CSS

專業人士須知

免費程式設計書籍

免責聲明

這是一本出於教育目的而創建的非官方免費書籍,

不隸屬於官方 css 團體或公司.

所有商標和註冊商標均為

其各自所有者的財產

目錄

第0章 - 介紹

請隨時免費與任何人分享此 PDF,

本書的最新版本可以從以下網址下載:

https://goalkicker.com/cssBook

這本 css Notes for Professionals 書是從 Stack Overflow 編譯的

文檔,內容是由 Stack Overflow 的優秀人員編寫的.

請將回饋和更正發送至 web@petercv.com

GoalKicker.com

TW翻譯: 賀皓群 (discord: chris0527,email: chris960527ho@gmail.com)

第1章 - 開始使用css

版本釋出日:

css1版1996/12/17	css2版1998/05/12	css3版2015/10/13 // 目前版本

1-1節: 外部樣式表

1-1節: 外部樣式表	1-2節: 內部樣式	1-3節: css @import 規則(css at 規則之一)	1-4節: 內嵌樣式
1-5節: 用js更改css	1-6節: 使用css設定清單樣式		

透過在每個html文件中放置元素,可以將外部css樣式表套用到任意數量的html文件.

標籤的rel屬性必須設定為“stylesheet”,href屬性必須設定為相對(/絕對)樣式表的路徑.

雖然使用相對url路徑通常被認為是好的做法,但絕對路徑也可以使用.

在html5中,可以省略type屬性.

建議將 標記放置在 html 檔案的 標籤中,以便在先前載入樣式

否則,用戶將看到一閃而過的無樣式內容.

範例:

```
<!-- hello-world.html -->
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <link rel="stylesheet" href="index.css"> <!-- 等效於<link rel="stylesheet" type="text-css" href="index.css"> -->
    </head>
    <body>
        <h1>Hello world!</h1>
        <p>I ♥ css</p>
    </body>
</html>
```

```
/* index.css */
h1{
    color: green;
    text-decoration: underline;
}

p{
    font-size: 25px;
    font-family: 'Trebuchet MS', sans-serif;
}
```

請記得確保在href中包含css檔案的正確路徑.

如果css檔案與html位於相同資料夾中

則不需要資料夾路徑(如上面的範例).

但如果它保存在資料夾中,則需要像這樣指定它

href="資料夾名稱/style.css".

```
<link rel="stylesheet" href="foldername/style.css">
```

外部樣式表被認為是處理css的最佳方式

原因很簡單:如果當您正在管理一個超過100個頁面的網站,所有頁面均由單一樣式表控制,並且您想要更改連結顏色從藍色到綠色.

那麼呢比起在每個文件更改,在一個css文件中進行更改容易非常多,而且程式也會比較乾淨

您可以根據需要在html頁面中載入任意數量的css檔案. 例如:

```
<link rel="stylesheet" href="main.css">
<link rel="stylesheet" href="override.css">
```

css 規則與一些基本規則一起應用,並且順序很重要.

例如,如果您有一個 index.css 文件

例如:

```
p.green{
    color: #00FF00;
}
```

所有帶有.green類別的段落都將以淺綠色編寫,但您可以使用另一個.css覆蓋它

文件只需將其包含在index.css之後即可.您可以在index.css之後使用override.css並使用以下程式碼.

例如:

```
p.green{
    color: #006600;
}
```

現在,所有具有.green類別的段落都將以深綠色而不是淺綠色書寫.

其他原則也適用,例如!important規則,特異性和繼承.

當某人第一次造訪您的網站時,他們的瀏覽器會下載目前頁面的html以及連結的css文件.

然後,當他們導航到另一個頁面時,他們的瀏覽器只需要下載該頁面的html,這css檔案被(緩存)[],因此不需要再次下載.

由於瀏覽器(快取)[]外部樣式表,您的頁面載入速度更快.

1-2節:內部樣式

html文件中<style></style>標籤中包含的css功能類似於外部樣式表,

不同之處在於**它存在於其樣式的html文件中而不是單獨的文件中,因此只能應用於它所在的文檔**.

請注意,**此元素必須位於元素內以進行html驗證**(儘管它如果放在body中,將在所有目前瀏覽器中工作).

例如:

```
<head>
    <style>
        h1{
            color: green;
            text-decoration: underline;
        }
        p{
            font-family: 'Trebuchet MS', sans-serif;
            font-size: 25px;
        }
    </style>
</head>
<body>
    <h1>Hello world!</h1>
    <p>I ♥ css</p>
</body>
```

1-3節: css @import規則(css at-rule 之一)

@import是一個可以用來連結其他樣式表的規則

您可以透過以下方式使用@import規則: A. 帶有內部<style>標籤

```
<style>
    @import url("/css/index.css");
</style>
```

B. 使用外部樣式表 以下行將根目錄中名為index.css的css檔案匯入到它所在的css檔案中:

```
@import "/index.css";
```

導入外部css也是可能的.

一個常見的用例是字體檔案.

```
@import "https://fonts.googleapis.com/css?family=Lato";
```

@import規則的可選第二個參數是媒體查詢清單:

```
@import "/print-styles.css" print;
@import url("landscape.css") screen and (orientation:landscape);
```

1-4節: 內嵌樣式

使用內嵌樣式(inline style)將樣式套用至特定元素.

請注意,這不是最佳的.

仍然鼓勵將樣式規則放置在<style>標籤中或外部css文件,以保持內容和表示之間的差異.

內嵌樣式會覆蓋<style>標籤或外部樣式表中的任何css.(優先及最高)

雖然這在某些情況下可能有用,但在這種情況下,這一事實往往會降低專案的可維護性.

以下範例中的樣式直接套用於它們所附加的元素.

```
<h1 style="color: green; text-decoration: underline;">hello world!</h1>
<p style="font-size: 25px; font-family: 'Trebuchet MS';">I ♥ css</p>
```

內嵌樣式通常是確保各種電子郵件用戶端和程式和設備之間呈現相容性的最安全方法

但編寫起來可能很耗時,管理起來也有點困難.

1-5節: 用js更改css

js可以透過元素的style屬性使用js新增或刪除或修改css屬性值.

例如:

```
document.getElementById("element").style.opacity=0.5
document.getElementById("element").style.fontFamily="sans-serif"
```

請注意**樣式屬性以小駝峰式命名**

在範例中,您會看到css屬性font-family在js中變成了fontFamily.

作為直接處理元素的替代方法,您可以在js中建立<style>或<link>元素然後將其附加到html文件的<body>或<head>中.

1-6節: 使用css設定清單樣式

有三個不同屬性可以設定清單項目樣式: list-style-type list-style-image list-styleposition

應依該順序宣告.

預設值分別為disc outside none.

每個屬性可以單獨聲明,也可以使用列表樣式的簡寫屬性.

list-style-type 定義用於每個清單項目的項目符號點的形狀或類型.

list-style-type 的一些可接受的值:

- disc
- circle
- square
- decimal
- lower-roman
- upper-roman
- none (有關詳盡列表,請參閱 W3C 規範 wiki)

例如,若要為每個清單項目使用方形項目符號點,您可以使用下列屬性-值對:

```
li{
    list-style-type: square;
}
```

list-style-image 屬性決定清單項目圖示是否設定有影像,並接受下列值

- none
- 圖像url

```
li{
    list-style-image: url(images/bullet.png);
}
```

list-style-position 屬性定義清單項目標記的位置,它接受兩個值之一:

- inside
- outside

```
li{
    list-style-position: inside;
}
```

第2章-css規則的結構與格式

2-1節: 屬性清單

某些屬性(property)可以採用多個值,統稱為屬性清單(property list).

```
/* 該屬性清單中有兩個值 */
span{
    text-shadow: yellow 0 0 3px, green 4px 4px 10px;
}

/* 替代格式 */
span{
    text-shadow:
        yellow 0 0 3px,
        green 4px 4px 10px;
}
```

2-2節: 多重選擇器

當您將 css 選擇器分組時,您可以將相同的樣式套用於多個不同的元素,而無需重複樣式,在你的樣式表中**可以使用逗號分隔多個分組選擇器**.

例如:

```
div,p{
    color: blue
}
```

因此藍色適用於所有 元素和所有<p>元素.如果沒有逗號,則只有<p>元素是 的子元素時是紅色的.

這也適用於所有類型的選擇器.

```
p,.blue,#first,div span{
    color: blue
}
```

此規則適用於:

<p>	藍色類別的元素	元素id為first的元素	內的每個

2-3節: 規則,選擇器,聲明區塊

CSS規則由**選擇器**(例如 h1)和宣告區塊({})組成.

```
h1{
    /* property */
}
```

第3章-註解

單行註解

```
/* 這是註解 */
div{
    color: red; /* 這是註解 */
}
```

多行註解

```
/*
這
是
註
解
*/
div{
    color: red;
}
```

!請注意! css 沒有 // # 等註解方式

第4章-選擇器

css選擇器(selector)將特定的html元素識別為css樣式的目標.

本主題介紹css選擇器如何定位html元素.

選擇器使用css語言提供的50多種選擇方法.

包括元素(element),類別(class),id,偽元素(pseudo-element)和偽類選擇器(pseudo-class)以及模式(pattern).

4-1節: 基本選擇器

選擇器說明:

表示法	使用場景
*	通用選擇器(所有元素)
div	標籤選擇器(所有 元素)
.blue	類別選擇器(所有具有blue類別的元素)
.blue.red	所有具有blue和red類別的元素(一種複合選擇器)
#headline	id選擇器(id屬性設定為headline的元素)
:pseudo-class	所有具有偽類選擇器的元素
::pseudo-element	與偽元素相符的元素
:lang(en)與:lang	宣告相符的元素,例如
div>p	子選擇器

!注意! id的值在網頁中必須是唯一的.(但在不在乎標準的情形下仍可以使用,但極度不建議)

使用以下內容違反了html標準

> 在同一dom樹中多次使用同樣id的值.

完整的選擇器清單可以在css選擇器Lev3規格中找到.

4-2節: 屬性選擇器

概述

屬性選擇器可以與各種類型的運算子一起使用,從而相應地更改選擇標準.

他們使用給定屬性或屬性值的存在來選擇元素.

選擇器[#04021]	匹配元素	選擇元素...	css版本
[attr]	<div attr>	具有屬性attr	2
[attr="val"]		其中屬性attr的值為val	2
[attr~="val"]		其中val出現在以空格分隔的 attr 列表	2
[attr^="val"]		其中attr的值以val開頭	3
[attr$="val"]		其中attr的值以val結尾	3
[attr*="val"]		其中attr在任何地方包含val	3
[attr|="val"]		其中attr的值恰好是val,或以val開頭並立即隨後是"-"	2
[attr="val" i]		其中 attr 的值為val,忽略val的字母大小寫.	4[#04022]

註釋:

> #04021: 屬性值可以用單引號或雙引號括起來. 完全沒有引號也可以可以工作, 但根據css標準它是違規的,因此不鼓勵這樣做.

> #04022: 沒有單一的或整合的css4規範,因為它被分成單獨的模組. 但是有"level 4"模組. 請參閱瀏覽器支援.

細節:

[屬性(attribute)] 選擇**具有給定屬性的元素**.

```
div[data-color]{
    color: red;
}
```

```
<div data-color="red">這會變紅色</div>
<div data-color="green">這會變紅色</div>
<div data-background="red">這不會變紅色</div>
```

JSBin上的線上演示

[屬性="值(value)"(也可以是單引號)] 選擇**具有給定屬性和值的元素**.

```
div[data-color="red"]{
    color: red;
}
```

```
<div data-color="red">這會變紅色的</div>
<div data-color="green">這不會變紅色</div>
<div data-color="blue">這不會變紅色</div>
```

JSBin上的線上演示

ps: 此演示沒演示到 這不會變紅色 應該是錯誤的

[屬性*="值"] 選擇具有給定屬性和值的元素,其中**給定屬性在任何位置包含給定值**(如一個子串).

```
div[class*="foo"]{
    color: red;
}
```

```
<div class="foo-123">這會變紅色</div>
<div class="foo123">這會變紅色</div>
<div class="bar123foo">這會變紅色</div>
<div class="barfooo123">這會變紅色</div>
<div class="barfo0">這不會變紅色</div>
```

JSBin上的線上演示

[屬性~="值"] 選擇具有給定屬性和值的元素,其中**給定值出現在以空格分隔的清單中**.

```
div[class~="color-red"]{
    color: red;
}
```

```
<div class="color-red foo-bar the-div">這會變紅色</div>
<div class="color-blue foo-bar the-div">這不會變紅色</div>
```

JSBin上的線上演示

[屬性^="值"] 選擇具有給定屬性和值的元素,其中**給定屬性以該值開頭**.

```
div[class^="foo-"]{
    color: red;
}
```

```
<div class="foo-123">這會變紅色</div>
<div class="foo-234">這會變紅色</div>
<div class="bar-123">這不會變紅色</div>
```

JSBin上的線上演示

[屬性$="值"] 選擇具有給定屬性和值的元素,其中**給定屬性以給定值結尾**.

```
div[class$="file"]{
    color: red;
}
```

```
<div class="foobar-file">這會變紅色</div>
<div class="foobar-file">這會變紅色</div>
<div class="foobar-input">這不會變紅色</div>
```

JSBin上的線上演示

[屬性|=“值”] 選擇具有給定屬性和值的元素,其中**屬性的值恰好是給定值或恰好是給定值後面接著"-"**

```
div[lang|="EN"]{
    color: red;
}
```

```
<div lang="EN-us">這會變紅色</div>
<div lang="EN-gb">這會變紅色</div>
<div lang="PT-pt">這不會變紅色</div>
```

JSBin上的線上演示

[屬性="值" i] 選擇具有給定屬性和值的元素,其中**屬性的值可以表示為任何不區分大小寫的值**.(例如: [class="value"] 那: Value,VALUE,vAlUe 等皆可被使用)

```
div[lang|="EN" i]{
    color: red;
}
```

```
<div lang="EN">這會變紅色</div>
<div lang="en">這會變紅色</div>
<div lang="TW">這不會變紅色</div>
```

JSBin上的線上演示

屬性選擇器的優先級為: 0-1-0

與偽元素及偽類選擇器相同.

請注意,這表示屬性選擇器可用於按較低優先級(specificity)等級的id選擇元素與使用id選擇器選擇相比: [id="my-id"] 目標與 #my-id 相同的元素,但具有較低的優限性.

有關詳細信息,請參閱語法部分.

4-3節: 關係選擇器

概述:

範例	關係選擇器(combinator)名稱及說明
div span	後代選擇器(所有 都是 的後代)
div>span	子選擇器(所有 的直接子級)
a~span	通用同級選擇器(之後的所有同級)
a+span	相鄰同級選擇器(緊接在 之後的所有)

注意: 同級選擇器的目標元素是來源文件中緊接著的元素.

css本質上不能定位前一個元素或父元素.

但是使用flex order屬性,可以在視覺媒體上模擬先前的同級選擇器參見.

後代組合器: 選擇器 選擇器

後代組合器,由至少一個空格字元" "表示,選擇作為已定義元素的後代的元素. **此組合器選擇該元素的所有後代(從子元素向下)**.

```
div p{
    color: red;
}
```

```
<div>
    <p>我的文字是紅色</p>
    <span>
        <p>我的文字是紅色</p>
    </span>
</div>
<p>我的文字不是紅色</p>
```

JSBin上的線上演示

在上面的範例中,選擇了前兩個<p>元素,因為它們都是 的後代.

子組合器: 選擇器>選擇器

子(>)組合符用於選擇作為**指定元素的子元素或直接後代的元素**

```
div>p{
    color: red;
}
```

```
<div>
    <p>我的文字是紅色</p>
    <span>
        <p>我的文字不是紅色</p>
    </span>
</div>
```

JSBin上的現場演示

上面的css**只選擇第一個<p>元素,因為它是唯一直接從 繼承的段落**.

未選擇第二個<p>元素,因為它不是 的直接子元素.

鄰近兄弟組合器: 選擇器+選擇器

相鄰同級(+)組合符選擇緊接在指定元素之後的同級元素.

```
div+p{
    color: red;
}
```

```
<p>我的文字不是紅色</p>
<p>我的文字是紅色</p>
<p>我的文字是紅色</p>
<hr>
<p>我的文字不是紅色</p>
```

JSBin上的現場演示

上面的範例**僅選擇直接位於另一個<p>元素前面的那些<p>元素**.

通用兄弟組合器: 選擇器 ~ 選擇器

通用同級(~)組合器選擇指定元素後面的所有同級.

```
div~p{
    color: red;
}
```

```
<p>我的文字不是紅色</p>
<p>我的文字是紅色</p>
<hr>
<h1>標題</h1>
<p>我的文字是紅色</p>
```

JSBin上的現場演示

上面的範例**選擇前面有另一個<p>元素的所有<p>元素,無論它們是否是緊鄰**.

4-4節: 偽類選擇器

偽類選擇器(同:偽類)(pseudo-classes)是關鍵字,允許根據文檔樹以外的資訊進行選擇或不能由其他選擇器或組合器來表達.

該資訊可以與某個狀態相關聯(狀態和動態偽類),到位置(結構和目標偽類),到前者的否定(否定偽類)或語言(lang偽類).

例如連結是否已被跟隨(:visited),滑鼠懸停在元素上(:hover),選取核取方塊(:checked)等.

更多詳細功能及介紹可至此連結查看.

語法

```
選擇器:偽類{
    property: VALUE;
}
```

偽類列表:

名稱	描述
:active	適用於**使用者啟動(即點擊)的任何元素**.
:any	允許您透過**建立群組來建立相關選擇器集包含的項目將會匹配**.這是重複整個選擇器的替代方法.
:checked	適用於**已選取**的單選,核取方塊或選項元素或切換到"開啟"狀態.
:default	表示一組**預設的任何使用者介面元素**相似的元素.
:disabled	適用於**任何處於停用狀態的UI元素**.
:empty	適用於**任何沒有子元素的元素**.
:enabled	適用於**任何處於啟用狀態的UI元素**.
:first	與@page規則結合使用,**選擇一個頁面中的第一頁列印文件**.
:first-child	表示作為其**父元素的第一個子元素**的任何元素.
:first-of-type	當**元素是所選元素類型的第一個時應用在其父級內部**.這可能是也可能不是第一個子元素.
:focus	適用於**任何具有使用者焦點的元素**.這可以由下式給出: 使用者的鍵盤,滑鼠事件或其他形式的輸入.
:focus-within	當其中的**一個元素獲得焦點時**,可用於突出顯示整個部分.
:focus	偽類匹配的**任何元素或具有後代焦點的元素**.
:full-screen	適用於**以全螢幕模式顯示的任何元素**.它選擇整個堆疊元素而不僅僅是頂級元素.
:hover	適用於使用者**指標裝置懸停的任何元素**,但是未激活.
:indeterminate	套用既**未選取也未選取**的單選或複選框UI元素處於不確定狀態.這可能是由於元素的屬性或DOM操作.
:in-range	**其value屬性在此元素的指定範圍限制內**.它允許頁面給出當前定義的值的回饋使用該元素在範圍限制內.
:invalid	適用於**其值無效的<input>元素**.和type=attribute相同.
:lang	適用於包裝<body>元素的任何元素,**該元素具有正確的指定lang屬性**. 為了使偽類有效,它必須包含有效的兩個或三個字母的語言代碼.
:last-child	表示作為其**父元素的最後一個子元素的任何元素**.
:last-of-type	當元素是**內部所選元素類型的最後一個時適用它的父級**.這可能是也可能不是最後一個子元素.
:left	與@page規則結合使用,**選擇所有左側列印文件中的頁面**.
:link	適用於使用者**尚未造訪過的任何連結**.

名稱	描述
:not()	適用於**與傳遞給的值不符的所有元素**(例如: :not(p)或:not(.class-name)).它必須有一個值有效且只能包含一個選擇器.但是,您可以連結多個:not選擇器一起.
:nth-child()	當**元素是其父元素的第n個元素時適用**,其中n可以是整數,數學表達式(例如 n+3)或關鍵字奇數或偶數.
:nth-of-type	當**一個元素是其父元素的第n個元素時適用相同的元素類型**,其中 n 可以是整數,數學表達式表達式(例如 n+3)或關鍵字 odd 或 Even.
:only-child	代表**任何元素這是其父母的唯一孩子**. 這與:first-child :last-child 或 :nth-child(1):nth-last-child(1),但優先級較低.
:optional	代表任何元素**沒有設定所需的屬性**. 這允許表單可以輕鬆指示可選欄位並相應地設定它們的樣式.
:out-of-range	當一個元素有它的值時**會匹配value屬性超出了該元素的指定範圍限制**.它允許頁面給出當前使用定義的值的回饋元素超出範圍限制. 如果值是,則該值可能超出範圍小於或大於最大和最小設定值.
($) :placeholder-shown	適用於目前**顯示佔位符文字(placeholder)的任何表單元素**.
:read-only	適用於**任何使用者不可編輯的元素**.
:read-write	適用於**使用者可編輯的任何元素**,例如<input>元素.
:right	與@page規則結合使用,**這會選擇a中的所有正確頁面列印文件**.
:root	符合表示**元素樹的根元素**.
:range	符合作為**引用的元素選擇器要匹配的點**.
:target	表示一個唯一的元素(目標元素),其id與目前URL片段相符
:visited[#04041]	適用於使用者**已造訪過的任何連結**.

ps:

> #04041: :visited偽類不能再用於許多現代瀏覽器中的大多數樣式,因為這是一個安全漏洞.請參閱此連結以供參考.

4-5節: nth偽類

> Represents elements whose numeric position in a series of siblings matches the pattern An+B, for every positive integer or zero value of n, where: A is an integer step size, B is an integer offset, n is all nonnegative integers, starting from 0. It can be read as the An+B-th element of a list. The A and B must both have values. - MDN :nth-child

---

也就是說:

表示元素在兄弟元素列表中的位置是An+B模式的元素,其中n為正整數或0,A和B為整數且A不為0.其中:

> A是整數步長 | B是增量偏移量
>
> n是從0開始的所有非負整數 它可以被理解為列表中的第An+B一個元素.A和B必須都是"integer"值.

此表為假設子元素有10個的情形下會被選擇的元素

偽類選擇器	1	2	3	4	5	6	7	8	9	10
:first-child	✔									
:nth-child(3)			✔							
:nth-child(n+3)			✔	✔	✔	✔	✔	✔	✔	✔
:nth-child(3n)			✔			✔			✔	
:nth-child(3n+1)	✔			✔			✔			✔
:nth-child(-n+3)	✔	✔	✔							
:nth-child(odd)	✔		✔		✔		✔		✔	
:nth-child(even)		✔		✔		✔		✔		✔
:last-child										✔
:nth-last-child(3)							✔			

4-6節: 類別選擇器

類別選擇器選擇具有目標類別名稱的所有元素.

例如: .warning將選擇以下 元素:

```
<div class=“警告”>
  <p>這將是一些警告文案.</p>
</div>
```

您也可以將類別名稱組合到更具體的目標元素.

讓我們以上面的例子為基礎展示更複雜的類別選擇.

```
.important{
    color: orange;
}
.warning{
    color: blue;
}
.warning.important{
    color: red;
}
```

```
<div class="warning">
    <p>這將是一些警告文案.</p>
</div>
<div class="important warning">
    <p class="important">這是一些非常重要的警告文案.</p>
</div>
```

在此範例中所有具有.warning類別的元素都將具有藍色文字顏色,並具有.important類別的元素的文字顏色為橘色,同時具有.important和.warning類別名稱的所有元素都將具有紅色文字顏色.

請注意,在css中: .warning.important 聲明的兩個類別名稱之間沒有任何空格.

這意味著它只會尋找在其類別屬性中**同時包含**類別名稱 warning 和 important 的元素. 這些類別名稱在元素上**可以按任意順序排列**.

如果css宣告中的兩個類別之間包含空格,則它只會選擇**具有**.warning**類別名稱的父元素和具有**.important**類別名稱的子元素的元素**.

4-7節: id選擇器

id選擇器選擇具有目標id的dom元素.

要在CSS中透過特定ID選擇元素,前綴是#

例如,以下 html div 元素:

```
<div id="exampleid">
    <p>範例</p>
</div>
```

可以透過css中的 #exampleid 來選擇,如下所示:

```
#exampleid{
    width: 20px;
}
```

id選擇器的優先級為: 1-0-0

!!!請注意 html規範不允許多個元素具有相同的ID!!!

4-8節: id選擇器,不含高位id選擇器的優先級

這個技巧可以幫助您使用id作為屬性選擇器的值來選擇元素,以避免id選擇器的高優先級

```
<div id="element">...</div>
```

```
#element{ ... } /* 高優先級將覆蓋許多選擇器 */

[id="element"]{ ... } /* 低優先級,可以輕鬆覆蓋 */
```

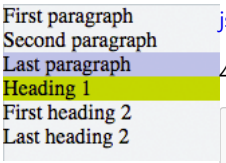
4-9節: :last-of-type選擇器

:last-of-type 選擇作為其父元素的特定類型的最後一個子元素的元素. 在下面的例子中,css選取最後一段和最後一個標題 h1.

```
p:last-of-type{
    background: #C5CAE9；
}

h1:last-of-type{
    background: #CDDC39；
}
```

```
<div class="container">
    <p>第一段</p>
    <p>第二段</p>
    <p>最後一段</p>
    <h1>標題1</h1>
    <h2>第一個標題 2</h2>
    <h2>最後一個標題 2</h2>
</div>
```



jsFiddle

4-10節: css3 :in-range 選擇器範例

```
<style>
input:in-range{
    border: 1px blue solid;
}
</style>

<input type="number" min="10" max="20" value="15">
<p>當在10~20時會變有藍色邊框</p>
```

當元素的value屬性在指定範圍內時(10~20),:in-range偽類選擇器匹配該元素.

它允許頁面給出當前使用元素定義的值的回饋在範圍限制之內.

參見

4-11節: :not 偽類範例

以下選擇器符合html文件中所有未停用且不具有類別.

```
<form>
    Phone: <input type="tel" class="example">
    E-mail: <input type="email" disabled>
    Password: <input type="password">
</form>
```

```
input:not([disabled]):not(.example){
    background-color: #ccc;
}
```

:not() 偽類別也將支援選擇器lev4中的逗號分隔選擇器:

JSBin上的線上演示

請參閱此處的背景語法.

4-12節: :focus-within 偽類範例

html:

```
<h3>如果輸入獲得焦點則背景變為藍色.</h3>
<div>
```

```
    <input type="text">
</div>
```

```
div{
    height: 80px;
}

input{
    margin:30px;
}

div:focus-within{
    background-color: #1565C0;
}
```

4-13節: 帶複選框的全域布林值 checkbox:checked 和 一般兄弟組合器(~)

使用~選擇器,您可以輕鬆實現全域可存取的布林值,而無需使用js.

新增布林值作為複選框

在文件的開頭,添加盡可能多的布林值以及唯一的id和隱藏的屬性集:

```
<input type="checkbox" id="sidebarshown" hidden>
<input type="checkbox" id="darkthemeused" hidden>
<!-- 這裡開始實際內容,例如: -->
<div id="container">
    <div id="sidebar">
        <!-- 選單、搜尋、... -->
    </div>
    <!-- 更多內容... -->
</div>
<div id="footer">
    <!-- ... -->
</div>
```

更改布林值

您可以透過新增帶有for屬性集的標籤來切換布林值:

```
<label for="sidebarshown">顯示/隱藏側邊欄！</label>
```

使用CSS存取布林值

普通選擇器(如.colorred)指定預設屬性. 它們可以透過遵循布林值(true|false)來覆蓋選擇器

```
/* true: */
element:checked ~ \[複選框的同級和目標的父級\] target

/* false: */
element:not(:checked) ~ \[複選框的同級和目標的父級\] target
```

element,[複選框的同級和目標的父級],target應替換為正確的選擇器.

[複選框的同級和目標的父級]可以是一個特定的選擇器,(通常如果你很懶的話)簡單地 * 或什麼都沒有.

上述 html 結構的範例如下:

```
#sidebarShown:checked ~ #container #sidebar{
    margin-left: 300px;
}

#darkThemeUsed:checked ~ #container, #darkThemeUsed:checked ~ #footer{
    background: #333333;
}
```

請參閱此連結以了解這些全域布林值的實作.

4-14節: :only-child 偽類選擇器範例

:only-child 偽類選擇器表示任何作為其父元素的唯一子元素的元素。 html:

```
<div>
    <p>此段落是div的唯一子級,它將具有藍色</p>
</div>
<div>
    <p>此段落是div的兩個子級之一</p>
    <p>此段落是其父級的兩個子級之一</p>
</div>
```

css:

```
p:only-child{
    color: blue;
}
```

上面的範例選擇<p>元素,它是其父元素中唯一的子元素,在本例中是 .

JSBin上的線上演示

Chapter 5: Backgrounds With css you can set colors, gradients, and images as the background of an element. It is possible to specify various combinations of images, colors, and gradients, and adjust the size, positioning, and repetition (among others) of these. Section 5.1: Background Color The background-color property sets the background color of an element using a color value or through keywords, such as transparent, inherit or initial. transparent, specifies that the background color should be transparent. This is default. inherit, inherits this property from its parent element. initial, sets this property to its default value. This can be applied to all elements, and ::first-letter/::first-line pseudo-elements. Colors in css can be specified by different methods. Color names css div { background-color: red; /* red */ } html

This will have a red background

The example used above is one of several ways that css has to represent a single color. Hex color codes Hex code is used to denote RGB components of a color in base-16 hexadecimal notation. #ff0000, for example, is bright red, where the red component of the color is 256 bits (ff) and the corresponding green and blue portions of the color is 0 (00). If both values in each of the three RGB pairings (R, G, and B) are the same, then the color code can be shortened into three characters (the first digit of each pairing). #ff0000 can be shortened to #f00, and #ffffff can be shortened to #fff. Hex notation is case-insensitive. body { background-color: #de1205; /* red */ } .main { background-color: #00f; /* blue */ } RGB / RGBa Another way to declare a color is to use RGB or RGBa. RGB stands for Red, Green and Blue, and requires of three separate values between 0 and 255, put between brackets, that correspond with the decimal color values for respectively red, green and blue. RGBa allows you to add an additional alpha parameter between 0.0 and 1.0 to define opacity. header { background-color: rgb(0, 0, 0); /* black */ } footer { background-color: rgba(0, 0, 0, 0.5); /* black with 50% opacity */ } HSL / HSLa Another way to declare a color is to use HSL or HSLa and is similar to RGB and RGBa. HSL stands for hue, saturation, and lightness, and is also often called HLS: Hue is a degree on the color wheel (from 0 to 360). Saturation is a percentage between 0% and 100%. Lightness is also a percentage between 0% and 100%. HSLa allows you to add an additional alpha parameter between 0.0 and 1.0 to define opacity. li a { background-color: hsl(120, 100%, 50%); /* green */ } #p1 { background-color: hsla(120, 100%, 50%, .3); /* green with 30% opacity */ } Interaction with background-image The following statements are all equivalent: body { background: red; background-image: url(partiallytransparentimage.png); } body { background-color: red; background-image: url(partiallytransparentimage.png); } body { background-image: url(partiallytransparentimage.png); background-color: red; } body { background: red url(partiallytransparentimage.png); } They will all lead to the red color being shown underneath the image, where the parts of the image are transparent, or the image is not showing (perhaps as a result of background-repeat). Note that the following is not equivalent: body { background-image: url(partiallytransparentimage.png); background: red; } Here, the value of background overrides your background-image. For more info on the background property, see Background Shorthand Section 5.2: Background Gradients Gradients are new image types, added in css3. As an image, gradients are set with the background-image property, or the background shorthand. There are two types of gradient functions, linear and radial. Each type has a non-repeating variant and a repeating variant: linear-gradient() repeating-linear-gradient() radial-gradient() repeating-radial-gradient() linear-gradient() A linear-gradient has the following syntax background: linear-gradient(?, , , ...); Value Meaning Could be an argument like to top, to bottom, to right or to left; or an angle as 0deg, 90deg... . The angle starts from to top and rotates clockwise. Can be specified in deg, grad, rad, or turn. If omitted, the gradient flows from top to bottom List of colors, optionally followed each one by a percentage or length to display it at. For example, yellow 10%, rgba(0,0,0,.5) 40px, #fff 100%... For example, this creates a linear gradient that starts from the right and transitions from red to blue .linear-gradient { background: linear-gradient(to left, red, blue); /* you can also use 270deg */ } You can create a diagonal gradient by declaring both a horizontal and vertical starting position. .diagonal-linear-gradient { background: linear-gradient(to left top, red, yellow 10%); } It is possible to specify any number of color stops in a gradient by separating them with commas. The following examples will create a gradient with 8 color stops .linear-gradient-rainbow { background: linear-gradient(to left, red, orange, yellow, green, blue, indigo, violet) } radial-gradient() .radial-gradient-simple { background: radial-gradient(red, blue); } .radial-gradient { background: radial-gradient(circle farthest-corner at top left, red, blue); } Value Meaning circle Shape of gradient. Values are circle or ellipse, default is ellipse. farthest-corner Keywords describing how big the ending shape must be. Values are closest-side, farthestside, closest-corner, farthest-corner top left Sets the position of the gradient center, in the same way as background-position. Repeating gradients Repeating gradient functions take the same arguments as the above examples, but tile the gradient across the background of the element. .bullseye { background: repeating-radial-gradient(red, red 10%, white 10%, white 20%); } .warning { background: repeating-linear-gradient(-45deg, yellow, yellow 10%, black 10%, black 20%); } Value Meaning -45deg Angle unit. The angle starts from to top and rotates clockwise. Can be specified in deg, grad, rad, or turn. to left Direction of gradient, default is to bottom. Syntax: to [y-axis(top OR bottom)] [x-axis(left OR right)] ie to top right yellow 10% Color, optionally followed by a percentage or length to display it at. Repeated two or more times. Note that HEX, RGB, RGBa, HSL, and HSLa color codes may be used instead of color names. Color names were used for the sake of illustration. Also note that the radial-gradient syntax is much more complex than linear-gradient, and a simplified version is shown here. For a full explanation and specs, see the MDN Docs Section 5.3: Background Image The background-image property is used to specify a background image to be applied to all matched elements. By default, this image is tiled to cover the entire element, excluding margin. .myClass { background-image: url('/path/to/image.jpg'); } To use multiple images as background-image, define comma separated url() .myClass { background-image: url('/path/to/image.jpg'), url('/path/to/image2.jpg'); } The images will stack according to their order with the first declared image on top of the others and so on. Value Result url('/path/to/image.jpg') Specify background image's path(s) or an image resource specified with data URI schema (apostrophes can be omitted), separate multiples by comma none No background image initial Default value inherit Inherit parent's value More css for Background Image This following attributes are very useful and almost essential too. background-size: xpx ypx | x% y%; background-repeat: no-repeat | repeat | repeat-x | repeat-y; background-position: left offset (px/%) right offset (px/%) | center center | left top | right bottom; Section 5.4: Background Shorthand The background property can be used to set one or more background related properties: Value Description css Ver. background-image Background image to use 1+ background-color Background color to apply 1+ background-position Background image's position 1+ background-size Background image's size 3+ background-repeat How to repeat background image 1+ background-origin How the background is positioned (ignored when background-attachment is fixed) 3+ background-clip How the background is painted relative to the content-box, border-box, or the padding-box 3+ background-attachment How the background image behaves, whether it scrolls along with its containing block or has a fixed position within the viewport 1+ initial Sets the property to value to default 3+ inherit Inherits property value from parent 2+ The order of the values does not matter and every value is optional Syntax The syntax of the background shorthand declaration is: background: [] [] []/[] [] [] [] [] []; Examples background: red; Simply setting a background-color with the redvalue. background: border-box red; Setting a background-clip to border-box and a background-color to red. background: no-repeat center url("somepng.jpg"); Sets a background-repeat to no-repeat, background-origin to center and a background-image to an image. background: url('pattern.png') green; In this example, the background-color of the element would be set to green with pattern.png, if it is available, overlayed on the colour, repeating as often as necessary to fill the element. If pattern.png includes any transparency then the green colour will be visible behind it. background: #000000 url("picture.png") top left / 600px auto no-repeat; In this example we have a black background with an image 'picture.png' on top, the image does not repeat in either axis and is positioned in the top left corner. The / after the position is to be able to include the size of the background image which in this case is set as 600px width and auto for the height. This example could work well with a feature image that can fade into a solid colour. NOTE: Use of the shorthand background property resets all previously set background property values, even if a value is not given. If you wish only to modify a background property value previously set, use a longhand property instead. Section 5.5: Background Size General overview The background-size property enables one to control the scaling of the background-image. It takes up to two values, which determine the scale/size of the resulting image in vertical and and horizontal direction. If the property is missing, its deemed auto in both width and height. auto will keep the image's aspect ratio, if it can be determined. The height is optional and can be considered auto. Therefore, on a 256 px × 256 px image, all the following background-size settings would yield an image with height and width of 50 px: background-size: 50px; background-size: 50px auto; /* same as above */ background-size: auto 50px; background-size: 50px 50px; So if we started with the following picture (which has the mentioned size of 256 px × 256 px), we'll end up with a 50 px × 50 px on the user's screen, contained in the background of our element: One can also use percentage

values to scale the image with respect of the element. The following example would yield a 200 px × 133 px drawn image: #withbackground { background-image: url(to/some/background.png); background-size: 100% 66%;

width: 200px; height: 200px; padding: 0; margin: 0; } The behaviour depends on the background-origin. Keeping the aspect ratio The last example in the previos section lost its original aspect ratio. The circle got into an ellipse, the square into a rectangle, the triangle into another triangle. The length or percentage approach isn't flexible enough to keep the aspect ratio at all times. auto doesn't help, since you might not know which dimension of your element will be larger. However, to cover certain areas with an image (and correct aspect ratio) completely or to contain an image with correct aspect ratio completely in a background area, the values, contain and cover provide the additional functionality. Eggsplanation for contain and cover Sorry for the bad pun, but we're going to use a picture of the day by Biswarup Ganguly for demonstration. Lets say that this is your screen, and the gray area is outside of your visible screen. For demonstration, We're going to assume a 16 × 9 ratio. We want to use the aforementioned picture of the day as a background. However, we cropped the image to 4x3 for some reason. We could set the background-size property to some fixed length, but we will focus on contain and cover. Note that I also assume that we didn't mangle the width and/or height of body. contain contain Scale the image, while preserving its intrinsic aspect ratio (if any), to the largest size such that both its width and its height can fit inside the background positioning area. This makes sure that the background image is always completely contained in the background positioning area, however, there could be some empty space filled with your background-color in this case: cover cover Scale the image, while preserving its intrinsic aspect ratio (if any), to the smallest size such that both its width and its height can completely cover the background positioning area. This makes sure that the background image is covering everything. There will be no visible background-color, however depending on the screen's ratio a great part of your image could be cut off: Demonstration with actual code div > div { background-image: url(http://i.stack.imgur.com/r5CAq.jpg); background-repeat: no-repeat; background-position: center center; background-color: #ccc; border: 1px solid; width: 20em; height: 10em; } div.contain { background-size: contain; } div.cover { background-size: cover; } /********** Additional styles for the explanation boxes **********/ div > div { margin: 0 1ex 1ex 0; float: left; } div + div { clear: both; border-top: 1px dashed silver; padding-top:1ex; } div > div::after { background-color: #000; color: #fefefe; margin: 1ex; padding: 1ex; opacity: 0.8; display: block; width: 10ex; font-size: 0.7em; content: attr(class); }

Note the grey background. The image does not cover the whole region, but it's fully *contained*.

Note the ducks/geese at the bottom of the image. Most of the water is cut, as well as a part of the sky. You don't see the complete image anymore, but neither do you see any background color; the image *covers* all of the `<div>`.

Section 5.6: Background Position The background-position property is used to specify the starting position for a background image or gradient .myClass { background-image: url('path/to/image.jpg'); background-position: 50% 50%; } The position is set using an X and Y co-ordinate and be set using any of the units used within css. Unit Description value% value% A percentage for the horizontal offset is relative to (width of background positioning area - width of background image). A percentage for the vertical offset is relative to (height of background positioning area - height of background image) The size of the image is the size given by background-size. valuepx valuepx Offsets background image by a length given in pixels relative to the top left of the background positioning area Units in css can be specified by different methods (see here). Longhand Background Position Properties In addition to the shorthand property above, one can also use the longhand background properties backgroundposition-x and background-position-y. These allow you to control the x or y positions separately. NOTE: This is supported in all browsers except Firefox (versions 31-48) 2. Firefox 49, to be released September 2016, will support these properties. Until then, there is a Firefox hack within this Stack Overflow answer. Section 5.7: The background-origin property The background-origin property specifies where the background image is positioned. Note: If the background-attachment property is set to fixed, this property has no effect. Default value: padding-box Possible values: padding-box - The position is relative to the padding box border-box - The position is relative to the border box content-box - The position is relative to the content box initial inherit css .example { width: 300px; border: 20px solid black; padding: 50px; background: url(https://static.pexels.com/photos/6440/magazines-desk-work-workspace-medium.jpg); background-repeat: no-repeat; } .example1 {} .example2 { background-origin: border-box; } .example3 { background-origin: content-box; } html

No background-origin (padding-box is default):

Lorem Ipsum Dolor

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

background-origin: border-box:

Lorem Ipsum Dolor

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

background-origin: content-box:

Lorem Ipsum Dolor

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Result: More: https://www.w3.org/TR/css3-background/#the-background-origin https://developer.mozilla.org/en-US/docs/Web/css/background-origin Section 5.8: Multiple Background Image In css3, we can stack multiple background in the same element. #mydiv { background-image: url(img_1.png), /* top image */ url(img_2.png), /* middle image */ url(img_3.png); /* bottom image */ background-position: right bottom, left top, right top; background-repeat: no-repeat, repeat, no-repeat; } Images will be stacked atop one another with the first background on top and the last background in the back. img_1 will be on top, the img_2 and img_3 is on bottom. We can

also use background shorthand property for this: #mydiv { background: url(img_1.png) right bottom no-repeat, url(img_2.png) left top repeat, url(img_3.png) right top no-repeat; } We can also stack images and gradients: #mydiv { background: url(image.png) right bottom no-repeat, linear-gradient(to bottom, #fff 0%,#000 100%); } Demo Section 5.9: Background Attachment The background-attachment property sets whether a background image is fixed or scrolls with the rest of the page. body { background-image: url('img.jpg'); background-attachment: fixed; } Value Description scroll The background scrolls along with the element. This is default. fixed The background is fixed with regard to the viewport. local The background scrolls along with the element's contents. initial Sets this property to its default value. inherit Inherits this property from its parent element. Examples background-attachment: scroll The default behaviour, when the body is scrolled the background scrolls with it: body { background-image: url('image.jpg'); background-attachment: scroll; } background-attachment: fixed The background image will be fixed and will not move when the body is scrolled: body { background-image: url('image.jpg'); background-attachment: fixed; } background-attachment: local The background image of the div will scroll when the contents of the div is scrolled. div { background-image: url('image.jpg'); background-attachment: local; } Section 5.10: Background Clip Definition and Usage: The background-clip property specifies the painting area of the background. Default value: border-box Values border-box is the default value. This allows the background to extend all the way to the outside edge of the element's border. padding-box clips the background at the outside edge of the element's padding and does not let it extend into the border; content-box clips the background at the edge of the content box. inherit applies the setting of the parent to the selected element. css .example { width: 300px; border: 20px solid black; padding: 50px; background: url(https://static.pexels.com/photos/6440/magazines-desk-work-workspace-medium.jpg); background-repeat: no-repeat; } .example1 {} .example2 { background-origin: border-box; } .example3 { background-origin: content-box; } html

No background-origin (padding-box is default):

Lorem Ipsum Dolor

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

background-origin: border-box:

Lorem Ipsum Dolor

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

background-origin: content-box:

Lorem Ipsum Dolor

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat.

Section 5.11: Background Repeat The background-repeat property sets if/how a background image will be repeated. By default, a background-image is repeated both vertically and horizontally. div { background-image: url("img.jpg"); background-repeat: repeat-y; } Here's how a background-repeat: repeat-y looks like: Section 5.12: background-blend-mode Property .my-div { width: 300px; height: 200px; background-size: 100%; background-repeat: no-repeat; background-image: linear-gradient(to right, black 0%,white 100%), url('https://static.pexels.com/photos/54624/strawberry-fruit-red-sweet-54624-medium.jpeg'); background-blend-mode:saturation; } Lorem ipsum

See result here: https://jsfiddle.net/MadalinaTn/y69d28Lb/ css Syntax: background-blend-mode: normal | multiply | screen | overlay | darken | lighten | color-dodge | saturation | color | luminosity; Section 5.13: Background Color with Opacity If you set opacity on an element it will affect all its child elements. To set an opacity just on the background of an element you will have to use RGBA colors. Following example will have a black background with 0.6 opacity. /* Fallback for web browsers that don't support RGBa */ background-color: rgb(0, 0, 0); /* RGBa with 0.6 opacity */ background-color: rgba(0, 0, 0, 0.6); /* For IE 5.5 - 7*/ filter: progid:DXImageTransform.Microsoft.gradient(startColorstr=#99000000, endColorstr=#99000000); /* For IE 8*/ -ms-filter: "progid:DXImageTransform.Microsoft.gradient(startColorstr=#99000000, endColorstr=#99000000)"; Chapter 6: Centering Section 6.1: Using Flexbox html: css: html, body, .container { height: 100%; } .container { display: flex; justify-content: center; /* horizontal center */ } img { align-self: center; /* vertical center */ } View Result html: css: html, body { height: 100%; } body { display: flex; justify-content: center; /* horizontal center */ align-items: center; /* vertical center */ } View Result See Dynamic Vertical and Horizontal Centering under the Flexbox documentation for more details on flexbox and what the styles mean. Browser Support Flexbox is supported by all major browsers, except IE versions before 10. Some recent browser versions, such as Safari 8 and IE10, require vendor prefixes. For a quick way to generate prefixes there is Autoprefixer, a third-party tool. For older browsers (like IE 8 & 9) a Polyfill is available. For a more detailed look at flexbox browser support, see this answer. Section 6.2: Using css transform css transforms are based on the size of the elements so if you don't know how tall or wide your element is, you can position it absolutely 50% from the top and left of a relative container and translate it by 50% left and upwards to center it vertically and horizontally. Keep in mind that with this technique, the element could end being rendered at a non-integer pixel boundary, making it look blurry. See this answer in SO for a workaround. html css .container { position: relative; } .element { position: absolute; top: 50%; left: 50%; transform: translate(-50%, -50%); } View example in JSFiddle CROSS BROWSER COMPATIBILITY The transform property needs prefixes to be supported by older browsers. Prefixes are needed for Chrome<=35, 0 9 safari<="8," opera<="22," android browser<="4.4.4," and ie9. css transforms are not supported by ie8 older versions. here is a common transform declaration for the previous example: -webkit-transform: translate(-50%, -50%); * chrome, safari, opera, -ms-transform: ie transform: more information see caniuse. element being positioned according to first non-static parent (position: relative, absolute, or fixed). explore in this fiddle documentation topic. horizontal-only centering, use left: 50% translatex(-50%). same goes verticalonly centering: center with top: translatey(-50%). using width height elements method of centering can cause centered appear squished. mostly happens containing text, be fixed adding: marginright: -50%; margin-bottom: -50%;. view information. section 6.3: margin: auto; objects if they block have defined width. html

This is a centered paragraph.

css .containerDiv { width: 100%; height: 100px; padding-bottom: 40px; } #centeredDiv { margin: 0 auto; width: 200px; height: 100px; border: 1px solid #000; } #centeredParagraph { width: 200px; margin: 0 auto; } #centeredImage { display: block; width: 200px; margin: 0 auto; } Result: JSFiddle example: Centering objects with

margin: 0 auto; Section 6.4: Using text-align The most common and easiest type of centering is that of lines of text in an element. css has the rule text-align: center for this purpose: html

Lorem ipsum

css p { text-align: center; } This does not work for centering entire block elements. text-align controls only alignment of inline content like text in its parent block element. See more about text-align in Typography section. Section 6.5: Using position: absolute Working in old browsers (IE >= 8) Automatic margins, paired with values of zero for the left and right or top and bottom offsets, will center an absolutely positioned elements within its parent. View Result html

css .parent { position: relative; height: 500px; } .center { position: absolute; margin: auto; top: 0; right: 0; bottom: 0; left: 0; } Elements that don't have their own implicit width and height like images do, will need those values defined. Other resources: Absolute Centering in css Section 6.6: Using calc() The calc() function is the part of a new syntax in css3 in which you can calculate (mathematically) what size/position your element occupies by using a variety of values like pixels, percentages, etc. Note: Whenever you use this function, always take care of the space between two values calc(100% - 80px). css .center { position: absolute; height: 50px; width: 50px; background: red; top: calc(50% - 50px / 2); /* height divided by 2*/ left: calc(50% - 50px / 2); /* width divided by 2*/ } html

Section 6.7: Using line-height You can also use line-height to center vertically a single line of text inside a container : css div { height: 200px; line-height: 200px; } That's quite ugly, but can be useful inside an element. The line-height property works only when the text to be centered spans a single line. If the text wraps into multiple lines, the resulting output won't be centered. Section 6.8: Vertical align anything with 3 lines of code Supported by IE11+ View Result Use these 3 lines to vertical align practically everything. Just make sure the div/image you apply the code to has a parent with a height. css div.vertical { position: relative; top: 50%; transform: translateY(-50%); } html

Vertical aligned text!

Section 6.9: Centering in relation to another item We will see how to center content based on the height of a near element. Compatibility: IE8+, all other modern browsers. html

text 1

content content content content content content content content content content content content content content

button

css .content * { box-sizing: border-box; } .content .position-container { display: table; } .content .details { display: table-cell; vertical-align: middle; width: 33.333333%; padding: 30px; font-size: 17px; text-align: center; } .content .thumb { width: 100%; } .content .thumb img { width: 100%; } Link to JSFiddle The main points are the 3 .thumb, .details and .position-container containers: The .position-container must have display: table. The .details must have the real width set width: .... and display: table-cell, vertical-align: middle. The .thumb must have width: 100% if you want that it will take all the remaining space and it will be influenced by the .details width. The image (if you have an image) inside .thumb should have width: 100%, but it is not necessary if you have correct proportions. Section 6.10: Ghost element technique (Michał Czernow's hack) This technique works even when the container's dimensions are unknown. Set up a "ghost" element inside the container to be centered that is 100% height, then use vertical-align: middle on both that and the element to be centered. css /* This parent can be any width and height */ .block { text-align: center; /* May want to do this if there is risk the container may be narrower than the element inside */ white-space: nowrap; } /* The ghost element */ .block:before { content: ''; display: inline-block; height: 100%; vertical-align: middle; /* There is a gap between ghost element and .centered, caused by space character rendered. Could be eliminated by nudging .centered (nudge distance depends on font family), or by zeroing font-size in .parent and resetting it back (probably to 1rem) in .centered. */ margin-right: -0.25em; } /* The element to be centered, can also be of any width and height */ .centered { display: inline-block; vertical-align: middle; width: 300px; white-space: normal; /* Resetting inherited nowrap behavior */ } html

Section 6.11: Centering vertically and horizontally without worrying about height or width The following technique allows you to add your content to an html element and center it both horizontally and vertically without worrying about its height or width. The outer container should have display: table; The inner container should have display: table-cell; should have vertical-align: middle; should have text-align: center; The content box should have display: inline-block; should re-adjust the horizontal text-alignment to eg. text-align: left; or text-align: right;, unless you want text to be centered Demo html

You can put anything here!

css body { margin : 0; } .outer-container { position : absolute; display: table; width: 100%; /* This could be ANY width */ height: 100%; /* This could be ANY height */ background: #ccc; } .inner-container { display: table-cell; vertical-align: middle; text-align: center; } .centered-content { display: inline-block; text-align: left; background: #fff; padding: 20px; border: 1px solid #000; } See also this Fiddle! Section 6.12: Vertically align an image inside div html

css .wrap { height: 50px;/* max image height */ width: 100px; border: 1px solid blue; text-align: center; } .wrap:before { content:""; display: inline-block; height: 100%; vertical-align: middle; width: 1px; } img { vertical-align: middle; } Section 6.13: Centering with fixed size If the size of your content is fixed, you can use absolute positioning to 50% with margin that reduces half of your content's width and height: html

Center vertically and horizontally

css .center { position: absolute; background: #ccc; left: 50%; width: 150px; margin-left: -75px; /* width * -0.5 */ top: 50%; height: 200px; margin-top: -100px; /* height * -0.5 */ } Horizontal centering with only fixed width You can center the element horizontally even if you don't know the height of the content: html

Center only horizontally

css .center { position: absolute; background: #ccc; left: 50%; width: 150px; margin-left: -75px; /* width * -0.5 */ } Vertical centering with fixed height You can center the element vertically if you know the element's height: html

Center only vertically

css .center { position: absolute; background: #ccc; top: 50%; height: 200px; margin-top: -100px; /* width * -0.5 */ } Section 6.14: Vertically align dynamic height elements Applying css intuitively doesn't produce the desired results because vertical-align:middle isn't applicable to block-level elements margin-top:auto and margin-

bottom:auto used values would compute as zero margin-top:-50% percentage-based margin values are calculated relative to the width of containing block For widest browser support, a workaround with helper elements: html
css .vcenter--container { display: table; height: 100%; position: absolute; overflow: hidden; width: 100%; } .vcenter--helper { display: table-cell; vertical-align: middle; } .vcenter--content { margin: 0 auto; width: 200px; } jsfiddle from original question. This approach works with dynamic height elements respects content flow is supported by legacy browsers Section 6.15: Horizontal and Vertical centering using table layout One could easily center a child element using table display property. html
css .wrapper { display: table; vertical-align: center; width: 200px; height: 200px; background-color: #9e9e9e; } .parent { display: table-cell; vertical-align: middle; text-align: center; } .child { display: inline-block; vertical-align: middle; text-align: center; width: 100px; height: 100px; background-color: teal; } Chapter 7: The Box Model Parameter Detail content-box Width and height of the element only includes content area. padding-box Width and height of the element includes content and padding. border-box Width and height of the element includes content, padding and border. initial Sets the box model to its default state. inherit Inherits the box model of the parent element. Section 7.1: What is the Box Model? The Edges The browser creates a rectangle for each element in the html document. The Box Model describes how the padding, border, and margin are added to the content to create this rectangle. Diagram from css2.2 Working Draft The perimeter of each of the four areas is called an edge. Each edge defines a box. The innermost rectangle is the content box. The width and height of this depends on the element's rendered content (text, images and any child elements it may have). Next is the padding box, as defined by the padding property. If there is no padding width defined, the padding edge is equal to the content edge. Then we have the border box, as defined by the border property. If there is no border width defined, the border edge is equal to the padding edge. The outermost rectangle is the margin box, as defined by the margin property. If there is no margin width defined, the margin edge is equal to the border edge. Example div { border: 5px solid red; margin: 50px; padding: 20px; } This css styles all div elements to have a top, right, bottom and left border of 5px in width; a top, right, bottom and left margin of 50px; and a top, right, bottom, and left padding of 20px. Ignoring content, our generated box will look like this: Screenshot of Google Chrome's Element Styles panel As there is no content, the content region (the blue box in the middle) has no height or width (0px by 0px). The padding box by default is the same size as the content box, plus the 20px width on all four edges we're defining above with the padding property (40px by 40px). The border box is the same size as the padding box, plus the 5px width we're defining above with the border property (50px by 50px). Finally the margin box is the same size as the border box, plus the 50px width we're defining above with the margin property (giving our element a total size of 150px by 150px). Now lets give our element a sibling with the same style. The browser looks at the Box Model of both elements to work out where in relation to the previous element's content the new element should be positioned: The content of each of element is separated by a 150px gap, but the two elements' boxes touch each other. If we then modify our first element to have no right margin, the right margin edge would be in the same position as the right border edge, and our two elements would now look like this: Section 7.2: box-sizing The default box model (content-box) can be counter-intuitive, since the width / height for an element will not represent its actual width or height on screen as soon as you start adding padding and border styles to the element. The following example demonstrates this potential issue with content-box: textarea { width: 100%; padding: 3px; box-sizing: content-box; /* default value */ } Since the padding will be added to the width of the textarea, the resulting element is a textarea that is wider than 100%. Fortunately, css allows us to change the box model with the box-sizing property for an element. There are three different values for the property available: content-box: The common box model - width and height only includes the content, not the padding or border padding-box: Width and height includes the content and the padding, but not the border border-box: Width and height includes the content, the padding as well as the border To solve the textarea problem above, you could just change the box-sizing property to padding-box or borderbox. border-box is most commonly used. textarea { width: 100%; padding: 3px; box-sizing: border-box; } To apply a specific box model to every element on the page, use the following snippet: html { box-sizing: border-box; } *, *:before, *:after { box-sizing: inherit; } In this coding box-sizing:border-box; is not directly applied to *, so you can easily overwrite this property on individual elements. Chapter 8: Margins Parameter Details 0 set margin to none auto used for centering, by evenly setting values on each side units (e.g. px) see parameter section in Units for a list of valid units inherit inherit margin value from parent element initial restore to initial value Section 8.1: Margin Collapsing When two margins are touching each other vertically, they are collapsed. When two margins touch horizontally, they do not collapse. Example of adjacent vertical margins: Consider the following styles and markup: div{ margin: 10px; }
some content
some more content
They will be 10px apart since vertical margins collapse over one and other. (The spacing will not be the sum of two margins.) Example of adjacent horizontal margins: Consider the following styles and markup: span{ margin: 10px; } somecontent They will be 20px apart since horizontal margins don't collapse over one and other. (The spacing will be the sum of two margins.) Overlapping with different sizes .top{ margin: 10px; } .bottom{ margin: 15px; }
some content
some more content
These elements will be spaced 15px apart vertically. The margins overlap as much as they can, but the larger margin will determine the spacing between the elements. Overlapping margin gotcha .outer-top{ margin: 10px; } .inner-top{ margin: 15px; } .outer-bottom{ margin: 20px; } .inner-bottom{ margin: 25px; }
some content
some more content
What will be the spacing between the two texts? (hover to see answer) The spacing will be 25px. Since all four margins are touching each other, they will collapse, thus using the largest margin of the four. Now, what about if we add some borders to the markup above. div{ border: 1px solid red; } What will be the spacing between the two texts? (hover to see answer) The spacing will be 59px! Now only the margins of .outer-top and .outer-bottom touch each other, and are the only collapsed margins. The remaining margins are separated by the borders. So we have 1px + 10px + 1px + 15px + 20px + 1px + 25px + 1px. (The 1px's are the borders...) Collapsing Margins Between Parent and Child Elements: html:

Title

Paragraph

css h1 { margin: 0; background: #cff; } div { margin: 50px 0 0 0; background: #cfc; } p { margin: 25px 0 0 0; background: #cf9; } In the example above, only the largest margin applies. You may have expected that the paragraph would be located 60px from the h1 (since the div element has a margin-top of 40px and the p has a 20px margin-top). This does not happen because the margins collapse together to form one margin. Section 8.2: Apply Margin on a Given Side Direction-Specific Properties css allows you to specify a given side to apply margin to. The four properties provided for this purpose are: margin-left margin-right margin-top margin-bottom The following code would apply a margin of 30 pixels to the left side of the selected div. View Result html
css #myDiv { margin-left: 30px; height: 40px; width: 40px; background-color: red; } Parameter Details margin-left The direction in which the margin should be applied. 30px The width of the margin. Specifying Direction Using Shorthand Property The standard margin property can be expanded to specify differing widths to each side of the selected elements. The syntax for doing this is as follows: margin: ; The following example applies a zero-width margin to the top of the div, a 10px margin to the right side, a 50px margin to the left side, and a 100px margin to the left side. View Result html
css #myDiv { margin: 0 10px 50px 100px; height: 40px; width: 40px; background-color: red; } Section 8.3: Margin property simplification p { margin:1px; /* 1px margin in all directions */

/equals to:/

margin:1px 1px;

/equals to:/

margin:1px 1px 1px;

/equals to:/

margin:1px 1px 1px 1px; } Another exapmle: p{ margin:10px 15px; /* 10px margin-top & bottom And 15px margin-right & left*/

/equals to:/

margin:10px 15px 10px 15px;

/equals to:/

margin:10px 15px 10px; /* margin left will be calculated from the margin right value (=15px) */ } *Section 8.4: Horizontally center elements on a page using margin As long as the element is a block, and it has an explicitly set width value, margins can be used to center block elements on a page horizontally. We add a width value that is lower than the width of the window and the auto property of margin then distributes the remaining space to the left and the right: #myDiv { width:80%; margin:0 auto; } In the example above we use the shorthand margin declaration to first set 0 to the top and bottom margin values (although this could be any value) and then we use auto to let the browser allocate the space automatically to the left and right margin values. In the example above, the #myDiv element is set to 80% width which leaves use 20% leftover. The browser distributes this value to the remaining sides so: (100% - 80%) / 2 = 10% Section 8.5: Example 1: It is obvious to assume that the percentage value of margin to margin-left and margin-right would be relative to its parent element. .parent { width : 500px; height: 300px; } .child { width : 100px; height: 100px; margin-left: 10%; /* (parentWidth * 10/100) => 50px */ } But that is not the case, when comes to margin-top and margin-bottom. Both these properties, in percentages, aren't relative to the height of the parent container but to the width of the parent container. So, .parent { width : 500px; height: 300px; } .child { width : 100px; height: 100px; margin-left: 10%; /* (parentWidth * 10/100) => 50px */ margin-top: 20%; /* (parentWidth * 20/100) => 100px */ }* Section 8.6: Negative margins Margin is one of a few css properties that can be set to negative values. This property can be used to overlap elements without absolute positioning. div{ display: inline; } #over{ margin-left: -20px; }

Base div
Overlapping div

Chapter 9: Padding Section 9.1: Padding Shorthand The padding property sets the padding space on all sides of an element. The padding area is the space between the content of the element and its border. Negative values are not allowed. To save adding padding to each side individually (using padding-top, padding-left etc) can you write it as a shorthand, as below: Four values:

Three values:

Two values:

One value:

Section 9.2: Padding on a given side The padding property sets the padding space on all sides of an element. The padding area is the space between the content of the element and its border. Negative values are not allowed. You can specify a side individually: padding-top padding-right padding-bottom padding-left The following code would add a padding of 5px to the top of the div:

Chapter 10: Border Section 10.1: border-radius The border-radius property allows you to change the shape of the basic box model. Every corner of an element can have up to two values, for the vertical and horizontal radius of that corner (for a maximum of 8 values). The first set of values defines the horizontal radius. The optional second set of values, preceded by a '/' , defines the vertical radius. If only one set of values is supplied, it is used for both the vertical and horizontal radius. border-radius: 10px 5% / 20px 25em 30px 35em; The 10px is the horizontal radius of the top-left-and-bottom-right. And the 5% is the horizontal radius of the topright-and-bottom-left. The other four values after '/' are the vertical radii for top-left, top-right, bottom-right and bottom-left. As with many css properties, shorthands can be used for any or all possible values. You can therefore specify anything from one to eight values. The following shorthand allows you to set the horizontal and vertical radius of every corner to the same value: html:
css: .box { width: 250px; height: 250px; background-color: black; border-radius: 10px; } Border-radius is most commonly used to convert box elements into circles. By setting the border-radius to half of the length of a square element, a circular element is created: .circle { width: 200px; height: 200px; border-radius: 100px; } Because border-radius accepts percentages, it is common to use 50% to avoid manually calculating the borderradius value: .circle { width: 150px; height: 150px; border-radius: 50%; } If the width and height properties are not equal, the resulting shape will be an oval rather than a circle. Browser specific border-radius example: -webkit-border-top-right-radius: 4px; -webkit-border-bottom-right-radius: 4px; -webkit-border-bottom-left-radius: 0; -webkit-border-top-left-radius: 0; -moz-border-radius-topright: 4px; -moz-border-radius-bottomright: 4px; -moz-border-radius-bottomleft: 0; -moz-border-radius-topleft: 0; border-top-right-radius: 4px; border-bottom-right-radius: 4px; border-bottom-left-radius: 0; border-top-left-radius: 0; Section 10.2: border-style The border-style property sets the style of an element's border. This property can have from one to four values (for every side of the element one value.) Examples: border-style: dotted; border-style: dotted solid double dashed; border-style can also have the values none and hidden. They have the same effect, except hidden works for border conflict resolution for elements. In a
with multiple borders, none has the lowest priority (meaning in a conflict, the border would show), and hidden has the highest priority (meaning in a conflict, the border would not show). Section 10.3: Multiple Borders Using outline: .div1{ border: 3px solid black; outline: 6px solid blue; width: 100px; height: 100px; margin: 20px; } Using box-shadow: .div2{ border: 5px solid green; box-shadow: 0px 0px 0px 4px #000; width: 100px; height: 100px; margin: 20px; } Using a pseudo element: .div3 { position: relative; border: 5px solid #000; width: 100px; height: 100px; margin: 20px; } .div3:before { content: " "; position: absolute; border: 5px solid blue; z-index: -1; top: 5px; left:

5px; right: 5px; bottom: 5px; } http://jsfiddle.net/MadalinaTn/bvqpcohm/2/ Section 10.4: border (shorthands) In most cases you want to define several border properties (border-width, border-style and border-color) for all sides of an element. Instead of writing: border-width: 1px; border-style: solid; border-color: #000; You can simply write: border: 1px solid #000; These shorthands are also available for every side of an element: border-top, border-left, border-right and border-bottom. So you can do: border-top: 2px double #aaaaaa; Section 10.5: border-collapse The border-collapse property applies only to tables (and elements displayed as display: table or inlinetable) and sets whether the table borders are collapsed into a single border or detached as in standard html. table { border-collapse: separate; /* default */ border-spacing: 2px; /* Only works if border-collapse is separate */ } Also see Tables - border-collapse documentation entry Section 10.6: border-image With the border-image property you have the possibility to set an image to be used instead of normal border styles. A border-image essentially consist of a border-image-source: The path to the image to be used border-image-slice: Specifies the offset that is used to divide the image into nine regions (four corners, four edges and a middle) border-image-repeat: Specifies how the images for the sides and the middle of the border image are scaled Consider the following example wheras border.png is a image of 90x90 pixels: border-image: url("border.png") 30 stretch; The image will be split into nine regions with 30x30 pixels. The edges will be used as the corners of the border while the side will be used in between. If the element is higher / wider than 30px this part of the image will be stretched. The middle part of the image defaults to be transparent. Section 10.7: Creating a multi-colored border using borderimage css .bordered { border-image: linear-gradient(to right, red 20%, green 20%, green 40%, blue 40%, blue 60%, maroon 60%, maroon 80%, chocolate 80%); /* gradient with required colors */ border-image-slice: 1; } html
Border on all sides
The above example would produce a border that comprises of 5 different colors. The colors are defined through a linear-gradient (you can find more information about gradients in the docs). You can find more information about border-image-slice property in the border-image example in same page. (Note: Additional properties were added to the element for presentational purpose.) You'd have noticed that the left border has only a single color (the start color of the gradient) while the right border also has only a single color (the gradient's end color). This is because of the way that border image property works. It is as though the gradient is applied to the entire box and then the colors are masked from the padding and content areas, thus making it look as though only the border has the gradient. Which border(s) have a single color is dependant on the gradient definition. If the gradient is a to right gradient, the left border would be the start color of the gradient and right border would be the end color. If it was a to bottom gradient the top border would be the gradient's start color and bottom border would be end color. Below is the output of a to bottom 5 colored gradient. If the border is required only on specific sides of the element then the border-width property can be used just like with any other normal border. For example, adding the below code would produce a border only on the top of the element. border-width: 5px 0px 0px 0px; Note that, any element that has border-image property won't respect the border-radius (that is the border won't curve). This is based on the below statement in the spec: A box's backgrounds, but not its border-image, are clipped to the appropriate curve (as determined by 'background-clip'). Section 10.8: border-[left|right|top|bottom] The border-[left|right|top|bottom] property is used to add a border to a specific side of an element. For example if you wanted to add a border to the left side of an element, you could do: #element { border-left: 1px solid black; } Chapter 11: Outlines Parameter Details dotted dotted outline dashed dashed outline solid solid outline double double outline groove 3D grooved outline, depends on the outline-color value ridge 3D ridged outline, depends on the outline-color value inset 3D inset outline, depends on the outline-color value outset 3D outset outline, depends on the outline-color value none no outline hidden hidden outline Section 11.1: Overview Outline is a line that goes around the element, outside of the border. In contrast to border, outlines do not take any space in the box model. So adding an outline to an element does not affect the position of the element or other elements. In addition, outlines can be non-rectangular in some browsers. This can happen if outline is applied on a span element that has text with different font-size properties inside it. Unlike borders, outlines cannot have rounded corners. The essential parts of outline are outline-color, outline-style and outline-width. The definition of an outline is equivalent to the definition of a border: An outline is a line around an element. It is displayed around the margin of the element. However, it is different from the border property. outline: 1px solid black; Section 11.2: outline-style The outline-style property is used to set the style of the outline of an element. p { border: 1px solid black; outline-color:blue; line-height:30px; } .p1{ outline-style: dotted; } .p2{ outline-style: dashed; } .p3{ outline-style: solid; } .p4{ outline-style: double; } .p5{ outline-style: groove; } .p6{ outline-style: ridge; } .p7{ outline-style: inset; } .p8{ outline-style: outset; } html

A dotted outline

A dashed outline

A solid outline

A double outline

A groove outline

A ridge outline

An inset outline

An outset outline

Chapter 12: Overflow Overflow Value Details visible Shows all overflowing content outside the element scroll Hides the overflowing content and adds a scroll bar hidden Hides the overflowing content, both scroll bars disappear and the page becomes fixed auto Same as scroll if content overflows, but doesn't add scroll bar if content fits inherit Inherit's the parent element's value for this property Section 12.1: overflow-wrap overflow-wrap tells a browser that it can break a line of text inside a targeted element onto multiple lines in an otherwise unbreakable place. Helpful in preventing an long string of text causing layout problems due to overflowing it's container. css div { width:100px; outline: 1px dashed #bbb; } #div1 { overflow-wrap:normal; } #div2 { overflow-wrap:break-word; } html
#div1: Small words are displayed normally, but a long word like supercalifragilisticexpialidocious is too long so it will overflow past the edge of the line-break
#div2: Small words are displayed normally, but a long word like supercalifragilisticexpialidocious will be split at the line break and continue on the next line.
overflow-wrap – Value Details normal Lets a word overflow if it is longer than the line break-word Will split a word into multiple lines, if necessary inherit Inherits the parent element's value for this property Section 12.2: overflow-x and overflow-y These two properties work in a similar fashion as the overflow property and accept the same values. The overflow-x parameter works only on the x or left-to-right axis. The overflow-y works on the y or top-to-bottom axis. html
If this div is too small to display its contents, the content to the left and right will be clipped.
If this div is too small to display its contents, the content to the top and bottom will be clipped.
css div { width: 200px; height: 200px; } #div-x { overflow-x: hidden; } #div-y { overflow-y: hidden; } Section 12.3: overflow: scroll html
This div is too small to display its contents to display the effects of the overflow property.
css div { width:100px; height:100px; overflow:scroll; } Result The content above is clipped in a 100px by 100px box, with scrolling available to view overflowing content. Most desktop browsers will display both horizontal and vertical scrollbars, whether or not any content is clipped. This can avoid problems with scrollbars appearing and disappearing in a dynamic environment. Printers may print overflowing content. Section 12.4: overflow: visible html
Even if this div is too small to display its contents, the content is not clipped.

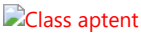
css div { width:50px; height:50px; overflow:visible; } Result Content is not clipped and will be rendered outside the content box if it exceeds its container size. Section 12.5: Block Formatting Context Created with Overflow Using the overflow property with a value different to visible will create a new block formatting context. This is useful for aligning a block element next to a floated element. css img { float:left; margin-right: 10px; } div { overflow:hidden; /* creates block formatting context */ } html

Lorem ipsum dolor sit amet, cum no paulo mollis pertinacia.

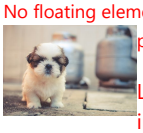
Ad case omnis nam, mutat deseruisse persequeris eos ad, in tollit debitis sea.

Result This example shows how paragraphs within a div with the overflow property set will interact with a floated image. Chapter 13: Media Queries Parameter Details mediatype (Optional) This is the type of media. Could be anything in the range of all to screen. not (Optional) Doesn't apply the css for this particular media type and applies for everything else. media feature Logic to identify use case for css. Options outlined below. Media Feature Details aspect-ratio Describes the aspect ratio of the targeted display area of the output device. color Indicates the number of bits per color component of the output device. If the device is not a color device, this value is zero. color-index Indicates the number of entries in the color look-up table for the output device. grid Determines whether the output device is a grid device or a bitmap device. height The height media feature describes the height of the output device's rendering surface. max-width css will not apply on a screen width wider than specified. min-width css will not apply on a screen width narrower than specified. max-height css will not apply on a screen height taller than specified. min-height css will not apply on a screen height shorter than specified. monochrome Indicates the number of bits per pixel on a monochrome (greyscale) device. orientation css will only display if device is using specified orientation. See remarks for more details. resolution Indicates the resolution (pixel density) of the output device. scan Describes the scanning process of television output devices. width The width media feature describes the width of the rendering surface of the output device (such as the width of the document window, or the width of the page box on a printer). Deprecated Features Details device-aspect-ratio Deprecated css will only display on devices whose height/width ratio matches the specified ratio. This is adeprecatedfeature and is not guaranteed to work. max-device-width Deprecated Same as max-width but measures the physical screen width, rather than the display width of the browser. min-device-width Deprecated Same as min-width but measures the physical screen width, rather than the display width of the browser. max-device-height Deprecated Same as max-height but measures the physical screen width, rather than the display width of the browser. min-device-height Deprecated Same as min-height but measures the physical screen width, rather than the display width of the browser. Section 13.1: Terminology and Structure Media queries allow one to apply css rules based on the type of device / media (e.g. screen, print or handheld) called media type, additional aspects of the device are described with media features such as the availability of color or viewport dimensions. General Structure of a Media Query @media [...] { /* One or more css rules to apply when the query is satisfied */ } A Media Query containing a Media Type @media print { /* One or more css rules to apply when the query is satisfied */ } A Media Query containing a Media Type and a Media Feature @media screen and (max-width: 600px) { /* One or more css rules to apply when the query is satisfied */ } A Media Query containing a Media Feature (and an implicit Media Type of "all") @media (orientation: portrait) { /* One or more css rules to apply when the query is satisfied */ } Section 13.2: Basic Example @media screen and (min-width: 720px) { body { background-color: skyblue; } } The above media query specifies two conditions: 1. The page must be viewed on a normal screen (not a printed page, projector, etc). 2. The width of the user's view port must be at least 720 pixels. If these conditions are met, the styles inside the media query will be active, and the background color of the page will be sky blue. Media queries are applied dynamically. If on page load the conditions specified in the media query are met, the css will be applied, but will be immediately disabled should the conditions cease to be met. Conversely, if the conditions are initially not met, the css will not be applied until the specified conditions are met. In our example, if the user's view port width is initially greater than 720 pixels, but the user shrinks the browser's width, the background color will cease to be sky blue as soon as the user has resized the view port to less than 720 pixels in width. Section 13.3: mediatype Media queries have an optional mediatype parameter. This parameter is placed directly after the @media declaration (@media mediatype), for example: @media print { html { background-color: white; } } The above css code will give the DOM html element a white background color when being printed. The mediatype parameter has an optional not or only prefix that will apply the styles to everything except the specified mediatype or only the specified media type, respectively. For example, the following code example will apply the style to every media type except print. @media not print { html { background-color: green; } } And the same way, for just showing it only on the screen, this can be used: @media only screen { .fadeInEffects { display: block; } } The list of mediatype can be understood better with the following table: Media Type Description all Apply to all devices screen Default computers print Printers in general. Used to style print-versions of websites handheld PDA's, cellphones and hand-held devices with a small screen projection For projected presentation, for example projectors aural Speech Systems braille Braille tactile devices embossed Paged braille printers tv Television-type devices tty Devices with a fixed-pitch character grid. Terminals, portables. Section 13.4: Media Queries for Retina and Non Retina Screens Although this works only for WebKit based browsers, this is helpful: /* ---------- Non-Retina Screens ---------- */ @media screen and (min-width: 1200px) and (max-width: 1600px) and (-webkit-min-device-pixel-ratio: 1) { } /* ---------- Retina Screens ---------- */ @media screen and (min-width: 1200px) and (max-width: 1600px) and (-webkit-min-device-pixel-ratio: 2) and (min-resolution: 192dpi) { } Background Information There are two types of pixels in the display. One is the logical pixels and the other is the physical pixels. Mostly, the physical pixels always stay the same, because it is the same for all the display devices. The logical pixels change based on the resolution of the devices to display higher quality pixels. The device pixel ratio is the ratio between physical pixels and logical pixels. For instance, the MacBook Pro Retina, iPhone 4 and above report a device pixel ratio of 2, because the physical linear resolution is double the logical resolution. The reason why this works only with WebKit based browsers is because of: The vendor prefix -webkit- before the rule. This hasn't been implemented in engines other than WebKit and Blink. Section 13.5: Width vs Viewport When we are using "width" with media queries it is important to set the meta tag correctly. Basic meta tag looks like this and it needs to be put inside the tag. Why this is important? Based on MDN's definition "width" is The width media feature describes the width of the rendering surface of the output device (such as the width of the document window, or the width of the page box on a printer). What does that mean? View-port is the width of the device itself. If your screen resolution says the resolution is 1280 x 720, your view-port width is "1280px". More often many devices allocate different pixel amount to display one pixel. For an example iPhone 6 Plus has 1242 x 2208 resolution. But the actual viewport-width and viewport-height is 414 x 736. That means 3 pixels are used to create 1 pixel. But if you did not set the meta tag correctly it will try to show your webpage with its native resolution which results in a zoomed out view (smaller texts and images). Section 13.6: Using Media Queries to Target Di□erent Screen Sizes Often times, responsive web design involves media queries, which are css blocks that are only executed if a condition is satisfied. This is useful for responsive web design because you can use media queries to specify different css styles for the mobile version of your website versus the desktop version. @media only screen and (min-width: 300px) and (max-width: 767px) { .site-title { font-size: 80%; } /* Styles in this block are only applied if the screen size is atleast 300px wide, but no more than 767px */ } @media only screen and (min-width: 768px) and (max-width: 1023px) { .site-title { font-size: 90%; } /* Styles in this block are only applied if the screen size is atleast 768px wide, but no more than 1023px */ } @media only screen and (min-width: 1024px) { .site-title { font-size: 120%; } /* Styles in this block are only applied if the screen size is over 1024px wide. */ } Section 13.7: Use on link tag This stylesheet is still downloaded but is applied only on devices with screen width larger than 600px. Section 13.8: Media queries and IE8 Media queries are not supported at all in IE8 and below. A Javascript based workaround To add support for IE8, you could use one of several JS solutions. For example, Respond can be added to add media query support for IE8 only with the following code : css Mediaqueries is another library that does the same thing. The code for adding that library to your html would be identical : The alternative If you don't like a JS based solution, you should also consider adding an IE<9 only stylesheet where you adjust your styling specific to ie<9. for that, should add the following html code: Note : Technically it's one more alternative: using css hacks to target IE<9. it has the same impact as an ie<9 only stylesheet, but you don't need a separate stylesheet for that. i do not recommend this option, though, they produce invalid css code (which is one of several reasons why use hacks generally frowned upon today). chapter 14: floats section 14.1: float image within text most basic having wrap around image. below will two paragraphs and image, with second paragraph flowing notice that always content after floated element flows element. html:

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero. Sed cursus ante dapibus diam. Sed nisi. Nulla quis sem at nibh elementum imperdiet. Duis sagittis ipsum. Praesent mauris. Fusce nec tellus sed augue semper porta. Mauris massa. Vestibulum lacinia arcu eget nulla.

 Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Curabitur sodales ligula in libero. Sed dignissim lacinia nunc. Curabitur tortor. Pellentesque nibh. Aenean quam. In scelerisque sem at dolor. Maecenas mattis. Sed convallis tristique sem. Proin ut ligula vel nunc egestas porttitor. Morbi lectus risus, iaculis vel, suscipit quis, luctus non, massa. Fusce ac turpis quis ligula lacinia aliquet.

css: img { float:left; margin-right:1rem; } This will be the output Codepen Link Section 14.2: clear property The clear property is directly related to floats. Property Values: none - Default. Allows floating elements on both sides left - No floating elements allowed on the left side right - No floating elements allowed on the right side both - No floating elements allowed on either the left or the right side initial - Sets this property to its default value. Read about initial inherit - Inherits this property from its parent element. Read about inherit

 Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum

Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum Lorem ipsoum

Section 14.3: Clearfix The clearfix hack is a popular way to contain floats (N. Gallagher aka @necolas) Not to be confused with the clear property, clearfix is a concept (that is also related to floats, thus the possible confusion). To contain floats, you've to add .cf or .clearfix class on the container (the parent) and style this class with a few rules described below. 3 versions with slightly different effects (sources :A new micro clearfix hack by N. Gallagher and clearfix reloaded by T. J. Koblentz): Clearfix (with top margin collapsing of contained floats still occurring) .cf:after { content: ""; display: table; } .cf:after { clear: both; } Clearfix also preventing top margin collapsing of contained floats /** * For modern browsers * 1. The space content is one way to avoid an Opera bug when the * contenteditable attribute is included anywhere else in the document. * Otherwise it causes space to appear at the top and bottom of elements * that are clearfixed. * 2. The use of `table` rather than `block` is only necessary if using * `:before` to contain the top-margins of child elements. */ .cf:before, .cf:after { content: " "; /* 1 */ display: table; /* 2 */ } .cf:after { clear: both; } Clearfix with support of outdated browsers IE6 and IE7 .cf:before, .cf:after { content: " "; display: table; } .cf:after { clear: both; } /** * For IE 6/7 only * Include this rule to trigger hasLayout and contain floats. */ .cf { *zoom: 1; } Codepen showing clearfix effect Other resource: Everything you know about clearfix is wrong (clearfix and BFC - Block Formatting Context while hasLayout relates to outdated browsers IE6 maybe 7) Section 14.4: In-line DIV using float The div is a block-level element, i.e it occupies the whole of the page width and the siblings are place one below the other irrespective of their width.

This is DIV 1

This is DIV 2

The output of the following code will be We can make them in-line by adding a float css property to the div. html:

This is DIV 1

This is DIV 2

css .inner-div1 { width: 50%; margin-right:0px; float:left; background : #337ab7; padding:50px 0px; } .inner-div2 { width: 50%; margin-right:0px; float:left; background : #dd2c00; padding:50px 0px; } p { text-align:center; } Codepen Link Section 14.5: Use of overflow property to clear floats Setting overflow value to hidden,auto or scroll to an element, will clear all the floats within that element. Note: using overflow:scroll will always show the scrollbox Section 14.6: Simple Two Fixed-Width Column Layout A simple two-column layout consists of two fixed-width, floated elements. Note that the sidebar and content area are not the same height in this example. This is one of the tricky parts with multi-column layouts using floats, and requires workarounds to make multiple columns appear to be the same height. html:

Sidebar

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio.

Content

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Curabitur sodales ligula in libero. Sed dignissim lacinia nunc. Curabitur tortor. Pellentesque nibh. Aenean quam. In scelerisque sem at dolor. Maecenas mattis. Sed convallis tristique sem. Proin ut ligula vel nunc egestas porttitor. Morbi lectus risus, iaculis vel, suscipit quis, luctus non, massa. Fusce ac turpis quis ligula lacinia aliquet.

css: .wrapper { width:600px; padding:20px; background-color:pink; /* Floated elements don't use any height. Adding "overflow:hidden;" forces the parent element to expand to contain its floated children. */ overflow:hidden; } .sidebar { width:150px; float:left; background-color:blue; } .content { width:450px; float:right; background-color:yellow; } Section 14.7: Simple Three Fixed-Width Column Layout html:

Left Sidebar

Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Content

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Curabitur sodales ligula in libero. Sed dignissim lacinia nunc. Curabitur tortor. Pellentesque nibh. Aenean quam. In scelerisque sem at dolor. Maecenas mattis. Sed convallis tristique sem. Proin ut ligula vel nunc egestas porttitor. Morbi lectus risus, iaculis vel, suscipit quis, luctus non, massa.

Right Sidebar

Fusce ac turpis quis ligula lacinia aliquet.

css: .wrapper { width:600px; background-color:pink; padding:20px; /* Floated elements don't use any height. Adding "overflow:hidden;" forces the parent element to expand to contain its floated children. */ overflow:hidden; } .left-sidebar { width:150px; background-color:blue; float:left; } .content { width:300px; background-color:yellow; float:left; } .right-sidebar { width:150px; background-color:green; float:right; } Section 14.8: Two-Column Lazy/Greedy Layout This layout uses one floated column to create a two-column layout with no defined widths. In this example the left sidebar is "lazy," in that it only takes up as much space as it needs. Another way to say this is that the left sidebar is "shrink-wrapped." The right content column is "greedy," in that it takes up all the remaining space. html:

Sidebar

 Content

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero. Sed cursus ante dapibus diam. Sed nisi. Nulla quis sem at nibh elementum imperdiet. Duis sagittis ipsum. Praesent mauris. Fusce nec tellus sed augue semper porta. Mauris massa. Vestibulum lacinia arcu eget nulla.

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Curabitur sodales ligula in libero. Sed dignissim lacinia nunc. Curabitur tortor. Pellentesque nibh. Aenean quam. In scelerisque sem at dolor. Maecenas mattis. Sed convallis tristique sem. Proin ut ligula vel nunc egestas porttitor. Morbi lectus risus, iaculis vel, suscipit quis, luctus non, massa. Fusce ac turpis quis ligula lacinia aliquet. Mauris ipsum. Nulla metus metus, ullamcorper vel, tincidunt sed, euismod in, nibh.

css: .sidebar { /* `display:table;` shrink-wraps the column */ display:table; float:left; background-color:blue; } .content { /* `overflow:hidden;` prevents `.content` from flowing under `.sidebar` */ overflow:hidden; background-color:yellow; } Fiddle Chapter 15: Typography Parameter Details font-style italics or oblique font-variant normal or small-caps font-weight normal, bold or numeric from 100 to 900. font-size The font size given in %, px, em, or any other valid css measurement line-height The line height given in %, px, em, or any other valid css measurement font-family This is for defining the family's name. color Any valid css color representation, like red, #00FF00, hsl(240, 100%, 50%) etc. font-stretch Whether or not to use a confenced or expanded face from font. Valid values are normal, ultracondensed, extra-condensed, condensed, semi-condensed, semi-expanded, expanded, extraexpanded or ultra-expanded text-align start, end, left, right, center, justify, match-parent text-decoration none, underline, overline, line-through, initial, inherit; Section 15.1: The Font Shorthand With the syntax: element { font: [font-style] [font-variant] [font-weight] [font-size/line-height] [font-family]; } You can have all your font-related styles in one declaration with the font shorthand. Simply use the font property, and put your values in the correct order. For example, to make all p elements bold with a font size of 20px and using Arial as the font family typically you would code it as follows: p { font-weight: bold; font-size: 20px; font-family: Arial, sans-serif; } However with the font shorthand it can be condensed as follows: p { font: bold 20px Arial, sans-serif; } Note: that since font-style, font-variant, font-weight and line-height are optional, the three of them are skipped in this example. It is important to note that using the shortcut resets the other attributes not given. Another important point is that the two necessary attributes for the font shortcut to work are font-size and fontfamily. If they are not both included the shortcut is ignored. Initial value for each of the properties: font-style: normal; font-variant: normal; font-weight: normal; font-stretch: normal; font-size: medium; line-height: normal; font-family – depends on user agent Section 15.2: Quotes The quotes property is used to customize the opening and closing quotation marks of the " tag. q { quotes: "«" "»"; } Section 15.3: Font Size html:
Hello I am some text.
Hello I am some smaller text.
css: #element-one { font-size: 30px; } #element-two { font-size: 10px; } The text inside #element-one will be 30px in size, while the text in #element-two will be 10px in size. Section 15.4: Text Direction div { direction: ltr; /* Default, text read read from left-to-right */ } .ex { direction: rtl; /* text read from right-to-left */ } .horizontal-tb { writing-mode: horizontal-tb; /* Default, text read from left-to-right and top-to-bottom. */ } .vertical-rtl { writing-mode: vertical-rl; /* text read from right-to-left and top-to-bottom */ } .vertical-ltr { writing-mode: vertical-rl; /* text read from left-to-right and top to bottom */ } The direction property is used to change the horizontal text direction of an element. Syntax: direction: ltr | rtl | initial | inherit; The writing-mode property changes the alignment of text so it can be read from top-to-bottom or from left-to-right, depending on the language. Syntax: direction: horizontal-tb | vertical-rl | vertical-lr; Section 15.5: Font Stacks font-family: 'Segoe UI', Tahoma, sans-serif; The browser will attempt to apply the font face "Segoe UI" to the characters within the elements targeted by the above property. If this font is not available, or the font does not contain a glyph for the required character, the browser will fall back to Tahoma, and, if necessary, any sans-serif font on the user's computer. Note that any font names with more than one word such as "Segoe UI" need to have single or double quotes around them. font-family: Consolas, 'Courier New', monospace; The browser will attempt to apply the font face "Consolas" to the characters within the elements targeted by the above property. If this font is not available, or the font does not contain a glyph for the required character, the browser will fall back to "Courier New," and, if necessary, any monospace font on the user's computer. Section 15.6: Text Overflow The text-overflow property deals with how overflowed content should be signaled to users. In this example, the ellipsis represents clipped text. .text { overflow: hidden; text-overflow: ellipsis; } Unfortunately, text-overflow: ellipsis only works on a single line of text. There is no way to support ellipsis on the last line in standard css, but it can be achieved with non-standard webkit-only implementation of flexboxes. .giveMeEllipsis { overflow: hidden; text-overflow: ellipsis; display: -webkit-box; -webkit-box-orient: vertical; -webkit-line-clamp: N; /* number of lines to show */ line-height: X; /* fallback */ max-height: X*N; /* fallback */ } Example (open in Chrome or Safari): http://jsfiddle.net/csYjC/1131/ Resources: https://www.w3.org/TR/2012/WD-css3-ui-20120117/#text-overflow0 Section 15.7: Text Shadow To add shadows to text, use the text-shadow property. The syntax is as follows: text-shadow: horizontal-offset vertical-offset blur color; Shadow without blur radius h1 { text-shadow: 2px 2px #0000FF; } This creates a blue shadow effect around a heading Shadow with blur radius To add a blur effect, add an option blur radius argument h1 { text-shadow: 2px 2px 10px #0000FF; } Multiple Shadows To give an element multiple shadows, separate them with commas h1 { text-shadow: 0 0 3px #FF0000, 0 0 5px #0000FF; } Section 15.8: Text Transform The text-transform property allows you to change the capitalization of text. Valid values are: uppercase, capitalize, lowercase, initial, inherit, and none css .example1 { text-transform: uppercase; } .example2 { text-transform: capitalize; } .example3 { text-transform: lowercase; } html

all letters in uppercase

all letters in capitalize

all letters in lowercase

Section 15.9: Letter Spacing h2 { /* adds a 1px space horizontally between each letter; also known as tracking */ letter-spacing: 1px; } The letter-spacing property is used to specify the space between the characters in a text. ! letter-spacing also supports negative values: p { letter-spacing: -1px; } Resources: https://developer.mozilla.org/en-US/docs/Web/css/letter-spacing Section 15.10: Text Indent p { text-indent: 50px; } The text-indent property specifies how much horizontal space text should be moved before the beginning of the first line of the text content of an element. Resources: Indenting only the first line of text in a paragraph? https://www.w3.org/TR/css21/text.html#propdef-text-indent https://developer.mozilla.org/en-US/docs/Web/css/text-indent Section 15.11: Text Decoration The text-decoration property is used to set or remove decorations from text. h1 { text-decoration: none; } h2 { text-decoration: overline; } h3 { text-decoration: line-through; } h4 { text-decoration: underline; } text-decoration can be used in combination with text-decoration-style and text-decoration-color as a shorthand property: .title { text-decoration: underline dotted blue; } This is a shorthand version of .title { text-decoration-style: dotted; text-decoration-line: underline; text-decoration-color: blue; } It should be noted that the following properties are only supported in Firefox text-decoration-color text-decoration-line text-decoration-style text-decoration-skip Section 15.12: Word Spacing The word-spacing property specifies the spacing behavior between tags and words. Possible values a positive or negative length (using em px vh cm etc.) or percentage (using %) the keyword normal uses the font's default word spacing the keyword inherit takes the value from the parent element css .normal { word-spacing: normal; } .narrow { word-spacing: -3px; } .extensive { word-spacing: 10px; } html

This is an example, showing the effect of "word-spacing".
This is an example, showing the effect of "word-spacing".
This is an example, showing the effect of "word-spacing".

Online-Demo Try it yourself Further reading: word-spacing – MDN word-spacing – w3.org Section 15.13: Font Variant Attributes: normal Default attribute of fonts. small-caps Sets every letter to uppercase, but makes the lowercase letters(from original text) smaller in size than the letters that originally uppercase. css: .smallcaps{ font-variant: small-caps; } html:

Documentation about css Fonts
aNd ExAmpLe

Output: Note: The font-variant property is a shorthand for the properties: font-variant-caps, font-variant-numeric, fontvariant-alternates, font-variant-ligatures, and font-variant-east-asian. Chapter 16: Flexible Box Layout (Flexbox) The Flexible Box module, or just 'flexbox' for short, is a box model designed for user interfaces, and it allows users to align and distribute space among items in a container such that elements behave predictably when the page layout must accommodate different, unknown screen sizes. A flex container expands items to fill available space and shrinks them to prevent overflow. Section 16.1: Dynamic Vertical and Horizontal Centering (alignitems, justify-content) Simple Example (centering a single element) html
...
css .aligner { display: flex; align-items: center; justify-content: center; } .aligner-item { max-width: 50%; /*for demo. Use actual width instead.*/ } Here is a demo. Reasoning Property Value Description align-items center This centers the elements along the axis other than the one specified by flex-direction, i.e., vertical centering for a horizontal flexbox and horizontal centering for a vertical flexbox. justify-content center This centers the elements along the axis specified by flex-direction. I.e., for a horizontal (flex-direction: row) flexbox, this centers horizontally, and for a vertical flexbox (flex-direction: column) flexbox, this centers vertically) Individual Property Examples All of the below styles are applied onto this simple layout:
where #container is the flex-box. Example: justify-content: center on a horizontal flexbox css: div#container { display: flex; flex-direction: row; justify-content: center; } Outcome: Here is a demo. Example: justify-content: center on a vertical flexbox css: div#container { display: flex; flex-direction: column; justify-content: center; } Outcome: Here is a demo. Example: align-content: center on a horizontal flexbox css: div#container { display: flex; flex-direction: row; align-items: center; } Outcome: Here is a demo. Example: align-content: center on a vertical flexbox css: div#container { display: flex; flex-direction: column; align-items: center; } Outcome: Here is a demo. Example: Combination for centering both on horizontal flexbox div#container { display: flex; flex-direction: row; justify-content: center; align-items: center; } Outcome: Here is a demo. Example: Combination for centering both on vertical flexbox div#container { display: flex; flex-direction: column; justify-content: center; align-items: center; } Outcome: Here is a demo. Section 16.2: Sticky Variable-Height Footer This code creates a sticky footer. When the content doesn't reach the end of the viewport, the footer sticks to the bottom of the viewport. When the content extends past the bottom of the viewport, the footer is also pushed out of the viewport. View Result html:

Header

Content

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero. Sed cursus ante dapibus diam. Sed nisi. Nulla quis sem at nibh elementum imperdiet. Duis sagittis ipsum. Praesent mauris. Fusce nec tellus sed augue semper porta. Mauris massa. Vestibulum lacinia arcu eget nulla. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Curabitur sodales ligula in libero.

Footer

css: html, body { height: 100%; } body { display: flex; flex-direction: column; } .content { /* Include `0 auto` for best browser compatibility. */ flex: 1 0 auto; } .header, .footer { background-color: grey; color: white; flex: none; } Section 16.3: Optimally fit elements to their container One of the nicest features of flexbox is to allow optimally fitting containers to their parent element. Live demo. html:
1
2
3
4
5
css: .flex-container { background-color: #000; height: 100%; display:flex; flex-direction: row; flex-wrap: wrap; justify-content: flex-start; align-content: stretch; align-items: stretch; } .flex-item { background-color: #ccf; margin: 0.1em; flex-grow: 1; flex-shrink: 0; flex-basis: 200px; /* or % could be used to ensure a specific layout */ } Outcome: Columns adapt as screen is resized. Section 16.4: Holy Grail Layout using Flexbox Holy Grail layout is a layout with a fixed height header and footer, and a center with 3 columns. The 3 columns include a fixed width sidenav, a fluid center, and a column for other content like ads (the fluid center appears first in the markup). css Flexbox can be used to achieve this with a very simple markup: html Markup:
Header
Content

Nav
Ads
Footer
css: body { margin: 0; padding: 0; } .container { display: flex; flex-direction: column; height: 100vh; } .header { flex: 0 0 50px; } .content-body { flex: 1 1 auto; display: flex; flex-direction: row; } .content-body .content { flex: 1 1 auto; overflow: auto; } .content-body .sidenav { order: -1; flex: 0 0 100px; overflow: auto; } .content-body .ads { flex: 0 0 100px; overflow: auto; } .footer { flex: 0 0 50px; } Demo Section 16.5: Perfectly aligned buttons inside cards with flexbox It's a regular pattern in design these days to vertically align call to actions inside its containing cards like this: This can be achieved using a special trick with flexbox html

Lorem ipsum Magna proident ex anim dolor ullamco pariatur reprehenderit culpa esse enim mollit labore dolore voluptate ullamco et ut sed qui minim.

Action

Lorem ipsum Magna proident ex anim dolor ullamco pariatur reprehenderit culpa esse enim mollit labore dolore voluptate ullamco et ut sed qui minim.

Lorem ipsum Magna proident ex anim dolor ullamco pariatur reprehenderit culpa esse enim mollit labore dolore voluptate ullamco et ut sed qui minim.

Lorem ipsum Magna proident ex anim dolor ullamco pariatur reprehenderit culpa esse enim mollit labore dolore voluptate ullamco et ut sed qui minim.

Lorem ipsum Magna proident ex anim dolor ullamco pariatur reprehenderit culpa esse enim mollit labore dolore voluptate ullamco et ut sed qui minim.

Action

First of all, we use css to apply display: flex; to the container. This will create 2 columns equal in height with the content flowing naturally inside it css .cards { display: flex; } .card { border: 1px solid #ccc; margin: 10px 10px; padding: 0 20px; } button { height: 40px; background: #fff; padding: 0 40px; border: 1px solid #000; } p:last-child { text-align: center; } The layout will change and become like this: In order to move the buttons to the bottom of the block, we need to apply display: flex; to the card itself with the direction set to column. After that, we should select the last element inside the card and set the margin-top to auto. This will push the last paragraph to the bottom of the card and achieve the required result. Final css: .cards { display: flex; } .card { border: 1px solid #ccc; margin: 10px 10px; padding: 0 20px; display: flex; flex-direction: column; } button { height: 40px; background: #fff; padding: 0 40px; border: 1px solid #000; } p:last-child { text-align: center; margin-top: auto; } Section 16.6: Same height on nested containers This code makes sure that all nested containers are always the same height. This is done by assuring that all nested elements are the same height as the containing parent div. See working example: https://jsfiddle.net/3wwh7ewp/ This effect is achieved due to the property align-items being set to stretch by default. html

Fewer
lines
css .container { display: flex; align-items: stretch; // Default value } Note: Does not work on IE versions under 10 Chapter 17: Cascading and Specificity Section 17.1: Calculating Selector Specificity Each individual css Selector has its own specificity value. Every selector in a sequence increases the sequence's overall specificity. Selectors fall into one of three different specificity groups: A, B and c. When multiple selector sequences select a given element, the browser uses the styles applied by the sequence with the highest overall specificity. Group Comprised of Examples A id selectors #foo B class selectors attribute selectors pseudo-classes .bar [title], [colspan="2"] :hover, :nth-child(2) c type selectors pseudo-elements div, li ::before, ::first-letter Group A is the most specific, followed by Group B, then finally Group c. The universal selector (*) and combinators (like > and ~) have no specificity. Example 1: Specificity of various selector sequences #foo #baz {} /* a=2, b=0, c=0 */ #foo.bar {} /* a=1, b=1, c=0 */ #foo {} /* a=1, b=0, c=0 */ .bar:hover {} /* a=0, b=2, c=0 */ div.bar {} /* a=0, b=1, c=1 */ :hover {} /* a=0, b=1, c=0 */ [title] {} /* a=0, b=1, c=0 */ .bar {} /* a=0, b=1, c=0 */ div ul + li {} /* a=0, b=0, c=3 */ p::after {} /* a=0, b=0, c=2 */ *::before {} /* a=0, b=0, c=1 */ ::before {} /* a=0, b=0, c=1 */ div {} /* a=0, b=0, c=1 */ * {} /* a=0, b=0, c=0 */ Example 2: How specificity is used by the browser Imagine the following css implementation: #foo { color: blue; } .bar { color: red; background: black; } Here we have an ID selector which declares color as blue, and a class selector which declares color as red and background as black. An element with an ID of #foo and a class of .bar will be selected by both declarations. ID selectors have a Group A specificity and class selectors have a Group B specificity. An ID selector outweighs any number of class selectors. Because of this, color:blue; from the #foo selector and the background:black; from the .bar selector will be applied to the element. The higher specificity of the ID selector will cause the browser to ignore the .bar selector's color declaration. Now imagine a different css implementation: .bar { color: red; background: black; } .baz { background: white; } Here we have two class selectors; one of which declares color as red and background as black, and the other declares background as white. An element with both the .bar and .baz classes will be affected by both of these declarations, however the problem we have now is that both .bar and .baz have an identical Group B specificity. The cascading nature of css resolves this for us: as .baz is defined after .bar, our element ends up with the red color from .bar but the white background from .baz. Example 3: How to manipulate specificity The last snippet from Example 2 above can be manipulated to ensure our .bar class selector's color declaration is used instead of that of the .baz class selector. .bar {} /* a=0, b=1, c=0 */ .baz {} /* a=0, b=1, c=0 */ The most common way to achieve this would be to find out what other selectors can be applied to the .bar selector sequence. For example, if the .bar class was only ever applied to span elements, we could modify the .bar selector to span.bar. This would give it a new Group C specificity, which would override the .baz selector's lack thereof: span.bar {} /* a=0, b=1, c=1 */ .baz {} /* a=0, b=1, c=0 */ However it may not always possible to find another common selector which is shared between any element which uses the .bar class. Because of this, css allows us to duplicate selectors to increase specificity. Instead of just .bar, we can use .bar.bar instead (See The grammar of Selectors, W3C Recommendation). This still selects any element with a class of .bar, but now has double the Group B specificity: .bar.bar {} /* a=0, b=2, c=0 */ .baz {} /* a=0, b=1, c=0 */ !important and inline style declarations The !important flag on a style declaration and styles declared by the html style attribute are considered to have a greater specificity than any selector. If these exist, the style declaration they affect will overrule other declarations regardless of their specificity. That is, unless you have more than one declaration that contains an !important flag for the same property that apply to the same element. Then, normal specificity rules will apply to those properties in reference to each other. Because they completely override specificity, the use of !important is frowned upon in most use cases. One should use it as little as possible. To keep css code efficient and maintainable in the long run, it's almost always better to increase the specificity of the surrounding selector than to use !important. One of those rare exceptions where !important is not frowned upon, is when implementing generic helper classes like a .hidden or .background-yellow class that are supposed to always override one or more properties wherever they are encountered. And even then, you need to know what you're doing. The last thing you want,

when writing maintainable css, is to have !important flags throughout your css. A final note A common misconception about css specificity is that the Group A, B and c values should be combined with each other (a=1, b=5, c=1 => 151). This is not the case. If this were the case, having 20 of a Group B or c selector would be enough to override a single Group A or B selector respectively. The three groups should be regarded as individual levels of specificity. Specificity cannot be represented by a single value. When creating your css style sheet, you should maintain the lowest specificity as possible. If you need to make the specificity a little higher to overwrite another method, make it higher but as low as possible to make it higher. You shouldn't need to have a selector like this: body.page header.container nav div#main-nav li a {} This makes future changes harder and pollutes that css page. You can calculate the specificity of your selector here Section 17.2: The !important declaration The !important declaration is used to override the usual specificity in a style sheet by giving a higher priority to a rule. Its usage is: property : value !important; #mydiv { font-weight: bold !important; /* This property won't be overridden by the rule below */ } #outerdiv #mydiv { font-weight: normal; /* #mydiv font-weight won't be set to normal even if it has a higher specificity because of the !important declaration above */ } Avoiding the usage of !important is strongly recommended (unless absolutely necessary), because it will disturb the natural flow of css rules which can bring uncertainty in your style sheet. Also it is important to note that when multiple !important declarations are applied to the same rule on a certain element, the one with the higher specificity will be the ona applied. Here are some examples where using !important declaration can be justified: If your rules shouldn't be overridden by any inline style of the element which is written inside style attribute of the html element. To give the user more control over the web accessibility, like increasing or decreasing size of the font-size, by overriding the author style using !important. For testing and debugging using inspect element. See also: W3C - 6 Assigning property values, Cascading, and Inheritance -- 6.4.2 !important rules Section 17.3: Cascading Cascading and specificity are used together to determine the final value of a css styling property. They also define the mechanisms for resolving conflicts in css rule sets. css Loading order Styles are read from the following sources, in this order: 1. User Agent stylesheet (The styles supplied by the browser vendor) 2. User stylesheet (The additional styling a user has set on his/her browser) 3. Author stylesheet (Author here means the creator of the webpage/website) Maybe one or more .css files In the OUTPUT Chapter 21: Pseudo-Elements pseudo-element Description ::after Insert content after the content of an element ::before Insert content before the content of an element ::first-letter Selects the first letter of each element ::first-line Selects the first line of each element ::selection Matches the portion of an element that is selected by a user ::backdrop Used to create a backdrop that hides the underlying document for an element in the top layer's stack ::placeholder Allows you to style the placeholder text of a form element (Experimental) ::marker For applying list-style attributes on a given element (Experimental) ::spelling-error Represents a text segment which the browser has flagged as incorrectly spelled (Experimental) ::grammar-error Represents a text segment which the browser has flagged as grammatically incorrect (Experimental) Pseudo-elements, just like pseudo-classes, are added to a css selectors but instead of describing a special state, they allow you to scope and style certain parts of an html element. For example, the ::first-letter pseudo-element targets only the first letter of a block element specified by the selector. Section 21.1: Pseudo-Elements Pseudo-elements are added to selectors but instead of describing a special state, they allow you to style certain parts of a document. The content attribute is required for pseudo-elements to render; however, the attribute can have an empty value (e.g. content: ""). div::after { content: 'after'; color: red; border: 1px solid red; } div { color: black; border: 1px solid black; padding: 1px; } div::before { content: 'before'; color: green; border: 1px solid green; } Section 21.2: Pseudo-Elements in Lists Pseudo-elements are often used to change the look of lists (mostly for unordered lists, ul). The first step is to remove the default list bullets: ul { list-style-type: none; } Then you add the custom styling. In this example, we will create gradient boxes for bullets. li:before { content: ""; display: inline-block; margin-right: 10px; height: 10px; width: 10px; background: linear-gradient(red, blue); } html

Test I Test II

Result Chapter 22: Positioning Parameter Details static Default value. Elements render in order, as they appear in the document flow. The top, right, bottom, left and z-index properties do not apply. relative The element is positioned relative to its normal position, so left:20px adds 20 pixels to the element's LEFT position fixed The element is positioned relative to the browser window absolute The element is positioned relative to its first positioned (not static) ancestor element initial Sets this property to its default value. inherit Inherits this property from its parent element. sticky Experimental feature. It behaves like position: static within its parent until a given offset threshold is reached, then it acts as position: fixed. unset Combination of initial and inherit. More info here. Section 22.1: Overlapping Elements with z-index To change the default stack order positioned elements (position property set to relative, absolute or fixed), use the z-index property. The higher the z-index, the higher up in the stacking context (on the z-axis) it is placed. Example In the example below, a z-index value of 3 puts green on top, a z-index of 2 puts red just under it, and a z-index of 1 puts blue under that. html
css div { position: absolute; height: 200px; width: 200px; } div#div1 { z-index: 1; left: 0px; top: 0px; background-color: blue; } div#div2 { z-index: 3; left: 100px; top: 100px; background-color: green; } div#div3 { z-index: 2; left: 50px; top: 150px; background-color: red; } This creates the following effect: See a working example at JSFiddle. Syntax z-index: [number] | auto; Parameter Details number An integer value. A higher number is higher on the z-index stack. 0 is the default value. Negative values are allowed. auto Gives the element the same stacking context as its parent. (Default) Remarks All elements are laid out in a 3D axis in css, including a depth axis, measured by the z-index property. z-index only works on positioned elements: (see: Why does z-index need a defined position to work?). The only value where it is ignored is the default value, static. Read about the z-index property and Stacking Contexts in the css Specification on layered presentation and at the Mozilla Developer Network. Section 22.2: Absolute Position When absolute positioning is used the box of the desired element is taken out of the Normal Flow and it no longer affects the position of the other elements on the page. Offset properties: 1. top 2. left 3. right 4. bottom specify the element should appear in relation to its next non-static containing element. .abspos{ position:absolute; top:0px; left:500px; } This code will move the box containing element with attribute class="abspos" down 0px and right 500px relative to its containing element. Section 22.3: Fixed position Defining position as fixed we can remove an element from the document flow and set its position relatively to the browser window. One obvious use is when we want something to be visible when we scroll to the bottom of a long page. #stickyDiv { position:fixed; top:10px; left:10px; } Section 22.4: Relative Position Relative positioning moves the element in relation to where it would have been in normal flow .Offset properties: 1. top 2. left 3. right 4. bottom are used to indicate how far to move the element from where it would have been in normal flow. .relpos{ position:relative; top:20px; left:30px; } This code will move the box containing element with attribute class="relpos" 20px down and 30px to the right from where it would have been in normal flow. Section 22.5: Static positioning The default position of an element is static. To quote MDN: This keyword lets the element use the normal behavior, that is it is laid out in its current position in the flow. The top, right, bottom, left and z-index properties do not apply. .element{ position:static; } Chapter 23: Layout Control Value Effect none Hide the element and prevent it from occupying space. block Block element, occupy 100% of the available width, break after element. inline Inline element, occupy no width, no break after element. inline-block Taking special properties from both inline and block elements, no break, but can have width. inline-flex Displays an element as an inline-level flex container. inline-table The element is displayed as an inline-level table. grid Behaves like a block element and lays out its content according to the grid model. flex Behaves like a block element and lays out its content according to the flexbox model. inherit Inherit the value from the parent element. initial Reset the value to the default value taken from behaviors described in the html specifications or from the browser/user default stylesheet. table Behaves like the html table element. table-cell

Let the element behave like a "element table-row Let the element behave like a element list-item Let the element behave like a

element. Section 23.1: The display property The display css property is fundamental for controlling the layout and flow of an html document. Most elements have a default display value of either block or inline (though some elements have other default values). Inline An inline element occupies only as much width as necessary. It stacks horizontally with other elements of the same type and may not contain other non-inline elements. This is some **bolded** text! As demonstrated above, two inline elements, and **, are in-line (hence the name) and do not break the flow of the text. Block A block element occupies the maximum available width of its' parent element. It starts with a new line and, in contrast to inline elements, it does not restrict the type of elements it may contain.**
Hello world!
This is an example!
The div element is block-level by default, and as shown above, the two block elements are vertically stacked and, unlike the inline elements, the flow of the text breaks. Inline Block The inline-block value gives us the best of both worlds: it blends the element in with the flow of the text while allowing us to use padding, margin, height and similar properties which have no visible effect on inline elements. Elements with this display value act as if they were regular text and as a result are affected by rules controlling the flow of text such as text-align. By default they are also shrunk to the the smallest size possible to accommodate their content.

First Element	Second Element	Third Element
First Element	Second Element	Third Element
First Element	Second Element	Third Element

none An element that is given the none value to its display property will not be displayed at all. For example let's create a div-element that has an id of myDiv:
This can now be marked as not being displayed by the following css rule: #myDiv { display: none; } When an element has been set to be display:none; the browser ignores every other layout property for that specific element (both position and float). No box will be rendered for that element and its existence in html does not affect the position of following elements. Note that this is different from setting the visibility property to hidden. Setting visibility: hidden; for an element would not display the element on the page but the element would still take up the space in the rendering process as if it would be visible. This will therefore affect how following elements are displayed on the page. The none value for the display property is commonly used along with JavaScript to show or hide elements at will, eliminating the need to actually delete and re-create them. Section 23.2: To get old table structure using div This is the normal html table structure

element table-column Let the element behave like a

I'm a table

You can do same implementation like this
I behave like a table now
Chapter 24: Grid Grid layout is a new and powerful css layout system that allows to divide a web page content into rows and columns in an easy way. Section 24.1: Basic Example Property Possible Values display grid / inline-grid The css Grid is defined as a display property. It applies to a parent element and its immediate children only. Consider the following markup:
item1
item2
item3
item4
The easiest way to define the markup structure above as a grid is to simply set its display property to grid: .container { display: grid; } However, doing this will invariably cause all the child elements to collapse on top of one another. This is because the children do not currently know how to position themselves within the grid. But we can explicitly tell them. First we need to tell the grid element .container how many rows and columns will make up its structure and we can do this using the grid-columns and grid-rows properties (note the pluralisation): .container { display: grid; grid-columns: 50px 50px 50px; grid-rows: 50px 50px; } However, that still doesn't help us much because we need to give an order to each child element. We can do this by specifying the grid-row and grid-column values which will tell it where it sits in the grid: .container .item1 { grid-column: 1; grid-row: 1; } .container .item2 { grid-column: 2; grid-row: 1; } .container .item3 { grid-column: 1; grid-row: 2; } .container .item4 { grid-column: 2; grid-row: 2; } By giving each item a column and row value it identifies the items order within the container. View a working example on JSFiddle. You'll need to view this in IE10, IE11 or Edge for it to work as these are currently the only browsers supporting Grid Layout (with vendor prefix -ms-) or enable a flag in Chrome, Opera and Firefox according to caniuse in order to test with them. Chapter 25: Tables Section 25.1: table-layout The table-layout property changes the algorithm that is used for the layout of a table. Below an example of two tables both set to width: 150px: The table on the left has table-layout: auto while the one on the right has table-layout: fixed. The former is wider than the specified width (210px instead of 150px) but the contents fit. The latter takes the defined width of 150px, regardless if the contents overflow or not. Value Description auto This is the default value. It defines the layout of the table to be determined by the contents of its' cells. fixed This value sets the table layout to be determined by the width property provided to the table. If the content of a cell exceeds this width, the cell will not resize but instead, let the content overflow. Section 25.2: empty-cells The empty-cells property determines if cells with no content should be displayed or not. This has no effect unless border-collapse is set to separate. Below an example with two tables with different values set to the empty-cells property: The table on the left has empty-cells: show while the one on the right has empty-cells: hide. The former does display the empty cells whereas the latter does not. Value Description show This

is the default value. It shows cells even if they are empty. hide This value hides a cell altogether if there are no contents in the cell. More Information: https://www.w3.org/TR/css21/tables.html#empty-cells https://developer.mozilla.org/en-US/docs/Web/css/empty-cells http://codepen.io/SitePoint/pen/yfhtq https://css-tricks.com/almanac/properties/e/empty-cells/ Section 25.3: border-collapse The border-collapse property determines if a tables' borders should be separated or merged. Below an example of two tables with different values to the border-collapse property: The table on the left has border-collapse: separate while the one on the right has border-collapse: collapse. Value Description separate This is the default value. It makes the borders of the table separate from each other. collapse This value sets the borders of the table to merge together, rather than being distinct. Section 25.4: border-spacing The border-spacing property determines the spacing between cells. This has no effect unless border-collapse is set to separate. Below an example of two tables with different values to the border-spacing property: The table on the left has border-spacing: 2px (default) while the one on the right has border-spacing: 8px. Value Description This is the default behavior, though the exact value can vary between browsers. This syntax allows specifying separate horizontal and vertical values respectively. Section 25.5: caption-side The caption-side property determines the vertical positioning of the element within a table. This has no effect if such element does not exist. Below an example with two tables with different values set to the caption-side property: The table on the left has caption-side: top while the one on the right has caption-side: bottom. Value Description top This is the default value. It places the caption above the table. bottom This value places the caption below the table. Chapter 26: Transitions Parameter Details transition-property The specific css property whose value change needs to be transitioned (or) all, if all the transitionable properties need to be transitioned. transition-duration The duration (or period) in seconds (s) or milliseconds (ms) over which the transition must take place. transition-timing-function A function that describes how the intermediate values during the transition are calculated. Commonly used values are ease, ease-in, ease-out, ease-in-out, linear, cubic-bezier(), steps(). More information about the various timing functions can be found in the W3C specs. transition-delay The amount of time that must have elapsed before the transition can start. Can be specified in seconds (s) or milliseconds (ms) Section 26.1: Transition shorthand css div{ width: 150px; height:150px; background-color: red; transition: background-color 1s; } div:hover{ background-color: green; } html

This example will change the background color when the div is hovered the background-color change will last 1 second. Section 26.2: cubic-bezier The cubic-bezier function is a transition timing function which is often used for custom and smooth transitions. transition-timing-function: cubic-bezier(0.1, 0.7, 1.0, 0.1); The function takes four parameters: cubic-bezier(P1_x, P1_y, P2_x, P2_y) These parameters will be mapped to points which are part of a Bézier curve: For css Bézier Curves, P0 and P3 are always in the same spot. P0 is at (0,0) and P3 is at (1,1), which menas that the parameters passed to the cubic-bezier function can only be between 0 and 1. If you pass parameters which aren't in this interval the function will default to a linear transition. Since cubic-bezier is the most flexible transition in css, you can translate all other transition timing function to cubic-bezier functions: linear: cubic-bezier(0,0,1,1) ease-in: cubic-bezier(0.42, 0.0, 1.0, 1.0) ease-out: cubic-bezier(0.0, 0.0, 0.58, 1.0) ease-in-out: cubic-bezier(0.42, 0.0, 0.58, 1.0) Section 26.3: Transition (longhand) css div { height: 100px; width: 100px; border: 1px solid; transition-property: height, width; transition-duration: 1s, 500ms; transition-timing-function: linear; transition-delay: 0s, 1s; } div:hover { height: 200px; width: 200px; } html

transition-property: Specifies the css properties the transition effect is for. In this case, the div will expand both horizontally and vertically when hovered. transition-duration: Specifies the length of time a transition takes to complete. In the above example, the height and width transitions will take 1 second and 500 milliseconds respectively. transition-timing-function: Specifies the speed curve of the transition effect. A linear value indicates the transition will have the same speed from start to finish. transition-delay: Specifies the amount of time needed to wait before the transition effect starts. In this case, the height will start transitioning immediately, whereas the width will wait 1 second. Chapter 27: Animations Transition Parameter Details property Either the css property to transition on, or all, which specifies all transition-able properties. duration Transition time, either in seconds or milliseconds. timing-function Specifies a function to define how intermediate values for properties are computed. Common values are ease, linear, and step-end. Check out the easing function cheatsheet for more. delay Amount of time, in seconds or milliseconds, to wait before playing the animation. @keyframes [from | to |] You can either specify a set time with a percentage value, or two percentage values, ie 10%, 20%, for a period of time where the keyframe's set attributes are set. block Any amount of css attributes for the keyframe. Section 27.1: Animations with keyframes For multi-stage css animations, you can create css @keyframes. Keyframes allow you to define multiple animation points, called a keyframe, to define more complex animations. Basic Example In this example, we'll make a basic background animation that cycles between all colors. @keyframes rainbow-background { 0% { background-color: #ff0000; } 8.333% { background-color: #ff8000; } 16.667% { background-color: #ffff00; } 25.000% { background-color: #80ff00; } 33.333% { background-color: #00ff00; } 41.667% { background-color: #00ff80; } 50.000% { background-color: #00ffff; } 58.333% { background-color: #0080ff; } 66.667% { background-color: #0000ff; } 75.000% { background-color: #8000ff; } 83.333% { background-color: #ff00ff; } 91.667% { background-color: #ff0080; } 100.00% { background-color: #ff0000; } } .RainbowBackground { animation: rainbow-background 5s infinite; } View Result There's a few different things to note here. First, the actual @keyframes syntax. @keyframes rainbow-background{ This sets the name of the animation to rainbow-background. 0% { background-color: #ff0000; } This is the definition for a keyframe within the animation. The first part, the 0% in the case, defines where the keyframe is during the animation. The 0% implies it is 0% of the total animation time from the beginning. The animation will automatically transition between keyframes. So, by setting the next background color at 8.333%, the animation will smoothly take 8.333% of the time to transition between those keyframes. .RainbowBackground { animation: rainbow-background 5s infinite; } This code attaches our animation to all elements which have the .RainbowBackground class. The actual animation property takes the following arguments. animation-name: The name of our animation. In this case, rainbow-background animation-duration: How long the animation will take, in this case 5 seconds. animation-iteration-count (Optional): The number of times the animation will loop. In this case, the animation will go on indefinitely. By default, the animation will play once. animation-delay (Optional): Specifies how long to wait before the animation starts. It defaults to 0 seconds, and can take negative values. For example, -2s would start the animation 2 seconds into its loop. animation-timing-function (Optional): Specifies the speed curve of the animation. It defaults to ease, where the animation starts slow, gets faster and ends slow. In this particular example, both the 0% and 100% keyframes specify { background-color: #ff0000; }. Wherever two or more keyframes share a state, one may specify them in a single statement. In this case, the two 0% and 100% lines could be replaced with this single line: 0%, 100% { background-color: #ff0000; } Cross-browser compatibility For older WebKit-based browsers, you'll need to use the vendor prefix on both the @keyframes declaration and the animation property, like so: @-webkit-keyframes{} -webkit-animation: ... Section 27.2: Animations with the transition property Useful for simple animations, the css transition property allows number-based css properties to animate between states. Example .Example{ height: 100px; background: #fff; } .Example:hover{ height: 120px; background: #ff0000; } View Result By default, hovering over an element with the .Example class would immediately cause the element's height to jump to 120px and its background color to red (#ff0000). By adding the transition property, we can cause these changes to occur over time: .Example{ ... transition: all 400ms ease; } View Result The all value applies the transition to all compatible (numbers-based) properties. Any compatible property name (such as height or top) can be substituted for this keyword. 400ms specifies the amount of time the transition takes. In this case, the element's change in height will take 400 milliseconds to complete. Finally, the value ease is the animation function, which determines how the animation is played. ease means start slow, speed up, then end slow again. Other values are linear, ease-out, and ease-in. Cross-Browser Compatibility The transition property is generally well-supported across all major browsers, excepting IE 9. For earlier versions of Firefox and Webkit-based browsers, use vendor prefixes like so: .Example{ transition: all 400ms ease; -moz-transition: all 400ms ease; -webkit-transition: all 400ms ease; } Note: The transition property can animate changes between any two numerical values, regardless of unit. It can also transition between units, such as 100px to 50vh. However, it cannot transition between a number and a default or automatic value, such as transitioning an element's height from 100px to auto. Section 27.3: Syntax Examples Our first syntax example shows the animation shorthand property using all of the available properties/parameters: animation: 3s ease-in 1s 2 reverse both paused slidein; /* duration | timing-function | delay | iteration-count | direction | fill-mode |

playstate | name */ Our second example is a little more simple, and shows that some properties can be omitted: animation: 3s linear 1s slidein; /* duration | timing-function | delay | name */ Our third example shows the most minimal declaration. Note that the animation-name and animation-duration must be declared: animation: 3s slidein; /* duration | name */ It's also worth mentioning that when using the animation shorthand the order of the properties makes a difference. Obviously the browser may confuse your duration with your delay. If brevity isn't your thing, you can also skip the shorthand property and write out each property individually: animation-duration: 3s; animation-timing-function: ease-in; animation-delay: 1s; animation-iteration-count: 2; animation-direction: reverse; animation-fill-mode: both; animation-play-state: paused; animation-name: slidein; Section 27.4: Increasing Animation Performance Using the `will-change` Attribute When creating animations and other GPU-heavy actions, it's important to understand the will-change attribute. Both css keyframes and the transition property use GPU acceleration. Performance is increased by offloading calculations to the device's GPU. This is done by creating paint layers (parts of the page that are individually rendered) that are offloaded to the GPU to be calculated. The will-change property tells the browser what will animate, allowing the browser to create smaller paint areas, thus increasing performance. The will-change property accepts a comma-separated list of properties to be animated. For example, if you plan on transforming an object and changing its opacity, you would specify: .Example{ ... will-change: transform, opacity; } Note: Use will-change sparingly. Setting will-change for every element on a page can cause performance problems, as the browser may attempt to create paint layers for every element, significantly increasing the amount of processing done by the GPU. Chapter 28: 2D Transforms Function/Parameter Details rotate(x) Defines a transformation that moves the element around a fixed point on the Z axis translate(x,y) Moves the position of the element on the X and Y axis translateX(x) Moves the position of the element on the X axis translateY(y) Moves the position of the element on the Y axis scale(x,y) Modifies the size of the element on the X and Y axis scaleX(x) Modifies the size of the element on the X axis scaleY(y) Modifies the size of the element on the Y axis skew(x,y) Shear mapping, or transvection, distorting each point of an element by a certain angle in each direction skewX(x) Horizontal shear mapping distorting each point of an element by a certain angle in the horizontal direction skewY(y) Vertical shear mapping distorting each point of an element by a certain angle in the vertical direction matrix() Defines a 2D transformation in the form of a transformation matrix. angle The angle by which the element should be rotated or skewed (depending on the function with which it is used). Angle can be provided in degrees (deg), gradians (grad), radians (rad) or turns (turn). In skew() function, the second angle is optional. If not provided, there will be no (0) skew in Y-axis. length-or-percentage The distance expressed as a length or a percentage by which the element should be translated. In translate() function, the second length-or-percentage is optional. If not provided, then there would be no (0) translation in Y-axis. scale-factor A number which defines how many times the element should be scaled in the specified axis. In scale() function, the second scale-factor is optional. If not provided, the first scale-factor will be applied for Y-axis also. Section 28.1: Rotate html
css .rotate { width: 100px; height: 100px; background: teal; transform: rotate(45deg); } This example will rotate the div by 45 degrees clockwise. The center of rotation is in the center of the div, 50% from left and 50% from top. You can change the center of rotation by setting the transform-origin property. transform-origin: 100% 50%; The above example will set the center of rotation to the middle of the right side end. Section 28.2: Scale html
css .scale { width: 100px; height: 100px; background: teal; transform: scale(0.5, 1.3); } This example will scale the div to 100px * 0.5 = 50px on the X axis and to 100px * 1.3 = 130px on the Y axis. The center of the transform is in the center of the div, 50% from left and 50% from top. Section 28.3: Skew html
css .skew { width: 100px; height: 100px; background: teal; transform: skew(20deg, -30deg); } This example will skew the div by 20 degrees on the X axis and by - 30 degrees on the Y axis. The center of the transform is in the center of the div, 50% from left and 50% from top. See the result here. Section 28.4: Multiple transforms Multiple transforms can be applied to an element in one property like this: transform: rotate(15deg) translateX(200px); This will rotate the element 15 degrees clockwise and then translate it 200px to the right. In chained transforms, the coordinate system moves with the element. This means that the translation won't be horizontal but on an axis rotate 15 degrees clockwise as shown in the following image: Changing the order of the transforms will change the output. The first example will be different to transform: translateX(200px) rotate(15deg);
.transform { transform: rotate(15deg) translateX(200px); } As shown in this image: Section 28.5: Translate html
css .translate { width: 100px; height: 100px; background: teal; transform: translate(200px, 50%); } This example will move the div by 200px on the X axis and by 100px * 50% = 50px on the Y axis. You can also specify translations on a single axis. On the X axis: .translate { transform: translateX(200px); } On the Y axis: .translate { transform: translateY(50%); } Section 28.6: Transform Origin Transformations are done with respect to a point which is defined by the transform-origin property. The property takes 2 values : transform-origin: X Y; In the following example the first div (.tl) is rotate around the top left corner with transform-origin: 0 0; and the second (.tr)is transformed around it's top right corner with transform-origin: 100% 0. The rotation is applied on hover : html: css: .transform { display: inline-block; width: 200px; height: 100px; background: teal; transition: transform 1s; } .origin1 { transform-origin: 0 0; } .origin2 { transform-origin: 100% 0; } .transform:hover { transform: rotate(30deg); } The default value for the transform-origin property is 50% 50% which is the center of the element. Chapter 29: 3D Transforms Section 29.1: Compass pointer or needle shape using 3D transforms css div.needle { margin: 100px; height: 150px; width: 150px; transform: rotateY(85deg) rotateZ(45deg); /* presentational */ background-image: linear-gradient(to top left, #555 0%, #555 40%, #444 50%, #333 97%); box-shadow: inset 6px 6px 22px 8px #272727; } html
In the above example, a needle or compass pointer shape is created using 3D transforms. Generally when we apply the rotate transform on an element, the rotation happens only in the Z-axis and at best we will end up with diamond shapes only. But when a rotateY transform is added on top of it, the element gets squeezed in the Y-axis and thus ends up looking like a needle. The more the rotation of the Y-axis the more squeezed the element looks. The output of the above example would be a needle resting on its tip. For creating a needle that is resting on its base, the rotation should be along the X-axis instead of along Y-axis. So the transform property's value would have to be something like rotateX(85deg) rotateZ(45deg);. This pen uses a similar approach to create something that resembles the Safari logo or a compass dial. Screenshot of element with no transform: Screenshot of element with only 2D transform: Screenshot of element with 3D transform: Section 29.2: 3D text e□ect with shadow html:

HOVER

css: *{margin:0;padding:0;} html,body{height:100%;width:100%;overflow:hidden;background:#0099CC;} #title{ position:absolute; top:50%; left:50%; transform:translate(-50%,-50%); perspective-origin:50% 50%; perspective:300px; } h1{ text-align:center; font-size:12vmin; font-family: 'Open Sans', sans-serif; color:rgba(0,0,0,0.8); line-height:1em; transform:rotateY(50deg); perspective:150px; perspective-origin:0% 50%; } h1:after{ content:attr(data-content); position:absolute; left:0;top:0; transform-origin:50% 100%; transform:rotateX(-90deg); color:#0099CC; } #title:before{ content:''; position:absolute; top:-150%; left:-25%; width:180%; height:328%; background:rgba(255,255,255,0.7); transform-origin: 0 100%; transform: translatez(-200px) rotate(40deg) skewX(35deg); border-radius:0 0 100% 0; } View example with additional hover effect In this example, the text is transformed to make it look like it is going into the screen away from the user. The shadow is transformed accordingly so it follows the text. As it is made with a pseudo element and the data attribute, it inherits the transforms form it's parent (the H1 tag). The white "light" is made with a pseudo element on the #title element. It is skewed and uses border-radius for the rounded corner. Section 29.3: backface-visibility The backface-visibility property relates to 3D transforms. With 3D transforms and the backface-visibility property, you're able to rotate an element such that the original front of an element no longer faces the screen. For example, this would flip an element away from the screen: JSFIDDLE
Loren ipsum
Lorem ipsum

.flip { -webkit-transform: rotateY(180deg); -moz-transform: rotateY(180deg); -ms-transform: rotateY(180deg); -webkit-backface-visibility: visible; -moz-backface-visibility: visible; -ms-backface-visibility: visible; } .flip.back { -webkit-backface-visibility: hidden; -moz-backface-visibility: hidden; -ms-backface-visibility: hidden; } Firefox 10+ and IE 10+ support backface-visibility without a prefix. Opera, Chrome, Safari, iOS, and Android all need -webkit-backface-visibility. It has 4 values: 1. visible (default) - the element will always be visible even when not facing the screen. 2. hidden - the element is not visible when not facing the screen. 3. inherit - the property will gets its value from the its parent element 4. initial - sets the property to its default, which is visible Section 29.4: 3D cube 3D transforms can be use to create many 3D shapes. Here is a simple 3D css cube example: html:
css: body { perspective-origin: 50% 100%; perspective: 1500px; overflow: hidden; } .cube { position: relative; padding-bottom: 20%; transform-style: preserve-3d; transform-origin: 50% 100%; transform: rotateY(45deg) rotateX(0); } .cubeFace { position: absolute; top: 0; left: 40%; width: 20%; height: 100%; margin: 0 auto; transform-style: inherit; background: #C52329; box-shadow: inset 0 0 0 5px #333; transform-origin: 50% 50%; transform: rotateX(90deg); backface-visibility: hidden; } .face2 { transform-origin: 50% 50%; transform: rotatez(90deg) translateX(100%) rotateY(90deg); } .cubeFace:before, .cubeFace:after { content: ''; position: absolute; width: 100%; height: 100%; transform-origin: 0 0; background: inherit; box-shadow: inherit; backface-visibility: inherit; } .cubeFace:before { top: 100%; left: 0; transform: rotateX(-90deg); } .cubeFace:after { top: 0; left: 100%; transform: rotateY(90deg); } View this example Additional styling is added in the demo and a transform is applied on hover to view the 6 faces of the cube. Should be noted that: 4 faces are made with pseudo elements chained transforms are applied Chapter 30: Filter Property Value Description blur(x) Blurs the image by x pixels. brightness(x) Brightens the image at any value above 1.0 or 100%. Below that, the image will be darkened. contrast(x) Provides more contrast to the image at any value above 1.0 or 100%. Below that, the image will get less saturated. drop-shadow(h, v, x, y, z) Gives the image a drop-shadow. h and v can have negative values. x, y, and z are optional. greyscale(x) Shows the image in greyscale, with a maximum value of 1.0 or 100%. hue-rotate(x) Applies a hue-rotation to the image. invert(x) Inverts the color of the image with a maximum value of 1.0 or 100%. opacity(x) Sets how opaque/transparent the image is with a maximum value of 1.0 or 100%. saturate(x) Saturates the image at any value above 1.0 or 100%. Below that, the image will start to de-saturate. sepia(x) Converts the image to sepia

with a maximum value of 1.0 or 100%. Section 30.1: Blur html css img { -webkit-filter: blur(1px); filter: blur(1px); } Result Makes you wanna rub your glasses. Section 30.2: Drop Shadow (use box-shadow instead if possible) html

My shadow always follows me.

css p { -webkit-filter: drop-shadow(10px 10px 1px green); filter: drop-shadow(10px 10px 1px green); } Result Section 30.3: Hue Rotate html

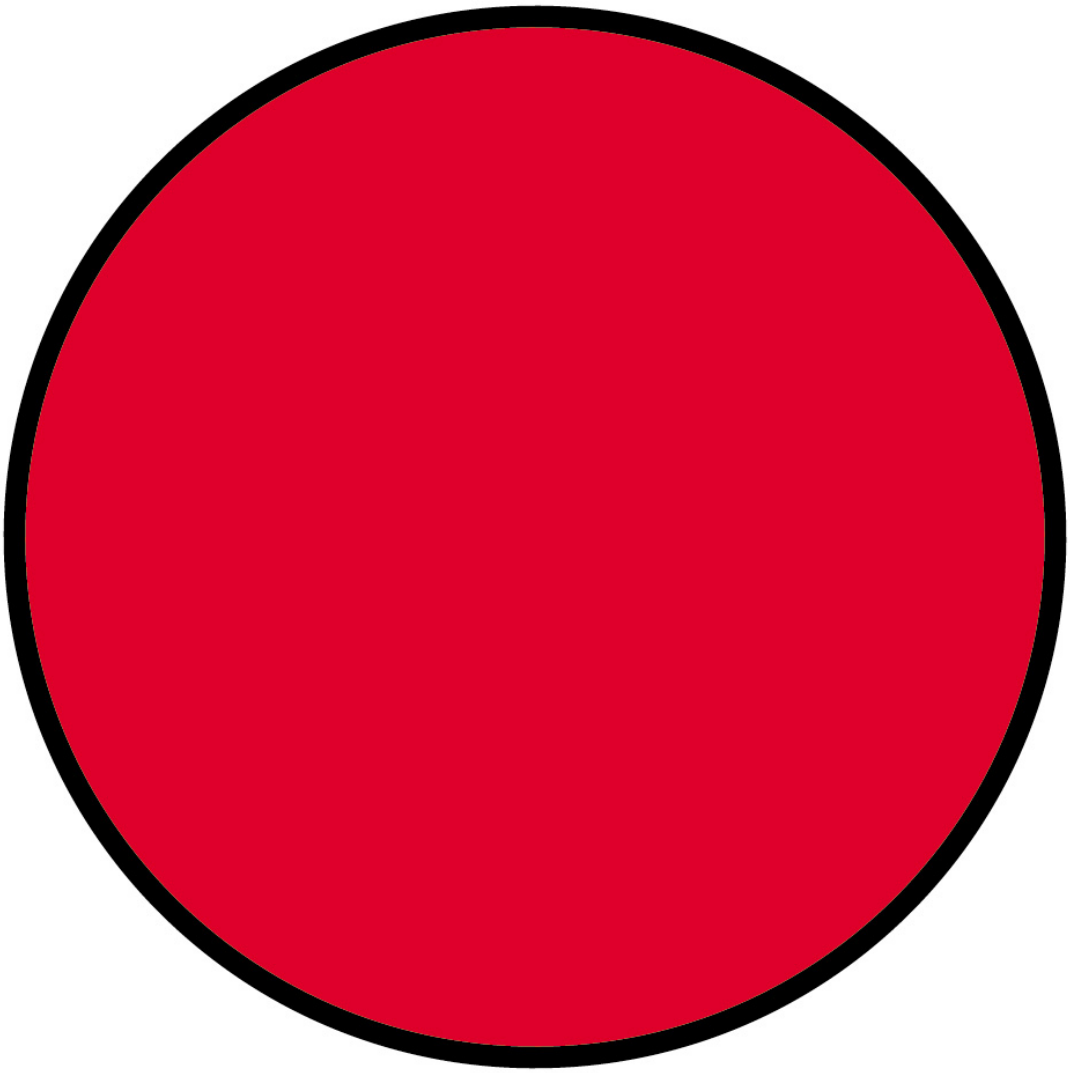
css img { -webkit-filter: hue-rotate(120deg); filter: hue-rotate(120deg); } Result Section 30.4: Multiple Filter Values To use multiple filters, separate each value

with a space. html css img { -webkit-filter: brightness(200%) grayscale(100%) sepia(100%) invert(100%); filter: brightness(200%) grayscale(100%) sepia(100%) invert(100%); } Result Section 30.5: Invert Color html
css div { width: 100px; height: 100px; background-color: white; -webkit-filter: invert(100%); filter: invert(100%); } Result Turns from white to black. Chapter 31: Cursor Styling Section 31.1: Changing cursor type cursor: value; Examples: Value Description none No cursor is rendered for the element auto Default. The browser sets a cursor help The cursor indicates that help is available wait The cursor indicates that the program is busy move The cursor indicates something is to be moved pointer The cursor is a pointer and indicates a link Section 31.2: pointer-events The pointer-events property allows for control over how html elements respond to mouse/touch events. .disabled { pointer-events: none; } In this example, 'none' prevents all click, state and cursor options on the specified html element [[1]] Other valid values for html elements are: auto; inherit. 1. https://css-tricks.com/almanac/properties/p/pointer-events/ Other resources: https://developer.mozilla.org/en-US/docs/Web/css/pointer-events https://davidwalsh.name/pointer-events Section 31.3: caret-color The caret-color css property specifies the color of the caret, the visible indicator of the insertion point in an element where text and other content is inserted by the user's typing or editing. html

css #example { caret-color: red; } Resources: https://developer.mozilla.org/en-US/docs/Web/css/caret-color Chapter 32: box-shadow Parameters Details inset by default, the shadow is treated as a drop shadow. the inset keyword draws the shadow inside the frame/border. offset-x the horizontal distance offset-y the vertical distance blur-radius 0 by default. value cannot be negative. the bigger the value, the bigger and lighter the shadow becomes. spread-radius 0 by default. positive values will cause the shadow to expand. negative values will cause the shadow to shrink. color can be of various notations: a color keyword, hexadecimal, rgb(), rgba(), hsl(), hsla() Section 32.1: bottom-only drop shadow using a pseudoelement JSFiddle: https://jsfiddle.net/UnsungHero97/80qod7aL/2/ html
css .box_shadow { background-color: #1C90F3; width: 200px; height: 100px; margin: 50px; } .box_shadow:after { content: ""; width: 190px; height: 1px; margin-top: 98px; margin-left: 5px; display: block; position: absolute; z-index: -1; -webkit-box-shadow: 0px 0px 8px 2px #444444; -moz-box-shadow: 0px 0px 8px 2px #444444; box-shadow: 0px 0px 8px 2px #444444; } Section 32.2: drop shadow JSFiddle: https://jsfiddle.net/UnsungHero97/80qod7aL/ html
css .box_shadow { -webkit-box-shadow: 0px 0px 10px -1px #444444; -moz-box-shadow: 0px 0px 10px -1px #444444; box-shadow: 0px 0px 10px -1px #444444; } Section 32.3: inner drop shadow html
css .box_shadow { background-color: #1C90F3; width: 200px; height: 100px; margin: 50px; -webkit-box-shadow: inset 0px 0px 10px 0px #444444; -moz-box-shadow: inset 0px 0px 10px 0px #444444; box-shadow: inset 0px 0px 10px 0px #444444; } Result: JSFiddle: https://jsfiddle.net/UnsungHero97/80qod7aL/1/ Section 32.4: multiple shadows JSFiddle: https://jsfiddle.net/UnsungHero97/80qod7aL/5/ html
css .box_shadow { width: 100px; height: 100px; margin: 100px; box-shadow: -52px -52px 0px 0px #f65314, 52px -52px 0px 0px #7cbb00, -52px 52px 0px 0px #00a1f1, 52px 52px 0px 0px #ffbb00; } Chapter 33: Shapes for Floats Parameter Details none A value of none means that the float area (the area that is used for wrapping content around a float element) is unaffected. This is the default/initial value. basic-shape Refers to one among inset(), circle(), ellipse() or polygon(). Using one of these functions and its values the shape is defined. shape-box Refers to one among margin-box, border-box, padding-box, content-box. When only is provided (without) this box is the shape. When it is used along with , this acts as the reference box. image When an image is provided as value, the shape is computed based on the alpha channel of the image specified. Section 33.1: Shape Outside with Basic Shape – circle() With the shape-outside css property one can define shape values for the float area so that the inline content wraps around the shape instead of the float's box. css img:nth-of-type(1) { shape-outside: circle(80px at 50% 50%); float: left; width: 200px; } img:nth-of-type(2) { shape-outside: circle(80px at 50% 50%); float: right; width: 200px; } p { text-align: center; line-height: 30px; /* purely for demo */ } html

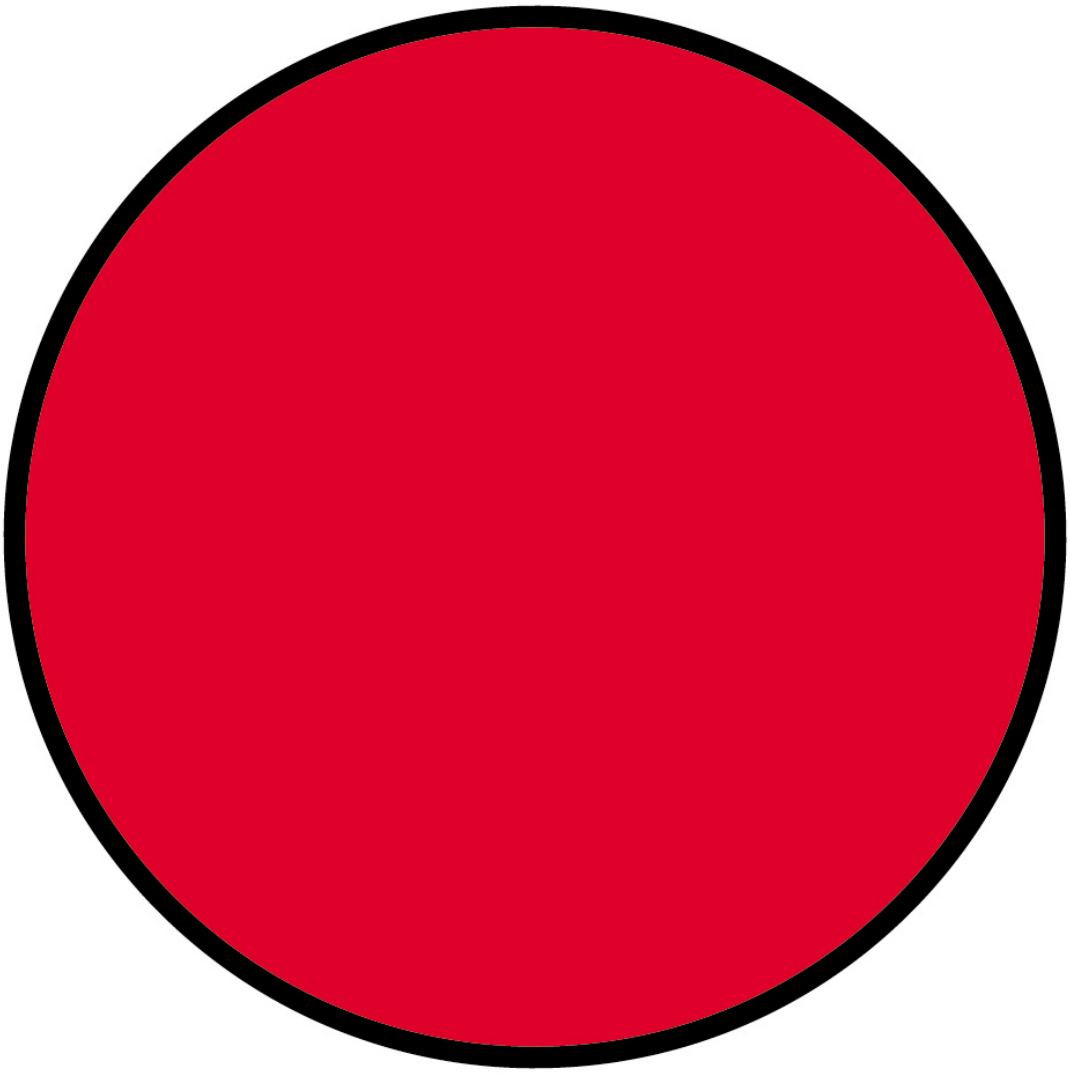
Some paragraph whose text content is required to be wrapped such that it follows the curve of the circle on either side. And then there is some filler text just to make the text long enough. Lorem Ipsum Dolor Sit Amet....

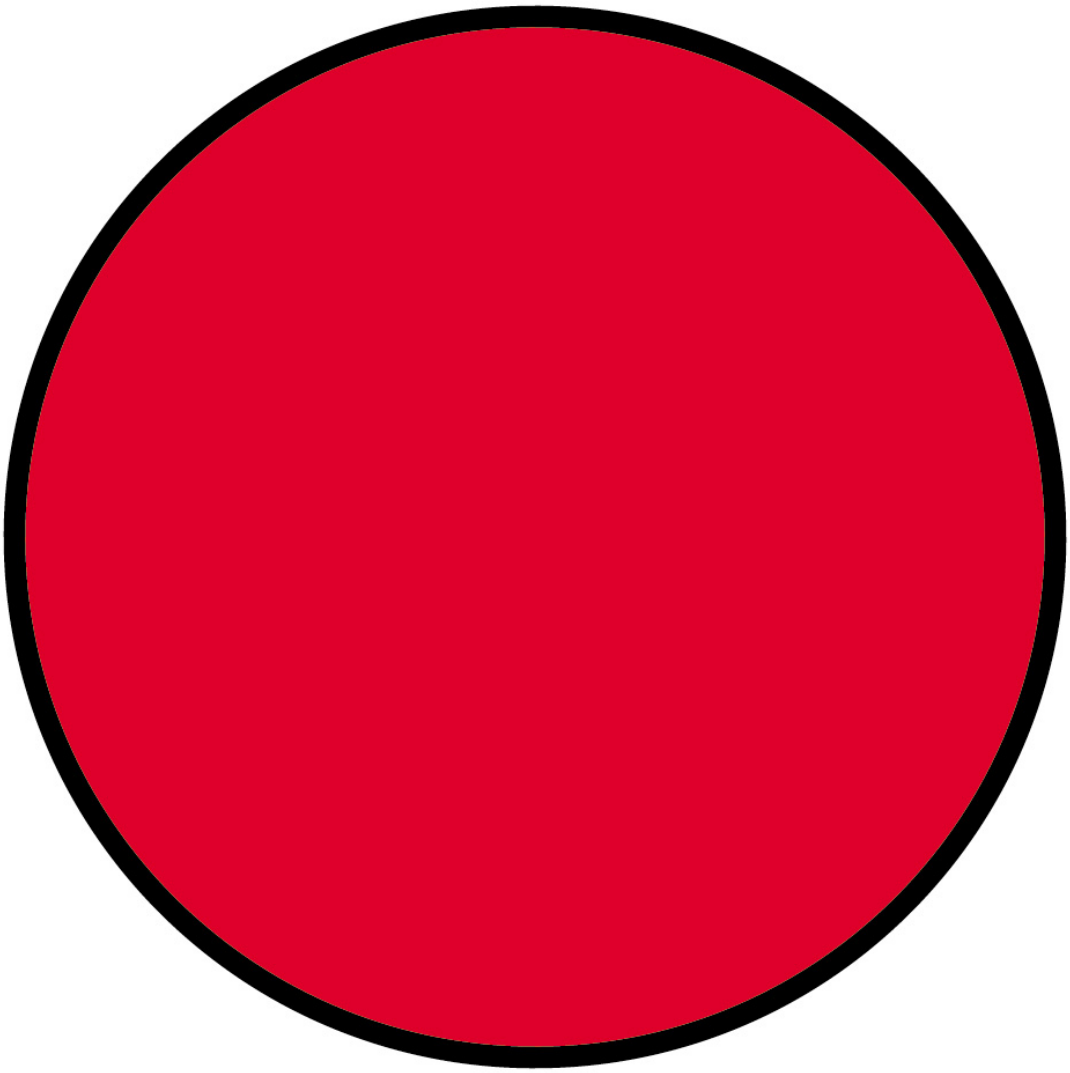
In the above example, both the images are actually square images and when the text is placed without the shapeoutside property, it will not flow around the circle on either side. It will flow around the containing box of the image only. With shape-outside the float area is re-defined as a circle and the content is made to flow around this imaginary circle that is created using shape-outside. The imaginary circle that is used to re-define the float area is a circle with radius of 80px drawn from the center-mid point of the image's reference box. Below are a couple of screenshots to illustrate how the content would be wrapped around when shape-outside is used and when it is not used. Output with shape-outside Output without shape-outside Section 33.2: Shape margin The shape-margin css property adds a margin to shape-outside. css img:nth-of-type(1) { shape-outside: circle(80px at 50% 50%); shape-margin: 10px; float: left; width: 200px; } img:nth-of-type(2) { shape-outside: circle(80px at 50% 50%); shape-margin: 10px; float: right; width: 200px; } p { text-align: center; line-height: 30px; /* purely for demo */ } html

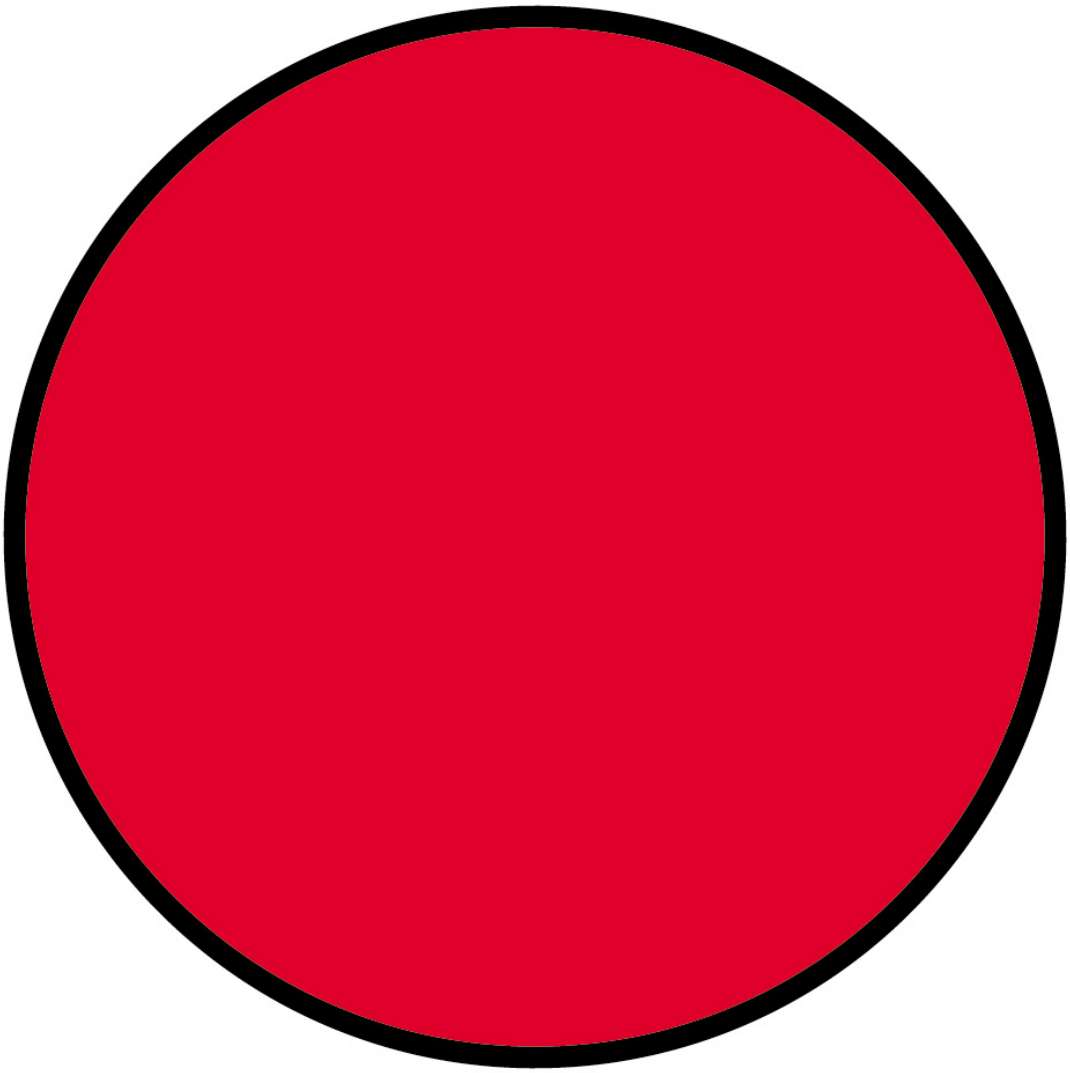


Some paragraph whose text content is required to be wrapped such that it follows the curve of the circle on either side. And then there is some filler text just to make the text long enough. Lorem Ipsum Dolor Sit Amet....

In this example, a 10px margin is added around the shape using shape-margin. This creates a bit more space between the imaginary circle that defines the float area and the actual content that is flowing around. Output: Chapter 34: List Styles Value Description list-style-type the type of list-item marker. list-style-position specifies where to place the marker list-style-image specifies the type of list-item marker initial sets this property to its default value inherit inherits this property from its parent element Section 34.1: Bullet Position A list consists of







elements inside a containing element (

or

). Both the list items and the container can have margins and paddings which influence the exact p
specifically. Each list item gets a 'marker box', which contains the bullet marker. This box can eithe

element, pushing the content to the right as needed. list-style-po

element. If there is not enough space in the padding of the conta
positioning: isfiddle Section 34.2: Removing Bullets / Numbers S

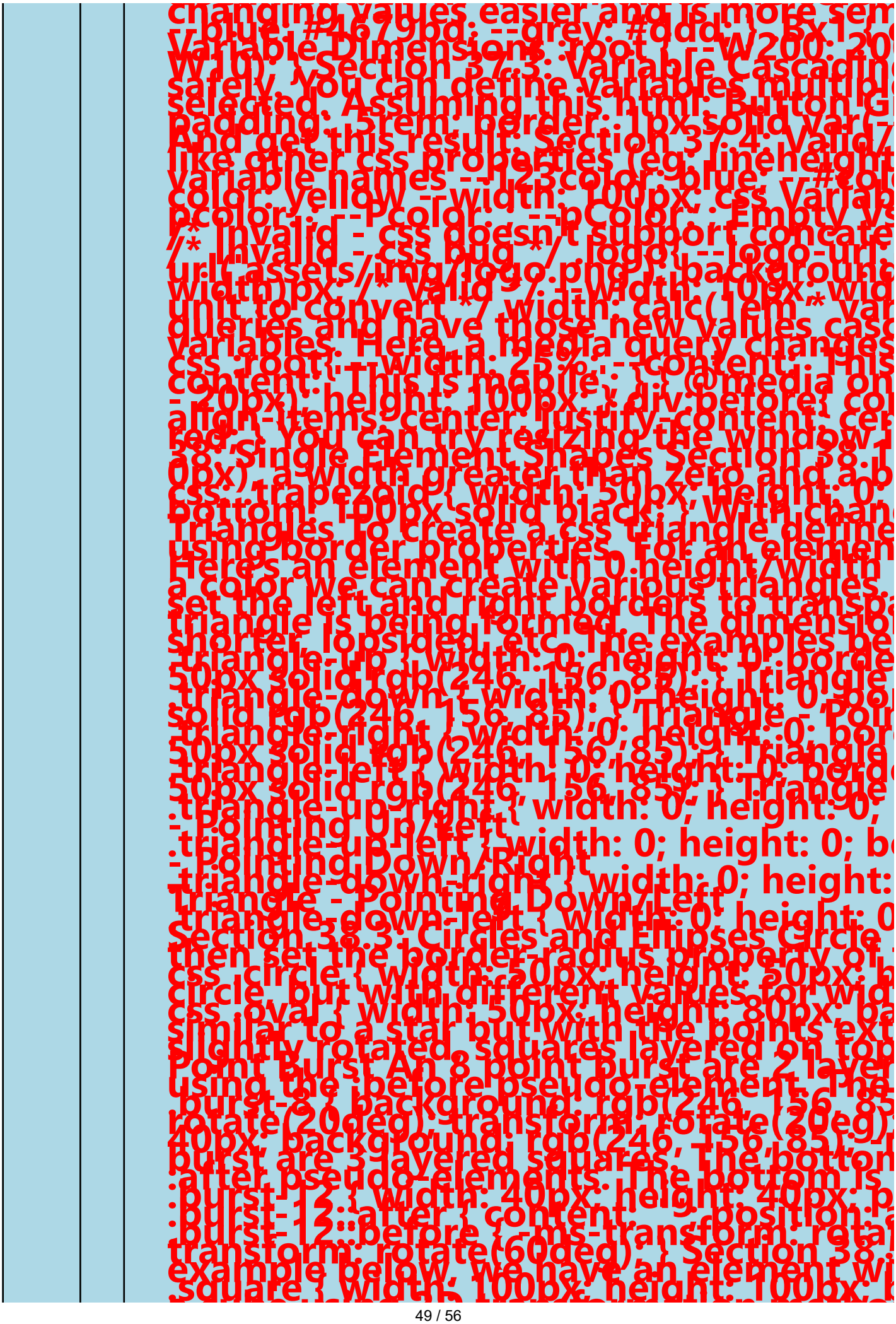
first item second item

css ul { list-style-type: none; } li { margin: 0; padding: 0; } Section

tags within an unordered list (
): list-style: disc; /* A filled circle (default) */ list-style: circle; /*

tags within an ordered list (
p list-style: decimal; /* Decimal numbe
padded by initial zeros (01, 02, 03, 1
upper-roman; /* Uppercase roman num
y); } /* list-style-type: lower-alpha;
('a', 'b', 'c', d Rom) */ list-style-type: upper


Level 1
Level 1.1
Level 3



a cube using 2D transformation metho
css .cube { background: #dc2e2e; widt
display: inline-block; background: #f15
-20px; left: 8px; } .cube:after { content:
skewY(-50deg); position: absolute; top:
create a pyramid using borders and 2D
css .pyramid { width: 100px; height: 20
display: inline-block; width: 0; height: 0
transparent #f13656 transparent; transi
transparent transparent #d64444 transi
Section 39.1: Simple Example (column-
Code
Lorem ipsum dolor sit amet, consectetu
ut enim ad minim veniam, quis nostruc
dolor in reprehenderit in voluptate vel
proident, sunt in culpa qui officia dese
.multi-columns { -moz-column-count: 4
column-width property sets the minim
columns as fit in the available width. C
Lorem ipsum dolor sit amet, consectetu
ut enim ad minim veniam, quis nostruc
dolor in reprehenderit in voluptate vel
proident, sunt in culpa qui officia dese
.multi-columns { -moz-column-width: 1
Multiple columns css allows to define t
Section 40.1: Create Multiple Columns
Lorem ipsum dolor sit amet, consectetu
aliquam erat volutpat. Ut wisi enim ad
ex ea commodo consequat. Duis autem
dolore eu feugiat nulla facilisis at vero
augue duis dolore te feugait nulla facili
id quod mazim placerat facer possim as
css .content { -webkit-column-count: 3;
Section 40.2: Basic example Consider th
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dol

Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, con
dolor sit amet.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dol

With the following css applied the cont
columns: 3; column-gap: 40px; column-
Layout Section 41.1: Justified navigatio
should be justified. The first (left) item
within the container. The distance betw

abc abcdefghijkl abcde

css .nav { width: 100%; line-height: 1.4e
justify; margin-bottom: -1.4em; } ul.aft
The .nav, ul and li tags were chosen for
used of course. The :after pseudo-elem
pushing other content down. This is so
line-height (but negative). If the page l
from the right) and be justified on this
42.1: Automatic inheritance Inheritance
properties of an element are applied to
elements rather than having to set said
automatically inherited are font, color,
padding: 5px; } This will apply color: re
element but also to the

and

elements. However, due to the nature of padding its value will not be inherited to those elements.

Some header

Some paragraph

Section 42.2: Enforced inheritance S
children. This is because those prop
elements) to which the property is g
etc. However, sometimes inheritance
property that should be inherited. #h
Assume the following stylesheet: #h
This will apply color: red to both the

and

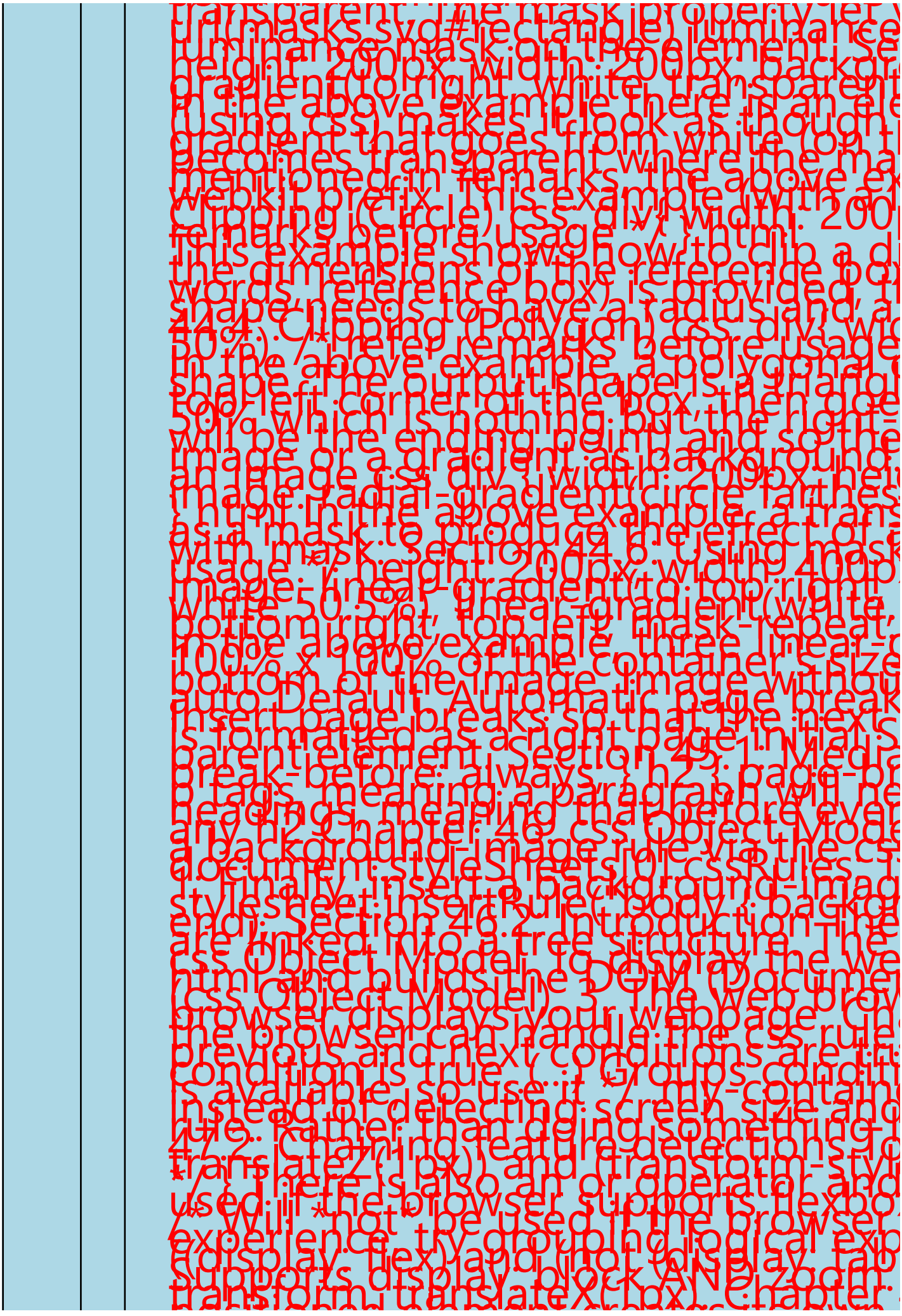
elements due to the inheritance nature of the color property. However, the

element will also inherit the padding value from its' parent because this was specified.

Some header

Some paragraph

Chapter 43: css Image Sprites Sectio
single asset located within an image
one asset that can be extracted from
those stars is a sprite with the spri
reducing the number of HTTP reque
some example code: html
css: .icon3 {background-url("icons-sp
background-position: 0px 0px; /* ic
0px */} By using setting the sprite's w
and y value, you can easily extract
parameter defaults clip-source. A URL
file that contains the clip paths defi
using one of these functions. The cli
do in. Shapes for floats clip-geomet
margin-box, fill-box, stroke-box, vie
correspond box is used as the ba
mask-reference. This can be none or
specifies how the mask should be re
should be treated as a alpha mask. O
repeat, space, round, no-repeat, r
image then it would be considered
as luminance mask. position This sp
background-position property. The k
top right 50% 50%; /* geometry-box
area) or the box which should be us
property. The first of possible values
view-box. Detailed explanation of h
represents the size of each mask-im
or percentage or auto or cover or co
as one for each axis. composing-o
and defines the type of compositing
explanation about each value. Mask
Difference With clipping and Mask
both can be applied to any html ele
transparent inside. SVG opaque. The
that also exists in SVG you can use
ellipse() Example clip-path: circle(1
positioned at the center of the elem
of defining a path you define a mas
consist of mainly two colors: black a
is transparent, but there is also a
Alpha Mask. Only on the transpare
used as a luminance mask to make



positioned element creates its own stacking context, so content inside Division Element #4 will stack in front of Division Element #5 even though the latter has a higher number. Note that Div #4, Div #5, and Div #6 within Div #3 create stacking contexts organized as stacking in the root element with re

Division Element #1

position: relative;
z-index: 5;

Division Element #2

position: relative;
z-index: 2;

Division Element #4

position: relative;
z-index: 6;

Division Element #3

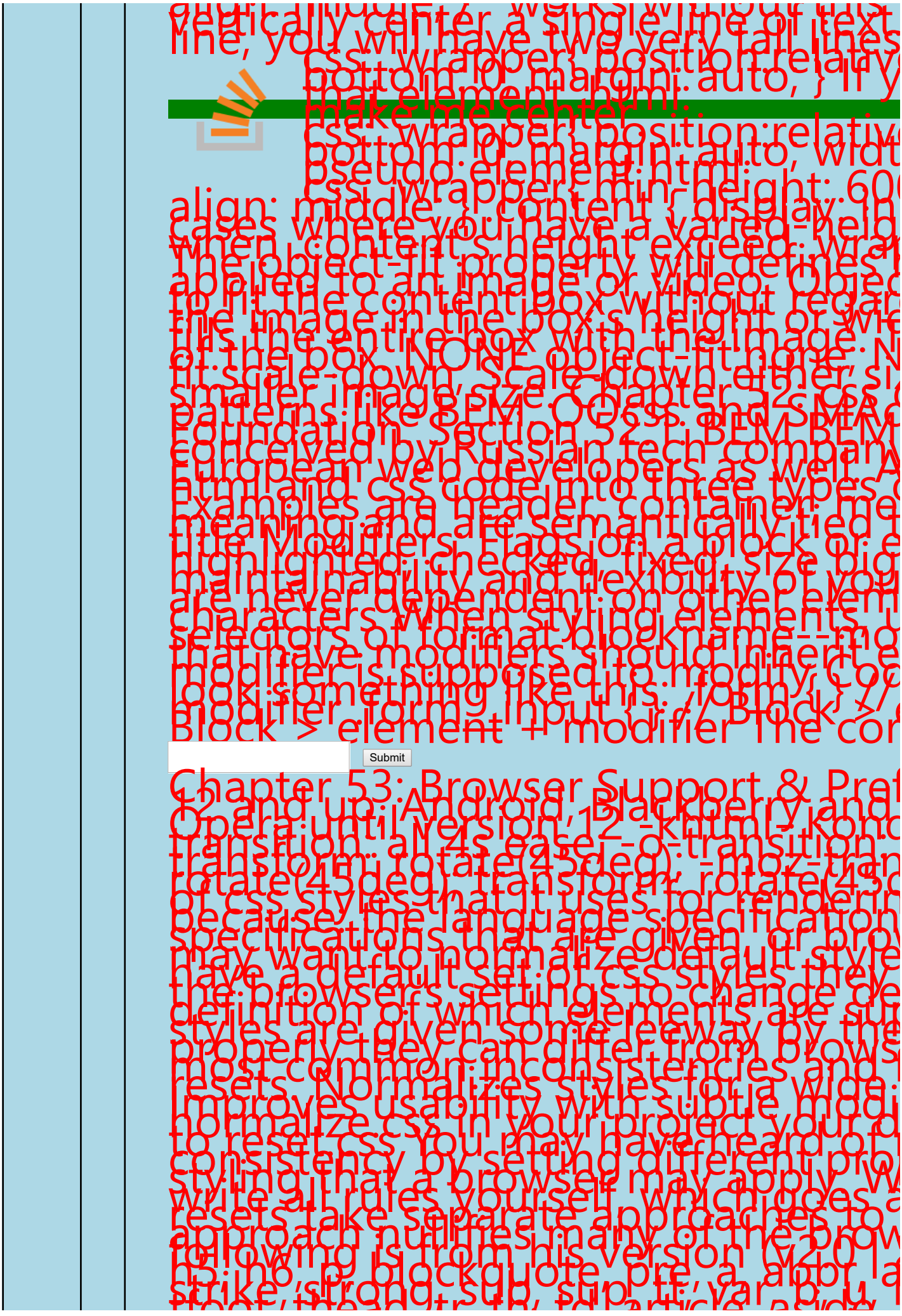
position: absolute;
z-index: 4;

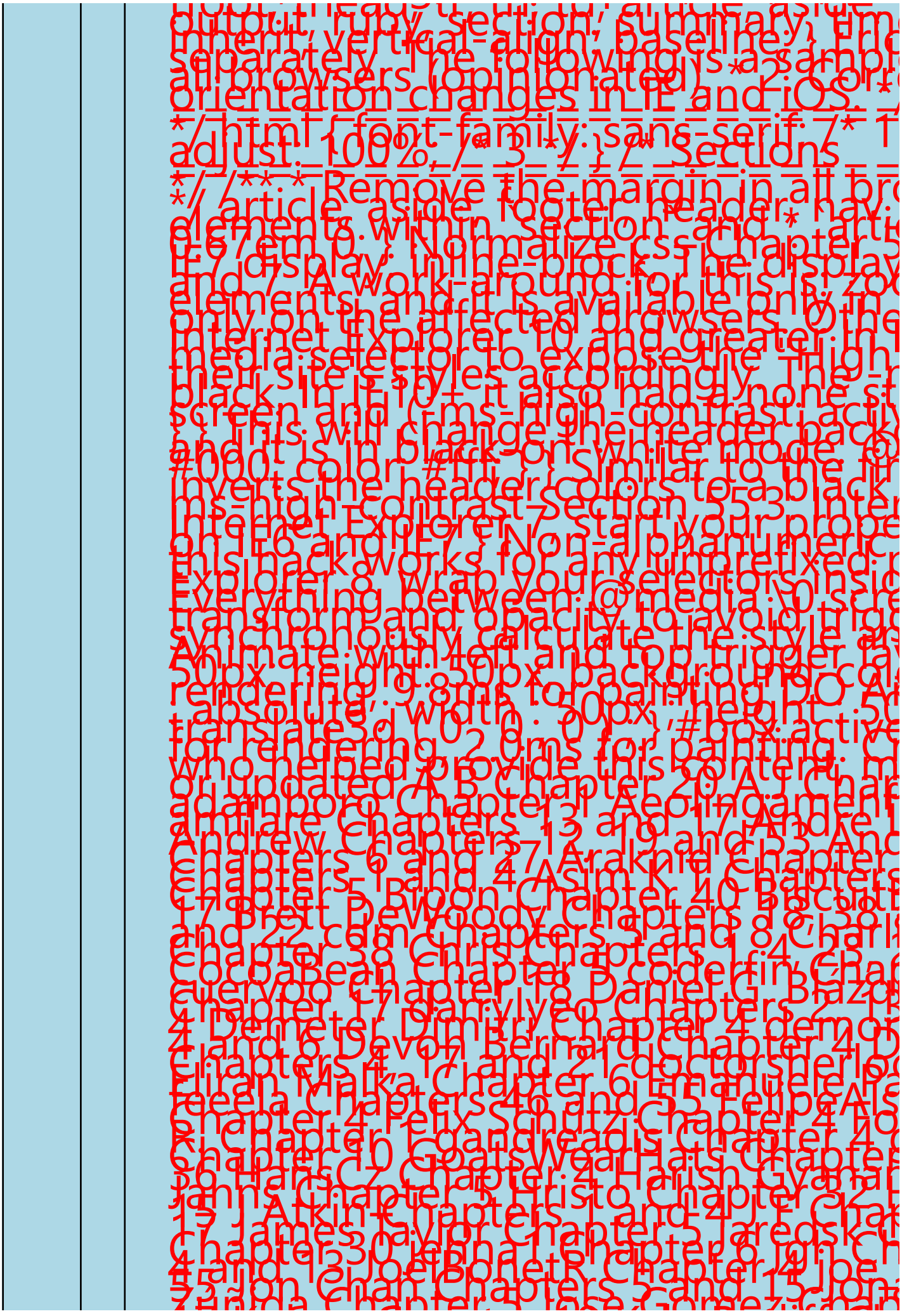
Division Element #5

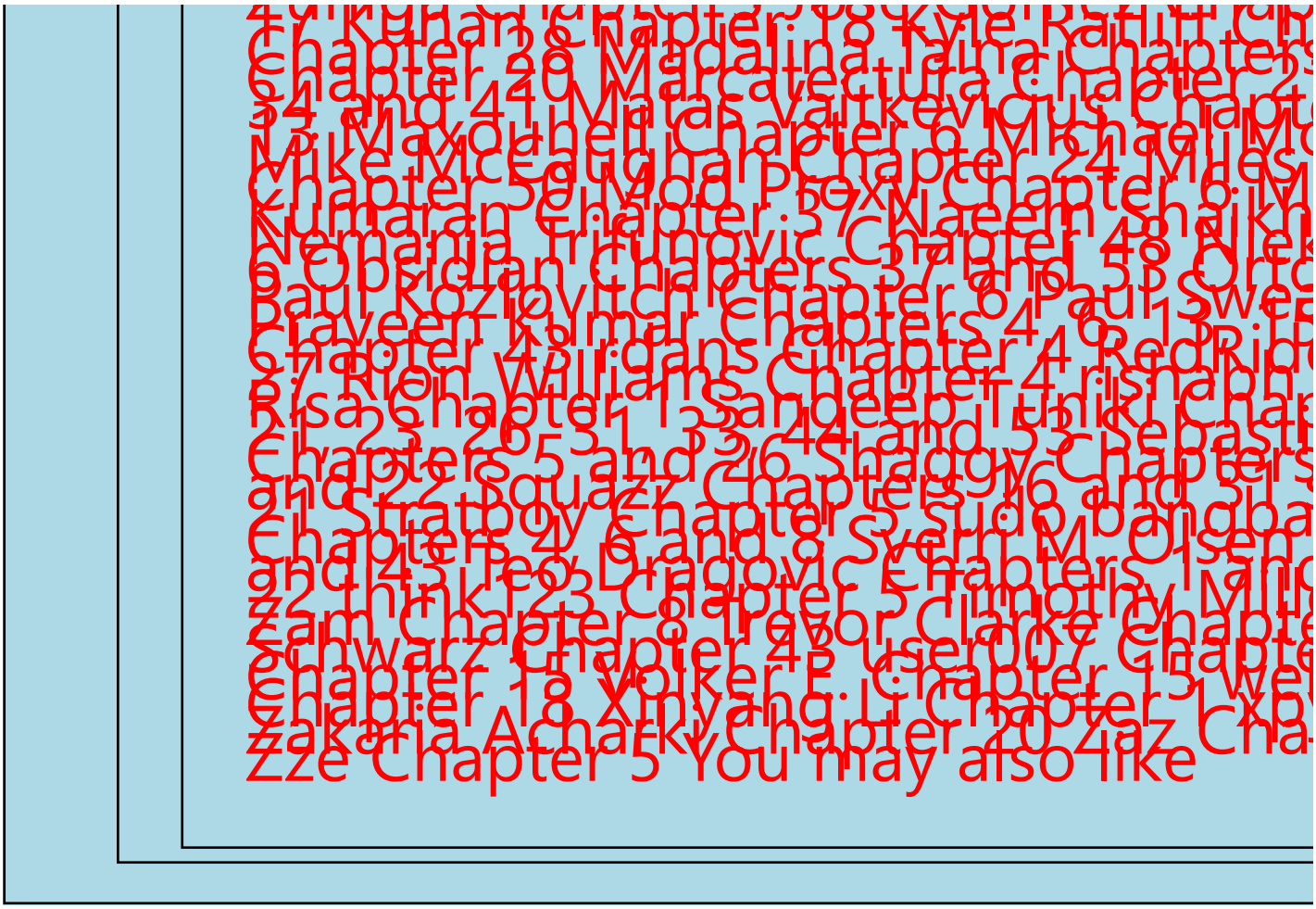
position: relative;
z-index: 1;

Division Element #6

position: absolute;
z-index: 3;


Submit




Chapter 5 You may also like