JAVASCRIPT

WEB PROGRAMMING

(INTEGRATIVE-GENERATIVE AI EDITION)

Student Price Book Center

คำนำ

ในยุคที่เทคโนโลยีเว็บก้าวหน้าอย่างรวดเร็ว JavaScript ได้กลายเป็นหนึ่งในภาษาหลักที่ขับเคลื่อนโลกของการพัฒนาเว็บอย่างแท้จริง ตั้งแต่การสร้างปฏิสัมพันธ์พื้นฐานบนหน้าเว็บ ไปจนถึงการพัฒนาแอปพลิเคชันที่ซับซ้อนทั้งฝั่งผู้ใช้ (Frontend) และฝั่งเซิร์ฟเวอร์ (Backend) ภาษา JavaScript ได้พิสูจน์ตัวเองว่าเป็นเครื่องมือสำคัญที่นักพัฒนายุคใหม่ขาดไม่ได้

หนังสือเล่มนี้จัดทำขึ้นเพื่อเป็นแนวทางสำหรับผู้ที่ต้องการเริ่มต้นเรียนรู้ JavaScript อย่างเป็นระบบ ตั้งแต่ระดับ **พื้นฐาน (Beginner)** ไปจนถึงระดับ **ขั้นสูง (Advanced)** พร้อมนำไปประยุกต์ใช้ในการพัฒนาเว็บแอปพลิเคชันจริง โดยเนื้อหาครอบคลุม 4 หัวข้อหลัก ได้แก่

1. **พื้นฐานของ JavaScript** – ทำความเข้าใจโครงสร้างของภาษา การใช้งานร่วมกับ HTML และ CSS การเขียนฟังก์ชันเบื้องต้น รวมถึงการจัดการข้อมูลประเภทต่างๆ
2. **Intermediate JavaScript** – เจาะลึกแนวคิดสำคัญ เช่น Scope, Closure, การจัดการ DOM และการทำงานกับ API ผ่าน Fetch/AJAX
3. **Advanced JavaScript** – ศึกษาแนวคิดเชิงลึกเกี่ยวกับ Asynchronous JavaScript, Functional Programming, OOP, และเทคนิคการเพิ่มประสิทธิภาพ
4. **JavaScript สำหรับ Web Development** – ประยุกต์ใช้ความรู้ JavaScript กับการพัฒนา Frontend Framework (เช่น React, Vue) และ Backend (Node.js, Express) รวมถึงการเชื่อมต่อฐานข้อมูลและสร้าง REST API

เพื่อให้ผู้อ่านสามารถนำความรู้ไปใช้ได้จริง หนังสือเล่มนี้มี **แบบฝึกหัดและโปรเจกต์ตัวอย่าง** ในแต่ละบทเรียน พร้อมทั้งแนะนำแหล่งเรียนรู้ออนไลน์ที่เชื่อถือได้ เช่น MDN, JavaScript.info, freeCodeCamp และช่อง YouTube ยอดนิยม

เนื้อหาทั้งหมดถูกเรียบเรียงอย่างเป็นลำดับขั้น เหมาะสำหรับทั้งผู้เริ่มต้นที่ไม่มีพื้นฐานมาก่อน ไปจนถึงผู้ที่ต้องการต่อยอดสู่การเป็นนักพัฒนา JavaScript มืออาชีพ ด้วยแนวทางการเรียนรู้ที่ชัดเจน ครบถ้วน และสามารถปฏิบัติตามได้จริง

ขอให้หนังสือเล่มนี้เป็นเสมือน “คู่มือเดินทาง” ที่ช่วยให้ผู้อ่านทุกท่านได้ก้าวไปสู่โลกของ JavaScript อย่างมั่นใจและเข้าใจลึกซึ้ง

ด้วยความปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

หน้า

บทที่ 1

พื้นฐานจาวาสคริปต์

(JavaScript Basic)

เนื้อหา

- JavaScript คืออะไร?
- ข้อมูลเพิ่มเติมเกี่ยวกับ JavaScript
- ประวัติความเป็นมาของ JavaScript
- การแทรก JavaScript ลงใน HTML (<script> tag) คืออะไร?
- การใช้งาน Console และ Developer Tools คืออะไร?
- การใช้งาน Console และ Developer Tools อย่างละเอียด

JavaScript คือภาษาสคริปต์ที่ทำงานฝั่งผู้ใช้ (client-side) ซึ่งใช้เพื่อเพิ่มความสามารถในการโต้ตอบ (interactivity) ให้กับหน้าเว็บ โดยทำงานร่วมกับ HTML ที่ใช้สร้างโครงสร้างของเว็บ และ CSS ที่ใช้กำหนดรูปแบบการแสดงผล JavaScript ช่วยให้เว็บเพจสามารถตอบสนองต่อการกระทำของผู้ใช้ เช่น การคลิกปุ่ม การกรอกแบบฟอร์ม หรือการเลื่อนเมาส์ โดยสามารถแทรก JavaScript ลงใน HTML ได้โดยตรงผ่านแท็ก <script> ซึ่งอาจวางไว้ในส่วน <head> หรือท้าย <body> เพื่อควบคุมการทำงานของหน้าเว็บ

นอกจากนี้ นักพัฒนาสามารถใช้ **Console** ซึ่งเป็นส่วนหนึ่งของ **Developer Tools** ในเว็บเบราว์เซอร์ เพื่อเขียน ทดสอบ และดีบักโค้ด JavaScript ได้สะดวกยิ่งขึ้น Console ช่วยให้สามารถดูผลลัพธ์ของคำสั่งได้ทันที ตรวจสอบค่าตัวแปร และตรวจจับข้อผิดพลาดที่เกิดขึ้นขณะทำงาน การใช้งาน Developer Tools ถือเป็นเครื่องมือพื้นฐานที่ช่วยให้การพัฒนาเว็บไซต์มีประสิทธิภาพมากขึ้น โดยเฉพาะในขั้นตอนของการทดสอบและแก้ไขปัญหา.

JavaScript คืออะไร?

JavaScript (JS) เป็นภาษาโปรแกรมที่ใช้พัฒนา **เว็บแอปพลิเคชัน** โดยสามารถทำงานได้ทั้งฝั่ง **Frontend** (เบราว์เซอร์) และ **Backend** (เซิร์ฟเวอร์)

- **Frontend (ฝั่งผู้ใช้)** ⟶ ใช้ร่วมกับ **HTML** และ **CSS** เพื่อสร้างเว็บไซต์ที่โต้ตอบได้ เช่น การเปลี่ยนเนื้อหาแบบไดนามิก, การตรวจสอบฟอร์ม, การสร้างเอฟเฟกต์อนิเมชัน ฯลฯ
- **Backend (ฝั่งเซิร์ฟเวอร์)** ⟶ ใช้กับ **Node.js** เพื่อพัฒนา API หรือจัดการฐานข้อมูล เช่น MongoDB, MySQL

ทำไมต้องใช้ JavaScript?

JavaScript เป็นภาษาที่ **จำเป็น** ต่อการพัฒนาเว็บและแอปพลิเคชันเพราะเหตุผลต่อไปนี้

1. ทำให้เว็บโต้ตอบกับผู้ใช้ได้ (Interactivity)

- HTML และ CSS ใช้แค่แสดงผล **แต่ไม่สามารถโต้ตอบกับผู้ใช้ได้**
- JavaScript ช่วยเพิ่มความสามารถ เช่น
  - คลิกปุ่มแล้วเปลี่ยนเนื้อหา
  - ตรวจสอบข้อมูลในแบบฟอร์ม (Validation)
  - โหลดข้อมูลจากเซิร์ฟเวอร์โดยไม่ต้องรีเฟรชหน้าเว็บ (**AJAX**)

☐ **ตัวอย่าง** คลิกปุ่มเพื่อเปลี่ยนข้อความ

```
<!DOCTYPE html>
<html>
<body>
  <p id="text">Hello, JavaScript!</p>
  <button onclick="changeText()">คลิกเพื่อเปลี่ยนข้อความ</button>

  <script>
    function changeText() {
      document.getElementById("text").innerHTML = "คุณกดปุ่มแล้ว!";
    }
  </script>
</body>
</html>
```

---

2. ทำงานร่วมกับ HTML และ CSS ได้ดี

JavaScript ช่วยควบคุมโครงสร้าง (HTML) และการแสดงผล (CSS) ของเว็บ เช่น

- เปลี่ยนสีพื้นหลังหรือขนาดตัวอักษรโดยอัตโนมัติ
- ซ่อนหรือแสดงองค์ประกอบในหน้าเว็บ

☐ **ตัวอย่าง** เปลี่ยนสีพื้นหลังเมื่อคลิกปุ่ม

```
<!DOCTYPE html>
<html>
<body>
  <button onclick="changeColor()">เปลี่ยนสีพื้นหลัง</button>

  <script>
```

```
    function changeColor() {
      document.body.style.backgroundColor = "lightblue";
    }
  </script>
</body>
</html>
```

---

3. ใช้พัฒนาเว็บแอปพลิเคชันที่โหลดเร็ว (Single Page Application - SPA)

- ปกติเมื่อเปลี่ยนหน้าเว็บ เบราว์เซอร์ต้องโหลดใหม่ทั้งหมด
- แต่ **JavaScript + AJAX** ช่วยให้โหลดเฉพาะส่วนที่เปลี่ยนแปลง ทำให้เว็บเร็วขึ้น
- ใช้กับ **React, Vue.js, Angular** เพื่อสร้าง **Single Page Application (SPA)**

☐ **ตัวอย่าง** ดึงข้อมูลจากเซิร์ฟเวอร์โดยไม่ต้องรีเฟรช

```
<!DOCTYPE html>
<html>
<body>
  <button onclick="loadData()">โหลดข้อมูล</button>
  <p id="result"></p>

  <script>
    function loadData() {
      fetch("https://jsonplaceholder.typicode.com/posts/1")
        .then(response => response.json())
        .then(data => {
          document.getElementById("result").innerHTML = data.title;
        });
    }
  </script>
</body>
</html>
```

---

4. ใช้งานได้ทั้งฝั่งหน้าเว็บ (Frontend) และเซิร์ฟเวอร์ (Backend)

- **Frontend** ⟶ ใช้ควบคุมการแสดงผล
- **Backend** ⟶ ใช้กับ **Node.js** เพื่อพัฒนา API, จัดการฐานข้อมูล

☐ **ตัวอย่าง** Node.js ใช้สร้างเซิร์ฟเวอร์

```
const http = require("http");

const server = http.createServer((req, res) => {
  res.writeHead(200, { "Content-Type": "text/plain" });
  res.end("Hello from Node.js Server");
});

server.listen(3000, () => console.log("Server running at http://localhost:3000"));
```

---

5. มีไลบรารีและเฟรมเวิร์กมากมายให้ใช้งาน

- **React.js, Vue.js, Angular** ⟶ พัฒนาเว็บแอป
- **Node.js, Express.js** ⟶ พัฒนา Backend
- **Three.js** ⟶ พัฒนา 3D กราฟิก
- **Chart.js, D3.js** ⟶ สร้างกราฟและแผนภูมิ

 ตัวอย่าง ใช้ **Chart.js** วาดกราฟ

```
<canvas id="myChart"></canvas>
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
<script>
  const ctx = document.getElementById("myChart").getContext("2d");
  new Chart(ctx, {
    type: "bar",
    data: {
      labels: ["A", "B", "C"],
      datasets: [{ data: [10, 20, 30], backgroundColor: ["red", "blue", "green"] }]
    }
  });
</script>
```

สรุป

 JavaScript เป็นภาษาจำเป็น ในการพัฒนาเว็บ เพราะช่วยให้เว็บโต้ตอบกับผู้ใช้ได้ ทำงานร่วมกับ HTML และ CSS ใช้งานได้ทั้ง **Frontend และ Backend** และมีเครื่องมือมากมายที่ช่วยให้พัฒนาแอปได้ง่ายขึ้น

JavaScript สำคัญเพราะว่า

1. **ทำให้เว็บไซต์โต้ตอบได้** (เปลี่ยนเนื้อหา, โหลดข้อมูล AJAX)

2. **ควบคุม HTML และ CSS ได้** (เปลี่ยนสี, ซ่อน/แสดงองค์ประกอบ)
3. **ช่วยให้เว็บเร็วขึ้นด้วย SPA** (React, Vue, Angular)
4. **พัฒนา Backend ได้ด้วย Node.js**
5. **มีไลบรารีและเฟรมเวิร์กที่ช่วยพัฒนาแอปได้ง่ายขึ้น**

ข้อมูลเพิ่มเติมเกี่ยวกับ JavaScript

1. JavaScript เป็นภาษาแบบไดนามิก

- เป็นภาษาที่ไม่ต้องกำหนดชนิดข้อมูล (loosely typed)
- เปลี่ยนค่าตัวแปรได้โดยไม่ต้องระบุประเภท
- รองรับ **Object-Oriented Programming (OOP)** และ **Functional Programming**

2. การทำงานของ JavaScript บนเว็บ

- **Client-Side (ฝั่งเบราว์เซอร์)** ⟶ รันโดยตรงบนเบราว์เซอร์ เช่น Chrome, Firefox, Edge
- **Server-Side (ฝั่งเซิร์ฟเวอร์)** ⟶ ใช้ร่วมกับ **Node.js** เพื่อสร้าง API หรือเชื่อมต่อฐานข้อมูล
- ทำงานแบบ **Single-Threaded** แต่รองรับ **Asynchronous Processing** ด้วย **Promises และ async/await**

3. เทคโนโลยีที่เกี่ยวข้องกับ JavaScript

- **AJAX (Asynchronous JavaScript and XML)** ⟶ โหลดข้อมูลจากเซิร์ฟเวอร์โดยไม่ต้องรีเฟรชหน้า
- **JSON (JavaScript Object Notation)** ⟶ รูปแบบข้อมูลที่ใช้แลกเปลี่ยนระหว่างเซิร์ฟเวอร์กับไคลเอนต์
- **Web APIs** เช่น Local Storage, Fetch API, WebSockets

4. ไลบรารีและเฟรมเวิร์กที่ได้รับความนิยม

- **Frontend:** React.js, Vue.js, Angular
- **Backend:** Node.js, Express.js
- **UI Frameworks:** Bootstrap, Tailwind CSS
- **Data Visualization:** Chart.js, D3.js

5. ความสำคัญของ JavaScript ในยุคปัจจุบัน

- เป็นภาษาหลักของ **เว็บแอปพลิเคชัน**
- รองรับ **Progressive Web Apps (PWA)** ซึ่งทำให้เว็บใช้งานเหมือนแอปมือถือ
- ใช้ใน **Full Stack Development** ได้ทั้งฝั่งหน้าเว็บและเซิร์ฟเวอร์
- รองรับ **Machine Learning และ AI** ผ่านไลบรารีเช่น TensorFlow.js

ประวัติความเป็นมาของ JavaScript

1. จุดกำเนิดของ JavaScript

JavaScript ถูกสร้างขึ้นในปี **1995** โดย **Brendan Eich** ซึ่งเป็นนักพัฒนาที่ทำงานให้กับ **Netscape Communications** บริษัทที่เป็นผู้พัฒนาเว็บเบราว์เซอร์ **Netscape Navigator**

- ในยุคนั้น **เว็บยังไม่มีความโต้ตอบมากนัก** เนื่องจากใช้เพียง **HTML** และ **CSS** ซึ่งแค่แสดงข้อมูลแบบคงที่
- Netscape ต้องการภาษาสคริปต์ที่ **เขียนง่าย** และสามารถ **ทำให้เว็บโต้ตอบกับผู้ใช้ได้**
- Brendan Eich ถูกมอบหมายให้พัฒนาภาษานี้ภายในเวลาเพียง **10 วัน**

ผลลัพธ์ที่ได้คือ **JavaScript** ซึ่งในตอนแรกมีชื่อว่า **Mocha** และภายหลังเปลี่ยนเป็น **LiveScript** ก่อนจะเปลี่ยนชื่อเป็น **JavaScript** เพื่อการตลาด เนื่องจากในขณะนั้น **Java** (ของบริษัท Sun Microsystems) กำลังได้รับความนิยม

---

2. JavaScript กับยุคเบราว์เซอร์แข่งขัน (Browser Wars)

ในช่วงปลายยุค 1990s เว็บเบราว์เซอร์สองตัวที่แข่งขันกันอย่างดุเดือดคือ

- **Netscape Navigator** (ของ Netscape)
- **Internet Explorer** (ของ Microsoft)

Microsoft เล็งเห็นความสำคัญของ JavaScript และพัฒนาเวอร์ชันของตัวเองชื่อ **JScript** ซึ่งทำให้เกิด **ปัญหาความเข้ากันได้** ระหว่างเบราว์เซอร์ต่างๆ

---

3. การมาตรฐานของ JavaScript (ECMAScript)

เนื่องจากปัญหาการใช้งานที่แตกต่างกันระหว่างเบราว์เซอร์ JavaScript จึงถูกส่งไปให้ **ECMA International** องค์กรที่กำหนดมาตรฐานสำหรับภาษาโปรแกรม

- ในปี **1997** ได้มีการสร้างมาตรฐานชื่อ **ECMAScript (ES1)**
- JavaScript เวอร์ชันที่ใช้งานจึงอ้างอิงจาก ECMAScript
- เวอร์ชันที่สำคัญ ได้แก่
    - **ES5 (2009)** ⟶ รองรับ JSON, Strict Mode
    - **ES6 (2015)** ⟶ มี let, const, Arrow Functions, Promises
    - **ESNext (2016-ปัจจุบัน)** ⟶ มีการอัปเดตภาษาอย่างต่อเนื่องทุกปี

---

4. ยุคของ JavaScript สมัยใหม่

หลังจากปี 2010 JavaScript กลายเป็นภาษาสำคัญของเว็บแอปพลิเคชัน โดยมีการพัฒนาไลบรารีและเฟรมเวิร์กต่างๆ เช่น

- **Node.js (2009)** ⟶ ทำให้ JavaScript ใช้งานฝั่งเซิร์ฟเวอร์ได้
- **jQuery (2006)** ⟶ ช่วยให้การจัดการ DOM ง่ายขึ้น

- **Angular (2010), React (2013), Vue.js (2014)** ⟶ เฟรมเวิร์กยอดนิยมสำหรับสร้างเว็บแอป

---

5. บทบาทของ JavaScript ในปัจจุบัน

JavaScript เป็นภาษาที่ได้รับความนิยมสูงสุดในโลก โดยใช้ทั้งใน

- **Frontend (เว็บแอปพลิเคชัน)** ⟶ React, Vue.js, Angular
- **Backend (เซิร์ฟเวอร์)** ⟶ Node.js, Express.js
- **Mobile Apps** ⟶ React Native, Ionic
- **Machine Learning** ⟶ TensorFlow.js
- **Game Development** ⟶ Three.js, Babylon.js

---

6. สรุป

JavaScript เป็นภาษาที่พัฒนามาไกลจากการเป็นเพียงสคริปต์บนเบราว์เซอร์จนกลายเป็นภาษาหลักของ **Full Stack Development** ใช้ได้ทั้งฝั่ง **Frontend และ Backend** และมีบทบาทสำคัญในการพัฒนาเทคโนโลยีเว็บสมัยใหม่

ข้อมูลเพิ่มเติมเกี่ยวกับประวัติของ JavaScript

1. แรงบันดาลใจในการสร้าง JavaScript

ในช่วงต้นทศวรรษ 1990s อินเทอร์เน็ตกำลังเติบโตขึ้นอย่างรวดเร็ว แต่เว็บยังมีข้อจำกัดด้านการโต้ตอบกับผู้ใช้ เช่น

- เว็บเพจในยุคนั้นใช้ **HTML และ CSS** เท่านั้น ทำให้เป็นเพียงหน้าเว็บแบบคงที่
- ต้องใช้ภาษาโปรแกรมที่ซับซ้อน เช่น **Java หรือ Perl** เพื่อสร้างฟีเจอร์แบบไดนามิก แต่การใช้งานยุ่งยาก

Netscape ต้องการภาษาสคริปต์ที่ **เรียนรู้ง่าย** และ **ใช้งานในเบราว์เซอร์ได้โดยตรง** เพื่อให้ผู้ใช้สามารถโต้ตอบกับเว็บได้ทันที โดยไม่ต้องรีเฟรชหน้า

---

2. พัฒนาการของ JavaScript ตามเวอร์ชัน ECMAScript

ยุคแรก (1995-2008)

- **1995:** JavaScript ถูกสร้างขึ้นโดย Brendan Eich
- **1997:** ECMAScript 1 (ES1) ⟶ มาตรฐานแรกของ JavaScript
- **1999:** ECMAScript 3 (ES3) ⟶ เพิ่มคุณสมบัติเช่น Regular Expressions และ Error Handling
- **2005:** AJAX ทำให้เว็บโต้ตอบแบบเรียลไทม์โดยไม่ต้องโหลดหน้าใหม่
- **2006:** jQuery ถูกพัฒนา ทำให้ JavaScript ใช้งานง่ายขึ้น

- **2009:** ECMAScript 5 (ES5) ⟶ รองรับ JSON, เพิ่ม strict mode

ยุคสมัยใหม่ (2015-ปัจจุบัน)

- **2015:** ECMAScript 6 (ES6) ⟶ การเปลี่ยนแปลงครั้งใหญ่ เช่น let, const, Arrow Functions, Promises
- **2017:** ECMAScript 8 (ES8) ⟶ เพิ่ม async/await ทำให้การเขียนโค้ดแบบ asynchronous ง่ายขึ้น
- **2020:** ECMAScript 11 (ES11) ⟶ เพิ่ม Nullish Coalescing (??), Optional Chaining (?.)
- **ปัจจุบัน:** มีการอัปเดต ECMAScript ทุกปี ทำให้ JavaScript มีฟีเจอร์ใหม่อย่างต่อเนื่อง

---

3. การแข่งขันของ JavaScript กับภาษาอื่น ๆ

- **JavaScript vs Java:** แม้ชื่อคล้ายกัน แต่เป็นภาษาที่แตกต่างกัน Java ใช้สำหรับแอปพลิเคชันขนาดใหญ่ ขณะที่ JavaScript ใช้กับเว็บ
- **JavaScript vs Python:** Python ใช้งานง่ายกว่าสำหรับงาน Data Science และ AI แต่ JavaScript เหมาะกับเว็บแอปพลิเคชัน
- **JavaScript vs TypeScript:** TypeScript เป็น JavaScript ที่เพิ่ม **Static Typing** เพื่อช่วยลดข้อผิดพลาดของโค้ด

---

4. เทคโนโลยีที่ JavaScript รองรับในปัจจุบัน

- **Progressive Web Apps (PWA):** ทำให้เว็บทำงานได้เหมือนแอปมือถือ
- **Serverless Computing:** ใช้ JavaScript กับแพลตฟอร์มเช่น AWS Lambda และ Firebase Functions
- **WebAssembly (WASM):** ทำให้เว็บสามารถรันโค้ดจากภาษาอื่น (C++, Rust) ได้เร็วขึ้น
- **Machine Learning บนเว็บ:** ใช้ TensorFlow.js สำหรับสร้าง AI บนเบราว์เซอร์

---

5. อนาคตของ JavaScript

- มีการพัฒนา **Web APIs** ใหม่ ๆ เช่น WebXR (Virtual Reality)
- Node.js ยังคงได้รับความนิยมในการพัฒนา Backend
- TypeScript กำลังเป็นที่นิยมมากขึ้นในหมู่บริษัทใหญ่
- WebAssembly อาจช่วยให้ JavaScript ทำงานร่วมกับภาษาอื่นได้ดีขึ้น

การแทรก JavaScript ลงใน HTML (<script> tag) คืออะไร?

<script> เป็นแท็กที่ใช้ในการฝัง JavaScript ลงในหน้าเว็บ HTML ทำให้เว็บสามารถทำงานแบบไดนามิกและโต้ตอบกับผู้ใช้ได้

รูปแบบการใช้งาน <script> ใน HTML

มี 3 วิธีหลักในการแทรก JavaScript ลงใน HTML:

1. **เขียน JavaScript ภายในแท็ก <script> ในไฟล์ HTML** (Internal Script)
2. **เรียกใช้ไฟล์ JavaScript ภายนอก** (External Script)
3. **ใช้ JavaScript ภายในแท็ก HTML โดยตรง (Inline Script)**

---

1. เขียน JavaScript ภายในแท็ก <script> ในไฟล์ HTML

เป็นการเขียนโค้ด JavaScript โดยตรงภายในไฟล์ HTML ใช้แท็ก <script> และใส่ JavaScript ไว้ข้างใน

ตัวอย่าง

```
<!DOCTYPE html>
<html lang="th">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ตัวอย่าง JavaScript ภายใน HTML</title>
</head>
<body>
    <h1>สวัสดี, ผู้ใช้!</h1>
    <button onclick="sayHello()">คลิกที่นี่</button>

    <script>
        function sayHello() {
            alert("Hello, ยินดีต้อนรับสู่เว็บไซต์!");
        }
    </script>
</body>
</html>
```

อธิบายโค้ด

1. <button onclick="sayHello()"> ⟶ ปุ่มที่เรียกใช้ฟังก์ชัน sayHello() เมื่อกด
2. <script> ... </script> ⟶ บรรจุโค้ด JavaScript
3. function sayHello() { alert("Hello, ยินดีต้อนรับสู่เว็บไซต์!"); }
    - ฟังก์ชัน sayHello() ทำให้เกิดกล่องแจ้งเตือน (alert) เมื่อกดปุ่ม

- **ข้อดี**: ใช้งานง่าย เหมาะกับสคริปต์ขนาดเล็ก
- **ข้อเสีย**: ถ้าสคริปต์ยาว อาจทำให้ HTML ดูรก

---

2. ใช้ไฟล์ JavaScript ภายนอก (External Script)

เป็นวิธีที่ดีที่สุด เพราะช่วยแยกโค้ด JavaScript ออกจาก HTML ทำให้โค้ดอ่านง่ายและจัดการได้ง่ายขึ้น

ตัวอย่าง

ไฟล์ HTML (index.html)

```
<!DOCTYPE html>
<html lang="th">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ตัวอย่าง JavaScript ภายนอก</title>
    <script src="script.js" defer></script>
</head>
<body>
    <h1>ยินดีต้อนรับ!</h1>
    <button id="myButton">กดเพื่อดูข้อความ</button>
</body>
</html>
```

ไฟล์ JavaScript (script.js)

```
document.addEventListener("DOMContentLoaded", function() {
    document.getElementById("myButton").addEventListener("click", function() {
        alert("Hello, ยินดีต้อนรับ!");
    });
});
```

อธิบายโค้ด

1. **ในไฟล์ HTML**
    - <script src="script.js" defer></script> ⟶ โหลดไฟล์ script.js และใช้ defer เพื่อให้โค้ดโหลดหลังจาก HTML โหลดเสร็จ
    - <button id="myButton">กดเพื่อดูข้อความ</button> ⟶ ปุ่มที่รอให้ JavaScript จัดการ
2. **ในไฟล์ script.js**

- document.addEventListener("DOMContentLoaded", function() {...}) ⟶ ทำให้แน่ใจว่า JavaScript รันหลังจากหน้าเว็บโหลดเสร็จ
- document.getElementById("myButton").addEventListener("click", function() {...}) ⟶ จับเหตุการณ์เมื่อกดปุ่ม
- alert("Hello, ยินดีต้อนรับ!"); ⟶ แสดงกล่องแจ้งเตือน

☐ **ข้อดี**:

☐ ทำให้ HTML เป็นระเบียบ

☐ ใช้ซ้ำได้หลายหน้า

☐ แก้ไข JavaScript ได้ง่าย

☐ **ข้อเสีย**:

☐ ต้องใช้ <script src="script.js" defer> หรือใส่ <script> ไว้ท้าย <body> เพื่อป้องกันปัญหาโหลดช้า

---

3. ใช้ JavaScript ภายในแท็ก HTML โดยตรง (Inline Script)

เป็นการใช้ JavaScript โดยตรงในแอตทริบิวต์ HTML เช่น onclick หรือ onmouseover

ตัวอย่าง

```
<!DOCTYPE html>
<html lang="th">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ตัวอย่าง Inline JavaScript</title>
</head>
<body>
    <button onclick="alert('Hello, ยินดีต้อนรับ!')">กดเพื่อแสดงข้อความ</button>
</body>
</html>
```

อธิบายโค้ด

1. <button onclick="alert('Hello, ยินดีต้อนรับ!')"> ⟶ เมื่อกดปุ่ม จะเรียกใช้ฟังก์ชัน alert() ทันที

☐ **ข้อดี**:

☐ ง่ายและเร็ว

☐ เหมาะกับโค้ดสั้นๆ

☐ **ข้อเสีย**:

☐ ทำให้โค้ด HTML ดูรก

☐ ไม่สามารถจัดการโค้ดได้ดี ถ้ามีหลายองค์ประกอบ

☐ **สรุป**

วิธี	คำอธิบาย	ข้อดี	ข้อเสีย
Internal Script	เขียนโค้ด JavaScript ภายใน <script>	ใช้งานง่าย, ไม่ต้องแยกไฟล์	ถ้าโค้ดยาวจะทำให้ HTML รก
External Script	ใช้ไฟล์ .js แยกต่างหาก	โค้ดสะอาด, ใช้ซ้ำได้	ต้องมีการโหลดไฟล์เพิ่ม
Inline Script	เขียน JavaScript ในแอตทริบิวต์ HTML	ใช้ได้ทันที	ทำให้ HTML ยุ่งเหยิง

☐ **แนะนำ:** ควรใช้ **External Script** เป็นหลัก เพื่อให้โค้ดสะอาดและจัดการง่าย

ตัวอย่างการแทรก **JavaScript** ทั้ง **3 วิธี** (แบบละ 3 โปรแกรม) พร้อมคำอธิบาย

---

1. Internal JavaScript (เขียนใน <script> ของ HTML)

☐ **ตัวอย่างที่ 1: เปลี่ยนข้อความเมื่อกดปุ่ม**

```
<!DOCTYPE html>
<html lang="th">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>เปลี่ยนข้อความ</title>
</head>
<body>
    <p id="text">ข้อความเดิม</p>
    <button onclick="changeText()">เปลี่ยนข้อความ</button>

    <script>
        function changeText() {
            document.getElementById("text").innerHTML = "ข้อความใหม่!";
        }
    </script>
</body>
</html>
```

☐ **อธิบาย:** เมื่อกดปุ่ม ฟังก์ชัน changeText() จะเปลี่ยนข้อความใน <p> เป็น "ข้อความใหม่!"

☐ **ตัวอย่างที่ 2: เปลี่ยนสีพื้นหลัง**

```
<!DOCTYPE html>
<html lang="th">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>เปลี่ยนสีพื้นหลัง</title>
</head>
<body>
    <button onclick="changeColor()">เปลี่ยนสีพื้นหลัง</button>

    <script>
        function changeColor() {
            document.body.style.backgroundColor = "lightblue";
        }
    </script>
</body>
</html>
```

☐ **อธิบาย:** เมื่อกดปุ่ม หน้าเว็บจะเปลี่ยนสีพื้นหลังเป็นสีฟ้า

☐ **ตัวอย่างที่ 3: นับจำนวนครั้งที่กดปุ่ม**

```
<!DOCTYPE html>
<html lang="th">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>นับจำนวนคลิก</title>
</head>
<body>
    <button onclick="countClicks()">คลิกฉัน</button>
    <p>คุณกดปุ่ม <span id="count">0</span> ครั้ง</p>
```

```
    <script>
        let count = 0;
        function countClicks() {
            count++;
            document.getElementById("count").innerText = count;
        }
    </script>
</body>
</html>
```

☐ **อธิบาย:** กดปุ่มแล้ว ตัวเลขใน จะเพิ่มขึ้นเรื่อยๆ

---

2. External JavaScript (แยกไฟล์ .js ออกไปต่างหาก)

☐ **ตัวอย่างที่ 1: แสดงกล่องแจ้งเตือน**

ไฟล์ index.html

```
<!DOCTYPE html>
<html lang="th">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>แสดงข้อความแจ้งเตือน</title>
    <script src="script.js" defer></script>
</head>
<body>
    <button id="alertButton">กดเพื่อดูแจ้งเตือน</button>
</body>
</html>
```

ไฟล์ script.js

```
document.addEventListener("DOMContentLoaded", function() {
    document.getElementById("alertButton").addEventListener("click", function() {
        alert("Hello, ยินดีต้อนรับ!");
    });
});
```

☐ **อธิบาย:** กดปุ่มแล้วแสดงกล่องแจ้งเตือน (alert)

☐ **ตัวอย่างที่ 2: เปลี่ยนรูปภาพ**

ไฟล์ index.html

```
<!DOCTYPE html>
<html lang="th">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>เปลี่ยนรูปภาพ</title>
    <script src="script.js" defer></script>
</head>
<body>
    <img id="myImage" src="image1.jpg" width="300">
    <button onclick="changeImage()">เปลี่ยนรูปภาพ</button>
</body>
</html>
```

ไฟล์ script.js

```
function changeImage() {
    document.getElementById("myImage").src = "image2.jpg";
}
```

☐ **อธิบาย:** เมื่อกดปุ่ม รูปภาพจะเปลี่ยนจาก image1.jpg เป็น image2.jpg

---

☐ **ตัวอย่างที่ 3: แสดงเวลาปัจจุบัน**

ไฟล์ index.html

```
<!DOCTYPE html>
<html lang="th">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>แสดงเวลาปัจจุบัน</title>
    <script src="script.js" defer></script>
</head>
<body>
    <p>เวลาปัจจุบัน: <span id="time"></span></p>
</body>
```

```
</html>
```

ไฟล์ script.js

```
function showTime() {
    let now = new Date();
    document.getElementById("time").innerText = now.toLocaleTimeString();
}
setInterval(showTime, 1000);
```

อธิบาย: เวลาใน จะอัปเดตทุกๆ 1 วินาที

---

3. Inline JavaScript (ใช้ในแอตทริบิวต์ HTML โดยตรง)

ตัวอย่างที่ 1: เปลี่ยนสีข้อความ

```
<!DOCTYPE html>
<html lang="th">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>เปลี่ยนสีข้อความ</title>
</head>
<body>
    <p id="text" style="color: black;">ข้อความนี้จะเปลี่ยนสี</p>
    <button onclick="document.getElementById('text').style.color='red'">เปลี่ยนเป็นสีแดง</button>
</body>
</html>
```

อธิบาย: เมื่อกดปุ่ม ข้อความจะเปลี่ยนเป็นสีแดงทันที

ตัวอย่างที่ 2: ซ่อนและแสดงข้อความ

```
<!DOCTYPE html>
<html lang="th">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ซ่อน/แสดงข้อความ</title>
</head>
<body>
```

```
    <p id="text">นี่คือข้อความ</p>
    <button onclick="document.getElementById('text').style.display='none'">ซ่อน</button>
    <button onclick="document.getElementById('text').style.display='block'">แสดง</button>
</body>
</html>
```

☐ **อธิบาย:** กดปุ่ม "ซ่อน" ข้อความจะหายไป กด "แสดง" ข้อความจะกลับมา

☐ **ตัวอย่างที่ 3: เปลี่ยนพื้นหลังด้วย Dropdown**

```
<!DOCTYPE html>
<html lang="th">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>เปลี่ยนพื้นหลัง</title>
</head>
<body>
    <select onchange="document.body.style.backgroundColor=this.value">
        <option value="white">ขาว</option>
        <option value="lightblue">ฟ้า</option>
        <option value="lightgreen">เขียว</option>
    </select>
</body>
</html>
```

☐ **อธิบาย:** เลือกสีจาก Dropdown แล้วพื้นหลังจะเปลี่ยนสีทันที

☐ **สรุป**

☐ **Internal** ⟶ ใช้งานง่าย เหมาะกับโค้ดสั้นๆ

☐ **External** ⟶ โค้ดสะอาด ใช้ซ้ำได้ เหมาะกับโค้ดใหญ่

☐ **Inline** ⟶ ใช้งานเร็ว แต่โค้ดอาจรก

การใช้งาน Console และ Developer Tools คืออะไร?

Console และ **Developer Tools** เป็นเครื่องมือที่มีอยู่ในเว็บเบราว์เซอร์ เช่น **Google Chrome, Firefox, Edge, Safari** ซึ่งช่วยในการตรวจสอบและดีบัก (Debug) โค้ด JavaScript รวมถึงแก้ไข HTML และ CSS แบบเรียลไทม์

☐ 1. Developer Tools คืออะไร?

Developer Tools (DevTools) เป็นชุดเครื่องมือสำหรับนักพัฒนา ช่วยให้สามารถดูโครงสร้างของเว็บเพจ ตรวจสอบโค้ด HTML, CSS และ JavaScript รวมถึงแก้ไขโค้ดและทดสอบฟังก์ชัน JavaScript ได้

☐ วิธีเปิด Developer Tools ในเบราว์เซอร์

1. **กดปุ่มลัด**
   - **Windows/Linux** ⟶ F12 หรือ Ctrl + Shift + I
   - **Mac** ⟶ Cmd + Option + I
2. **ใช้เมนูเบราว์เซอร์**
   - คลิกขวาที่หน้าเว็บ ⟶ เลือก **ตรวจสอบ (Inspect)**

☐ ส่วนสำคัญใน Developer Tools

1. **Elements** ⟶ ดูและแก้ไข HTML และ CSS
2. **Console** ⟶ รันคำสั่ง JavaScript และดูข้อผิดพลาด
3. **Sources** ⟶ ดูไฟล์ JavaScript และดีบักโค้ด
4. **Network** ⟶ ตรวจสอบการโหลดไฟล์ เช่น API, รูปภาพ
5. **Performance** ⟶ วิเคราะห์ความเร็วของเว็บไซต์

☐ 2. Console คืออะไร?

Console เป็นพื้นที่สำหรับรัน JavaScript โดยตรง สามารถใช้ในการทดสอบโค้ด แสดงค่าตัวแปร หรือดูข้อผิดพลาดของ JavaScript

☐ วิธีเปิด Console

1. เปิด **Developer Tools** (F12 หรือ Ctrl + Shift + I)
2. ไปที่แท็บ **Console**

☐ 3. ตัวอย่างการใช้งาน Console

☐ ตัวอย่างที่ 1: แสดงข้อความใน Console

```
console.log("Hello, JavaScript!");
```

☐ **คำอธิบาย**: ใช้ console.log() เพื่อพิมพ์ข้อความไปที่ Console

☐ วิธีดูผลลัพธ์

1. เปิด Developer Tools

2. ไปที่แท็บ Console
3. ดูผลลัพธ์ที่แสดง

☐ **ตัวอย่างที่ 2: แสดงค่าตัวแปร**

```
let name = "John";
console.log("ชื่อของฉันคือ:", name);
```

☐ **คำอธิบาย**: แสดงค่าตัวแปร name ใน Console

☐ **ตัวอย่างที่ 3: แสดงข้อผิดพลาด**

```
console.error("นี่คือข้อความแจ้งข้อผิดพลาด!");
```

☐ **คำอธิบาย**: ใช้ console.error() เพื่อแสดงข้อผิดพลาดใน Console

---

☐ **ตัวอย่างที่ 4: แสดงคำเตือน**

```
console.warn("นี่คือข้อความแจ้งเตือน!");
```

☐ **คำอธิบาย**: ใช้ console.warn() เพื่อแสดงคำเตือน

☐ **ตัวอย่างที่ 5: ทดสอบเงื่อนไขด้วย Console**

```
let age = 18;
console.assert(age >= 20, "อายุไม่ถึงเกณฑ์!");
```

☐ **คำอธิบาย**: ใช้ console.assert() เพื่อตรวจสอบเงื่อนไข ถ้า false จะแสดงข้อความแจ้งเตือน

☐ **ตัวอย่างที่ 6: วัดเวลาในการรันโค้ด**

```
console.time("myTimer");
for (let i = 0; i < 100000; i++) {} // วนลูป
console.timeEnd("myTimer");
```

☐ **คำอธิบาย**: ใช้ console.time() และ console.timeEnd() เพื่อนับเวลาที่ใช้ในการรันโค้ด

☐ **4. วิธีใช้ Developer Tools สำหรับ Debugging JavaScript**

☐ **วิธีใช้งาน**

1. เปิด Developer Tools (F12 หรือ Ctrl + Shift + I)
2. ไปที่แท็บ **Sources**
3. เลือกไฟล์ JavaScript ที่ต้องการ
4. คลิกที่หมายเลขบรรทัดเพื่อวาง Breakpoint
5. รีเฟรชหน้าเว็บและดูการทำงานของโค้ดทีละขั้นตอน

☐ **ตัวอย่าง Debugging**

```
function addNumbers(a, b) {
    let sum = a + b;
    console.log("ผลรวม:", sum);
    return sum;
}

addNumbers(5, 10);
```

☐ **คำอธิบาย**:

- ใส่ **Breakpoint** ที่บรรทัด let sum = a + b;
- รันโค้ดและดูค่าตัวแปร sum

☐ **5. สรุป**

เครื่องมือ	ใช้ทำอะไร?	วิธีเปิด
Console	แสดงผลลัพธ์, ดีบัก, ดูข้อผิดพลาด	F12 ⟶ Console
Elements	แก้ไข HTML, CSS แบบเรียลไทม์	F12 ⟶ Elements
Sources	Debug JavaScript	F12 ⟶ Sources
Network	ตรวจสอบ API, โหลดไฟล์	F12 ⟶ Network

☐ **แนะนำ:** ใช้ console.log() และ Debugger เพื่อหาข้อผิดพลาดในโค้ด JavaScript

การใช้งาน Console และ Developer Tools อย่างละเอียด

☐ **1. Developer Tools คืออะไร?**

Developer Tools (DevTools) เป็นเครื่องมือสำหรับนักพัฒนาที่ช่วยตรวจสอบและแก้ไข **HTML, CSS, JavaScript** บนเว็บเบราว์เซอร์แบบเรียลไทม์ โดยไม่ต้องแก้ไขไฟล์โค้ดจริง

☐ **วิธีเปิด Developer Tools**

1. **ใช้ปุ่มลัด**
   - **Windows/Linux** ⟶ F12 หรือ Ctrl + Shift + I
   - **Mac** ⟶ Cmd + Option + I
2. **ใช้เมนูเบราว์เซอร์**
   - คลิกขวาที่หน้าเว็บ ⟶ เลือก **ตรวจสอบ (Inspect)**
3. **เปิด Console โดยตรง**
   - กด Ctrl + Shift + J (Windows/Linux) หรือ Cmd + Option + J (Mac)

☐ 2. ส่วนสำคัญของ Developer Tools

ส่วน	ใช้ทำอะไร?
Elements	ดูและแก้ไขโค้ด HTML/CSS
Console	รันคำสั่ง JavaScript, แสดงผลลัพธ์, Debug
Sources	ดูไฟล์ JavaScript และดีบักโค้ด
Network	ดูการโหลดไฟล์ เช่น API, รูปภาพ
Performance	วิเคราะห์ความเร็วของเว็บไซต์
Application	จัดการ Local Storage, Cookies

☐ 3. Console คืออะไร?

Console เป็นพื้นที่สำหรับรัน JavaScript โดยตรง สามารถใช้เพื่อตรวจสอบค่าตัวแปร, แสดงผลลัพธ์, และดีบักโค้ดได้

☐ 4. ตัวอย่างการใช้งาน Console อย่างละเอียด

☐ 1. การใช้ console.log() แสดงค่าตัวแปร

```
let message = "สวัสดี!";
console.log(message);
```

☐ **ผลลัพธ์ใน Console:**

สวัสดี!

☐ 2. การแสดงค่าตัวแปรหลายตัว

```
let name = "John";
let age = 25;
console.log("ชื่อ:", name, "อายุ:", age);
```

☐ **ผลลัพธ์:**

ชื่อ: John อายุ: 25

☐ 3. การใช้ console.error() แสดงข้อผิดพลาด

```
console.error("เกิดข้อผิดพลาด!");
```

☐ **ผลลัพธ์:**

ข้อความแจ้งเตือนเป็นสีแดงใน Console

☐ **4. การใช้ console.warn() แสดงคำเตือน**

```
console.warn("นี่คือคำเตือน!");
```

☐ **ผลลัพธ์:**

ข้อความแจ้งเตือนเป็นสีเหลืองใน Console

☐ **5. การตรวจสอบเงื่อนไขด้วย console.assert()**

```
let age = 18;
console.assert(age >= 20, "อายุไม่ถึง 20 ปี!");
```

☐ **ผลลัพธ์:**

หาก age < 20 จะแสดงข้อความแจ้งเตือน

☐ **6. วัดเวลาในการรันโค้ดด้วย console.time()**

```
console.time("test");
for (let i = 0; i < 1000000; i++) {} // วนลูป
console.timeEnd("test");
```

☐ **ผลลัพธ์:**

test: 5.23ms

แสดงเวลาที่ใช้ในการรันโค้ด

☐ **5. การ Debug JavaScript ด้วย Developer Tools**

☐ **วิธี Debug โค้ด**

1. เปิด **Developer Tools** (F12 หรือ Ctrl + Shift + I)
2. ไปที่แท็บ **Sources**
3. เลือกไฟล์ JavaScript ที่ต้องการ
4. คลิกที่หมายเลขบรรทัด (Breakpoint)
5. รีเฟรชหน้าเว็บเพื่อดูผลลัพธ์

☐ **ตัวอย่างโค้ดสำหรับ Debug**

```
function addNumbers(a, b) {
    let sum = a + b;  // ใส่ Breakpoint ที่นี่
    console.log("ผลรวม:", sum);
    return sum;
}
```

addNumbers(5, 10);

☐ **ผลลัพธ์:**

เมื่อรันโค้ด ระบบจะหยุดที่ sum = a + b; และเราสามารถดูค่าตัวแปร a, b, sum ใน Debugger ได้

☐ **6. การตรวจสอบ HTML และ CSS ด้วย Developer Tools**

1. เปิด Developer Tools (F12)
2. ไปที่แท็บ **Elements**
3. เลือก Element ที่ต้องการแก้ไข
4. เปลี่ยนค่าภายใน HTML หรือ CSS

☐ **ตัวอย่าง: แก้ไขสีพื้นหลัง**

1. ไปที่แท็บ **Elements**
2. คลิก <body>
3. ไปที่ Styles แล้วเพิ่ม
4. background-color: lightblue;

☐ **ผลลัพธ์:**

พื้นหลังของเว็บเพจจะเปลี่ยนเป็นสีฟ้าทันที

☐ **7. การตรวจสอบ Network Requests**

1. ไปที่แท็บ **Network**
2. รีเฟรชหน้าเว็บ (F5)
3. ดูไฟล์ที่ถูกโหลด เช่น CSS, JavaScript, API Requests
4. คลิกที่ API เพื่อดู Response

☐ **ตัวอย่าง:**

ถ้ามีการเรียก API ที่ https://api.example.com/data

เราสามารถดู **Headers, Response, Timing** ได้

☐ **8. สรุป**

☐ **Console** ⟶ ใช้ทดสอบโค้ด JavaScript

☐ **Elements** ⟶ ใช้แก้ไข HTML และ CSS

☐ **Sources** ⟶ ใช้ Debug JavaScript

☐ **Network** ⟶ ใช้ตรวจสอบ API Requests

ตัวอย่างการใช้งาน Console และ Developer Tools พร้อมโค้ดแบบเต็ม

□ 1. ตัวอย่างโค้ด HTML + JavaScript สำหรับใช้กับ Developer Tools

ไฟล์นี้เป็นไฟล์ **index.html** ที่มีโค้ด JavaScript ฝังอยู่ เพื่อทดสอบการใช้งาน **Console, Debugging** และ **Network Requests** ใน Developer Tools

□ ไฟล์: index.html

```
<!DOCTYPE html>
<html lang="th">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ทดสอบ Developer Tools</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin-top: 50px;
        }
        #btn {
            padding: 10px 20px;
            font-size: 18px;
            background-color: blue;
            color: white;
            border: none;
            cursor: pointer;
        }
    </style>
</head>
<body>

    <h1>กดปุ่มเพื่อทดสอบ JavaScript</h1>
    <button id="btn">คลิกที่นี่</button>

    <script>
        // แสดงข้อความใน Console
```

```
    console.log("เริ่มต้นการทำงานของ JavaScript...");

    // ฟังก์ชันเมื่อกดปุ่ม
    document.getElementById("btn").addEventListener("click", function() {
        console.log("ปุ่มถูกกด!");

        let num1 = 10;
        let num2 = 20;
        let result = addNumbers(num1, num2);

        console.log("ผลรวมคือ:", result);
    });

    // ฟังก์ชันคำนวณผลรวม
    function addNumbers(a, b) {
        console.time("Execution Time");
        let sum = a + b;
        console.timeEnd("Execution Time");

        console.assert(sum > 0, "ผลรวมควรมากกว่า 0!");
        return sum;
    }

    // แสดงคำเตือนและข้อผิดพลาด
    console.warn("นี่คือคำเตือน!");
    console.error("นี่คือตัวอย่างข้อผิดพลาด!");

    // ทดสอบ Network Request
    fetch("https://jsonplaceholder.typicode.com/posts/1")
        .then(response => response.json())
        .then(data => console.log("API Response:", data))
        .catch(error => console.error("เกิดข้อผิดพลาดในการดึงข้อมูล:", error));
</script>
```

```
</body>
</html>
```

---

2. คำอธิบายโค้ด

1. **แสดงข้อความใน Console**
2. console.log("เริ่มต้นการทำงานของ JavaScript...");
   - แสดงข้อความใน Developer Tools ⟶ **Console**
3. **จับเหตุการณ์เมื่อคลิกปุ่ม**
4. document.getElementById("btn").addEventListener("click", function() {
5. console.log("ปุ่มถูกกด!");
6. });
   - เมื่อคลิกปุ่ม "คลิกที่นี่" ระบบจะพิมพ์ "ปุ่มถูกกด!" ลงใน Console
7. **ฟังก์ชันคำนวณผลรวม + Debugging**
8. let num1 = 10;
9. let num2 = 20;
10. let result = addNumbers(num1, num2);
11. console.log("ผลรวมคือ:", result);
    - คำนวณค่า num1 + num2 และแสดงผลใน Console
12. **ใช้ console.time() วัดเวลาการทำงาน**
13. console.time("Execution Time");
14. let sum = a + b;
15. console.timeEnd("Execution Time");
    - แสดงเวลาที่ใช้ในการรันโค้ด
16. **ใช้ console.assert() เพื่อตรวจสอบเงื่อนไข**
17. console.assert(sum > 0, "ผลรวมควรมากกว่า 0!");
    - ถ้าผลรวม < 0 จะแสดงข้อความแจ้งเตือน
18. **แสดงคำเตือนและข้อผิดพลาด**
19. console.warn("นี่คือคำเตือน!");
20. console.error("นี่คือตัวอย่างข้อผิดพลาด!");
    - console.warn() จะแสดงข้อความสีเหลือง, console.error() จะแสดงข้อความสีแดง
21. **ทำ Network Request เพื่อดึงข้อมูลจาก API**
22. fetch("https://jsonplaceholder.typicode.com/posts/1")
23. .then(response => response.json())
24. .then(data => console.log("API Response:", data))

25. .catch(error => console.error("เกิดข้อผิดพลาดในการดึงข้อมูล:", error));

□ โหลดข้อมูลจาก API และแสดงผลใน Console

□ **3. วิธีทดสอบโค้ดใน Developer Tools**

1. **เปิดไฟล์ index.html บนเว็บเบราว์เซอร์**
2. กดปุ่ม **F12 หรือ Ctrl + Shift + I**
3. ไปที่แท็บ **Console**
4. ลองคลิกปุ่ม "คลิกที่นี่" แล้วสังเกตข้อความที่แสดง

□ **คุณจะเห็นอะไรใน Console?**

- "เริ่มต้นการทำงานของ JavaScript..."
- "ปุ่มถูกกด!" (เมื่อกดปุ่ม)
- "ผลรวมคือ: 30"
- "Execution Time: X ms"
- "นี่คือคำเตือน!"
- "นี่คือตัวอย่างข้อผิดพลาด!"
- API Response: { userId: 1, id: 1, title: "...", body: "..." }

□ **4. การ Debugging ด้วย Breakpoint**

1. เปิด **Developer Tools** (F12 หรือ Ctrl + Shift + I)
2. ไปที่แท็บ **Sources**
3. เลือกไฟล์ index.html ⟶ ไปที่โค้ด JavaScript
4. คลิกที่หมายเลขบรรทัดของฟังก์ชัน addNumbers(a, b) เพื่อวาง **Breakpoint**
5. กด **Refresh (F5)**
6. คลิกปุ่ม "คลิกที่นี่"
7. ดูค่าตัวแปรใน Debugger

□ **5. สรุป**

□ **Console** ⟶ ใช้แสดงผลลัพธ์, Debug, วัดเวลา

□ **Elements** ⟶ ใช้แก้ไข HTML/CSS แบบเรียลไทม์

□ **Sources** ⟶ ใช้ Debug JavaScript โดยตั้ง Breakpoint

□ **Network** ⟶ ใช้ตรวจสอบ API Requests

□ **ลองรันโค้ดและเปิด Console เพื่อดูผลลัพธ์!** □

ตัวอย่างเพิ่มเติมอีก 3 ตัวอย่าง สำหรับการใช้งาน Console และ Developer Tools

---

□ ตัวอย่างที่ 1: การใช้ Console Debugging แบบละเอียด

ไฟล์นี้จะแสดงการใช้ console.log(), console.error(), และ console.table()

□ ไฟล์: debugging.html

```
<!DOCTYPE html>
<html lang="th">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ทดสอบ Debugging</title>
</head>
<body>

    <h1>กดปุ่มเพื่อแสดงข้อมูลใน Console</h1>
    <button onclick="showDebugInfo()">แสดงข้อมูล</button>

    <script>
        function showDebugInfo() {
            console.log("เริ่มต้น Debugging...");

            let user = {
                id: 101,
                name: "สมชาย",
                age: 30
            };

            console.log("ข้อมูลผู้ใช้:", user);
            console.table(user); // แสดงข้อมูลในรูปแบบตาราง

            // ทดสอบการเกิดข้อผิดพลาด
            try {
                let result = 10 / 0; // หารด้วยศูนย์
                console.log("ผลลัพธ์:", result);
```

```
        } catch (error) {
            console.error("เกิดข้อผิดพลาด:", error);
        }

        console.warn("นี่คือคำเตือน!");
    }
</script>

</body>
</html>
```

สิ่งที่จะแสดงใน Console

- "เริ่มต้น Debugging..."
- ข้อมูล user ในรูปแบบตาราง
- ข้อผิดพลาด "เกิดข้อผิดพลาด: ..."
- "นี่คือคำเตือน!"

ตัวอย่างที่ 2: การใช้ Console ในการตรวจสอบ Event Listener

ไฟล์นี้ใช้ console.dir() เพื่อตรวจสอบโครงสร้างของ HTML Element ที่ถูกกด

ไฟล์: event-console.html

```
<!DOCTYPE html>
<html lang="th">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ทดสอบ Console Event</title>
</head>
<body>

    <h1>คลิกปุ่มแล้วดูผลลัพธ์ใน Console</h1>
    <button id="testButton">กดฉัน</button>

    <script>
        let button = document.getElementById("testButton");
```

```
        button.addEventListener("click", function(event) {
            console.log("ปุ่มถูกกด!");
            console.dir(event.target); // แสดงโครงสร้าง Element
        });
    </script>

</body>
</html>
```

สิ่งที่จะแสดงใน Console

- "ปุ่มถูกกด!"
- ข้อมูลโครงสร้างของ <button> ที่ถูกกด

ตัวอย่างที่ 3: การใช้ Network Tab ตรวจสอบ API Request

ไฟล์นี้ใช้ fetch() เพื่อเรียก API และตรวจสอบการรับ-ส่งข้อมูล

ไฟล์: fetch-api.html

```
<!DOCTYPE html>
<html lang="th">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ทดสอบ API Request</title>
</head>
<body>

    <h1>ดึงข้อมูล API และแสดงใน Console</h1>
    <button onclick="fetchData()">โหลดข้อมูล</button>

    <script>
        function fetchData() {
            console.log("กำลังโหลดข้อมูล...");

            fetch("https://jsonplaceholder.typicode.com/users/1")
                .then(response => {
                    console.log("สถานะการตอบกลับ:", response.status);
```

```
            return response.json();
        })
        .then(data => {
            console.log("ข้อมูลที่ได้รับ:", data);
        })
        .catch(error => {
            console.error("เกิดข้อผิดพลาด:", error);
        });
    }
</script>

</body>
</html>
```

☐ **วิธีตรวจสอบข้อมูลใน Developer Tools**

1. เปิดหน้าเว็บ
2. กดปุ่ม "โหลดข้อมูล"
3. เปิด Developer Tools (F12) ⟶ ไปที่ **Console** และ **Network**
4. ดูข้อมูล API Response

☐ **สรุป**

☐ **ตัวอย่างที่ 1**: ใช้ Console Debugging (console.table(), console.error())

☐ **ตัวอย่างที่ 2**: ใช้ Console ตรวจสอบ Event (console.dir())

☐ **ตัวอย่างที่ 3**: ใช้ Console และ Network Tab ตรวจสอบ API Requests (fetch())

สรุป

JavaScript เป็นภาษาโปรแกรมที่ทำงานบนเว็บเบราว์เซอร์ ซึ่งช่วยให้หน้าเว็บมีความสามารถในการโต้ตอบและเปลี่ยนแปลงข้อมูลได้แบบไดนามิก โดยสามารถทำงานร่วมกับ HTML ซึ่งใช้สร้างโครงสร้างของหน้าเว็บ และ CSS ที่ใช้ควบคุมการแสดงผล ส่วน JavaScript จะรับหน้าที่ควบคุมพฤติกรรมขององค์ประกอบต่างๆ เช่น การตรวจสอบฟอร์ม การเปลี่ยนข้อความบนปุ่ม หรือการโหลดข้อมูลแบบไม่ต้องรีเฟรชหน้าเว็บ (AJAX) การแทรก JavaScript ลงใน HTML สามารถทำได้ 3 วิธี ได้แก่ **Inline** (เขียนใน Event Attribute เช่น onclick), **Internal** (เขียนภายใน <script> ในไฟล์ HTML), และ **External** (แยกไว้ในไฟล์ .js แล้วเชื่อมผ่าน <script src="..."></script>) ซึ่งวิธี External ช่วยให้โค้ดดูเป็นระเบียบและนำกลับมาใช้ซ้ำได้ง่าย

เครื่องมือสำคัญที่นักพัฒนามักใช้คือ **Console** ซึ่งอยู่ใน **Developer Tools** ของเว็บเบราว์เซอร์ เช่น Chrome หรือ Firefox โดยสามารถเปิดได้ด้วยการกด F12 หรือ Ctrl + Shift + I แล้วเลือกแท็บ Console ที่นี่สามารถพิมพ์คำสั่ง JavaScript เพื่อทดสอบ ตรวจสอบค่าตัวแปร แสดงข้อความ หรือดูโครงสร้างของ Object ได้ทันที นอกจากนี้ยังสามารถตั้ง **Breakpoint** เพื่อตรวจสอบขั้นตอนการทำงานของโค้ดแบบละเอียด ทำให้สามารถวิเคราะห์และแก้ไขข้อผิดพลาดได้อย่างมีประสิทธิภาพ การฝึกใช้งาน Console และ Debugger ช่วยให้พัฒนาเว็บแอปได้เร็วขึ้นและลดปัญหาบั๊กในระยะยาว.

บทที่ 2

ไวยากรณ์และคำสั่งของจาวาสคริปต์

(Syntax and Commands of JavaScript)

เนื้อหา

- Syntax และโครงสร้างพื้นฐานของ JavaScript
- องค์ประกอบหลักของ JavaScript Syntax
- ขั้นตอนการเรียนรู้ JavaScript อย่างมีประสิทธิภาพ
- ตัวแปร (var, let, const) ใน JavaScript
- ชนิดของข้อมูลใน JavaScript (Primitive & Reference Types)
- การดำเนินการทางคณิตศาสตร์และตรรกะ (Operators) ใน JavaScript
- คำสั่งเงื่อนไข (if-else, switch-case) ใน JavaScript
- ลูป (for, while, do-while) ใน JavaScript
- ฟังก์ชัน (Function) ใน JavaScript

ใน JavaScript การประกาศตัวแปรสามารถทำได้ด้วย var, let และ const โดย var เป็นรูปแบบดั้งเดิมที่มีขอบเขตแบบฟังก์ชัน ขณะที่ let และ const ถูกแนะนำใน ES6 ซึ่งมีขอบเขตแบบบล็อก (block scope) และเหมาะกับการใช้งานที่ปลอดภัยกว่า โดย let ใช้เมื่อค่าของตัวแปรอาจเปลี่ยนแปลงได้ ส่วน const ใช้เมื่อค่าคงที่ ไม่เปลี่ยนแปลง นอกจากนี้ JavaScript ยังรองรับชนิดข้อมูลหลากหลาย แบ่งเป็นสองประเภทหลัก ได้แก่ **Primitive types** (เช่น number, string, boolean, null, undefined, symbol, และ bigint) และ **Reference types** เช่น object, array, และ function ซึ่งอ้างอิงข้อมูลในหน่วยความจำ

สำหรับการคำนวณและการตัดสินใจ JavaScript มี **Operators** เช่น ตัวดำเนินการทางคณิตศาสตร์ (+, -, *, /, %), ตัวดำเนินการเปรียบเทียบ (==, ===, !=, !==, <, >) และตรรกะ (&&, ||, !) คำสั่งเงื่อนไข เช่น if-else และ switch-case ช่วยให้เขียนโปรแกรมที่ตัดสินใจได้ตามเงื่อนไขที่กำหนด ส่วนคำสั่งวนลูป ได้แก่ for, while และ do-while ช่วยในการทำงานซ้ำตามจำนวนครั้งที่ต้องการหรือตามเงื่อนไข เช่น แสดงรายการสินค้าในตะกร้า หรือวนหาผลรวมของข้อมูลใน array

ฟังก์ชัน (Function) เป็นส่วนสำคัญในการเขียนโปรแกรม JavaScript โดยสามารถประกาศได้ด้วยคำสั่ง function หรือใช้รูปแบบสั้นอย่าง **Arrow Function** (เช่น const sum = (a, b) => a + b) ซึ่งทำให้โค้ดกระชับมากขึ้น ฟังก์ชันสามารถรับค่าพารามิเตอร์และส่งค่ากลับได้ นอกจากนี้ JavaScript ยังรองรับ **Callback Function** ซึ่งคือฟังก์ชันที่ถูกส่งเป็นอาร์กิวเมนต์เข้าไปในฟังก์ชันอื่น เพื่อให้ถูกเรียกใช้งานใน

ภายหลัง เช่น เมื่อโหลดข้อมูลเสร็จ หรือเมื่อผู้ใช้คลิกปุ่ม เป็นแนวทางการเขียนโปรแกรมแบบ **Asynchronous** ที่พบได้บ่อยใน JavaScript โดยเฉพาะเมื่อทำงานกับ Event และ API.

Syntax และโครงสร้างพื้นฐานของ JavaScript

□ ความหมายของ Syntax ใน JavaScript

Syntax (ไวยากรณ์) ของ JavaScript หมายถึง กฎและโครงสร้างของภาษาที่ใช้ในการเขียนโปรแกรม JavaScript ให้ถูกต้อง เพื่อให้คอมพิวเตอร์สามารถแปลและทำงานตามที่ต้องการได้

Syntax ของ JavaScript มีความสำคัญเพราะ:

- เป็นมาตรฐานที่นักพัฒนาใช้ร่วมกัน
- ช่วยให้โค้ดสามารถทำงานได้ถูกต้องโดยไม่มีข้อผิดพลาด
- ทำให้การเรียนรู้และพัฒนาโค้ดมีโครงสร้างที่เข้าใจง่าย

□ โครงสร้างพื้นฐานของ JavaScript

JavaScript ประกอบด้วยองค์ประกอบหลักๆ ที่ต้องเข้าใจ ดังนี้

1. คำสั่งและตัวดำเนินการ (Statements & Operators)

JavaScript ประกอบด้วยชุดคำสั่งที่ทำงานตามลำดับ โดยแต่ละคำสั่งมักจะจบด้วย ; (เครื่องหมายเซมิโคลอน)

- ตัวอย่างของคำสั่งทั่วไป เช่น การกำหนดตัวแปร, การคำนวณทางคณิตศาสตร์, เงื่อนไข, ลูป

ตัวดำเนินการหลักใน JavaScript

- **ตัวดำเนินการทางคณิตศาสตร์:** +, -, *, /, %
- **ตัวดำเนินการเปรียบเทียบ:** ==, ===, !=, !==, <, >, <=, >=
- **ตัวดำเนินการตรรกะ:** &&, ||, !

2. ตัวแปร (Variables) และประเภทข้อมูล (Data Types)

JavaScript ใช้ตัวแปรเพื่อเก็บค่าหรือข้อมูล โดยสามารถประกาศตัวแปรได้ 3 แบบ

- var (ไม่แนะนำให้ใช้)
- let (ใช้สำหรับค่าที่สามารถเปลี่ยนแปลงได้)
- const (ใช้สำหรับค่าคงที่ที่ไม่ต้องการให้เปลี่ยนแปลง)

ประเภทข้อมูลใน JavaScript

- **Primitive Types** (ค่าพื้นฐาน): String, Number, Boolean, Null, Undefined, Symbol, BigInt
- **Object Types** (อ้างอิงถึงข้อมูลอื่นๆ): Object, Array, Function

3. ฟังก์ชัน (Functions) และขอบเขตของตัวแปร (Scope)

ฟังก์ชันใช้สำหรับจัดกลุ่มคำสั่งเพื่อให้สามารถเรียกใช้ซ้ำได้

- มีทั้ง **ฟังก์ชันปกติ** และ **Arrow Function (=>)**

- ขอบเขตของตัวแปรแบ่งเป็น **Global Scope**, **Local Scope**, และ **Block Scope**

4. คำสั่งควบคุมการไหลของโปรแกรม (Control Flow)

- **เงื่อนไข (if, else if, else, switch)** ใช้เพื่อตัดสินใจว่าคำสั่งใดควรถูกดำเนินการ
- **ลูป (for, while, do-while)** ใช้สำหรับทำซ้ำคำสั่งหลายๆ ครั้ง

5. การทำงานกับ DOM (Document Object Model)

JavaScript สามารถควบคุมและเปลี่ยนแปลงเนื้อหา HTML ได้โดยใช้ document.getElementById(), document.querySelector(), document.createElement(), ฯลฯ

☐ **ความจำเป็นของ JavaScript Syntax และโครงสร้างพื้นฐาน**

- ทำให้เว็บไซต์มีความโต้ตอบ (Interactive)
- ช่วยให้สามารถสร้างเว็บแอปพลิเคชันที่ซับซ้อนได้
- เป็นพื้นฐานสำหรับการเรียนรู้ JavaScript ขั้นสูง เช่น ES6, React.js, Node.js
- ใช้เป็นมาตรฐานในการพัฒนาเว็บไซต์ที่รองรับทุกเบราว์เซอร์

☐ **ขั้นตอนการเรียนรู้และเขียนโปรแกรม JavaScript**

1. **เริ่มต้นด้วยพื้นฐาน**
   - ทำความเข้าใจกับ Syntax และโครงสร้างของ JavaScript
   - ฝึกเขียนตัวแปรและฟังก์ชันง่ายๆ
2. **ทดลองใช้ JavaScript กับ HTML & CSS**
   - ใช้ <script> ใน HTML และเชื่อมโยงไฟล์ .js
3. **ศึกษาการทำงานกับ DOM และ Event Handling**
   - ลองใช้ document.getElementById() เพื่อเปลี่ยนแปลง HTML
   - ฝึกใช้ Event เช่น onclick, onchange
4. **ฝึก Debugging ด้วย Console และ Developer Tools**
   - ใช้ console.log() เพื่อตรวจสอบค่าตัวแปร
   - ใช้ Breakpoint ใน Developer Tools
5. **พัฒนาโครงการจริง**
   - ลองสร้างเว็บแอปเล็กๆ เช่น To-Do List
   - ศึกษาการใช้งาน JavaScript Frameworks (เช่น React.js, Vue.js)

☐ **สรุป**

- **Syntax ของ JavaScript** เป็นกฎพื้นฐานที่ต้องปฏิบัติตามเพื่อให้โค้ดทำงานได้ถูกต้อง
- **โครงสร้างพื้นฐานของ JavaScript** ประกอบด้วย Statements, Variables, Data Types, Functions, Control Flow และ DOM

- **Syntax และโครงสร้างพื้นฐานของ JavaScript** มีความจำเป็นต่อการพัฒนาเว็บไซต์แบบโต้ตอบ
- **ขั้นตอนการเรียนรู้ JavaScript** ควรเริ่มจากพื้นฐาน ฝึกเขียนโค้ด และพัฒนาโปรเจ็กต์จริง

 ถ้าสามารถเข้าใจ Syntax และโครงสร้างพื้นฐานของ JavaScript ได้ดี ก็จะสามารถพัฒนาเว็บแอปที่มีประสิทธิภาพและโต้ตอบได้ง่ายขึ้น!

 ข้อมูลเพิ่มเติมเกี่ยวกับ Syntax และโครงสร้างพื้นฐานของ JavaScript

 Syntax (ไวยากรณ์) ของ JavaScript

ไวยากรณ์ของ JavaScript เป็นชุดของกฎที่กำหนดวิธีการเขียนโค้ดให้ถูกต้อง โดย JavaScript เป็นภาษาที่มีความยืดหยุ่น และสามารถใช้ได้ทั้งแบบ **Imperative (เชิงคำสั่ง)** และ **Declarative (เชิงประกาศ)**

องค์ประกอบหลักของ JavaScript Syntax

1. การประกาศตัวแปร (Variable Declaration)

ตัวแปรใน JavaScript ใช้สำหรับเก็บข้อมูล โดยมี 3 วิธีการประกาศตัวแปรหลัก ได้แก่

- var ⟶ ใช้ได้ทุกที่ แต่มีปัญหาเรื่อง Scope และ Hoisting
- let ⟶ ใช้สำหรับค่าที่สามารถเปลี่ยนแปลงได้ (แนะนำ)
- const ⟶ ใช้สำหรับค่าคงที่ที่ไม่ต้องเปลี่ยนแปลง

ตัวอย่าง:

```
let name = "Alice";
const age = 25;
var city = "Bangkok";
```

---

2. ชนิดข้อมูล (Data Types) ใน JavaScript

ข้อมูลใน JavaScript แบ่งออกเป็น 2 ประเภทหลัก

 Primitive Data Types (ข้อมูลพื้นฐาน)

- **String** (ข้อความ) ⟶ "Hello"
- **Number** (ตัวเลข) ⟶ 100, 3.14
- **Boolean** (ค่าความจริง) ⟶ true, false
- **Null** (ค่าว่าง) ⟶ null
- **Undefined** (ไม่ได้กำหนดค่า) ⟶ undefined
- **Symbol** (ค่าที่ไม่ซ้ำกัน) ⟶ Symbol("id")
- **BigInt** (จำนวนเต็มขนาดใหญ่) ⟶ 12345678901234567890n

☐ **Reference Data Types (ข้อมูลอ้างอิง)**

- **Object** (ออบเจ็กต์) ⟶ { name: "Alice", age: 25 }
- **Array** (อาร์เรย์) ⟶ [1, 2, 3, 4, 5]
- **Function** (ฟังก์ชัน) ⟶ function greet() {}

---

3. ตัวดำเนินการ (Operators)

JavaScript มีตัวดำเนินการหลายประเภท

☐ **ตัวดำเนินการทางคณิตศาสตร์ (Arithmetic Operators)**

- + บวก
- - ลบ
- * คูณ
- / หาร
- % หารเอาเศษ
- ++ เพิ่มค่า 1
- -- ลดค่า 1

☐ **ตัวดำเนินการเปรียบเทียบ (Comparison Operators)**

- == เท่ากันแบบไม่เข้มงวด
- === เท่ากันแบบเข้มงวด (ตรวจสอบประเภทข้อมูลด้วย)
- != ไม่เท่ากัน
- !== ไม่เท่ากันแบบเข้มงวด
- > มากกว่า
- < น้อยกว่า
- >= มากกว่าหรือเท่ากับ
- <= น้อยกว่าหรือเท่ากับ

☐ **ตัวดำเนินการตรรกะ (Logical Operators)**

- && (AND) ⟶ true && false
- || (OR) ⟶ true || false
- ! (NOT) ⟶ !true

☐ **ตัวดำเนินการกำหนดค่า (Assignment Operators)**

- = กำหนดค่า
- += เพิ่มค่าแล้วกำหนดใหม่
- -= ลดค่าแล้วกำหนดใหม่
- *= คูณค่าแล้วกำหนดใหม่

- /= หารค่าแล้วกำหนดใหม่

---

4. ฟังก์ชัน (Functions)

ฟังก์ชันคือกลุ่มของโค้ดที่สามารถเรียกใช้ซ้ำได้

รูปแบบการประกาศฟังก์ชัน

1 ฟังก์ชันปกติ (Function Declaration)

```
function sayHello() {
    console.log("Hello, World!");
}
```

2 ฟังก์ชันแบบนิพจน์ (Function Expression)

```
const greet = function() {
    console.log("Hello!");
};
```

3 ฟังก์ชันลูกศร (Arrow Function)

```
const sum = (a, b) => a + b;
```

---

5. คำสั่งควบคุมการไหลของโปรแกรม (Control Flow)

ใช้เพื่อควบคุมลำดับการทำงานของโค้ด

เงื่อนไข (Conditional Statements)

ใช้ if, else if, else เพื่อตรวจสอบเงื่อนไข

```
let score = 85;
if (score >= 80) {
    console.log("Grade A");
} else if (score >= 70) {
    console.log("Grade B");
} else {
    console.log("Grade C");
}
```

ลูป (Loops) ใช้สำหรับทำซ้ำคำสั่ง

```
for (let i = 0; i < 5; i++) {
    console.log("Hello", i);
}
```

---

6. การจัดการ DOM (Document Object Model)

DOM ช่วยให้ JavaScript สามารถเข้าถึงและเปลี่ยนแปลงโครงสร้าง HTML ได้

```
document.getElementById("myButton").addEventListener("click", function() {
    alert("Button clicked!");
});
```

ขั้นตอนการเรียนรู้ JavaScript อย่างมีประสิทธิภาพ

1 **เข้าใจพื้นฐานของ JavaScript**

- เรียนรู้เกี่ยวกับตัวแปร (let, const), ฟังก์ชัน และตัวดำเนินการ

2 **ฝึกใช้ JavaScript กับ HTML & CSS**

- ใช้ <script> แทรก JavaScript ในหน้าเว็บ
- ทดลองสร้างปุ่มที่ตอบสนองต่อการคลิก

3 **ศึกษา DOM และการจัดการเหตุการณ์ (Event Handling)**

- ใช้ getElementById() และ querySelector() เพื่อควบคุม HTML

4 **ฝึก Debugging ด้วย Console และ Developer Tools**

- ใช้ console.log() และ Breakpoint ใน Developer Tools

5 **ทำโปรเจ็กต์จริง**

- สร้าง To-Do List, Calculator หรือ Form Validation

6 **ศึกษาการใช้งาน JavaScript Frameworks**

- เช่น **React.js, Vue.js, Angular**

สรุป

- **Syntax ของ JavaScript** เป็นกฎพื้นฐานที่ใช้ในการเขียนโปรแกรม
- **องค์ประกอบหลักของ JavaScript** ได้แก่ ตัวแปร, ชนิดข้อมูล, ฟังก์ชัน, ตัวดำเนินการ, เงื่อนไข และลูป
- **DOM Manipulation** ช่วยให้ JavaScript ควบคุม HTML ได้
- **การเรียนรู้ JavaScript อย่างมีประสิทธิภาพ** ควรเริ่มจากพื้นฐานไปสู่การสร้างโปรเจ็กต์จริง

เมื่อเข้าใจ Syntax และโครงสร้างของ JavaScript แล้ว จะสามารถเขียนเว็บแอปพลิเคชันแบบโต้ตอบได้ง่ายขึ้น!

ตัวแปร (var, let, const) ใน JavaScript

ความหมายของตัวแปรใน JavaScript

ตัวแปร (Variable) ใน JavaScript คือ ที่เก็บข้อมูลที่สามารถนำมาใช้งานและเปลี่ยนแปลงได้ภายในโปรแกรม เช่น การเก็บตัวเลข ข้อความ หรือค่าต่าง ๆ ที่ใช้ในการคำนวณ

JavaScript มี 3 วิธีหลักในการประกาศตัวแปร ได้แก่

1. var ⟶ รูปแบบเก่า (ไม่แนะนำให้ใช้)
2. let ⟶ ใช้สำหรับค่าที่สามารถเปลี่ยนแปลงได้ (แนะนำให้ใช้)
3. const ⟶ ใช้สำหรับค่าคงที่ที่ไม่สามารถเปลี่ยนแปลงได้

ความจำเป็นของตัวแปร

- ใช้เก็บข้อมูลเพื่อนำไปใช้ในโปรแกรม
- ทำให้โค้ดสามารถจัดการกับข้อมูลได้อย่างมีประสิทธิภาพ
- ช่วยให้สามารถอ้างอิงข้อมูลได้ง่ายขึ้น
- ป้องกันข้อผิดพลาดจากการใช้ค่าคงที่ผิดพลาด (โดยใช้ const)

การใช้งานตัวแปร (var, let, const)

1. var – ตัวแปรรูปแบบเก่า

คุณสมบัติของ var

- มี Scope เป็น **Function Scope**
- สามารถใช้งานก่อนประกาศตัวแปรได้ (Hoisting)
- สามารถประกาศตัวแปรซ้ำได้โดยไม่มีข้อผิดพลาด

ตัวอย่างการใช้งาน var

```
function exampleVar() {
    var x = 10;
    if (true) {
        var x = 20; // ค่าของ x ถูกเปลี่ยนภายใน block
    }
    console.log(x); // แสดงค่า 20 (ค่าถูกเปลี่ยนจากภายใน if)
}

exampleVar();
```

ข้อเสียของ var

- ทำให้เกิดปัญหาจากการเปลี่ยนค่าภายใน Block Scope
- ใช้ Hoisting ได้ อาจทำให้เกิดพฤติกรรมที่คาดไม่ถึง

---

2. let – ตัวแปรที่สามารถเปลี่ยนแปลงค่าได้

 คุณสมบัติของ let

- มี Scope เป็น **Block Scope**
- ไม่สามารถใช้งานก่อนประกาศตัวแปรได้ (No Hoisting)
- ไม่สามารถประกาศตัวแปรซ้ำใน Scope เดียวกันได้

 ตัวอย่างการใช้งาน let

```
function exampleLet() {
    let y = 10;
    if (true) {
        let y = 20; // ตัวแปรนี้มี Scope อยู่ภายใน if เท่านั้น
        console.log(y); // แสดงค่า 20
    }
    console.log(y); // แสดงค่า 10 (ค่าภายใน if ไม่กระทบค่าภายนอก)
}

exampleLet();
```

 ข้อดีของ let

- ป้องกันปัญหาจากการเปลี่ยนค่าตัวแปรโดยไม่ตั้งใจ
- เหมาะสำหรับการใช้ตัวแปรที่ต้องการเปลี่ยนแปลงค่า

---

3. const – ตัวแปรค่าคงที่

 คุณสมบัติของ const

- มี Scope เป็น **Block Scope**
- ต้องกำหนดค่าเริ่มต้นทันทีที่ประกาศตัวแปร
- ไม่สามารถเปลี่ยนแปลงค่าของตัวแปรได้

 ตัวอย่างการใช้งาน const

```
function exampleConst() {
    const z = 10;
    // z = 20; //  จะเกิดข้อผิดพลาด เพราะ const เปลี่ยนค่าไม่ได้
    console.log(z); // แสดงค่า 10
}

exampleConst();
```

 ข้อดีของ const

- ป้องกันไม่ให้ค่าถูกเปลี่ยนแปลงโดยไม่ได้ตั้งใจ

- ใช้ได้กับค่าที่ไม่ควรถูกแก้ไข เช่น ค่าคงที่ (PI, URL API, ค่าคงที่ในโปรแกรม)

□ **ข้อควรระวัง**

- const ใช้ได้กับ **ค่าของตัวแปรเท่านั้น** แต่สามารถเปลี่ยนแปลงค่าภายใน **Object หรือ Array** ได้

```
const person = { name: "Alice" };
person.name = "Bob"; // □ สามารถเปลี่ยนค่าภายใน Object ได้
console.log(person.name); // แสดงค่า "Bob"
```

□ **เปรียบเทียบ var, let, const**

คุณสมบัติ	var	let	const
Scope	Function Scope	Block Scope	Block Scope
Hoisting	□ ได้	□ ไม่ได้	□ ไม่ได้
เปลี่ยนค่าได้ไหม	□ ได้	□ ได้	□ ไม่ได้
ประกาศซ้ำได้ไหม	□ ได้	□ ไม่ได้	□ ไม่ได้
เหมาะสำหรับ	ตัวแปรทั่วไป (ไม่แนะนำ)	ตัวแปรที่เปลี่ยนแปลงค่าได้	ค่าคงที่ที่ไม่เปลี่ยนแปลง

□ **คำแนะนำในการเลือกใช้ var, let, const**

□ **ใช้ const เป็นค่าเริ่มต้นเสมอ**

□ **ใช้ let เมื่อค่าของตัวแปรต้องเปลี่ยนแปลง**

□ **หลีกเลี่ยงการใช้ var เพราะอาจทำให้เกิดปัญหาจาก Scope และ Hoisting**

□ **สรุป**

- **var** ⟶ มี Function Scope, เปลี่ยนค่าได้, ประกาศซ้ำได้ แต่ทำให้เกิดปัญหา Scope และ Hoisting
- **let** ⟶ มี Block Scope, เปลี่ยนค่าได้ แต่ประกาศซ้ำไม่ได้ (แนะนำให้ใช้แทน var)
- **const** ⟶ มี Block Scope, ไม่สามารถเปลี่ยนค่าได้, เหมาะสำหรับค่าคงที่ที่ไม่ควรเปลี่ยนแปลง

ตัวอย่างโค้ดแบบเต็มสำหรับการใช้งานตัวแปร var, let, และ const ใน JavaScript พร้อมอธิบายการทำงานของแต่ละตัวแปร

□ **ตัวอย่างการใช้งาน var**