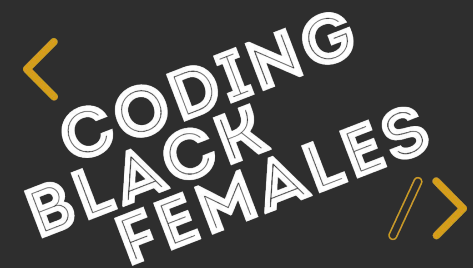


BLACK CODHER

CODING PROGRAMME

Black Codher Bootcamp

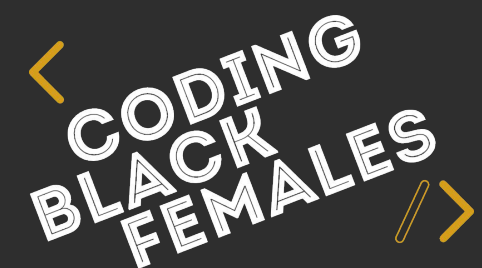


BLACK CODHER

CODING PROGRAMME

UNIT 2

HTML & CSS



BLACK CODHER

CODING PROGRAMME

SESSION 3
Preview Time

< CODING
BLACK
FEMALES >



Recap: What we learnt last week

- Tags, elements, declarations, properties, property values, content, comments, browser compatibility
- Layout: **<!DOCTYPE html>**, **<html lang>**, **<head>** for non visual e.g. metadata, **<meta charset>**, **<title>**
- External CSS **<link rel>**, internal/embedded, inline style +3 simple *selectors*: **element/name**; **ID & class**;
- Responsive web design with **<viewport>**, universal selector (and many more), {indentation}, nesting;
- Semantic web, **<body>** for visuals: **<header>**, **<footer>**, **<nav>**, **<main>**, **<section>** (and many more)
- Hierarchy **heading** up and down **<h1>** - **<h6>**, **<p>**aragraph, **
, **<a href>nchor link, states e.g. pseudo
- Non semantic block level containers e.g. **<div>**, and inline ones like **** (and many more)shorthand
- Formatting **** and for ****phasis vs **** and **<i>** and inserting ****, **<audio>**, **<video>**s
- Security, accessibility, testing, standards, documentation, style guides, SEO, c.theory, design thinking
- GUI + CLI, Git commands e.g. git init, git status, git add, git commit, git remote, push to GitHub

P.S. Don't forget your cheat sheets and our best friend Google!

Learning objectives

1. Learn advanced CSS concepts
2. Learn how CSS Flexbox works
3. Learn how CSS Grids work

BLACK CODHER

CODING PROGRAMME



At peace with ancestral inheritance or fearing family feud?

Combinators

Combinators are used to select the CSS we want to change: they work by combining selectors.

The **descendant combinator ()** selector chooses all specified descendants of an element: grandkids too.

The **child combinator >** selector chooses only the direct children.

There are also **sibling combinator** selectors: general and adjacent ones.

The **general sibling combinator ~** selects all elements that are siblings of the first one specified.

The **adjacent sibling combinator +** selects the ones which immediately follow the element specified.

```
section p {  
    text-decoration: underline overline;  
}  
  
section > p {  
    letter-spacing: 10px;  
}  
  
section ~ p {  
    word-spacing: 30px;  
}  
  
section + p {  
    font-variant: small-caps;  
}
```

BONUS Universal Selector

The **universal selector** selects all the elements on a page. They have the lowest ranking score for specificity.

This means that they will always be superceded by any other CSS rules you set. Still, they are a good fallback when the browser default isn't doing what we want for instance, like how we used it.

```
* {  
  colour: orange;  
}
```


Specificity

At times we might have 2 or more CSS rules which come into conflict. Similar to cascading, browsers follow rules to know determine which one is most specific - a *ranking score* which decides what to apply.

There are 4 categories which define this:






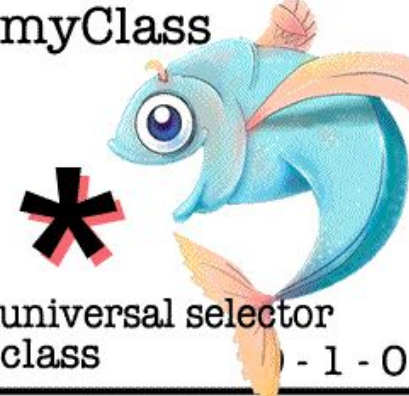



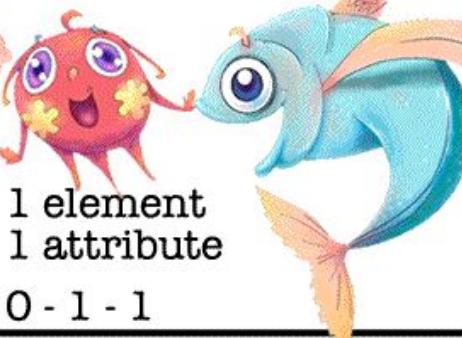

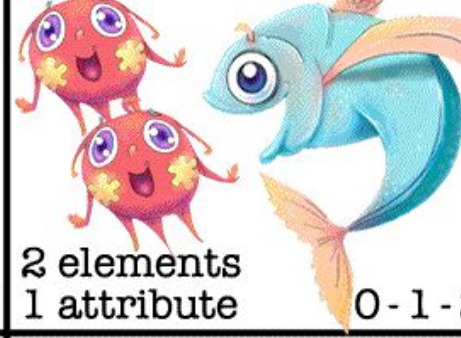
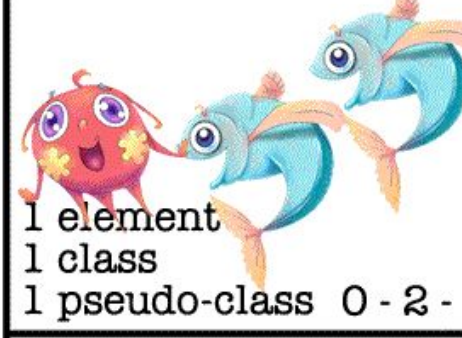
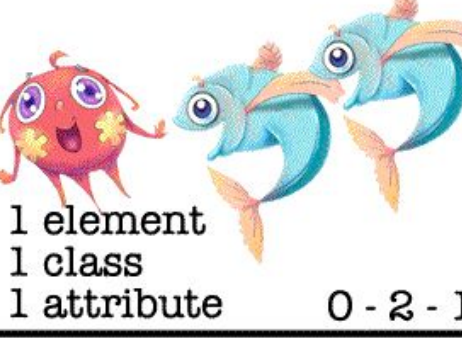

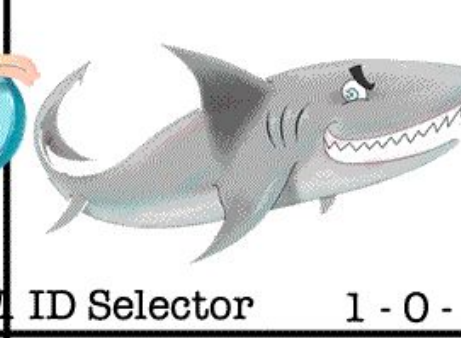
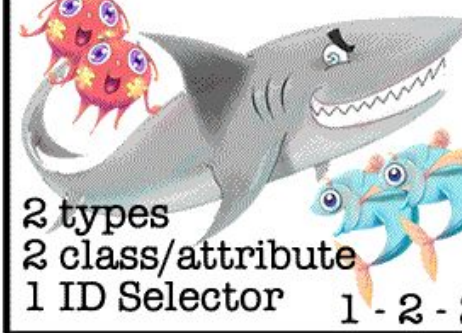
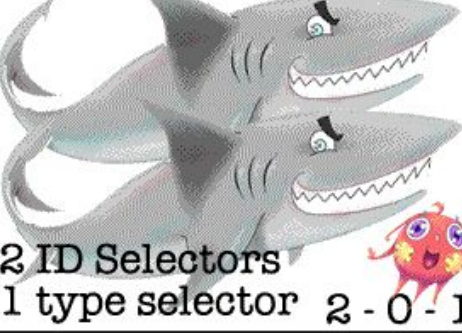


1. Inline style attributes - *1000 points*
2. IDs (unique element identifiers) - *100 points*
3. Classes, attributes, pseudo-classes - *10 points*
4. Elements and pseudo-elements - *1 point*

```
#home {  
    color: red;  
}  
  
.about {  
    color: red;  
}  
  
main {  
    color: red;  
}
```


BLACK CODHER

CSS SPECIFISHITY

WITH PLANKTON, FISH AND SHARKS

*  universal selector 0 - 0 - 0	div  1 element 0 - 0 - 1	li > ul  2 elements 0 - 0 - 2	body div ... ul li p a  12 elements 0 - 0 - 12
.myClass  1 class 0 - 1 - 0	*.myClass  1 universal selector 1 class 0 - 1 - 0	[type=checkbox]  1 attribute selector 0 - 1 - 0	:only-of-type  1 pseudo-class 0 - 1 - 0
li.myClass  1 element 1 class 0 - 1 - 1	li[attr]  1 element 1 attribute 0 - 1 - 1	li:nth-of-type(3n)~li  2 elements 1 pseudo-class 0 - 1 - 2	form input[type=email]  2 elements 1 attribute 0 - 1 - 2
li.class:nth-of-type(3n)  1 element 1 class 1 pseudo-class 0 - 2 - 1	input[type]:not(.class)  1 element 1 class 1 attribute 0 - 2 - 1	cl:nth-child(odd)chk[type]...  10 class/attribute/pseudo-classes 0 - 10 - 0	#myDiv  ID Selector 1 - 0 - 0
#myDiv li.class a[href]  2 types 2 class/attribute 1 ID Selector 1 - 2 - 2	#divitis #myDiv a  2 ID Selectors 1 type selector 2 - 0 - 1	style=""  inline style 1 - 0 - 0 - 0	!important  !important 1 - 0 - 0 - 0 - 0

ESTELLE WEYL * @ESTELLEW * WWW.STANDARDISTA.COM * 2104

X-0-0: The number of ID selectors

0-Y-0: The number of class selectors, attributes selectors, and pseudo-classes

0-0-Z: The number of type selectors and pseudo-elements

*, +, >, ~ : Universal selector and combinators do not increase specificity

:not(x): Negation selector has no value. Argument increases specificity



Let's have an interim break

@rules (at-rules)

@rules are CSS statements that start with an at sign @ followed by an identifier. They can be used for metadata e.g. **@charset**, descriptive information e.g. **@font-face** and conditional logic e.g. **@media** queries.

@media queries are another way of making our content responsive: they query the media type to see if something is true first and adjust accordingly.

As you see on the left, technically, there is also a 4th way to insert CSS - importing. You can import an entire style sheet, or just a select few rules.

Be careful as it can affect the speed of your page. It must always go at the top.

```
@import url('https://fonts.googleapis.com/css?family=Muli&display=swap');  
  
@import url("glossy.css") print;  
  
@import url("textto.css") speech;  
  
@import url('portrait.css') screen and (orientation:portrait);
```


Task 1: Diving deeper into our CSS

1. Create CSS rules with all 4 **combinator** selectors - who listens to elders and which siblings get along?
2. See if you can find or make an element(s) have at least more than 1 rule. What has more **specificity** ?
3. **@import** a CSS stylesheet at the top - look up some examples online. What does it do and or add?

If you have time, you can also try to use **@charset** and **@font-face** to see how they enhance things.

Intermission

CSS Flexbox

Flexible box - shortened to **flexbox**, are 1D one-dimensional layout models, which means they display content either within a row or a column. Their purpose is to space out and automatically align items more flexibly.

So when we lay things out with flexbox, that element becomes a flex container - the *parent* with its *children* inside. To do that, we put the value of the display property to **flex** or **inline-flex**.

There are two axes to consider: the main axis set by **flex-direction** and the resulting cross axis perpendicular to it. Feel free to play around with this and look up other flex properties. It's a good one to have in your toolbox for easily arranging content.

```
<section class="box">  
  <img src... >  
  <img src... >
```

```
.box {  
  display: flex;  
  
/* OR  
  display: inline-flex;  
AND */  
  flex-direction: row;  
  
/* OR  
  row-reverse OR column OR column-reverse; */  
}
```

Flexitime

Task 2: Check out how CSS flexes

Create or choose 4 elements to turn into a **flex container**. You can use the class attribute to identify an element as a **flexbox** or just use the display *property* on any of them with the *property values* set accordingly. Make 2 of them *inline*.

Ensure that you have at least 3 *children* within each *parent* - you might find that images work best.

Test out assigning each of the different **flex directions**. Does this save you time? See if you can create interesting shapes e.g. a T shape on its side.

CSS Grids

Grids are created in a way similar to flexboxes, but are 2D instead: they handle two dimensions. This makes them the most powerful layout system (for now) and more convenient than others options.

First, we set a container element as a grid using the display property as before. Then we define the measurements for our **rows** and **columns**. If we simply decide values, the automatic lines names are 1, 2, 3, 4, 5 ...

We can choose names for them according to a syntax : [linename] size [linename] size ...

```
<section class="grid-container">
  <div class="grid-item">item1</div>
  <div class="grid-item">item2</div>
```

```
.grid-container {
  display: grid; /* OR inline-grid; */

  grid-template-columns: 25px auto 50%
  grid-template-rows: auto auto auto

  grid-template-columns: [c1] 25px [2nd] auto [3] 50%
  grid-template-rows: [row1-start] 33% [row1-end] 00px
```

Break

Task 3: Upgrading via grids for CSS

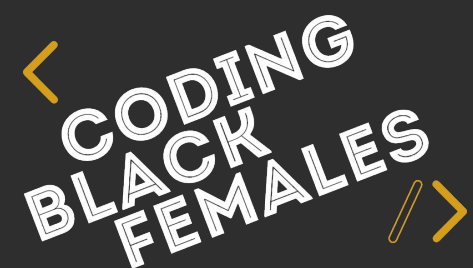
Show what you're working on to the people in your group if you can and are comfortable doing so. Ask for their feedback on where it could be useful for you to put use (a) **grid**(s), in addition to flexbox containers.

Aim to get your portfolio to at least a semi-finished stage, as next time we'll start building a bookshop site. You can take inspiration from the ideas I'll show you or keep researching for your own distinct style ;) !

BLACK CODHER

CODING PROGRAMME

Any questions?



BLACK CODHER

CODING PROGRAMME

Well done! Get some rest and see you soon!

