Notes

Making a website, Unit 3

Intro to CSS

When a website loads it does things in an order. First it parses the HTML (changes the code to computer logic) then creates the DOM and applies the CSS to the finished DOM.

The DOM

The Document Object Model is kind of like a tree with branches that are all spread out.

Applying CSS to DOM

When creating the CSS selector, it will look through the tree. Then find part of the tree that matches the selector, changing the CSS for that part.

Good News

Like HTML, CSS will not break when you mess up in code. Instead it skips the code.

Three ways to do CSS

External. Creating CSS then linking it. Internal. Using style element. Inline. Using style attribute.

Specificity

Specificity refers to how specific the selector is. The more selective the better, less specific will be overwritten. .special is more specific than p so .special would overwrite p. If there is 2 selectors with equal specificity (p and p) then the bottommost one will be the one that overwrites the top one.

CSS ruleset

The things you put inside selectros are called properties and values. Put all three together and you have a ruleset. Ex.

```
Selector{

Property: value units;

}

Footer{

Margin-top: 10px

}
```

Definitions

CSS selector: Chooses a DOM element. CSS property: Changes a part of it. CSS Value: states how properties should change. CSS declarations: When a property is paired with a value. CSS Declaration Block: When several properties are paired with several values.

Functions

CSS can be done with calc. Ex.

```
  <p>
    <div>
      This box is 60% full
    </div>
  </p>
```

(Separation because HTML to CSS)

```
div{
  width: calc(60%);
  background-color: palegreen;
}
```

Rules

There are two important ones. @import loads a different CSS into your CSS. A great way to load a reset style sheet withought having to link two in the HTML. @media shows which styles should be shown when a certain value is met.

Shorthand properties

There are some properties that can cover multiple properties in 1 shorthand property. Ex.

```
text-decoration:dashed underline palevioletred;
```

Selectors

The words at the start are called selectors. They tell the DOM where the different styles will be applied. Ex. P{} h1{} a{} etc.

Classes and ID's

You will want to choose specific elements from time to time. Maybe a p tag holds a formula you don't want to affect every p. For this we use classes and id. You can give any element in HTML a special name you can reference.

Classes

A class is a property in HTML that will mark a consistent theme in your code. A class should be used if you need to change more than one thing. Basically, if you want to make columns in your work then the class="col" should be used to define every column in the html. You call a class in CSS by using a . before the selector. Ex

```
    <p class="col-1-2">This is the first colom</p>
.col-1-2{
    width: calc(50% - 10px);
    float: left;
}
```

ID

Ids are like classes in that they modify an element they make it easier to call out in the CSS. ID should be used in specific situations (unlike Classes). Every ID should be unique, or it should be a class. You can call ID by using #

```
    <header id="grey">
}
#grey{
    color: grey;
}
```

Specificity

Id's, classes, elements, asterisks

Universal selector

If you want to select everything on the page you can use the asterisk *. This has extremely low specificity and will be overwritten by everything. Ex.

```
*{
Margin: 0px;
}
```

Attribute Selectors

You can use selectors that will affect an element based on what attributes are present or if the attribute as a specific value for that attribute. Ex.

```
a{color: blue:}
a[href]{color: purple}
a[href="west.mec.org"]{color: orange}
```

pseudo class and pseudo element

Some elements have properties that we can't see, or they have events that can also affect the style. For instance, if you hover over a link you can change the style or if the link has been visited before. Pseudo elements can interact with the elements in an HTML, for instance if you want to change the first letter of every

paragraph with a tag, you can do that. Ex

P::first-letter{

Font-familly: ariel

}

Combinators

If you want to effect any paragraphs in the article elements you use a space. Ex

Article p{

Color: royalblue

}

If you only want to affect the children (nested) then use a greater than. Ex

Article > p{

Color: royalblue

}

Comments

Just as you would expect, CSS, JS and HTML have different comment styles.

HTML = <!-- -->

CSS = /* */

JS = //

White space

Be careful about whitespace in the properties and ruleset as it can break the code. You can use it all you want around the properties.

Every element is a box

Content with writing, Padding around the content, Border around the padding, Margin around the border. You can see this with inspect.

Block and Inline Boxes

If you define a box as block it will always
 into a new line, it will extend it's width, width and height will be respected, padding, margins, and borders will cause other elements to be pushed away from the box.

Inline boxes will always: never
 into a new line, width and height properties will not apply, vertical padding, margins, and borders will apply but will not cause other inline boxes to move away, horizontal ones will apply and will move other inline boxes away.

Display Block

Default value for most elements. Auto breaks the container elements.

Display inline

Does not break the container into a new line. Horizontal changes work. Vertical changes are ignored. Doesn't break but merges.

Display Grid

Grid works like how it should. It creates a grid pattern on the page, and you can fill in the cells. Block style by default with more format options.

Display flex

Flexbox is one of the newest types of display that you can use. It is quickly becoming standard use for most sites. Works like block but with different formatting options.

Grid properties

Display

Display: grid. Gives you a block level grid

Display: inline grid. Gives you an inline level grid

Typically, you want to stick with block level grid, but you can mess with either if you want to play around.

Grid templates

Grid-template-columns. Defines the columns of the grid with space separated values.

Grid-template-rows. Defines the rows of the grid with space separated values.

The values represent each cells width, and the spaces can be thought of as the dividing grid lines.

You can also name the columns if you like by adding [name] between the sizes but its not required. Ex

Grid-template-rows: [My Name Jeff] 100px;

If you don't name them they will be numbered instead.

There is also a shortcut that we can use if you want to create several columns or rows that are the same. It is repeat(put function here)

More grid templates

Percents – 25% . Will fill the cell to a certain percentage based on the container, based on the screen.

Pixels – 100px . A fixed size

Auto – auto . Fills in the remaining space of the cell.

Fractions – 1fr . Fills the content based on a fraction of the total fr's declared.

Grid fractions

Free space to be divided up by all the fractions, and they will be calculated after everything else. (like auto)

Grid areas step 1

You can create grid areas with names and defined grid cells. First name your areas using classes or elements. Ex

```
    <div class="item-a item">This is the Header</div>
.item-a{
  grid-area: header;
}
```

Grid template areas

You can divide out the cells of the grid to make areas.

"name" represents a grid item,

. represents an empty cell,

None represents no grid area defined.

Grid Area step 2

Once you have a name for your item you can use it to create the grid layout. Every time you say the name a new cell is reserved for that grid area.

Grid gap

This only creates spaces between the areas in the grid, it will not have a gap between the areas and the container. If you only say 1 it will change both. Ex

Gap: 10px

Justify items

Start: Aligns items to be flush with the starting edge of their cell.

End: Aligns items to be flush with the end edge of their cell.

Center: Aligns items in the center of their cell.

Stretch: Fills the whole width of the cell.

Stretch is the default justify.

Align

Justify left right, align up down, same 4 values as justify, just for up and down.

Justify content

Works similarly but affects the container instead of grid cells. (has same four as old and three new.)

Space around: splits the space evenly around the grid items

Space between: splits the space evenly between the grid items

Space evenly: all items will be the same size and spaced evenly

Align content

Sometimes there is extra space in the container and align content fixes that. This will reposition the entire grid based on the grid.

Grid-Auto-Columns/rows

If items are placed outside of outgrid some messy stuff can happen. We can adjust for this by creating auto columns/rows. This will tell your code how to fill in the extra space that you didn't plan for.

Wrapping it all together

You can use "grid" as shorthand. This can be used instead of any other grid, but is really complicated.

Flexbox

Container properties

The following will be properties that you can apply to the container of the flexbox. We call this the flex container.

Display.flex

just like grid, if you want to use the properties that are associated with a certain type of display, the first thing you need to do it is declare the display type. There is one called inline-flex, be you will never use it.

Grid v flex

Grid has its focus on the cells and laying out the grid before hand so it cab be filled in. Flexbox instead focuses on the items in the container, there aren't cells in the container like there are in grid. Items in the container and treated similarly to the cells but they have more flexibility and are not locked into the grid cells.

Flex-direction

The first thing you need to do when using flex is to decide which direction the overflowing will happen. There is row(default), row-reverse, column and column-reverse.

Flex-wrap

By default flex items will try to fit all on one line but you can change that. There is nowrap (default), wrap, wrap-reverse.

Justify-content

It is important to say which way the flex items should be aligned within the container along the main axis(horizontal for flex-direction: row or vertical for column) These work similar to grid. There are flex-start, center, space-around, flex-end, space-between, space-evenly.

Align-items

This works similarly to the box model and positions the items vertically in the container. There is a new value called baseline, this will move every items so that the text in the items is aligned. There are flex-start, center, stretch, flex-end, baseline.

Align-content

This aligns the flex containers lines between the items. This property doesn't do anything if there are not multiple lines in the container.

Item Properties

These are the properties that you want to add to the flex items. These should be placed directly on the items that you want to affect.

Order

The order properties will allow you change the placement of your items inside of your container. This is a powerful tool if you want to order you items when the screen teaches a certain size.

flex-basis

This defines the default size of an element before the remaining space is distributed. It can be any length 920%, 5rem, etc.). The auto keyword means that the item

will fill in the remaining space if possible.

If set to 0 the space around the content is not factored in. If set to auto the extra space is distributed based on its flex-grow value.

Flex-grow

This defines the ability for a flex item to grow if necessary. If all items are set to one then they will try to divide the remaining space equally. If one of them is set to 2 then that item will take up twice as much of the remaining space if possible.

Flex-Shrink

This defines the ability for a flex item to shrink if necessary. Negative values are invalid.

Align-Self

This allows the default alignment (the one specified by align-items) to be overridden for individual flex-items. These have the same 6 keywords as the align-items property.

Floats

The float property was introduced to allow web developers to implement simple layouts involving an image floating inside a column of text. Floats have been used to create entire website layouts featuring multiple columns of information floated so they sit alongside on another.

Float:

The float property is the way that you can move your items around the page and have the rest of the content move around them. There is none, left and right.

Clearing Floats

When you use floats the element will move other elements out of the way and they will wrap around the floated object. If you want to stop the next element from moving up and filling in the space you use the clear property. There is clear left, clear right, and clear both.

Container Collapse

This is a common issue with floats. Since the square has float left, it is taken out of the .wrapper container which causes the blue square to collapse.

The clearfix hack

This is a common hack to fix this. The following code will fix that issue.

.container::after{

Content: “”;

Clear: both;

Display: table;

}

Overflow property

You can also add overflow: auto and it will basically do the same thing.

Positioning

Positioning allows you to place elements and objects where you want and override the setting the default style sheet gives. Just use the position property.

Position: static

This is the default setting and will place the element in order with the rest of the web page. There is no reason to use this since it won’t move anything and you could just leave it blank.

Position: relative

This will allow you to move the element around the page relative to where it should be placed. There is top left bottom and right.

Position: absolute

This removed the element from the rest of the document and the flow of the elements. Instead of being positions within the rest of the document and the flow it will instead be moved to the top left of the page no matter what is in the way.

z-index

layering your content on top f each other is not always easy when you are dealing with position. To make sure the correct element is shown on the top layer we use z-index. The higher the number the higher priority of being on top.

Position: fixed

This works similar to the absolute positioning, it will move the element to the top left of the page and remove it from the list of elements in the document. The difference here is that the fixed will move with the screen when you scroll.

Position: sticky

The value of sticky is a mix of fixed and relative. What it does is it starts at one part of the screen and once you go past it will stick to the top like fixed.

SVG

Scalable Vector Graphics. These are used to create icons, backgrounds, images, and sometimes whole logos. The benefit is that they don't use pixels and they scale up and down. The teacher expects you to use these and credit the author.

Coding Notebook

```
<meta property="og:image" content="URL to an image">
```

git clone "URL" – Clones git repository

```
<meta property="og:description" content="short decription about your page">

<meta property="og:title" content="Title">

<!--Coding Notebook-->
```

<!—Holding alt ctrl and moving up/down gives you multiple cursors-->

Notes for test

Cross Site Scripting

When an attacker finds a website vulnerable to an injection attack and sends an injected website to the user. This copies the data of the user from the injected site.

To prevent this, don't download untrusted programs, encode your html to prevent weak spots in js or CSS.

Broken Access Control

They grant or deny access to information upon user request. This can be as simple as password protecting the data.

Broken access controls don't have those restrictions and is caused by bad authorization controls. This can be exploited by hackers.

(simply) This can be caused by changing the URL and can be prevented with API.