Selectors and Visual Rules

Link Element `<link>`

The `<link>` element is used to link HTML documents to external resources like CSS files. It commonly uses:

- `href` attribute to specify the URL to the external resource
- `rel` attribute to specify the relationship of the linked document to the current document
- `type` attribute to define the type of content being linked

```
<!-- How to link an external stylesheet with href, rel, and
type attributes -->

<link href="./path/to/stylesheet/style.css" rel="stylesheet"
type="text/css">
```

Purpose of CSS

CSS, or Cascading Style Sheets, is a language that is used in combination with HTML that customizes how HTML elements will appear. CSS can define styles and change the layout and design of a sheet.

Class and ID Selectors

CSS classes can be reusable and applied to many elements. Class selectors are denoted with a period `.` followed by the class name. CSS ID selectors should be unique and used to style only a single element. ID selectors are denoted with a hash sign `#` followed by the id name.

```
/* Selects all elements with class="column" */
.column {
}

/* Selects element with id="first-item" */
#first-item {
}
```

Write CSS in Separate Files

CSS code can be written in its own files to keep it separate from the HTML code. The extension for CSS files is **.css**. These can be linked to an HTML file using a `<link>` tag in the `<head>` section.

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

Groups of CSS Selectors

Match multiple selectors to the same CSS rule, using a comma-separated list. In this example, the text for both `h1` and `h2` is set to red.

```css
h1, h2 {
  color: red;
}
```

Write CSS in HTML File

CSS code can be written in an HTML file by enclosing the code in `<style>` tags. Code surrounded by `<style>` tags will be interpreted as CSS syntax.

```html
<head>
  <style>
    h1 {
      color: blue;
    }
  </style>
</head>
```

Selector Chaining

CSS *selectors* define the set of elements to which a CSS rule set applies. For instance, to select all `<p>` elements, the `p` selector can be used to create style rules.

`!important` Rule

The CSS `!important` rule is used on declarations to override any other declarations for a property and ignore selector specificity. `!important` rules will ensure that a specific declaration always applies to the matched elements. However, generally it is good to avoid using `!important` as bad practice.

```css
#column-one {
  width: 200px !important;
}

.post-title {
  color: blue !important;
}
```

Chaining Selectors

CSS selectors can be chained so that rule sets apply only to elements that match all criteria. For instance, to select `<h3>` elements that also have the `section-heading` class, the selector `h3.section-heading` can be used.

```
/* Select h3 elements with the section-heading class */
h3.section-heading {
  color: blue;
}

/* Select elements with the section-heading and button class
*/
.section-heading.button {
  cursor: pointer;
}
```

CSS Type Selectors

CSS *type selectors* are used to match all elements of a given type or tag name. Unlike for HTML syntax, we do not include the angle brackets when using type selectors for tag names. When using type selectors, elements are matched regardless of their nesting level in the HTML.

```
/* Selects all <p> tags */
p {
}
```

CSS class selectors

The CSS class selector matches elements based on the contents of their `class` attribute. For selecting elements having `calendar-cell` as the value of the `class` attribute, a `.` needs to be prepended.

```
.calendar-cell {
  color: #fff;
}
```

HTML attributes with multiple values

Some HTML attributes can have multiple attribute values. Multiple attribute values are separated by a space between each attribute.

```
<div class="value1 value2 value3"></div>
```

Inline Styles

CSS styles can be directly added to HTML elements by using the `style` attribute in the element's opening tag. Each style declaration is ended with a semicolon. Styles added in this manner are known as *inline styles*.

```
<h2 style="text-align: center;">Centered text</h2>

<p style="color: blue; font-size: 18px;">Blue, 18-point
text</p>
```

Selector Specificity

Specificity is a ranking system that is used when there are multiple conflicting property values that point to the same element. When determining which rule to apply, the selector with the highest specificity wins out. The most specific selector type is the ID selector, followed by class selectors, followed by type selectors. In this example, only `color: blue` will be implemented as it has an ID selector whereas `color: red` has a type selector.

```
h1#header {
  color: blue;
} /* implemented */

h1 {
  color: red;
} /* Not implemented */
```

Separating HTML code from CSS code

It is common practice to separate content code in HTML files from styling code in CSS files. This can help make the code easier to maintain, by keeping the syntax for each file separate, and any changes to the content or styling can be made in their respective files.

CSS ID selectors

The CSS ID selector matches elements based on the contents of their `id` attribute. The values of `id` attribute should be unique in the entire DOM. For selecting the element having `job-title` as the value of the `id` attribute, a `#` needs to be prepended.

```
#job-title {
  font-weight: bold;
}
```

CSS descendant selector

The CSS *descendant* selector combinator is used to match elements that are descended from another matched selector. They are denoted by a single space between each selector and the descended selector. All matching elements are selected regardless of the nesting level in the HTML.

```
div p { }

section ol li { }
```

CSS declarations

In CSS, a *declaration* is the key-value pair of a CSS property and its value. CSS declarations are used to set style properties and construct rules to apply to individual or groups of elements. The property name and value are separated by a colon, and the entire declaration must be terminated by a semi-colon.

```css
/*
CSS declaration format:
property-name: value;
*/

/* CSS declarations */
text-align: center;
color: purple;
width: 100px;
```

Font Size

The `font-size` CSS property is used to set text sizes. Font size values can be many different units or types such as pixels.

```css
font-size: 30px;
```

Background Color

The `background-color` CSS property controls the background color of elements.

```css
background-color: blue;
```

Opacity

The `opacity` CSS property can be used to control the transparency of an element. The value of this property ranges from `0` (transparent) to `1` (opaque).

```css
opacity: 0.5;
```

Font Weight

The `font-weight` CSS property can be used to set the weight (boldness) of text. The provided value can be a keyword such as `bold` or `normal` .

```css
font-weight: bold;
```

Text Align

The `text-align` CSS property can be used to set the text alignment of inline contents. This property can be set to these values: `left` , `right` , or `center` .

```css
text-align: right;
```

CSS Rule Sets

A CSS rule set contains one or more selectors and one or more declarations. The selector(s), which in this example is `h1`, points to an HTML element. The declaration(s), which in this example are `color: blue` and `text-align: center` style the element with a property and value. The rule set is the main building block of a CSS sheet.

```css
h1 {
  color: blue;
  text-align: center;
}
```

Setting foreground text color in CSS

Using the `color` property, foreground text color of an element can be set in CSS. The value can be a valid color name supported in CSS like `green` or `blue`. Also, 3 digit or 6 digit color code like `#22f` or `#2a2aff` can be used to set the color.

```css
p {
  color : #2a2aff ;
}

span {
  color : green ;
}
```

Resource URLs

In CSS, the `url()` function is used to wrap resource URLs. These can be applied to several properties such as the `background-image`.

```css
background-image: url("../resources/image.png");
```

Background Image

The `background-image` CSS property sets the background image of an element. An image URL should be provided in the syntax `url("moon.jpg")` as the value of the property.

```css
background-image: url("nyan-cat.gif");
```

Font Family

The `font-family` CSS property is used to specify the typeface in a rule set. Fonts must be available to the browser to display correctly, either on the computer or linked as a web font. If a font value is not available, browsers will display their default font. When using a multi-word font name, it is best practice to wrap them in quotes.

```css
h2 {
  font-family: Verdana;
}

#page-title {
  font-family: "Courier New";
}
```

Color Name Keywords

Color name keywords can be used to set color property values for elements in CSS.

```css
h1 {
  color: aqua;
}

li {
  color: khaki;
}
```

The Box Model

CSS Margin Collapse

CSS *margin collapse* occurs when the top and bottom margins of blocks are combined into a single margin equal to the largest individual block margin.
Margin collapse only occurs with vertical margins, not for horizontal margins.

```
/* The vertical margins will collapse to 30 pixels
instead of adding to 50 pixels. */
.block-one {
  margin: 20px;
}

.block-two {
  margin: 30px;
}
```

CSS `auto` keyword

The value `auto` can be used with the property `margin` to horizontally center an element within its container. The `margin` property will take the width of the element and will split the rest of the space equally between the left and right margins.

```
div {
  margin: auto;
}
```

Dealing with `overflow`

If content is too large for its container, the CSS `overflow` property will determine how the browser handles the problem.
By default, it will be set to `visible` and the content will take up extra space. It can also be set to `hidden`, or to `scroll`, which will make the overflowing content accessible via scroll bars within the original container.

```
small-block {
  overflow: scroll;
}
```

Height and Width Maximums/Minimums

The CSS `min-width` and `min-height` properties can be used to set a minimum width and minimum height of an element's box. CSS `max-width` and `max-height` properties can be used to set maximum widths and heights for element boxes.

```css
/* Any element with class "column" will be at most 200
pixels wide, despite the width property value of 500 pixels.
*/

.column {
  max-width: 200px;
  width: 500px;
}
```

The `visibility` Property

The CSS `visibility` property is used to render `hidden` objects invisible to the user, without removing them from the page. This ensures that the page structure and organization remain unchanged.

```css
.invisible-elements {
  visibility: hidden;
}
```

The property `box-sizing` of CSS *box model*

The CSS *box model* is a box that wraps around an HTML element and controls the design and layout. The property `box-sizing` controls which aspect of the box is determined by the `height` and `width` properties. The default value of this property is `content-box`, which renders the actual size of the element including the content box; but not the paddings and borders. The value `border-box`, on the other hand, renders the actual size of an element including the content box, paddings, and borders.

```css
.container {
  box-sizing: border-box;
}
```

CSS `box-sizing: border-box`

The value `border-box` of the `box-sizing` property for an element corresponds directly to the element's total rendered size, including padding and border with the `height` and `width` properties.

The default value of the `border-box` property is `content-box`. The value `border-box` is recommended when it is necessary to resize the `padding` and `border` but not just the content. For instance, the value `border-box` calculates an element's `height` as follows: `height = content height + padding + border`.

```css
#box-example {
  box-sizing: border-box;
}
```

codecademy

Display and Positioning

CSS `z-index` property

The CSS `z-index` property specifies how far back or how far forward an element will appear on a web page when it overlaps other elements.
The `z-index` property uses integer values, which can be positive or negative values. The element with the highest `z-index` value will be at the foreground, while the element with the lowest `z-index` value will be at the back.

```
//`element1` will overlap `element2`
.element1 {
  position: absolute;
  z-index: 1;
}

.element2 {
  position: absolute;
  z-index: -1;
}
```

Fixed CSS Positioning

Positioning in CSS provides designers and developers options for positioning HTML elements on a web page. The CSS `position` can be set to `static` , `relative` , `absolute` or `fixed` . When the CSS position has a value of `fixed` , it is set/pinned to a specific spot on a page. The fixed element stays the same regardless of scrolling. The navigation bar is a great example of an element that is often set to `position:fixed;` , enabling the user to scroll through the web page and still access the navigation bar.

```
navbar {
postion : fixed;
 }
```

CSS `display` property

The CSS `display` property determines the type of render block for an element. The most common values for this property are `block`, `inline`, and `inline-block`.

Block-level elements take up the full width of their container with line breaks before and after, and can have their height and width manually adjusted.

Inline elements take up as little space as possible, flow horizontally, and cannot have their width or height manually adjusted.

Inline-block elements can appear next to each other, and can have their width and height manually adjusted.

```css
.container1 {
  display: block;
}

.container2 {
  display: inline;
}

.container3 {
  display: inline-block;
}
```

CSS `position: absolute`

The value `absolute` for the CSS property `position` enables an element to ignore sibling elements and instead be positioned relative to its closest parent element that is positioned with `relative` or `absolute`. The `absolute` value removes an element entirely from the document flow. By using the positioning attributes `top`, `left`, `bottom` and `right`, an element can be positioned anywhere as expected.

```css
.element {
  position: absolute;
}
```

CSS `position: relative`

The value `relative` of the CSS `position` property enables an element to be positioned relative to where it would have originally been on a web page. The offset properties can be used to determine the actual position of the element relative to its original position. Without the offset properties, this declaration will have no effect on its positioning, it will act as the default value `static` of the `position` property.

```css
.element {
  position: relative;
}
```

CSS `float` property

The CSS `float` property determines how far left or how far right an element should float within its parent element. The value `left` floats an element to the left side of its container and the value `right` floats an element to the right side of its container. For the property `float`, the `width` of the container must be specified or the element will assume the full width of its containing element.

```css
/* The content will float to the left side of the container.
*/
.left {
  float: left;
}

/* The content will float to the right side of the
container. */
.right {
  float: right;
}
```

The CSS `clear` property

The CSS `clear` property specifies how an element should behave when it bumps into another element within the same containing element.The `clear` is usually used in combination with elements having the CSS `float` property. This determines on which sides floating elements are allowed to float.

```css
/*This determines that no other elements within the same
containing element are allowed to float on the left side of
this element.*/
.element {
  clear: left;
}

/*This determines that no other elements within the same
containing element are allowed to float on the right side of
this element.*/
.element {
  clear: right;
}

/*This determines that no elements within the same
containing element are allowed to float on either side of
this element.*/
.element {
  clear: both;
}

/*This determines that other elements within the same
containing element are allowed to float on both side of this
element.*/
.element {
  clear: none;
}
```

Colors

Color Name Keywords

Color name keywords can be used to set color property values for elements in CSS.

```css
h1 {
  color: aqua;
}

li {
  color: khaki;
}
```

CSS Color Alpha Values

Alpha values determine the transparency of colors in CSS. Alpha values can be set for both RGB and HSL colors by using `rgba()` and `hsla()` and providing a fourth value representing alpha. Alpha values can range between `0.0` (totally transparent) and `1.0` (totally opaque).

The CSS `transparent` value can also be used to create a fully transparent element.

```css
.midground {
  background-color: rgba(0, 255, 0, 0.5);
}

.foreground {
  background-color: hsla(34, 100%, 50%, 0.1);
}

.transparent {
  color: transparent;
}
```

CSS Hexadecimal Colors

CSS colors can be represented in *hexadecimal* (or *hex*) notation. Hexadecimal digits can represent sixteen different values using `0` - `9` and `a` - `f`.

Hexadecimal colors are composed of 6 characters–each group of two represents a value between 0 and 255 for red, green, or blue. For example `#ff0000` is all red, no green, and no blue.

When both characters of all three colors are repeated, hex colors can be abbreviated to only three values, so `#0000ff` could also be represented as `#00f`.

```css
.red {
  color: #ff0000;
}

.short-blue {
  color: #00f;
}
```

CSS HSL Colors

CSS colors can be declared with the *HSL* color system using `hsl()` syntax. This syntax contains three values: *hue* (the color value itself), *saturation* (intensity), and *lightness*.

Hue values range from 0 to 360 while saturation and lightness values are represented as percentages.

```css
.light-blue {
  background-color: hsl(200, 70%, 50%);
}
```

CSS rgb() Colors

CSS colors can be declared with *RGB colors* using `rgb()` syntax.

`rgb()` should be supplied with three values representing red, green, and blue. These values range can from 0 to 255.

```css
.hot-pink {
  color: rgb(249, 2, 171);
}

.green {
  color: rgb(0, 255, 0);
}
```

Typography

CSS `font-weight` Property

The CSS `font-weight` property declares how thick or thin should be the characters of a text. Numerical values can be used with this property to set the thickness of the text. The numeric scale range of this property is from 100 to 900 and accepts only multiples of 100.

The default value is `normal` while the default numerical value is `400` . Any value less than `400` will have text appear lighter than the default while any numerical value greater than the `400` will appear bolder.

In the given example, all the `<p>` elements will appear in a bolder font.

```css
/* Sets the text as bolder. */
p {
  font-weight: 700;
}
```

CSS `font-style` property

The CSS `font-style` property determines the font style in which text will appear.

It accepts `italic` as a value to set the font style to italic.

```css
.text {
  font-style: italic;
}
```

CSS *@font-face* rule

The CSS *@font-face* rule allows external fonts or font files to be imported directly into stylesheets.The location of the font file must be specified in the CSS rule so that the files can be loaded from that location. This rule also allows locally hosted fonts to be added using a relative file path instead of a web URL.

```css
@font-face {
  font-family: 'Glegoo';
  src: url('../fonts/Glegoo-Regular.ttf')
format('truetype');
}
```

CSS Fallback Fonts

The CSS `font-family` property can have multiple fonts declared in order of preference. In this case the fonts following the initial font are known as the *fallback fonts*.

If the initial value of the property `font-family` fails to load to the webpage, the fallback fonts will be used.

```css
/* Here `Arial` is the fallback font for <p> tags */
p {
  font-family: "Helvetica", "Arial";
}
```

The CSS `line-height` property

The CSS `line-height` property declares the vertical spacing between lines of text. It accepts both unitless numbers as a ratio (eg. `2`) and numbers specified by unit as values (eg. `12px`) but it does not accept negative numbers. A unitless number is an absolute value that will compute the line height as a ratio to the font size and a unit number can be any valid CSS unit (eg. pixels, percents, ems, rems, etc.). To set the `line-height` of the `<p>` elements to `10px` , the given CSS declaration can be used.

```css
p {
line-height: 10px;
}
```

CSS *Linking fonts*

Linking fonts allow user to use web fonts in the document. They can be imported in an HTML document by using the `<link>` tag. Once the web font URL is placed within the `href` attribute, the imported font can then be used in CSS declaration.

```html
<head>
  <link href="https://fonts.googleapis.com/css?
family=Droid%20Serif" rel="stylesheet">
</head>
```

Learn Responsive Design

CSS unit `em`

`em` is a CSS unit used to create relatively-sized content. 1 `em` unit represents the base font size of the current element.

In the example code block, `<span>` elements nested inside of elements with class `nav-container` will have a font size of `15px`.

```css
.nav-container {
  font-size: 10px;
}

.nav-container span {
  font-size: 1.5em; /* 10 x 1.5 = 15px */
}
```

Usage of Percentages in CSS

Instead of defining a fixed width using units like `px`, or `cm`, percentages can be used in CSS to set the height and width of child elements relative to the dimensions of their parent. The child elements will grow or shrink according to the set percentage values if the parent element changes in size.

```css
.news-row {
  height: 300px;
  width: 500px;
}

.news-row .news-column {
  height: 80%; /* 240px */
  width: 50%; /* 250px */
}
```

background-size: cover;

The CSS `background-size` property is used to specify the size of the background image. When given the value `cover`, like `background-size:cover`, the image covers the complete background of the container in which it is being displayed.
The proportions of the image are maintained, so if the dimensions exceed the container's, then some parts of the image will be left out.

CSS unit `rem`

The CSS unit `rem` can be used to set the font size of HTML elements relative to the font size of the root element. 1 `rem` represents the size of the base font within the root element - the `<html>` tag.

In the example code block, the font size for all the `<span>` elements will be twice the size of the font size declared for the root element.

```css
html {
  font-size: 18px;
}

span {
  font-size: 2rem;
}
```

Media Type in Media Queries

When writing media queries in CSS we use the `only` keyword to specify the type of device. Initially, CSS incorporated a variety of media types such as `screen`, `print` and `handheld`. In order to ensure responsive design and to accommodate a variety of screen sizes the keyword `screen` is now always used for displaying content.

```css
@media only screen and (min-width: 600px) {
    /* ruleset for >= 600px */
}
```

And Operator Media Queries

Through using the `and` operator when writing media queries in CSS, we can specify that multiple media features are required. For example we could require a certain width as well as a specific resolution for a CSS ruleset to apply. The `and` operator when chaining together multiple media features allows us to be more specific with the screens that we are targeting as we build responsive designs.

```css
@media only screen and (max-width: 480px) and (min-
resolution: 300dpi) {
    /* CSS ruleset */
}
```

css media query

A CSS *media query* can be used to adapt a website's display and layout to different screen sizes. A media query begins with the `@media` keyword and is followed by one or more conditions that check screen size, orientation, resolution, and/or other properties. If these conditions are met, all CSS rules within the media query will be applied to the page. Media queries are used for responsive web design by tailoring specific stylesheets to different types of devices such as laptops, tablets, mobiles, and more.

```css
/* For screen sizes less than or equal to 480px (most likely
a mobile device), the body element's font size will be set
to 12px and the #photo element's width will be set to 100%
*/
@media only screen and (max-width: 480px) {
  body {
    font-size: 12px;
  }

  #photo {
    width: 100%;
  }
}
```

CSS Media Features in Media Queries

Media queries in web development ensure that a website design is responsive. It does so through creating tailored style sheets based on the resolution of the device that it is displayed upon. The browser will detect the screen size and apply the CSS styles for that screen size based on specified media features. Media queries can set rules based on features like the screen width, the screen resolution and device orientation. These media rules can be separated by commas. When one of the media rules is true then the accompanying CSS will apply.

Ranges in Media Queries

Media queries can use CSS to target screen sizes within a certain range through using multiple widths and/or heights. This is an effective tool for responsive design that will address a variety of screen sizes in one CSS media query. In order to set a range for width or the height, set the minimum screen size through using `min-width` and/or `min-height` and then set the maximum through using `max-width` or `max-height` . These properties are used in combination with the `and` operator.

```css
@media only screen and (min-width: 480px) and (max-width:
600px) {
    /* ruleset for 480px - 600px */
}
```

Layout with Flexbox

CSS Flexbox

The CSS `display: flex` property sets an HTML element as a block level flex container which takes the full width of its parent container. Any child elements that reside within the flex container are called flex items.
Flex items change their size and location in response to the size and position of their parent container.

```
div {
  display: flex;
}
```

`justify-content` Property

The CSS `justify-content` flexbox property defines how the browser distributes space between and around content items along the main-axis of their container. This is when the content items do not use all available space on the major-axis (horizontally).
`justify-content` can have the values of:

- `flex-start`
- `flex-end`
- `center`
- `space-between`
- `space-around`

```
/* Items based at the center of the parent container: */
div {
  display: flex;
  justify-content: center;
}

/* Items based at the upper-left side of the parent
container: */
div {
  display: flex;
  justify-content: flex-start;
}
```

`flex` Property

The `flex` CSS property specifies how a flex item will grow or shrink so as to fit within the space available in its `flex` container. This is a shorthand property that declares the following properties in order on a single line:

- `flex-grow`
- `flex-shrink`
- `flex-basis`

```
/* Three properties declared on three lines: */
.first-flex-item {
  flex-grow: 2;
  flex-shrink: 1;
  flex-basis: 150px;
}

/* Same three properties declared on one line: */
.first-flex-item {
  flex: 2 1 150px;
}
```

`flex-direction` Property

The `flex-direction` CSS property specifies how flex items are placed in the flex container - either vertically or horizontally. This property also determines whether those flex items appear in order or in reverse order.

```
div {
  display: flex;
  flex-direction: row-reverse;
}
```

`align-content` Property

The `align-content` property modifies the behavior of the flex-wrap property. It determines how to space rows from top to bottom (ie. along the cross axis). Multiple rows of items are needed for this property to take effect.

`flex-grow` Property

The CSS `flex-grow` property allows flex items to grow as the parent container increases in size horizontally. This property accepts numerical values and specifies how an element should grow relative to its sibling elements based on this value.

The default value for this property is `0`.

```css
.panelA {
  width: 100px;
  flex-grow: 1;
}

/* This panelB element will stretch twice wider than the panelA element */
.panelB {
  width: 100px;
  flex-grow: 2;
}
```

`flex-shrink` Property

The CSS `flex-shrink` property determines how an element should shrink as the parent container decreases in size horizontally. This property accepts a numerical value which specifies the ratios for the shrinkage of a flex item compared to its other sibling elements within its parent container.

The default value for this property is `1`.

```css
.container {
  display: flex;
}

.item-a {
  flex-shrink: 1;
  /* The value 1 indicates that the item should shrink. */
}

.item-b {
  flex-shrink: 2;
  /* The value 2 indicates that the item should shrink twice than the element item-a. */
}
```

Css flex-basis property

The `flex-basis` CSS property sets the initial base size for a flex item before any other space is distributed according to other flex properties.

```css
// Default Syntax
flex-basis: auto;
```

The CSS `flex-flow` property

The CSS property `flex-flow` provides a shorthand for the properties `flex-direction` and `flex-wrap`. The value of the `flex-direction` property specifies the direction of the flex items and the value of the `flex-wrap` property allows flex items to move to the next line instead of shrinking to fit inside the flex container. The `flex-flow` property should be declared on the flex container.

```
// In this example code block, "column" is the value of the
property "flex-direction" and "wrap" is the value of the
property "flex-wrap".

.container {
  display: flex;
  flex-flow: column wrap;
}
```

CSS `display: inline-flex` property

The CSS `display: inline-flex` property sets an HTML element as an inline flex container which takes only the required space for the content. Any child elements that reside within the flex container are caleld flex items. Flex items change their size and location in response to the size and position of their parent container.

```
.container{
  display: inline-flex;
}
```

Flexbox Properties align-items

When working with CSS flexbox `align-items` is used to align flex items vertically within a parent container.

Css flex-wrap property

The `flex-wrap` property specifies whether flex items should wrap or not. This applies to flex items only. Once you tell your container to `flex-wrap`, wrapping become a priority over shrinking. Flex items will only begin to wrap if their combined `flex-basis` value is `greater` than the current size of their flex container.

```
.container {
  display: flex;
  flex-wrap: wrap;
  width: 200px;
}
```

codecademy

Grid

Grid Template Columns

To specify the number of columns of the grid and the widths of each column, the CSS property `grid-template-columns` is used on the grid container. The number of width values determines the number of columns and each width value can be either in pixels(`px`) or percentages(%).

```css
#grid-container {
  display: grid;
  width: 100px;
  grid-template-columns: 20px 20% 60%;
}
```

fr Relative Unit

The CSS grid relative sizing unit `fr` is used to split rows and/or columns into proportional distances. Each `fr` unit is a fraction of the grid's overall length and width. If a fixed unit is used along with `fr` (like pixels for example), then the `fr` units will only be proportional to the distance left over.

```css
/*
In this example, the second column take 60px of the avaiable 
100px so the first and third columns split the remaining 
available 40px into two parts (`1fr` = 50% or 20px)
*/

.grid {
  display: grid;
  width: 100px;
  grid-template-columns: 1fr 60px 1fr;
}
```

Grid Gap

The CSS `grid-gap` property is a shorthand way of setting the two properties `grid-row-gap` and `grid-column-gap` . It is used to determine the size of the gap between each row and each column. The first value sets the size of the gap between rows and while the second value sets the size of the gap between columns.

```css
// The distance between rows is 20px
// The distance between columns is 10px

#grid-container {
  display: grid;
  grid-gap: 20px 10px;
}
```

CSS Block Level Grid

CSS Grid is a two-dimensional CSS layout system. To set an HTML element into a block-level *grid container* use `display: grid` property/value. The nested elements inside this element are called *grid items*.

```
#grid-container {
  display: block;
}
```

CSS grid-row

The CSS `grid-row` property is shorthand for the `grid-row-start` and `grid-row-end` properties specifying a grid item's size and location within the grid row. The starting and ending row values are separated by a `/`. There is a corresponding `grid-column` property shorthand that implements the same behavior for columns.

```
/*CSS Syntax */
grid-row: grid-row-start / grid-row-end;

/*Example*/
.item {
  grid-row: 1 / span 2;
}
```

CSS Inline Level Grid

CSS Grid is a two-dimensional CSS layout system. To set an HTML element into a inline-level *grid container* use `display: inline-grid` property/value. The nested elements inside this element are called *grid items*.

The difference between the values `inline-grid` and `grid` is that the `inline-grid` will make the element inline while `grid` will make it a block-level element.

```
#grid-container {
  display: inline-grid;
}
```

`minmax()` Function

The CSS Grid `minmax()` function accepts two parameters:

- The first parameter is the minimum size of a row or column.
- The second parameter is the maximum size.

The grid must have a variable width for the `minmax()` function.
If the maximum value is less than the minimum, then the maximum value is ignored and only the minimum value is used.
The function can be used in the values of the `grid-template-rows`, `grid-template-columns` and `grid-template` properties.

```
/* In this example, the second column will vary in size
between 100px and 500px depending on the size of the web
browser" */

.grid {
  display: grid;
  grid-template-columns: 100px minmax(100px, 500px) 100px;
}
```

grid-row-start & grid-row-end

The CSS `grid-row-start` and `grid-row-end` properties allow single grid items to take up multiple rows. The `grid-row-start` property defines on which row-line the item will start. The `grid-row-end` property defines how many rows an item will span, or on which row-line the item will end. The keyword `span` can be used with either property to automatically calculate the ending value from the starting value or vice versa. There are complementary `grid-column-start` and `grid-column-end` properties that apply the same behavior to columns.

```
/* CSS syntax:
grid-row-start: auto|row-line;
grid-row-end: auto|row-line|span n;
*/
grid-row-start: 2;
grid-row-end: span 2;
```

CSS grid-row-gap

The CSS `grid-row-gap` property determines the amount of blank space between each row in a CSS grid layout or in other words, sets the size of the gap (gutter) between an element's grid rows. The `grid-column-gap` provides the same functionality for space between grid columns.

```
/*CSS Syntax */
grid-row-gap: length; /*Any legal length value, like px or
%. 0 is the default value*/
```

CSS grid-area

The CSS `grid-area` property specifies a grid item's size and location in a grid layout and is a shorthand property for the `grid-row-start`, `grid-column-start`, `grid-row-end`, and `grid-column-end` in that order. Each value is separated by a `/`.

In the included example, `Item1` will start on row 2 and column 1, and span 2 rows and 3 columns

```
.item1 {
  grid-area: 2 / 1 / span 2 / span 3;
}
```

Justify Items

The `justify-items` property is used on a grid container. It's used to determine how the grid items are spread out along a row by setting the default `justify-self` property for all child boxes.

The value `start` aligns grid items to the left side of the grid area.

The value `end` aligns grid items to the right side of the grid area.

The value `center` aligns grid items to the center of the grid area.

The value `stretch` stretches all items to fill the grid area.

```
#container {
  display: grid;
  justify-items: center;
  grid-template-columns: 1fr;
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;
}
```

Align Self

The CSS `align-self` property is used to set how an individual grid item positions itself along the column or block axis. By default grid items inherit the value of the `align-items` property on the container. So if the `align-self` value is set, it would over-ride the inherited `align-items` value.

The value `start` positions grid items on the top of the grid area.

The value `end` aligns the grid on the bottom of the grid area.

The value `center` positions grid items on the center of the grid area.

The value `stretch` positions grid items to fill the grid area (default).

CSS grid-template-areas

The CSS `grid-template-areas` property allows the naming of sections of a webpage to use as values in the `grid-row-start`, `grid-row-end`, `grid-column-start`, `grid-column-end`, and `grid-area` properties. They specify named grid areas within a CSS grid.

```
/* Specify two rows, where "item" spans the first two
columns in the first two rows (in a four column grid
layout)*/
.item {
  grid-area: nav;
}
.grid-container {
  display: grid;
  grid-template-areas:
    'nav nav . .'
    'nav nav . .';
}
```

CSS grid-auto-flow

The CSS `grid-auto-flow` property specifies whether implicity-added elements should be added as rows or columns within a grid or, in other words, it controls how auto-placed items get inserted in the grid and this property is declared on the grid container.

The value `row` specifies the new elements should fill rows from left to right and create new rows when there are too many elements (default).

The value `column` specifies the new elements should fill columns from top to bottom and create new columns when there are too many elements.

The value `dense` invokes an algorithm that attempts to fill holes earlier in the grid layout if smaller elements are added.

```
/*CSS Syntax */
grid-auto-flow: row|column|dense|row dense|column dense;
```

Justify Content

Sometimes the total size of the grid items can be smaller than the grid container. If this is the case, the CSS property `justify-content` can be used to position the entire grid along the row or inline axis of the grid container.

The value `start` aligns the grid to the left side of the grid container.

The value `end` aligns the grid to the right side of the grid container.

The value `center` centers the grid horizontally in the grid container.

The value `stretch` stretches the grid items to increase the size of the grid to expand horizontally across the container.

The value `space-around` includes an equal amount of space on each side of a grid element, resulting in double the amount of space between elements as there is before the first and after the last element.

The value `space-between` includes an equal amount of space between grid items and no space at either end.

The value `space-evenly` places an even amount of space between grid items and at either end.

Align Content

Some times the total size of the grid items can be smaller than the grid container. If this is the case, the CSS property `align-content` can be used to position the entire grid along the column axis of the grid container.
The property is declared on the grid container.

The value `start` aligns the grid to the top of the grid container.

The value `end` aligns the grid to the bottom of the grid container.

The value `center` centers the grid vertically in the grid container.

The value `stretch` stretches the grid items to increase the size of the grid to expand vertically across the container.

The value `space-around` includes an equal amount of space on each side of a grid element, resulting in double the amount of space between elements as there is before the first and after the last element.

The value `space-between` includes an equal amount of space between grid items and no space at either end.

The value `space-evenly` places an even amount of space between grid items and at either end.

CSS grid-auto-rows

The CSS `grid-auto-rows` property specifies the height of implicitly added grid rows or it sets a size for the rows in a grid container. This property is declared on the grid container. `grid-auto-columns` provides the same functionality for columns. Implicitly-added rows or columns occur when there are more grid items than cells available.

Justify Self

The CSS `justify-self` property is used to set how an individual grid item positions itself along the row or inline axis. By default grid items inherit the value of the `justify-items` property on the container. So if the `justify-self` value is set, it would over-ride the inherited `justify-items` value.

The value `start` positions grid items on the left side of the grid area.

The value `end` positions the grid items on the right side of the grid area.

The value `center` positions grid items on the center of the grid area.

The value `stretch` positions grid items to fill the grid area (default).

```
// The grid items are positioned to the right (end) of the
row.

#grid-container {
  display: grid;
  justify-items: start;
}

.grid-items {
  justify-self: end;
}
```

CSS grid-area

The CSS `grid-area` property allows for elements to overlap each other by using the `z-index` property on a particular element which tells the browser to render that element on top of the other elements.

Align Items

The `align-items` property is used on a grid container. It's used to determine how the grid items are spread out along the column by setting the default `align-self` property for all child grid items.

The value `start` aligns grid items to the top side of the grid area.

The value `end` aligns grid items to the bottom side of the grid area.

The value `center` aligns grid items to the center of the grid area.

The value `stretch` stretches all items to fill the grid area.

```
#container {
  display: grid;
  align-items: start;
  grid-template-columns: 1fr;
  grid-template-rows: 1fr 1fr 1fr;
  grid-gap: 10px;
}
```