# Advanced Web Programming

# Lab1 - HTML5 & CSS3

The aim of lab 1 is to review the most important idea ( tags and properties) HTML and CSS. We focus on creating a website layout. We will recall the properties of Flex, Grid and what is RWD (Responsible Web Design)

## Section 1. Flex

**The aim of the exercises in that section is to practice the Flexbox properties.**

CSS provides many tools and properties that you can use to position elements on a webpage. In this assignment you will learn about Flexible Box Layout or flexbox, a tool that greatly simplifies how to position or manage elements.

There are two important components to a flexbox layout: flex containers and flex items. A flex container is an element on a page that contains flex items. All direct child elements of a flex container are flex items.

To designate an element as a flex container, set the element’s display property to flex or inline-flex. Once an item is a flex container, there are several properties we can use to specify how its children behave. In this lab you can get to know and test all these properties:

1. justify-content
2. align-items
3. align-self
4. flex-grow
5. flex-shrink
6. flex-basis
7. flex (shorthand for 4, 5,6)
8. flex-wrap
9. align-content
10. flex-direction

## 11. flex-flow

You can create a Flexbox layout in context by declaring a container using this property:

display : flex

A flexbox container has two axes:  main axis  && cross axis. The main axis is determined by the value of the flex-direction property. The main axis can go in these directions:

flex-direction: row          // left to right     (default)

flex-direction: row-reverse    // right to left

flex-direction: column      // top to bottom

flex-direction: column-reverse    // bottom to top

Each one of the flexbox properties works either on the container or on the items, defining the way how items should be laid out inside the container. Flex items can be laid out in all directions across their particular axis: left to right, right to left, top to bottom or bottom to top.

Exercise 1.  In Catalog FlexExample/Example1  you can find files index.html and style.css. Edit the style.css file and modify class container to display items in one row. Later display items in one column in reverse order.

The **justify-content** property specifies how flex-items are distributed along the main axis of their parent flex-container. This property has 5 different possible values:

- flex-start — all items will be positioned in order, starting from the left of the parent container, with no extra space between or before them.
- flex-end — all items will be positioned in order, with the last item starting on the right side of the parent container, with no extra space between or after them.
- center — all items will be positioned in order, in the center of the parent container with no extra space before, between, or after them.
- space-around — items will be positioned with equal space before and after each item, resulting in double the space between elements.
- space-between — items will be positioned with equal space between them, but no extra space before the first or after the last elements.

The size of each individual flex item is not changed by this property.

Exercise 2. In Catalog FlexExample/Example2 you can find files index.html and style.css. Edit the style.css file and modify appropriate classes to display items in a row according to the expected effect. Later change flex direction to allow display items in one column in reverse order. Verify how each value justify-content works when you change main-axis direction. If you wish you can test the different values of the justify-content property for another flex direction value.

As you see the justify-content property specifies how the flex-items are distributed along the main axis, whereas the align-items property specifies how the flex-items are distributed along the cross axis of the flex-container. Then flex direction is set to row align-items defines vertical alignment. It is defined by adding an **align-items** property to a flex container.

The available options for align-items is similar to justify-content:

center

flex-start (top)

flex-end (bottom)

stretch

baseline

Most of these are pretty straightforward. The stretch option is worth a taking a minute to play with because it lets you display the background of each element. The box for each item extends the full height of the flex container, regardless of how much content it contains. A common use case for this behavior is creating equal-height columns with a variable amount of content in each one—something very difficult to do for example with floats.

Exercise 3. In Catalog FlexExample/Example3 you can find files index.html and style.css. Edit the style.css file and modify appropriate classes to display items using every align-items value.

Next change height of item's container and check using stretch value for new height. Verify how each value align-items works when you change main-axis direction.

In Catalog FlexExample/Example3 you can find files index3.html . There are two elements in the HTML below, a div with a class of .parent which contains another div with a class of .child. Use Flexbox to center the child inside the parent.

Sometimes we want to change position only one items in container.  What to do? You can align it individually! This is where the **align-self** property comes in. Adding this to a flex item overrides the align-items value from its container.

You can align elements in other ways using the same values as the align-items property, listed below for convenience.

center

flex-start   (top)

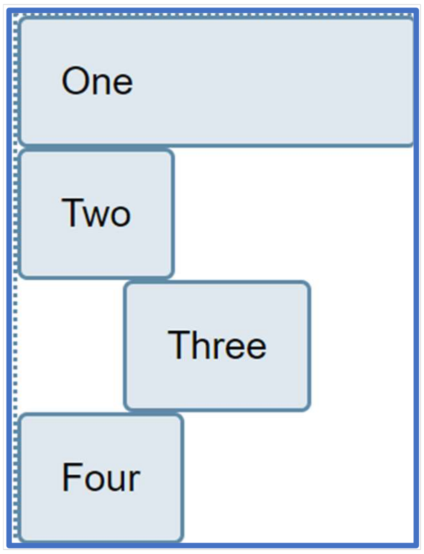
flex-end      (bottom)

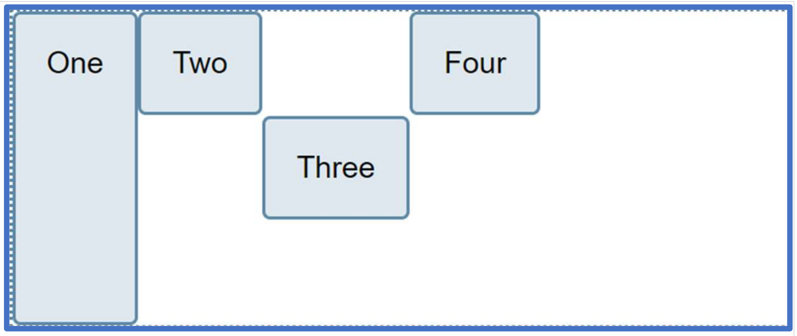
stretch

baseline

Exercise 4.  In Catalog FlexExample/Example3  you can find files index2.html and style2.css.  Edit the style2.css file and modify the code to get

a.



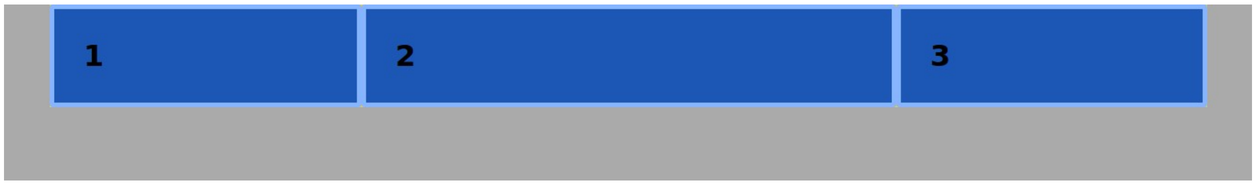
b. 

It is recommended not to modify HTML code (problem is how to get box nr 1). If you don't know how to do it you can get id into html code and style it.

The flex-grow property specifies how a flex-item inside the flex container will grow - along the main axis - relative to its sibling items, taking into account the available space inside the flex container.

Flex items are flexible: they can shrink and stretch to match the width of their containers. The flex-grow property defines the width of individual items in a flex container. Or, more accurately, it allows them to have flexible widths. It works as a weight that tells the flex container how to distribute extra space to each item. If all items have flex-grow set to 1, the remaining space in the container will be distributed equally to all children. If one of the children has a value of 2, the remaining space would take up twice as much space as the others.

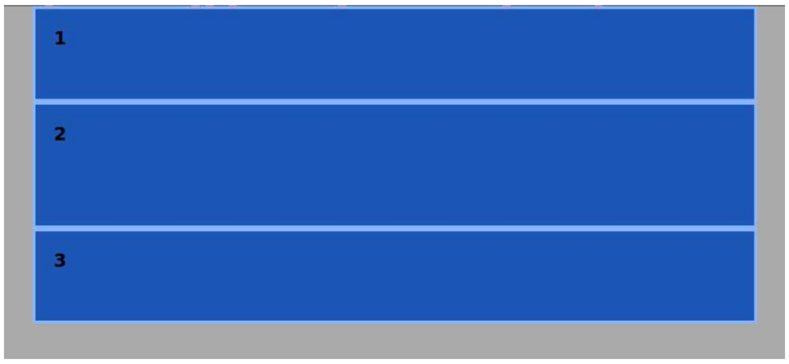
Exercise 5. In Catalog FlexExample/Example4 you can find files index.html and style.css. Edit the file style.css to get finish effect as bellow



the second item takes twice as much space, as the other two items. If you resize your browser window, you'll notice that the second item will keep this proportion, if there's enough space available considering the width of the viewport.

When working on the block axis, it is important to remember that the height of items is by default set to auto This means that each flex-item is only as high as the content inside it.

In order to test the flex-grow property on the block axis, you have to declare a fixed height for the flex container from exercise 5.



The second item is now twice as high as the other two items, according to the available vertical space within the flex container (which has a fixed width). This makes sense if you have a fixed design and you know exactly the size of the content. The flex-grow property will not work if the content of one of the items is higher than the height the item is supposed to have.

Add to context second item for example:

```
Lorem ipsum dolor sit amet consectetur adipisicing elit. Consequuntur eos dolores impedit nulla delectus illo dolorum quas, laudantium voluptatem iusto ab tempore porro, earum aliquid debitis repudiandae magni aperiam reiciendis natus odit explicabo. Ut deleniti ea autem voluptatem, explicabo neque modi atque recusandae commodi molestiae quasi itaque repellat similique debitis pariatur aliquid ipsam soluta error doloremque ab quas. Soluta, nemo!
```

Verify the ending effect.

The **flex-basis** property sets the initial length of the flex-items inside a flex-container. You can think of it as an improved version of the width or height values. That is, flex-basis has always prevalence over width or height.

This defines the default size of an element before the remaining space is distributed. It can be a length (e.g. 20%, 5rem, etc.) or a keyword. If set to 0, the extra space around content isn't factored in. If set to auto, the extra space is distributed based on its flex-grow value

The **flex-shrink** property specifies how items behave when there's not enough space available for all of them to fit. The default value for flex-shrink is 1. This means that all items shrink in an equal proportion

Exercise 6. In Catalog FlexExample/Example4 you can find files index2.html and style2.css. First define width for item using flex-basic. Set width as 33,3%. Check finish effect. Now define for `.item { flex-basis: 25%; width: 33.33%; }`. Check result. Make a conclusion how flex-basic works.

Go back to your code editor in order to change the width of the parent container to 600px. Now, change the width of the items to 250px each.

Despite the fact that you declared each item to be 250px wide, items cannot overflow the width of their parent. So each item will shrink to 200px in the same proportion (flex-shrink: 1) to fill the available space.

Now  define flex-shrink: 2 to item nr 2  and flex-shrink: 0 to item nr 3. Verify effects.

You also learned that flex-shrink sets how items shrink relative to each other if there is not enough space available within the flex-container. The flex-basis property assigns a fixed, ideal width (or height) to the flex-items. All these properties can be summed up in one shorthand property, the **flex** property

The three properties assigned to the item can be shortened in one line using the flex shorthand property. You have to enter the values in this order:

1. flex-grow
2. flex-shrink
3. flex-basis.

For example .item { flex: 1 1 150px; }

Exercise 7.  In Catalog FlexExample/Example4  you can find files index3.html and style3.css.   Using property flex set item box width to 150px. Next for item nr 1 set width as twice as other.

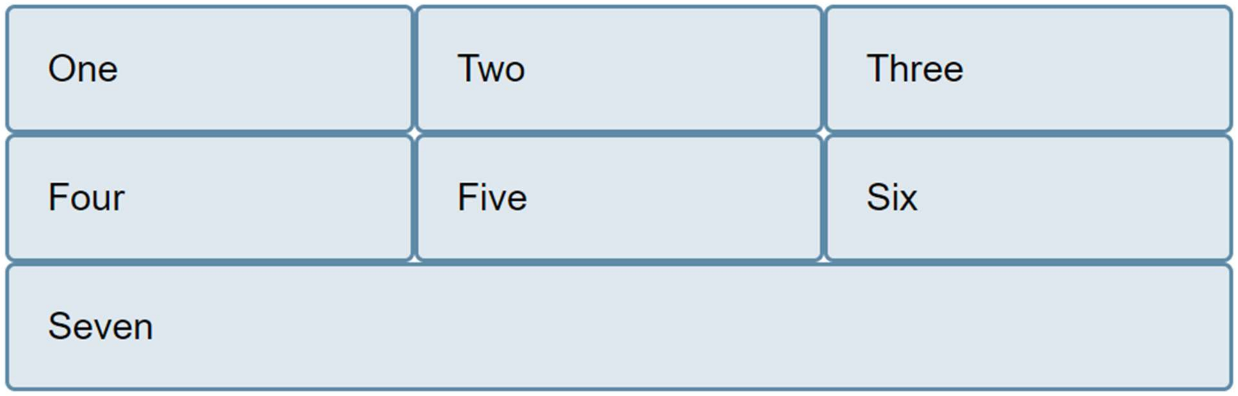
Until now, you have seen that flex-items stay on the same line, regardless of the content inside them and the space available. With the flex-wrap property, it is possible to make flex-items wrap over to the next line.

To create a grid, we need the flex-wrap property. It has fallow value:

- nowrap (default): all flex items will be on one line
- wrap: flex items will wrap onto multiple lines, from top to bottom.
- wrap-reverse: flex items will wrap onto multiple lines from bottom to top.

Exercise 8.  In Catalog FlexExample/Example5  you can find files index.html and style.css.   Your container width is set to 600px.  Create next four block items in html. Check what your see in browser. As you see items overflow containers width. Check flex-wrap properties.

Exercise 9. In Catalog FlexExample/Example5 you can find files index2.html and style2.css. Modify code to get a view as you see below.



The align-content property specifies how the lines inside the container will be distributed, once you have applied the flex-wrap property.

The align-content property accepts 6 possible values:

- stretch (default)
- center
- flex-start
- flex-end
- space-between
- space-around

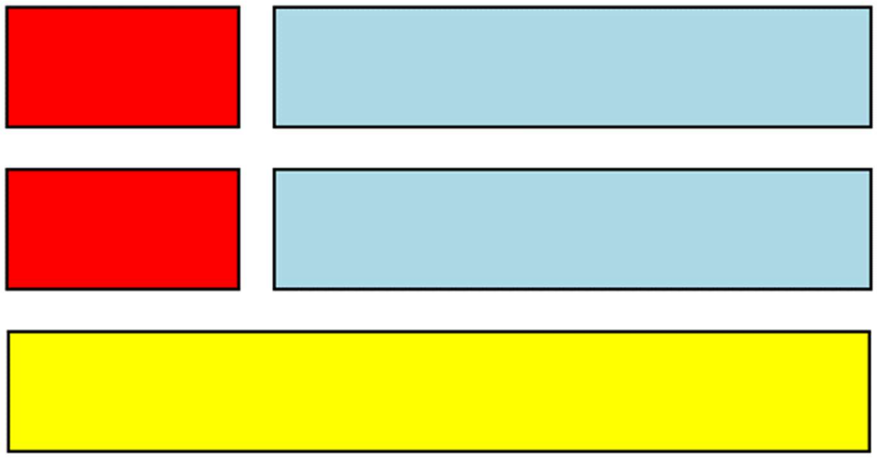
Exercise 10. For example with exercise 8 test align-content. Check all possible value.

If you want to place flex-items in a particular sequence inside their flex-container, independently of how they are placed in the HTML code, you use the CSS Flexbox **order** property. The order property gives you much more flexibility because it allows you to visually change the order of each item and still keep the source order in the markup (HTML code). Adding an order property to a flex item defines its order in the container without affecting surrounding items. Its default value is 0, and increasing or decreasing it from there moves the item to the right or left, respectively.

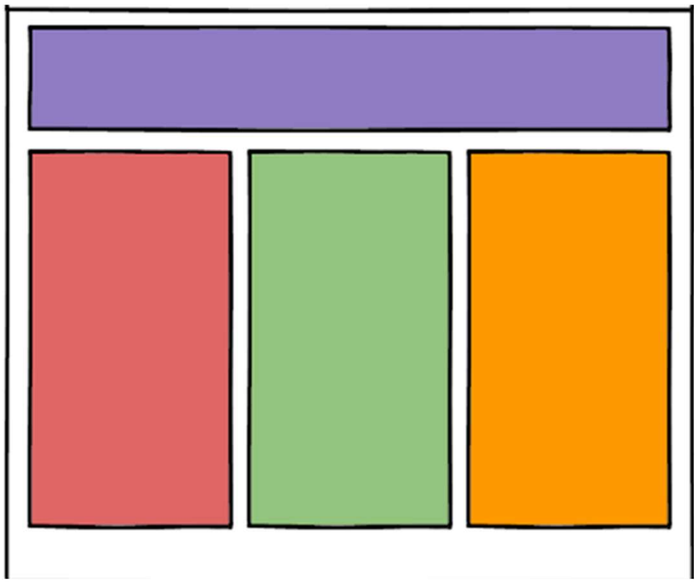
Exercise 11. In Catalog FlexExample/Example5 you can find files index2.html and style2.css. Display rows in the order: 4 1 3 5 6 2.

Now time to verify you knowledge of Flexbox properties.

Exercise 12. Bellow you can see three mocks. Create html and css files ( using flexbox properties) which looks like mocks.

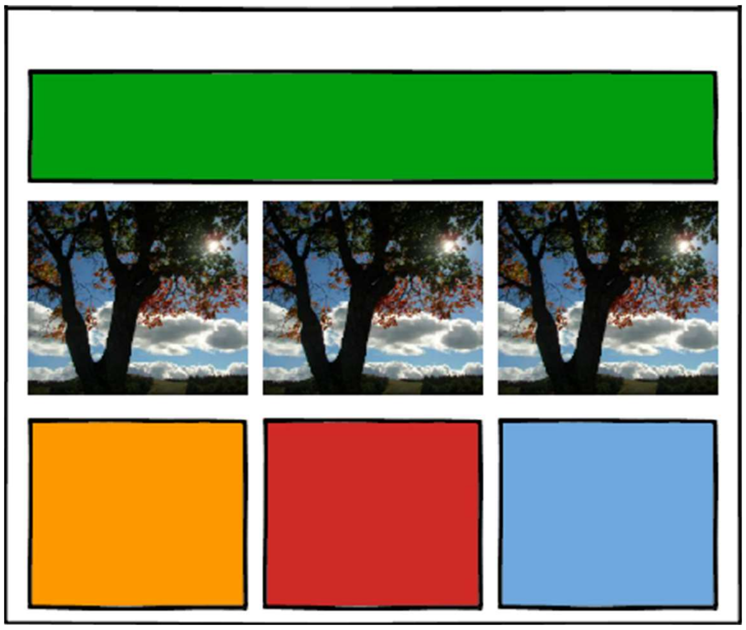


a. Mock 1



b. Mock 2

c. Mock 3

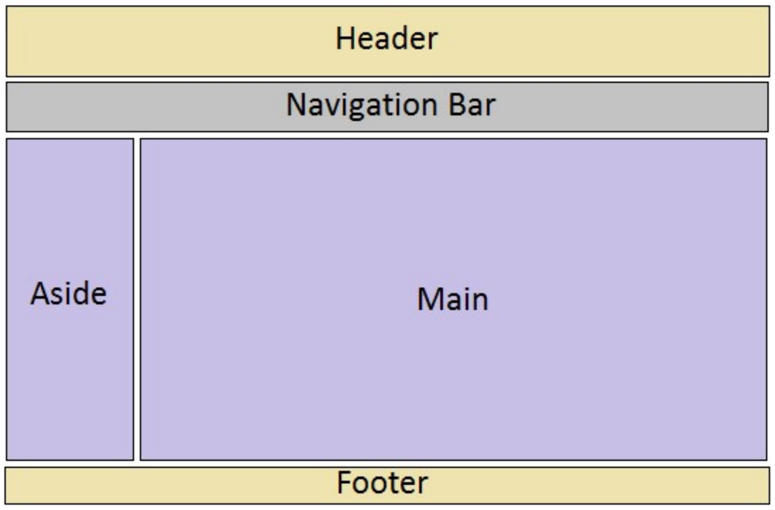


---------------------------------------------------------------------------------------

## Section 2.  How to get effect

**The aim of the exercises in that section is to use Flexbox properties to get standard web design effect.**

Exercise 13.  We start by design page layout.   We create  typical two columns layout which looks like



Important!!! Use new semantic tags to generate structure our pages. Not use div tag to create main elements in structure. Let divs be using only to aggregate element for styling.

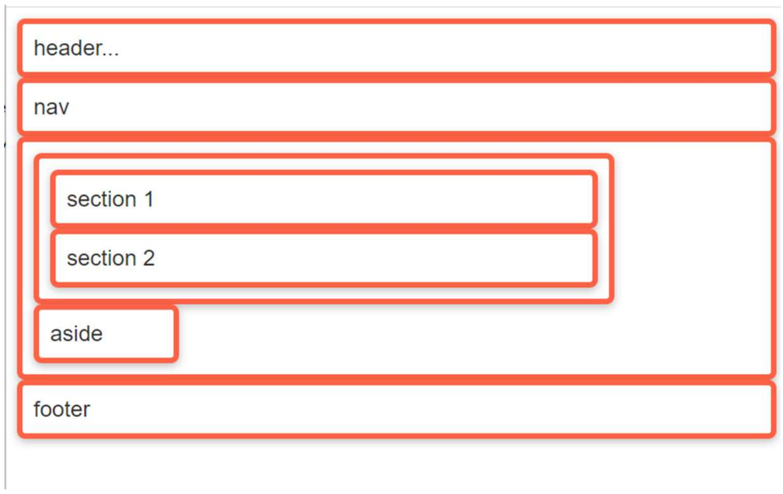
Try to do it yourself. If you have problem in section3/example1_layout you can find template of index.html.

Now our layout webpage works. But if you resize your viewport you can see that yours webpages looks not very well. When width of your viewport is less when 900px will be better to display all context in one column. How to do it?

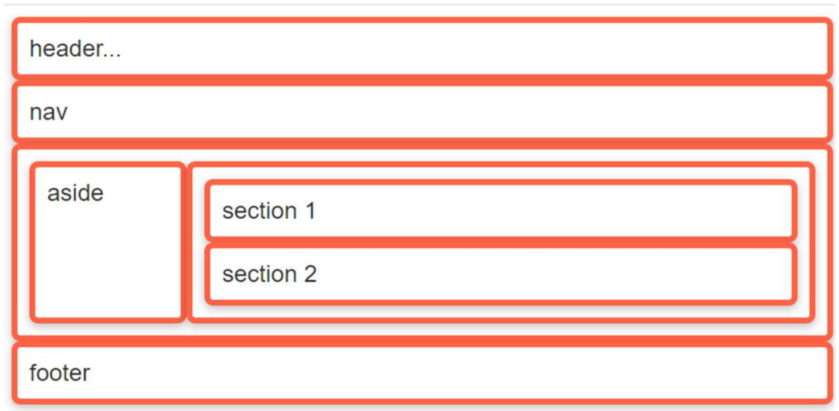
We can use media query and generate Responsive Web Design (RWD) pages.

Exercise 14. In section2/example1_layout you can find file flex-based-rwd,html. Run in browser and verify resize viewport. As you see when width is less then 600px number of columns is changed. Modify your layout from exercise 13 and less 900px display all context only in one column.

Example:



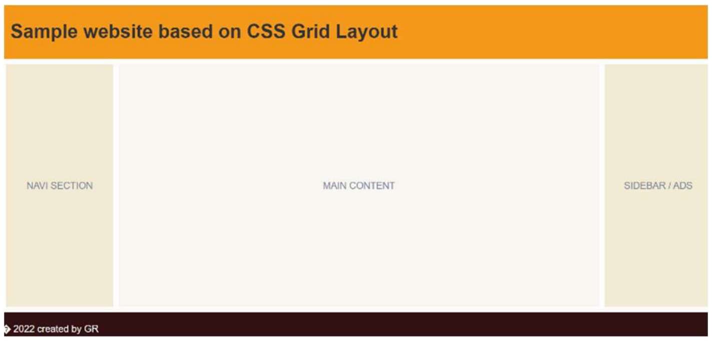
Less then 900px



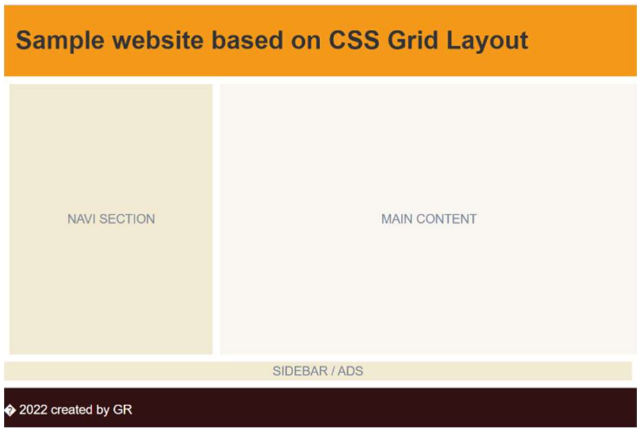
More then 900px

Exercise 15. In section2/example1B_layout you can find file index,html and style.css. Modify your css file by using the conditional styling, let the resolution change the layout of web page. Let the 3 column layout changes to a 2 column layout (the right column is to be moved below) and then a 1 column layout - as you see below. Use the display: grid approach

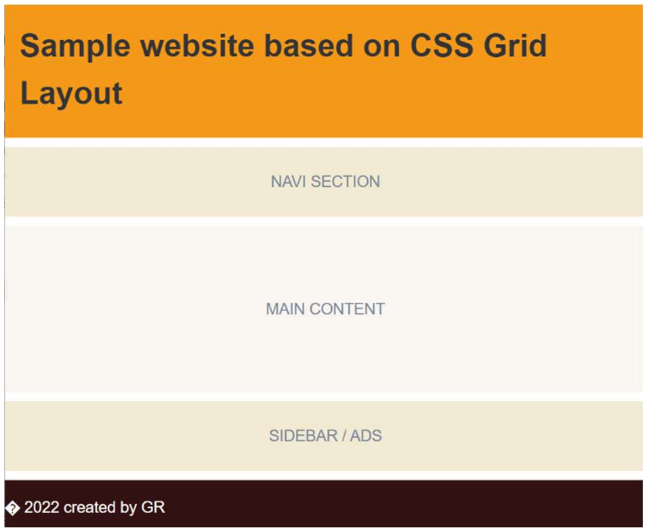
3 columns example:



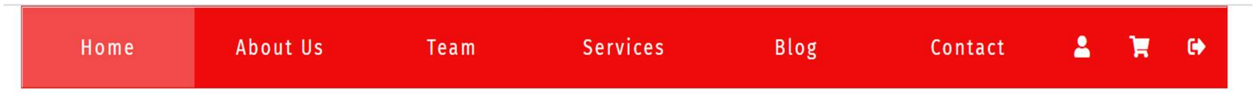
2 columns example



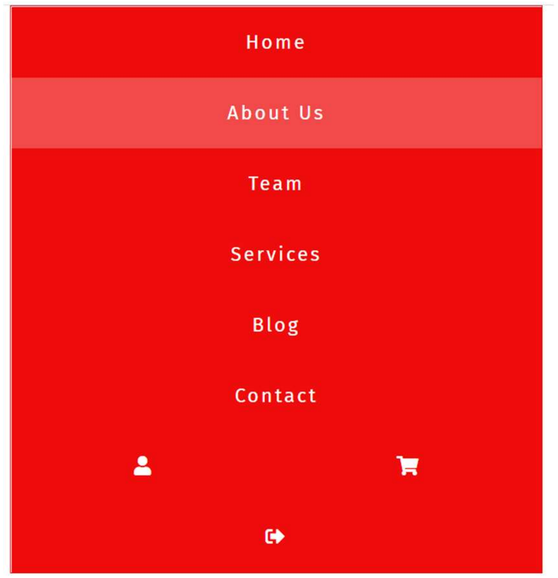
One column example:



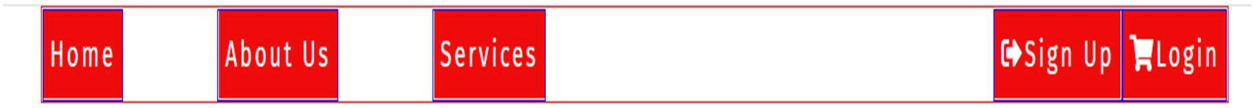
Exercise 16. In section2/example2_menu you can find file index,html. Create style.css which give you menu looks like in picture bellow



Let yours nav works as responsive. It means that the size of viewport will be less 900px then nav will be display this.



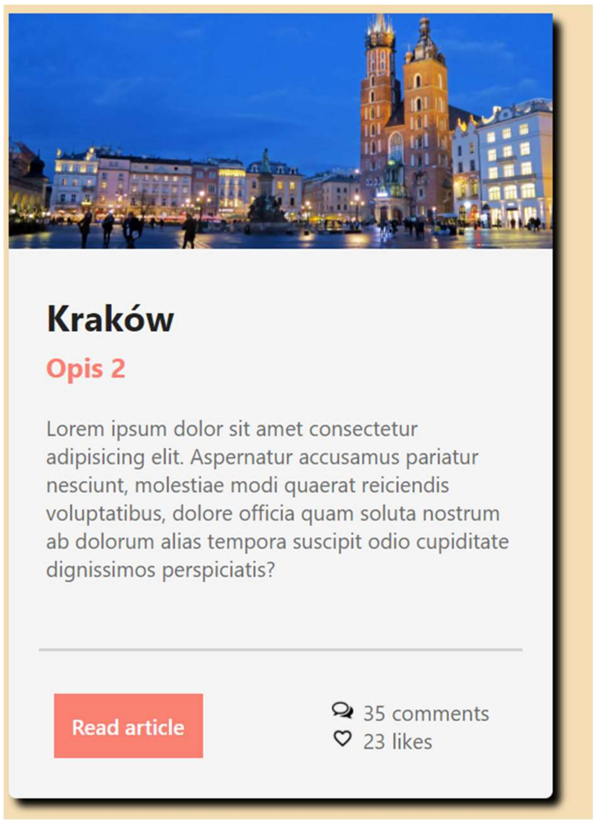
Exercise 17. Change menu from exercise 16 to get effect as below



Hint!.  Flex containers only know how to position elements that are one level deep. They don't care one bit about what's inside their flex items. This means that grouping flex items is another weapon in your layout-creation arsenal. Wrapping a bunch of items in an extra <div> results in a totally different web page.

Exercise 18.  Cards.

In section2/example3_cards   you can find file  index.html. Create style.css which give you set of cards. Single card should looks like as below;

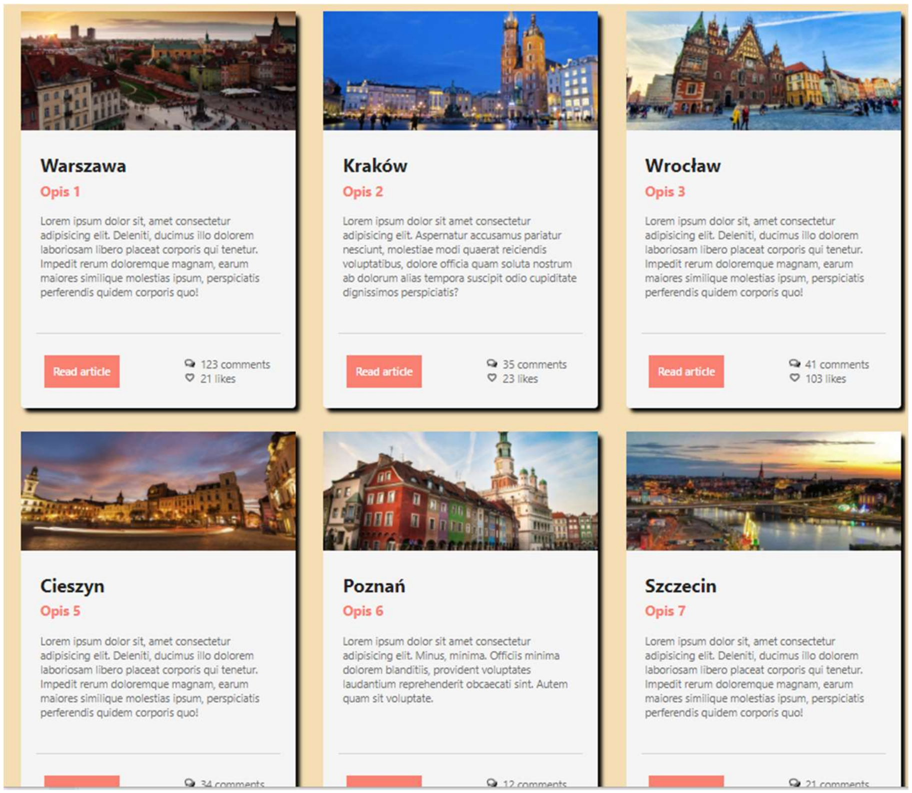


Kraków

Opis 2

Lorem ipsum dolor sit amet consectetur adipisicing elit. Aspernatur accusamus pariatur nesciunt, molestiae modi quaerat reiciendis voluptatibus, dolore officia quam soluta nostrum ab dolorum alias tempora suscipit odio cupiditate dignissimos perspiciatis?

All page should be display something like this.


Warszawa

Opis 1

Lorem ipsum dolor sit, amet consectetur adipisicing elit. Deleniti, ducimus illo dolorem laboriosam libero placeat corporis qui tenetur. Impedit rerum doloremque magnam, earum maiores similique molestias ipsum, perspiciatis perferendis quidem corporis quo!

Read article

123 comments
21 likes

Kraków

Opis 2

Lorem ipsum dolor sit amet consectetur adipisicing elit. Aspernatur accusamus pariatur nesciunt, molestiae modi quaerat reiciendis voluptatibus, dolore officia quam soluta nostrum ab dolorum alias tempora suscipit odio cupiditate dignissimos perspiciatis?

Read article

35 comments
23 likes

Wrocław

Opis 3

Lorem ipsum dolor sit, amet consectetur adipisicing elit. Deleniti, ducimus illo dolorem laboriosam libero placeat corporis qui tenetur. Impedit rerum doloremque magnam, earum maiores similique molestias ipsum, perspiciatis perferendis quidem corporis quo!

Read article

41 comments
103 likes

Cieszyn

Opis 5

Lorem ipsum dolor sit, amet consectetur adipisicing elit. Deleniti, ducimus illo dolorem laboriosam libero placeat corporis qui tenetur. Impedit rerum doloremque magnam, earum maiores similique molestias ipsum, perspiciatis perferendis quidem corporis quo!

34 comments

Poznań

Opis 6

Lorem ipsum dolor sit amet consectetur adipisicing elit. Minus, minima. Officiis minima dolorem blanditiis, provident voluptates laudantium reprehenderit obcaecati sint. Autem quam sit voluptate.

12 comments

Szczecin

Opis 7

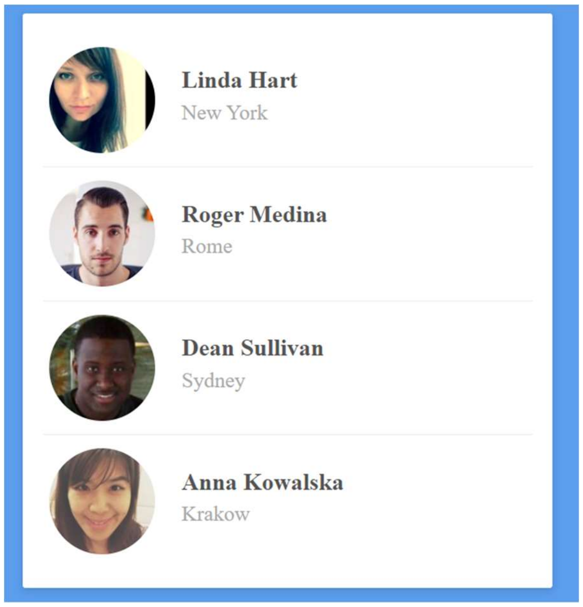
Lorem ipsum dolor sit, amet consectetur adipisicing elit. Deleniti, ducimus illo dolorem laboriosam libero placeat corporis qui tenetur. Impedit rerum doloremque magnam, earum maiores similique molestias ipsum, perspiciatis perferendis quidem corporis quo!

21 comments

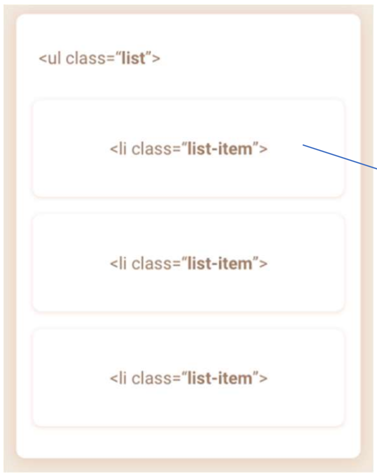
## Exercise 19.  Thumbnail List

Lists are used to display rows of information, such as a playlists, user details, or movies. Often, these lists contain images that we know as “thumbnails”. Aligning images with traditional methods can be a little tricky and inconsistent, but with Flexbox we can style a list component with minimal CSS.

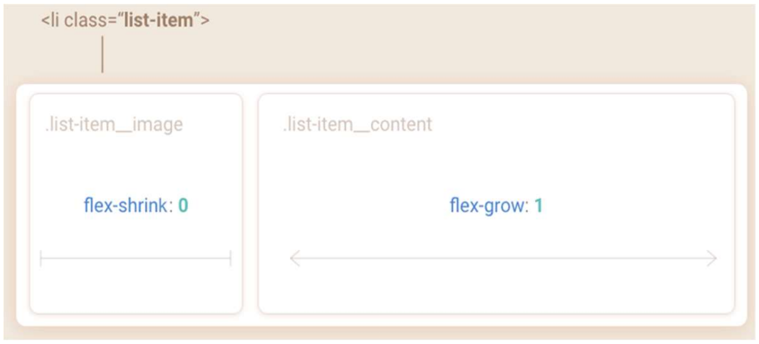
In section2/example4_sidebar   you can find file  index,html. Create style.css which give you list of thumnails.  Single thumnail consist with image, personal data and extra information.  Example of list you can see as



Tips. Main element your page



and



Exercise 20. Banner.

In section2/example5_banner you can find example banner. Browse the code, check how was made text effect. Modify the code to give a different transition effect. If you don't know about transition you can read about it for example [https://www.w3schools.com/css/css3_transitions.asp](https://www.w3schools.com/css/css3_transitions.asp) or test lihttps://cssgenerator.pl/transition-generator/