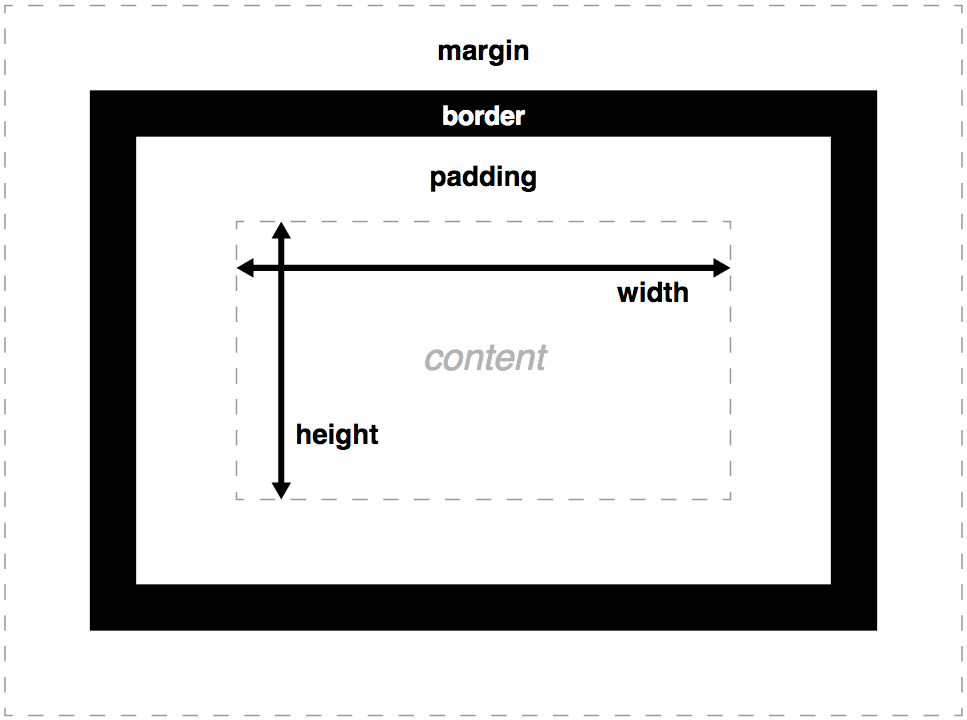
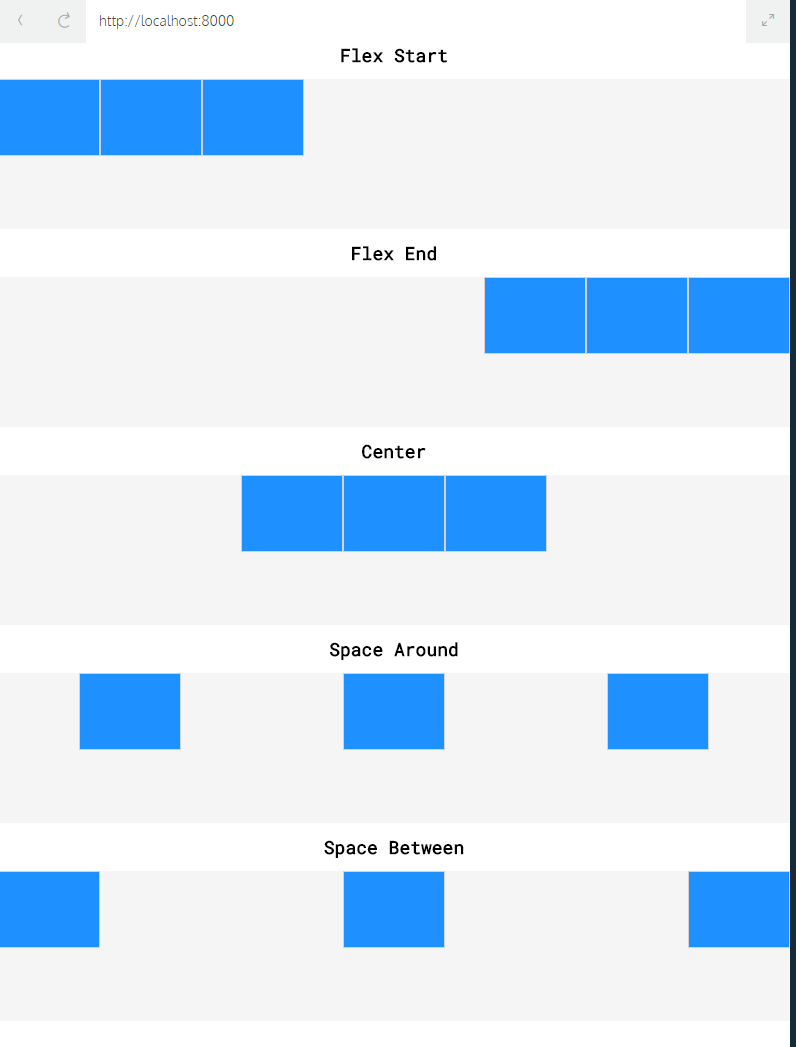
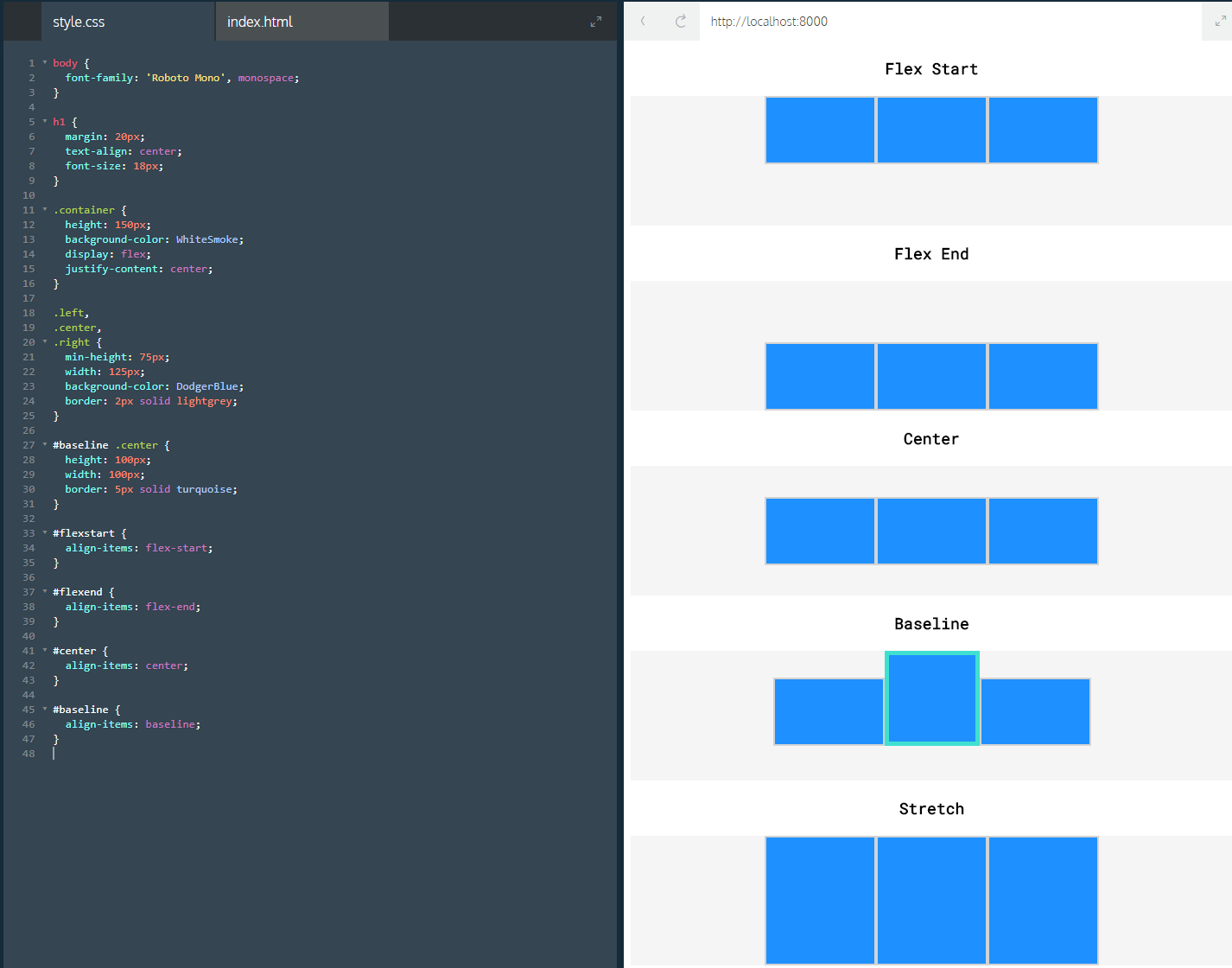
8/8/17

* Day 1
* Will learn about how to lay out websites with CSS techniques
* Learning the Box Model
* Learn the aspects of the box model
  + The dimensions of an element’s box
  + The borders of an element’s box
  + The paddings of an element’s box
  + The margins of an element’s box
* Model includes size, padding, border, and margin
  + Width and height: height and width of the content area
  + Padding: Space between the content area and the border
  + Border: The thickness and style of the border surrounding the content area and padding
  + Margin: Amount of space between border and the very outside of the element
* 
* By default, the dimensions (Height and Width) are set to hold the raw contents of the box
* Use height and width CSS properties to edit these
* NOTE – using #px for dimensions specifies how many pixels. An item that fills a computer monitor would overflow on a phone screen
* A border is like a frame, a line that surrounds an element. Borders have specific
  + Width
    - Can be set in pixels
    - Or, thin, medium, thick
  + Style: 10 different styles, including
    - None
    - Dotted
    - Solid
  + Color – just like other colors
* Default settings for a border is “medium none color”, where color = current color of the element
* You don’t type “width”, “style”, or “color”, you just type
  + P{
  + Border: 3px solid coral;
  + }
* Padding
* Padding is like the space between the picture and the frame surrounding it
* Padding: 10px
  + Puts 10 pixels of space between the content and the borders, on all four sides
* Padding-top
* Padding-right
* Padding-left
* Padding-bottom
* You can also define padding on different sides in the same declaration
* Padding: 6px 11px 4px 9px;
  + Defines padding for top, right, bottom, left (clockwise rotation, starting at top)
* Padding: 5px 10px
  + Defines top and bottoms at the same time, then left and right at the same time
  + So, top and bottom will be 5px, left & right will be 10px
* Margins are the space directly outside the box
* Margin: 20px;
  + Sets the margin for 20px on all four sides
  + Means other HTML elements can’t come within 20 pixels of the element’s border
* Margin-top, right, bottom, and left also exist
* Can do the same tricks as padding, declaring in the same line
* Auto
  + Margin: 0 auto;
    - This will center the element in their containing elements
    - The “0” sets the top and bottom margins to 0px
    - The “auto” sets the left and right margins until the element is centered within its containing element
    - NOTE: For this centering to work, you must also declare the width of the element via CSS. Otherwise it won’t center because it defaults to taking up all the space it has available.
* Padding is space added within an element’s border, while margins are space added outside an element’s border
* Margin Collapse
  + Vertical (top and bottom) margins, unlike vertical paddings, collapse
  + Horizontal margins (next two each other) add together
  + However, vertical margins **do not add**. Instead, the **larger vertical margin** is the one that sets the distance.
* Overflow
  + What happens if an element’s size (meaning it’s dimensions, AND the added dimensions of its padding, borders, and margins) are bigger than the parent’s containing area?
  + CSS property “overflow” can have three values
    - Hidden – any content that overflows is hidden from view
    - Scroll – overflow content can be viewed by scrolling with the added scroll bar
    - Visible – the overflow content is visible outside the container
  + Default overflow value is visible
  + NOTE: You set the overflow value on the parent element, to tell the browser how to render the child elements.

8/9/17

* Day 2
* Quiz on Box Model: Got a 13/13 on first try
* CSS Display Properties
  + Used to arrange elements on a webpage
* All HTML elements have a default “display” property
  + *Display* dictates how an element shares horizontal space - does it take up the entire left-to-right row, or just what it needs and can share with another element?
* Three values of the display property:
  + Inline, block, and inline-block
* Inline elements have a box that wraps **tightly** around the content, taking only the needed space for the element
  + Example: the <em> (italics) tag is inline because it doesn’t take up a whole line, the rest of the text in a <p> for example can occupy the same horizontal line
* You can turn elements into inline elements (if they aren’t by default) with the display property in CSS; set it to “inline”
* If you set display to block, the element fills the entire width of the page and the height necessary to accommodate the element
* If you set display to “inline-block”, these elements are displayed next to each other, but you can also define the dimensions with height and width
  + Images are default inline-block
* In addition to the display property (which has three values: block, inline, inline-block), there is the position property
  + Position specifies where the element is placed in relation to other elements
  + Also tells whether or not the element moves when the user scrolls
* Static – (default) positioned where it normally occurs in the flow of the document, left to right and top to bottom
* Absolute – positioned exactly where specified, in relation to the nearest non-static element. Removed from the flow of the web page.
* Element positioned in relation to where it would have occurred. NOT removed from the flow, space is reserved for it.
* Fixed – positioned in the same place in the viewport at all time, even while scrolling. Removed from the flow.
* Absolute
  + Allows you to specify exactly where an element should appear
  + Because it’s removed from the flow of the page, the element basically floats on top of the other content, the space it used to occupy is filled
  + When changed to static, the element moves to the top left of the nearest container that is not statically positioned
  + Absolute positioning overrides display property, its box will wrap tightly and not span the page
  + Top – specifies how far from the top of the non-static parent container the element should be
  + Left, right, and bottom are the same for their sides
  + Absolute positioning is not popular now with the advent of mobile development
  + Not used much
* Fixed
  + Remains visible to the user at all times, even when scrolling.
  + Has the same top/bottom/left/right values, but this time the distances are relative to the viewport, not the page itself
* Relative
  + Does not remove the element from the flow of the page
* Z-Index
  + Value accepts only integers
  + Controls how far “back” or “forward”, to stack/overlap elements how you desire
  + No negative values
  + Doesn’t need to start at 1… Any element you want higher than another just has to have a higher z-index value
  + Standard practice is increments of 5: 5, 10, 15,…
  + NOTE: z-index will be ignored if position is static(default)

Day 4

* Quiz on CSS display and positioning
  + Got a 14/14 on third try
* Flexbox
  + New tool developed for CSS3 that simplifies positioning
  + Short for Flexible Box Layout
  + Two components: flex containers and flex items
  + All child elements of a flex container are flex items
  + To set an item as a flex container
    - Set its display value to “flex” or “inline-flex”
    - Once an item is a flex container, you can then use several properties (to be talked about in this lesson) to specify how the children behave
  + Any element can be a flex container
  + Changing the container to a flex container does not make that container a flex item, just its children
    - So changing a div to a flex container will not change the fact that the div will still remain block level, making a new line for adjacent elements
  + Inline-flex: makes the flex container itself inline
    - So now you can have multiple flex containers next to each other
  + Justify-content
    - Default behavior of flex containers and their children is to move to the top left corner of their parent containers
    - Use justify-content to position items from left to right
    - Justify content can have five values
      * Flex-start: all items positioned in order, starting from the left of the parent, with no extra space in between items
      * Flex-end: all items, starting from the last item, start at the right of the parent container and no extra space in between
      * Center: Positioned in order, starting in the center of the parent container
      * Space-around: items are positioned with equal space before and after each item (resulting in double the space between items)
      * Space-between: same as space around, but no space before the first or after the last items
      * SEE IMAGE BELOW
    - 
* Align-items
  + Makes it possible to space items vertically in a flex container
  + Align-items property can have five values:
    - Flex-start: all elements will be at the top of the parent container
    - Flex-end: all elements will be positioned at the bottom of the parent container
    - Center: the center of all elements will be positioned at the bottom of the parent container
    - Baseline: the bottom **of the content** all elements will be aligned with each other
    - Stretch: if possible, the items will stretch from top to bottom of the container (unless the element has a specifies height)
  + 
* Flex-grow
  + Flex-grow allows items to grow proportionally when their containers grow larger than needed
  + Flex-grow: 2 will grow twice as fast as elements with flex-grow: 1. These elements can be next to each other.
  + If a max-width is set, the element won’t grow larger than that.
* Flex-shrink:
  + Same as flex-grow but for shrinking
  + Default value of flex-shrink is 1
    - Fact: the default value of flex-grow is 0, so that one must be declared to actually grow
* Flex-basis:
  + Another way of specifying the width of a flex item
  + Allows you to specify the width of an before it stretches/shrinks
  + Good way of specifying width of items in a flex container expected to shrink/grow
* Flex property
  + CONVENIENT way of declaring the following on one line
    - Flex grow
    - Flex shrink
    - Flex basis
  + So we can turn
    - Div{
    - Flex-grow: 2;
    - Flex-shrink: 1;
    - Flex-basis: 150px;
    - }
  + Into…
    - Div{
    - Flex: 2 1 150px;
    - }
  + Oh wow
  + Declared (flex grow), (flex shrink), (flex-basis), in that order
  + If you just declare two numbers (no “XXpx”) that just declares shrink and grow
  + If you just declare one number and one “XXpx” that just declares grow and basis
    - Note there is no way to declare just shrink and basis with only two numbers
* Flex-wrap
  + Accepts three values
    - Wrap: child elements in a flex container will move down to the next line starting at the final item and moving toward the first
    - Nowrap: (DEFAULT) prevents wrapping
    - Wrap-reverse: The wrapped element appears on top of the other items. (Mirror image of wrap)
* Align-content:
  + For aligning elements within a single row. If a flex container has multiple rows of content, align-content will space the rows from top to bottom
  + Has six values
    - Flex-start: all rows will be positioned at the top of the parent with no space between
    - Flex-end: all rows positioned at bottom of parent, no extra space
    - Center: all rows at center of parent, no extra space
    - Space-between: all rows will be spaced evenly from top to bottom (no space above or below end elements)
    - Space-around: same as space-between but with space above and below the end elements
    - Stretch: (DEFAULT) if minimum height or no height is defined, the tows will stretch to fill the parent top to bottom
  + NOTE – align-content is declared on the container, not the flex items
  + Similar output as align items
* Flex-direction:
  + By default with flex items, the “major axis” is horizontal and the “cross axis” is vertical
  + By default, the following use the major axis:
    - Justify-content
    - Flex-wrap
    - Flex-grow
    - Flex-shrink
  + …and by default the default use the cross axis
    - Align-items
    - Align-content
  + But these axis are interchangeable, switch them with “flex-direction”
  + Add the flex-direction property and give it a value of “column”
  + Four values
    - Row: (DEFAULT) elements are positioned left to right across the parent element starting in the top left corner
    - Row-reverse: right to left, starting in top right corner
    - Column: Elements positioned top to bottom, starting in the top left corner
    - Column-reverse: elements positioned bottom to top, starting in bottom left corner
  + NOTE: flex-direction is declared on flex-containers, not items
* Flex-flow
  + Similar to the flex property, this one declares in one line:
    - Flex-wrap
    - Flex-direction
  + Now we can take this:
    - .container{
    - Display: flex;
    - Flex-wrap: wrap;
    - Flex-direction: column;
    - }
  + …and turn it into this
    - .container{
    - Display: flex;
    - Flex-flow: column wrap;
    - }
  + Note: first value is flex-direction, and the second is flex-wrap
* Nested flexboxes
  + It’s possible to position flex containers within one another
* REVIEW:
  + 
* Quiz on flexbox: got a 16/16 on first try
* Project using flexbox – Dave’s burgers
* Project using CSS display properties (fixing a broken site): Survey
* Day 8: Project on flexbox
  + Note how to break down a complete site
* Days 9-13
  + First Positioning Project on my own
  + NOTE: Day 13 should be WEDNESDAY 23 AUGUST
  + Create a website with news called My Times
    - USE: box model, flexbox, and CSS display
    - This was a bad exercise and I didn’t finish