



# CSS 排版方式

v.20



田甜甜教案實驗室



@tientientien.design



# CSS TYPESETTING

---

## CSS 佈局

行內

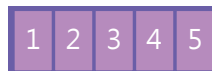
**display: inline;**

display:inline;

元素顯示形態：行內元素。

寬高樣式設定：不支援。

未設定寬高時：元素自動抓取內容尺寸。



區塊

**display: block;**

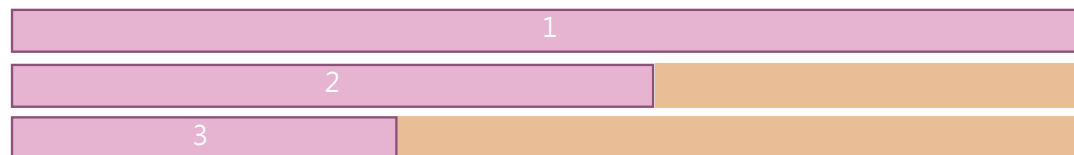
display:block;

元素顯示形態：區塊。

寬高樣式設定：有支援。

未設定寬高時：寬度自動延展，高度自動抓取內容尺寸。

註：自動補滿寬度，區塊無法併排。



行內區塊

**display: inline-block;**

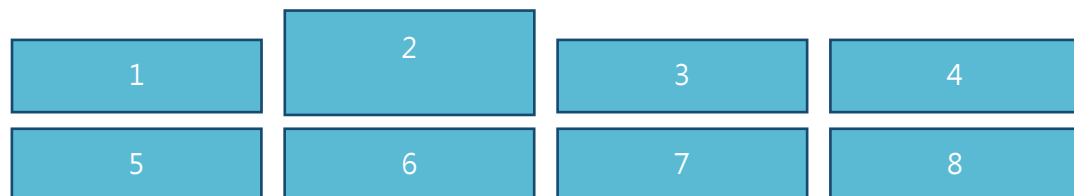
display:inline-block;

元素顯示形態：行內區塊。

寬高樣式設定：有支援。

未設定寬高時：元素自動抓取內容尺寸。

註：每個區塊於網頁中會佔用一個空白的間距。



浮動區塊

**float: left | right;**

float:left;

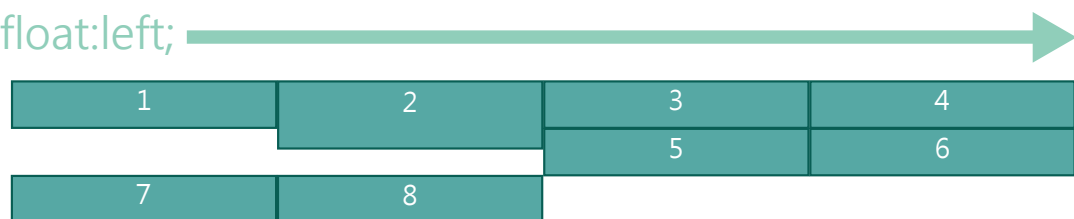
元素顯示形態：浮動區塊。

寬高樣式設定：有支援。

未設定寬高時：寬高自動抓取內容尺寸。

註：後方的元素會受影響，

使用clear:both或clear:left可清除設定。



float:right;



# FLEXBOX LAYOUT

---

## CSS 彈性佈局

flexbox學習網站 : <http://flexboxfroggy.com/>



## display: flex | inline-flex;

定義父層為彈性區塊，使父層提供彈性環境給所有子層

// 以flex主軸水平方向(row)為例

彈性區塊

display: flex;

元素顯示形態：彈性區塊。

寬高樣式設定：有支援。

未設定寬高時：父層主軸方向自動延展，副軸自動抓取內容尺寸。

子元素的主軸方向會抓內容尺寸，副軸方向則自動延展。

display: flex;



子元素設定寬高時：

子元素設定主軸方向尺寸時，於父層中自動伸展均分，超過父層尺寸時，則自動收縮。副軸方向尺寸(如height:50px;)則正常顯示。

(如父層寬為300，子元素寬為100，則自動收縮為75)。

行內彈性區塊

display: inline-flex;

元素顯示形態：行內彈性區塊。

寬高樣式設定：有支援。

未設定寬高時：寬高自動抓取內容尺寸。

註1：每個區塊於網頁中會佔用一個空白的間距。

註2：主軸方向屬性無效，副軸方向屬性有效。

display: inline-flex;



子元素設定寬高時：

子元素設定尺寸時，均正常顯示。

## 父層Flexbox (容器元素)

- **flex-direction**: 建立父層容器主軸方向(水平或垂直/正向或逆向)  
row(預設) | row-reverse | column | column-reverse;
- **flex-wrap**: 建立父層容器換行方式(不換行/正向換行/逆向換行)  
nowrap(預設) | wrap | wrap-reverse;  
// 縮寫 : **flex-flow**: < 'flex-direction' > || < 'flex-wrap' >
- **justify-content**: 主軸元素的排列方式(對齊/平均)  
flex-start(預設) | flex-end | center | space-between | space-around | space-evenly;
- **align-items**: 單行元素在副軸的排列方式(對齊/平均)  
flex-start | flex-end | center | baseline | stretch(預設);
- **align-content**: 多行元素在副軸的排列方式  
flex-start | flex-end | center | space-between | space-around | stretch(預設);

## 子層Flexbox (項目元素)

- **align-self**: 彈性子元素元素副軸的單一位置控制  
auto(預設) | flex-start | flex-end | center | baseline | stretch(需搭配高度auto);
- **order**: 控制單一元素排列順序(預設為0)
- **flex-grow**: 彈性子元素可用空間的分配 (預設為0)
- **flex-shrink**: 彈性子元素空間不足分配收縮比例 (預設為1)
- **flex-basis**: 單一元素基本尺寸 (預設為 auto)  
// 縮寫 : **flex**: < 'flex-grow' > | < 'flex-shrink' > | < 'flex-basis' >

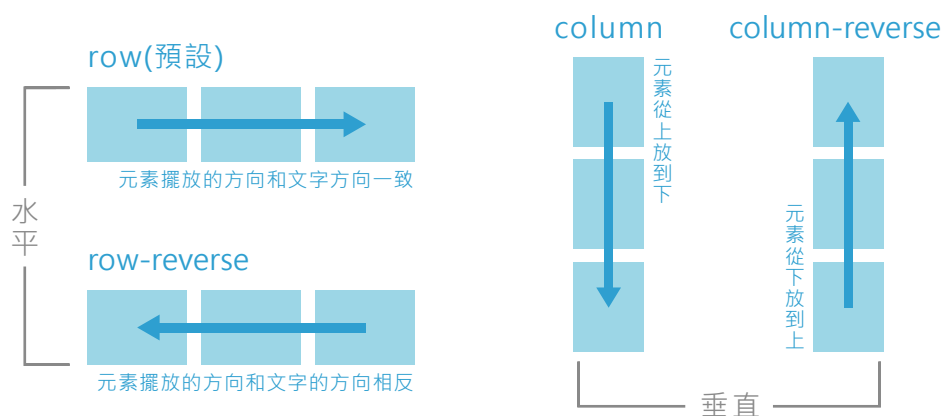
## 父層Flexbox (容器元素) / Flex Direction(彈性方向)

定義父層為彈性區塊，使父層提供彈性環境給所有子層，僅對下一層子層作用。

建立父層容器內主軸方向(水平或垂直/正向或逆向)

`flex-direction: /*彈性方向*/`

`row | row-reverse | column | column-reverse;`

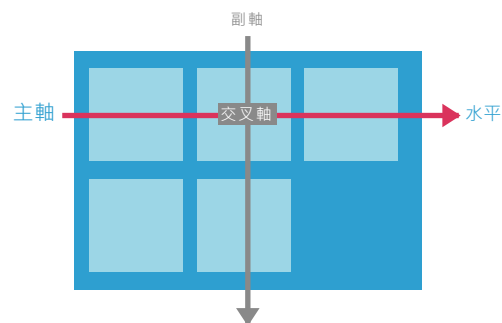


這個屬性是放在父元素盒子裡的，用來控制子元素排列的軸(水平軸、垂直軸)

主軸、副軸 (交叉軸) 基本觀念

`flex-direction: row;`

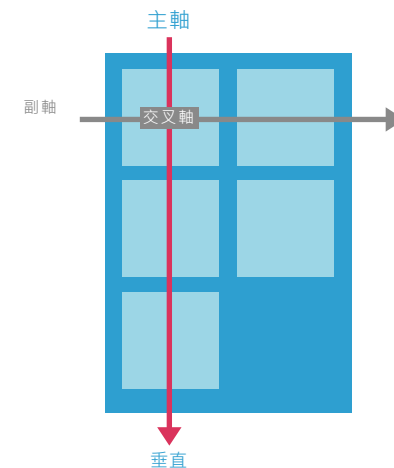
主軸:水平



`/* 主軸水平方向寬度自動延展 */`

`flex-direction: column;`

主軸:垂直



`/* 主軸垂直方向寬度不會自動延展 */`

## 父層（容器元素） / Flex Wrap(彈性換行)

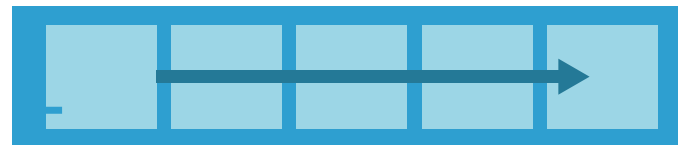
以主軸水平方向(row)為例

建立父層容器內換行方式(不換行/正向換行/逆向換行)

`flex-wrap: /*彈性換行*/`  
`nowrap | wrap | wrap-reverse;`

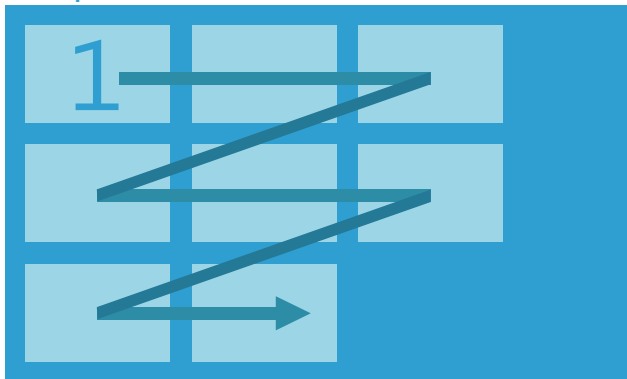


nowrap(預設)



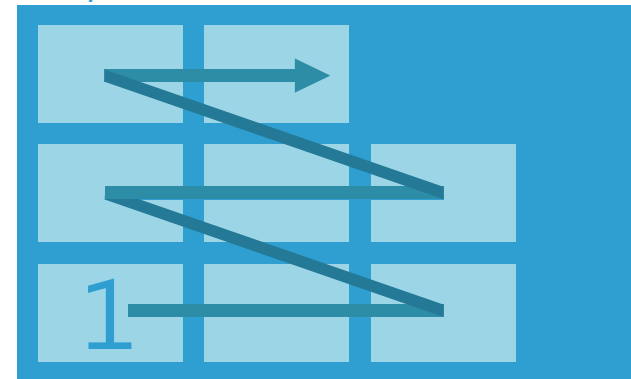
所有的元素都在一行，並自動伸縮子元素寬度

wrap



元素自動換成多行

wrap-reverse



元素自動換成逆序的多行

`flex-flow: < 'flex-direction' > | < 'flex-wrap' >`

`flex-flow: row wrap;`



## 父層Flexbox (容器元素) / justify-content(主軸對齊方式) | align-items(副軸對齊方式)

以flex主軸水平方向(row)為例

justify-content: /\*主軸對齊方式\*/

flex-start | flex-end | center |

space-between | space-around | space-evenly;

控制父層內資料流的主軸分佈(對齊/平均)/主軸元素的排列方式

- 只控制「主軸」 -

flex-start(預設值-齊頭)



元素和容器的左端對齊

flex-end(齊尾)



元素和容器的右端對齊

center(水平居中)



元素在容器裏居中

space-between(貼齊start/end)



元素之間保持相等的距離

space-around(均分間距至二側)



元素周圍保持相等的距離

space-evenly(均分間距範圍)



align-items: /\*副軸對齊方式\*/

flex-start | flex-end | center |

baseline | stretch;

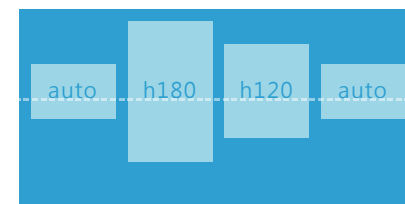
控制父層內資料流的副軸分佈(對齊/平均)/副軸(交叉軸)的排列方式

- 只控制「副軸」 -

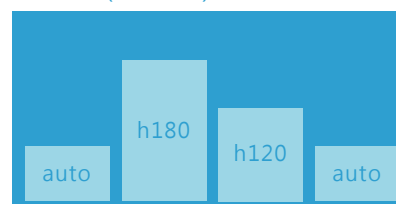
flex-start(對齊頂部)



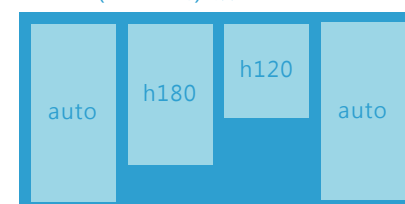
baseline(基線對齊)



flex-end(對齊底部)

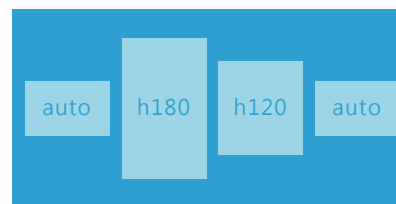


stretch(拉伸填滿)-預設



· 仍然尊重最小寬度/最大寬度  
· 子元素設定高度，則拉伸無效

center(垂直居中)



## 父層 Flexbox (容器元素) align-content(副軸多行對齊方式)

以flex主軸水平方向(row)為例

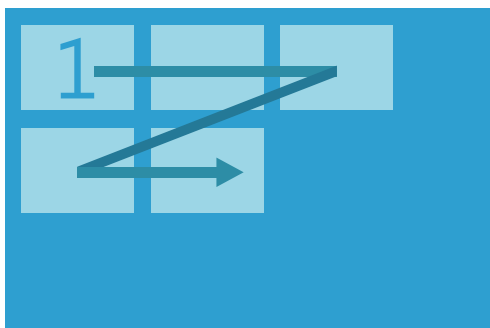
align-content: /\*副軸多行對齊方式\*/  
flex-start | flex-end | center | space-between | space-around | stretch;

建立父層資料流的對齊(控制整塊對齊)/副軸(交叉軸)的多行排列方式

/\*只有一行柔性項目時，此屬性不起作用\*/

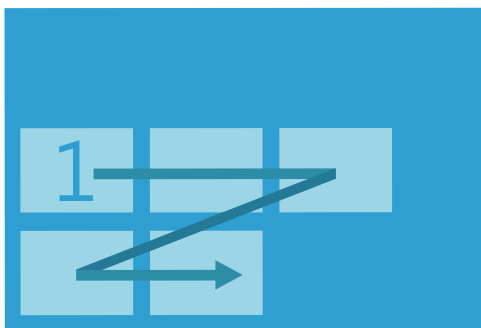
-副軸(交叉軸)多行的對齊方式-

flex-start



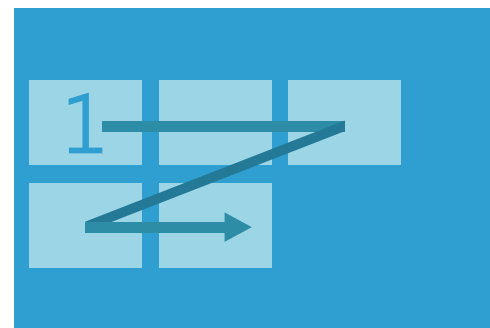
多行都集中在頂部

flex-end



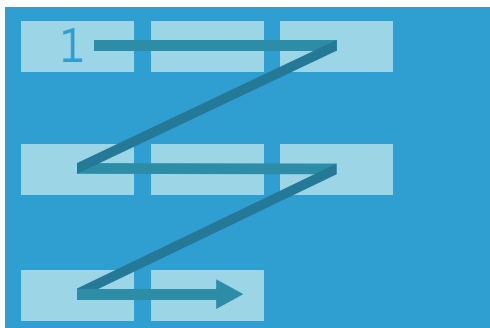
多行都集中在底部

center



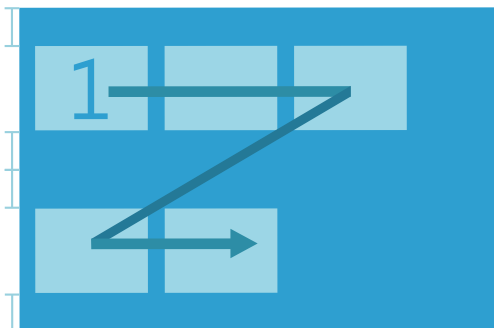
多行居中

space-between(置頂、置底、均分)



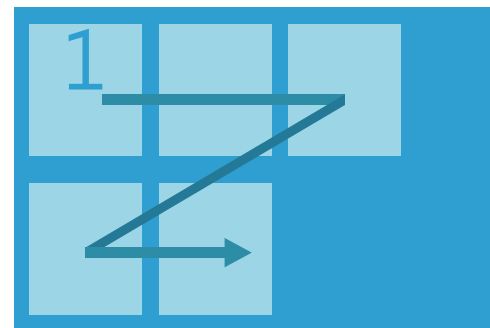
行與行之間保持相等距離

space-around(均分間距至上下)



每行的周圍保持相等距離

stretch(平均填滿)-預設



每一行都被拉伸以填滿容器

子層Flex item(項目元素) align-self(單一子元素對齊方式) | order(子元素排列順序)

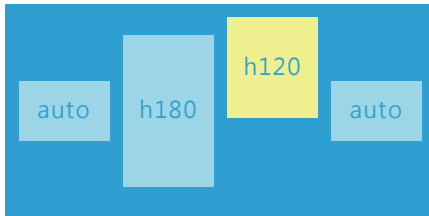
以flex主軸水平方向(row)為例

align-self: /\*單一子元素對齊方式\*/

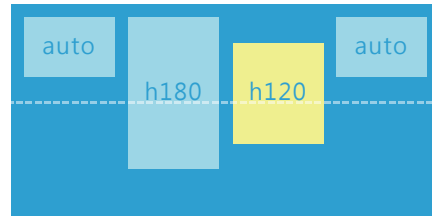
auto | flex-start | flex-end | center | baseline | stretch;

單獨的子元素設置 align-self 行為，會覆蓋父元素 align-items  
單一元素排列方式

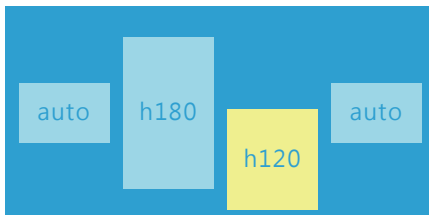
align-self: flex-start ;  
(父 align-items:center)



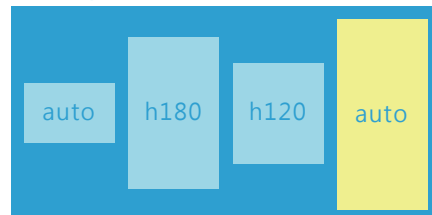
align-self: baseline ;(基線對齊)  
(父 align-items:flex-start)



align-self: flex-end ;  
(父 align-items:center)

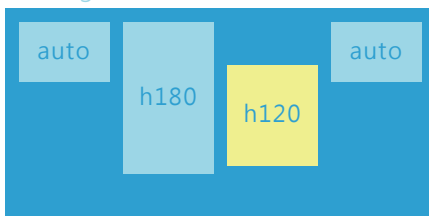


align-self: stretch ;(拉伸填充)  
(父 align-items:center)

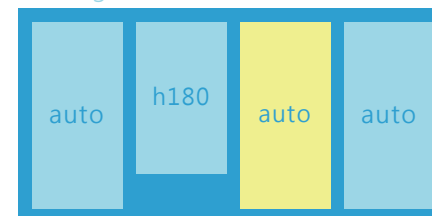


需搭配height:auto; | 仍然尊重最小寬度/最大寬度

align-self: center ;  
(父 align-items:flex-start)



align-self: auto ; (預設)  
(父 align-items:flex-start)

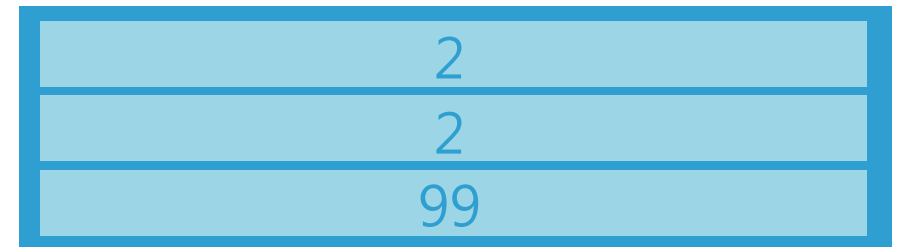


繼承其父容器的 align-items 屬性

order: /\*子元素排列順序\*/

<整數> /\* 預設值為 '0' \*/

控制單一子層項目在父層(柔性)容器中的順序



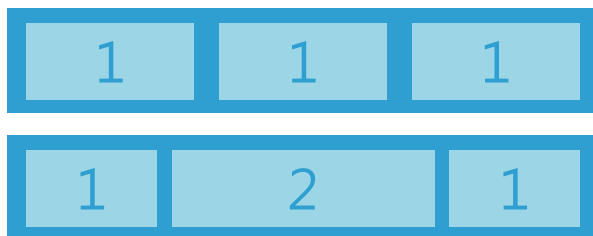
## 子層Flex item(項目元素) flex-grow(伸展比) | flex-shrink(收縮比) | flex-basis(基本比)

flex-grow: /\*伸展比\*/

<數字>(負數無效) /\* 預設值為 '0' \*/

子元素伸展至父層(柔性)容器中可用空間

單一元素空間分配方式:拉伸比/伸展比



當子元素的 flex-basis 長度「小」於它自己在父元素分配到的長度

例：

若總寬度是550px-(每個div100px\*3)=剩下250px。

第2個div:1等分，第3個div:3等分，總共四等分。

故  $250/4 = 62.5\text{px}$ 。

第2個佔1等分， $62.5 + \text{原本的}100 = 162.5\text{px}$

第3個佔3等分， $62.5*3=187.5 + \text{原本的}100 = 287.5\text{px}$

flex-shrink: /\*收縮比\*/

<數字>(負數無效) /\* 預設值為 '1' \*/

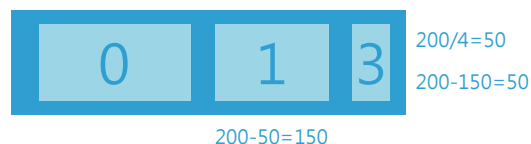
單一元素”必須設定尺寸”，當子元素”空間不足”分配時，才需要收縮比例，依數字比例收縮。

flex-shrink: 0; 則為不收縮，預設為”1”。



3個div比例為200px\*3。

當總寬度400px-(200\*3)=還差-200px



若第1個不佔等分,第2個div:1等分，第3個div:3等分

$0+1+3=4$ 等分

則不足的200px,分成4等分( $200/4$ ),每一等分為50px。

1等分=50px。

第2個div收縮 $1=200-50=150\text{px}$ 。

第3個div收縮 $3=200-150=50\text{px}$ 。

flex-basis: /\*基本比\*/

<尺寸> | <auto> /\* 預設值為 ' auto' \*/

子元素的初始長度以自己的基本大小為單位

視主軸方向判斷基本比為「寬度」或「高度」

\*權重比重：

min-/max- > flex-basis > width/height

flex: < 'flex-grow' > | < 'flex-shrink' > | < 'flex-basis' >  
flex: 0 0 200px;

flex: 1; (單一數字為定義伸展比)

# CSS POSITION

---

## CSS 定位元素

# CSS POSITION\ CSS定位元素

## CSS

```
body{
  margin: 0;
}
div{
  opacity: .75;
  height: 300px;
  width: 300px;
}
```

## HTML

```
<div id="box1">
  <div id="box-
2"></div>
</div>
<div id="box3"></div>
```

## position: static;(預設值)

「不特別定位」，會依HTML排版配置呈現，就算指定座標位置也不會有作用。

畫面在捲動時，元素還是會隨著頁面捲動。

## position: relative;

沒有設定屬性時呈現方式和 static 一樣。

設定座標位置後，會依照指定座標定位。

畫面在捲動時，元素還是會隨著頁面捲動。

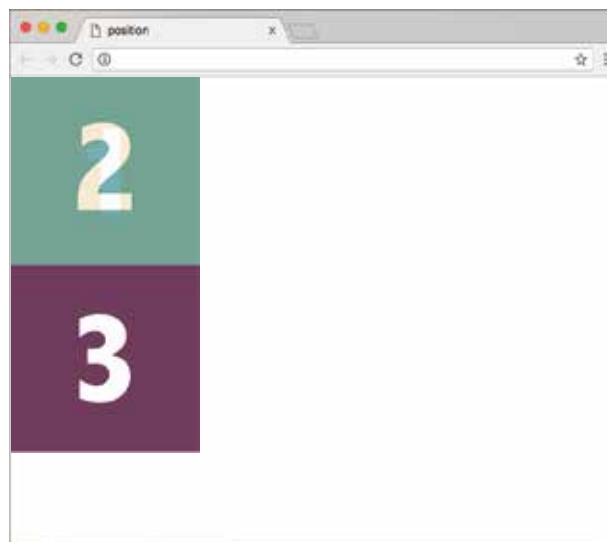
#box1沒有放在任何元素裡面，因此其定位會以 <body>元素為座標起點來定位。

#box2放在#box1裡面，即#box1為父元素，#box2為子元素，#box2會以#box1作為定位目標來呈現位置。

#box3為獨立元素，因relative不能排列，因此會#box1向下推壓，且並不會依照設定的定位去排列，而產生錯誤排版。

## static (不特別定位)

```
#box1 {
  background: #F2B035 url(1.svg);
  position: static;
  left: 30px;
  top: 30px;
}
#box2 {
  background: #15959F url(2.svg);
  position: static;
  left: 40px;
  top: 40px;
}
#box3 {
  background: #540032 url(3.svg);
  position: static;
  left: 50px;
  top: 50px;
}
```



## relative (相對定位)

```
#box1 {
  background: #F2B035 url(1 .svg);
  position: relative;
  left: 30px;
  top: 30px;
}
#box2 {
  background: #15959F url(2.svg);
  position: relative;
  left: 40px;
  top: 40px;
}
#box3 {
  background: #540032 url(3.svg);
  position: relative;
  left: 50px;
  top: 50px;
}
```



# CSS POSITION\ CSS定位元素

## position: absolute;

absolute為浮動元素，可重疊排版，類似photoshop的圖層。  
。畫面在捲動時，元素還是會隨著頁面捲動。

#box1沒有放在任何元素裡面，因此其定位會以<body>元素為座標起點來定位。

#box2放在#box1裡面，即#box1為父元素，#box2為子元素。

#box3為獨立元素，該元素的定位也是以 <body> 元素為座標起點來定位。

「子元素」指定為依據「父元素」位置中的絕對位置，則「子元素」可設定「position:absolute」(絕對位置)，但「父元素」必須設定為「static」以外的「position」定位值。

## position: fixed;

fixed元素其定位會以瀏覽器(DOM)為座標起點來定位。其不會跟著頁面捲動，而會固定指定的位置。

固定定位元素不會保留它原本在頁面應有的空間，不會跟其他元素的配置互相干擾。

z-index: 3;

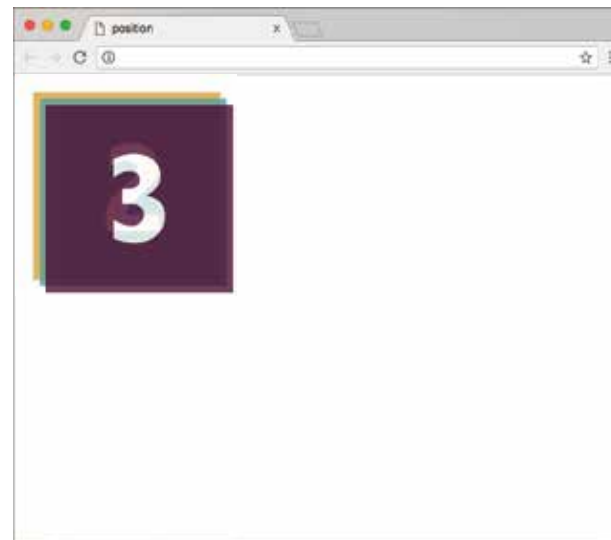
## absolute (絕對定位)

```
#box1 {  
  background: #F2B035 url(1.svg);  
  position: absolute;  
  left: 30px;  
  top: 30px;  
  z-index: 1;  
}  
#box2 {  
  background: #15959F url(2.svg);  
  position: absolute;  
  left: 40px;  
  top: 40px;  
  z-index: 2;  
}  
#box3 {  
  background: #540032 url(3.svg);  
  position: absolute;  
  left: 50px;  
  top: 50px;  
  z-index: 3;  
}
```



## fixed(固定定位)

```
#box1 {  
  background: #F2B035 url(1.svg);  
  position: fixed;  
  left: 30px;  
  top: 30px;  
}  
#box2 {  
  background: #15959F url(2.svg);  
  position: fixed;  
  left: 40px;  
  top: 40px;  
}  
#box3 {  
  background: #540032 url(3.svg);  
  position: fixed;  
  left: 50px;  
  top: 50px;  
}
```



# CSS POSITION\ CSS定位元素

position: sticky;

sticky根據用戶的滾動位置定位元素。

它將相對定位，直到在視窗中遇到設定的偏移位置為止，然後將其“黏貼”在指定的適當位置。



navbar

## What is Lorem Ipsum?

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

## Why do we use it?

sticky(黏貼定位)

```
#box-nav {
  position: sticky;
  left: 0;
  top: 0;
}
```

navbar

## What is Lorem Ipsum?

Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset sheets containing Lorem Ipsum passages, and more recently with desktop publishing software like Aldus PageMaker including versions of Lorem Ipsum.

## Why do we use it?

It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed to using 'Content here, content here', making it look like readable English. Many desktop publishing packages and web page editors now use Lorem Ipsum as their default model text, and a search for 'lorem ipsum' will uncover many web sites still in their infancy. Various versions have evolved over the years, sometimes by accident, sometimes on purpose (injected humour and the like).

## Where does it come from?

Contrary to popular belief, Lorem Ipsum is not simply random text. It has roots in a piece of classical Latin literature from 45 BC, making it over 2000 years old. Richard McClintock, a Latin professor at Hampden-Sydney College in Virginia, looked up one of the more obscure Latin words, consectetur, from a Lorem Ipsum passage, and going through the cites of the word in classical literature, discovered the undoubtable source. Lorem Ipsum comes from sections 1.10.32 and 1.10.33 of "de Finibus Bonorum et Malorum" (The Extremes of Good and Evil) by Cicero, written in 45 BC. This book is a treatise on the theory of ethics, very popular during



END.