunt, de iis, si placet, disseramus. Nam de isto magna dissensio est. Ita nemo beato beatior. Duo Reges: constructio interrete.

Column 2 (flex)

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Haec et tu ita posuisti, et verba vestra sunt. Quae in controversiam veniunt, de iis, si placet, disseramus. Nam de isto magna dissensio est. Ita nemo beato beatior. Duo Reges: constructio interrete.

```
<div id="column_container">
    <section class= "fixed">
       <h2>Column 1 (fixed)</h2>
       <p>
           content content
       </p>
    </section>

    <section class= "flex">
        <h2>Column 2 (flex)</h2>
      <p>
           content content content
       </p>
   </section>
</div>
```

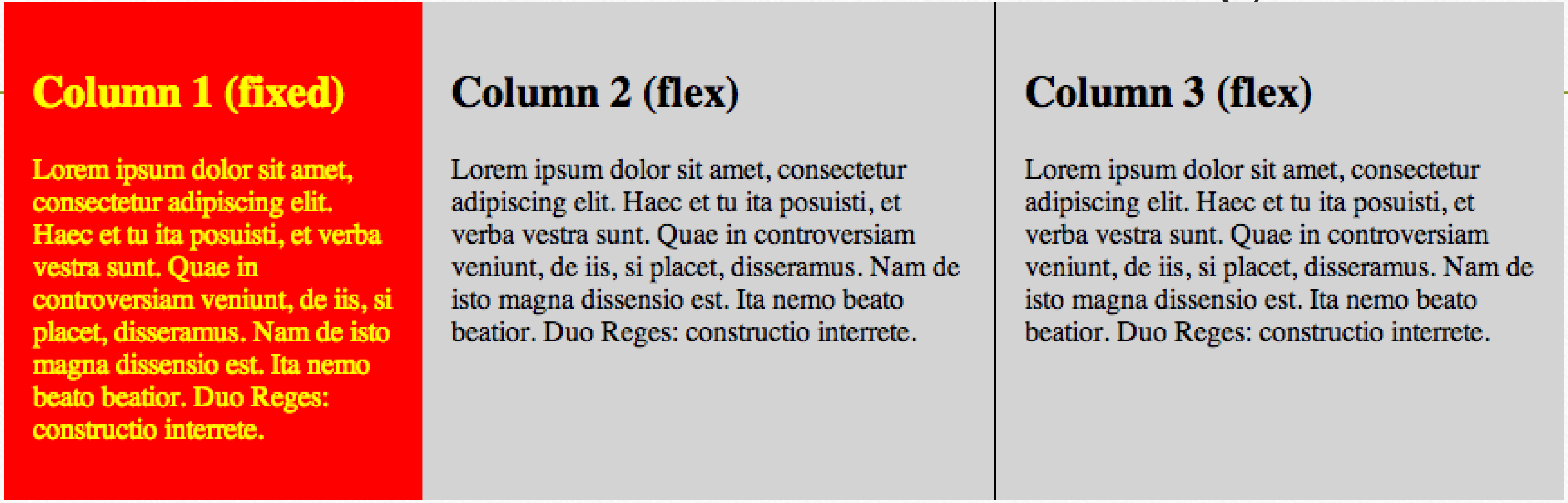
Each section given a class
– so can be styled differently

CSS

```
#column_container section.flex {
    -webkit-flex: 1;
    flex: 1;

    background-color: lightgray;
    padding: 1em;
}

#column_container section.fixed {
    width: 200px;
    background-color: red;
    color: yellow;
    padding: 1em;
}
```

Only one section class (right column) flexes (so it takes up all the spare width …)

Add another flex column – no change to CSS



Column 1 (fixed)

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Haec et tu ita posuisti, et verba vestra sunt. Quae in controversiam veniunt, de iis, si placet, disseramus. Nam de isto magna dissensio est. Ita nemo beato beatior. Duo Reges: constructio interrete.

Column 2 (flex)

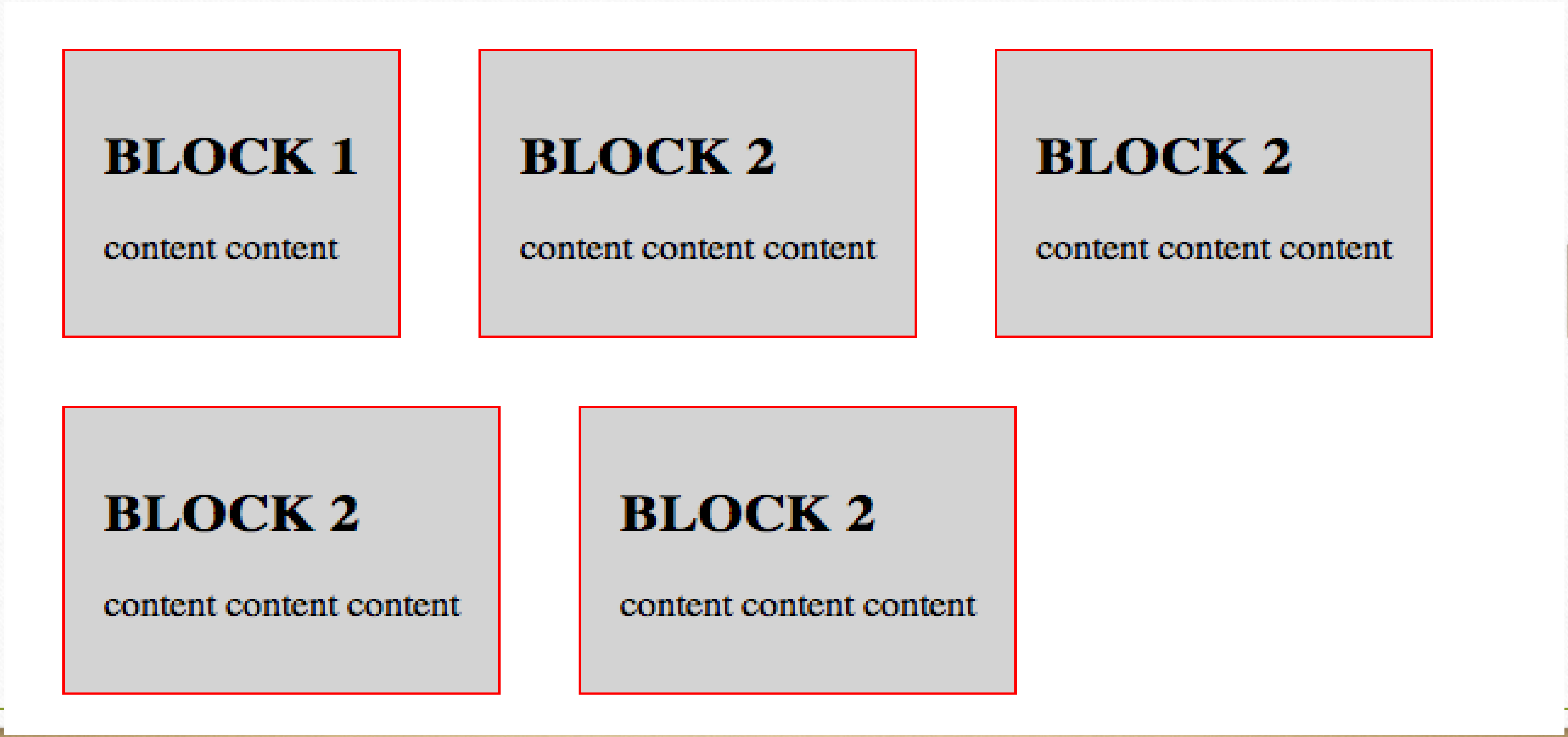
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Haec et tu ita posuisti, et verba vestra sunt. Quae in controversiam veniunt, de iis, si placet, disseramus. Nam de isto magna dissensio est. Ita nemo beato beatior. Duo Reges: constructio interrete.

Column 3 (flex)

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Haec et tu ita posuisti, et verba vestra sunt. Quae in controversiam veniunt, de iis, si placet, disseramus. Nam de isto magna dissensio est. Ita nemo beato beatior. Duo Reges: constructio interrete.

blocks_in_flex_parent_allowing_wrap

Style flex container (parent) to allowing wrapping



Required: style flex container to allow wrapping

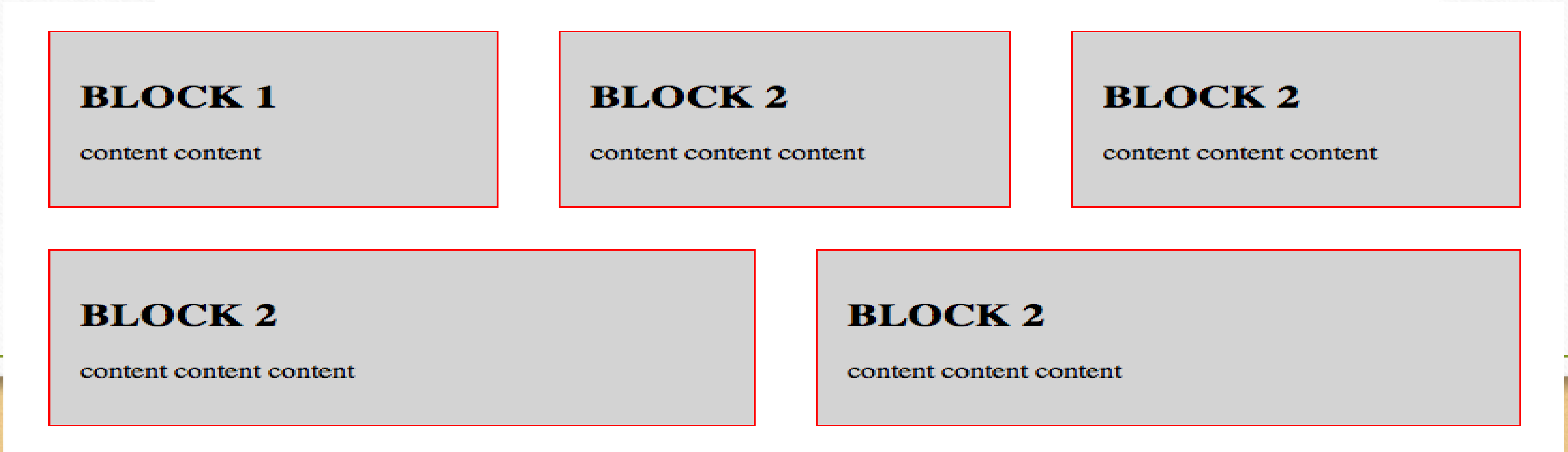
- Add code to flex container (parent) block, to allow wrapping

```
#column_container {
    -webkit-display: flex;
    display: flex;


    -webkit-flex-wrap: wrap;
    flex-wrap: wrap;
}
```

Usually desired: style flex items to stretch (and then wrap when a min-width occurs)

- Flex item (child) blocks to stretch: with third argument:
  - Minimum width to trigger wrapping …

```
#column_container section {
        -webkit-flex: 1 0 200px;
        flex: 1 0 200px;          }
}
```


BLOCK 1

content content

BLOCK 2

content content content

BLOCK 2

content content content

BLOCK 2

content content content

BLOCK 2

content content content

Breaking it down

```
.gallery-item {
  flex: 1 0 200px;
```

`flex-grow`	`flex-shrink`	`flex-basis`
give every item 1 share of extra width	don't let the items shrink at all (but they wouldn't anyway due to `flex-wrap`)	start them out at 200 pixels wide (basically, min-width)

CSS flexible boxes in a nutshell

- BLOCK level elements
  - Basics are just 2 steps

(1) Flex 'container'

```
display: flex;
```

(2) Flex 'items' (children of container)

- In most cases wish to Make items equal width
- (and 'stretch' to fill available width of flex container)

```
flex: 1;
```

Other issues
wrap items

- 2 steps for wrapping blocks for narrow widths:

(1) for flex container add:

```
flex-wrap: wrap;
```

(2) for flex item need to specify min-width to trigger wrap

```
flex: 1 0 200px;
```

Other issues

ITB older Chrome versions – need –webkit-

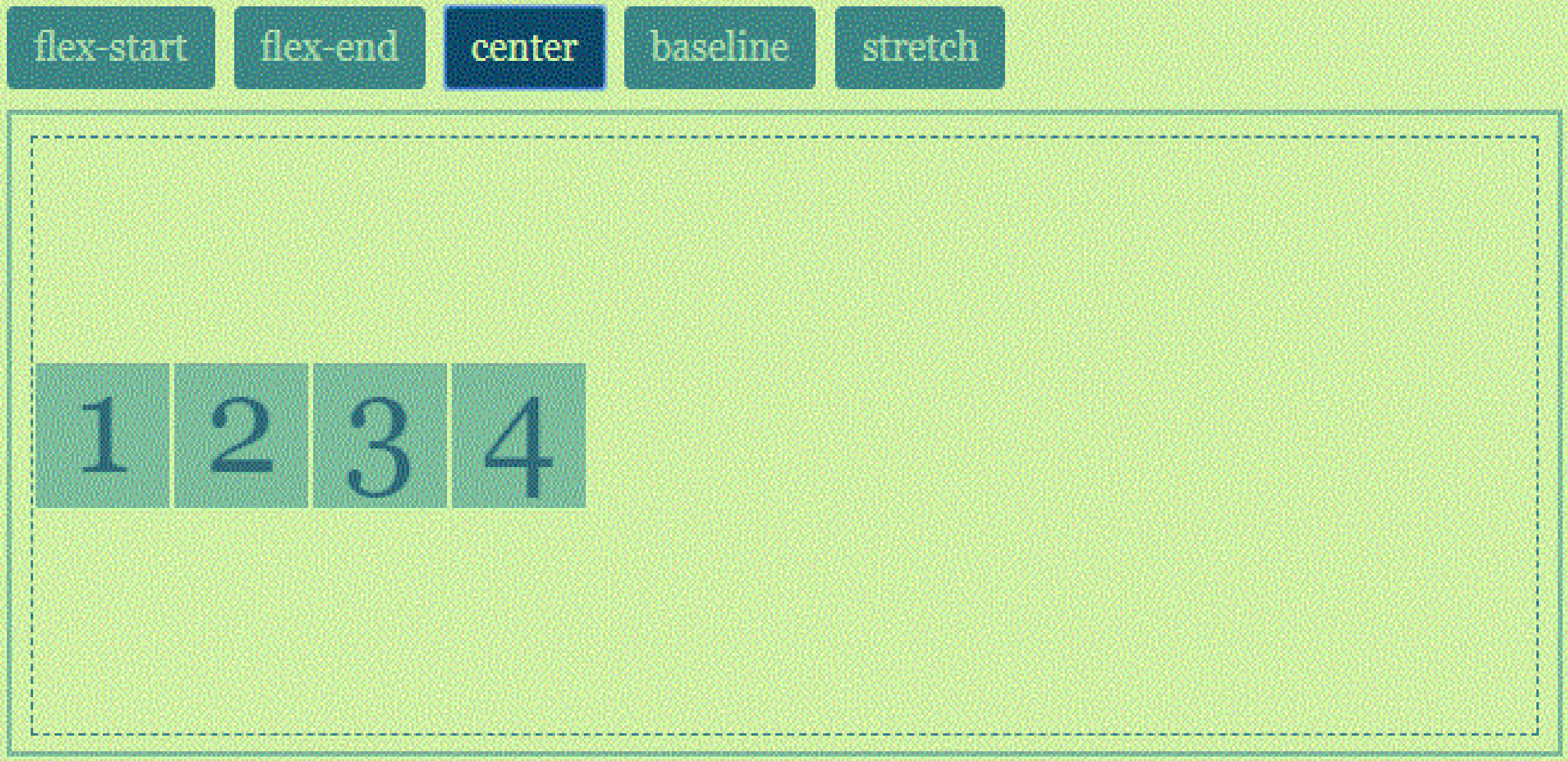
```
display: flex;
display: -webkit-flex;
flex-wrap: wrap;
-webkit-flex-wrap: wrap;
flex-direction: column;
-webkit-flex-direction: column;
align-items: center;
-webkit-align-items: center;

flex: 1;
-webkit-flex: 1;
flex: 1 0 10em;
-webkit-flex: 1 0 10em;
```

Flex <u>container</u> properties

Flex <u>item</u> properties

align-items

w3.org/TR/css-flexbox-1/#propdef-align-items

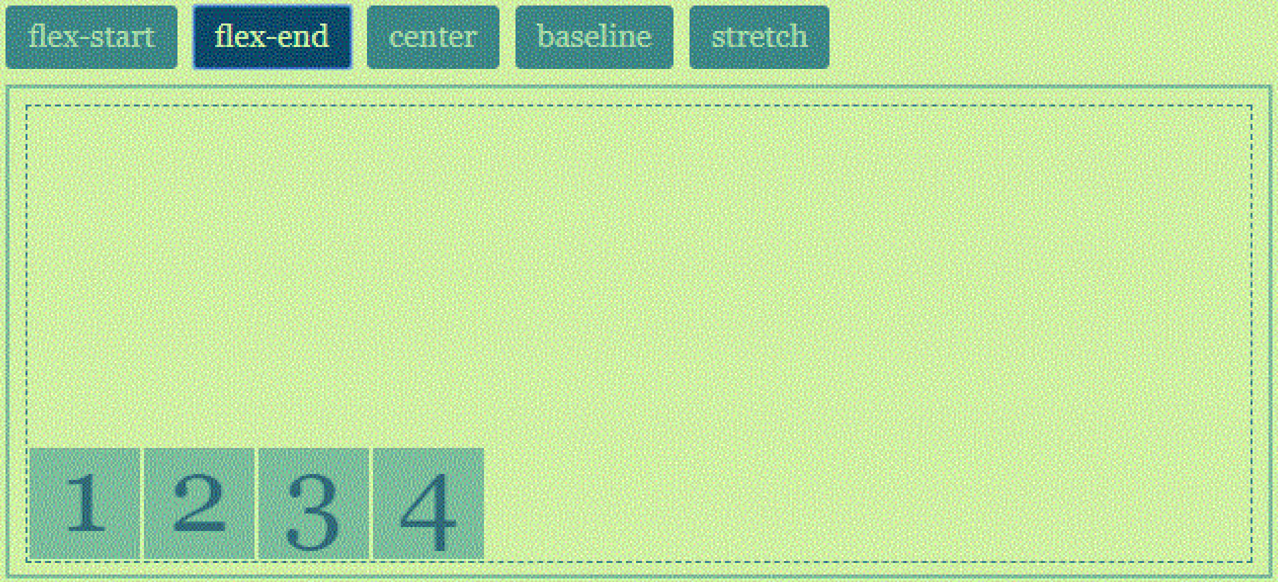


```
.parent {
  display: flex;
  align-items: center;
  height: 100%;
}
```

Flex items can be aligned in the *cross axis* of the current line of the flex container, similar to *justify-content* but in the perpendicular direction. *align-items* sets the default alignment for all of the flex container's *items*, including anonymous *flex items*. *align-self* allows this default alignmer to be overridden for individual *flex items*. (For anonymous flex items, *align-self* always matches the value of *align-items* on their associated flex container.)

align-items

w3.org/TR/css-flexbox-1/#propdef-align-items

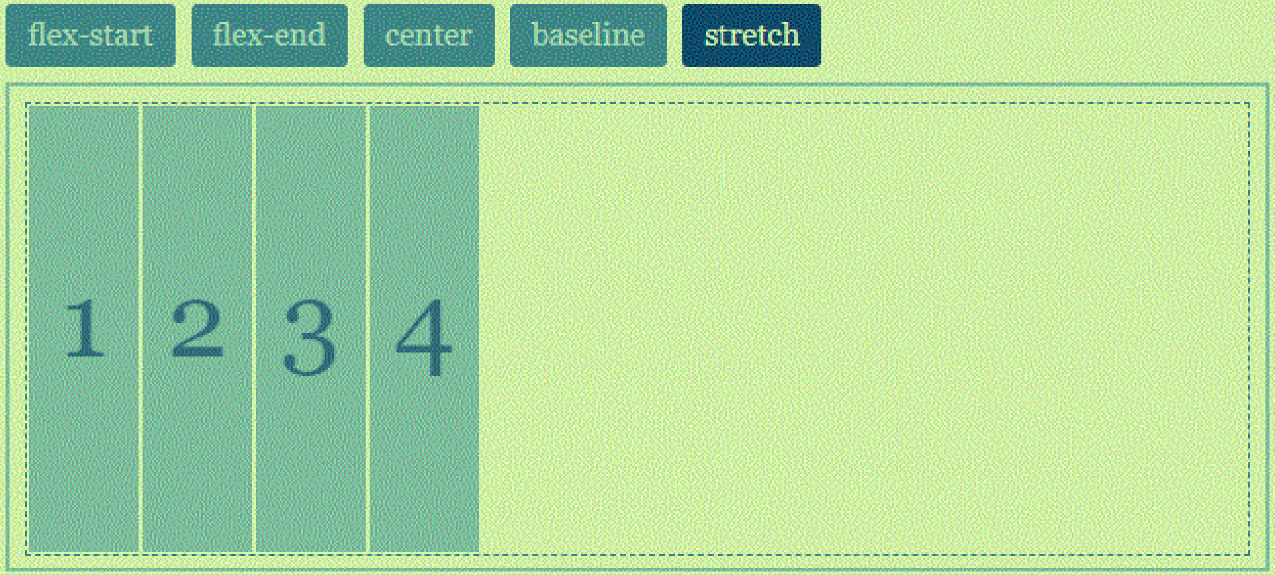


```
.parent {
  display: flex;
  align-items: flex-end;
  height: 100%;
}
```

Flex items can be aligned in the *cross axis* of the current line of the flex container, similar to *justify-content* but in the perpendicular direction. *align-items* sets the default alignment for all of the flex container's *items*, including anonymous *flex items*. *align-self* allows this default alignment to be overridden for individual *flex items*. (For anonymous flex items, *align-self* always matches the value of *align-items* on their associated flex container.)

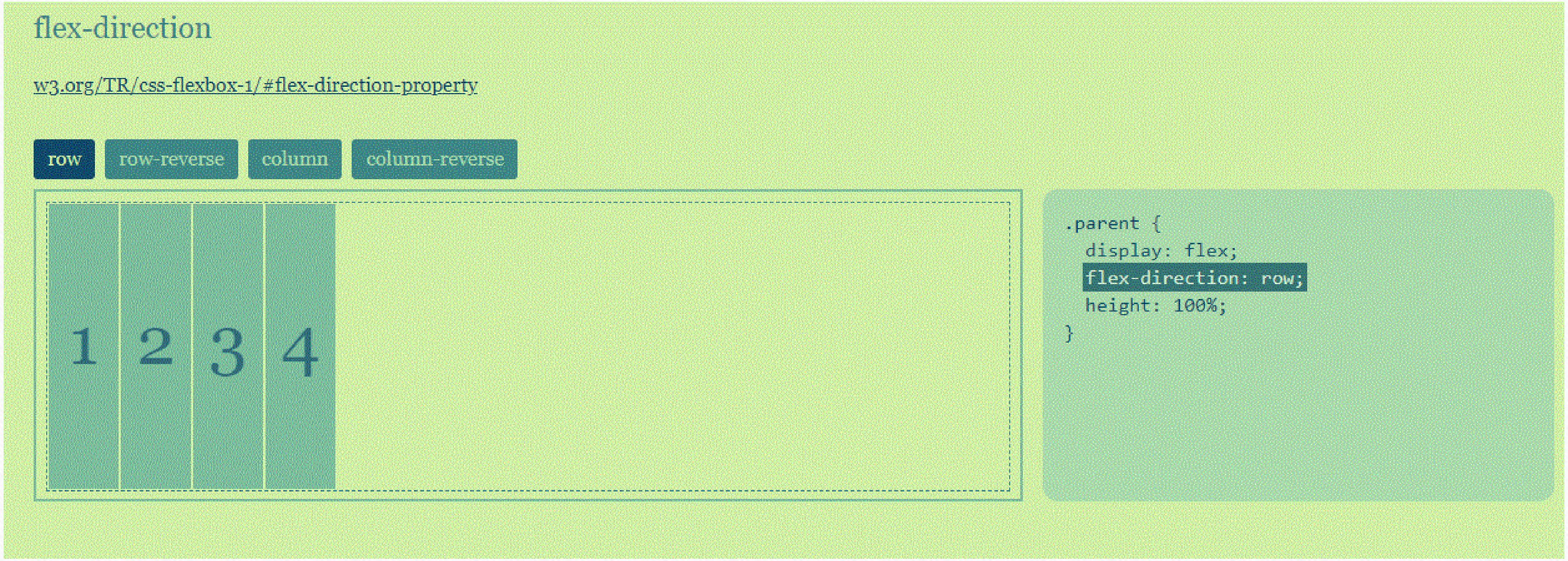
align-items

w3.org/TR/css-flexbox-1/#propdef-align-items



```
.parent {
  display: flex;
  align-items: stretch;
  height: 100%;
}
```

Flex items can be aligned in the *cross axis* of the current line of the flex container, similar to *justify-content* but in the perpendicular direction. *align-items* sets the default alignment for all of the flex container's *items*, including anonymous *flex items*. *align-self* allows this default alignment to be overridden for individual *flex items*. (For anonymous flex items, *align-self* always matches the value of *align-items* on their associated flex container.)



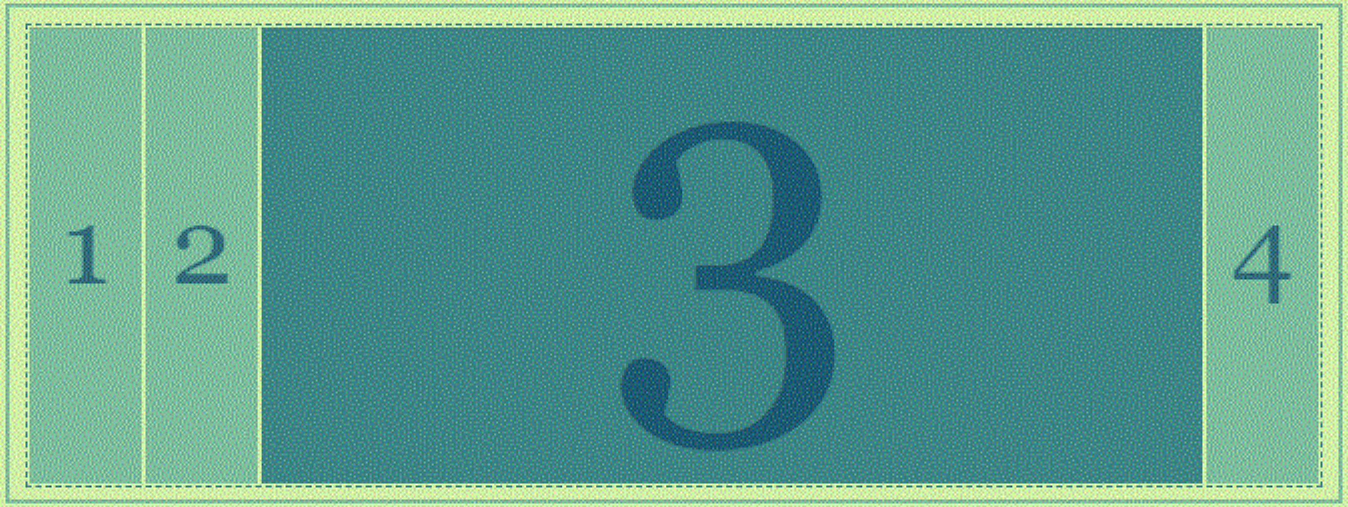
flex-direction

w3.org/TR/css-flexbox-1/#flex-direction-property

```
.parent {
  display: flex;
  flex-direction: row;
  height: 100%;
}
```

flex-grow

w3.org/TR/css-flexbox-1/#flex-grow-property

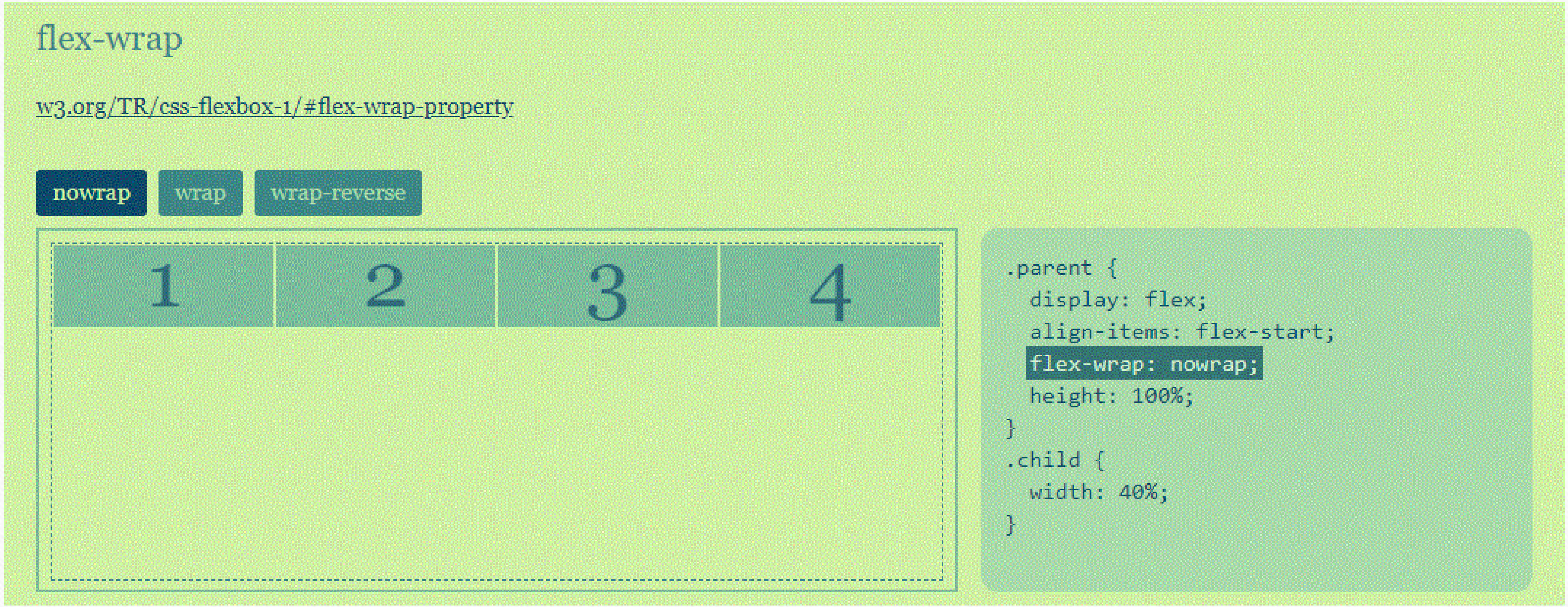


```
.parent {
  display: flex;
  height: 100%;
}
.child--featured {
  flex-grow: 1;
}
```

The *flex-grow* property sets the *flex grow factor* to the provided *<number>*. Negative numbers are invalid.

Applies to: flex items.

Initial: 0.



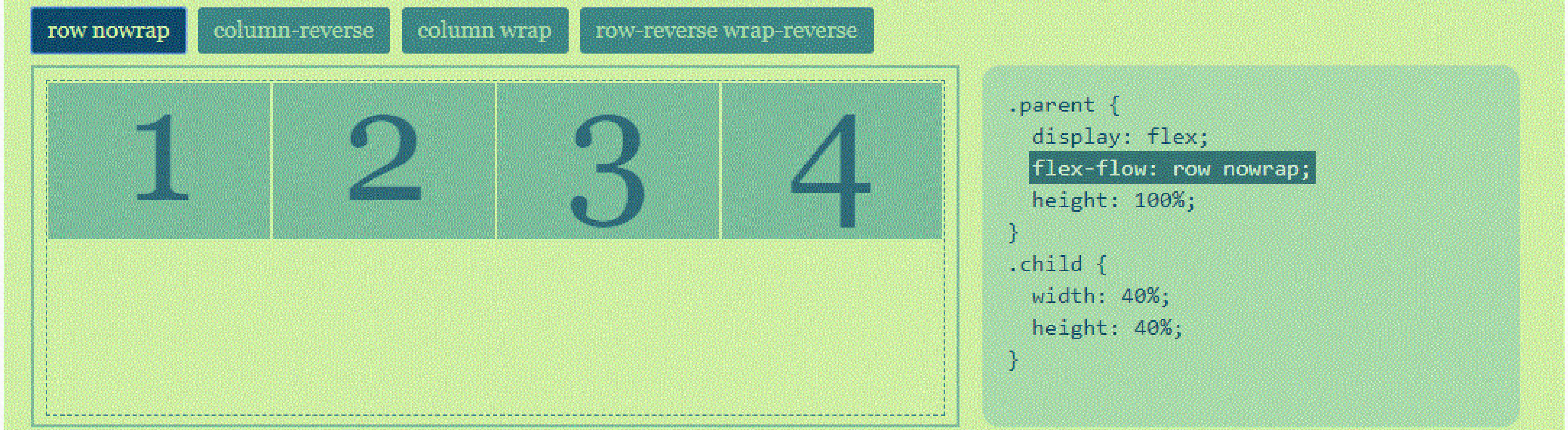
flex-wrap

w3.org/TR/css-flexbox-1/#flex-wrap-property

```
.parent {
  display: flex;
  align-items: flex-start;
  flex-wrap: nowrap;
  height: 100%;
}
.child {
  width: 40%;
}
```

flex-flow

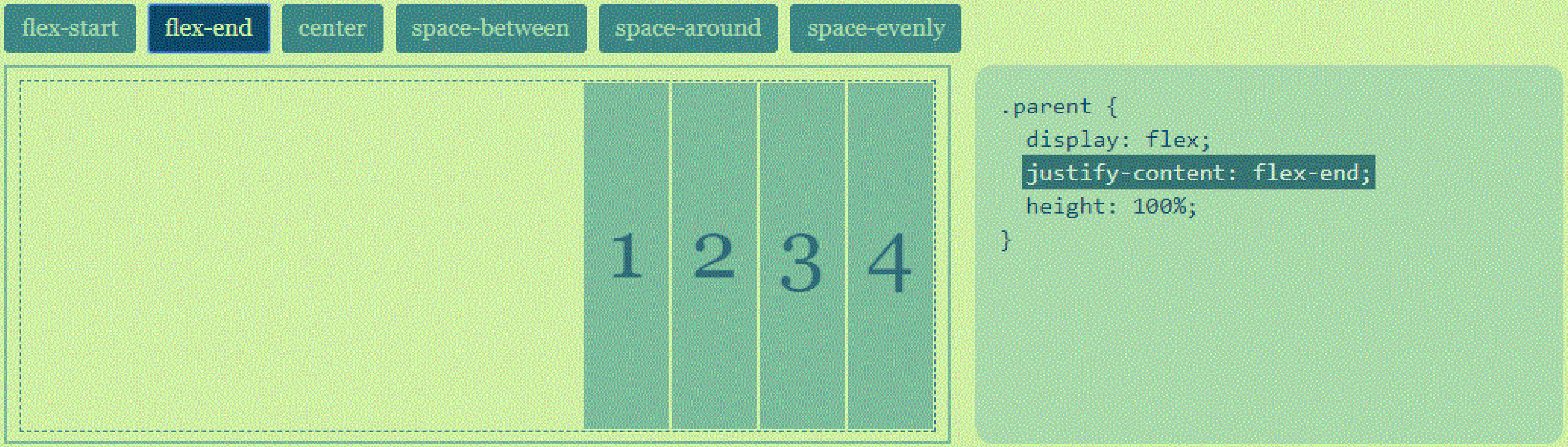
w3.org/TR/css-flexbox-1/#flex-flow-property



```
.parent {
  display: flex;
  flex-flow: row nowrap;
  height: 100%;
}
.child {
  width: 40%;
  height: 40%;
}
```

justify-content

w3.org/TR/css-flexbox-1/#justify-content-property

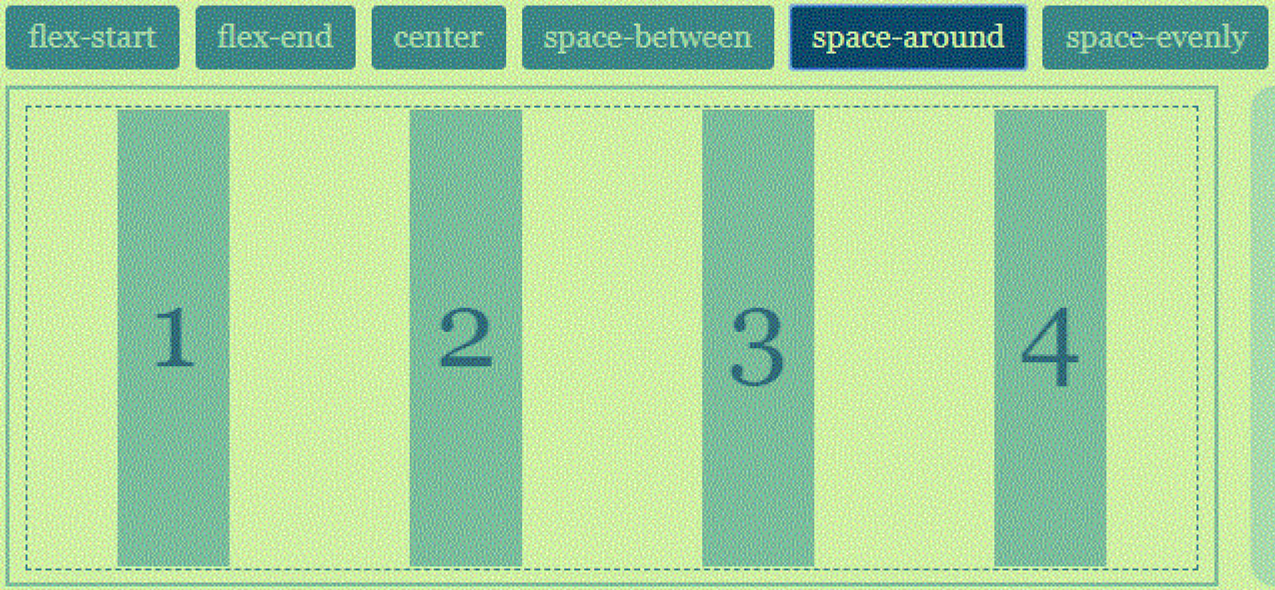


```
.parent {
  display: flex;
  justify-content: flex-end;
  height: 100%;
}
```

The *justify-content* property aligns *flex items* along the *main axis* of the current line of the flex container. This is done *after* any flexible lengths and any *auto margins* have been resolved. Typically it helps distribute extra free space leftover when either all the *flex items* on a line are inflexible, or are flexible but have reached their maximum size. It also exerts some control over the alignment of items when they overflow the line.

justify-content

w3.org/TR/css-flexbox-1/#justify-content-property

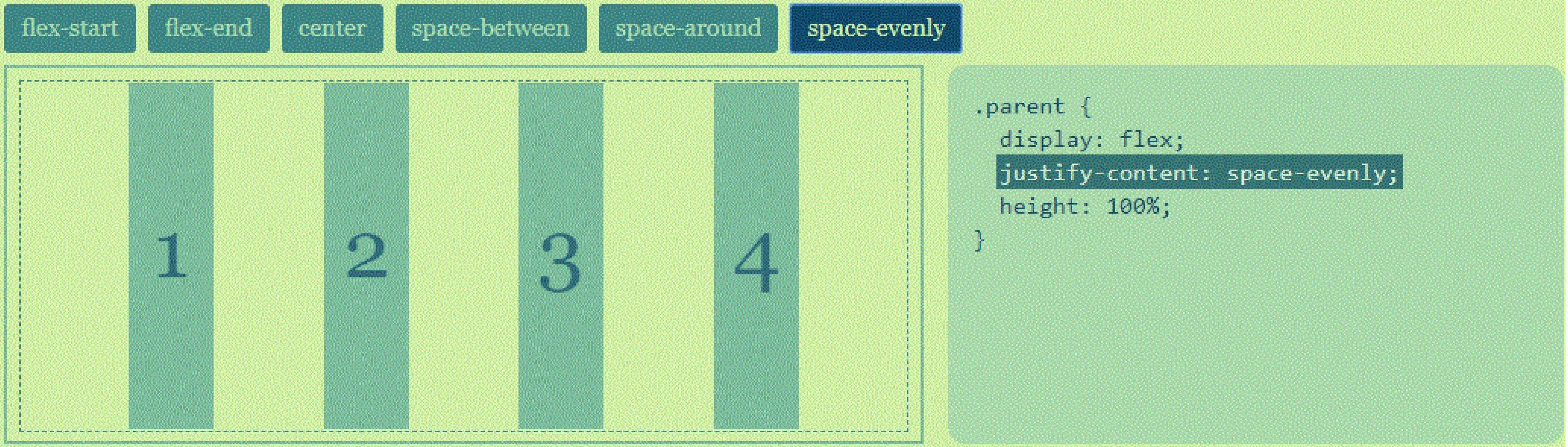


```
.parent {
  display: flex;
  justify-content: space-around;
  height: 100%;
}
```

The *justify-content* property aligns *flex items* along the *main axis* of the current line of the flex container. This is done *after* any flexible lengths and any *auto margins* have been resolved. Typically it helps distribute extra free space leftover when either all the *flex items* on a line are inflexible, or are flexible but have reached their maximum size. It also exerts some control over the alignment of items when they overflow the line.

justify-content

w3.org/TR/css-flexbox-1/#justify-content-property

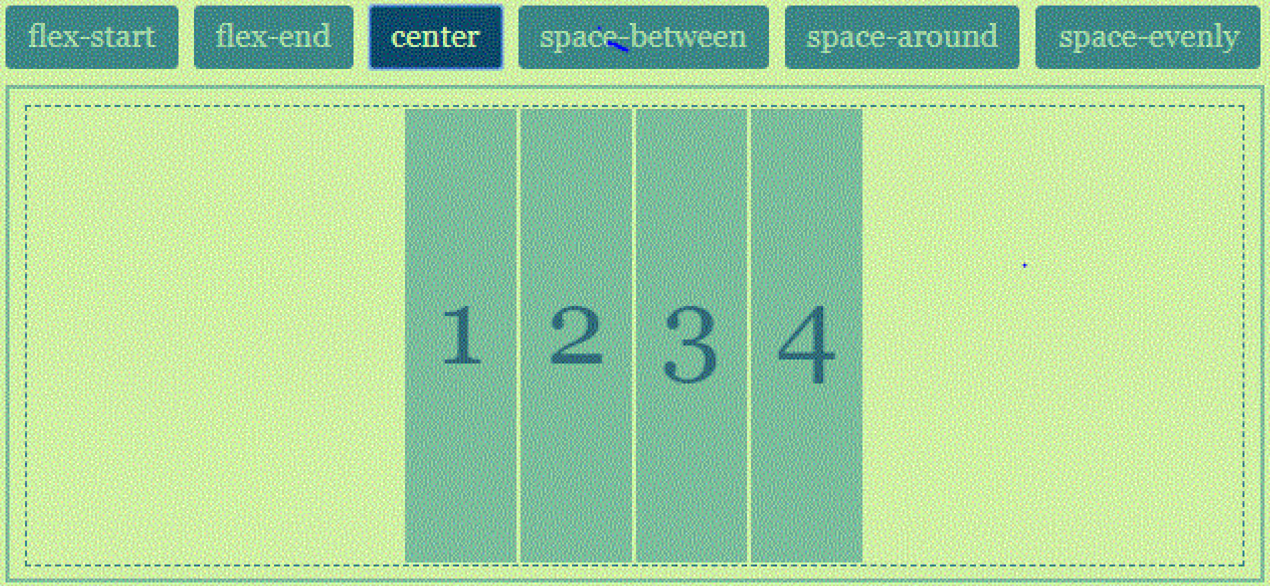


```
.parent {
  display: flex;
  justify-content: space-evenly;
  height: 100%;
}
```

The *justify-content* property aligns *flex items* along the *main axis* of the current line of the flex container. This is done *after* any flexible lengths and any *auto margins* have been resolved. Typically it helps distribute extra free space leftover when either all the *flex items* on a line are inflexible, or are flexible but have reached their maximum size. It also exerts some control over the alignment of items when they overflow the line.

justify-content

w3.org/TR/css-flexbox-1/#justify-content-property

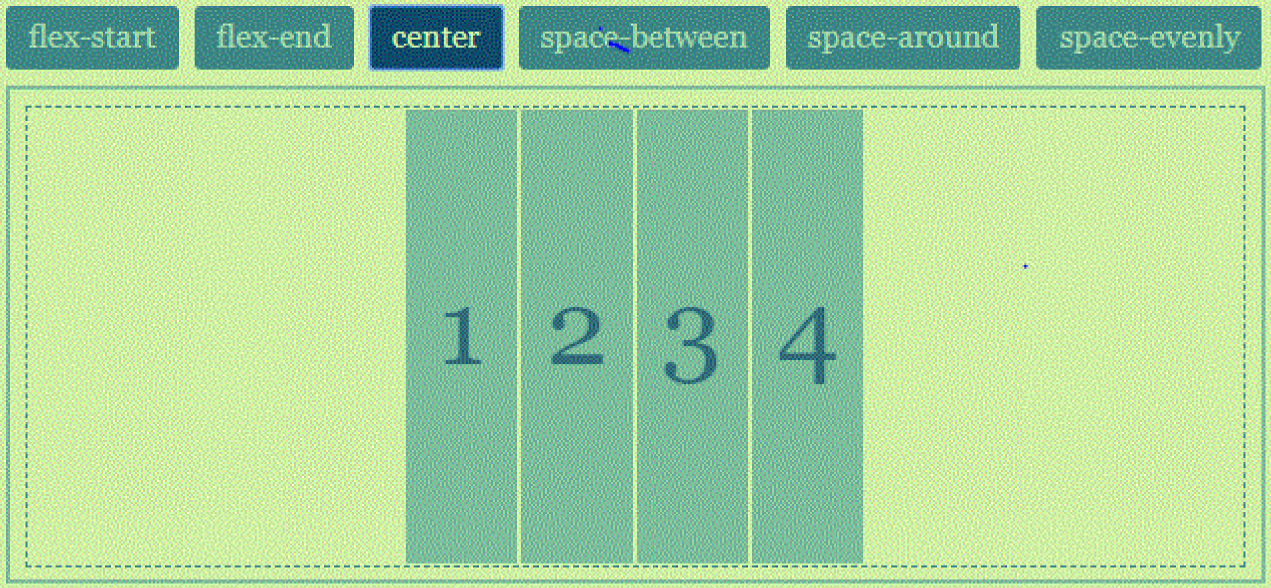


```
.parent {
  display: flex;
  justify-content: center;
  height: 100%;
}
```

The *justify-content* property aligns *flex items* along the *main axis* of the current line of the flex container. This is done *after* any flexible lengths and any *auto margins* have been resolved. Typically it helps distribute extra free space leftover when either all the *flex items* on a line are inflexible, or are flexible but have reached their maximum size. It also exerts some control over the alignment of items when they overflow the line.

justify-content

w3.org/TR/css-flexbox-1/#justify-content-property

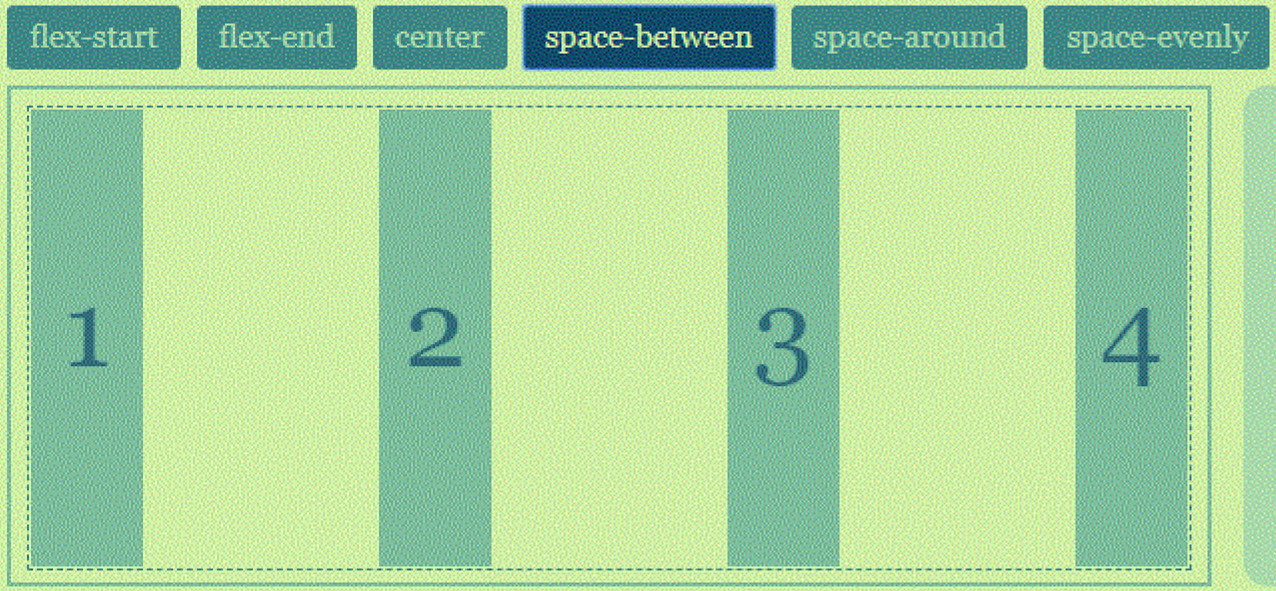


```
.parent {
  display: flex;
  justify-content: center;
  height: 100%;
}
```

The *justify-content* property aligns *flex items* along the *main axis* of the current line of the flex container. This is done *after* any flexible lengths and any *auto margins* have been resolved. Typically it helps distribute extra free space leftover when either all the *flex items* on a line are inflexible, or are flexible but have reached their maximum size. It also exerts some control over the alignment of items when they overflow the line.

justify-content

w3.org/TR/css-flexbox-1/#justify-content-property



```
.parent {
  display: flex;
  justify-content: space-between;
  height: 100%;
}
```

The *justify-content* property aligns *flex items* along the *main axis* of the current line of the flex container. This is done *after* any flexible lengths and any *auto margins* have been resolved. Typically it helps distribute extra free space leftover when either all the *flex items* on a line are inflexible, or are flexible but have reached their maximum size. It also exerts some control over the alignment of items when they overflow the line.

Remember

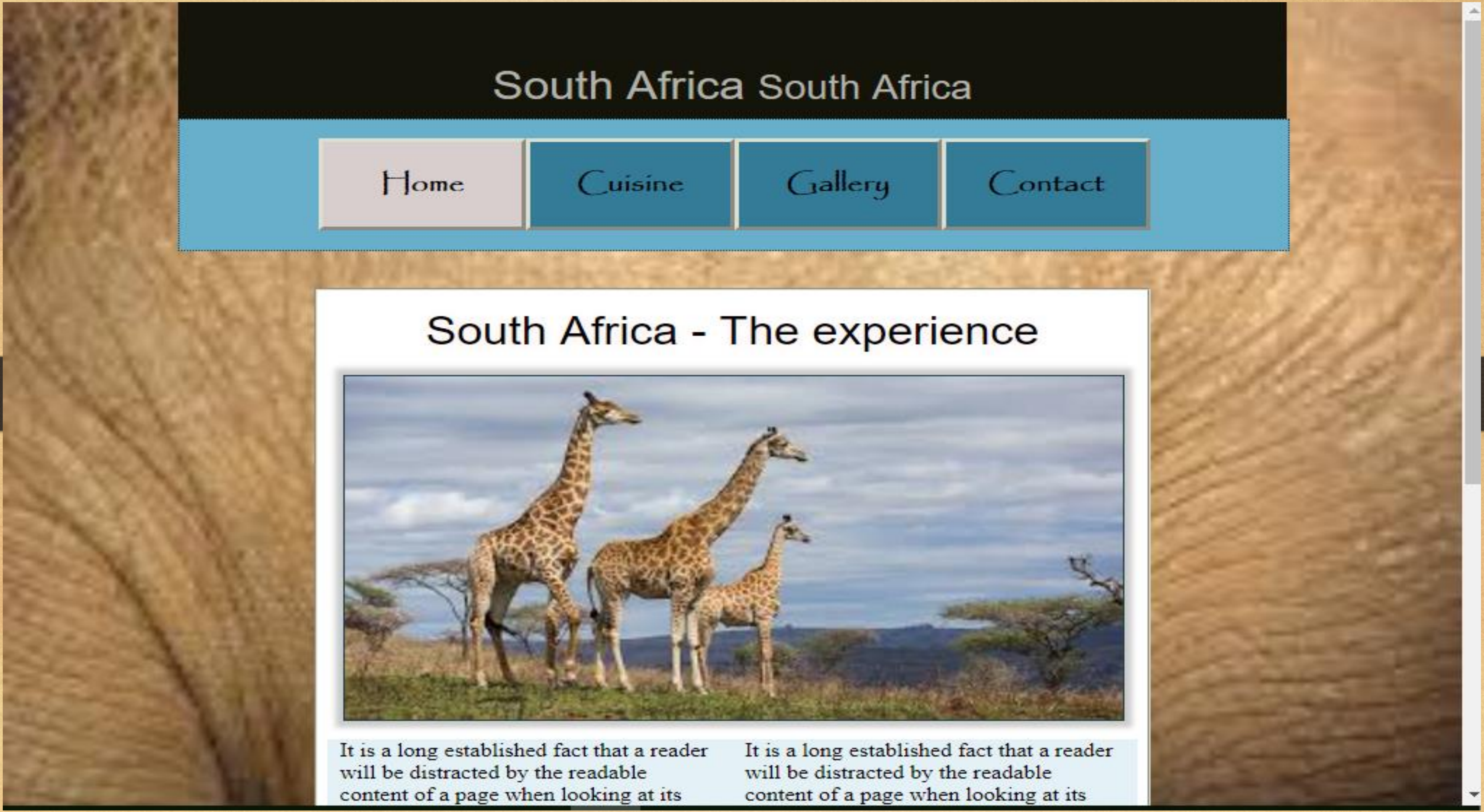
- Flex the header
- Flex the navigation bar
- Create a container – display flex
- Create the sections i.e <section><aside> <main> <article> for the main content of the page
- Add the above to the container
- Flex the footer
- Wrap and watch it resize

Header/footer – left right alignment

- Flexible container
- 2 flex items
- Text align first one LEFT (default)
- Text align second one RIGHT

wrapper

- Add the header, nav, container and footer to a wrapper
- <div id =“wrapper”>
- See example over



South Africa South Africa

South Africa - The experience

It is a long established fact that a reader will be distracted by the readable content of a page when looking at its

It is a long established fact that a reader will be distracted by the readable content of a page when looking at its

Lets look at some code