**A Brief History of (Web) Style**

* CSS was first proposed in 1994
* Work quickly advanced, and by late 1996, CSS1 was finished
* While the newly established CSS Working Group moved forward with CSS2, browsers struggled to implement CSS1 in an interoperable way.
* Before all that happened, though, the CSS Working Group had finalized the CSS2 specification in early 1998.
* Once CSS2 was finished, work immediately began on CSS3, as well as a clarified version of CSS2 called CSS2.1. In keeping with the spirit of the times, CSS3 was constructed as a series of (theoretically) standalone modules instead of a single monolithic specification. This approach reflected the then-active XHTML specification, which was split into modules for similar reasons. The rationale for modularizing CSS3 was that each module could be worked on at its own pace, and particularly critical (or popular) modules could be advanced along the W3C’s progress track without being held up by others.

**Replaced and Nonreplaced Elements**

* Elements are the basis of document structure. In HTML, the most common elements are easily recognizable, such as p, table, span, a, and div. In CSS, elements generally take two forms: replaced and nonreplaced.
* Replaced elements are those where the element’s content is replaced by something that is not directly represented by document content. Probably the most familiar HTML example is the img element, which is replaced by an image file external to the document itself.
* Most of HTML elements are nonreplaced elements. This means that their content is presented by the user agent (generally a browser) inside a box generated by the element itself. For example, <span>hi there</span> is a nonreplaced element, and the text “hi there” will be displayed by the user agent.

**Element Display Roles**

* In addition to replaced and nonreplaced elements, CSS uses two other basic types of elements: block-level and inline-level.
* Block-level elements generate an element box that (by default) fills its parent element’s content area and cannot have other elements at its sides.
* Inline-level elements generate an element box within a line of text and do not break up the flow of that line.
* Note that while the names “block” and “inline” share a great deal in common with block- and inline-level elements in HTML, there is an important difference. In HTML, block-level elements cannot descend from inline-level elements. In CSS, there is no restriction on how display roles can be nested within each other.

**How to Integrate CSS with HTML?**

* Purpose of link tag is to allow HTML authors to associate other documents with the document containing the tag. Stylesheets which are not part of the HTML document but are still used by it, are referred to as external stylesheets. One can use *@import* directive (can be placed style tag inside the document- must be before all style rules. Can also be placed inside other external stylesheets) to import external stylesheet. The tag consists of following attributes:
  + - “Rel” stands for relationship with value “stylesheet”
    - “type” attribute describes the type of data loaded in the document via the link tag.
    - Href
    - Media
    - Title

**Alternate Stylesheets**

Stylesheets which are defined with “rel” attribute “alternate stylesheet”. This kind of stylesheet is only used for presentation if selected by the user.  
  
**Difference between Alternate and Preferred Stylesheet?**

Stylesheets with “rel” attribute (with value “stylesheet”) can also have another attribute named “title”. With this attribute, the stylesheet becomes preferred stylesheet (meaning that is applied by default to the document). If a conforming browser (which supports alternate-stylesheets), have alternate-stylesheets (through “rel” attribute with value “alternate stylesheet”) and if one of them is selected, that selection supersedes with preferred stylesheet. However, if a stylesheet with no “title” attribute is mentioned, that is used always.

**Style Element**

Referred to as document or embedded stylesheet. This tag is present in the document. It needs to have type attribute with value ‘text/css’. It can also contain links to external stylesheets (via @import directive).

**@import directive**

Can be part of style element. It looks something like this:

@import url(style1.css);

Like “link”, import is used to load external stylesheets. The major difference is where it can be placed. While link element can be placed anywhere in body or head element, the directive can only be placed in style element (at the top – above all style rules). Another distinction between the two is that unlike stylesheets loaded via link tag, they are loaded and used (no concept of alternate stylesheets). You can provide media descriptors after stylesheet url.

**HTTP Linking (Obscure way)**

**Inline Styles**

Can be applied to any elements found in “body” tag. You cannot put import directive in it.

**Stylesheet Contents**

* No markup
* Made up of rules. Each rule comprises of two parts: selector and declaration block. Declaration block may contain one or more declaration. Each declaration is made up of two parts property and its value.

**Vendor Prefixes:**

Used by browsers vendors to mark properties as experimental or proprietary or both.

**Whitespace Handling**

Css treats whitespace just like html (combining multiple whitespaces into one).

**Comments**

/\* \*/ (cannot be nested)

**Media Queries**

With media queries, an author can define the media environment in which a given stylesheet is used by the browser. In the past, this was handled by setting media types via the media attribute on the link element, on a style element, or in the media descriptor of an @import or @media declaration. Media queries take this concept several steps further by allowing authors to choose stylesheets based on the features of a given media type, using what are called media descriptors.

h1 {color: maroon;}

@media projection {

body {background: yellow;}

}

You can have as many @media blocks as you like in a given stylesheet, each with its own set of media descriptors.

## **Media Types**

The most basic form of media queries are media types, which first appeared in CSS2. These are simple labels for different kinds of media:

all

Use in all presentational media.

print

Use when printing the document for sighted users and also when displaying a print preview of the document.

screen

Use when presenting the document in a screen medium like a desktop computer monitor. All web browsers running on such systems are screen-medium user agents.

Multiple media types can be specified using a comma-separated list.  
  
<link type="text/css" href="frobozz.css" media="screen, print">

<style type="text/css" media="screen, print">...</style>

@import url(frobozz.css) screen, print;

@media screen, print {...}

Each media descriptor is composed of a media type and one or more listed media features, with each media feature descriptor enclosed in parentheses.

If no media type is provided, then it is assumed to be all.

@import url(print-color.css) print and (color), screen and (min-color: 8);

In any situation where even one of the media queries evaluates to “true,” the associated stylesheet is applied. Thus, given the previous @import, print-color.css will be applied if rendering to a color printer or to a sufficiently colorful screen environment.

Multiple feature descriptors can be used to link two keywords “and” and “not”. ‘,’ is used to serve the purpose (used as an OR condition since “or” does not exists) and is preferred.

There is one more keyword, only, which is designed to create deliberate backward incompatibility (yes, really):

**Featured Queries**

Between 2015 and 2016, CSS gained the ability to apply blocks of CSS when certain CSS property-value combinations were supported by the user agent. These are known as feature queries.

@supports (color: black) {

body {color: black;}

h1 {color: purple;}

h2 {color: navy;}

}

That says, in effect, “If you recognize and can do something with the property-value combination color: black, apply these styles. Otherwise, skip these styles.” In user agents that don’t understand @supports, the entire block is skipped over.

As with media queries, feature queries also permit logical operators. Suppose we want to apply styles only if a user agent supports both grid layout *and* CSS shapes. Here’s how that might go:

@supports (display: grid) and (shape-outside: circle()) {

/\* grid-and-shape styles go here \*/

}

This is essentially equivalent to writing the following:

@supports (display: grid) {

@supports (shape-outside: circle()) {

/\* grid-and-shape styles go here \*/

}

}

Remember that these are feature queries, not correctness queries. A browser can understand the feature you’re testing for, but implement it with bugs. So you’re not getting an assurance from the browser that it supports something correctly.