SOCIAL HACKERS ACADEMY

HTML-CSS

January 10th, 2018

Who we are

Aristeidis Bampakos (Aris)

Front End Web Developer @ Plexscape

Twitter: [@abampakos]

Github: https://github.com/bampakoa

George Sisko (Jkr)

.NET Web Developer @ Relational SA

Twitter: @GeorgeJkrr

Github: https://github.com/jkrgS

Agenda

- Responsive web development
- Units of measurement
- Positioning elements
- Floating / Clearing elements
- Flexbox design
- Developer tools / Inspector

Responsive web development

*“**Responsive web design** (**RWD**) is an approach to web design which makes web pages render well on a variety of devices and window or screen sizes. Recent work also considers the viewer proximity as part of the viewing context as an extension for RWD. Content, design and performance are necessary across all devices to ensure usability and satisfaction.”*

RWD explained

- A practice of building a website suitable to work on every device and every screen size, no matter how large or small, mobile or desktop
- Dynamically adapt to different browser and device viewports, changing layout and content along the way
- Viewers of public displays perceive the content of a display at different sizes according to their distance from the display

Why should you use it

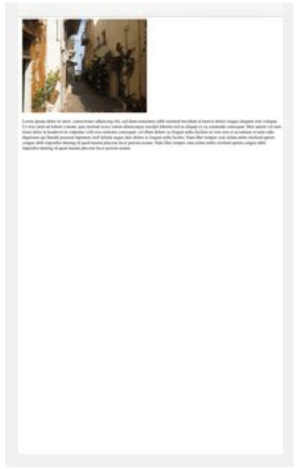
- No need to maintain separate websites for desktops and mobile phones
- Collect all social sharing links through a single URL address (SEO friendly)
- Provide the same user experience across all devices and screens

Demo time!

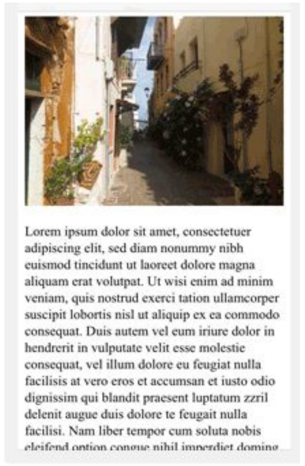
http://www.socialhackersacademy.org/

Viewport

- It is the area of a web page that is visible from the user. Varies according to the device.
- `<meta name="viewport" content="width=device-width, initial-scale=1.0">`



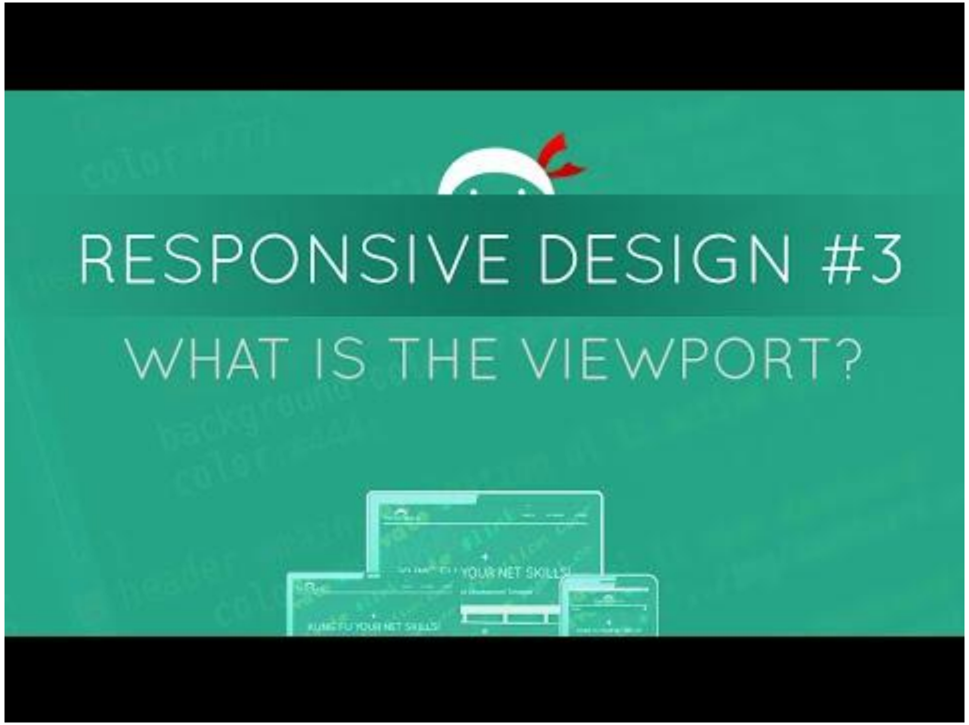
Without the viewport meta tag



Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming

With the viewport meta tag

Demo time!



Responsive elements

- Set **width** and **max-width** for images
    - If the width property is set to 100%, the image will be responsive and scale up and down

      ```
      <img src="myimage.jpg" style="width:100%;">
      ```

    - If the max-width property is set to 100%, the image will scale down if it has to, but never scale up to be larger than its original size

      ```
      <img src="myimage.jpg" style="max-width:100%;height:auto;">
      ```

Responsive elements

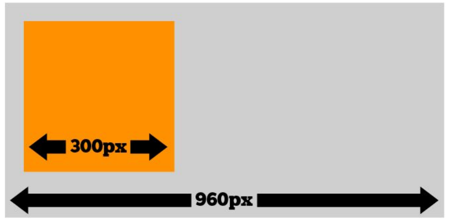
- Set **viewport width (vw)** for text
    - The text size will follow the size of the browser window

```
<h1 style="font-size:10vw">Hello World</h1>
```

Fluid layout

- Use percentage % rather than fixed measurement units such as pixels or inches
- A fluid grid is more carefully designed in terms of proportions
- All of the elements in the layout will resize their widths in relation to one another.
- Applicable to media such as images, videos and fonts.

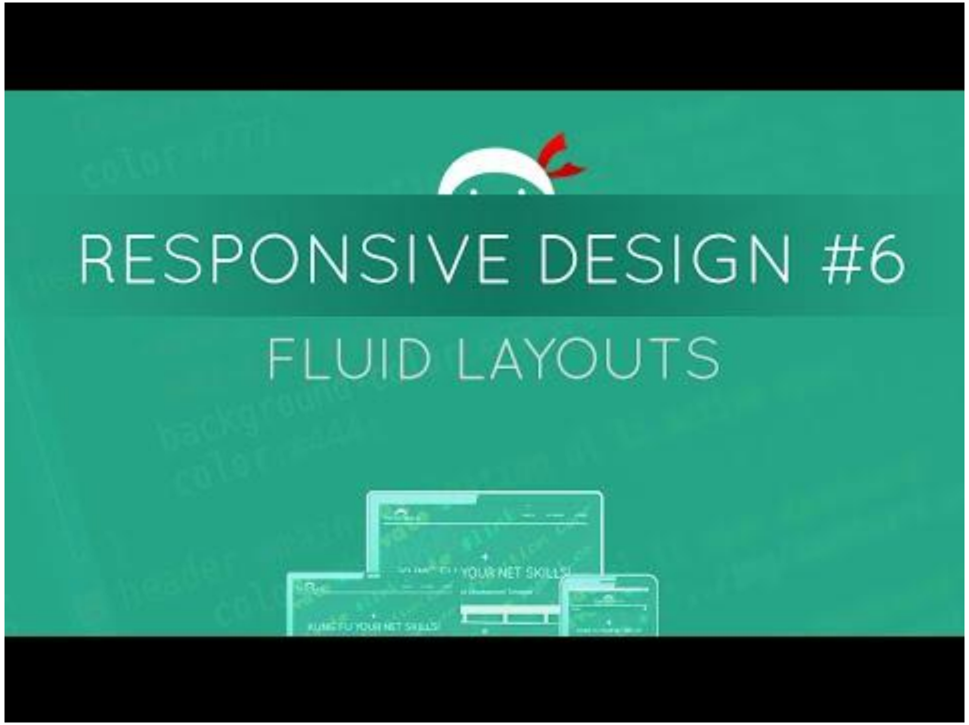
target / context = result



300px / 960px = 31.25%

*These measurements are not to scale.

Demo time!



Media queries

*“**Media Queries** is a CSS3 module allowing content rendering to adapt to conditions such as screen resolution (e.g. smartphone screen vs. computer screen). It became a W3C recommended standard in **June 2012**, and is a cornerstone technology of responsive web design (RWD)”*

Media queries explained

- Apply different CSS styles depending on the type of a device (such as a screen or content in print preview) or its specific characteristic (such as the width of the viewport)
- A useful tool due to the huge variety of connected devices
- Use **@media** rule to include a block of CSS properties when a certain condition is true

Media types

- Media types define the type of a device:
    - **all**: targets all devices
    - **print**: used for content viewed on a screen in print preview mode
    - **screen**: intended for color computer screens
    - **speech**: targets speech synthesizers
- The types that are most commonly used are **all** and **screen**.
- Usage: `@media type { ... }`

Media features

- Media features describe the specific characteristic of a browser, device or environment
- There is an extensive list of available characteristics here
- Range features are commonly used that are prefixed with **“min-”** or **“max-”** to define minimum or maximum constraints such as *min-width* or *max-height*.
- If a feature does not apply to a device, the styles of that feature will be ignored and never be applied
- Usage: `@media (feature) { ... }`

Breakpoints

- The point at which the content of your site will respond to provide the user with the best possible layout to consume the information
- Introduce them gradually
  - Design for the smallest viewport first
  - Expand the view at the point at which the design seems broken
  - Add a media rule
- Common device breakpoints
- Always design for mobile first

Complex media queries

- Need to create a media query that depends on multiple conditions
- Use logical operators:
    - **not**: inverts the meaning of a media query
    - **and**: combines a media feature with media type or other features
    - **only**: used in older browsers to avoid applying the rule
- Combine multiple media queries into a comma-separated list

Complex media queries

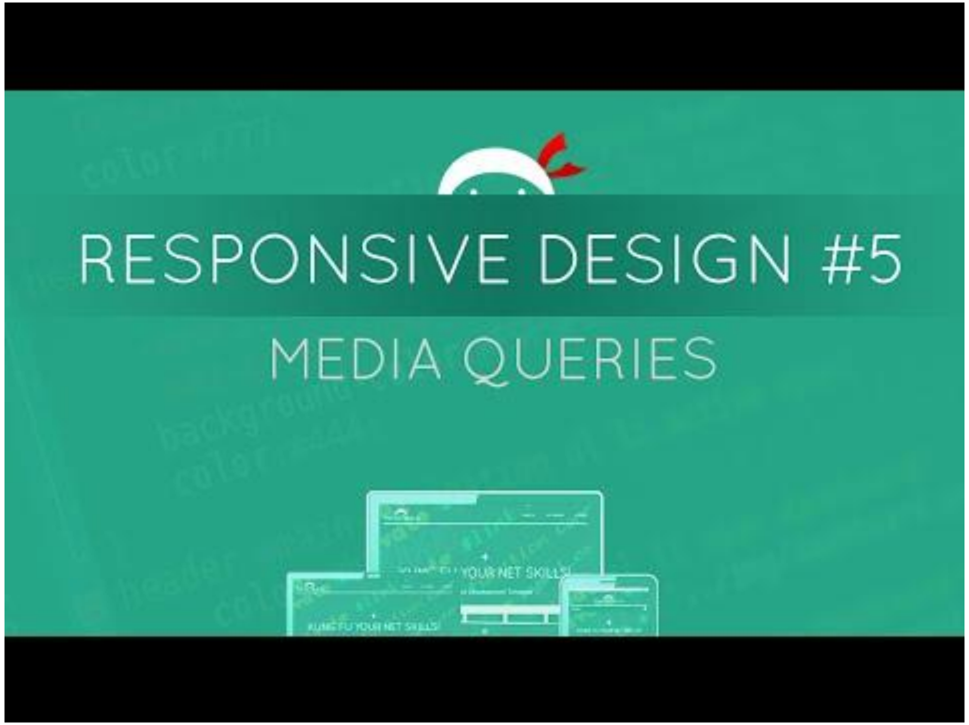
```
@media (min-width: 120px) and (orientation: landscape) { ... }

@media not screen { ... }

<link rel="stylesheet" media="only screen" href="modern-styles.css" />

@media (min-height: 680px), screen and (orientation: portrait) { ... }
```

Demo time!



Disadvantages

- Slow loading times due to unnecessary HTML/CSS files that need to be downloaded
- Extra time to create a responsive web site. Even more to convert an existing one
- Not compatible with old versions of IE

Measurement units

- Define the length of an element through properties such as width, margin and padding
- Combined with numbers to form the length of an element, e.g. 15px, 2em
- When the length is 0, the unit can be omitted
- Negative lengths are allowed in some cases
- There are two types of units: **absolute** and **relative**

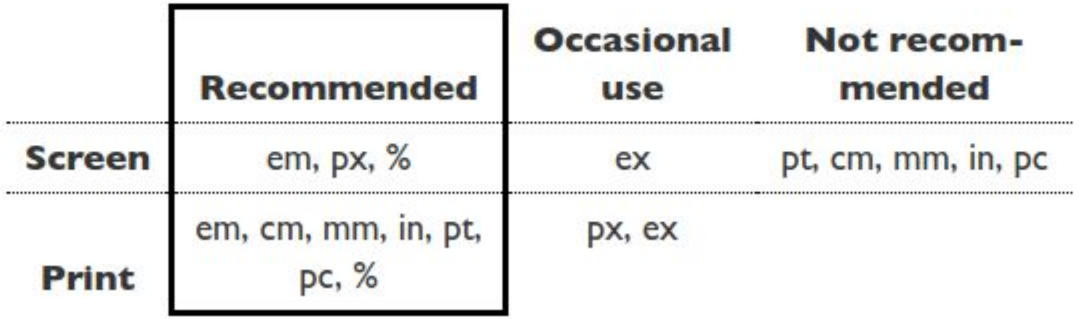
Absolute lengths

- They are fixed
- Elements appear exactly the size of the length
- Not recommended for use on screen
- They can be used when the output medium is known such as in a print layout
- Include physical units such as cm, mm, in and px.

Relative lengths

- Specify a length relative to another length property
- Scales better with both screen and print media types
- Most frequently used unit
- The most commonly used relative lengths are:
    - **%**: percentage, relative to the same property of the parent element
    - **em**: relative to font size of the element
    - **rem**: relative to font size of the root element
    - **vw**: relative to 1% of viewport width

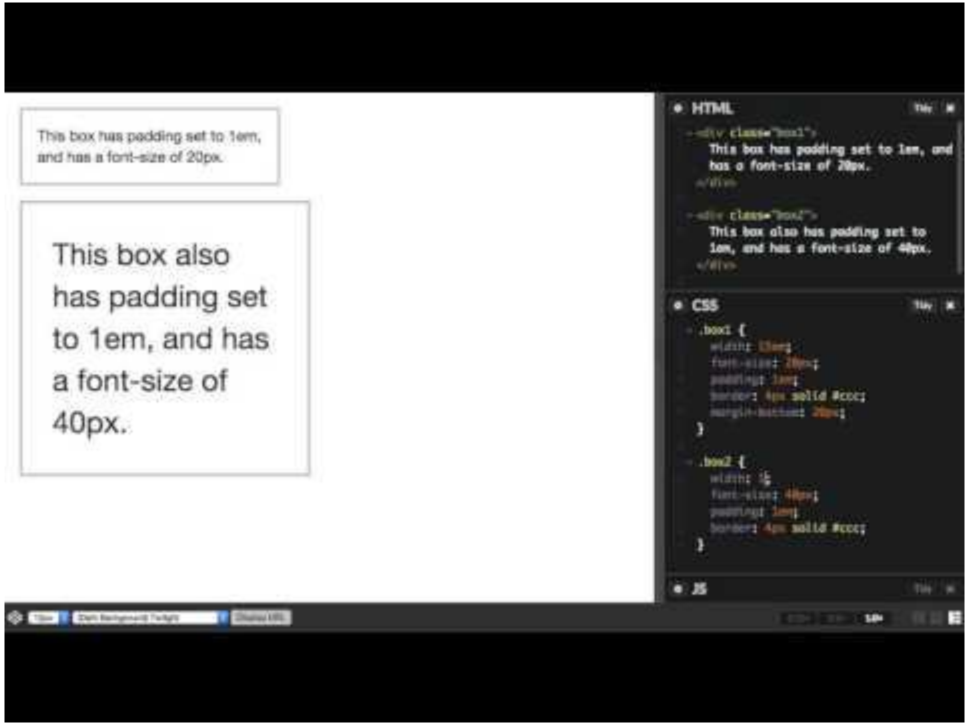
Recommended units per media type



	Recommended	Occasional use	Not recom-mended
Screen	em, px, %	ex	pt, cm, mm, in, pc
Print	em, cm, mm, in, pt, pc, %	px, ex	

https://www.w3.org/Style/Examples/007/units.en.html

Demo time!



This box has padding set to 1em, and has a font-size of 20px.

This box also has padding set to 1em, and has a font-size of 40px.

HTML

```html
<div class="box1">
  This box has padding set to 1em, and
  has a font-size of 20px.
</div>

<div class="box2">
  This box also has padding set to
  1em, and has a font-size of 40px.
</div>
```

CSS

```css
.box1 {
  width: 15em;
  font-size: 20px;
  padding: 1em;
  border: 4px solid #ccc;
  margin-bottom: 20px;
}

.box2 {
  width: 1
  font-size: 40px;
  padding: 1em;
  border: 4px solid #ccc;
}
```

JS

Positioning

“Positioning allows you to take elements out of the normal document layout flow, and make them behave differently, for example sitting on top of one another, or always remaining in the same place inside the browser viewport. This section will been explain the different position values, and how to use them.”

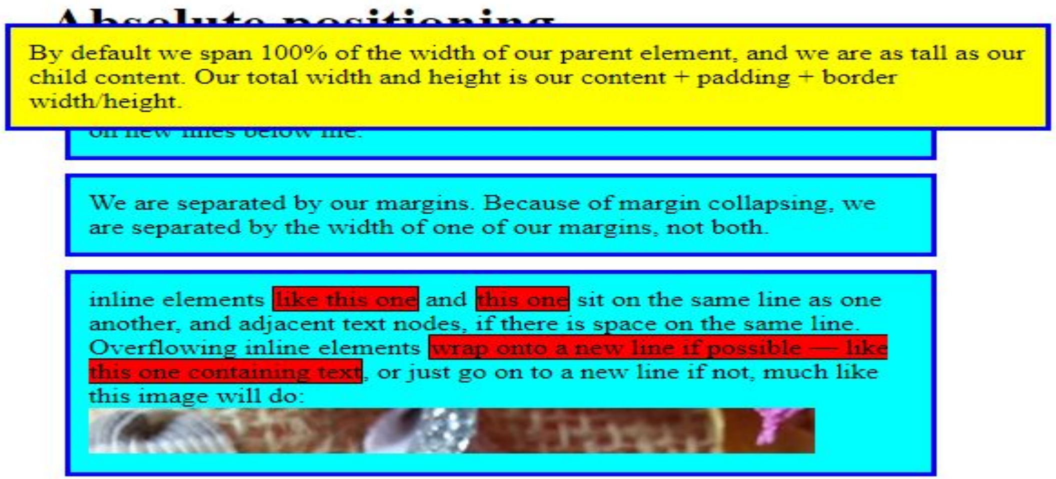
Positioning explained

- Positioning allow us to override the basic document flow behaviour, to produce interesting effects.
- There are a number of different types of positioning that you can put into effect on HTML elements. To make a specific type of positioning active on an element, we use the "position" property.

1. Absolute
2. Relative(Top, Bottom, Left, Right)
3. Fixed

Position: Absolute

We can create isolated UI features that don't interfere with the position of other elements on the page — for example popup information boxes and control menus, rollover panels, UI features that can be dragged and dropped anywhere on the page.



Absolute positioning

By default we span 100% of the width of our parent element, and we are as tall as our child content. Our total width and height is our content + padding + border width/height.

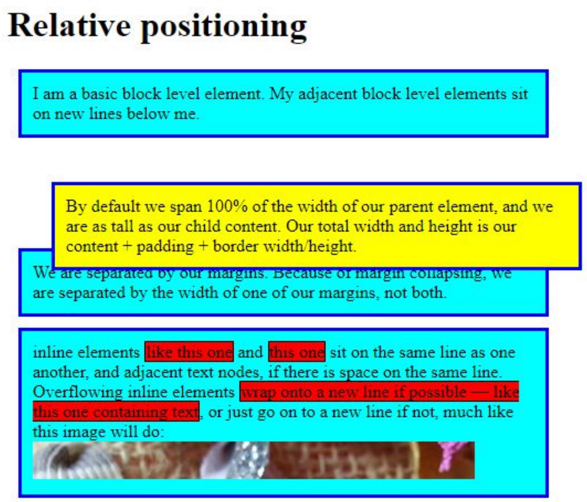
on new lines below me.

We are separated by our margins. Because of margin collapsing, we are separated by the width of one of our margins, not both.

inline elements like this one and this one sit on the same line as one another, and adjacent text nodes, if there is space on the same line. Overflowing inline elements wrap onto a new line if possible — like this one containing text, or just go on to a new line if not, much like this image will do:

Position: Relative

With Relative positioning you have to modify its final position , including making overlap other elements on the page. That elements are top, bottom, left and right.



Relative positioning

I am a basic block level element. My adjacent block level elements sit on new lines below me.

By default we span 100% of the width of our parent element, and we are as tall as our child content. Our total width and height is our content + padding + border width/height.

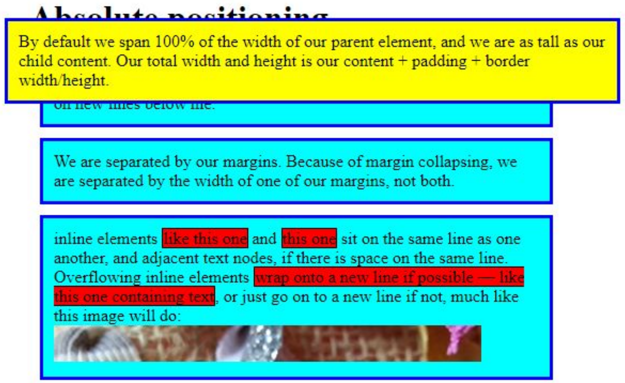
We are separated by our margins. Because of margin collapsing, we are separated by the width of one of our margins, not both.

inline elements like this one and this one sit on the same line as one another, and adjacent text nodes, if there is space on the same line. Overflowing inline elements wrap onto a new line if possible — like this one containing text, or just go on to a new line if not, much like this image will do:

Position: Fixed

This works in exactly the same way as *Absolute* positioning, with one key difference — whereas absolute positioning fixes an element in place relative to the *<html>* element or its nearest positioned ancestor, fixed positioning fixes an element in place relative to the browser viewport itself.

Absolute



Absolute positioning

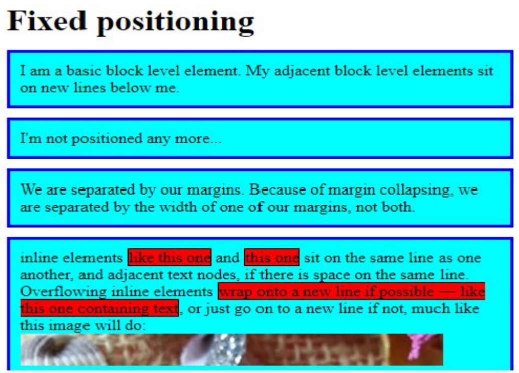
By default we span 100% of the width of our parent element, and we are as tall as our child content. Our total width and height is our content + padding + border width/height.

on new lines below me.

We are separated by our margins. Because of margin collapsing, we are separated by the width of one of our margins, not both.

inline elements like this one and this one sit on the same line as one another, and adjacent text nodes, if there is space on the same line. Overflowing inline elements wrap onto a new line if possible — like this one containing text, or just go on to a new line if not, much like this image will do:

Fixed



Fixed positioning

I am a basic block level element. My adjacent block level elements sit on new lines below me.

I'm not positioned any more...

We are separated by our margins. Because of margin collapsing, we are separated by the width of one of our margins, not both.

inline elements like this one and this one sit on the same line as one another, and adjacent text nodes, if there is space on the same line. Overflowing inline elements wrap onto a new line if possible — like this one containing text, or just go on to a new line if not, much like this image will do:

Demo time!

ABSOLUTE POSITION IN CSS
#9
CSS POSITIONING

RELATIVE POSITION IN CSS
#8
CSS POSITIONING

FIXED POSITION IN CSS
#10
Glue
CSS POSITIONING

Floating & Clearing

“The float property was introduced to allow you to implement simple layouts involving an image floating inside a column of text, with the text wrapping around the left or right of it. The kind of thing you might get in a newspaper layout.”

“The clear property. When you apply this to an element, it basically says "stop floating here" — this element and those after it in the source will not float, unless you apply a new float declaration to another element later on.”

Floating

- Involve an image floating inside a column of text
- Floats also are used very commonly these days to create entire web site layouts featuring multiple columns of information floated so they sit alongside one another (the default behaviour would be for the columns to sit below one another, in the same order as they appear in the source).

Clearing

- Float's sister property is clear. An element that has the clear property set on it will not move up adjacent to the float like the float desires, but will move itself down past the float.
- Clear has four valid values as well. ***Both*** is most commonly used, which clears floats coming from either direction. ***Left*** and ***Right*** can be used to only clear the float from one direction respectively.

Demo time!

FLOATING
ELEMENTS
IN CSS
#4
<html>
<p>
CSS POSITIONING

CLEARING
FLOATS
IN CSS
#5
<html>
CSS POSITIONING

Flexbox

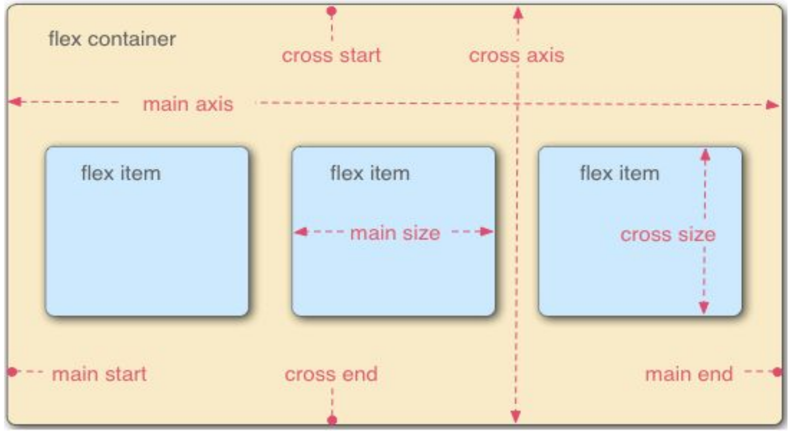
“A new technology, but with support now fairly widespread across browsers, Flexbox is starting to become ready for widespread use. Flexbox provides tools to allow rapid creation of complex, flexible layouts, and features that historically proved difficult with CSS.”

Flexbox explained

- Vertically centering a block of content inside its parent.
- Making all the children of a container take up an equal amount of the available width/height, regardless of how much width/height is available.
- Making all columns in a multiple column layout adopt the same height even if they contain a different amount of content.

Flex Model

- The **main axis** is the axis running in the direction the flex items are being laid out in (e.g. as rows across the page, or columns down the page.) The start and end of this axis are called the **main start** and **main end**.
- The **cross axis** is the axis running perpendicular to the direction the flex items are being laid out in. The start and end of this axis are called the **cross start** and **cross end**.
- The parent element that has display: flex set on it, is called the **flex container**.
- The items being laid out as flexible boxes inside the flex container are called **flex items** .



Demo time!

DOM/Style Inspector

- A tool to inspect the DOM, its elements and their styles
- To use it:
  - Open **Google Chrome**
  - Select the **Chrome Menu** ☰ -> **Tools** -> **Developer Tools**
  - Right click on any element of the page and select **Inspect Element.** Alternatively, you can use the **Select element tool** which is located on the top left corner
- Useful for debugging and modifying HTML attributes and CSS rules runtime

Elements tab

- Contains the actual DOM of the webpage
- Provides context menu for manipulating elements such as:
  - Add/Edit HTML attributes
  - Edit whole HTML blocks ☰
  - Hide/Delete elements
- **Styles** tab: Live-edit style property names and values
- **Computed** tab: Edit the current element's box model parameters
- **Event listeners** tab: provide hooks to the element’s DOM events
- **Properties** tab: displays the properties of the DOM element {https://developer.chrome.com/devtools}

Demo time!

http://www.socialhackersacademy.org/

Game is ON!

1. Go to https://kahoot.it/
2. Enter the **Game PIN** that you see on the screen
3. Pick up a nickname
4. Get ready!

Questions?

Thank you!

Aristeidis Bampakos

@abampakos

George Sisko

@GeorgeJkrr