BOOTCAMP

WELCOME TO CSS

goHarness India Pvt Ltd

goHARNESS
Live Your Potential

What is CSS?

CASCADING STYLE SHEETS

Cascading means pouring down in steps or adding in steps.

The process of combining several style sheets and resolving conflicts between them.

If we have a rule that is on the body tag it will "cascade" through every child tag. If we put a rule on any tag inside the body, it will adopt that rule, and so on.

What is CSS?

CASCADING STYLE SHEETS

CSS

```
p{font-size: 12pt;}
p{font-size: 14pt;}
<p>My Headline<p> What will be the font size?

body {
   background: blue; }
body {
   background: green; }
```

What will be the background colour?

What is CSS?

CASCADING STYLE SHEETS

Within CSS, all styles cascade from the top of a style sheet to the bottom, allowing different styles to be added or overwritten as the style sheet progresses.

All about CSS

IMPLEMENTATION OF CSS

Style information for a webpage can be defined in any of three different places, also known as **style levels**.

1. The preferred practice is to put style information in a separate file with a .css extension (external cascading style sheet).

```
<link rel="stylesheet" type="text/css" href="project.css">
```

All about CSS

IMPLEMENTATION OF CSS

2. Style information can be written within a <style> tag inside the webpage in the <head> tag. This is known as an internal style level.

Internal style level information within a webpage will override any style information provided by an external cascading style sheet.

All about CSS

IMPLEMENTATION OF CSS

3. All HTML5 tags have a style property that one can use to override any style information defined at either the page style level or in an external style sheet.
Using an HTML tag to define CSS information is referred to as an **inline style**.

Parent-level styles are overridden by page-level styles and page-level styles are overridden by tag-level styles is what is meant by **style sheets being cascading**.

ANATOMY OF A CSS RULE

CSS BOOTCAMP

goHARNESS
Live Your Potential

Anatomy of a CSS rule

CSS SYNTAX

```
p{
   color: blue;
   font-size: 24px;
}
```

p is a **Selector** with opening and closing curly brackets.

color: blue; is called **Declaration**, which has two parts

1. color - which is called **property** and
2. blue - which is called **value**

property-value is always separated by **:** and always terminated by **;**

CSS BOOTCAMP

goHARNESS

Live Your Potential

CODE ALONG EXERCISE

CSS BOOTCAMP

goHARNESS
Live Your Potential

ELEMENT, CLASS & ID SELECTORS

CSS BOOTCAMP

goHARNESS
Live Your Potential

Element Selector

CSS SELECTORS

```
p {
    color: blue;
    font-size: 24px;
}
```

p is an element sector

```
<p> Learn HTML & CSS </p>
<div> Xceedance Bootcamp </div>
```

What will be the outcome?

Class Selector

CSS SELECTORS

```
.blue {
    color: blue;
    font-size: 24px;
}
```

.blue is a class sector

```
<p class="blue"> Learn HTML & CSS </p>
<p> Learning CSS Selectors </p>
<div class="blue"> Xceedance Bootcamp </div>
```

What will be the outcome?

ID Selector

CSS SELECTORS

```
#blue {
    color: blue;
    font-size: 24px;
}
```

#blue is a ID sector

Element IDs should be unique within the entire document.

```
<p id="blue"> Learn HTML & CSS </p>
<p> Learning CSS Selectors </p>
<div class="blue"> Xceedance Bootcamp </div>
```

What will be the outcome?

Grouping Selectors

CSS SELECTORS

```
#blue, h1, .blue {
    color: blue;
    font-size: 24px;
}
```

Different selectors can be grouped together but they need to be separated by comma (,)

```
<p id="blue"> Learn HTML & CSS </p>
<h1> Learning CSS Selectors </h1>
<div class="blue"> Xceedance Bootcamp </div>
```

What will be the outcome?

Combining Selectors

CSS SELECTORS

```
p.blue {
    color: blue;
    font-size: 24px;
}
```

```
<p class="blue"> Learn HTML & CSS </p>
<h1> Learning CSS Selectors </h1>
<div class="blue"> Xceedance Bootcamp </div>
```

What will be the outcome?

Child Selectors

CSS SELECTORS

```
article > p {
    font-size: 24px;
}
```

Always read from right to left - Every **p element** that is a **Direct** child of **article element**

```
<article>
    <p> Learn HTML & CSS </p>
</article>
<article>
    <div> <p> Learn JavaScript </p> </div>
</article>
```

What will be the outcome?

CSS BOOTCAMP

goHARNESS
Live Your Potential

Descendant Selectors

CSS SELECTORS

```
article p {
    font-size: 24px;
}
```

Always read from right to left - Every **p element** that is **inside** (**at any level**) of **article element**

```
<article>
    <p> Learn HTML & CSS </p>
</article>
<article>
    <div> <p> Learn JavaScript </p> </div>
</article>
```

What will be the outcome?

Pseudo-class Selectors

CSS SELECTORS

pseudo-class is a keyword added to a selector that specifies a special **state** of the selected element(s).

```
selector:pseudo-class {
    .....
}
```

Few of pseudo-class selectors are

:link :visited :hover :active :nth-child(...)

```
a:link, a:visited {
    background-color: red;
}
```

CSS BOOTCAMP

goHARNESS
Live Your Potential

Pseudo-Elements

CSS ELEMENT

Pseudo-Element is a keyword added to a selector that lets you style a specific part of the selected element(s).

```
selector::pseudo-element {
  property: value;
}
```

As a rule, double colons (**::**) should be used instead of a single colon (**:**).

This distinguishes **pseudo-classes** from **pseudo-elements**.

Pseudo-Elements

CSS ELEMENT

```
/* The first line of every <p> element. */

p::first-line {
    color: blue;
    text-transform: uppercase;
}
```

Pseudo-Elements

CSS ELEMENT

```
<q>Some quotes,</q> he said, <q>are better than none.</q>

q::before {
  content: "«";
  color: blue;
}
q::after {
  content: "»";
  color: red;
}
```

CSS BOOTCAMP

goHARNESS
Live Your Potential

CODE ALONG EXERCISE

CSS BOOTCAMP

goHARNESS
Live Your Potential

CSS RULES

CSS BOOTCAMP

goHARNESS
Live Your Potential

CSS Rules

Origin Precedence
Merge
Inheritance
Specificity

Origin Precedence

CSS RULES

Last Declaration Wins - HTML is processed sequentially from top to bottom.

The declaration which is closest to the targeted element wins.

Merge

CSS RULES

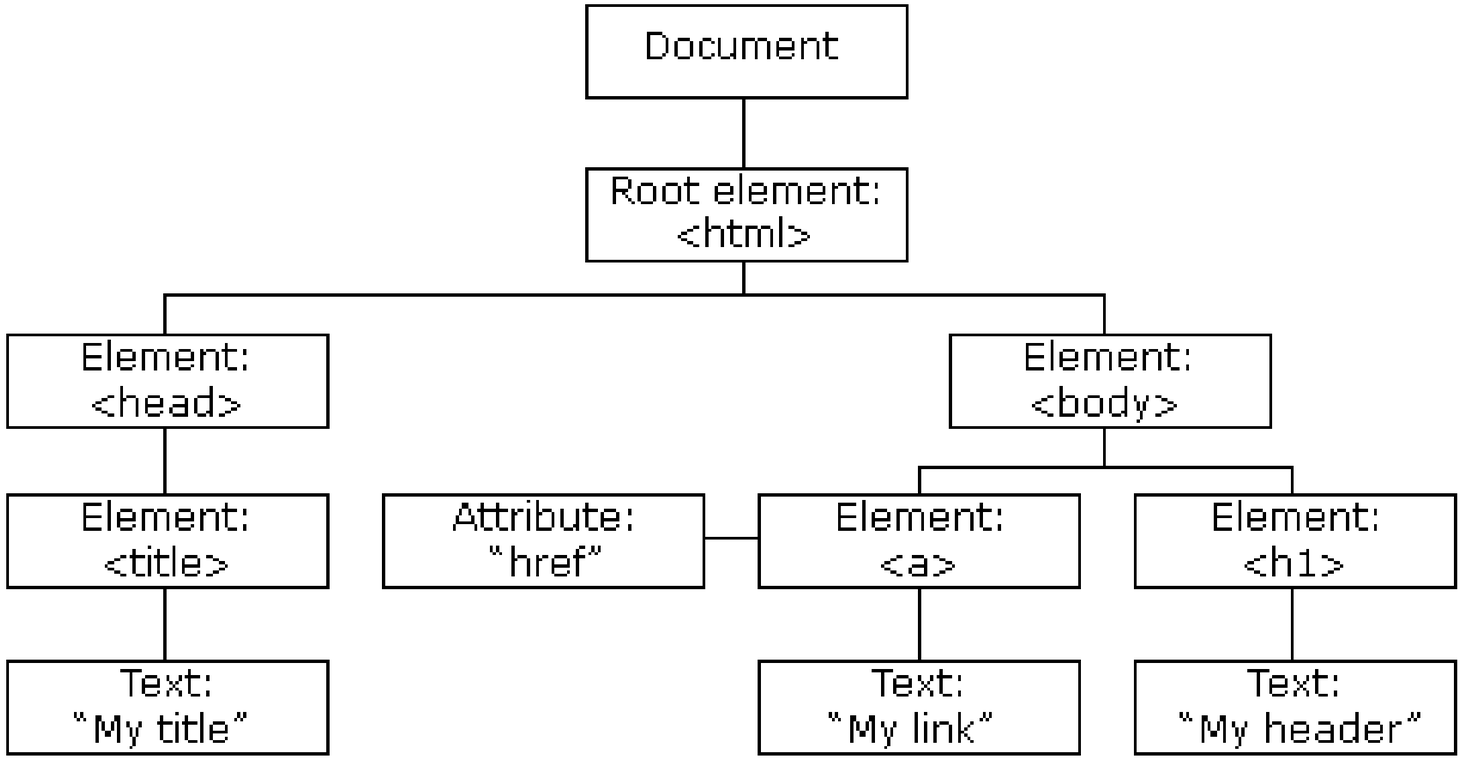
When different CSS declarations do not conflict, that is, they still target the same element, but the CSS properties with which they target that element are different, there's even a simpler rule and that is **That Declarations Merge.**

```
p {
    color: red;
}
p {
    font-size: 24px;
}
```

Inheritance

CSS RULES

DOM - Document Object Model



CSS BOOTCAMP

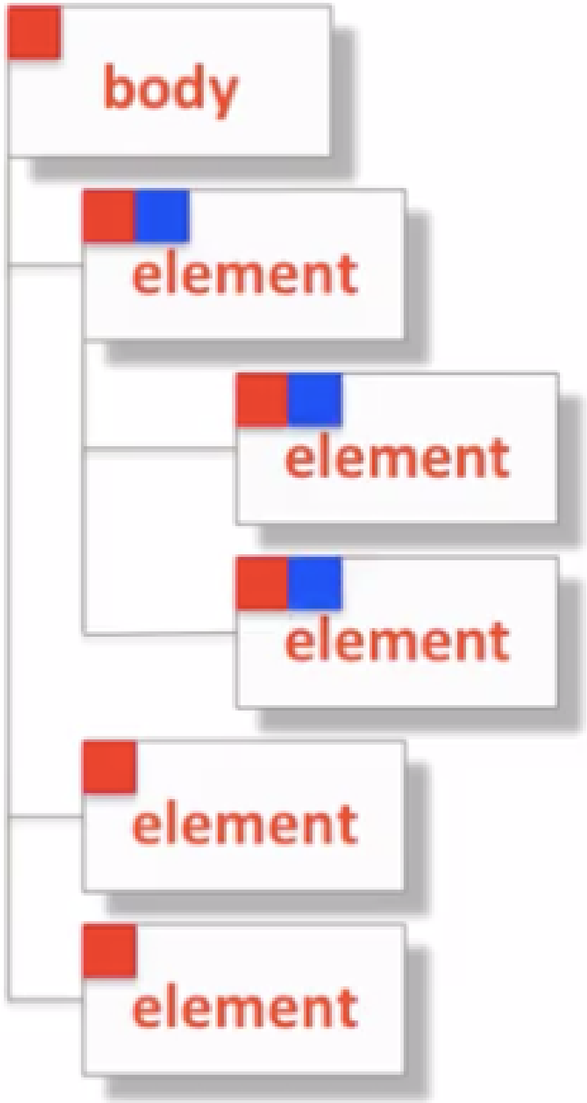
goHARNESS
Live Your Potential

Inheritance

CSS RULES

DOM

Document Object Model



CSS BOOTCAMP

goHARNESS
Live Your Potential

Specificity

CSS RULES

Specificity rule is that **most specific selector combination wins**

The selectors with the higher score would be considered the most specific.

go HARNESS
Live Your Potential

Specificity

CSS RULES

SICE - (especially in gambling) The six on a dice - Oxford Dictionaries Meaning

Style Attribute

Specificity: 1,000

ID Selector

Specificity: 100

Class Selector

Specificity: 10

Element Selector

Specificity: 1

Specificity Score

CSS RULES

```
<h2 style="color:red;"> ... </h2>
```

	S	I	C	E
Specificity Score				

```
div p { font-size: 24px; }
```

	S	I	C	E
Specificity Score				

CSS BOOTCAMP

goHARNESS
Live Your Potential

Specificity Score

CSS RULES

```
h2 #color { background-color: blue;}
```

	S	I	C	E
Specificity Score				

```
div.mainHeader p { font-size: 24px; }
```

	S	I	C	E
Specificity Score				

CSS BOOTCAMP

goHARNESS

Live Your Potential

BOX MODEL

CSS BOOTCAMP

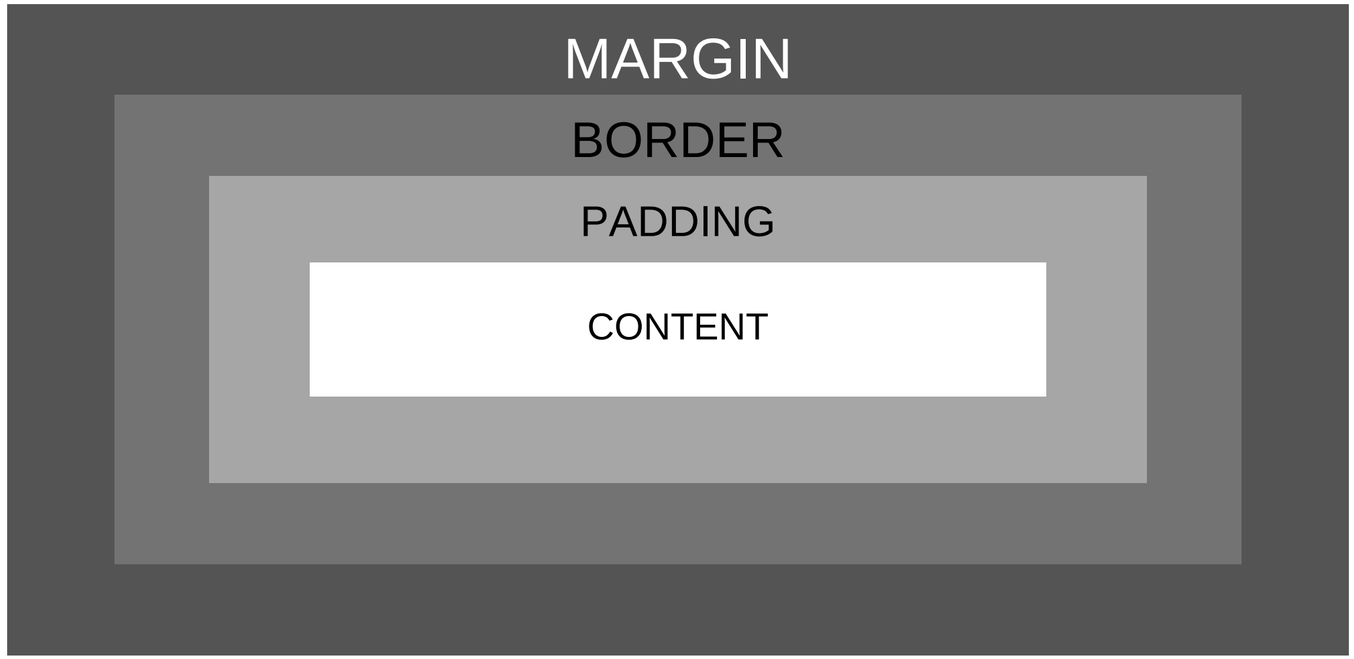
goHARNESS
Live Your Potential

Box Model

ALL HTML ELEMENTS CAN BE CONSIDERED AS BOXES.

The CSS box model is essentially a box that wraps around every HTML element. It consists of: **margins, borders, padding,** and the **actual content.**

Box Model

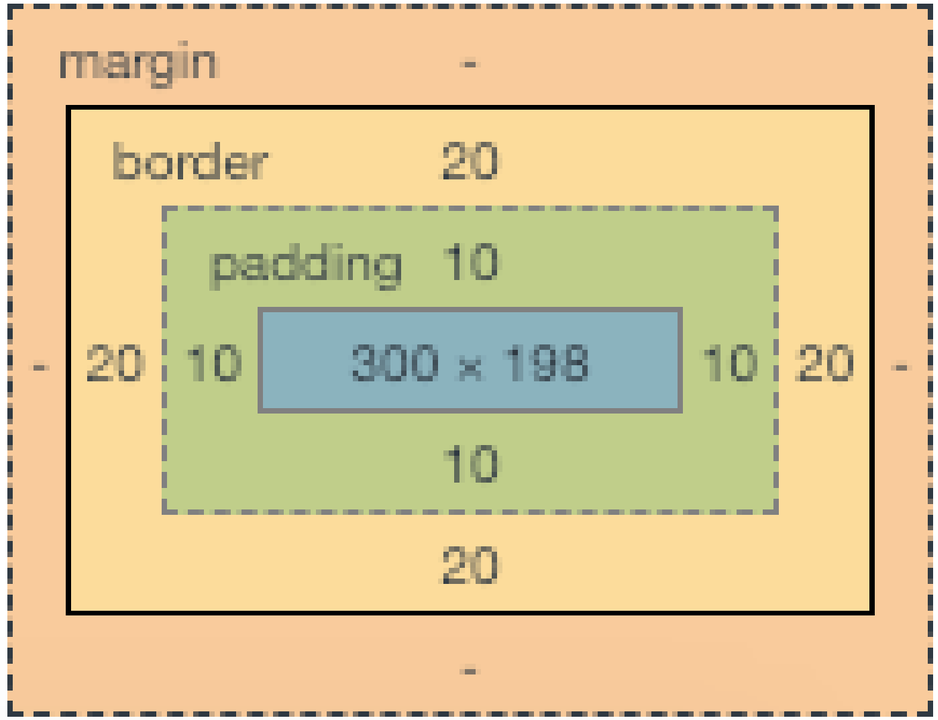


CSS BOOTCAMP

goHARNESS
Live Your Potential

box-sizing property is set to **content-box** (default)

Box Model



box-sizing: content-box;

```
#box {
        width: 300px;
        background-color: blue;
        color: white;
        margin: 0px;
        border: 20px solid red;
        padding: 10px 10px 10px 10px;
    }
```

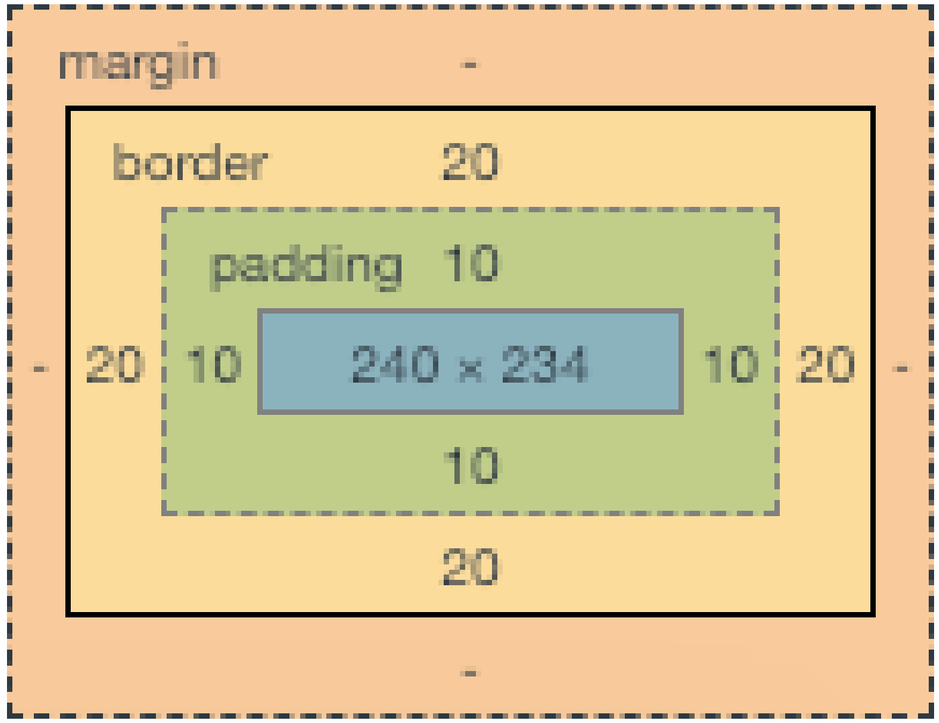
CSS BOOTCAMP

go HARNESS

Live Your Potential

Box Model

box-sizing: border-box;




```
#box {
        box-sizing: border-box;
        width: 300px;
        background-color: blue;
        color: white;
        margin: 0px;
        border: 20px solid red;
        padding: 10px 10px 10px 10px;
    }
```

CSS BOOTCAMP

go HARNESS

Live Your Potential

Box Model

Very Important

box sizing is one of those CSS properties that is not inherited. You can't set it on the parent element and then expect that the child elements will inherit that property.

CSS Selector

Universal Selector - Star

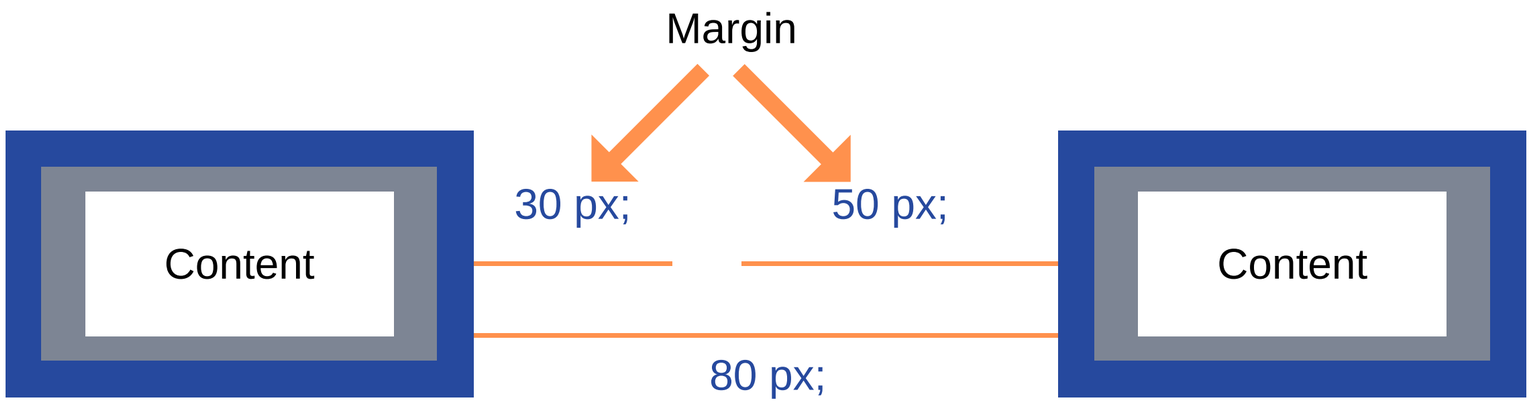
```
* {
    box-sizing: border-box;
}
```

Star Selector - select every element there is and apply these particular CSS properties to them.

Box Model Behaviour

Horizontal Content Placement

Margins that are left to right are **cumulative**.



Cumulative Margin

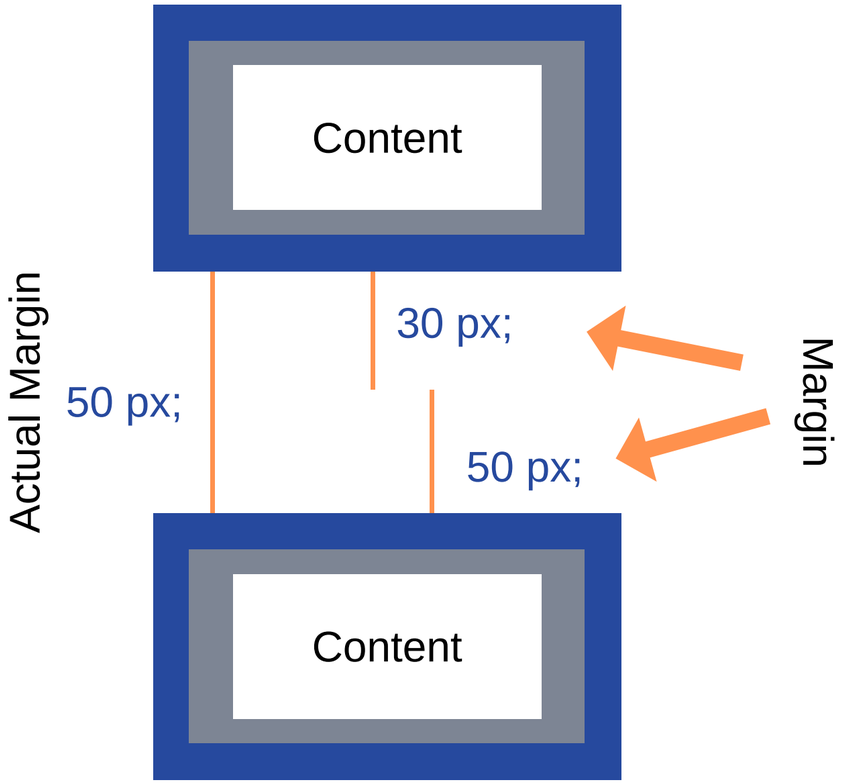
CSS BOOTCAMP

goHARNESS
Live Your Potential

Box Model Behaviour

Vertical Content Placement

Margins collapse and larger margin will win



CSS BOOTCAMP

goHARNESS
Live Your Potential

POSITIONING ELEMENTS

CSS BOOTCAMP

goHARNESS
Live Your Potential

Positioning

Floating Elements

When we float elements, the browser takes them out of the regular document flow.

When it comes to floated elements, the margins never collapse.

For browser to resume the regular document flow. We have to use the clear property.

clear: right or left or both;

CODE ALONG EXERCISE – SIMPLE FLOAT

CSS BOOTCAMP

goHARNESS
Live Your Potential

Positioning

Static Positioning

Static positioning is the default that every element gets in a normal document flow

```
<p id="staticPosition"> ... </p>

#staticPosition {
  position: static;
  background: blue;
}
```

Nothing will happen only paragraph colour will change to blue.

Positioning

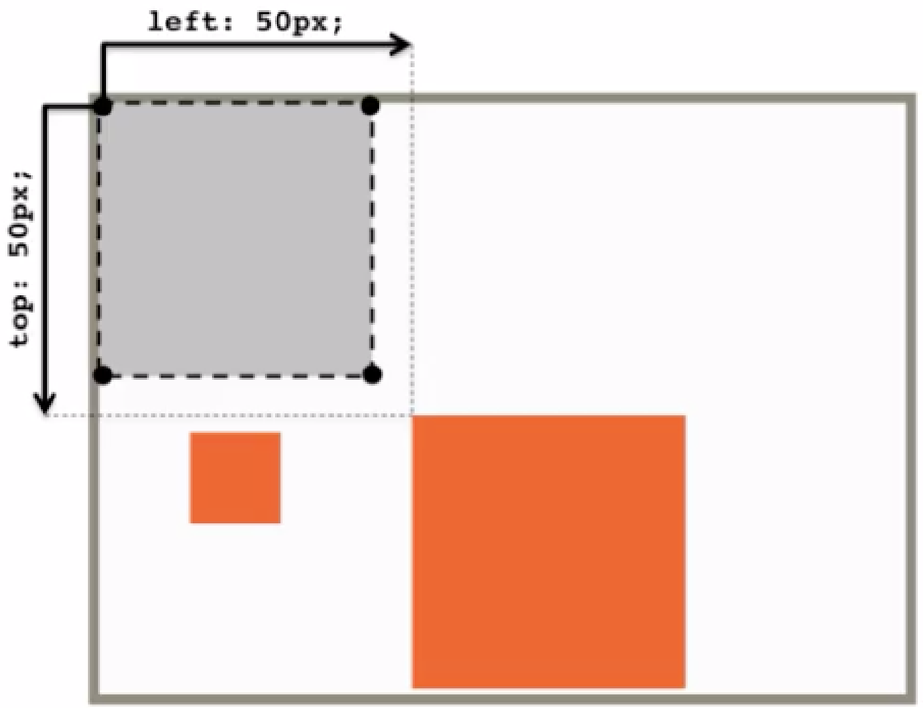
Relative Positioning

When we apply position relative on an element. The element is **positioned relative** to its **position** in the **normal document flow**.

Most Important to Remember

The important thing to know about **Relative Position** is that, the **element that is set to relative positioning** is **NOT** taken out of normal document flow.

Positioning



Relative Positioning

```
<p id="relativePosition"> ... </p>

#relativePosition {
  position: relative;
  top: 50px; /* From top 50px; */
  left: 50px; /* From left 50px; */
}
```

The original space for that element still remains and the originally laid out elements around that element still remain exactly the same, because they think the element is still sitting in its original spot.

Positioning

Absolute Positioning

Absolute positioning is that all offsets, top, bottom, left, right, are all relative to the position of the **nearest ancestor** which has positioning set on it other than static.

In other words, **Ancestor Element** has to have its positioning set other than static (relative), and then the absolute positioning will actually start working.

Positioning

Absolute Positioning

By default, **HTML element** is the only element that has **non-static positioning** set on it. And it's actually set to **relative**.

The element is **TAKEN OUT** of its normal document flow, if its positioning is set to **absolute**.

Most Important to Remember

Offsetting the relative container element, offsets its inner contents as well

Positioning

Z-index

"z-index" is a reference to the z-axis.
z-index values affect where positioned elements sit on that axis; positive values move them higher up the stack, and negative values move them lower down the stack.

By default, positioned elements all have a z-index of auto, which is effectively 0.
z-index: 1;
positive values move element up and negative down.

POSITIONING ELEMENTS

–

FLEXBOX

CSS BOOTCAMP

goHARNESS

Live Your Potential

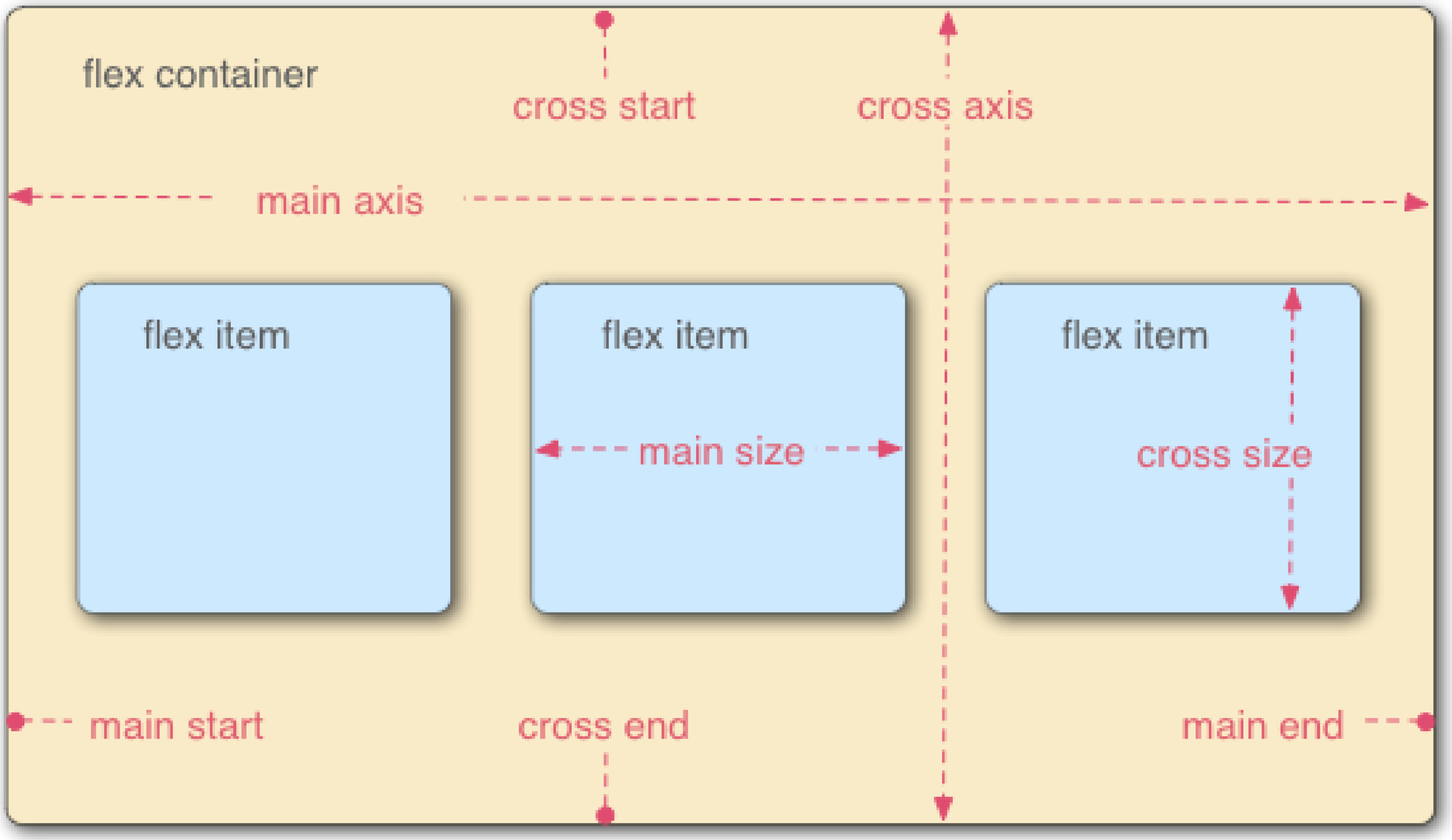
Positioning

Flexbox

Flexbox is a one-dimensional layout method for laying out items in rows or columns.

Example

```
section {
  display: flex;
}
```


flex container
cross start
cross axis
main axis
flex item
flex item
main size
flex item
cross size
main start
cross end
main end

CSS

Positioning

Flexbox Model

The **main axis** is the axis running in the direction the flex items are being laid out in.

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**.

Positioning

Flexbox Model

The **parent element** that has display: flex set on it (the <section> in our example) is called the **flex container**.

The items being laid out as flexible boxes inside the flex container are called **flex items** (the <article> elements in our example).

Positioning

Flexbox Model - Direction

Flexbox provides a property called flex-direction that specifies what direction the main axis runs in (what direction the flexbox children are laid out in) — **by default this is set to row**, which causes them to be laid out in a row

flex-direction: column; or row (default)

Positioning

Flexbox Model - Wrapping

When we have a fixed width or height in our layout, then eventually our flexbox children will overflow their container, breaking the layout. We can fix this by adding

flex-wrap: wrap;

on the flex container i.e., <section>

Positioning

Flexbox Model - Alignment

align-items controls where the flex items sit on the **cross axis**.

align-items: center;
By default, the value is stretch

We can also have values like flex-start and flex-end, which will align all items at the start and end of the cross axis respectively.

Positioning

Flexbox Model - Alignment

justify-content controls where the flex items sit on the **main axis**.

The default value is flex-start, which makes all the items sit at the start of the **main axis**. flex-end to make them sit at the end.

center will make the flex items sit in the center of the main axis.

There is another value, space-between, it doesn't leave any space at either end.

FONTS

CSS BOOTCAMP

goHARNESS
Live Your Potential

Fonts

F

Sans-serif

F

Serif

CSS BOOTCAMP

goHARNESS

Live Your Potential

Generic Font - Family

Serif - Serif fonts have small lines at the ends on some characters

Sans-serif - "Sans" means **without** - these fonts do not have the lines at the ends of characters

Monospace - All monospace characters have the same width

Fonts

Generic Font - Family Usage

```
p {
  font-family: "Times New Roman", Times, serif;
}
```

When we have Font on computer

```
@font-face {
  font-family: "Roboto";
  src: url("Roboto.ttf");
}
```

```
body { font-family: 'Roboto', sans-serif;}
```

Fonts

Google Font Usage

Website: https://fonts.google.com/

Embed this code into the <head> element of HTML document

```
<link href="https://fonts.googleapis.com/css?family=Roboto" rel="stylesheet">
```

CSS line on the element using required Font

```
p {
    font-family: 'Roboto', sans-serif;
}
```

Fonts

Google Font Usage

Import Google Fonts

Embed this code into the <head> element of HTML document

```
<style>
@import url('https://fonts.googleapis.com/css?family=Roboto');
</style>
```

CSS line on the element using required Font

```
p {
    font-family: 'Roboto', sans-serif;
}
```

FONT – SIZE

CSS BOOTCAMP

goHARNESS
Live Your Potential

Fonts Size

Pixels

A px value is **static**.

```
h1 {
 font-size: 40px;
}
h2 {
 font-size: 30px;
}
p {
 font-size: 14px;
}
```

Fonts Size

Ems

The size of an em value is **dynamic**.

When defining the font-size property, an **em** is equal to the size of the font that applies to the **parent** of the element in question.

If you haven't set the font size anywhere on the page, then it is the **browser default**, which is often **16px**.

So, by default 1em = 16px, and 2em = 32px.

Fonts Size

Ems

If we set a font-size of 20px on the body element, then 1em = 20px and 2em = 40px.

Note that the value 2 is essentially a **multiplier** of the **current em size**.

Fonts Size

Rem

CSS3 introduced Rem, which stands for "**root em**"

The em unit is relative to the font-size of the parent, which causes the **compounding issue**.

The rem unit is relative to the root—**html element**. That means that we can define a single font size on the html element and define all rem units to be a percentage of that.

Fonts Size

Rem

Setting value on the **Root HTML Element**

```
html { font-size: 16px; }
```

Setting **Relative** values wherever required

```
body { font-size: 1.5rem; } /* =24px */
h1   { font-size: 2rem; } /* =32px */
```

RESPONSIVE WEB DESIGN

CSS BOOTCAMP

goHARNESS
Live Your Potential

Viewport

Setting The Viewport

HTML5 introduced a method to let web designers take control over the viewport, through the <meta> tag.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

width=device-width it sets the width of the page to follow the **screen-width of the device** (which will vary depending on the device).

Viewport

Setting The Viewport

HTML5 introduced a method to let web designers take control over the viewport, through the <meta> tag.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

width=device-width it sets the width of the page to follow the **screen-width of the device** (which will vary depending on the device).

Viewport

Setting The Viewport

initial-scale=1.0 it sets the initial zoom level when the page is first loaded by the browser.

MEDIA QUERIES

CSS BOOTCAMP

goHARNESS
Live Your Potential

MEDIA QUERIES

Responsive Web Design

The **@media** rule is used in media queries to apply different styles for different media types/devices.

Media queries can be used to check many things, such as:

1. Width and height of the viewport
2. Width and height of the device
3. Orientation (is the tablet/phone in landscape or portrait mode?)
4. Resolution

MEDIA QUERIES

Breakpoints

CSS breakpoints are points where the website content responds according to the device width, allowing you to show the best possible layout to the user.

CSS breakpoints are also called **media query breakpoints**, as they are used with media query.

MEDIA QUERIES

Breakpoints

Custom, iPhone - min-width: **320px**

Extra Small Devices, Phones - min-width: **480px**

Small Devices, Tablets - min-width : **768px**

Medium Devices, Desktops - min-width : **992px**

Large Devices, Wide Screens - min-width : **1200px**

MEDIA QUERIES

Responsive Web Design

Media Query Syntax

Media Feature (resolves to true or false)

```
@media (max-width: 767px) {
  p {
    color: blue;
  }
}
```

If TRUE, styles within curly braces apply.

CSS BOOTCAMP

goHARNESS
Live Your Potential

MEDIA QUERIES

Responsive Web Design

Media Query Syntax

```
@media only screen
and (min-device-width: 320px)
and (max-device-width: 767px)
and (orientation: portrait) {
    p {
    }
}
```

CSS BOOTCAMP

goHARNESS
Live Your Potential

MEDIA QUERIES

Responsive Web Design

Media Query Common Logical Operator

Device width within a range - **And Operator**

```
@media only screen
and (min-device-width: 768px)
and (max-device-width: 991px) { ... }
```

Comma is equivalent to **OR Operator**

```
@media only screen
and (max-device-width: 767px) ,
and (min-device-width: 992px) { ... }
```

CSS BOOTCAMP

goHARNESS
Live Your Potential

MEDIA QUERIES

Responsive Web Design

Most Common Mistakes

@media only screen
and (min-device-width: 480px)
and (max-device-width: **767px**) { ... }

Careful not to overlap Range Boundaries or Breakpoints

@media only screen
and (min-device-width: **768px**)
and (max-device-width: 991px) { ... }

CSS BOOTCAMP

goHARNESS
Live Your Potential

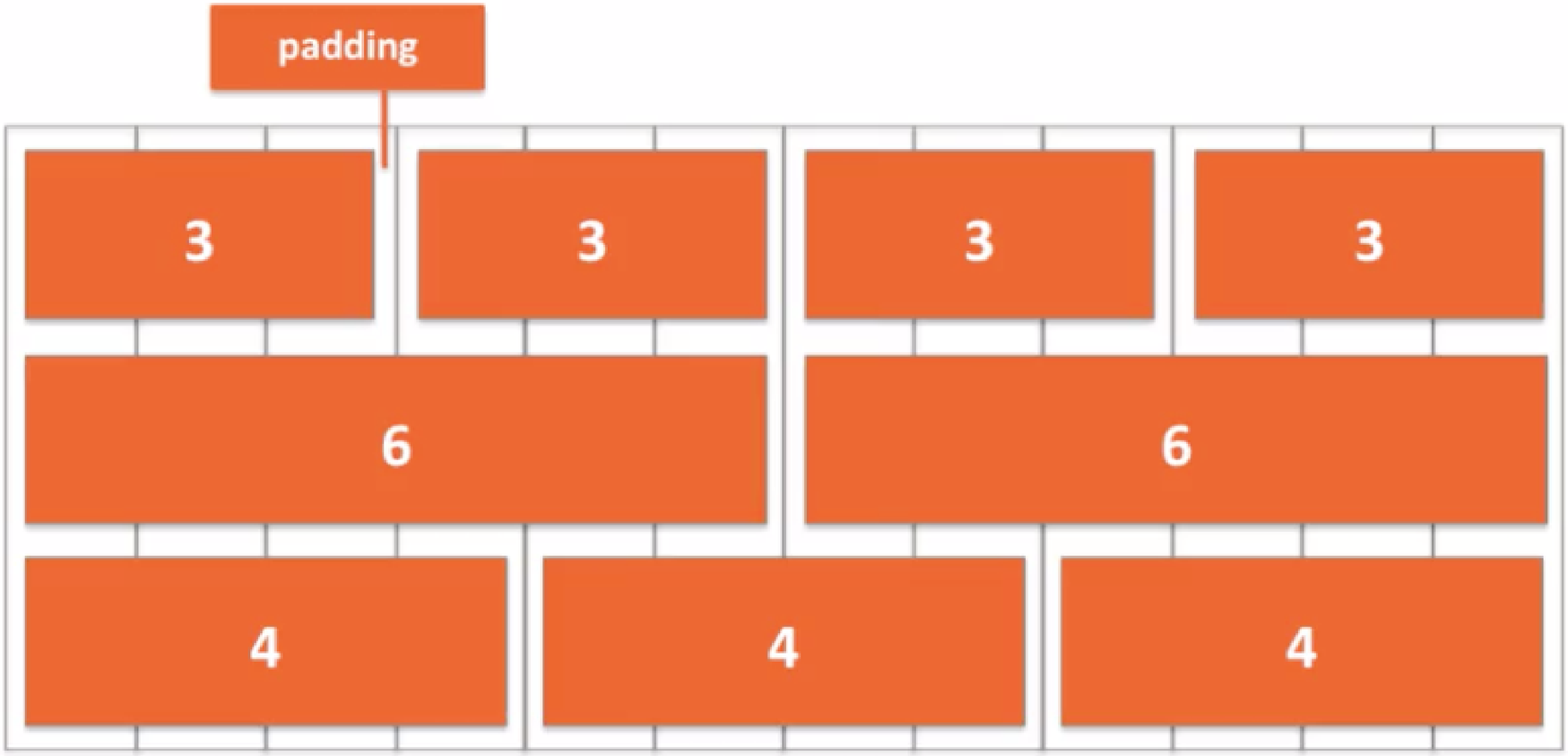
CODE ALONG EXERCISE – MEDIA QUERIES

CSS BOOTCAMP

goHARNESS
Live Your Potential

Responsive Layout

12 Column Grid

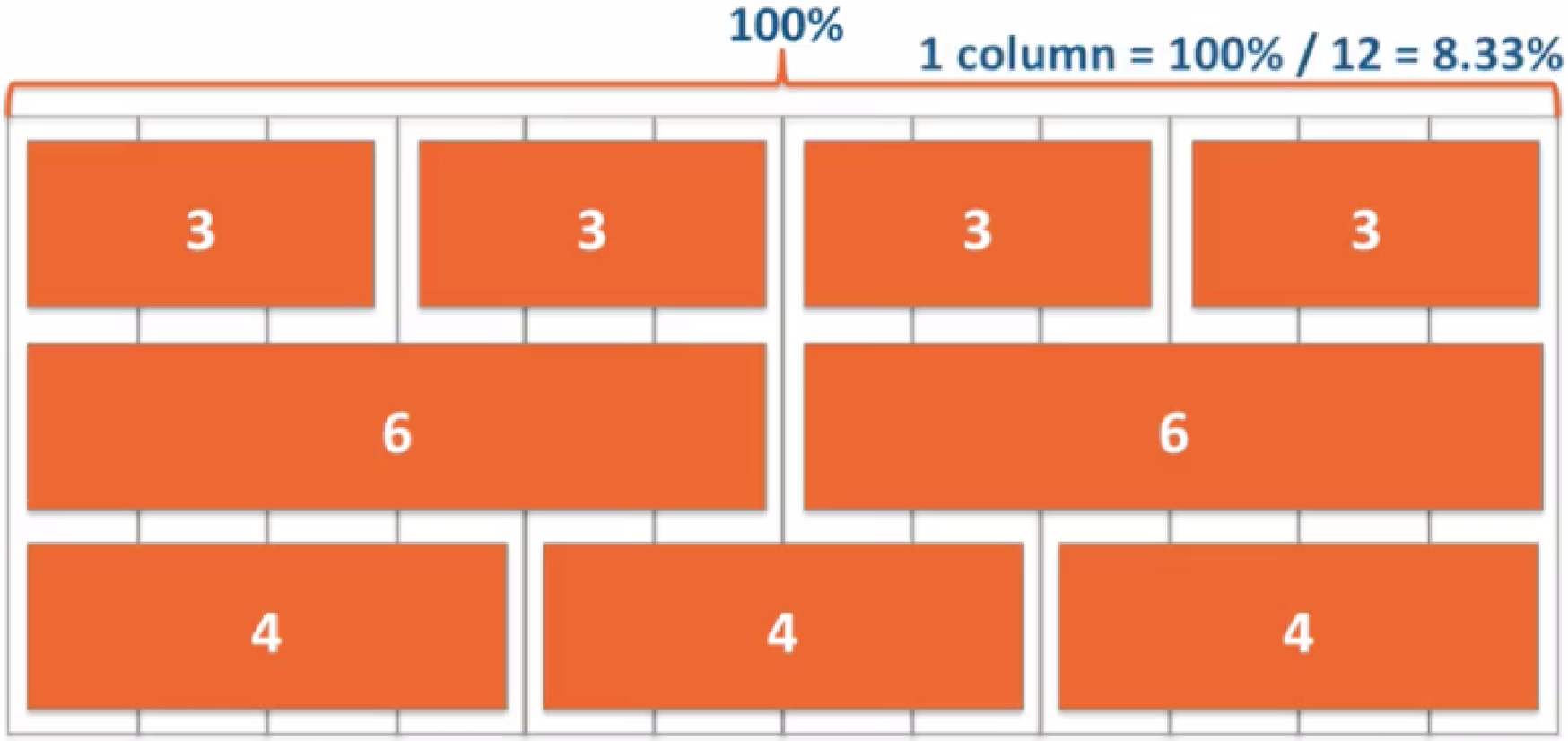


CSS BOOTCAMP

goHARNESS
Live Your Potential

Responsive Layout

12 Column Grid

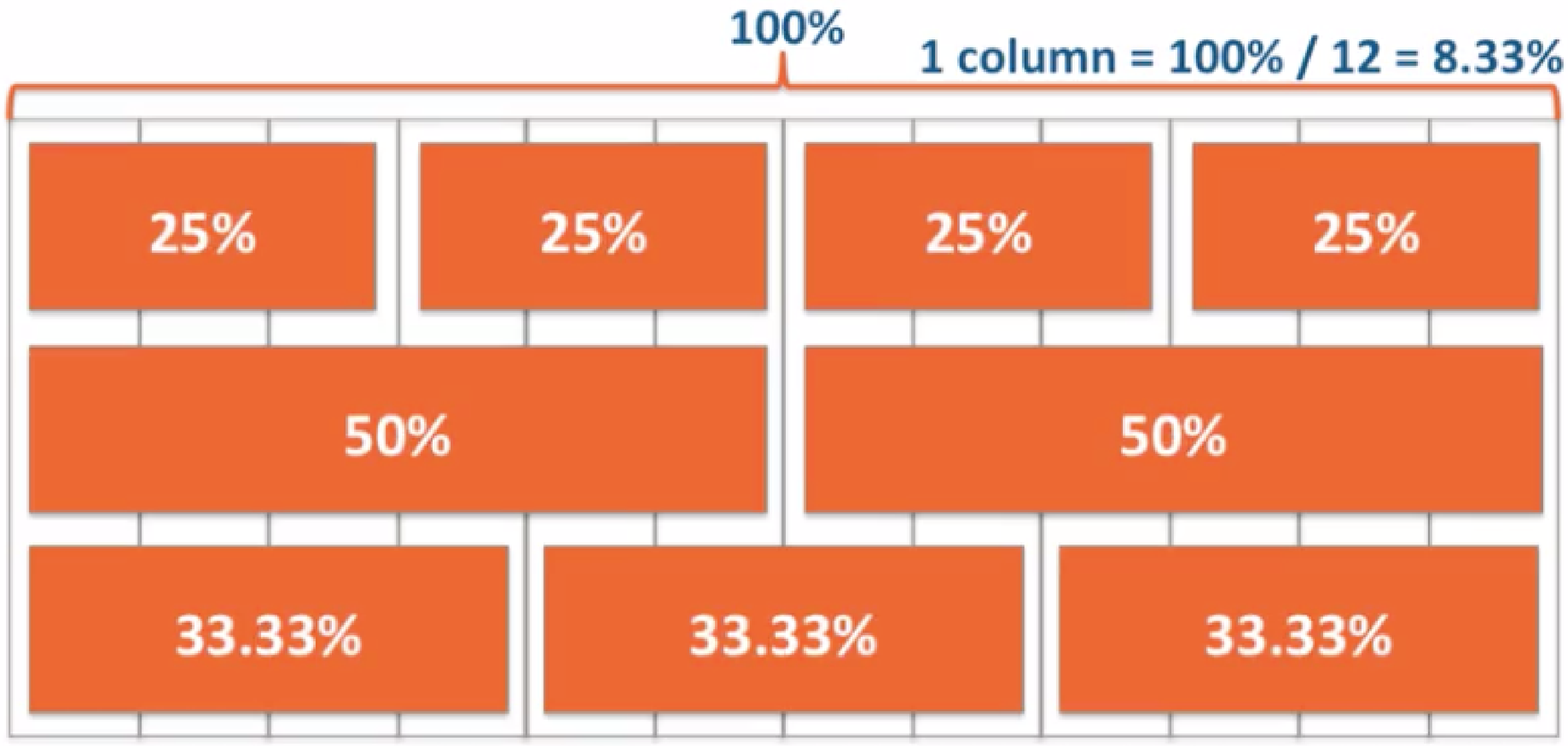


CSS BOOTCAMP

goHARNESS
Live Your Potential

Responsive Layout

12 Column Grid



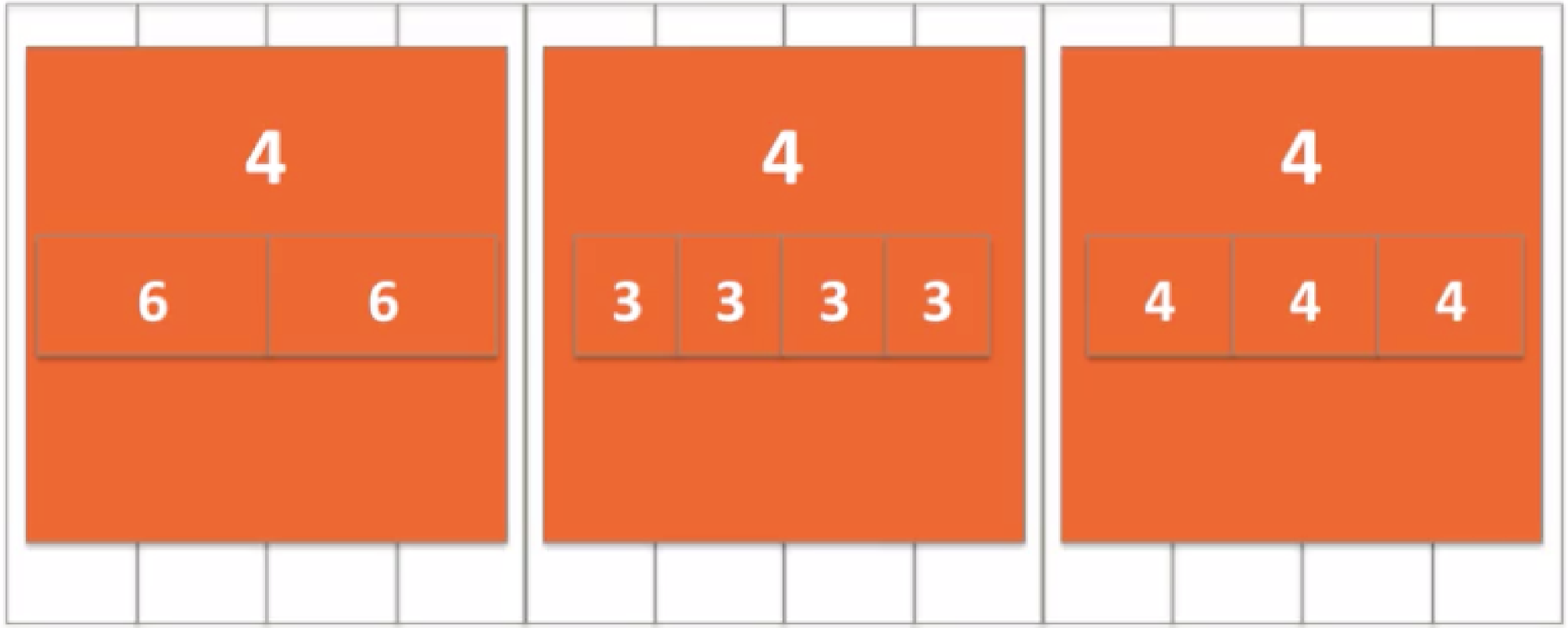
CSS BOOTCAMP

goHARNESS
Live Your Potential

Responsive Layout

12 Column Grid

Nested Grid



CSS BOOTCAMP

go HARNESS
Live Your Potential

CODE ALONG EXERCISE

CSS BOOTCAMP

goHARNESS
Live Your Potential

CSS PREPROCESSORS

CSS BOOTCAMP

goHARNESS
Live Your Potential

Preprocessors

Sass vs LESS

Writing CSS can become quite repetitive and little tasks such as having to look up hex colour values, closing your tags, etc. can become time-consuming. And so that is where a preprocessor comes into play.

A CSS preprocessor is basically a **scripting language** that extends CSS and then compiles it into regular CSS.

Preprocessors

Sass vs LESS

Advantages to using a Preprocessor

1. Cleaner code with reusable pieces and variables
2. Saves you time
3. Easier to maintain code with snippets and libraries
4. Calculations and logic
5. More organised and easy to setup

Preprocessors

Sass vs LESS

Sass and LESS are backward compatible so we can easily convert our existing CSS files just by renaming the **.css** file extension to **.less** or **.scss**, respectively.

LESS is **JavaScript based** and **Sass** is **Ruby based**.

EXERCISE & PROJECT

CSS BOOTCAMP

goHARNESS
Live Your Potential