**CSS**

**What is CSS?**

* **CSS stands for Cascading Style Sheets.**
* **CSS helps elements how to be displayed on the web.**
* **CSS files the external style sheets which consist of the styles for HTML file.**

**History of CSS**

**Hakon Wium Lie** released the first draft of “Cascading HTML Style Sheets” in October 1994. Although it took 3 years for any browser to come close to fully implementing CSS, August **1996** **Microsoft** **Internet** **Explorer** became the **first browser** to support CSS.

* The CSS 1 W3C Recommendation was made in December 1996.
* In May of 1998, the W3C released CSS 2 which added new capabilities including z-index, media types, bidirectional text, absolute, relative and fixed positioning, and support for aural style sheets.
* With the release of CSS 3, capabilities were broken into modules. Between June 2011 and June 2012, the following four modules were released as formal recommendations: color, selectors level 3, namespaces and media queries.

**Why Use CSS?**

CSS is used to **style** your web pages. Using CSS you will be able to put out pages with much less work, that load much faster, and will be easy to update and print!

**CSS Syntax**

Basic CSS rule consists of a **selector** and a **declaration** :

* CSS selector

All texts in <p> elements will blue:

p { color: blue; }

The **selector** points to the HTML element you want to style. In this example p is the selector for <p> elements in HTML file.

The declaration block contains one or more declarations separated by semicolons. In this example { color: blue; } is the declaration block.

Declarations must be surrounded by **curly braces** and end with a **semicolon**.

**Use of CSS**

### External CSS file

The styles can be located in a different file, so with an external style sheet, the design of the website can be changed from this file.

To include an external stylesheet, <link> element must be used as a reference to the external style sheet in the <head> section of the HTML page.

### Example

<!DOCTYPE html>

<html>

<head>

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

</head>

<body>

<p>Hello World.</p>

</body>

</html>

You can write an external style sheet in any text editor and must save it with .css extension.

Here is an example of a .css file :

body {

background-color: yellow;

}

p {

color: red;

}

### Internal CSS

For any single HTML file, internal styles can be used. The internal styles are written inside the <style> section of the HTML page.

### Example

<!DOCTYPE html>

<html>

<head>

<style>

body {

background-color: yellow;

}

p {

color: red;

}

</style>

</head>

<body>

<p>Hello World.</p>

</body>

</html>

### Inline CSS

For any single element in the HTML file can have inline styles. For inline styling, the style attribute is used.

### Example

<!DOCTYPE html>

<html>

<body>

<p style="color: blue;">Hello World</p>

</body>

</html>

### Multiple Styles and Cascade Rule.

All styles which are applied to the same element will apply in the cascade order of rules.

Inline styles have the top priority, then external and internal style sheets and finally browser default styles. So, an inline style will override external and internal styles and browser defaults.

### Example

In the external style sheet, color of <p> is defined as red.  
  
file.css

p { color: red; }

In the internal style, color of <p> is defined as blue, and in the inline style yellow where the browser default is black. In this example, the color of <p> element will be yellow.

html

<!DOCTYPE html>

<html>

<head>

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

<style>

p {

color: blue;

}

</style>

</head>

<body>

<p style="color: yellow;">Hello World</p>

</body>

</html>

**The Box Model**

All elements in the HTML file are considered as boxes. The "Box Model" term is used to describe a box that surrounds an element. It consists of: margins, borders, padding, and content.

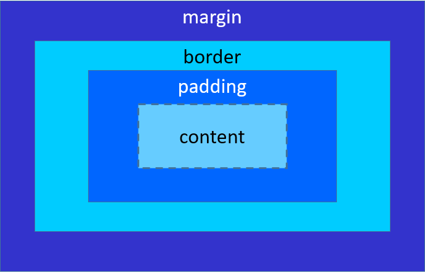
**Explanation of the different parts:**

**Content**: The content of the box.

**Padding**: A transparent area around the content.

**Border**: A border around the padding and content.

**Margin**: A transparent area around the border.



**Example**

div {

border: 1px solid black;

padding: 5px;

margin: 10px;

}

### CSS Selectors

CSS selectors are used to select HTML elements.

## CSS element Selector:

An element selector uses element name to select the HTML element.

### Example

**p selector is used to select all <p> elements in the HTML file.**

**p {**

**text-align: center;**

**color: red;**

**}**

## The CSS id Selector

The id selector uses **id**attribute to select the element in HTML file.

The id of an element must be unique in a page. A hash **(#)** character is used to select an element with a specific id.

### Example

The HTML element with the id="text1" is selected:

#text1 {

color: red;

}

## The CSS class Selector

The class selector uses class attribute to select all elements of that class in HTML file.

A period (.) character is used to select elements with a specific class.

### Example

All HTML elements with the class="error" is selected:

.error {

color: red;

}

p.error {

color: red;

}

### The Universal Selector

The universal selector \* selects all HTML elements on the page.

### Example

\* {

margin: 0;

padding: 0;

color: black;

}

### CSS Comments

Comments are used to add an explanation of the code. Comments are ignored by browsers.

### Example

p {

color: red; /\*Single-line comment\*/

}

/\* Multi-line

comment \*/

**What is a Media Query?**

**Media Query**

A Media query is a CSS3 feature that makes a webpage adapt its layout to different screen sizes and media types.

Syntax

@media media type and (condition: breakpoint) {

// CSS rules

}

We can target different media types under a variety of conditions. If the condition and/or media types meet, then the rules inside the media query will be applied, otherwise, they won’t.

@ Media Rule

We start defining media queries with @media rule and later include CSS rules inside the curly braces. The @ media rule is also used to specify target media types.

@media () {

// CSS rules

}

**Parenthesis**

Inside the parenthesis, we set a condition. For example, I want to apply a larger font size for mobile devices. To do that, we need to set a maximum width which checks the width of a device:

.text {

font-size: 14px;

}

@media (max-width: 480px) {

.text {

font-size: 16px;

}

}

Normally, the text size will be 14px. However since we applied a media query, it will change to 16px when a device has a maximum width of 480px or less.

**Important:** Always put your media queries at the end of your CSS file.

**Media Types**

If we don’t apply a media type, the @ media rule selects all types of devices by default. Otherwise, Media types come right after the @ media rule. There are many kinds of devices but we can group them into 4 categories:

* all — for all media types
* print — for printers
* screen — for computer screens, tablets and, smart-phones
* speech — for screen readers that “read” the page out loud

For example, when You want to select only screens, You will set the screen keyword right after the @ media rule. You also must concatenate the rules with the “and” keyword:

@media screen and (max-width: 480px) {

.text {

font-size: 16px;

}

}

**Breakpoints**

A breakpoint is a key to determine when to change the layout and adapt the new rules inside the media queries. Let’s go back to our example at the beginning:

@media (max-width: 480px) {

.text {

font-size: 16px;

}

}

Here, the breakpoint is 480px. Now the media query knows when to set or overwrite the new class. If the width of a device is smaller than 480px, the text class will be applied, otherwise, it won’t.

**Common Breakpoints**

One of the most commonly asked questions is “Which breakpoint should we use?”. There are a ton of devices on the market so we can’t and we shouldn’t define fixed breakpoints for each of them.

Common breakpoints for widths of devices:

* 320px — 480px: Mobile devices
* 481px — 768px: iPads, Tablets
* 769px — 1024px: Small screens, laptops
* 1025px — 1200px: Desktops, large screens
* 1201px and more —  Extra large screens, TV

As is said above, these breakpoints can differ and there is no standard exactly defined, but these are some commonly used ones.

**Logical Keyword "not"-"only"**

The logical keywords not or `only`` can be used **optionally** to include or exclude specific media types or screen sizes. For instance, we may want to specify that a navigation bar should extend across the width of our page, but only on screens larger than 700px in height. It might also be the case that you want a CSS rule to apply to all devices, but not screens smaller than 400px wide.

@media only screen and (max-width: 992px) {

body {

background-color: blue;

}

}

@media not screen and (max-width: 992px) {

body {

background-color: blue;

}

}

**Logical Keyword "and"-","**

We can use and and , to separate or combine conditions.

* Using and requires that both conditions on each side of the and are true in order for the query to apply.
* Using , stands for or, meaning that only one of the conditions on either side of the comma has to be true for the query to trigger.

@media screen and (min-width: 992px) and (max-width: 1136px) {

body {

background-color: blue;

}

}

@media screen and (min-width: 992px), (max-width: 1136px) {

body {

background-color: blue;

}

}

**How to Use Media Query-Examples**

To give better context, let's create a media query that will change a paragraph's text from red to green when the screen size falls below 800px:

/\* initial style \*/

p {

color: red;

}

/\* media query \*/

@media only screen and (max-width: 800px) {

p {

color: green;

}

}

On the second line, we provide some initial styles that will be applied unless our media query triggers. We have set our paragraph to have red text. In line 5, we create a media query using **@media** and set it to **only** trigger for **screen** devices that have a **max-width** of **800px**.

In other words, the style will only be applied to screens that are less than **801px** wide. Following the condition are a pair of curly braces (**{}**) that enclose the style(s) to be applied when the condition is met. In our case, on screens that are **800px** wide or less, we change the text color of paragraphs to green instead of red.  
Please [click here](https://codepen.io/clarusway/pen/xxVZgEP) to check.

For our size conditions, we can use the properties min-width, max-width, min-height, and max-height, all referring to the size of the device's viewport (i.e., the screen size). Above, we looked at **max-width**; now let's explore **min-width**:

/\* initial style \*/

p {

color: red;

}

/\* media query \*/

@media only screen and (min-width: 400px) {

p {

color: green;

}

}

In line 2, we set the default color of paragraphs to red again. In line 5, we set the terms of our media query — namely, that the width of the **screen** device must be **400px** or more.

So, **max-width** triggers when a screen is smaller than the specified size, and **min-width** triggers when a screen is larger than a certain size.  
Please [click here](https://codepen.io/clarusway/pen/YzqwNNJ) to check.

As mentioned before, we can use the **and** keyword to chain multiple conditions together. Let's look at an example of that:

/\* initial style \*/

p {

color: red;

}

/\* media query \*/

@media only screen and (min-width: 400px) and (max-width: 800px) {

p {

color: green;

}

}

Here we are changing the text color to green only on screens that are between 400px and 800px.  
Please [click here](https://codepen.io/clarusway/pen/QWNydpj) to check.

**Conclusion**

CSS Media Queries provide us a way to alter our CSS at specific screen sizes by setting breakpoints at different screen widths. Using the **max-width** condition expressions to trigger styles below a certain size and the **min-width** condition expressions to trigger styles above a certain size can help you build powerful, flexible interfaces for various screen sizes and devices. Instead of trying to target specific device sizes, use the in-browser dev tools for experimentation, and allow your unique content to determine at which sizes to write media query breakpoints. Write media queries as you need them at whatever size your content starts to become unsightly.