

รหัสวิชา: ENGSE611

หน่วยกิต: 3 (1-4-4)

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา



การพัฒนาเว็บด้วยเทคโนโลยีสมัยใหม่

MODERN WEB TECHNOLOGY DEVELOPMENT

CSS3 Styling และ Layout เป็นองตัวบัน

ประเภทวิชา: กลุ่มวิชาชีพเลือก

อาจารย์ผู้สอน

อาจารย์นริศ กำแพงแก้ว



E-mail : naris@rmutl.ac.th



Tel : 091-7915355



รหัสวิชา: ENGSE611

การพัฒนาเว็บด้วยเทคโนโลยีสมัยใหม่

หัวข้อ :

CSS3 Styling และ Layout เบื้องต้น

Basic CSS3 Styling and Layout

วัตถุประสงค์การเรียนรู้ (Learning Objectives)

1. เข้าใจพื้นฐานการกำหนดสไตล์ด้วย CSS3
2. เขียนโค้ด CSS เพื่อจัดรูปแบบเว็บเพจได้อย่างถูกต้อง
3. จัดวางองค์ประกอบของหน้าจอด้วย Layout เบื้องต้น
4. เชื่อมโยง CSS เข้ากับงานออกแบบหน้าจอจริง (UI Layout)
5. สร้างหน้าเว็บให้มีความสวยงามอย่างมืออาชีพมากขึ้น



CSS คืออะไร

Cascading Style Sheets (CSS) ใช้จัดรูปแบบเค้าโครงของเว็บไซต์

เป็นภาษาสำหรับกำหนดรูปแบบการแสดงผลของหน้าเว็บ ไม่ว่าจะเป็นสี ขนาดตัวอักษร การจัดตำแหน่ง ระยะห่าง ไปจนถึงโครงสร้างของ Layout ก็ง่ายดายของเว็บไซต์

ถ้า HTML คือ “โครงสร้าง”

CSS ก็คือ “การตกแต่งและจัดรูปแบบ” ให้เว็บสวยงาน เมื่อൺการแต่งบ้านให้ดูดีน่าอยู่

CSS จึงเข้ามาช่วยในหลายด้าน เช่น

- กำหนดสีพื้นหลัง สีตัวอักษร ให้เว็บดูเป็นแบรนด์มากขึ้น
- จัดวาง Layout ให้สวยงาม เช่น แบ่งคอลัมน์ กล่องข้อความ เมบูด้านข้าง
- ปรับรูปแบบตัวอักษร เช่น font-size, font-family
- สร้างเอฟเฟกต์ เช่น Hover, Animation
- ปรับให้เว็บรองรับมือถือ (Responsive Design)



*body {
font:
background
color:
margin:
padding:*

ทำไมต้องใช้ CSS?

แยก “โครงสร้าง”
ออกจาก “การตกแต่ง”
ทำให้โค้ดอ่านง่าย
ปรับแต่งได้เร็ว และ
ดูเป็นระเบียบ

ส่งเสริมการใช้โค้ดซ้ำ
(Reusability)
ไฟล์ CSS หนึ่งไฟล์
สามารถเชื่อมโยง
กับหลายหน้าเว็บได้

สร้างความสวยงามและ
ความเป็นมืออาชีพให้กับ
เว็บไซต์

CSS ช่วยให้เว็บไซต์มี
ความสวยงาม มีรูป^{แบบที่ชัดเจน และดูเป็น}
^{ระบบมากขึ้น}



รองรับการแสดงผล
แบบ Responsive

CSS มีเครื่องมือ^{สำคัญ} ช่วยให้เว็บไซต์
สามารถปรับการแสดง^{ผลได้อย่างเหมาะสม}



สร้างเอกลักษณ์ของ
แบรนด์และยกระดับ
ประสบการณ์ผู้ใช้

ช่วยให้เว็บไซต์สะท้อน^{ภาพลักษณ์ขององค์กร}
มีความสอดคล้อง และ^{ส่งเสริมประสบการณ์}
การใช้งานที่ดีแก่ผู้ใช้

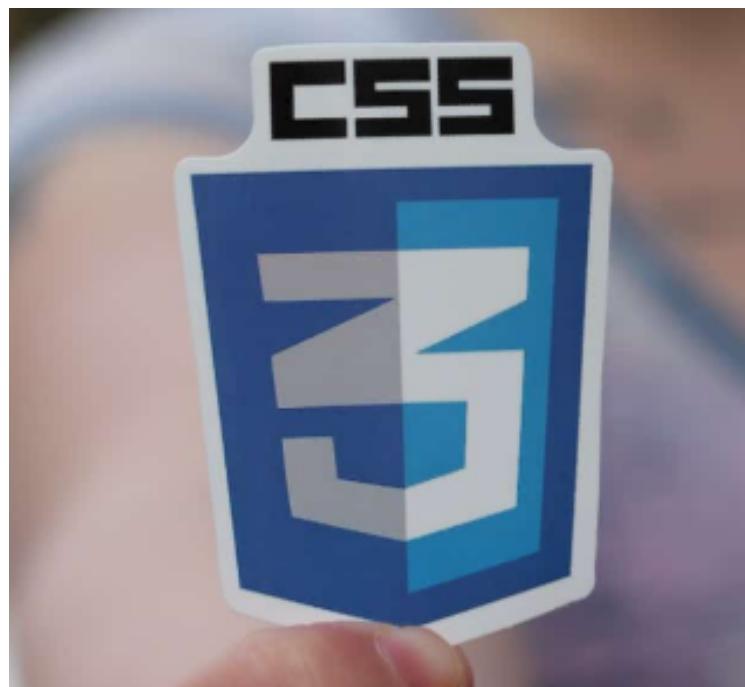


Version ของ CSS (CSS Versions)

CSS (Cascading Style Sheets) ได้มีการพัฒนามาอย่างต่อเนื่องเพื่อรองรับความต้องการของเว็บยุคใหม่ ทั้งด้านการออกแบบ การจัดวางโครงสร้าง และการสร้างประสบการณ์ใช้งานที่ลึกซึ้ง โดยเวอร์ชันหลักของ CSS มีดังนี้

CSS 1 หรือ CSS Level 1

- เปิดตัวครั้งแรกในปี 1996
- รองรับคุณสมบัติพื้นฐาน เช่น สี ตัวอักษร ระยะห่าง เส้นขอบ
- เป็นจุดเริ่มต้นของแนวคิดการแยกโครงสร้าง HTML ออกจาก การตกแต่ง



CSS2 (1998) พัฒนาการด้านการจัดวาง

- เพิ่มความสามารถสำคัญ เช่น
 - Positioning (absolute, relative)
 - Media types
 - Z-index
- รองรับการจัดเลyi เว็บที่ซับซ้อนมากขึ้น

CSS2.1 (2011) มาตรฐานที่มีเสถียรภาพมากขึ้น

- ปรับปรุงจาก CSS2
- แก้ไขคุณสมบัติที่ไม่เสถียรและกำหนดกฎ การแสดงผลให้ชัดเจน
- ถูกใช้เป็นพื้นฐานให้เบราว์เซอร์รองรับอย่าง แพร่หลาย

CSS3 (2011–ปัจจุบัน) ระบบโมดูล (Modular CSS)

CSS3 ไม่ได้เป็นเวอร์ชันแบบครบชุด แต่ทำงานแบบ “โมดูล” (Modules) เช่น

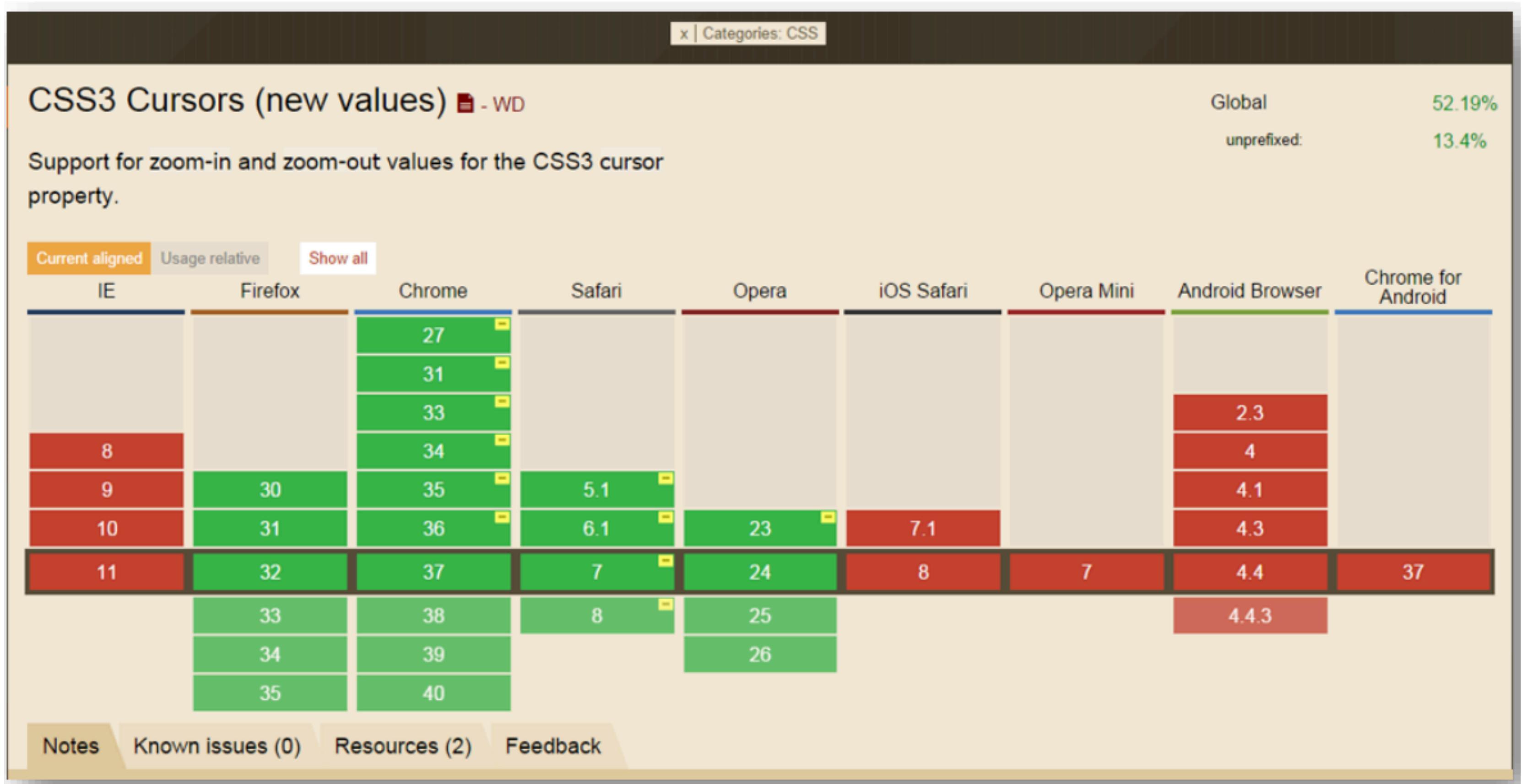
- Selectors Level 3
- Backgrounds & Borders
- Flexbox
- Grid Layout
- Transforms, Transitions, Animations
- Media Queries

ข้อดีของระบบโมดูล:

- พัฒนาแยกเป็นส่วน ๆ ทำให้เบราว์เซอร์รองรับได้เร็ว ขึ้น
- สร้างความสามารถใหม่ ๆ เช่น Responsive Layout, Animation, Shadow, Gradient

CSS3 คือเวอร์ชันที่ใช้กันในอุตสาหกรรมปัจจุบัน

http://caniuse.com/#cats=CSS



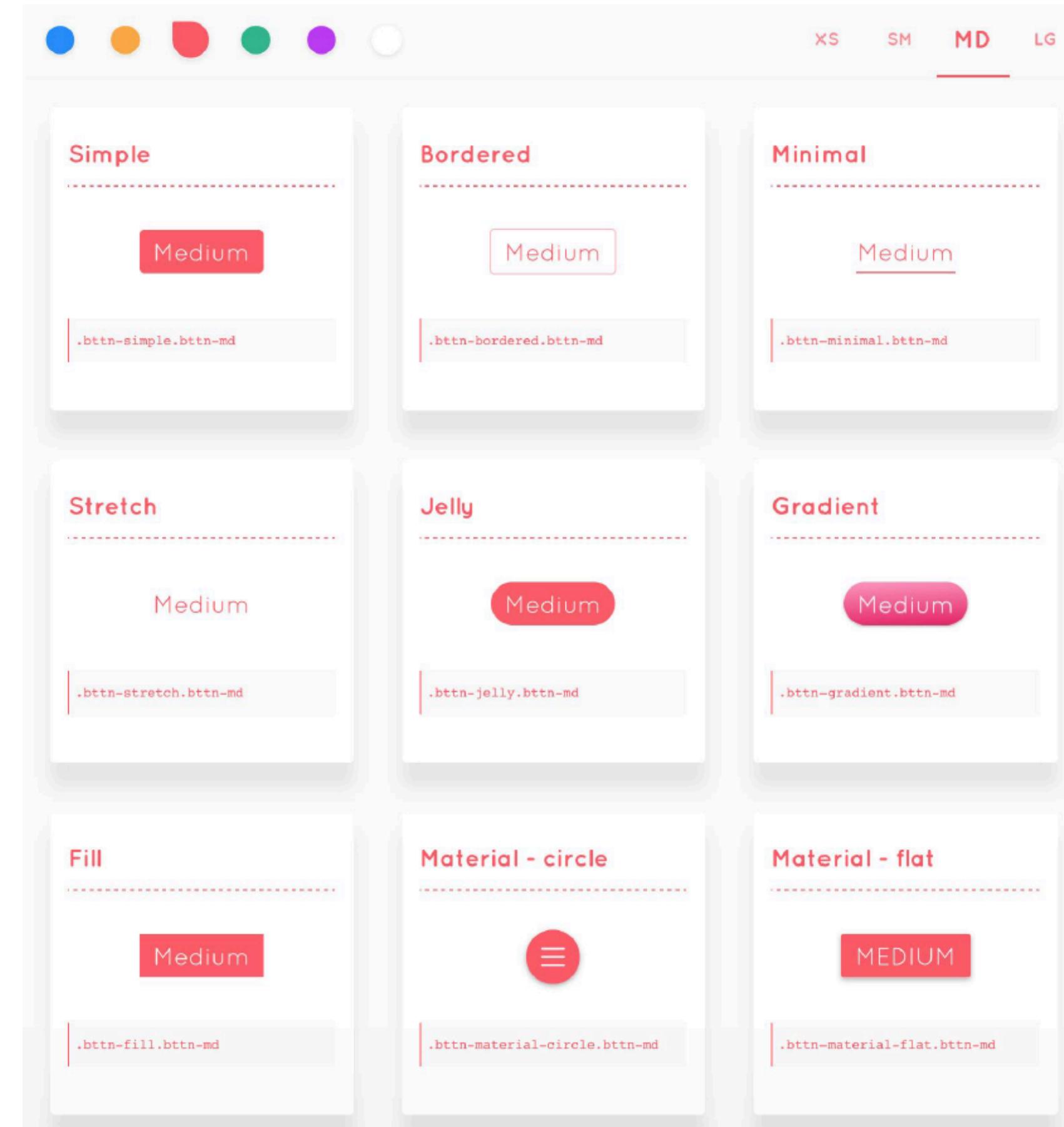
CSS คือเครื่องมือกำหนดการแสดงผลของ HTML

CSS (Cascading Style Sheets) คือภาษาที่ใช้ในการกำหนดรูปแบบการนำเสนอ (presentation) หรือการแสดงผลของเอกสารที่ถูกจัดเก็บอยู่ในไฟล์ HTML

CSS ช่วยให้เราควบคุมองค์ประกอบที่เกี่ยวข้องกับการ呈現 ของหน้าเว็บ และยังสามารถเพิ่มเติมคุณลักษณะอื่น ๆ ให้กับองค์ประกอบต่าง ๆ ได้อย่างยืดหยุ่น ตัวอย่างของสิ่งที่ CSS ควบคุมได้ มีดังนี้:

คุณสมบัติที่สามารถควบคุมด้วย CSS

- Colors (สี):** สีของข้อความ สีพื้นหลัง และสีของเส้นขอบ
- Margins (ระยะขอบ):** ระยะห่างภายใน (interior margin) และ ระยะห่างภายนอก (exterior margin)
- Position (ตำแหน่ง):** การกำหนดว่าจะวางองค์ประกอบไว้ตรงไหน
- Sizes (ขนาด):** ความกว้าง (width) และความสูง (height)
- Behaviour (พฤติกรรม):** การตอบสนองเมื่อมาส์ชีหรือมีการโต้ตอบอื่น ๆ



CSS example (ตัวอย่าง CSS)

```
* {  
    color: blue; /*a comment */  
    margin: 10px;  
    font: 14px Tahoma;  
}
```

กฎ CSS นี้จะปรับสไตล์ขององค์ประกอบทั้งหมดบนหน้าเว็บให้เป็นดังนี้:

- สีตัวอักษร (color): เป็นสีน้ำเงิน (blue)
- ระยะห่างรอบองค์ประกอบ (margin): กว้าง 10 พิกเซล
- รูปแบบตัวอักษร (font): ใช้ขนาด 14px และฟอนต์ Tahoma
- กล่าวคือ ทุกแท็ก HTML ในหน้าเว็บจะมีลักษณะตามนี้โดยอัตโนมัติ



CSS Fields (คุณสมบัติพื้นฐานของ CSS)

รายการของคุณสมบัติ (Properties) ที่ใช้บ่อยที่สุดใน CSS พร้อมตัวอย่างการใช้งาน:

- color: #FF0000; red; rgba(255,0,0,1.0); //วิธีการระบุสีเมื่อหลายรูปแบบ เช่น รหัส Hex, ชื่อสี, หรือค่า RGBA
- background-color: red; //→ กำหนดสีพื้นหลังขององค์ประกอบ (Element)
- background-image: url('file.png'); //กำหนดภาพพื้นหลัง
- font: 18px 'Tahoma'; //กำหนดลักษณะตัวอักษร เช่น ขนาด และฟอนต์ที่ใช้
- border: 2px solid black; //กำหนดเส้นขอบทุกด้านขององค์ประกอบ
- border-top: 2px solid red; //กำหนดเส้นขอบเฉพาะด้านบน
- border-radius: 2px; //กำหนดให้โค้งมน (Rounded corners)
- margin: 10px; //กำหนดระยะห่างระหว่างองค์ประกอบกับภายนอก (Outer spacing)
- padding: 2px; //กำหนดระยะห่างระหว่างเส้นขอบกับเนื้อหาภายใน (Inner spacing)
- width: 100%; 300px; 1.3em; //หน่วยของความกว้างมีหลายแบบ เช่น %, px, em เป็นต้น
- height: 200px; //กำหนดความสูงขององค์ประกอบ
- text-align: center; //จัดตำแหน่งข้อความ (ซ้าย/กลาง/ขวา)
- box-shadow: 3px 3px 5px black; //กำหนดเงาของกล่อง (Drop shadow)
- cursor: pointer; //กำหนดรูปแบบเมาส์เมื่อชี้ (เช่นรูปมือสำหรับลิงก์/ปุ่ม)
- display: inline-block; //กำหนดรูปแบบการแสดงผลขององค์ประกอบ เช่น block, inline-block
- overflow: hidden; //ซ่อนเนื้อหาที่ล้ำออกจากพื้นที่ขององค์ประกอบ

การเพิ่ม CSS ให้กับเว็บไซต์

โดยทั่วไปมีกั้งหมด 4 วิธีในการนำกฎของ CSS (CSS rules) มาใช้งานบนเว็บไซต์ ดังนี้:

1) Internal CSS

การเขียนโค้ด CSS ภายในแท็ก `<style>`

ใช้เมื่อ ต้องการกำหนดสไตล์เฉพาะหน้าเว็บเพียงหน้าเดียว

```
<style>
  p { color: blue; }
</style>
```

2) การเชื่อมไฟล์ CSS ภายนอก (External CSS File)

เป็นวิธีที่นิยมที่สุด เพราะแยกโครงสร้าง HTML ออกจากความต้องการ

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

ดีที่สุด! เพราะจัดการง่าย, นำกลับมาใช้ได้, และทำให้โค้ดสะอาด

3) การกำหนดสไตล์ผ่าน Attribute style

ภายในแท็ก HTML (Inline CSS)

เหมาะสมสำหรับการปรับแต่งเฉพาะจุด หรือทดลองสไตล์อย่างรวดเร็ว

```
<p style="color: blue; margin: 10px;">
```

4) การใช้ JavaScript เพื่อเพิ่มหรือเปลี่ยน CSS

เหมาะสมสำหรับการเปลี่ยนสไตล์แบบไดนามิกตามเหตุการณ์ต่าง ๆ

External CSS - ວິທີທີ່ດີທີ່ສຸດ!

1. ສ້າງໄຟ້ style.css

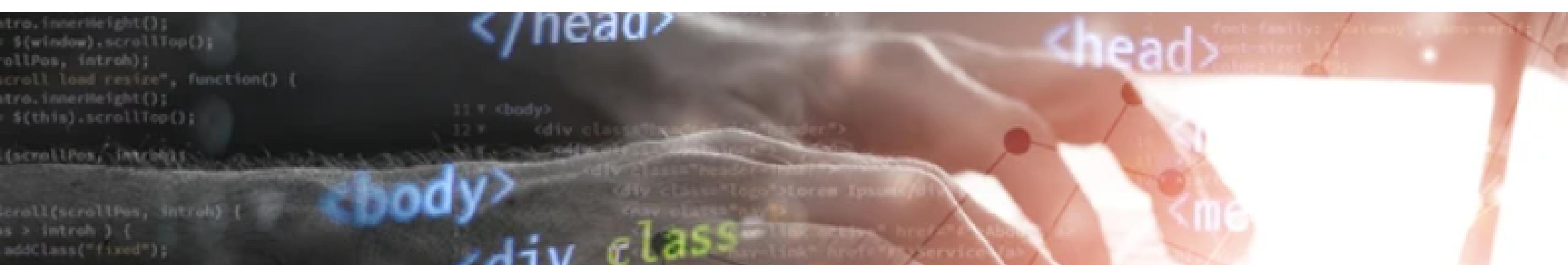
ສ້າງໄຟ້ໃໝ່ແລະບັນທຶກດ້ວຍນາມສກູລ .css

```
/* ບັນທຶກ style.css */
h1 {
    color: blue;
    font-size: 30px;
}
```

2. ເຊື່ອມຕ່ອນ HTML

ໃນໄຟ້ index.html ເພີ່ມ <link> ໃນ <head>

```
<head>
    <title>My Website</title>
    <link rel="stylesheet" href="style.css">
</head>
```



CSS Syntax พื้นฐาน

```
selector {  
    property: value;  
}
```

h1 { color: blue; }
↑ Selector ↑ Property ↑ Value

- Selector = ตัวเลือก (เช่น h1, p, .card)
- Property = คุณสมบัติ (เช่น color, font-size)
- Value = ค่า (เช่น blue, 16px)

CSS Selectors พื้นฐาน

1. Element Selector

```
p { color: green; }
```

2. Class Selector (.)

```
.highlight { background: yellow; }
```

3. ID Selector (#)

```
#header { background: blue; }
```

เราบีຍນໃใช Class Selector ມາກທີ່ສຸດໃນການ
ກຳຈານຈົງ ເພຣະມີຄວາມຢັດຫຍຸນສູງ

ความแตกต่าง Class vs ID

Class (.) - ใช้ซ้ำได้

เหมือน "ป้ายชื่อกลุ่ม" ใช้สำหรับจัดส太子ให้ element หลายๆ ตัวที่หน้าตาเหมือนกัน

```
<button class="btn-primary">OK</button>
<button class="btn-primary">Submit</button>
```

กฎทอง: Class = หลายคน, ID = คนเดียว

ID (#) - ห้ามซ้ำ

เหมือน "เลขบัตรประชาชน" ในหนึ่งหน้าจะใช้ชื่อ ID นี้ได้แค่ครั้งเดียวเท่านั้น

```
<div id="main-navigation"> ... </div>
```



คำamuraดสอบความเข้าใจ

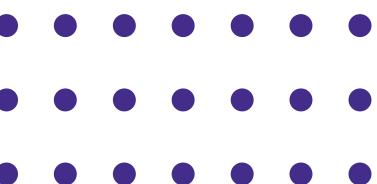
ถ้าคุณต้องการให้ข้อความทุกย่อหน้า (<p>) มีสีน้ำเงิน คุณจะเขียน CSS อย่างไร?

A) p { color: blue; }

B) .p { color: blue; }

C) #p { color: blue; }

D) <p style="color: blue;">



คำamuraดสอบความเข้าใจ

ถ้าคุณต้องการให้ข้อความทุกอย่างหน้า (<p>) มีสีน้ำเงิน คุณจะเขียน CSS อย่างไร?

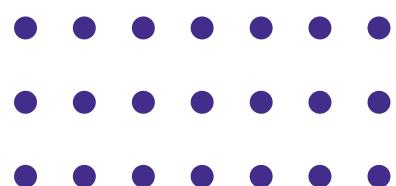


A) p { color: blue; }

เราต้องการเลือกทุก <p> Tag ดังนั้นจึงใช้ Element Selector p



- B) .p หมายถึง Element ที่มี class="p" ซึ่งไม่ใช่สิ่งที่เราต้องการ
- C) #p หมายถึง Element ที่มี id="p" ซึ่งไม่ใช่สิ่งที่เราต้องการ
- D) ตัวเลือก D เป็น Inline Style ซึ่งจะเปลี่ยนสีแค่ p Tag เดียว ไม่ใช่ทั้งหมด



ตัวอย่าง CSS Selectors พื้นฐาน

ทดลองสร้างไฟล์ `selectors.html`

```
<style>
/* 1. Element Selector */
p {
    color: green;
}

/* 2. Class Selector */
.highlight {
    background: yellow;
    padding: 5px;
}

/* 3. ID Selector */
#header {
    background: blue;
    color: white;
    padding: 10px;
}
</style>
```

```
<body>
    <h1>ตัวอย่าง CSS Selectors</h1>
    <!-- Element Selector --&gt;
    &lt;p&gt;
        ข้อความนี้ถูกจัดสีด้วย Element Selector (p { color: green; })
    &lt;/p&gt;
    <!-- Class Selector --&gt;
    &lt;p class="highlight"&gt;
        ข้อความนี้ถูกจัดสีโดย Class Selector (.highlight)
    &lt;/p&gt;
    <!-- ID Selector --&gt;
    &lt;div id="header"&gt;
        นี่คือพื้นที่ Header (ID Selector)
    &lt;/div&gt;
&lt;/body&gt;</pre>
```

CSS Selectors เพิ่มเติม (ตัวอย่าง)

เลือกตามสถานะ

(Pseudo-class – ใช้เครื่องหมาย :)

กำหนดสไตล์ตามพฤติกรรมของผู้ใช้ เช่น การวางเมาส์

```
p:hover { ... }
```

มีผลเมื่อผู้ใช้เอาเมาส์ไปวางบนแท็ป <p>

```
/* 1) Pseudo-class :hover */
p:hover {
    color: red;
    background-color: #ffe5e5;
    cursor: pointer;
}
```

```
<h2>Pseudo-class :hover</h2>
<p>นำเมาส์มาวางบนข้อความนี้เพื่อดูผลลัพธ์</p>
```

เลือกจากแอตทริบิวต์

(Attribute Selector – เขียนในวงเล็บ [])

ใช้เลือกแท็ปที่มีแอตทริบิวต์เฉพาะ

```
input[type="text"] { ... }
```

มีผลกับ <input> ที่ type="text" เท่านั้น

```
/* 2) Attribute Selector [type="text"] */
input[type="text"] {
    border: 2px solid blue;
    padding: 8px;
    border-radius: 5px;
}

input[type="text"]:focus {
    border-color: rgb(255, 0, 128);
    outline: none;
    background-color: #fff0fa;
}
```

```
<h2>Attribute Selector [type="text"]</h2>
<input type="text" placeholder="พิมพ์ข้อความที่นี่">
```

CSS Fonts

ตัวอย่างผลลัพธ์

```
body {  
  font-family: serif;  
  font-style: normal;  
  font-weight: bold;  
  font-size: 20px; /* em unit */  
}
```

Difference Between Serif and Sans-serif Fonts



หมายเหตุ:

- px (พิกเซล): หน่วยคงที่ ไม่ยืดหยุ่น ไม่ตอบสนองต่อโครงสร้างแบบ parent-child
- em = หน่วยสัมพัทธ์ที่ขึ้นกับ font-size ของ parent → ควบคุมทั้งกลุ่มได้ในจุดเดียว
- การใช้ em ช่วยให้การอ่านแบบ Responsive และการจัดการ Typography มีประสิทธิภาพสูงขึ้น เพราะควบคุมทั้งระบบจากจุดเดียวได้

ความแตกต่างของฟอนต์แต่ละแบบ

Sans-serif

(font-family: Arial; style: italic; weight: 400; size: 20px; color: black)

Serif

(font-family: Georgia; style: normal; weight: 600; size: 22px; color: black)

Serif (Red)

(font-family: Georgia; style: oblique; weight: 600; size: 22px; color: red)

Font-size: Parent → Child (em)

```
<style>

    /* Parent ที่ควบคุมฟอนต์ทั้งหมด */
    .parent {
        font-size: 16px; /* ค่าเริ่มต้น */
        border: 2px solid #ccc;
        padding: 20px;
        margin-top: 20px;
    }

    /* Child ใช้ em → เปลี่ยนตาม parent */
    .child-title {
        font-size: 2em; /* 2 × parent */
        color: #c0392b;
        font-weight: bold;
    }

    .child-text {
        font-size: 1.2em; /* 1.2 × parent */
        color: #2c3e50;
    }

    /* ตัวอย่างที่ใช้ px → ไม่ยืดหยุ่น */
    .fixed-text {
        font-size: 18px;
        color: #2980b9;
    }
</style>
```

ตัวอย่างผลลัพธ์

สาธิต Font-size: Parent → Child (em)

ใส่ขนาดฟอนต์ Parent (px): ปรับขนาดฟอนต์

หัวข้อหลัก (2em) = 32 px

ข้อความย่อ (1.2em) → เปลี่ยนตาม Parent = 19.2 px

ข้อความแบบ px (18px) → ไม่เปลี่ยน

- em = หน่วยสัมพัทธ์ที่ขึ้นกับ font-size ของ parent → ควบคุมทั้งกลุ่มได้ในจุดเดียว
- การใช้ em ช่วยให้การออกแบบ Responsive และการจัดการ Typography มีประสิทธิภาพสูงขึ้น เพราะควบคุมทั้งระบบจากจุดเดียวได้

- px (พิกเซล): หน่วยคงที่ ไม่ยืดหยุ่น ไม่ตอบสนองต่อโครงสร้างแบบ parent-child

CSS Text

Text Shadow = คุณสมบัติที่ใช้ใส่ “เงาให้ตัวอักษร”

Word-Wrap: break-word = บังคับข้อความตัดบรรทัดคำเกินกำหนด

```
<style>
    /* ตัวอย่าง text-shadow */
    h1 {
        font-size: 40px;
        text-shadow: 5px 5px 5px #FF0000;
        /* offset-x | offset-y | blur | color */
    }

    /* ตัวอย่าง word-wrap */
    p {
        width: 300px;                      /* บังคับกรอบข้อความ */
        border: 1px solid #ccc;
        padding: 10px;
        word-wrap: break-word;             /* ให้ตัดคำเมื่อยาวเกินพื้นที่ */
    }
</style>
```

ตัวอย่างผลลัพธ์

Text-shadow effect

This text is styled with some of the text formatting properties. The heading uses the `text-align`, `text-transform`, and `color` properties. The paragraph is indented, aligned, and the space between characters is specified. ThisIsAVeryLongWordThatWouldNormallyGoOutsideButBreakWordKeepsItInside.

Text Properties

ตัวอย่างผลลัพธ์

การจัดแนว:

```
text-align: left | center | right | justify;
```

ตัวอย่าง .align-left { text-align: left; }

การตกแต่งข้อความ:

```
text-decoration: underline | line-through | none;
```

ตัวอย่าง .underline { text-decoration: underline; }

ระยะห่างบรรทัด:

```
line-height: 1.5; /* 1.5 เท่าของฟอนต์ */  
line-height: 24px; /* ความสูงเฉพาะ */
```

ตัวอย่าง .line-height-normal{ line-height: 1.5; }
.line-height-normal{ line-height: 24px; }

Text Properties – Example

1. การจัดแนวข้อความ (text-align)

ข้อความนี้จัดซิดซ้าย (left)

ข้อความนี้จัดกึ่งกลาง (center)

ข้อความนี้จัดซิดขวา (right)

ข้อความนี้จัดเต็มบรรทัด (justify) เพื่อให้ชิดทั้งซ้ายและขวาโดยอัตโนมัติ

2. การตกแต่งข้อความ (text-decoration)

ข้อความนี้ขีดเส้นใต้ (underline)

ข้อความนี้ขีดผ่า (line-through)

ข้อความนี้ไม่มีการตกแต่ง (none)

3. การกำหนดระยะบรรทัด (line-height)

ระยะบรรทัด 1.5 เท่า – เหมาะสำหรับการอ่านแบบสวยงามทั่วไป

ระยะบรรทัด 24px – ควบคุมความสูงบรรทัดแบบคงที่

Background Properties

สีพื้นหลัง:

```
background-color: lightblue;
```

รูปพื้นหลัง:

```
background-image: url('image.jpg');  
background-size: cover;  
background-repeat: no-repeat;  
background-position: center;
```

ทึ้งหมดในบรรทัดเดียว:

```
background: url('bg.jpg') no-repeat center / cover;
```

Background Properties

สีพื้นหลัง:

```
<style>
.box-color {
    width: 300px;
    height: 120px;
    background-color: lightblue;
    padding: 20px;
}
</style>
```

รูปพื้นหลัง:

```
<style>
.box-image {
    width: 300px;
    height: 300px;
    background-image: url('image/rmutl.jpg');
    background-size: cover;          /* ปรับให้รูปเต็มกล่อง */
    background-repeat: no-repeat;    /* ไม่ต้องการให้รูปซ้ำ */
    background-position: center;    /* จัดรูปให้อยู่ตรงกลาง */
    border: 2px solid #333;
}
</style>
```

ทึ้งหมดในบรรทัดเดียว:

```
<style>
.box-short {
    width: 300px;
    height: 300px;
    background: url('image/rmutl.jpg') no-repeat center / cover;
    border: 2px solid black;
}
</style>
```

Background Properties

Color Name	HEX	Color
AliceBlue	#F0F8FF	
AntiqueWhite	#FAEBD7	
Aqua	#00FFFF	
Aquamarine	#7FFFAD	
Azure	#F0FFFF	
Beige	#F5F5DC	
Bisque	#FFE4C4	
Black	#000000	
BlanchedAlmond	#FFEB3B	
Blue	#0000FF	
BlueViolet	#8A2BE2	
Brown	#A52A2A	
BurlyWood	#DEB887	
CadetBlue	#5F9EA0	

ที่มา: https://www.w3schools.com/tags/ref_colornames.asp

Background Color

```
<style>
  .bg-yellow {
    background-color: yellow;
  }

  .bg-hex {
    background-color: #FFFF00;
  }
</style>
```

ตัวอย่างผลลัพธ์

Background-color set by using yellow

Background-color set by using #FFFF00

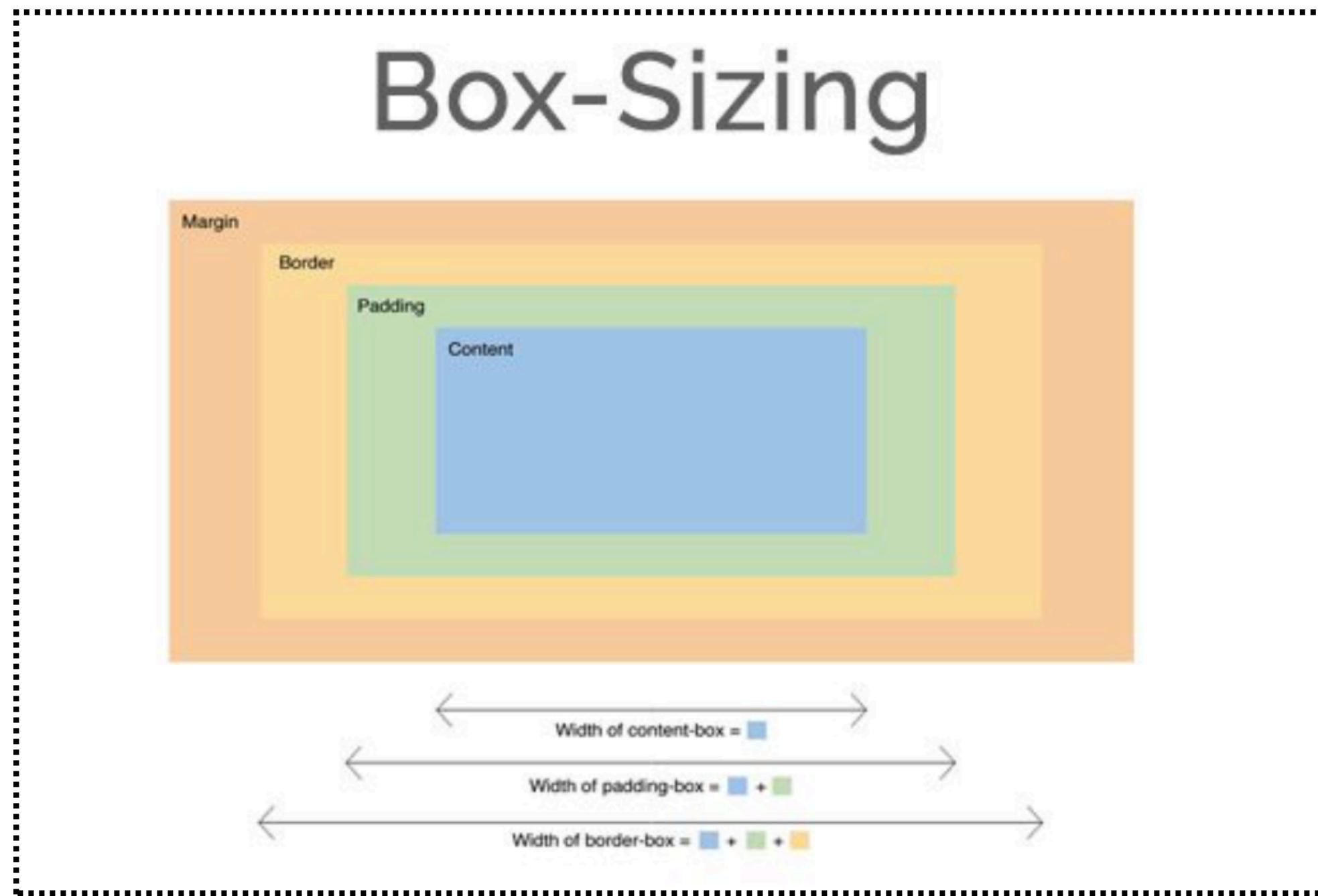
Demo Time! (15 นาที)

ตกแต่งเว็บของเรา กัน!

1. เปิดโปรเจกต์เดิม (index.html)
2. สร้างไฟล์ style.css และเชื่อมต่อไฟล์ใน <head>
3. ใช้ **Element Selector** (body, h1) เพื่อเปลี่ยนฟ้อนต์และสีพื้นหลังโดยรวม
4. สร้าง **Class Selector** (เช่น .highlight) เพื่อใช้แบ่งข้อความบางส่วนให้มีสีแตกต่าง
5. สร้าง **ID Selector** (เช่น #main-header) และกำหนดสไตล์ให้ส่วนหัวของเว็บ
6. ทดลองใช้ text-align, font-weight และ property อื่นๆ ที่เรียนมา

CSS Box Model

ทุก HTML Element บนหน้าเว็บคือ "กล่อง" สี่เหลี่ยม ซึ่งประกอบด้วย 4 ชั้น:



Content – เนื้อหาจริง เช่น ข้อความ / รูปภาพ
Padding – ระยะห่างระหว่าง content กับกรอบ
Border – เส้นขอบ
Margin – ระยะห่างภายนอกระหว่างองค์ประกอบอื่น ๆ

CSS Box Model

1) box-sizing: content-box (ค่าปกติ)

ความกว้างที่กำหนด = 200px
(ไม่รวม padding + border)

$$\begin{aligned}\text{total width} &= \text{content} \\ &+ \text{padding} + \text{border} \\ &= 200 + 20 + 20 + 5 + 5 \\ &= 250\text{px}\end{aligned}$$

2) box-sizing: border-box

ความกว้างที่กำหนด =
200px (รวม padding +
border อุปใน 200px)

$$\begin{aligned}\text{total width} &= 200\text{px} \\ (\text{รวม content} &+ \\ \text{padding} &+ \text{border} \\ \text{ภายในแล้ว})\end{aligned}$$

Padding $20\text{px} \times 2 = 40\text{px}$ มีทั้งบน-ล่าง หรือ ซ้าย-ขวา
Border $5\text{px} \times 2 = 10\text{px}$ มีทั้งบน-ล่าง หรือ ซ้าย-ขวา

CSS Border

1) Border (เส้นขอบของกล่อง HTML Element)

Border คือ เส้นขอบที่ล้อมรอบพื้นที่ของ box
(content + padding) ซึ่งกำหนดได้ 3 ค่าหลัก

```
border-width  
border-style  
border-color  
  
/* ตัวอย่าง border พื้นฐาน */  
.border-list {  
    border: 2px solid blue;  
}
```

2) Border-radius (ทำมนุกโค้งมน)

ใช้สำหรับ "ปัดมนุก" ของกล่องจากมนุกเหลี่ยม → มนุกโค้งมน

```
/* แบบ 1: โค้งทุกมนุกเท่ากัน */  
.radius-20 {  
    border-radius: 20px;  
}  
/* แบบ 2: วงรี (โค้งมาก) */  

```

1) โค้งทุกมนุกเท่ากัน

20px

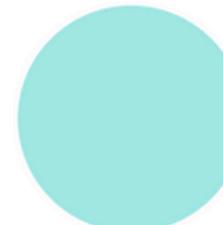
2) โค้งแบบวงรี

50px

3) โค้งเฉพาะด้านบน

Top only

4) วงกลม (Circle)



CSS Border

3) Box-shadow (การใส่เงา)

เพิ่มเงาให้กับกล่องเพื่อเพิ่มนิติ

```
.box-shadow-example {  
    box-shadow: 10px 10px 5px #888888;  
}
```

ความหมาย:

- 10px = ระยะเงาในแนวอน
 - 10px = ระยะเงาในแนวตั้ง
 - 5px = ความพุ่ง (blur)
 - #888888 = สีของเงา
-
- rgba(0,0,0,0.4) = สี + ความโปร่งใส

1) เงาธรรมดा

```
box-shadow: 10px 10px 5px  
rgba(0,0,0,0.4)
```

4) เงาด้านใน (Inset)

```
box-shadow: inset 0 0 15px  
rgba(0,0,0,0.4)
```

2) เงาพุ่งนุ่ม

```
box-shadow: 0 0 20px  
rgba(0,0,0,0.3)
```

5) เงาหลายชั้น

```
box-shadow: multi-layer
```

3) เงาทิศทางลงล่าง

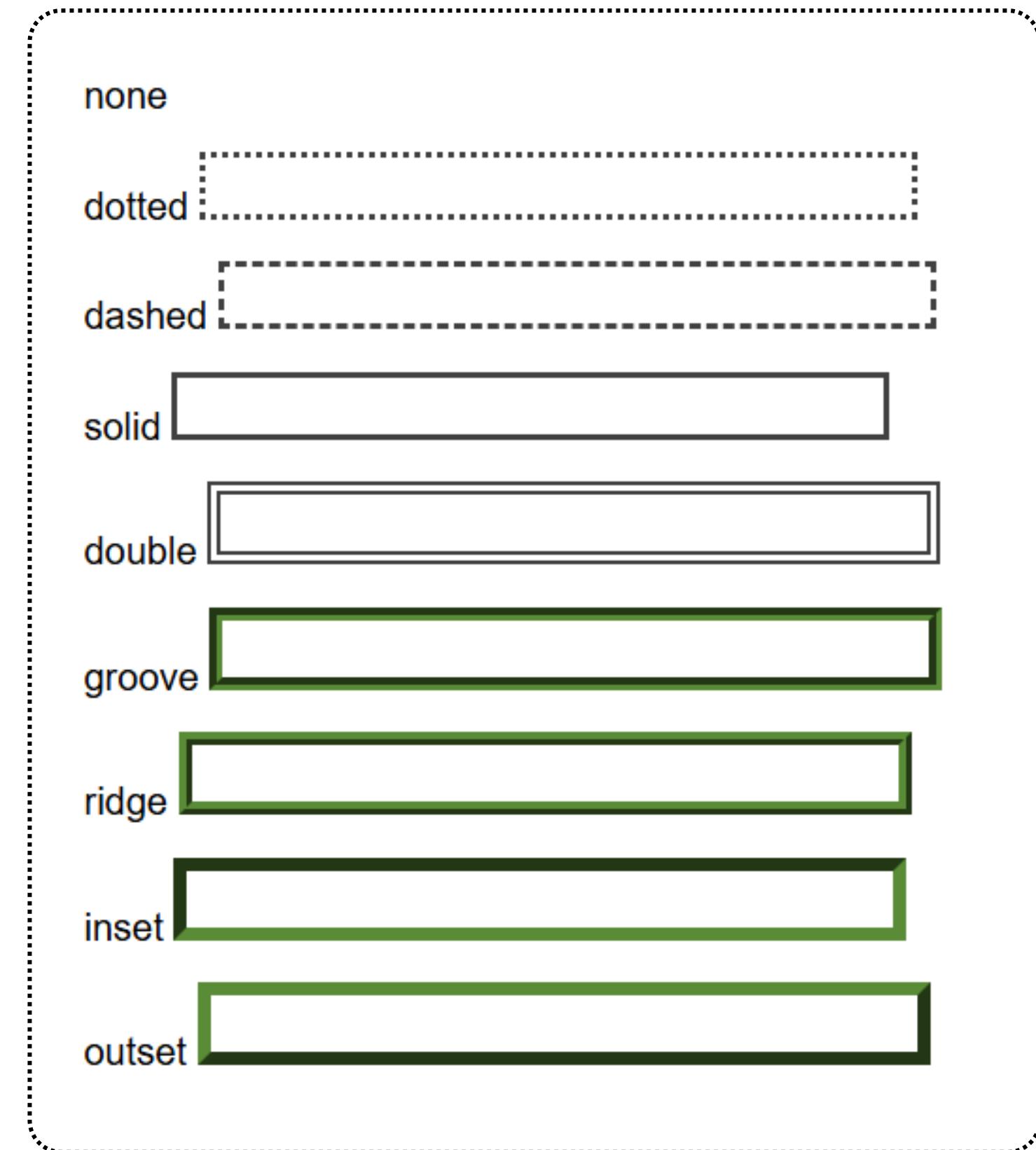
```
box-shadow: 0 12px 8px  
rgba(0,0,0,0.25)
```

CSS Border

4) Border-style ประเภทต่าง ๆ

CSS กำหนดชนิดของเส้นขอบได้หลากหลาย

Border Style	คำอธิบาย
dotted	จุด ๆ เรียงกัน
dashed	เส้นประ
solid	เส้นทึบ
double	เส้นสองชั้น
groove	เส้นแบบร่อง 3D
ridge	เส้นบุบ 3D
inset	ดูเหมือนกดลงจากพื้น
outset	ดูเหมือนยกออกจากพื้น



Display Properties

เป็นคุณสมบัติที่กำหนด “ลักษณะการแสดงผล” ของ HTML element ว่าจะวางตัวอย่างไรบนหน้าเว็บ เช่น

- วางแบบเต็มบรรทัด
- วางแบบต่อเนื่องในบรรทัด
- แสดง/ไม่แสดงผล

วางแบบเต็มบรรทัดและต่อเนื่องในบรรทัด

1. Block-level Elements

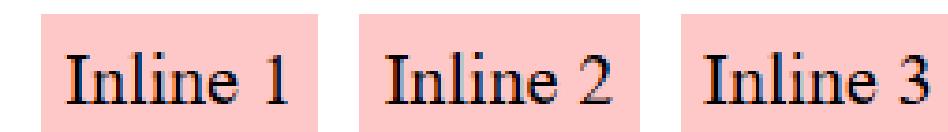
- คุณสมบัติของ block element
- กินพื้นที่ “เต็มความกว้างของคอนเทนเนอร์”
- บังคับให้ขึ้นบรรทัดใหม่เสมอ
- กำหนด width / height ได้



<div> = block เพราะ HTML นิยามมาเพื่อ
จัดโครงสร้างใหญ่

2. Inline Elements

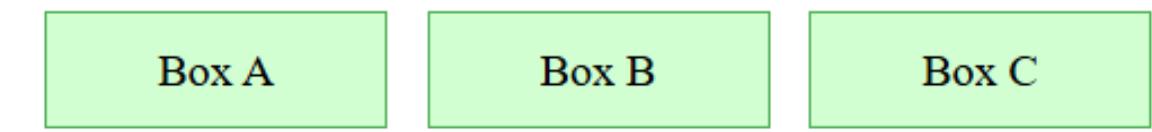
- วางต่อในบรรทัดเดียวกัน
- ไม่กินพื้นที่เต็มบรรทัด
- ไม่สามารถกำหนด width / height ได้โดยตรง
- ใช้เพื่อเน้นข้อความหรือเพื่อวางชิ้นส่วนเล็ก ๆ ภายในประโยค



 = inline เพราะสร้างมาเพื่อ
จัดรูปแบบเล็ก ๆ ในบรรทัด

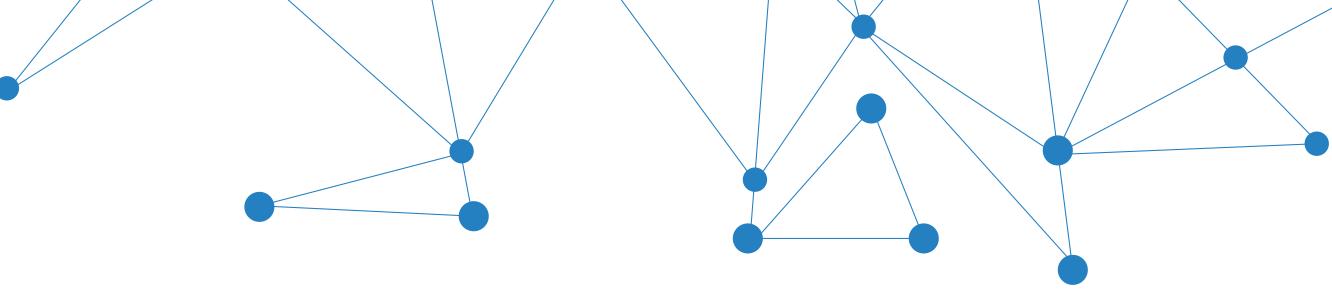
3) display: inline-block

- วางต่อในบรรทัดเดียวกัน (เหมือน inline)
- ไม่กินพื้นที่เต็มบรรทัด
- กำหนด width/height ได้ (เหมือน block)
- ใช้เพื่อเน้นข้อความหรือเพื่อวางชิ้นส่วนเล็ก ๆ ภายในประโยค



Display Properties

วิธีการตั้งค่า properties ของ element



```
<style>
/* ----- STYLE FOR DEMO ----- */
.block-example div {
  display: block;
  padding: 10px;
  margin-bottom: 5px;
  background: #c8e6ff; /* พื้น */
}

.inline-example span {
  display: inline;
  padding: 5px;
  background: #ffc8c8; /* ขอบ */
  margin-right: 5px;
}

.inline-block-example div {
  display: inline-block;
  width: 120px;
  height: 40px;
  background: #d2ffd2; /* เขียวอ่อน */
  margin-right: 10px;
  text-align: center;
  line-height: 40px;
  border: 1px solid #4caf50;
}
</style>
```

display: block;

- ✓ กินพื้นที่เต็มบรรทัด และขึ้นบรรทัดใหม่เสมอ

Block 1

Block 2

display: inline;

- ✓ อุยในบรรทัดเดียวกัน และไม่สามารถกำหนด width/height ได้

Inline 1

Inline 2

Inline 3

display: inline-block;

- ✓ รวมข้อดีของ inline และ block → ไม่ขึ้นบรรทัดใหม่ + กำหนด width/height ได้

Box A

Box B

Box C

Display Properties

ลักษณะการแสดงผล แบบแสดง/ไม่แสดงผล

```
.box {  
    padding: 10px;  
    margin: 10px 0;  
    background-color: #a0d8ff;  
    border: 2px solid #0077aa;  
    font-size: 20px;  
}  
  
/* ช่องแบบหายไปจาก layout */  
.none {  
    display: none;  
}  
  
/* ช่องแต่ยังคงพื้นที่อยู่ */  
.hidden {  
    visibility: hidden;  
}
```

```
<h2>แสดงผลปกติ</h2>  
<div class="box">Box 1 (ปกติ)</div>  
  
<h2>display: none; → หายไปจากหน้าเว็บ</h2>  
<div class="box none">Box 2 (display:none หายไป)</div>  
<div class="box">Box 3 (จะชิดขึ้นมาทันที เพราะ Box 2 หายไป)</div>  
  
<h2>visibility: hidden; → ช่อง แต่ยังคงพื้นที่</h2>  
<div class="box hidden">Box 4 (visibility:hidden ช่องแต่ยังคงขนาด)</div>  
<div class="box">Box 5 (ไม่ขยับขึ้น เพราะ Box 4 ยังมีพื้นที่อยู่)</div>
```

แสดงผลปกติ

Box 1 (ปกติ)

display: none; → **หายไปจากหน้าเว็บ**

Box 3 (จะชิดขึ้นมาทันที เพราะ Box 2 หายไป)

visibility: hidden; → **ช่อง แต่ยังคงพื้นที่**

display: none → ใช้เมื่อต้องให้ “หายไปจาก Layout เลย”
visibility: hidden → ใช้เมื่อต้อง “ช่อง แต่ยังคงพื้นที่ไว้”

Box 5 (ไม่ขยับขึ้น เพราะ Box 4 ยังมีพื้นที่อยู่)

1) **display: none;** เมนะสำหรับ

- ช่องเมนู
- ช่องปีอปอัป
- ช่องฟอร์มที่ต้องการให้โคล่เฉพาะบางเจ๊อนไข
- ใช้กับ JavaScript เวลา toggle UI

2) **visibility: hidden;** เมนะสำหรับ

- ต้องการรักษา layout
- ตาราง (table) กี่ต้องการซ่อนข้อมูลบางช่อง
- UI กี่ต้องการเก็บตำแหน่งองค์ประกอบไว้

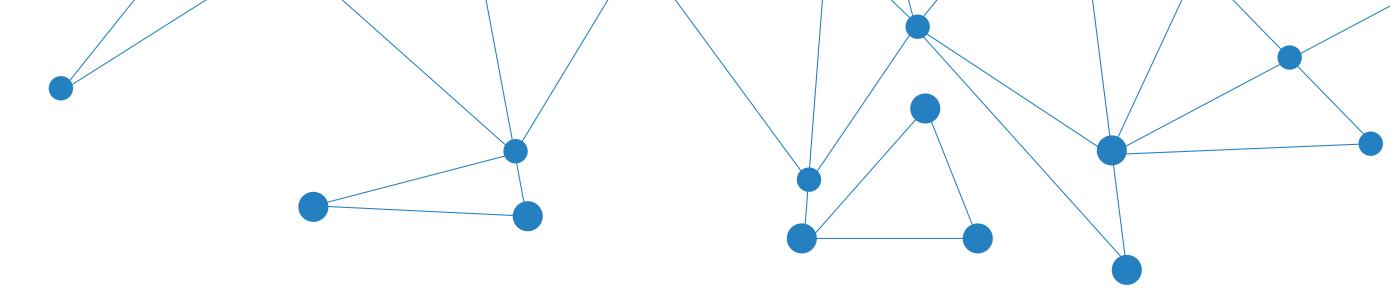
CSS Flexbox and Grid



Flexbox
one dimension



CSS Grids
two dimension



CSS Flexbox

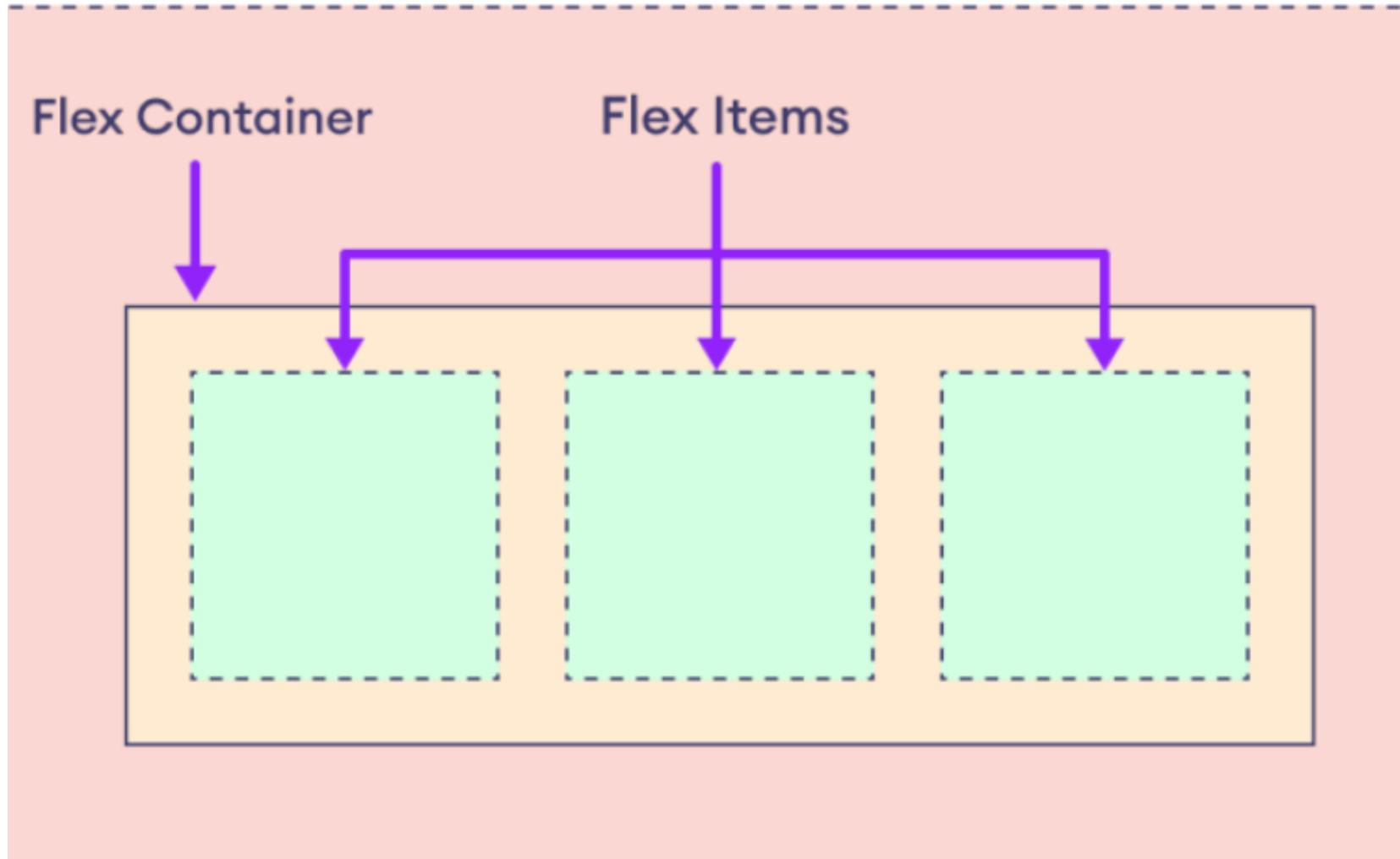


Flexbox
one dimension

Flex จัดวางไปในทิศทางใดทิศทางหนึ่ง (One Dimensions) แนวตั้งหรือแนวนอนนั่นเอง สามารถจัดการตามแน่นองการแสดงผล เวลา Element ด้านในที่เลือกสามารถพื้นที่ใน Container ได้ และสามารถใช้ร่วมกันกับ Grid ได้ด้วย

CSS Flexbox

Flex Container และ Flex Item



Flexbox มีองค์ประกอบหลัก 2 ส่วน

1. Flex Container

คือ element แม่ (parent) ต้องกำหนด `display: flex;` กำหนดคุณสมบัติทาง การจัดเรียง ระยะห่าง และการจัดตำแหน่งของลูก

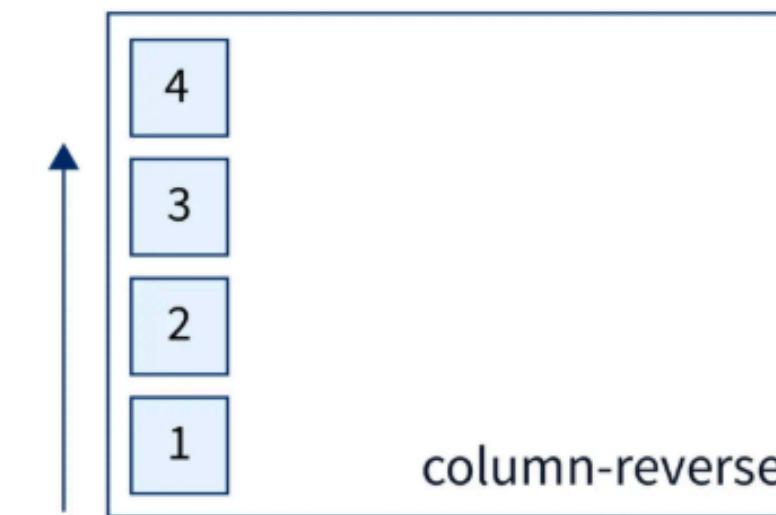
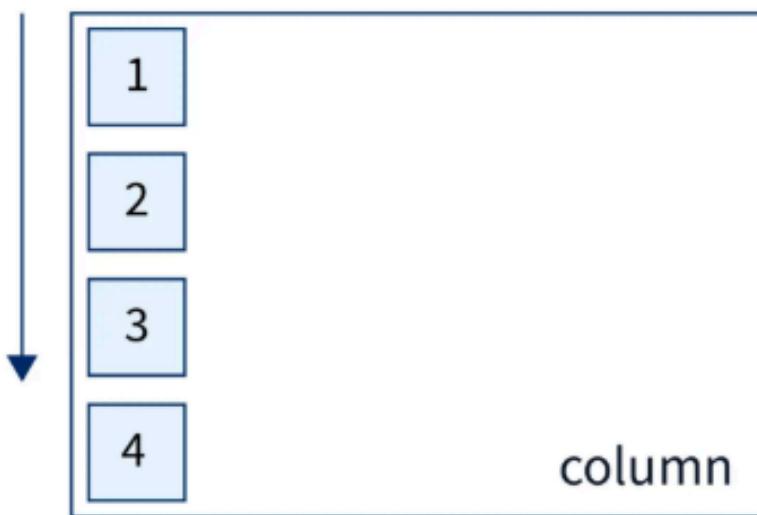
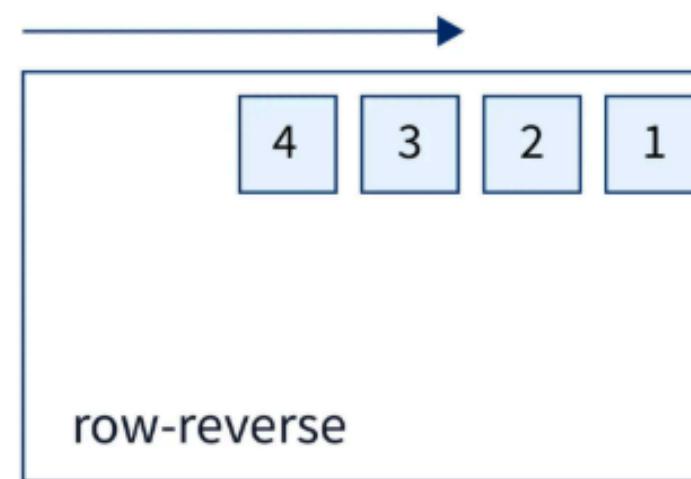
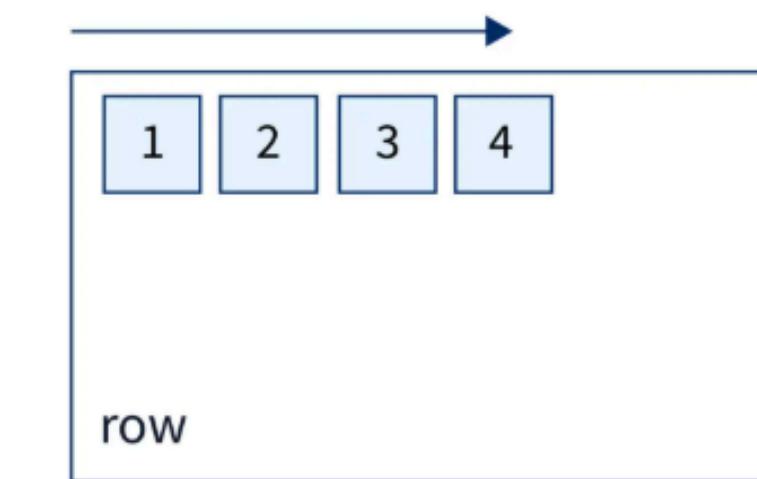
2. Flex Item

- คือ element แม่ (parent) ต้องกำหนดคือ element ลูกที่อยู่ภายใต้ Flex Container
- จะถูกจัดเรียงอัตโนมัติตามกฎของ Flexbox

CSS Flexbox

Property ต่าง ๆ ของ Flex Container

flex-direction ใช้สำหรับกำหนด ทิศทางการจัดเรียงของ Flex Item ภายใน Flex Container ว่าจะเรียงแนวบอนหรือแนวตั้ง และเรียงจากทิศทางใด



1. row (ค่าเริ่มต้น)

- เรียง Flex Item เป็นแนวอน
- จาก ซ้าย → ขวา
- ใช้ป้อยกับเมนู, การ์ด, ปุ่ม

`flex-direction: row;`

2. row-reverse

- เรียง Flex Item เป็นแนวอน
- จาก ขวา → ซ้าย

`flex-direction: row-reverse;`

3. column

- เรียง Flex Item เป็นแนวตั้ง
- จาก บน → ล่าง
- เหมาะสมกับ layout แบบฟอร์ม หรือ sidebar

`flex-direction: column;`

4. column-reverse

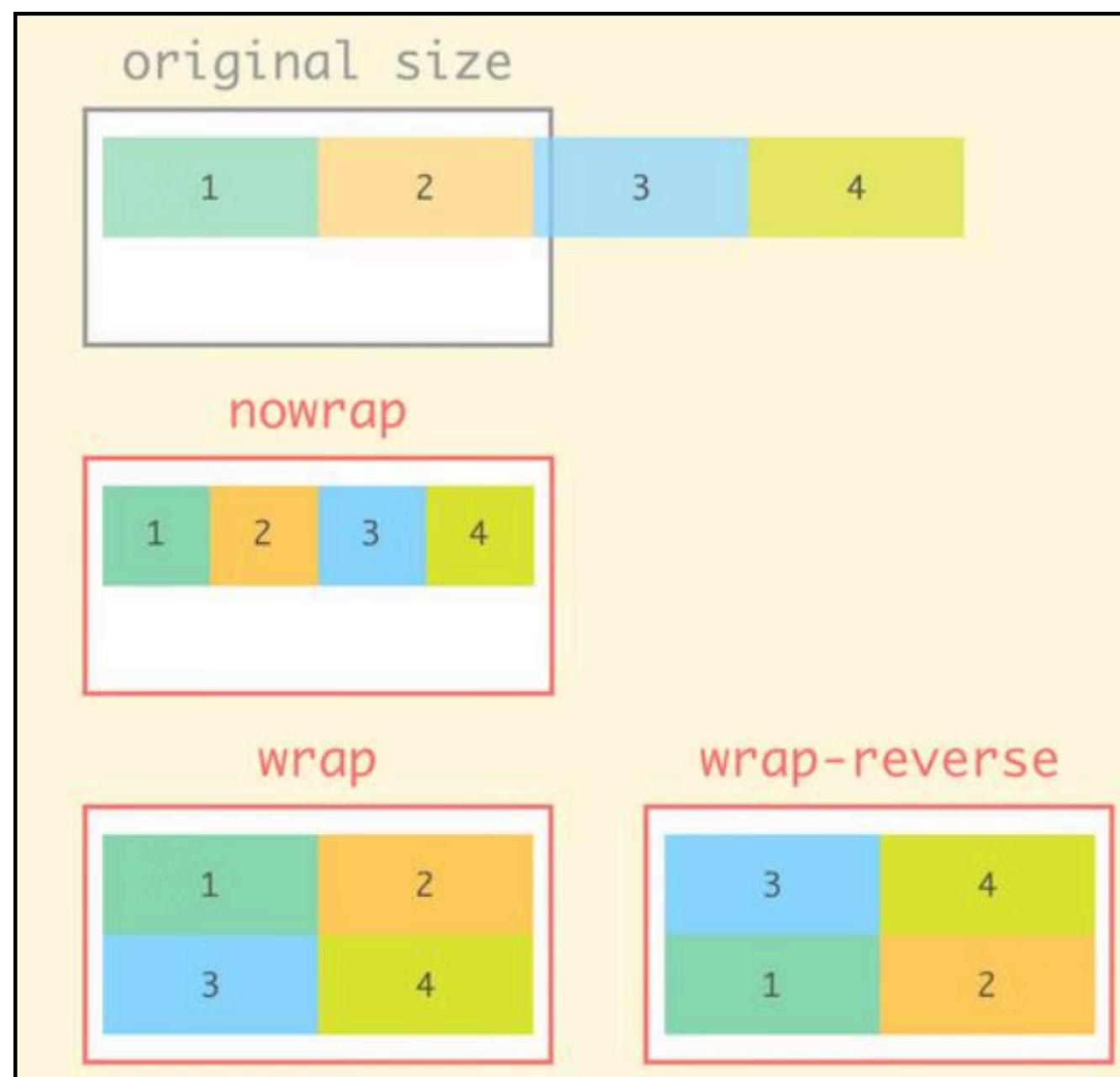
- เรียง Flex Item เป็นแนวตั้ง
- จาก ล่าง → บน

`flex-direction: column-reverse;`

CSS Flexbox

Property ต่าง ๆ ของ Flex Container

flex-wrap ใช้กำหนดว่า Flex Item จะขึ้นบรรทัดใหม่ หรือไม่ เมื่อพื้นที่ของ Flex Container ไม่เพียงพอ



1.nowrap (ค่าเริ่มต้น)

- ไม่อนุญาตให้ขึ้นบรรทัดใหม่
- Flex Item จะพยายามอยู่ในบรรทัดเดียวกันก็ังหมด
- หากพื้นที่ไม่พอ → กล่องจะ ถูกบีบขนาด หรือ สับออกด้านข้าง
- หมายเหตุ: เมนูที่ต้องอยู่บรรทัดเดียว เช่น Navbar

flex-wrap: nowrap;

2.wrap

- อนุญาตให้ขึ้นบรรทัดใหม่
- เมื่อพื้นที่ไม่พอ Flex Item จะ ตัดลงบรรทัดถัดไป
- การเรียงบรรทัด: จากบน → ล่าง
- หมายเหตุ: การ์ดสินค้า, Gallery, Responsive Layout

flex-wrap: wrap;

3.wrap-reverse

- อนุญาตให้ขึ้นบรรทัดใหม่
- แต่เรียงบรรทัดแบบ จากล่าง → บน

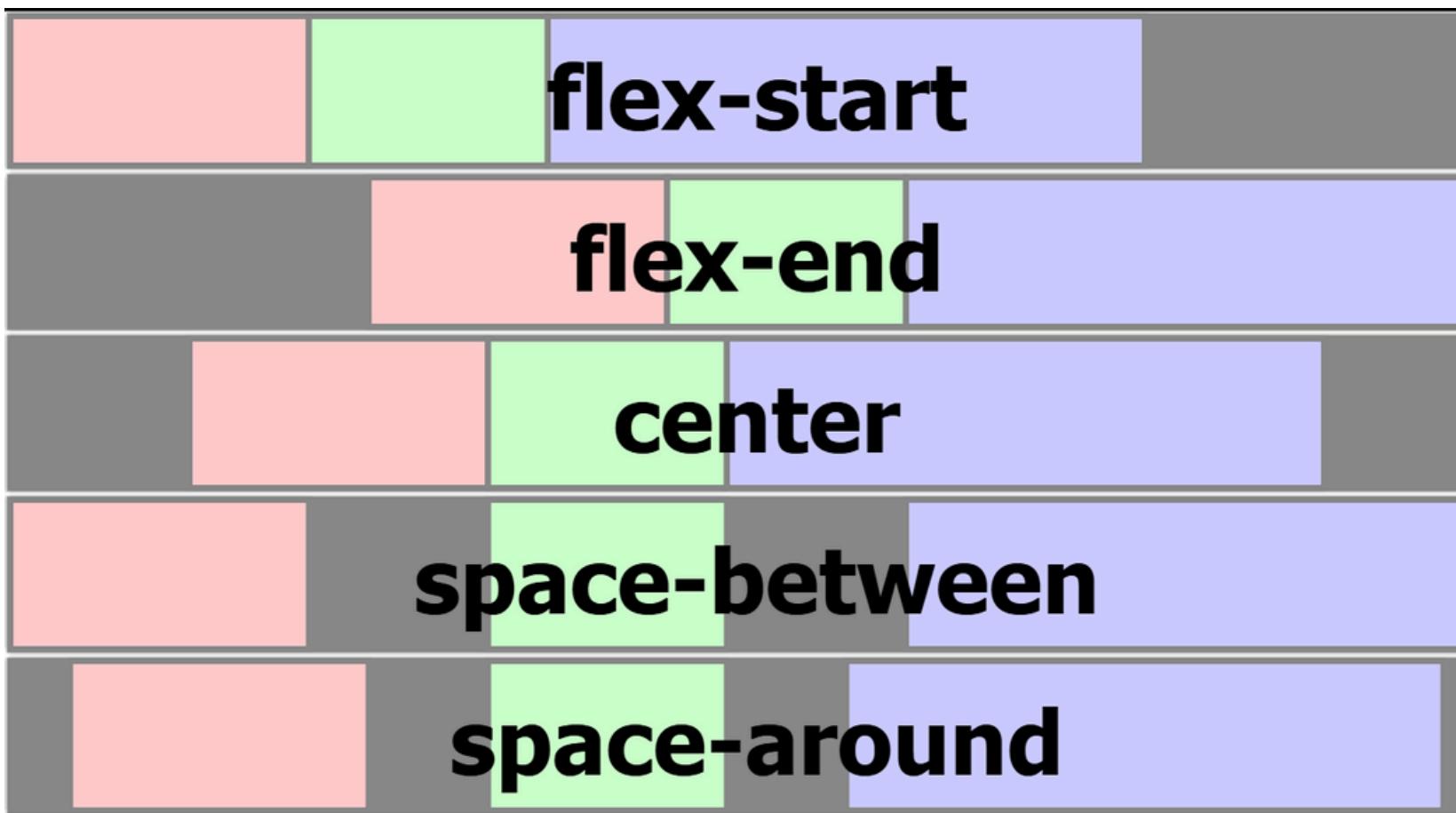
flex-wrap: wrap-reverse;

CSS Flexbox

Property ต่าง ๆ ของ Flex Container

justify-content ใช้กำหนดการจัดตำแหน่งของ Flex Item ตามแกนหลัก

- ถ้า flex-direction: row → จัดตามแนวนอน
- ถ้า flex-direction: column → จัดตามแนวตั้ง



1. flex-start

- จัด Flex Item ชิดด้านเริ่มต้นของแกน
- (row = ซิดซ้าย / column = ซิดบน)

`justify-content: flex-start;`

2. flex-end

- จัด Flex Item ชิดด้านปลายของแกน
- (row = ซิดขวา / column = ซิดล่าง)

`justify-content: flex-end;`

3. center

- ช่องว่างอยู่ระหว่างกล่อง
- กล่องแรกชิดซ้าย กล่องสุดท้ายชิดขวา

`justify-content: space-between;`

4. space-between

- ช่องว่างอยู่ระหว่างกล่อง
- กล่องแรกชิดซ้าย กล่องสุดท้ายชิดขวา

`justify-content: space-between;`

5. space-around

- แต่ละกล่องมี ช่องว่างรอบตัว
- ช่องว่างด้านซ้าย-ขวาของแต่ละกล่อง เก่ากัน

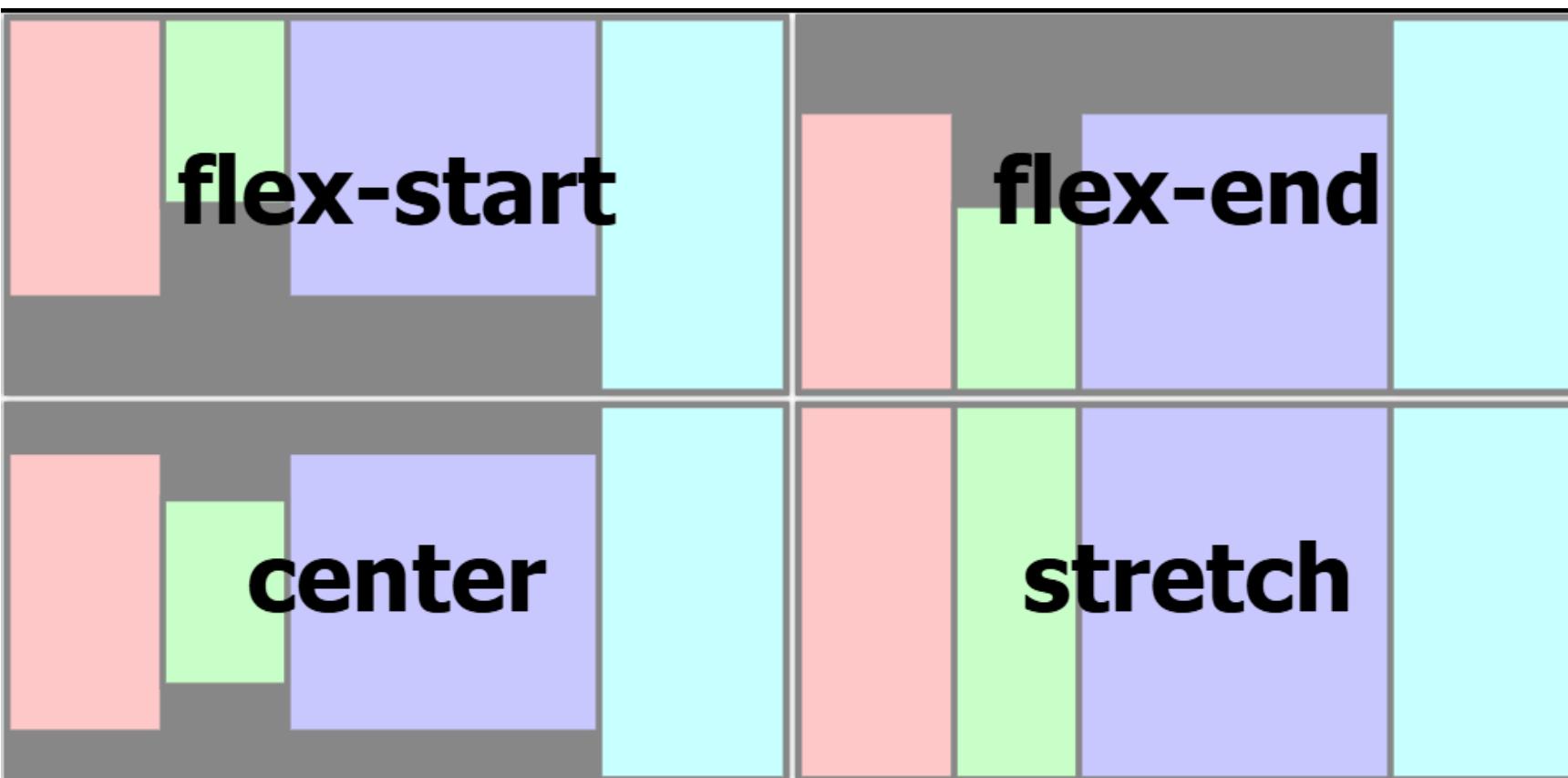
`justify-content: space-around;`

CSS Flexbox

Property ต่าง ๆ ของ Flex Container

align-items ใช้กำหนดการจัดตำแหน่งของ Flex Item ตามแกนรอง ซึ่งเป็นแกนที่ ตั้งจากกับ flex-direction

- ถ้า flex-direction: row → align-items จะจัดใน แนวตั้ง
- ถ้า flex-direction: column → align-items จะจัดใน แนวนอน



1. flex-start

- จัด Flex Item ชิดด้านเริ่มต้นของแกนรอง
- (row → ชิดบน, column → ชิดซ้าย)

align-items: flex-start;

2. flex-end

- จัด Flex Item ชิดด้านปลายของแกนรอง
- (row → ชิดล่าง, column → ชิดขวา)

align-items: flex-end;

3. center

- จัด Flex Item ให้อยู่กึ่งกลางของ Flex Container

align-items: center;

4. stretch

- ถ้า ไม่ได้กำหนดความสูง (height) ให้ Flex Item
- stretch จะ ยืดกล่องในแนวตั้ง ให้สูงเท่ากับ Container

align-items: stretch;

Flexbox (Flexible Box Layout)

เป็นระบบจัดวางตำแหน่งของกล่องใน CSS ที่ช่วยให้จัด layout ได้ง่ายขึ้น

- จัดให้อยู่ตรงกลาง (กึ่งแนวตั้ง–แนวนอน)
- เว้นระยะห่างอัตโนมัติ
- กระจายพื้นที่เท่ากัน
- ยืด–หดได้ตามขนาดหน้าจอ

โครงสร้างพื้นฐานของ Flexbox มี 2 ส่วน

1) Flex Container (ตัวแม่)

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

2) Flex Items (ตัวลูก) เป็นกล่องที่อยู่ภายใน container เช่น <div>, ,

```
.item { background:#4a90e2; color:#fff; padding:10px; }
```

html

```
<div class="container">
  <div class="item">กล่อง 1</div>
  <div class="item">กล่อง 2</div>
</div>
```

คำสั่งสำคัญ

1) justify-content (จัดตำแหน่งแนวอน)

ค่า	คำอธิบาย
flex-start	ซิดซ้าย
center	อยู่ตรงกลาง
flex-end	ซิดขวา
space-between	กระจายซิดสุดทุกด้าน
space-around	กระจายเมรยะรอบเท่ากัน

2) align-items (จัดตำแหน่งแนวตั้ง)

ค่า	คำอธิบาย
flex-start	ซิดบน
center	กึ่งกลางแนวตั้ง
flex-end	ซิดล่าง
stretch	ยืดเต็มสูง

Flexbox (Flexible Box Layout)

1.row Layout

```
/* กล่องแม่ */
.flex-box {
  display: flex;
  flex-direction: row; /* row หรือ column */
  gap: 10px;
  background: #f0f0f0;
  padding: 15px;
  border: 2px solid #ccc;
  height: 150px;
}

/* กล่องลูก */
.item {
  background: navy;
  color: white;
  font-size: 20px;
  width: 80px;
  text-align: center;
  padding: 10px 0;
}

/* ใช้ align-items: stretch */
.stretch .item {
  background: teal;
```

```
<!-- ===== -->
<h2>1) justify-content: center</h2>
<div class="flex-box" style="justify-content: center;">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
</div>

<!-- ===== -->
<h2>2) justify-content: space-between</h2>
<div class="flex-box" style="justify-content: space-between;">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
</div>

<!-- ===== -->
<h2>3) align-items: center (จัดตรงกลางแนวตั้ง)</h2>
<div class="flex-box" style="align-items: center;">
  <div class="item" style="height:40px;">1</div>
  <div class="item" style="height:80px;">2</div>
  <div class="item" style="height:60px;">3</div>
</div>

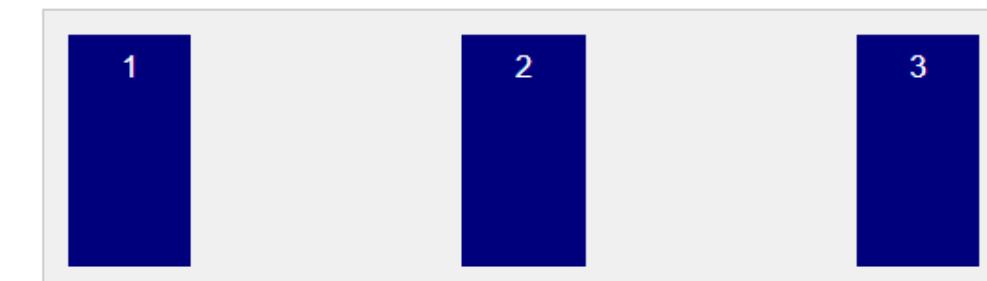
<!-- ===== -->
<h2>4) align-items: stretch (ลูกๆก็ยืดให้เท่ากัน)</h2>
<div class="flex-box stretch" style="align-items: stretch;">
  <div class="item">A</div>
  <div class="item">B</div>
  <div class="item">C</div>
</div>
```

CSS Flexbox Demo (row Layout)

1) justify-content: center



2) justify-content: space-between



3) align-items: center (จัดตรงกลางแนวตั้ง)



4) align-items: stretch (ลูกๆก็ยืดให้เท่ากัน)



Flexbox (Flexible Box Layout)

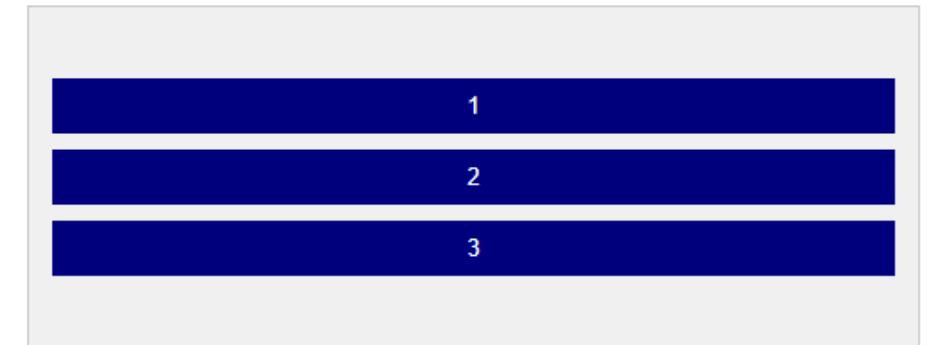
2. column Layout

```
.flex-box {  
    display: flex;  
    flex-direction: column; /* ← เป็นแนวตั้ง */  
    gap: 10px;  
    background: #f0f0f0;  
    padding: 15px;  
    border: 2px solid #ccc;  
    height: 200px; /* เพิ่มความสูงเพื่อเห็นการจัดวาง */  
}  
  
.item {  
    background: navy;  
    color: white;  
    padding: 10px;  
    text-align: center;  
}  
  
/* แสดงกรณี stretch */  
.stretch .item {  
    background: teal;  
    height: auto;  
}
```

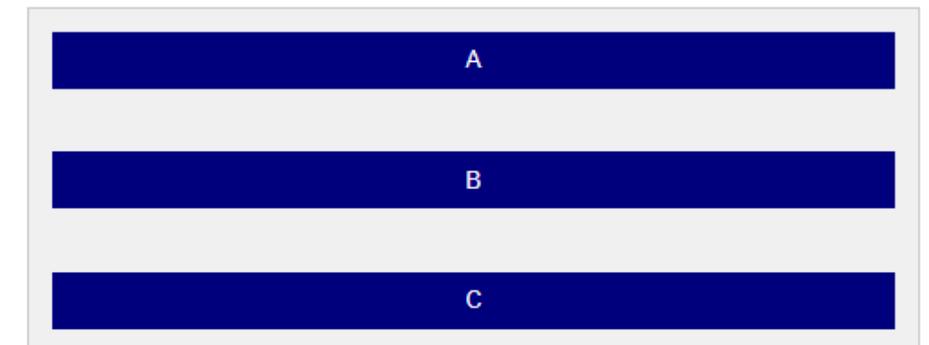
```
<!-- ===== -->  
<h2>1) justify-content: center (จัดลูกไห้อยู่กลาง “แนวตั้ง”)</h2>  
<div class="flex-box" style="justify-content: center;">  
    <div class="item">1</div>  
    <div class="item">2</div>  
    <div class="item">3</div>  
</div>  
  
<!-- ===== -->  
<h2>2) justify-content: space-between (กระจายบน-ล่าง)</h2>  
<div class="flex-box" style="justify-content: space-between;">  
    <div class="item">A</div>  
    <div class="item">B</div>  
    <div class="item">C</div>  
</div>  
  
<!-- ===== -->  
<h2>3) align-items: center (ลูกอยู่กึ่งกลาง “แนวอน”)</h2>  
<div class="flex-box" style="align-items: center;">  
    <div class="item" style="width:60px;">X</div>  
    <div class="item" style="width:80px;">Y</div>  
    <div class="item" style="width:40px;">Z</div>  
</div>  
  
<!-- ===== -->  
<h2>4) align-items: stretch (ลูกถูกยืดเต็มความกว้างแนวอน)</h2>  
<div class="flex-box stretch" style="align-items: stretch;">  
    <div class="item">One</div>  
    <div class="item">Two</div>  
    <div class="item">Three</div>  
</div>
```

Flexbox Demo (column Layout)

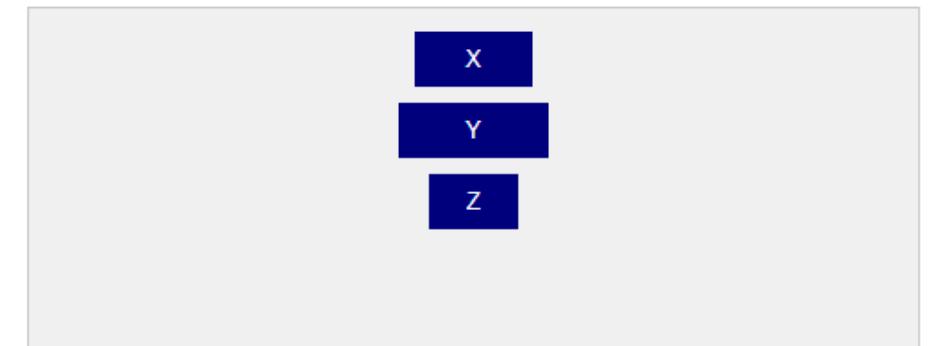
1) justify-content: center (จัดลูกไห้อยู่กลาง “แนวตั้ง”)



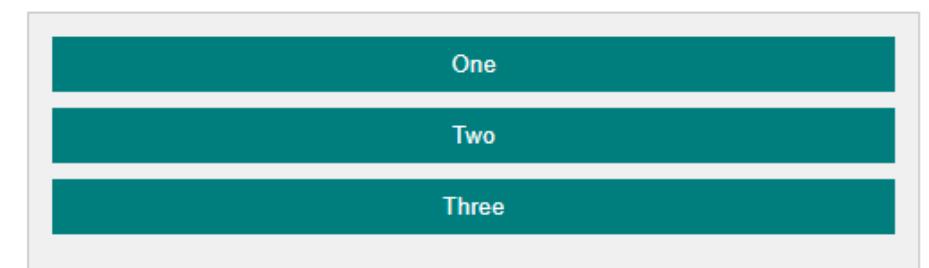
2) justify-content: space-between (กระจายบน-ล่าง)



3) align-items: center (ลูกอยู่กึ่งกลาง “แนวอน”)



4) align-items: stretch (ลูกถูกยืดเต็มความกว้างแนวอน)

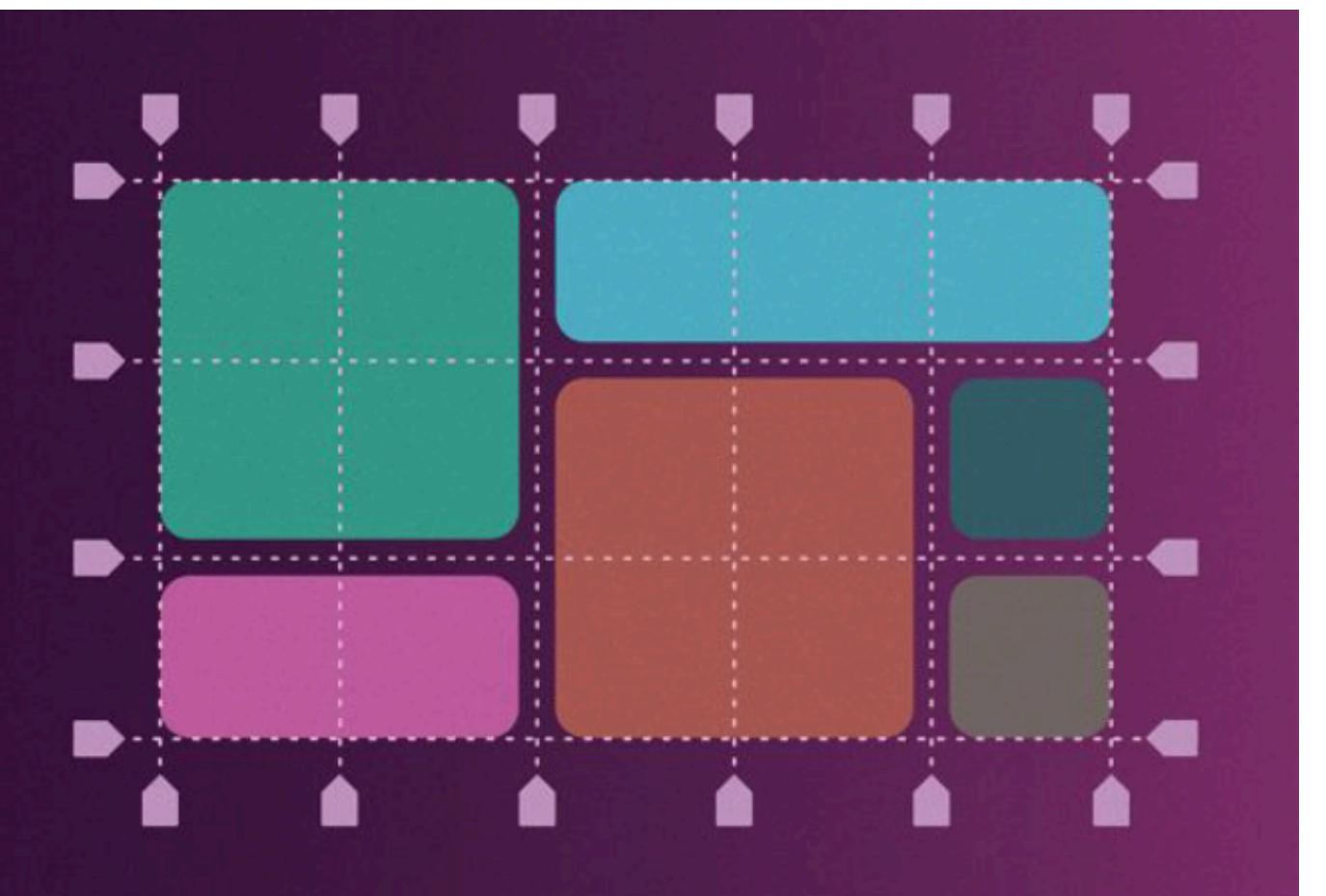
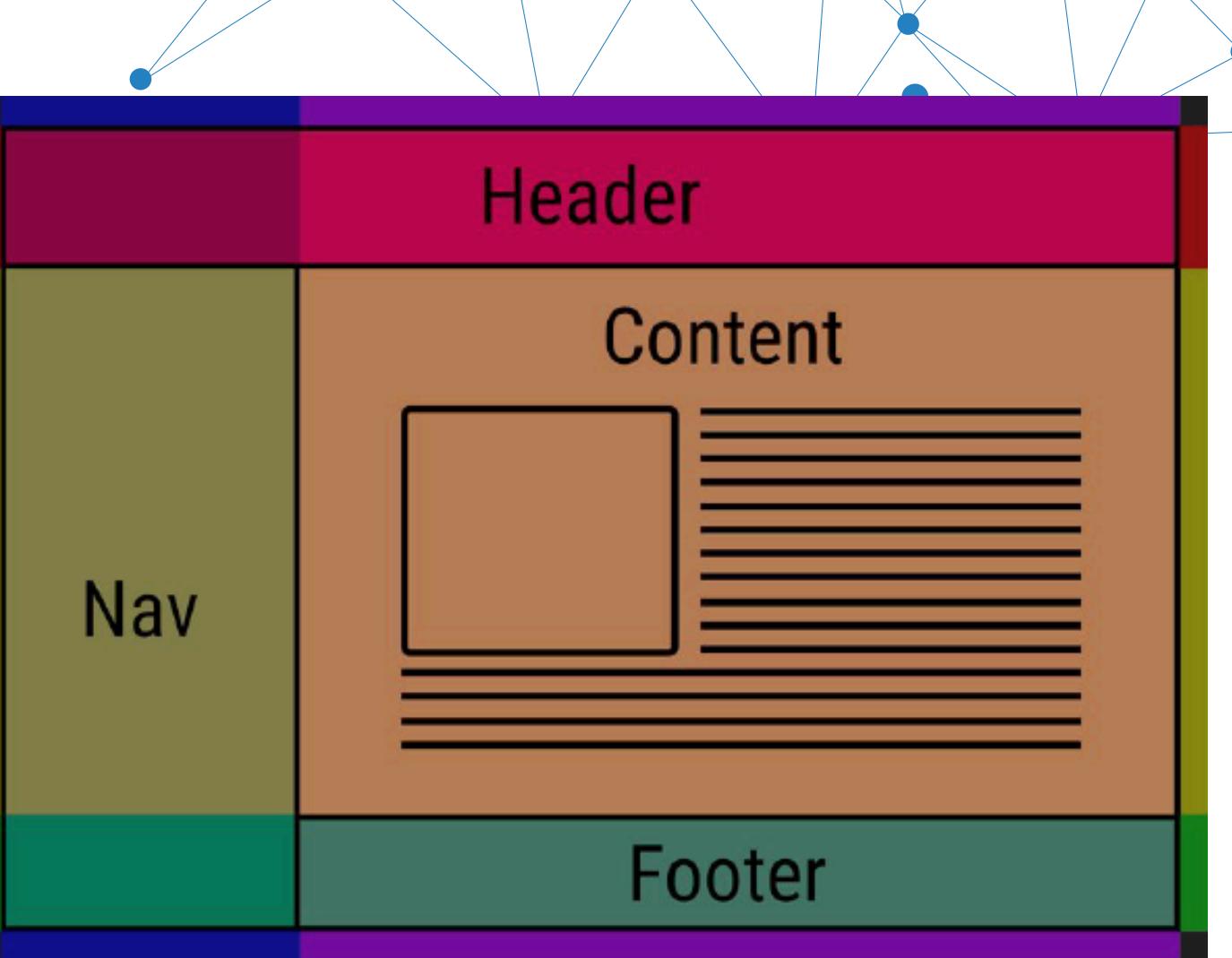


CSS Grid System

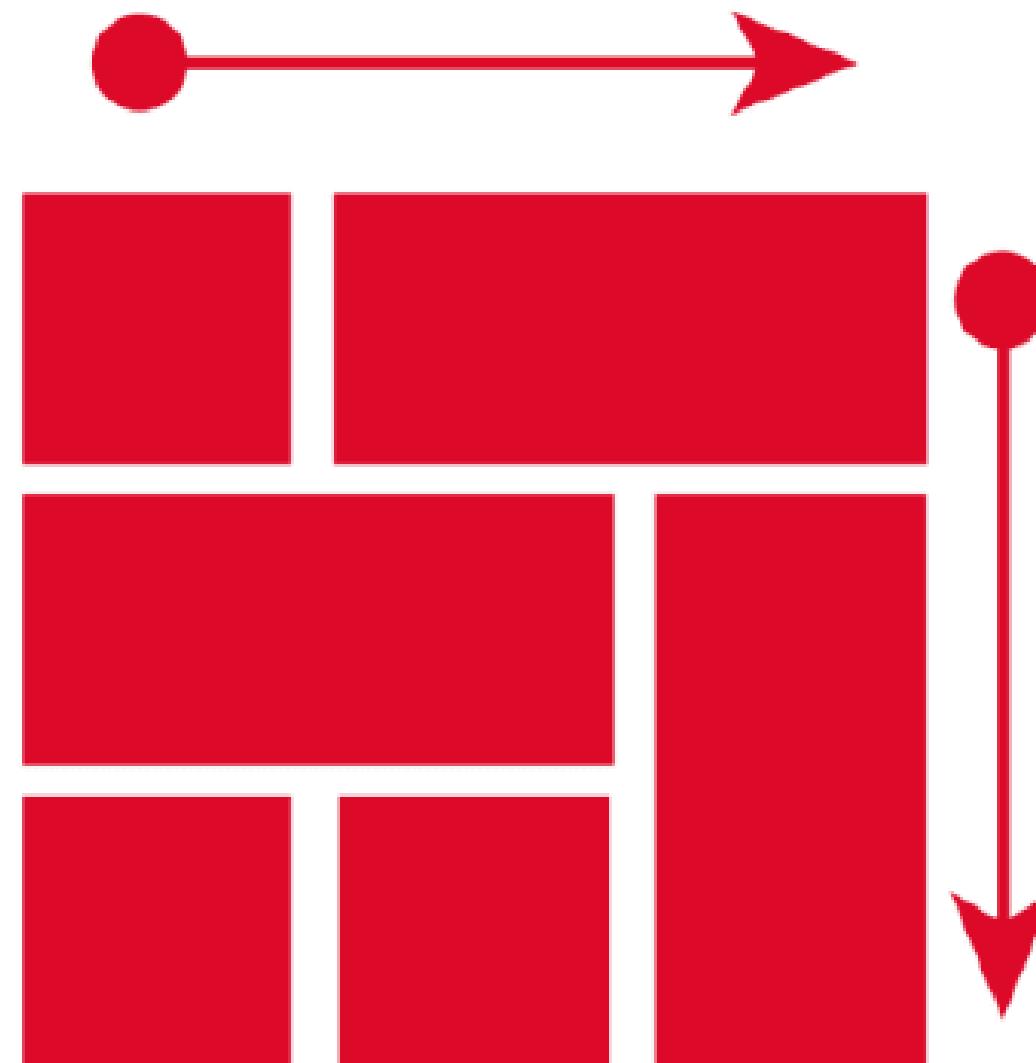
CSS Grid คือ ระบบจัดวางโครงสร้างเว็บแบบตาราง (Grid Layout) ช่วยให้สร้าง “layout ที่ซับซ้อน” ได้ง่าย เช่น dashboard, card layout, landing page

แนวคิดของ Grid System

- เว็บไซต์ส่วนใหญ่มีโครงสร้างที่เป็น “ช่องตาราง” เช่น Header / Sidebar / Content / Footer
- CSS Grid ช่วยให้เราวาง Layout แบบหลายแถว (rows) และหลายคอลัมน์ (columns) ได้อย่างเป็น ระเบียบ
- เราสามารถควบคุมขนาด, การจัดตำแหน่ง และพื้นที่ที่ แต่ละกล่องครอบคล้องได้ง่ายมาก



CSS Grids



CSS Grids
two dimension

Grid CSS คือ เครื่องมือช่วยจัดวางหน้าเว็บ
ที่สามารถ กำหนดตำแหน่งกล่องได้ทั้งแนว
นอนและแนวตั้งพร้อมกัน

นึกภาพเหมือน ตาราง Excel

- มีแถว (Row)
- มีคอลัมน์ (Column)
- เอกล่องไปทางซ่องไหนก็ได้

ทำไมต้องใช้ Grid

อดีตเวลาออกแบบ Layout ใหญ่ ๆ ต้องใช้

- float
- position

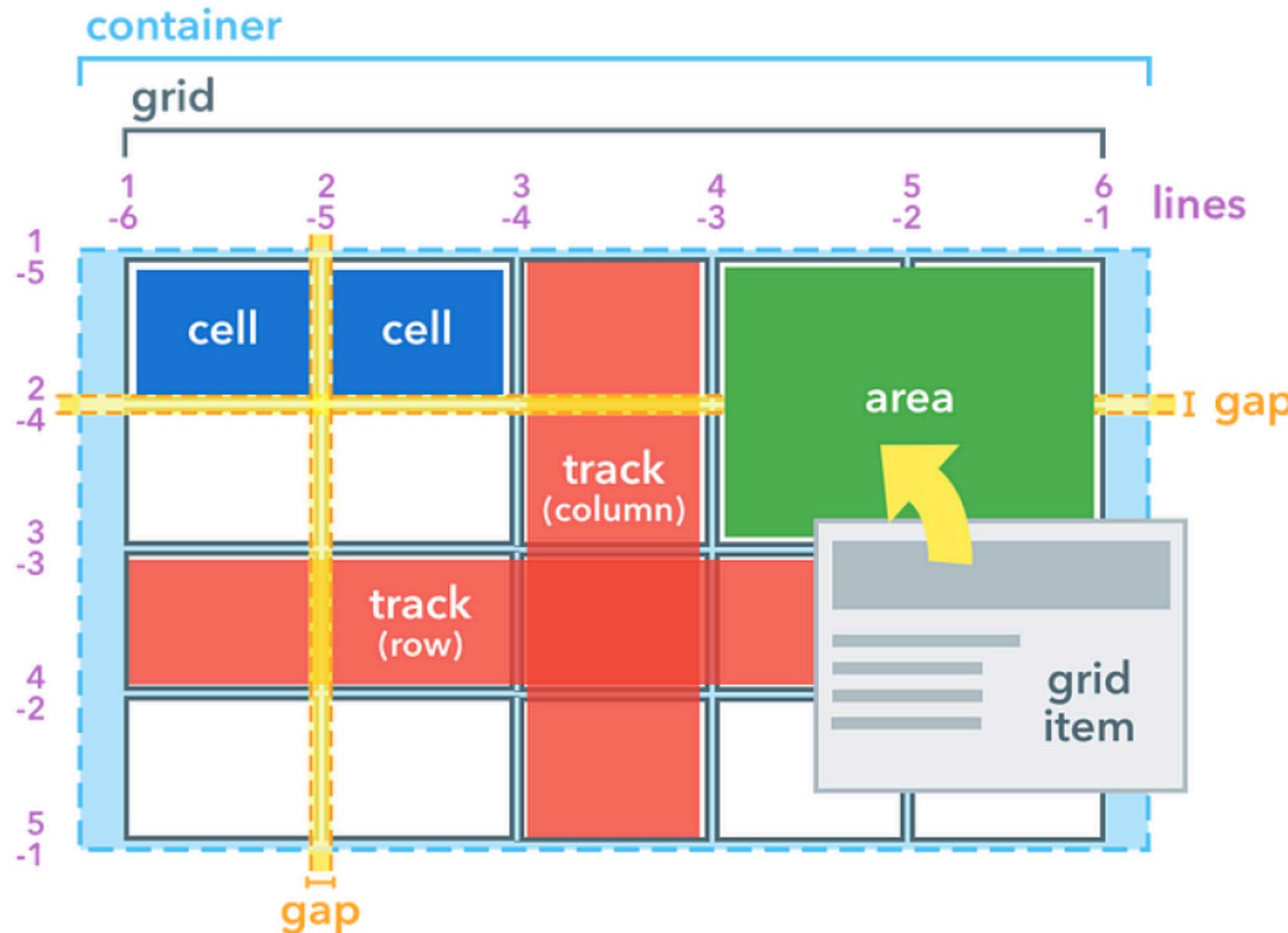
โค้ดยุ่งยาก / ควบคุมยาก

ปัจจุบันใช้ Grid แทนได้เลย

- โครงสร้างชัด
- อ่านโค้ดง่าย
- จัด Layout ได้แม่น

CSS Grids

Grid Layout คือ ระบบจัดวางหน้าเว็บของ CSS ที่ช่วยให้จัดตำแหน่งองค์ประกอบได้แบบ ตาราง (2 มิติ) คือ กั้งแ Kaw (Row) และคอลัมบ์ (Column) พร้อมกัน



1. Grid Container คือ คือกรอบใหญ่กั้งหมด

- เป็นตัวกำหนดว่า “ตรงนี้ใช้ Grid”

2. Grid Item คือ กล่องลูกทุกกล่องที่อยู่ใน Container

- ไม่ต้องประกาศอะไรเพิ่ม
 - ลูกของ Grid ทุกตัว = Grid Item อัตโนมัติ
- ใช้ Grid เพื่อ “บอกตำแหน่ง” ของ item

3. Grid Line คือ เส้นแบ่งตาราง

- แนวอน → Grid Line Row
- แนวตั้ง → Grid Line Column

เส้นจะมี “หมายเลข” ไว้ใช้อ้างอิงตำแหน่ง เช่นเริ่มที่เส้นที่ 1 ไปเส้นที่ 3

4. Grid Track คือ พื้นที่ระหว่างเส้น

- ระหว่างเส้นแนวอน → Grid Row
- ระหว่างเส้นแนวตั้ง → Grid Column

5. Grid Cell คือ ช่องสี่เหลี่ยม 1 ช่อง

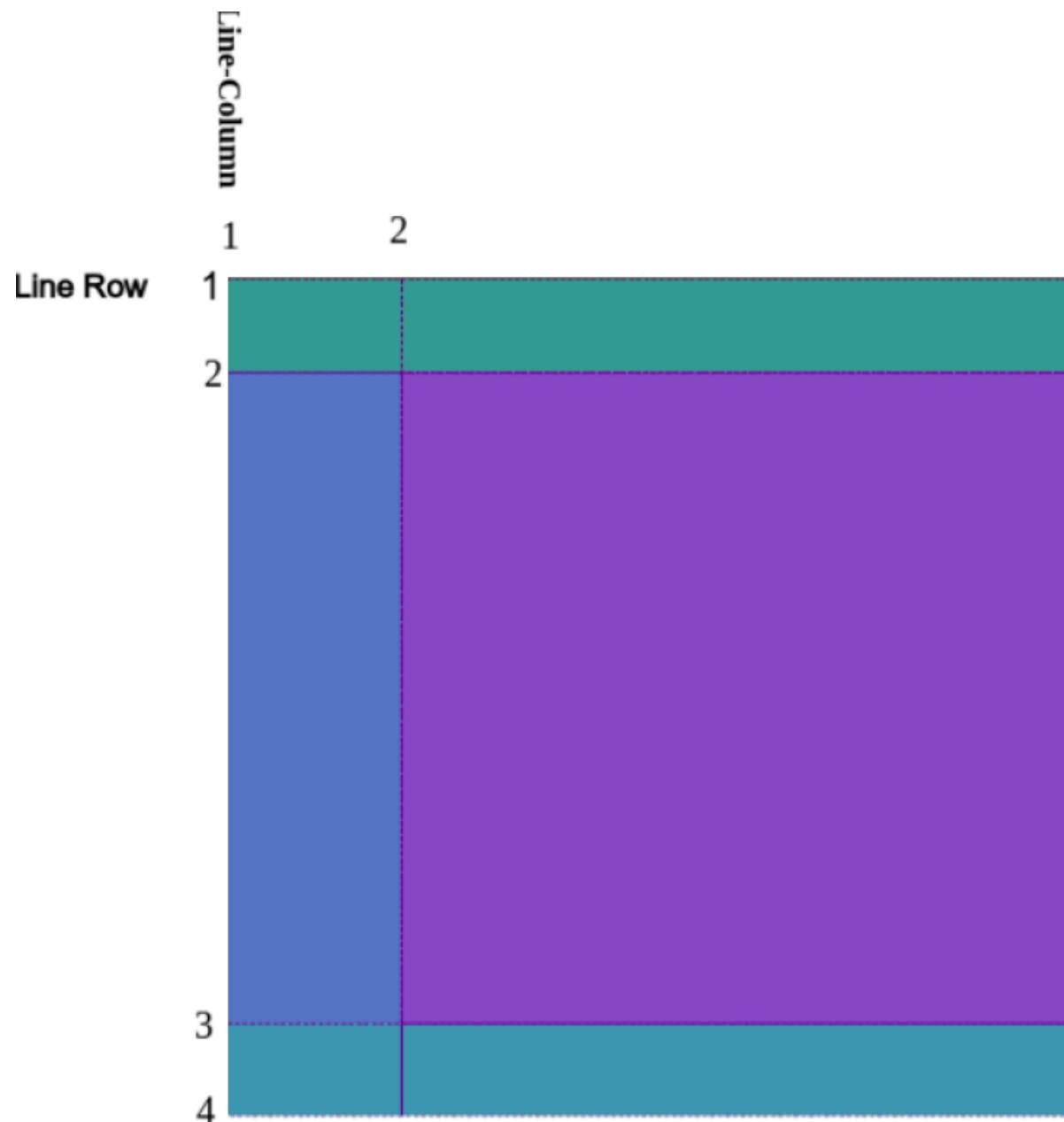
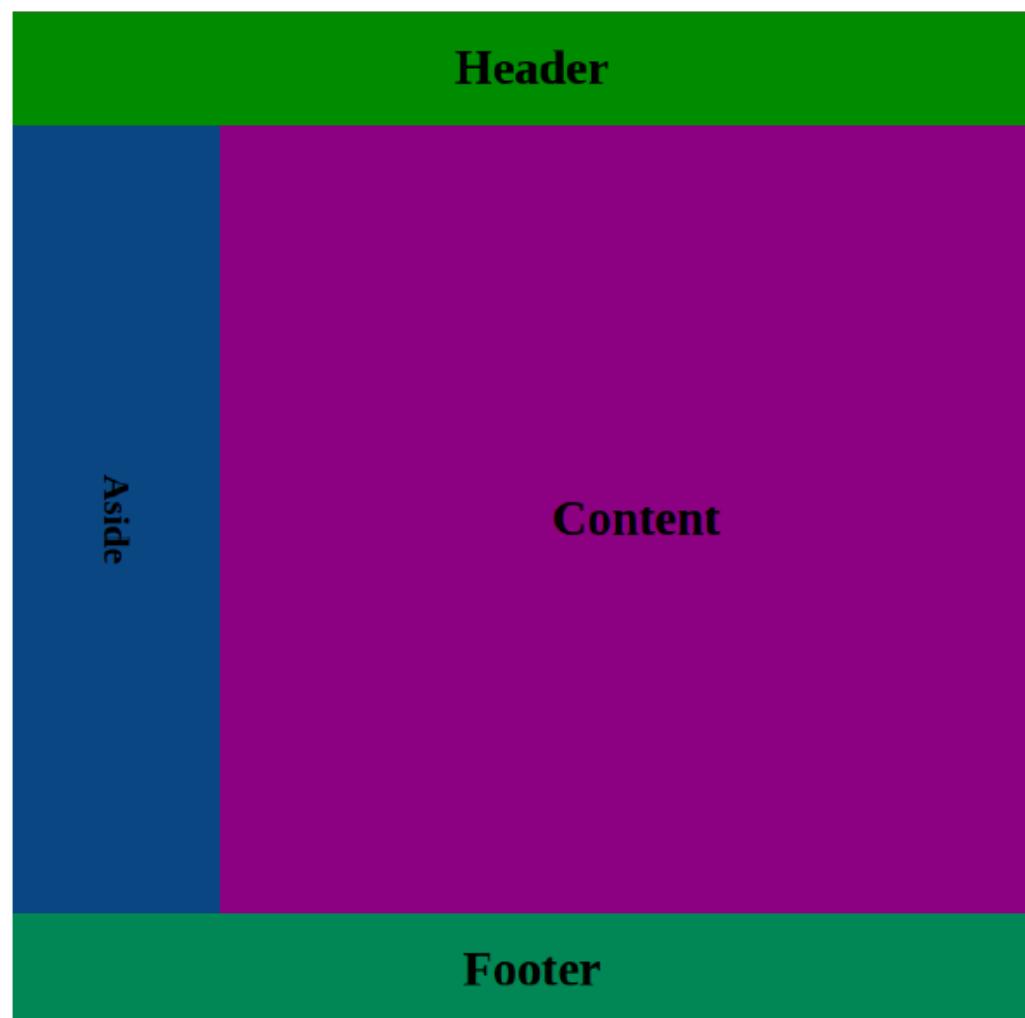
- เกิดจาก Row 1 × Column 1

6. Grid Area คือ พื้นที่ใหญ่ที่รวมหลาย Cell

- เช่น Header กิน 3 ช่อง
- Sidebar กิน 2 แ Kaw

7. Grid Gap คือ ช่องว่างระหว่างช่อง (ไม่ต้องใช้ margin ให้ยุ่งยาก)

CSS Grids



Grid Item ไม่จำเป็นต้องอยู่ใน Cell เดียวเสมอ
เพราอย่าง Header และ Footer อยู่ในพื้นที่ 2 Grid Track Column

การแบ่ง layout เป็นส่วนต่างๆ ให้ง่าย
ต่อการจัดวาง เราสามารถเป็น grid

- Grid Line Row 4 เส้น
- Grid Line Column 3 เส้น
- Grid Track Row 3 ช่อง
- Grid Track Column 2 ช่อง

CSS Grids

Grid Layout

แค่เขียน `display: grid;` ก็เริ่มใช้ Grid ได้กับที่
กล่องลูกทั้งหมดจะถูกจัดวางแบบ “ตาราง” อัตโนมัติ

html

```
<div class="container">
  <div class="grid-item">Home</div>
  <div class="grid-item">About us</div>
  <div class="grid-item">Portfolio</div>
  <div class="grid-item">Contact Us</div>
  <div class="grid-item">Privacy Policy</div>
</div>
```

- container = กรอบใหญ่ (Grid Container)
 - grid-item = กล่องลูก (Grid Item)
- ทุก div ที่อยู่ใน container จะกลายเป็น Grid Item กันที

เปิดโหมด Grid ด้วย CSS

```
.container {
  display: grid;
}
```

```
.container {
  display: grid;
  grid: auto auto / auto auto auto;
  background-color: mediumturquoise;
  grid-gap: 10px;
  padding: 10px;
}
```

```
.grid-item {
  background-color: white;
  text-align: center;
  padding: 25px 0;
  font-size: 30px;
}
```

→ 2 แล้ว → 3 คอลัมน์
→ กำหนดพื้นหลังสี
→ ช่องว่างระหว่างช่อง
→ ระยะห่างขอบภายใน

จัดหน้าตากล่องลูก

CSS Grids



CSS Grids

กำหนด property แบบปกติ

```
/* Grid Container */
.grid-container {
  display: grid;

  /* แบ่งคอลัมน์ */
  grid-template-columns: 20% 80%;

  /* แบ่งแถว */
  grid-template-rows: 70px 530px 70px;
  gap: 10px;
  padding: 10px;
  background-color: #e0f7f7;
  height: 100vh;
  box-sizing: border-box;
}
```

```
/* Header */
header{
  grid-row: 1/2; /* กำหนดให้ header เริ่มแรกอยู่ที่สิ้น row เส้นที่ 1 และ สิ้นสุดที่เส้นที่ 2*/
  grid-column: 1/3; /* กำหนดให้ header เริ่มแรกอยู่ที่สิ้น column เส้นที่ 1 และ สิ้นสุดที่เส้นที่ 3*/
  background-color: #rgb(52, 129, 0); /* สีส่วนของ header */
  color: white; text-align: center; line-height: 70px; font-size: 24px;
}

/* Sidebar */
aside{
  grid-row: 2/3; /* กำหนดให้ aside เริ่มแรกอยู่ที่สิ้น row เส้นที่ 2 และ สิ้นสุดที่เส้นที่ 3*/
  grid-column: 1/2; /* กำหนดให้ aside เริ่มแรกอยู่ที่สิ้น column เส้นที่ 1 และ สิ้นสุดที่เส้นที่ 2*/
  background-color: #rgb(0, 82, 129); /* สีส่วนของ sidebar */
  color: white;
  text-align: center; line-height: 530px; font-size: 24px;
}

/* Main Content */
content {
  grid-row: 2/3; /* กำหนดให้ content เริ่มแรกอยู่ที่สิ้น row เส้นที่ 2 และ สิ้นสุดที่เส้นที่ 3*/
  grid-column: 2/3; /* กำหนดให้ content เริ่มแรกอยู่ที่สิ้น column เส้นที่ 2 และ สิ้นสุดที่เส้นที่ 3*/
  background-color: #rgb(129, 0, 129); /* สีส่วนของ content */
  color: white; text-align: center; line-height: 530px; font-size: 24px;
}

/* Footer */
footer{
  grid-row: 3/4; /* กำหนดให้ footer เริ่มแรกอยู่ที่สิ้น row เส้นที่ 3 และ สิ้นสุดที่เส้นที่ 4*/
  grid-column: 1/3; /* กำหนดให้ footer เริ่มแรกอยู่ที่สิ้น column เส้นที่ 1 และ สิ้นสุดที่เส้นที่ 3*/
  background-color: #rgb(0, 129, 86); /* สีส่วนของ footer */
  color: white; text-align: center; line-height: 70px; font-size: 24px;
}
```

CSS Grids

ถ้ากำหนดชื่อให้พื้นที่นั้นๆ..จะเกิดอะไรขึ้น

```
/* Grid Container */
.grid-container {
  display: grid;

  /* ใช้ grid-template-areas และ grid-row / grid-column */
  grid-template:
    "header header" 70px /*ความสูงของ Grid Track Row ช่องที่ 1=70px */
    "aside main" 530px /*ความสูงของ Grid Track Row ช่องที่ 2=530px */
    "footer footer" 70px /*ความสูงของ Grid Track Row ช่องที่ 3=70px */
    / 20% 80%;
    /*ความสูงของ Grid Track Column ช่องที่ 1=20% และ ช่องที่ 2=80% */

  gap: 10px;
  padding: 10px;
  background-color: #e0f7f7;
  height: 100vh;
  box-sizing: border-box;
}
```

```
/* Header */
header{
  grid-area: header; /*กำหนดให้ header อยู่ในพื้นที่ที่ชื่อ header*/
  background-color: #rgb(52, 129, 0); /*สีส่วนของ header*/
  color: white; text-align: center; line-height: 70px; font-size: 24px;
}

/* Sidebar */
aside{
  grid-area: aside; /*กำหนดให้ aside อยู่ในพื้นที่ที่ชื่อ aside*/
  background-color: #rgb(0, 82, 129); /*สีส่วนของ sidebar*/
  color: white;
  text-align: center; line-height: 530px; font-size: 24px;
}

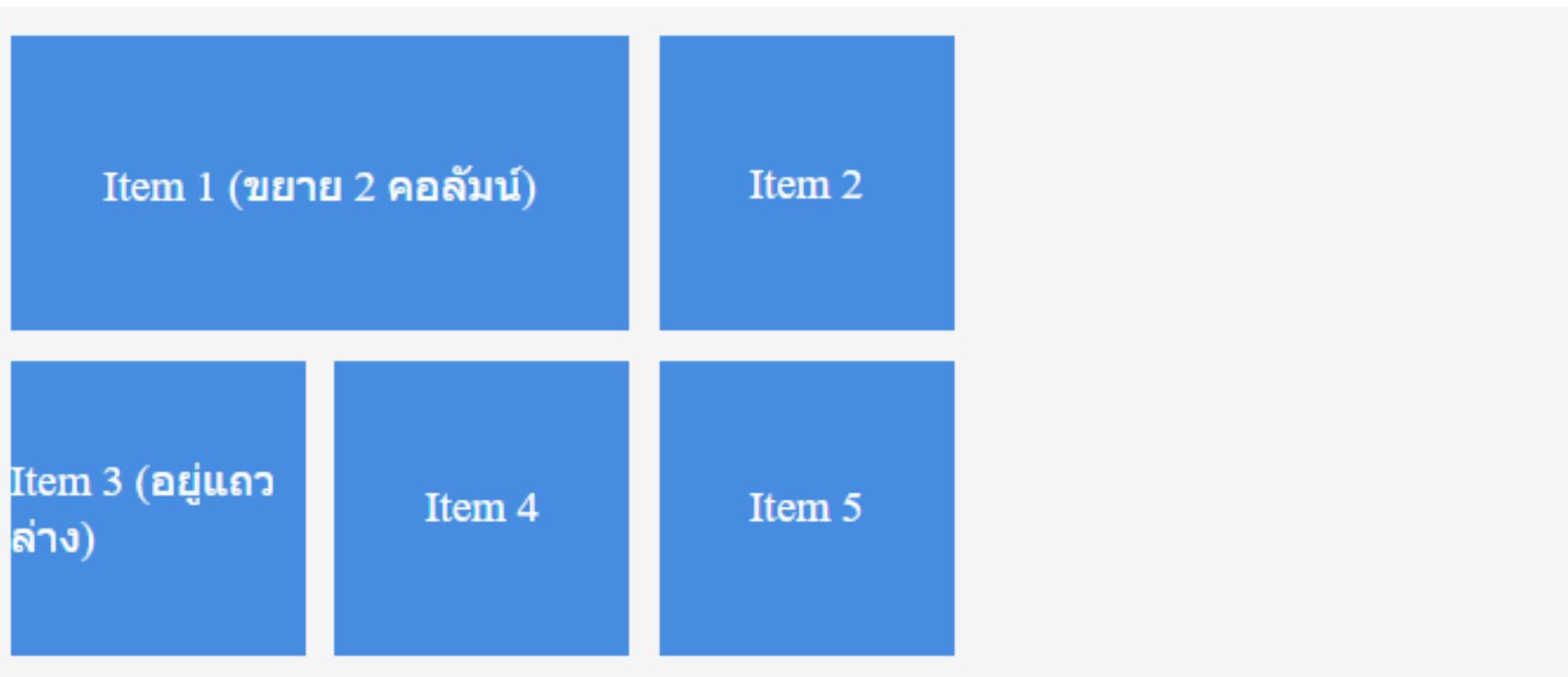
/* Main Content */
content {
  grid-area: main; /*กำหนดให้ content อยู่ในพื้นที่ที่ชื่อ main*/
  background-color: #rgb(129, 0, 129); /*สีส่วนของ content*/
  color: white; text-align: center; line-height: 530px; font-size: 24px;
}

/* Footer */
footer {
  grid-area: footer; /*กำหนดให้ footer อยู่ในพื้นที่ที่ชื่อ footer*/
  background-color: #rgb(0, 129, 86); /*สีส่วนของ footer*/
  color: white; text-align: center; line-height: 70px; font-size: 24px;
}
```

CSS Grid System

CSS Grid System Demo

ตัวอย่างพื้นฐานของ Grid: กำหนดคอลัมน์/แถว และกำหนดตำแหน่งกล่องลูก



```
<div class="grid-container">
    <div class="item item1">Item 1 (ขยาย 2 คอลัมน์)</div>
    <div class="item item2">Item 2</div>
    <div class="item item3">Item 3 (อยู่ແກວລ່າງ)</div>
    <div class="item item4">Item 4</div>
    <div class="item item5">Item 5</div>
</div>
```

```
/* กล่องแม่ที่เป็น Grid */
.grid-container {
    display: grid; /* เปิดโหมด Grid */
    grid-template-columns: 100px 100px 100px; /* 3 คอลัมน์ */
    grid-template-rows: 100px 100px; /* 2 แถว */
    gap: 10px; /* ระยะห่างระหว่างช่อง */
    background: #f5f5f5;
    padding: 10px;
}

/* --กล่องลูก -- */
.item {
    background: #4a90e2;
    color: white;
    font-size: 15px;
    display: flex;
    justify-content: center;
    align-items: center;
}

/* จัดตำแหน่งแบบกำหนดเอง */
/* item1 ให้กินพื้นที่ 2 คอลัมน์ */
.item1 {
    grid-column: 1 / 3; /* เริ่มเส้น 1 สิ้นสุด 3 */
}

/* item3 ให้ลงมาอยู่ແກວລ່າງ */
.item3 {
    grid-row: 2 / 3; /* เริ่มเส้น 2 สิ้นสุด 3 */
}
```

การใช้ CSS Grid และ CSS Flexbox ร่วมกัน



แนวคิดสำคัญ:

- Grid คุณโครงสร้าง
- Flexbox คุณรายละเอียดด้านใน

ทำอะไรในตัวอย่างนี้?

- ใช้ CSS Grid จัดโครงสร้างหลักของเมนู (Logo | Menu)
- ใช้ CSS Flexbox จัดเรียงรายการเมนู (Home, About, ...) ในแนวนอน

ทำไมต้องใช้ร่วมกัน?

- Grid → เก่งเรื่อง “แบ่งพื้นที่ใหญ่” (Layout หลัก)
- Flexbox → เก่งเรื่อง “จัดเรียงของในแนวเดียว/คลัมบ์เดียว”

การใช้ CSS Grid และ CSS Flexbox ร่วมกัน

MyLogo [Home](#) [About Us](#) [Portfolio](#) [Contact](#)

```
/* Grid Container (โครงสร้างหลัก) */
.menu-container {
  display: grid;
  grid-template-columns: 200px 1fr;
  grid-template-rows: 50px;
  align-items: center;
  background-color: #222;
  color: white;
  padding: 10px 20px;
}
```

ทำอะไรในตัวอย่างนี้?

- display: grid → แบ่ง Logo / Menu
- grid-template-columns: 200px 1fr → ช้ายคงที่ ขวาอีกด
- ul { display: flex; } → จัดเมนูเรียงแนวอน
- ใช้ Grid + Flexbox ร่วมกัน = วิธีมาตรฐานในเว็บจริง

```
/* Logo */
.logo {
  text-align: center;
  font-size: 24px;
  font-weight: bold;
}

/* Menu ใช้ Flexbox */
ul {
  display: flex;
  list-style: none;
  align-items: center;
  gap: 20px;
  margin: 0;
  padding: 0;
}

ul li a {
  color: white;
  text-decoration: none;
  font-size: 18px;
}
ul li a:hover {
  text-decoration: underline;
}
```

```
<div class="menu-container">
  <div class="logo">
    MyLogo
  </div>

  <ul>
    <li><a href="#">Home</a></li>
    <li><a href="#">About Us</a></li>
    <li><a href="#">Portfolio</a></li>
    <li><a href="#">Contact</a></li>
  </ul>
</div>
```

Fullscreen divs



เวลาเราต้องการให้ <div> ตัวหนึ่ง เติมจอด้วย 100% กว้างและสูง เช่น หน้าเว็บแบบ Web App หรือหน้า Landing Page ที่ไม่มีการเลื่อน (no scroll) เราต้องใช้ หน่วย % ในการกำหนดขนาด ต้องกำหนดให้ html และ body มีขนาด 100% ก่อนเสมอ

```
<style>
    /* ให้ html และ body มีขนาดเต็มจอ 100% */
    html, body {
        width: 100%;
        height: 100%;
        margin: 0;      /* ลบระยะขอบเริ่มต้น */
        padding: 0;     /* ลบ padding เริ่มต้น */
    }

    /* กล่องหลักให้กินพื้นที่เต็มจอ */
    #main {
        width: 100%;      /* กว้าง 100% ของหน้าจอ */
        height: 100%;     /* สูง 100% ของหน้าจอ */
        background: #4a90e2; /* สีพื้นหลังให้เท็งชัด */
        display: flex;      /* จัดข้อความให้อยู่กลาง */
        justify-content: center;
        align-items: center;
        color: white;
        font-size: 32px;
    }
</style>
```

CSS Position

CSS position ใช้เพื่อ “ควบคุมตำแหน่ง” ขององค์ประกอบ (element) ให้จัดวางอยู่ในตำแหน่งที่ต้องการ มีก็งหมด 4 แบบหลัก ได้แก่

1. static (ค่าเริ่มต้น)

- ทุก element จะอยู่ใน flow ปกติของหน้าเว็บ
- ไม่สามารถใช้ top / left / right / bottom ได้

ใช้เมื่อ: ไม่ได้ต้องการเลื่อนตำแหน่งเป็นพิเศษ

```
div.static {  
    position: static;  
    border: 3px solid #6c8cff;  
}
```

2. relative (ขยับจากตำแหน่งเดิม)

- อยู่ใน flow ปกติ แต่ เลื่อนจากตำแหน่งเดิมของตัวเองได้
- ใช้ top, left, right, bottom เพื่อขยับ

ใช้เมื่อ: ต้องการขยับบิดหน่อย โดยไม่ทำให้ตำแหน่งอื่นเสียรูป

```
div.relative {  
    position: relative;  
    top: 20px; /* ขยับลง */  
    left: 20px; /* ขยับขวา */  
    background: #ff8c42;  
}
```

CSS Position Demo (Basic)

position: static
(อยู่ตำแหน่งปกติ)

position: relative
(ขยับจากตำแหน่งเดิม)

Absolute (loybn parent)

absolute
top:20px; right:10px;

position: fixed
(ดีดหน้าจอ)

CSS Position

3. absolute (หลุดออกจาก flow ปกติ)

- ไม่กินพื้นที่ใน layout (เหมือนลอยออกมา)
- ถูกว่างตำแหน่งจาก parent ที่เป็น relative ใกล้ที่สุด
- ใช้ top, left, right, bottom ໄດ້ເຕີມກໍ

ใช้เมื่อ: ต้องการวาง element แบบ overlay / กับซ้อน / popup

```
.absolute-box {  
    position: absolute;  
    top: 20px;  
    right: 10px;  
    background: #ff4444;  
}
```

3. absolute (หลุดออกจาก flow ปกติ)

- ไม่ขึ้นกับ parent
- เลื่อนตาม scroll หน้าเว็บ
- ติดตามตำแหน่ง viewport เสมอ

ใช้เมื่อ: ต้องการสร้าง Navbar ติดบนสุด/ปุ่มช่วยเหลือ/Popup ลอย

```
.fixed-box {  
    position: fixed;  
    bottom: 20px;  
    right: 20px;  
    width: 150px;  
    background: #22bb33;  
}
```

CSS Position Demo (Basic)

position: static
(อยู่ตำแหน่งปกติ)

position: relative
(ขยับจากตำแหน่งเดิม)

Absolute (ลอยบน parent)

absolute
top:20px; right:10px;

position: fixed
(ติดหน้าจอ)

CSS Position

```
body {  
    font-family: Arial, sans-serif;  
    padding: 20px;  
    height: 1200px; /* เพื่อให้เห็น fixed ตอนเลื่อนหน้า */  
}  
  
.box {  
    width: 200px;  
    height: 80px;  
    background: #4da3ff;  
    padding: 10px;  
    margin-bottom: 20px;  
    color: white;  
    font-size: 18px;  
}
```

```
/* -----  
 static (ค่าเริ่มต้น) ไม่สามารถยับด้วย top/left  
-----*/  
.static-box {  
    position: static;  
    background: #6c8cff;  
}  
  
/* -----  
 relative ทำให้ "ขยับจากตำแหน่งเดิม"  
-----*/  
.relative-box {  
    position: relative;  
    top: 20px; /* ขยับลง */  
    left: 20px; /* ขยับขวา */  
    background: #ff8c42;  
}
```

```
/* -----  
 absolute ลอยอิสระ ว่างอิง *กlostongแมที่เป็น relative*  
-----*/  
.relative-parent {  
    position: relative;  
    width: 300px;  
    height: 150px;  
    background: #ddd;  
    margin-bottom: 40px;  
    border: 2px dashed gray;  
}  
  
.absolute-box {  
    position: absolute;  
    top: 20px;  
    right: 10px;  
    background: #ff4444;  
}  
  
/* -----  
 fixed ติดอยู่บนหน้าจอ ไม่เลื่อนตาม scroll  
-----*/  
.fixed-box {  
    position: fixed;  
    bottom: 20px;  
    right: 20px;  
    width: 150px;  
    background: #22bb33;  
}
```

html

```
<!-- ★ static -->  
<div class="box static-box">  
    position: static<br>(อยู่ตำแหน่งปกติ)  
</div>  
  
<!-- ★ relative -->  
<div class="box relative-box">  
    position: relative<br>(ขยับจากตำแหน่งเดิม)  
</div>  
  
<!-- ★ absolute --><br>  
<h2>Absolute (ลอยบน parent)</h2>  
<div class="relative-parent">  
    <div class="box absolute-box">  
        absolute<br>top:20px; right:10px;  
    </div>  
</div>  
  
<!-- ★ fixed -->  
<div class="box fixed-box">  
    position: fixed<br>(ติดหน้าจอ)  
</div>
```

การใช้ CSS Classes

เราสามารถใช้หลาย Class ใน Element เดียวเพื่อประกอบสไตล์ต่างๆ เช้าด้วยกัน

CSS

```
<style>
/* กล่องหลัก */
.card {
    background: white;
    padding: 20px;
    border-radius: 10px;
}

/* เงาของкар์ด */
.shadow {
    box-shadow: 0 4px 12px rgba(0,0,0,0.15);
}

/* ปุ่มพื้นฐาน */
.btn {
    padding: 10px 20px;
    border: none;
    cursor: pointer;
}

/* ปุ่มแบบ primary */
.btn-primary {
    background: #007bff;
    color: white;
    border-radius: 6px;
}
</style>
```

html

```
<div class="card shadow">    <!-- ใช้ 2 คลาสพร้อมกัน -->
    <h2 class="card-title">หัวข้อการ์ด</h2>
    <p class="card-text">เนื้อหาในการ์ด</p>

    <!-- ปุ่นใช้คลาส btn + btn-primary -->
    <button class="btn btn-primary">ปุ่ม</button>
</div>
```

หัวข้อการ์ด

เนื้อหาในการ์ด

ปุ่ม

CSS Specificity

Specificity คือ “น้ำหนักของตัวเลือก (Selector)” ใช้ตัดสินว่า เมื่อมี CSS หลายตัวมาชนกัน ตัวไหนจะถูกนำไปใช้จริง

เบราว์เซอร์จะให้ความสำคัญกับ ตัวเลือกที่เฉพาะเจาะจงกว่า เสมอ
➡ จัดเกิดลำดับความสำคัญ (Priority) เช่น Inline > ID > Class > Element

ลำดับความสำคัญของ CSS (มาก → น้อย)

1. **Inline Styles** เขียนใน element โดยตรง เช่น

```
<p id="mainText" class="text" style="color: red;">  
    ข้อความนี้มี style ชนกันทั้ง 4 แบบ!  
</p>
```

3. **Class Selector**

```
.text { color: green; }
```

2. **ID Selector** ใช้ # นำหน้า เช่น

```
#mainText { color: blue; }
```

4. **Element Selector**

```
<p style="color : yellow;">
```

CSS Specificity

```
<style>
    /* Element selector → คะแนนน้อยสุด */
    p {
        color: yellow;
    }

    /* Class selector → ชั้น Element */
    .text {
        color: green;
    }

    /* ID selector → ชั้น class */
    #mainText {
        color: blue;
    }
</style>

</head>
<body>
<h2>CSS Specificity ┌ ตัวอย่างการชนกันทั้ง 4 แบบ</h2>

<p id="mainText" class="text" style="color: red;">
    ข้อความนี้มี style ชนกันทั้ง 4 แบบ!
</p>

</body>
```

CSS Specificity – ตัวอย่างการชนกันทั้ง 4 แบบ

ข้อความนี้มี style ชนกันทั้ง 4 แบบ!