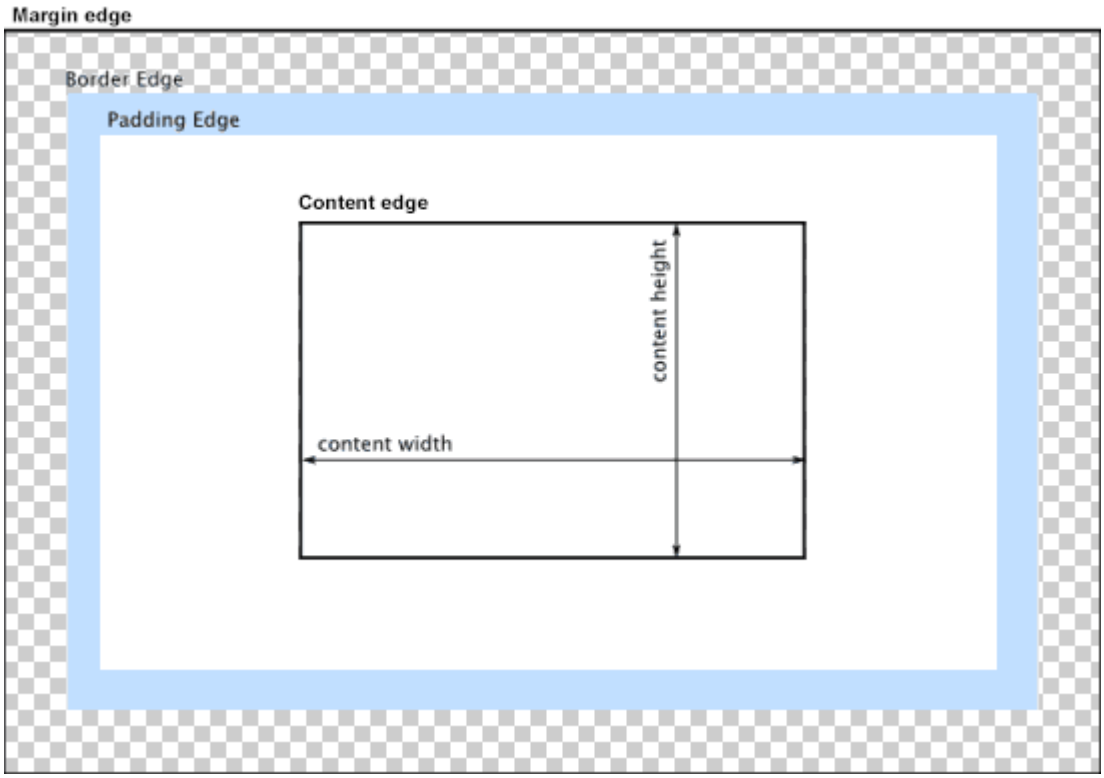
2016/2017

Using HTML and CSS for UI development

WEB APPLICATION DEVELOPMENT - LAB 3

1. CSS box model

In a document, each element is represented as a rectangular box.



In CSS, each of these rectangular boxes is described using the standard *box model*. This model describes the content of the space taken by an element. Each box has four edges: the **margin edge**, **border edge**, **padding edge**, and **content edge**.

The **content area** is the area containing the real content of the element.

The **padding area** extends to the border surrounding the padding. When the content area has a background, color, or image set on it, this will extend into the padding, which is why you can think of the padding as extending the content.

The space between the padding and the content edge can be controlled using the padding-top, padding-right, padding-bottom, padding-left and the shorthand padding CSS properties.

The **border area** extends the padding area to the area containing the borders. It is the area inside the *border edge*, and its dimensions are the *border-box width* and the *border-box height*. This area depends on the size of the border that is defined by the border-width property or the shorthand border.

The **margin area** extends the border area with an empty area used to separate the element from its neighbors. It is the area inside the *margin edge*, and its dimensions are the *margin-box width* and the *margin-box height*.

The size of the margin area is controlled using the margin-top, margin-right, margin-bottom, margin-left and the shorthand margin CSS properties.

2. Inline and block elements

Inline elements occupy only the space bounded by the tags that define the inline element, they wrap only their textual content.

In the next example, the span element (colored by a CSS rule) is an inline element.

HTML

```
<p>This <span>span</span> is an inline element; its background has been colored to display both the beginning and end of the inline element's influence</p>
```

CSS

```
span { background-color: #8ABB55; }
```

Result:

This span is an inline element; its background has been colored to display both the beginning and end of the inline element's influence.

With inline elements, we cannot specify the width and the height (they are automatically set depending on the content), nor the upper and lower margin (but we can change the left and the right margin, as well as the padding).

Examples of inline elements: `b, big, small, tt, code, var, kbd, em, a, img, span, sub, sup, button, input, label, select, textarea`.

In contrast to them, a block element occupies the entire space of its parent element (container), thereby creating a "block". A block element occupies the entire space of its parent element (container), thereby creating a "block".

HTML

```
<p>A paragraph</p> <p>This paragraph is a block-level element; its background has been colored to display the paragraph's parent element.</p> <p>Also a paragraph</p>
```

CSS

```
p { background-color: #8ABB55; }
```

Result:

A paragraph

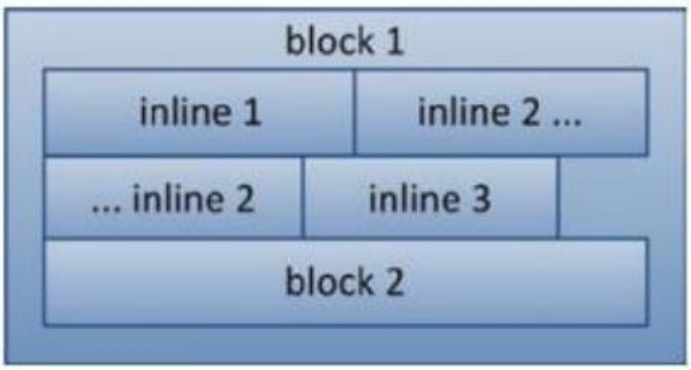
This paragraph is a block-level element; its background has been colored to display the paragraph's parent element.

A paragraph

With block elements we can change all properties of the css box model: width and height of the content, border, padding, margin. In case we haven't defined a width of a block element, the block element automatically fills the available horizontal space.

Examples of block elements: `address, article, aside, div, footer, form, h1 – h6, header, footer, main, nav, ol, ul, p, pre, section, table`.

The difference between inline and block elements is also shown in the following figure.



In case we need to stack elements horizontally (but require setting width, height, or all the margins), we can specify that an element is an **"inline-block"** element. This is done by using the css display property.

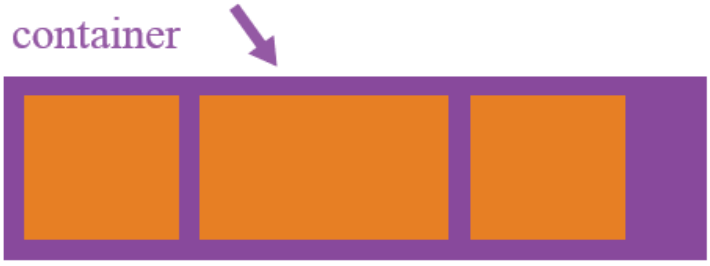
```
p { display: inline-block; }
```

3. Flex-box

The Flexbox Layout (Flexible Box) module (currently a W3C Last Call Working Draft) aims at providing a more efficient way to lay out, align and distribute space among items in a container, even when their size is unknown and/or dynamic (thus the word "flex").

The main idea behind the flex layout is to give the container the ability to alter its items' width/height (and order) to best fill the available space (mostly to accommodate to all kind of display devices and screen sizes). A flex container expands items to fill available free space, or shrinks them to prevent overflow.

Flex container is the container in which we place flex items.



We can turn any HTML element by using the `display:flex` CSS property:

```
.container {
  display: flex; /* or inline-flex */
}
```

Once we've specified an element as a flex container, we can also specify the direction in which the container items will be placed, by using the flex-direction property:

```
.container {
  flex-direction: row | row-reverse | column | column-reverse;
}
```

row (default): items are placed left to right; column: items are placed top to bottom.

Within flex container, we place flex items. Once an element is contained within a flex container, we can use the following properties:

justify-content (aligns the items if there's more space than necessary):

- **flex-start** (Items are positioned at the beginning of the container)
- **flex-end** (Items are positioned at the end of the container)
- **center** (Items are positioned at the center of the container)
- **space-between** (Items are positioned with space between the lines)
- **space-around** (Items are positioned with space before, between, and after the lines)

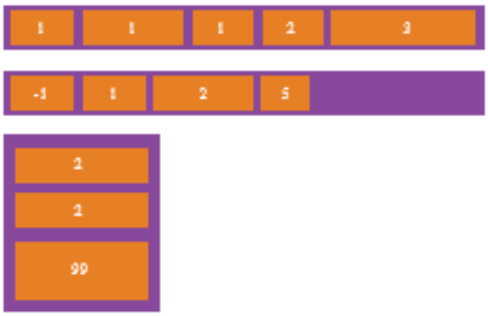
align-items (the default alignment for items inside the flexible container):

- stretch (Default. Items are stretched to fit the container)
- center (Items are positioned at the center of the container)
- flex-start (Items are positioned at the beginning of the container)
- flex-end (Items are positioned at the end of the container)
- baseline (Items are positioned at the baseline of the container)

flex-wrap (specifies whether the flexible items should wrap or not):

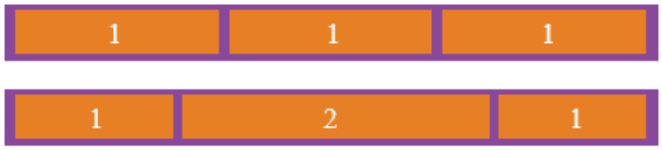
- nowrap
- wrap
- wrap-reverse

order (defines the order in which the flex items are laid out):




```
.item {
  order: <integer>;
}
```

flex-grow:




```
.item {
  flex-grow: <number>;
}
```

This defines the ability for a flex item to grow if necessary. It accepts a unitless value that serves as a proportion. It dictates what amount of the available space inside the flex container the item should take up.

If all items have flex-grow set to 1, every child will set to an equal size inside the container. If you were to give one of the children a value of 2, that child would take up twice as much space as the others. Of we set a value of 0, the item's dimensions won't be changed.

In addition, there's also the flex-shrink property which defines the ability for a flex item to shrink, if necessary.

flex-basis

Flex-basis defines the default size of an element before the remaining space is distributed. The main-size value makes it match the width or height, depending on which is relevant based on the flex-direction.

```
.item {
  flex-basis: <length> | auto; /* default auto */
}
```

Usually, we can write all these properties through a single property flex.

```
.item {
  flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]
}
```

For more info, visit:

- https://css-tricks.com/snippets/css/a-guide-to-flexbox/
- http://philipwalton.github.io/solved-by-flexbox/demos/holy-grail/
- http://www.sketchingwithcss.com/samplechapter/cheatsheet.html

4. Element positioning

Element positioning can also be done with CSS. We differentiate between the following properties:

- Static – the default value, the element is position in its current position in the document flow. Additional properties: top, right, bottom, left, and z-index are not applied.
- Relative – lays out the element in its current position in the document flow, but shifts it according to additional properties such as: top, right, bottom, left
- Absolute – takes the element out of the normal document flow and positions it relative to its closest positioned ancestor (or to the containing block). Absolutely positioned elements can have boxes, and we can adjust their position by using the top, left, right, bottom properties.
- Fixed – takes the element out of the normal document flow and positions it relative to the screen, the element stays in the same position when the screen is scrolled.

```
.container { position: static | relative | absolute | fixed }
```

Assignment

By using the given HTML, make the page look like in the following screenshot.