

INFOPRESS

พัฒนา Web Application ด้วย JavaScript และ Node.js

- เรียนรู้พื้นฐานและเทคนิคการพัฒนาเว็บไซต์ด้วยภาษา JavaScript อย่างละเอียด
- อธิบายการใช้งาน JavaScript ร่วมกัน Express และ MongoDB
- แนะนำการใช้งาน JavaScript/TypeScript บน Node และ Deno
- แนะนำเผยแพร่บริการ Web API แบบสาธารณะด้วย Heroku ร่วมกับ GitHub
- มีโค้ดตัวอย่างและคำอธิบายอย่างละเอียด อ่านเข้าใจง่าย
- อธิบายเป็นขั้นตอน เหมาะสำหรับผู้เริ่มต้น

ไฟล์ตัวอย่างภายในเล่ม
https://serazu.com/
9786164873582

ศุภชัย สมพานิช

IDC
PREMIER

มีเพียง “**ความรู้**” เท่านั้นที่มนุษย์ใช้พลิก “**โลก**”
และเปลี่ยน “**ชีวิต**” เราจึงสร้างสรรค์ และส่งมอบ “**ความรู้**”
ในรูปแบบที่ดีกว่า เพื่อให้คนไทย “**เรียนรู้**” ได้ตลอดชีวิต

Only “**Knowledge**” can help human
change “**The World**” and “**Their Lives**”.
With this truth, it drives us to deliver
“**Knowledge**” for Thai being able to
“**Learn**” better everyday.



Think Beyond
INFOPRESS
Read comics
Little HEART

พัฒนา Web Application ด้วย JavaScript และ Node.js

Author
ศุภชัย สมพานิช
Editorial
กิตินันท์ พลสวัสดิ์ kitinan_p@idcpremier.com
Production Manager
วรพล ณธิกุล
Graphic Designer
ชวนันท์ รัตนะ
Page Layout
วุฒิพันธ์ สมพระเมฆ
Proofreader
สุนทรี บรรลือศักดิ์
Publishing Coordinators
สุพัตรา อาจปรุ, สุรีย์รัตน์ จิอู๋
Product Specialist
ศรันย์ ชาติสุทธิผล

PUBLISHED AND DISTRIBUTED BY

บริษัท ไอดีซี พรีเมียร์ จำกัด
200 หมู่ 4 ชั้น 19 ห้อง 1901 อาคารจัสมินอินเตอร์เนชั่นแนล ทาวเวอร์ ถ.แจ้งวัฒนะ อ.ปากเกร็ด จ.นนทบุรี 11120
โทรศัพท์ 0-2962-1081 (อัตโนมัติ 10 คู่สาย)
โทรสาร 0-2962-1084
สมาชิกสัมพันธ์
โทรศัพท์ 0-2962-1081-3 ต่อ 121
โทรสาร 0-2962-1084
ร้านค้าและตัวแทนจำหน่าย
โทรศัพท์ 0-2962-1081-3 ต่อ 112-114
โทรสาร 0-2962-1084

Visual Studio Code เป็นเครื่องหมายการค้าของบริษัท Microsoft Corp., MongoDB เป็นเครื่องหมายการค้าของบริษัท MongoDB และเครื่องหมายการค้าอื่นๆ ที่อ้างถึงเป็นของบริษัทนั้นๆ

สงวนลิขสิทธิ์ตามพระราชบัญญัติลิขสิทธิ์ พ.ศ. 2537 โดยบริษัท ไอดีซี พรีเมียร์ จำกัด ห้ามลอกเลียนไม่ว่าส่วนใดส่วนหนึ่งของหนังสือเล่มนี้ ไม่ว่าในรูปแบบใดๆ นอกจากจะได้รับอนุญาตเป็นลายลักษณ์อักษรจากผู้จัดพิมพ์เท่านั้น

บริษัท ไอดีซี พรีเมียร์ จำกัด จัดตั้งขึ้นเพื่อเผยแพร่ความรู้ที่มีคุณภาพสู่ผู้อ่านชาวไทย เรายินดีรับงานเขียนของนักวิชาการและนักเขียนทุกท่าน ท่านผู้สนใจกรุณาติดต่อผ่านทางอีเมลที่ infopress@idcpremier.com หรือทางโทรศัพท์หมายเลข 0-2962-1081 (อัตโนมัติ 10 คู่สาย) โทรสาร 0-2962-1084

สร้างสรรค์โดย

พิมพ์ครั้งที่ 1 ตุลาคม 2565

ข้อมูลทางบรรณานุกรม
ศุภชัย สมพานิช
พัฒนา Web Application ด้วย JavaScript และ Node.js
นนทบุรี : ไอดีซีฯ, 2565
344 หน้า
1. JavaScript
2. ภาษาที่ใช้ ในการเขียนโปรแกรม
I ชื่อเรื่อง
005.262
BARCODE 885-916-101-075-3

ราคา 395 บาท

พิมพ์ที่ **บริษัท ส.พิจิตรการพิมพ์ จำกัด**
50/46 หมู่ 5 ต.ปากเกร็ด อ.ปากเกร็ด จ.นนทบุรี 11120
โทรศัพท์ 0-2587-0736 โทรสาร 0-2587-5758



คำนำ

ในปัจจุบัน หลักการพัฒนาโปรเจ็กต์แบ่งงานเป็นแบบ 2 ฝั่ง ได้แก่ ฝั่งหน้าบ้าน (Front End) และฝั่งหลังบ้าน (Back End) กลายเป็นเรื่องพื้นฐานไปแล้ว นักพัฒนาสามารถเลือกใช้งานภาษาต่างๆ มาใช้ทั้ง 2 ฝั่งได้อย่างอิสระ หนังสือเล่มนี้เลือกใช้ภาษา JavaScript/TypeScript รันบนแพลตฟอร์มของ Node ถือเป็นแพลตฟอร์มที่ได้รับความนิยมเป็นอย่างสูงอีกแพลตฟอร์มหนึ่ง

ช่วงท้ายของหนังสือนำเสนอวิธีการการเผยแพร่ (Deployment) งานฝั่งหลังบ้าน (Back End) โดยอาศัยบริการของ Heroku ทำงานร่วมกับ GitHub เพื่อให้ผู้อ่านเห็นกระบวนการตั้งแต่เริ่มต้นพัฒนาไปจนถึงขั้นตอนสุดท้าย นั่นคือ การเปิดบริการสู่สาธารณะ

คุณผู้อ่านท่านใดก็ตามที่ติดปัญหาหรือต้องการสอบถาม, ให้ข้อแนะนำ, ติติงรวมถึงข้อผิดพลาดต่างๆ ที่อาจจะเกิดขึ้นได้ สามารถติดต่อกับผู้เขียนได้ 5 ช่องทางคือ

1. แฟนเพจใน Facebook ชื่อว่า Thaivb.NET (https://www.facebook.com/thaivb.net)
2. กลุ่มพูดคุยเรื่องเขียนโปรแกรมใน Facebook ชื่อว่า Thaivb.NET (https://www.facebook.com/groups/Thaivb.NET)
3. ช่องใน YouTube ชื่อว่า Thaivb.NET
4. กลุ่มใน LINE ชื่อว่า Thaivb.NET
5. แฟนเพจใน Blockdit ชื่อว่า Thaivb.NET (https://www.blockdit.com/thaivb.net)

ตุลาคม 2565

ศุภชัย สมพานิช




JAVA

สารบัญ

บทที่ 1 JavaScript World 101 1

รูปแบบการนำเสนอสคริปต์ในหนังสือเล่มนี้ 1

JavaScript คืออะไร 2

การดาวน์โหลดและติดตั้ง PowerShell 3

การดาวน์โหลดและติดตั้ง Windows Terminal 4

การดาวน์โหลดและติดตั้งโปรแกรม Visual Studio Code 6

การติดตั้งฟอนต์ Cascadia Code สำหรับเขียนโปรแกรม 7

การกำหนดให้โปรแกรม VS Code ใช้ฟอนต์ Cascadia Code 9

บทที่ 2 พื้นฐานการใช้งานภาษา JavaScript 11

พื้นฐานการสร้างโปรเจ็กต์ JavaScript ในโปรแกรม Visual Studio Code 11

พื้นฐานการใช้งาน JavaScript ร่วมกับ HTML 13

การฝัง JavaScript กับอีลีเมนต์ตัวใดตัวหนึ่ง 14

การเขียน JavaScript ระหว่างอีลีเมนต์ <script>...</script> 15

การแยก JavaScript ออกเป็นไฟล์ต่างหาก 16

การใช้งาน Console ของบราวเซอร์ 18

ทำความรู้จักกับตัวแปร (Variable) ใน JavaScript (คำสั่ง var และ let) 19

ค่าคงที่ (คำสั่ง const) 22

ฟังก์ชัน (Function) และ Arrow Function 24

การกำหนดค่าเริ่มต้นของพารามิเตอร์ (Default Parameter) 27

ทำงานกับข้อความด้วย Template Literals (หรือ Template strings) 31

การสร้างพารามิเตอร์แบบรับได้หลายค่าด้วยตัวดำเนินการ 33

การแทรกอีลีเมนต์ HTML ด้วย JavaScript 35

บทที่ 3 ทำงานกับโครงสร้างข้อมูลอาร์เรย์ (Array) 37

พื้นฐานการสร้างอาร์เรย์ขึ้นมาใช้งาน 37

การสร้างอาร์เรย์จากฟังก์ชัน from() 39

การอ่านลำดับสมาชิกในอาร์เรย์ด้วยฟังก์ชัน keys() 40

การเพิ่มข้อมูลในอาร์เรย์ด้วยตัวดำเนินการ 41

พื้นฐานการทำงานกับโครงสร้างข้อมูลแบบอาร์เรย์ 43

การกรองข้อมูลด้วยฟังก์ชัน filter() 43

การอ่านข้อมูลในอาร์เรย์ด้วยฟังก์ชัน map() 44

การวนลูปอ่านข้อมูลในอาร์เรย์ด้วยฟังก์ชัน forEach() 45

การค้นหาข้อมูลในอาร์เรย์ด้วยฟังก์ชัน find() 45

การเพิ่มข้อมูลต่อท้ายเข้าไปในอาร์เรย์ด้วยฟังก์ชัน push() 46

การเพิ่มข้อมูลลำดับแรกเข้าไปในอาร์เรย์ด้วยฟังก์ชัน unshift() 47

การถอดข้อมูลล่าสุดออกจากอาร์เรย์ด้วยฟังก์ชัน pop() 47

การถอดข้อมูลลำดับแรกออกจากอาร์เรย์ด้วยฟังก์ชัน shift() 48

การถอดข้อมูลออกจากอาร์เรย์ด้วยฟังก์ชัน splice() 49

การตัด/แบ่งข้อมูลด้วยฟังก์ชัน slice() 51

การเรียงลำดับข้อมูลในอาร์เรย์ด้วยฟังก์ชัน sort() 54

การตรวจสอบข้อมูลในอาร์เรย์ด้วยฟังก์ชัน includes() 55

บทที่ 4 คำสั่งตรวจสอบและการวนลูป 57

การตรวจสอบเงื่อนไขด้วยคำสั่ง if 57

การตรวจสอบหลายเงื่อนไขด้วยคำสั่ง switch 59

การวนลูปแบบ for 61

การวนลูปแบบ for in 62

การวนลูปแบบ for of 63

การวนลูปแบบ while 65

การวนลูปแบบ do while 66

บทที่ 5 การเขียนโปรแกรมเชิงวัตถุใน JavaScript 69
พื้นฐานการเขียนโปรแกรมเชิงวัตถุ (Object Oriented Programming – OOP) 69
การสร้างออบเจ็กต์เก็บข้อมูลตามโครงสร้างที่เรากำหนด 73
การสร้างเมธอด (หรือฟังก์ชัน) ในคลาส 74
การสร้างเมธอดขอบเขตแบบ Private 75
การแปลงข้อมูลในออบเจ็กต์เป็นอาร์เรย์ด้วยฟังก์ชัน entries() และฟังก์ชัน values() 78

บทที่ 6 พื้นฐานการใช้งาน JavaScript ฝั่ง Server ด้วย Node 81
การดาวน์โหลดและติดตั้ง Node 81
พื้นฐานการใช้งานภาษา JavaScript บน Node 83
Express คืออะไร? 84
การดาวน์โหลดและติดตั้ง Express 85
พื้นฐานการสร้างโปรเจ็กต์ Express 86
การทดสอบรันและหยุดโปรเจ็กต์ Express 91
วิธีการจัดเก็บโปรเจ็กต์ Express 93
การเปิดโปรเจ็กต์ Express แบบปกติ 94
วิธีการ Restart โปรเจ็กต์ Express โดยอัตโนมัติ 96
ระบบเส้นทาง (Route) ของ Express ในขั้นต้น 101
พื้นฐานการจัดการพาธอื่นๆ ที่ไม่มี (404 page not found) 102
การใช้งาน JavaScript ร่วมกับ Express เบื้องต้น 103
การใช้งานไฟล์ Static ร่วมกับ Express (Static File) 105

บทที่ 7 พื้นฐานการใช้งานระบบฐานข้อมูล MongoDB 109
ระบบฐานข้อมูล NoSQL แบบ MongoDB คืออะไร 109
การดาวน์โหลดและติดตั้งฐานข้อมูล MongoDB 110
พื้นฐานการเชื่อมต่อกับฐานข้อมูล MongoDB ด้วย MongoDB Compass 112
การสร้างฐานข้อมูล MongoDB ใหม่ 113
การจัดการข้อมูลในฐานข้อมูล MongoDB เบื้องต้น 118
การ Export ข้อมูลออกจากฐานข้อมูล MongoDB 121
การ Import ข้อมูลเข้ามาในระบบฐานข้อมูล MongoDB 124

บทที่ 8 การสร้างบริการ CRUD ด้วย Express ร่วมกับฐานข้อมูล MongoDB 125

การเชื่อมต่อ Express กับฐานข้อมูล MongoDB (Read) 125

การทดสอบการทำงานของ Web API ด้วย REST Client 139

การสร้างบริการเพิ่ม, แก้ไข และลบข้อมูล (Create, Update และ Delete) 143

บทที่ 9 การสร้างส่วนแสดงผลด้วย Angular 155

การดาวน์โหลดและติดตั้ง Angular 156

การสร้างโปรเจ็กต์ Angular แรก 157

การทดสอบและหยุดรันโปรเจ็กต์ Angular 159

การติดตั้งส่วนขยาย (Extension) สำหรับ Angular ให้กับโปรแกรม Visual Studio Code 161

หลักการสร้างส่วนแสดงผลด้วย Angular 161

การสร้างส่วนแสดงผลสำหรับแสดงในหน้าจอทุกขนาดด้วย Bootstrap (Responsive Design) 168

การติดตั้ง Bootstrap ให้กับโปรเจ็กต์ Angular 168

ทำความรู้จักกับตัวบรรจุ container ของ Bootstrap 172

ทำความรู้จักกับระบบ Grid ของ Bootstrap 174

ทำความรู้จักกับขนาดของ Grid 176

บทที่ 10 การใช้งาน Express ร่วมกับ Angular (MEAN Stack) 181

การเพิ่ม Component สำหรับเมนูหลัก 182

การสร้างแถบ Footer 188

การสร้างส่วนแสดงผลสำหรับงาน CRUD ด้วย Angular 190

การสร้างบริการส่วนกลาง (Service) ใน Angular 190

การสร้างส่วนแสดงผลสำหรับงาน CRUD 195

การกำหนดเส้นทาง (Route) สำหรับงาน CRUD 204

การทดสอบการทำงาน CRUD ของ MEAN Stack 206

อธิบายการทำงานของสคริปต์การทำงาน CRUD 212

บทที่ 11 การสร้าง Express แบบมีส่วนแสดงผลด้วย Pug 227

การสร้างโปรเจ็กต์ Express ด้วย Express Generator 228

หลักการสร้างส่วนแสดงผลด้วย Pug 233

บทที่ 12 การสร้าง CRUD ด้วย Express ร่วมกับ Pug 237

การสร้างโปรเจ็กต์ Express และ Pug ด้วย Express Generator 237

การเขียนสคริปต์สำหรับโปรเจ็กต์ CRUD และสร้างส่วนแสดงผลด้วย Pug 241

การสร้างส่วนแสดงผลด้วย Pug 247

การทำงาน CRUD ของ Express ร่วมกับ Pug 253

อธิบายการทำงานของสคริปต์ของโปรเจ็กต์ Express และ Pug ด้วย Express Generator 258

บทที่ 13 การใช้งาน JavaScript/TypeScript บนแพลตฟอร์ม Deno 267

การดาวน์โหลดและติดตั้ง Deno 268

พื้นฐานการรัน JavaScript บน Deno 269

การติดตั้งส่วนขยายสำหรับพัฒนาแอปบน Deno 270

พื้นฐานการสร้าง Web API บน Deno ด้วย OAK Framework 272

การระบุเวอร์ชันให้กับ OAK 275

การจัดการเส้นทาง (Route) ของ OAK 277

พื้นฐานการสร้างพาธรองรับงาน CRUD 279

บทที่ 14 การใช้งานฐานข้อมูล MongoDB ด้วยบริการคลาวด์ของ MongoDB Atlas 287

การเปิดใช้บริการคลาวด์ของ MongoDB Atlas 288

การเชื่อมต่อโปรเจ็กต์ Express เข้ากับบริการ MongoDB Atlas แบบ localhost 298

บทที่ 15 การเปิดบริการ Web API สู่ภายนอกด้วย Heroku 301

Git คืออะไร? 301

การดาวน์โหลดและติดตั้งโปรแกรม Git 302

การจัดเก็บโปรเจ็กต์ใน GitHub 307

การเตรียมโปรเจ็กต์ Express ให้พร้อมใช้งานบริการของ Heroku 310

การอัพโหลด (push) โปรเจ็กต์ Express ไปเก็บใน GitHub 313

การใช้บริการ Heroku เบื้องต้น 319

การอัพโหลดโปรเจ็กต์ Express เข้าสู่ Heroku ด้วยวิธี Heroku CLI 322

การอัพโหลดโปรเจ็กต์ Express เข้าสู่ Heroku ด้วยวิธีดึงโปรเจ็กต์จาก GitHub 328


JavaScript



บทที่ 1

JavaScript World 101

หน้าที่ของภาษา JavaScript เดิม ถูกสร้างขึ้นมาเพื่อทำงานในบราวเซอร์ช่วยให้การแสดงผลหน้าเว็บเพจที่เกิดจาก HTML และ CSS เท่านั้น ถือเป็นการใช้งานภาษา JavaScript ฝั่ง Client แต่ในยุคปัจจุบันภาษา JavaScript มีบทบาทในโลกของการพัฒนา Desktop Apps, Mobile Apps และ Web Apps ครบถ้วนแล้ว

รูปแบบการนำเสนอสคริปต์ในหนังสือเล่มนี้

เนื้อหาของหนังสือเล่มนี้ เป็นการใช้งานภาษา JavaScript ทั้งฝั่ง Client กับฝั่ง Server ประกอบไปด้วย 2 ส่วนใหญ่ๆ คือ

1. พื้นฐานการใช้งานภาษา JavaScript ฝั่ง Client (รันในบราวเซอร์) เพื่อปูพื้นฐานและทำความรู้จักในขั้นต้น ถือเป็นงานฝั่งหน้าบ้าน (Front End) เพื่อสร้างส่วนแสดงผล (**User Interface** เรียกย่อๆ ว่า **UI**) เป็นหน้าเว็บเพจร่วมกับ HTML
2. การใช้งานภาษา JavaScript ฝั่ง Server เกิดจากการนำภาษา JavaScript ไปใช้พัฒนาระบบงานฝั่งหลังบ้าน (Back End) หรือการสร้าง Web API ขึ้นมาเพื่อทำหน้าที่บริการด้านต่างๆ มี 2 แพลตฟอร์มที่น่าสนใจคือ
   - แพลตฟอร์ม Node
   - แพลตฟอร์ม Deno

หมายความว่า เนื้อหาของหนังสือเล่มนี้ครอบคลุมงานทั้ง 2 ฝั่งคือ Front End และ Back End ด้วยภาษา JavaScript และยังรวมไปถึงภาษา TypeScript ด้วย จะกล่าวถึงรายละเอียดอีกครั้งเมื่อเนื้อหามีความเกี่ยวข้องแล้ว

JavaScript คืออะไร

โลกของเว็บในยุคเริ่มต้น มีองค์ประกอบพื้นฐาน 3 ส่วนคือ

1. **HTML** ทำหน้าที่สร้างส่วนแสดงผล เมื่อผู้อ่านเข้าสู่โลกของ Web Apps ทุกแพลตฟอร์ม นี่คือรากฐานของการสร้างส่วนแสดงผล
2. **CSS** ทำหน้าที่ตกแต่งส่วนแสดงผล
3. **JavaScript** ทำหน้าที่เพิ่มประสิทธิภาพและจัดการการทำงานต่างๆ ที่เกิดขึ้นในส่วนแสดงผลให้ดียิ่งขึ้น

ทั้ง 3 ส่วนทำงานร่วมกันใน Browser ประกอบกันมาเป็นหน้าเว็บเพจต่างๆ ที่เราพบเห็นได้โดยทั่วไป หลักการนี้ยังคงยึดถือทุกยุคจนถึงปัจจุบัน

ในยุคปัจจุบัน JavaScript ไม่ได้ทำงานอยู่ในบราวเซอร์ (หรือฝั่ง Client) เพียงอย่างเดียวอีกต่อไปแล้ว มีการนำภาษา JavaScript (ผู้เขียนขอกล่าวรวมถึงอีกภาษาคือ ภาษา TypeScript รวมไปด้วย) ไปรันฝั่ง Server อีกด้วย ซึ่งมี 2 แพลตฟอร์มหลักคือ

1. Node Platform
2. Deno Platform

หมายความว่า เมื่อเราพูดถึง JavaScript (หรือ TypeScript) จึงมีโลกอยู่ 2 ใบคือ

1. JavaScript ทำงานในบราวเซอร์ หรือฝั่ง Client
2. JavaScript ทำงานบน Node หรือ Deno หรือฝั่ง Server

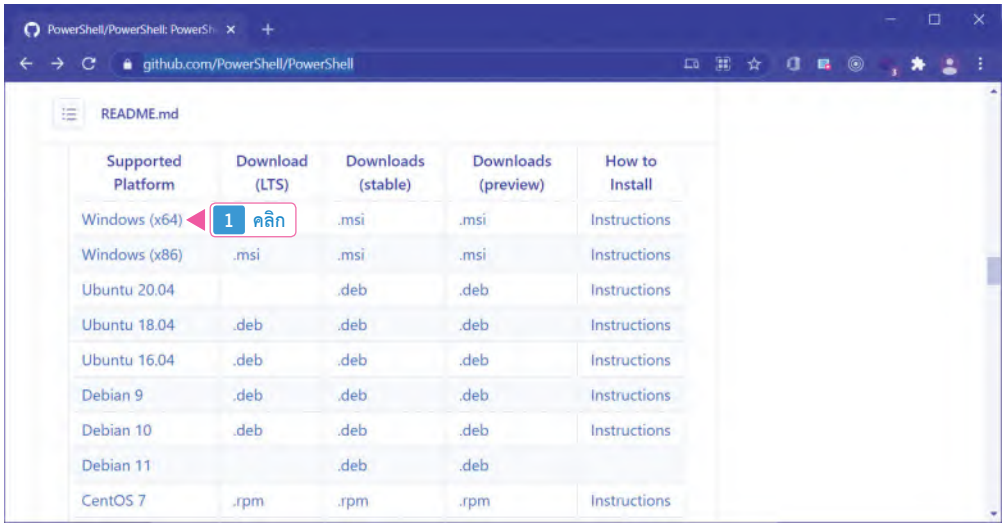
ในยุคปัจจุบันภาษา JavaScript มีการอัพเดตฟีเจอร์ทางภาษาโดยใช้มาตรฐาน ECMAScript + ระบุปี ค.ศ. กำกับไว้ เช่น ES2015, ES6 เป็นต้น เนื้อหาที่นำเสนอในหนังสือเล่มนี้ยึดถือตามแนวทางการเขียนภาษา JavaScript สมัยใหม่เป็นหลัก

อีกประเด็นหนึ่งที่ผู้เขียนขอกล่าวไว้ตั้งแต่ตอนต้นเลยก็คือ ภาษา JavaScript สมัยใหม่ไม่จำเป็นต้องใช้เครื่องหมาย ; ปิดท้ายบรรทัดแล้ว แต่ผู้เขียนยังคงใช้เครื่องหมาย ; ปิดท้ายเช่นเดิมเพื่อให้สคริปต์ดูง่ายขึ้น ส่งผลให้ผู้อ่านเห็นว่าจบบรรทัดแล้วนั่นเอง

การดาวน์โหลดและติดตั้ง PowerShell

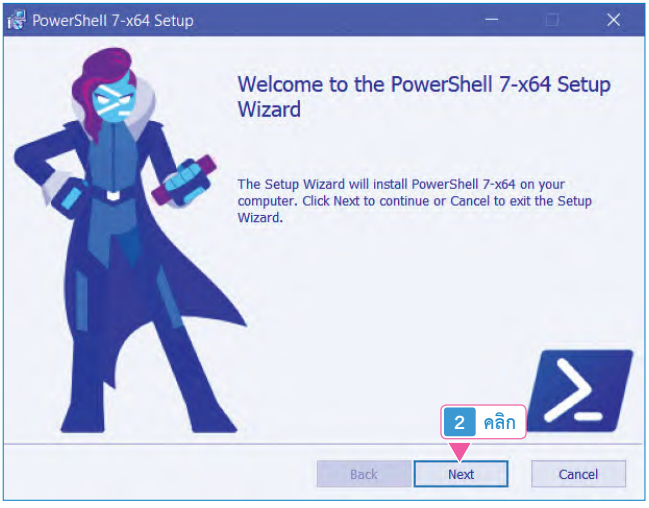
เพื่อให้ Windows 10 หรือ Windows 11 รองรับการเขียนโปรแกรมสมัยใหม่ มีข้อแนะนำให้ผู้อ่านติดตั้ง PowerShell เวอร์ชันล่าสุดได้ที่เว็บไซต์ https://github.com/PowerShell/PowerShell

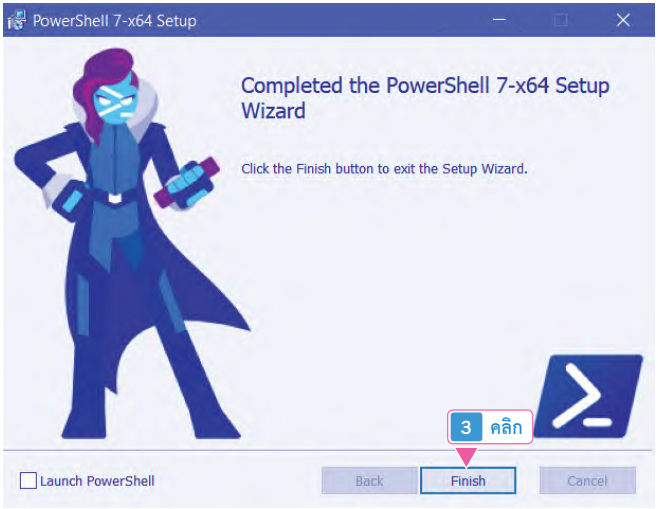
ในกรณีนี้ ผู้เขียนใช้งาน Windows 10/11 แบบ 64 บิต ให้เลือกดาวน์โหลดแบบ Windows (x64) ดังรูปที่ 1-1



รูปที่ 1-1 แสดงไฟล์ติดตั้ง PowerShell

ให้ผู้อ่านดับเบิลคลิกไฟล์ msi ที่ดาวน์โหลดมาเพื่อเข้าสู่ขั้นตอนการติดตั้ง PowerShell เวอร์ชันล่าสุด รอขั้นตอนการติดตั้งจนเสร็จสมบูรณ์ ดังรูปที่ 1-2



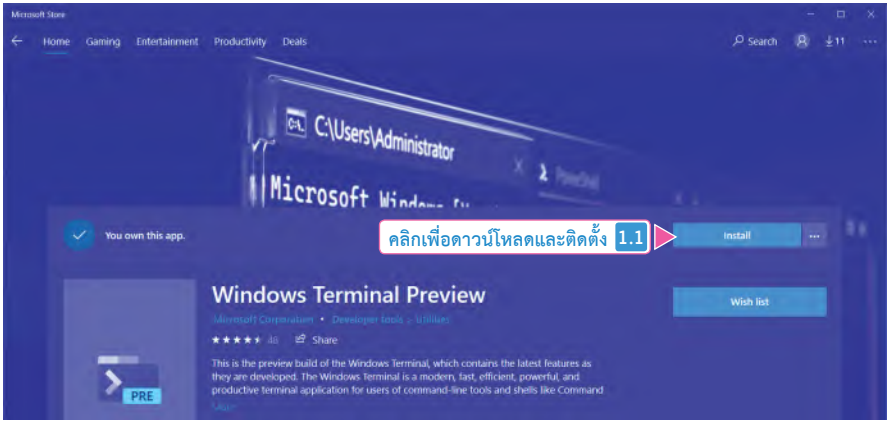


รูปที่ 1-2 แสดงการติดตั้ง PowerShell เวอร์ชันล่าสุด

การดาวน์โหลดและติดตั้ง Windows Terminal

ในยุคของ Windows 10/11 มีวิธีการป้อนคำสั่งหลายรูปแบบ แตกต่างจากยุคก่อนหน้านี้ที่เราคุ้นเคยกันเพียงรูปแบบเดียว นั่นคือ Command Prompt ไมโครซอฟท์สร้างแอปที่ชื่อว่า Windows Terminal ขึ้นมา ช่วยให้นักพัฒนาทุกภาษาสามารถเรียกใช้การป้อนคำสั่งได้ภายในแอป Windows Terminal โดยมีขั้นตอนดังนี้

1. ให้ผู้อ่านเปิด Microsoft Store ขึ้นมา โดยการคลิกเมนู Start > Microsoft Store จากนั้นค้นหาด้วยคำว่า "Windows Terminal" ให้ผู้อ่านรอการติดตั้งจนเสร็จสมบูรณ์ และ Restart เครื่อง 1 ครั้ง ดังรูปที่ 1-3

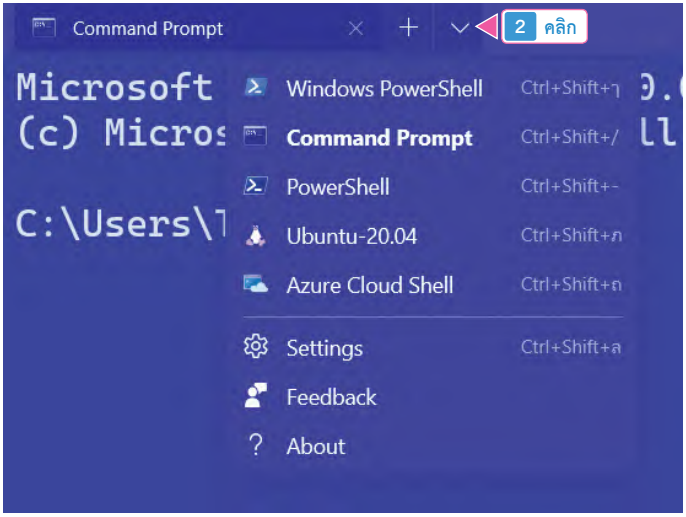




รูปที่ 1-3 แสดงการดาวน์โหลดและติดตั้ง Windows Terminal

2. ท้ายที่สุด ให้ผู้อ่านเปิด Windows Terminal ขึ้นมาก่อน แล้วคลิกปุ่ม ⌄ เพื่อเลือกว่า ต้องการป้อนคำสั่งด้วยวิธีใด ประกอบด้วย

- Windows PowerShell
- Command Prompt
- PowerShell
- Ubuntu
- Azure Cloud Shell

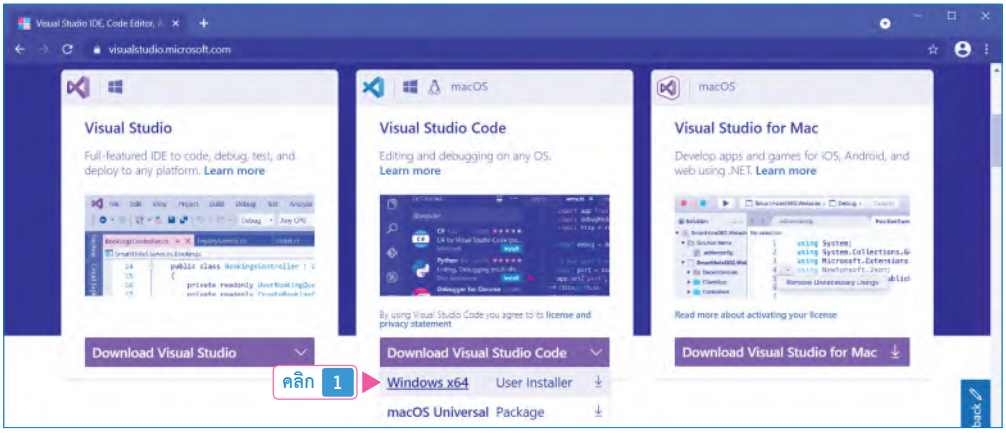


รูปที่ 1-4 แสดงวิธีการป้อนคำสั่งที่ Windows Terminal รองรับ

การดาวน์โหลดและติดตั้งโปรแกรม Visual Studio Code

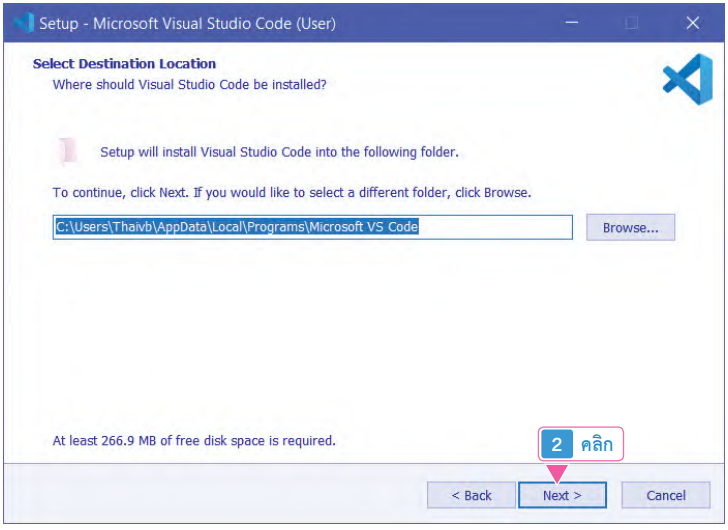
สำหรับโปรแกรมหลักที่เราจะใช้เขียน HTML, CSS และ JavaScript ในหนังสือเล่มนี้คือโปรแกรม Visual Studio Code เรียกย่อๆ ว่า VS Code ผู้อ่านสามารถดาวน์โหลดมาใช้งานได้ฟรีที่ http://www.visualstudio.com/

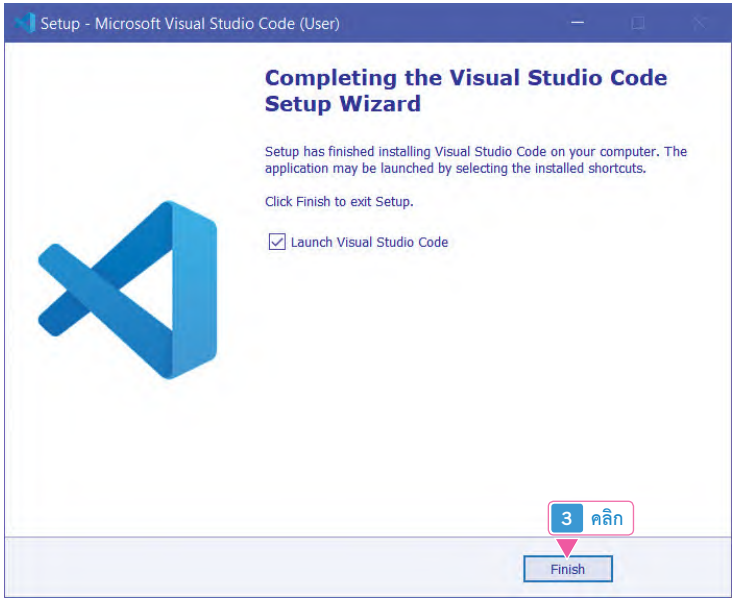
ในกรณีนี้ ผู้เขียนใช้ Windows 10/Windows 11 แบบ 64 บิต เลือก Windows x64 ดังรูปที่ 1-5



รูปที่ 1-5 แสดงเว็บไซต์ของไมโครซอฟท์ที่ให้ดาวน์โหลดโปรแกรม VS Code

ให้ผู้อ่านดับเบิลคลิกไฟล์ติดตั้งที่ดาวน์โหลดมา ใช้ค่าเริ่มต้นที่มากับตัวติดตั้งทั้งหมด รอจนเสร็จสมบูรณ์ ดังรูปที่ 1-6





รูปที่ 1-6 แสดงการติดตั้งโปรแกรม VS Code

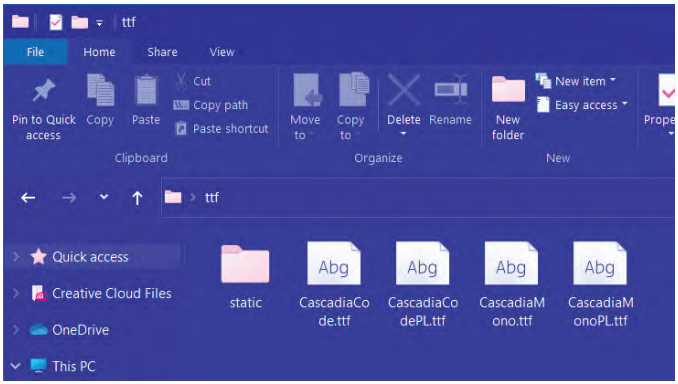
การติดตั้งฟอนต์ Cascadia Code สำหรับเขียนโปรแกรม

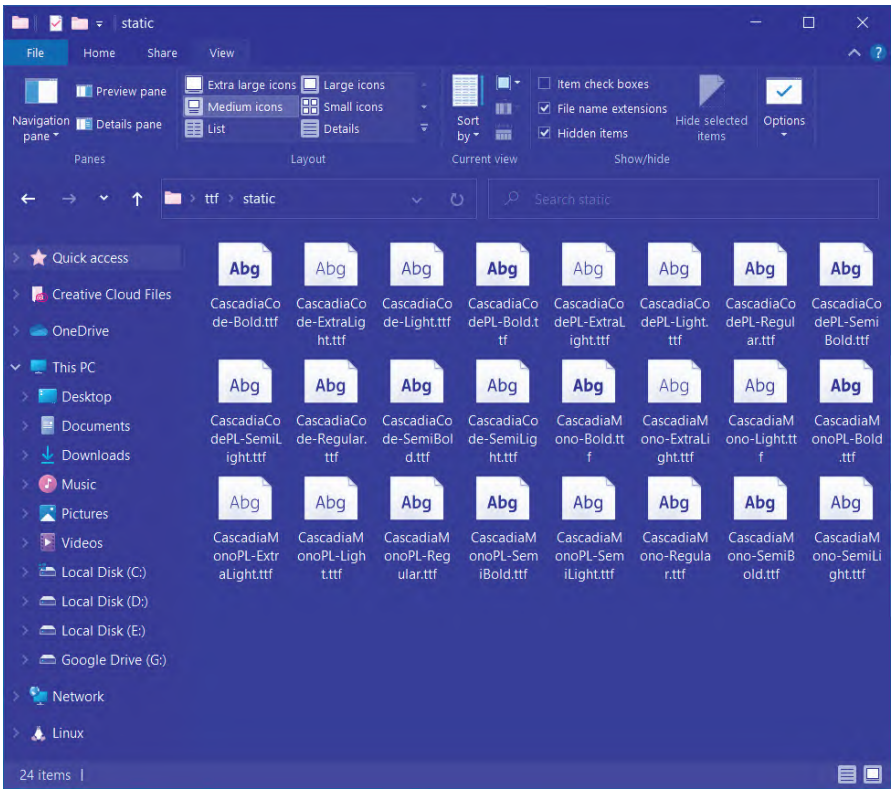
Cascadia Code เป็นฟอนต์ของไมโครซอฟท์ ถูกสร้างขึ้นมาสำหรับงานด้านเขียนโปรแกรม โดยเฉพาะ ผู้อ่านสามารถดาวน์โหลดมาใช้งานได้ที่ https://github.com/microsoft/cascadia-code/releases ดังรูปที่ 1-7



รูปที่ 1-7 แสดงการดาวน์โหลดฟอนต์ Cascadia Code

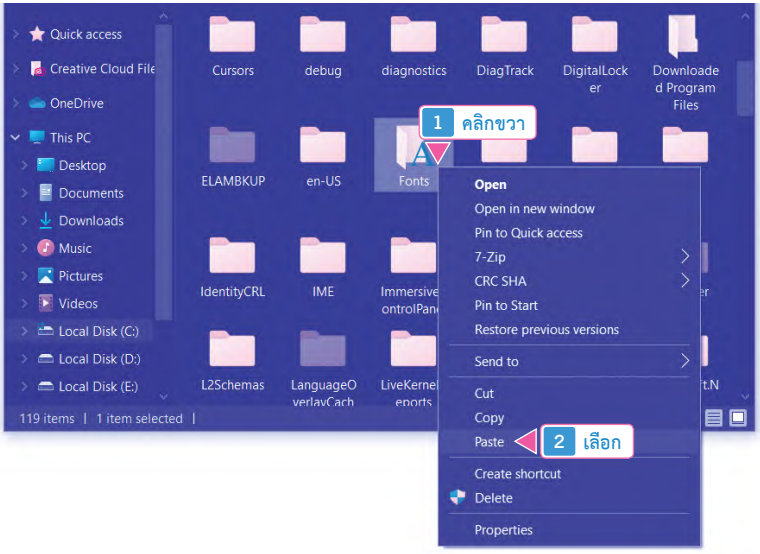
จากรูปที่ 1-7 ให้ผู้อ่านแตกไฟล์ zip ออกมา เราสนใจฟอนต์ประเภท ttf เท่านั้น ดังรูปที่ 1-8





รูปที่ 1-8 แสดงฟอนต์ประเภท ttf

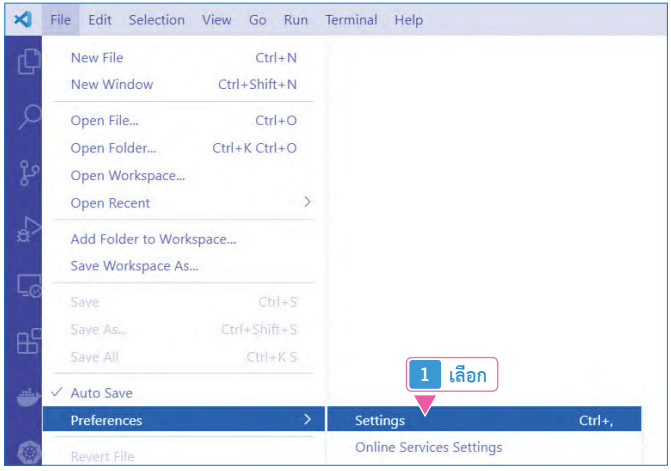
จากรูปที่ 1-8 ไฟล์ฟอนต์ (นามสกุล .ttf) ทั้งด้านนอกและในโฟลเดอร์ static ขอให้ผู้อ่าน Copy มาวางไว้ในโฟลเดอร์ C:\Windows\Fonts เพื่อติดตั้งฟอนต์ Cascadia Code ให้กับ Windows ดังรูปที่ 1-9

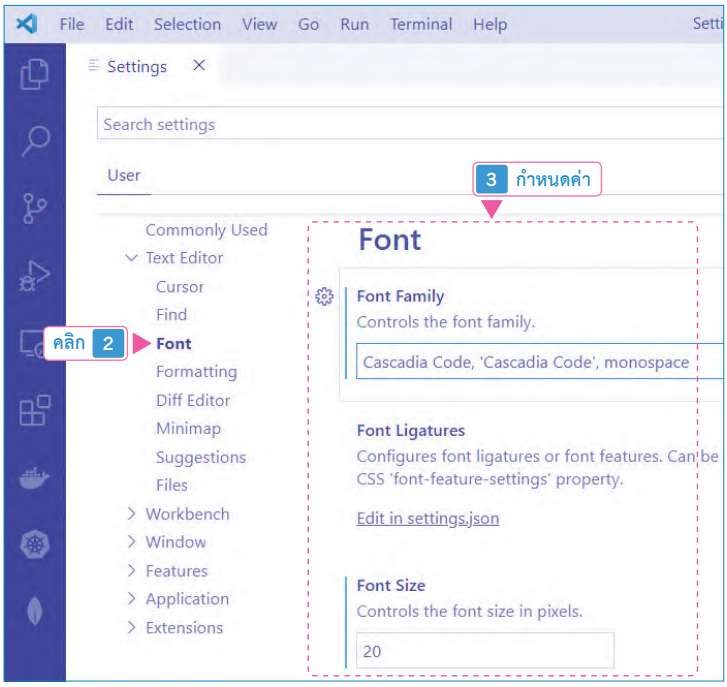


รูปที่ 1-9 แสดงการติดตั้งฟอนต์ Cascadia Code ให้กับ Windows

การกำหนดให้โปรแกรม VS Code ใช้ฟอนต์ Cascadia Code

ผู้อ่านสามารถกำหนดให้โปรแกรม VS Code ใช้งานฟอนต์ Cascadia Code ได้โดยการคลิกเมนู File > Preferences > Settings จากนั้นที่หัวข้อ Font ให้กำหนดค่า Cascadia Code, 'Cascadia Code', monospace ในกรณีนี้ต้องการใช้งานฟอนต์ขนาด 20 หรือตามขนาดที่ผู้อ่านต้องการ ดังรูปที่ 1-10





รูปที่ 1-10 แสดงการใช้งานฟอนต์ Cascadia Code ขนาด 20

สรุปท้ายบท

เนื้อหาของบทนี้ เป็นเพียงการเกริ่นนำให้ผู้อ่านทราบขอบเขตเนื้อหาที่นำเสนอในหนังสือเล่มนี้ว่า เรากำลังศึกษาเรื่องอะไรบ้าง และจะนำเสนอตั้งแต่บทที่ 2 เป็นต้นไป



บทที่ 2

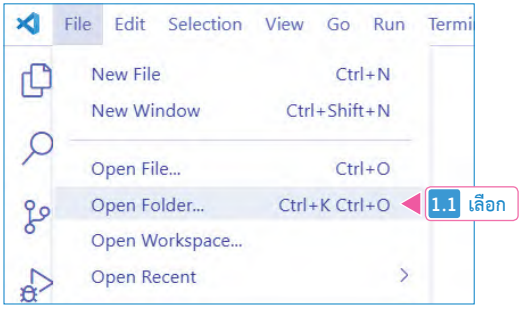
พื้นฐานการใช้งานภาษา JavaScript

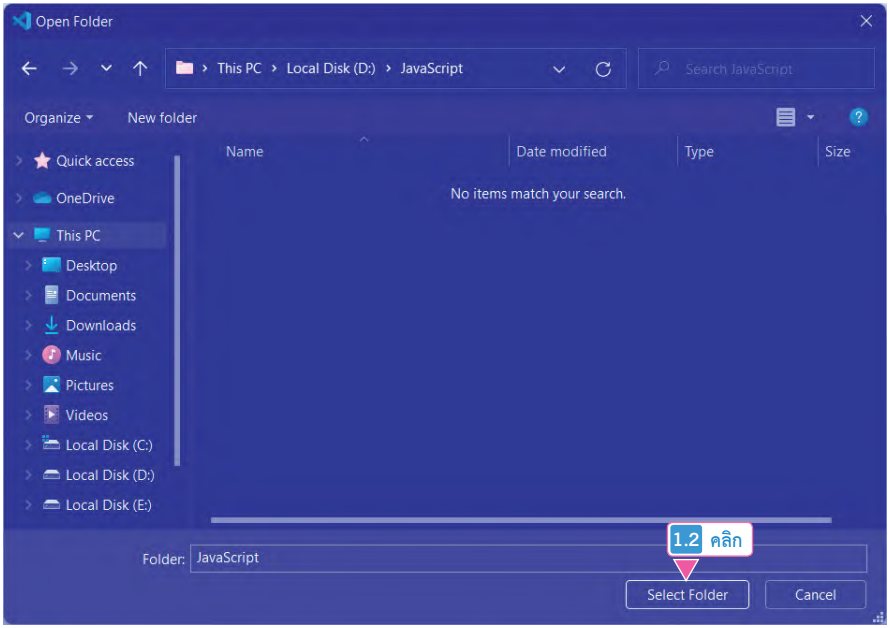
การศึกษาใช้งานภาษา JavaScript ลำดับแรกคือ เป็นการทำงานในฝั่ง Client ร่วมกับไฟล์ HTML เพื่อสร้างส่วนแสดงผลต่างๆ รันในบราวเซอร์

พื้นฐานการสร้างโปรเจ็กต์ JavaScript ในโปรแกรม Visual Studio Code

การเขียนโปรแกรมด้วยภาษา JavaScript ของหนังสือเล่มนี้ใช้งานโปรแกรม Visual Studio Code เป็นหลัก มีขั้นตอนดังนี้

1. ให้ผู้อ่านเปิดโปรแกรม Visual Studio Code ขึ้นมาก่อน แล้วคลิกเมนู File > Open Folder... เพื่อเลือกโฟลเดอร์จัดเก็บโปรเจ็กต์ของเรา ในกรณีนี้คือ เลือกพาธ D:\JavaScript (หรือพาธอื่นๆ ตามที่ต้องการ) ดังรูปที่ 2-1

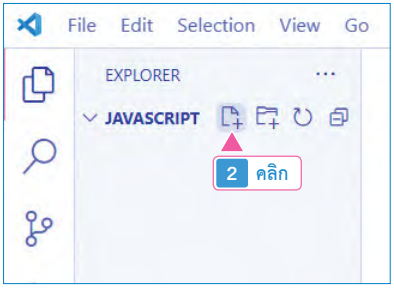






รูปที่ 2-1 แสดงการเลือกพารจัดเก็บโปรเจ็กต์

2. การใช้งาน JavaScript ฝั่ง Client ในระดับพื้นฐาน ผู้เขียนต้องการทดสอบการทำงานกับไฟล์ HTML โดยการคลิกปุ่ม สร้างไฟล์ที่ชื่อว่า index.html ขึ้นมาก่อน ดังรูปที่ 2-2

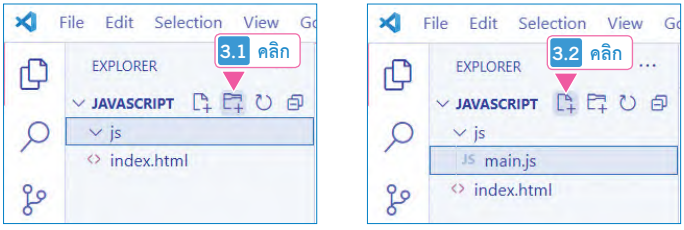




รูปที่ 2-2 แสดงการสร้างไฟล์ HTML ที่ชื่อว่า index.html

3. ท้ายที่สุด เมื่อได้ไฟล์ HTML มาแล้ว ก็จะสร้างไฟล์ JavaScript ต่อ โดยปกติแล้วเราจะไม่สร้างไฟล์ JavaScript อยู่ในระดับเดียวกับไฟล์ HTML แต่จะสร้างโฟลเดอร์ขึ้นมาเก็บไฟล์ JavaScript โดยเฉพาะ

ให้คลิกปุ่ม สร้างโฟลเดอร์ที่ชื่อว่า js ขึ้นมาก่อน จากนั้นคลิกปุ่ม สร้างไฟล์ JavaScript ในกรณีนี้ตั้งชื่อว่า main.js ดังรูปที่ 2-3



รูปที่ 2-3 แสดงการสร้างไฟล์ main.js ในโฟลเดอร์ js

จากรูปที่ 2-3 เราได้ไฟล์ index.html ทำหน้าที่ทดสอบการทำงานของไฟล์ main.js ที่เก็บอยู่ในโฟลเดอร์ js แล้ว ก็ขึ้นอยู่กับว่าเราต้องการทำงานอะไรบ้างกับ HTML

พื้นฐานการใช้งาน JavaScript ร่วมกับ HTML

การใช้งานภาษา JavaScript ในระดับพื้นฐานก็คือ การใช้งานร่วมกับ HTML ทำงานฝั่ง Client ถือเป็นวิธีการดั้งเดิมตั้งแต่โลกยุคแรกของเว็บ มี 3 รูปแบบคือ

- การฝัง JavaScript กับอีลีเมนต์ตัวใดตัวหนึ่ง
- การเขียน JavaScript ระหว่างอีลีเมนต์ <script>...</script>
- การแยก JavaScript ออกเป็นไฟล์ต่างหาก

การฝัง JavaScript กับอีลีเมนต์ตัวใดตัวหนึ่ง

วิธีนี้เป็นวิธีการใช้งาน JavaScript ที่มีขอบเขตแคบที่สุด กำหนดตายตัวว่า อีลีเมนต์ใดเป็นผู้ใช้งาน

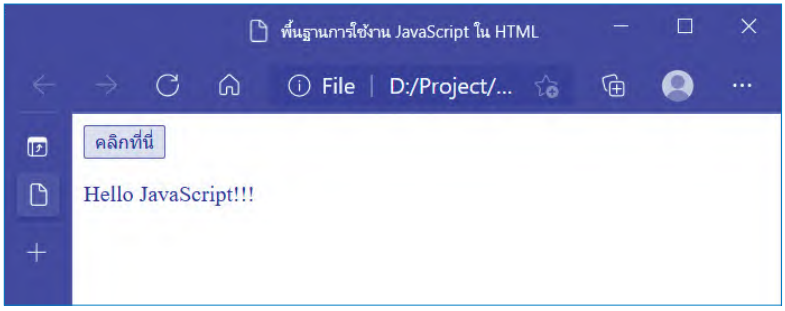
ตัวอย่างที่ 2-1 การฝัง JavaScript กับอีลีเมนต์ตัวใดตัวหนึ่ง

ต้องการสร้างปุ่ม button ให้แสดงคำว่า "Hello JavaScript!!!" ในส่วนแสดงผล ดังสคริปต์ต่อไปนี้

สคริปต์ HTML5 ที่ 2-1 การฝัง JavaScript กับอีลีเมนต์ตัวใดตัวหนึ่ง (BasicJavaScript.html)

```
<body>
    <div>
        <button type="button"
                onclick="document.getElementById('output').innerHTML = 'Hello JavaScript!!!'">
            คลิกที่นี่
        </button>
        <p id="output"></p>
    </div>
</body>
```

เมื่อรันสคริปต์จะได้ผลลัพธ์ดังรูป



รูปที่ 2-4 แสดงข้อความที่เกิดจากการทำงานของ JavaScript

อธิบายการทำงานของสคริปต์

1. เริ่มต้น สร้างย่อหน้าใหม่ด้วยอีลีเมนต์ ... ตั้งชื่อว่า output ก่อน

BasicJavaScript.html

```
<p id="output"></p>
```

2. ท้ายที่สุด สร้างปุ่ม button (type="button") ขึ้นมาก่อน เมื่อผู้ใช้งานคลิกปุ่ม button ส่งผลให้เกิดเหตุการณ์ click (onclick) ก็จะสั่งให้พื้นที่ที่ชื่อว่า output ที่เราสร้างไว้ก่อนหน้านี้แสดงข้อความ "Hello JavaScript!!!"

BasicJavaScript.html

```
<button type="button"
        onclick="document.getElementById('output').innerHTML = 'Hello JavaScript!!!'">
    คลิกที่นี่
</button>
```

การเขียน JavaScript ระหว่างอีลีเมนต์ <script>...</script>

วิธีนี้เป็นวิธีการเขียนภาษา JavaScript แยกออกมาต่างหาก อยู่ระหว่างอีลีเมนต์ <script>...</script>

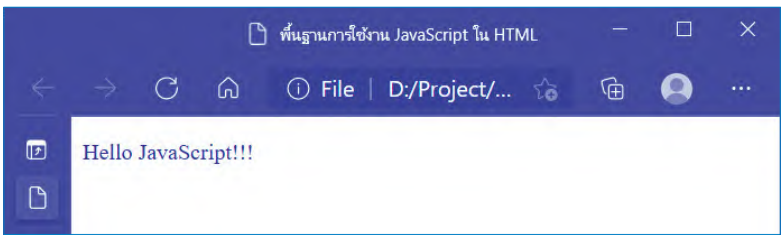
ตัวอย่างที่ 2-2 การเขียน JavaScript ระหว่างอีลีเมนต์ <script>...</script> ต้องการแสดงข้อความ "Hello JavaScript!!!" ในพื้นที่ที่ชื่อว่า output ดังสคริปต์ต่อไปนี้

สคริปต์ HTML5 ที่ 2-2 การเขียน JavaScript ระหว่างอีลีเมนต์ <script>...</script> (BasicJavaScript.html)

```
<body>
    <div>
        <p id="output"></p>
    </div>

    <script>
        document.getElementById('output').innerHTML = 'Hello JavaScript!!!';
    </script>
</body>
```

เมื่อรันสคริปต์จะได้ผลลัพธ์ดังรูป



รูปที่ 2-5 แสดงข้อความที่เกิดจาก JavaScript

การแยก JavaScript ออกเป็นไฟล์ต่างหาก

วิธีนี้เป็นวิธีการใช้งาน JavaScript ที่มีขอบเขตกว้างมากที่สุด เพราะว่าผู้อ่านแยก JavaScript ออกเป็นไฟล์ต่างหากแล้ว มีนามสกุล .js เมื่อต้องการใช้งานก็อ้างอิงไฟล์ .js โดยตรง

ตัวอย่างที่ 2-3 การแยก JavaScript ออกเป็นไฟล์ต่างหาก มีขั้นตอนดังนี้

1. สร้างโฟลเดอร์ที่ชื่อว่า js ทำหน้าที่เก็บไฟล์ JavaScript โดยเฉพาะ ตั้งชื่อไฟล์แรกนี้ว่า main.js จากนั้นให้เขียนสคริปต์ดังต่อไปนี้

สคริปต์ JavaScript ที่ 2-3 การแยก JavaScript ออกเป็นไฟล์ต่างหาก (\js\main.js)

```
function showmessage() {
    document.getElementById("output").innerHTML = "Hello JavaScript!!!";
}
```



รูปที่ 2-6 แสดงไฟล์ main.js ในโฟลเดอร์ js

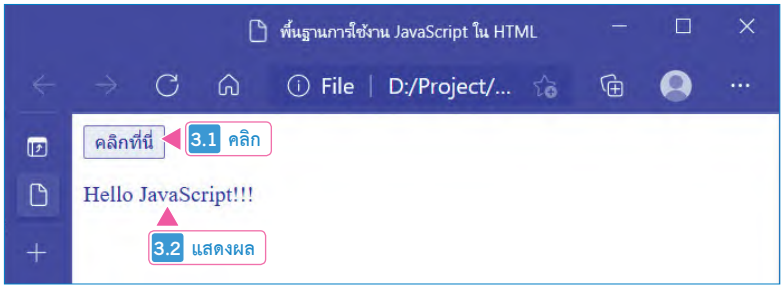
2. ในไฟล์ BasicJavaScript.html เรียกใช้งานไฟล์ main.js ดังสคริปต์ต่อไปนี้

สคริปต์ HTML5 ที่ 2-3 การแยก JavaScript ออกเป็นไฟล์ต่างหาก (BasicJavaScript.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>พื้นฐานการใช้งาน JavaScript ใน HTML</title>
</head>
<body>
    <div>
        <button type="button" onclick="showmessage()">คลิกที่นี่</button>
        <p id="output"></p>
    </div>

    <script src="js/main.js"></script>
</body>
</html>
```

3. ท้ายที่สุด ให้ลองเปิดไฟล์ BasicJavaScript.html คลิกปุ่ม button ก็จะได้ข้อความ "Hello JavaScript!!!" ที่เกิดจากไฟล์ main.js แสดงผลในพื้นที่ที่เรากำหนดไว้ ดังรูปที่ 2-7



รูปที่ 2-7 แสดงการทำงานของ JavaScript ในไฟล์ main.js

อธิบายการทำงานของสคริปต์

1. ในไฟล์ main.js สร้างฟังก์ชันที่ชื่อว่า showmessage() ทำหน้าที่แสดงข้อความ "Hello JavaScript!!!" ในพื้นที่ที่ชื่อว่า output

\js\main.js

```
function showmessage() {
    document.getElementById("output").innerHTML = "Hello JavaScript!!!";
}
```

2. ก่อนอีลีเมนต์ปิด </body> ให้อ้างอิงใช้งานไฟล์ main.js ที่อยู่ในโฟลเดอร์ js ก่อน เพื่อให้สามารถใช้งาน JavaScript ในไฟล์ HTML

BasicJavaScript.html

```
    <script src="js/main.js"></script>
</body>
</html>
```

3. ท้ายที่สุด สร้างปุ่ม button ในเหตุการณ์ onclick() สั่งให้ฟังก์ชัน showmessage() ทำงาน และสร้างพื้นที่ที่ชื่อว่า output ขึ้นมาเพื่อแสดงผลที่เกิดจาก JavaScript

BasicJavaScript.html

```
    <div>
        <button type="button" onclick="showmessage()">คลิกที่นี่</button>
        <p id="output"></p>
    </div>
```

การใช้งาน Console ของบราวเซอร์

การแสดงผลการทำงานของ JavaScript อีกวิธีหนึ่งที่ไม่จำเป็นต้องพึ่งพาไฟล์ HTML ก็คือสั่งให้แสดงผลในหน้าต่าง Console ของบราวเซอร์ ถือเป็นอีก 1 วิธีที่นิยมใช้กันอย่างแพร่หลาย มีขั้นตอนดังนี้

1. ที่ไฟล์ BasicJavaScript.html อ้างอิงไฟล์ main.js ที่อยู่ในโฟลเดอร์ js ก่อน ดังสคริปต์ต่อไปนี้

BasicJavaScript.html

```
<body>

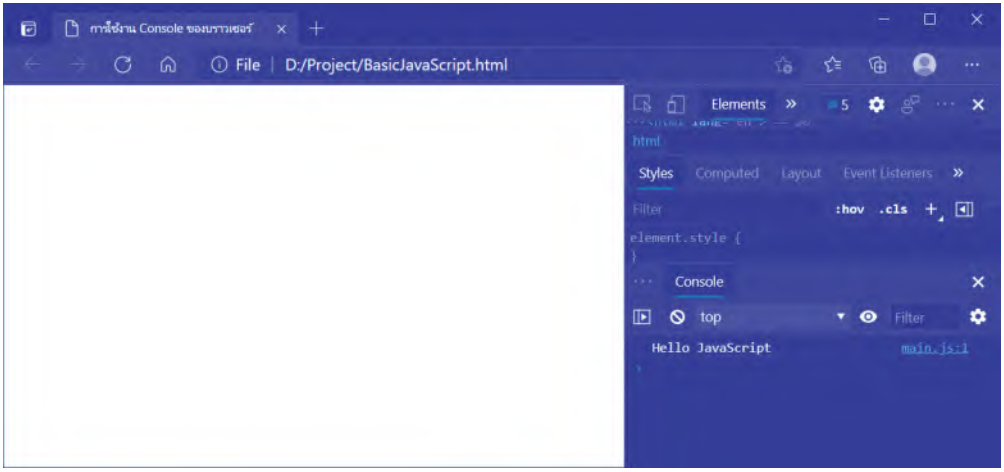
    <script src="js/main.js"></script>
</body>
```

2. ส่วนที่ไฟล์ main.js สั่งให้แสดงข้อความ "Hello JavaScript" ใน Console ของบราวเซอร์

\js\main.js

```
console.log("Hello JavaScript")
```

3. ท้ายที่สุด ให้ดับเบิลคลิกไฟล์ BasicJavaScript.html จากนั้นกดปุ่ม F12 ที่คีย์บอร์ดเพื่อเปิดส่วนของ Console ขึ้นมา ก็จะพบกับข้อความ "Hello JavaScript" ที่เกิดจากการใช้งานฟังก์ชัน log() ดังรูปที่ 2-8



รูปที่ 2-8 ผลการทำงานของ JavaScript ใน Console ของบราวเซอร์

จากรูปที่ 2-8 การทดสอบใช้งานภาษา JavaScript ในด้านต่างๆ ผู้อ่านมีทางเลือกสั่งให้แสดงผลการทำงานได้ 2 แบบคือ

1. สั่งให้แสดงผลการทำงานในบราวเซอร์ โดยอาศัยออบเจ็กต์ document
2. สั่งให้แสดงผลการทำงานในคอนโซล โดยอาศัยออบเจ็กต์ console

ทำความรู้จักกับตัวแปร (Variable) ใน JavaScript (คำสั่ง var และ let)

ตัวแปร (Variable) ทำหน้าที่เก็บข้อมูลที่เราต้องการใช้งานในหน้าเว็บเพจ หรือในการทำงานของเรา ถือเป็นการจัดเก็บข้อมูลแบบชั่วคราว หมายความว่า เมื่อสิ้นสุดการทำงานนั้นๆ ค่าของตัวแปรก็จะหายไป

การสร้างตัวแปรในภาษา JavaScript ขึ้นมาใช้งานในหน้าเว็บเพจของเรามี 2 แบบคือ คำสั่ง **var** กับ **let** เพื่อให้ผู้อ่านเห็นความแตกต่างระหว่างคำสั่งทั้ง 2 ข้างต้น ผู้เขียนขอเริ่มต้นที่การสร้างตัวแปรด้วยคำสั่ง var ก่อน

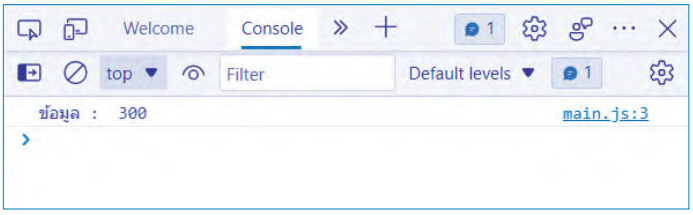
ต้องการสร้างตัวแปรที่ชื่อว่า product ทำหน้าที่เก็บตัวเลขจำนวนเต็ม 300 สั่งให้แสดงผลใน Console ดังรูปที่ 2-9

\js\main.js

```
var product = 300;

console.log("ข้อมูล : ", product);
```

เมื่อรันสคริปต์จะได้ผลลัพธ์ดังรูป



รูปที่ 2-9 แสดงค่าที่อ่านได้จากตัวแปร product

จากรูปที่ 2-9 ที่มาของตัวเลข 300 เกิดจากการทำงานของสคริปต์บรรทัดที่ 3 ที่อยู่ในไฟล์ main.js ถือเป็นการแสดงค่าของตัวแปรโดยพื้นฐานทั่วไป

การสร้างตัวแปร var ส่งผลให้ตัวแปรดังกล่าวมีขอบเขตแบบ **Global** หมายความว่า สามารถเข้ามาแก้ไข, เปลี่ยนแปลงค่า หรือเข้าถึงค่าที่ได้อย่างอิสระ ตัวแปรที่มีขอบเขตแบบ Global สามารถมองได้ทั้งแบบเป็นข้อดีและข้อเสียในเวลาเดียวกัน กล่าวคือ

ตัวแปรที่เราสามารถแก้ไขหรือเข้าถึงค่าได้ ย่อมที่จะสร้างความสะดวกให้กับเรา แต่ในทางกลับกันก็ส่งผลเสียได้เช่นกันคือ ค่าของตัวแปรอาจจะถูกแก้ไข, เปลี่ยนแปลงโดยที่เราไม่ทราบว่าใครเป็นผู้กระทำ มาจากที่ไหน ดังนั้น การใช้งานตัวแปรแบบ Global ต้องใช้ให้น้อยที่สุดเท่าที่จะเป็นไปได้หรือไม่ใช้งานเลย

ตัวแปรแบบ Global ที่เกิดจากคำสั่ง var สร้างปัญหาให้กับการทำงานเป็นอย่างมาก ผู้เขียนขอใช้สคริปต์ต่อไปนี้ประกอบคำอธิบาย

\js\main.js

```
var data = "JS";
var point = 10;

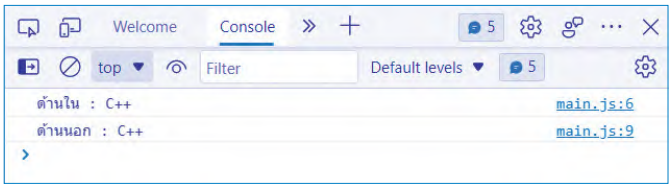
if (point===10){
    var data = "C++";
    console.log("ด้านใน : " + data);
}

console.log("ด้านนอก : " + data)
```

จากสคริปต์ข้างต้น สามารถสรุปได้ 2 ข้อคือ

1. มีการสร้างตัวแปรขึ้นมา 3 ตัว ด้วยคำสั่ง var จำนวน 3 ครั้ง ได้แก่ ตัวแปร data, point และ data ส่งผลให้ตัวแปรทั้ง 3 ตัวมีขอบเขตแบบ Global
2. มีการสร้างตัวแปรที่มีชื่อเหมือนกันคือ data แต่สร้างในพื้นที่แตกต่างกันคือ
   - **ตัวแปร data ตัวที่ 1** สร้างอยู่ด้านนอก กำหนดค่าเริ่มต้นด้วยข้อความ "JS" (var data = "JS")
   - **ตัวแปร data ตัวที่ 2** สร้างอยู่ในบล็อกของคำสั่งตรวจสอบเงื่อนไข if (อยู่ในขอบเขตปีกกา { }) กำหนดข้อความเริ่มต้น "C++" (var data = "C++")

ถ้าเรามองสคริปต์ชุดนี้ด้วยความรู้สึกเพียงอย่างเดียว พบว่าไม่น่าจะมีปัญหาแต่อย่างใด เราแค่สั่งให้แสดงค่าของตัวแปร data ที่มีชื่อเหมือนกันแต่อยู่คนละพื้นที่ ค่าที่ได้น่าจะเป็นข้อความ "JS" และข้อความ "C++" ตามลำดับ



รูปที่ 2-10 แสดงค่าของตัวแปรที่ได้จากคำสั่ง var

จากรูปที่ 2-10 การทำงานของโค้ดชุดนี้เป็นการตรวจสอบว่า ถ้าตัวแปร point มีค่าเท่ากับ 10 ให้ทำงานในบล็อก if พบว่าตัวแปร point มีค่าเท่ากับ 10 อยู่แล้ว จึงมาทำในบล็อกของ if

ปัญหาก็คือ ค่าที่เราอ่านได้จากตัวแปร data "ทั้ง 2 ตัว" กลับกลายเป็นค่าเดียวกันคือ ข้อความ "C++" ซึ่งเป็นค่าล่าสุด ทั้งๆ ที่เราสร้างตัวแปรขึ้นมา 2 ตัวแต่มีชื่อเหมือนกัน ด้วยคำสั่ง var จำนวน 2 ครั้ง

Note

การตรวจสอบว่า "เท่ากัน" ในภาษา JavaScript ให้ผู้อ่านใช้ตัวดำเนินการ === มีความหมาย 2 อย่างในเวลาเดียวกัน กล่าวคือ

- **ความหมายที่ 1** ตรวจสอบว่าเท่ากัน ใช่หรือไม่?
- **ความหมายที่ 2** ตรวจสอบในฐานะตัวเลขกับตัวเลข หรือตรวจสอบชนิดข้อมูลด้วย

ตัวแปรที่เกิดจากคำสั่ง var มีขอบเขตแบบ Global ตั้งแต่เกิด ภาษา JavaScript มองตัวแปร data ทั้ง 2 ตัวเป็นตัวแปรตัวเดียวกันทั้งๆ ที่เราสร้างขึ้นมา 2 ครั้ง นี่คือปัญหาใหญ่ของการสร้างตัวแปรด้วยคำสั่ง var

เพื่อให้การควบคุมค่าของตัวแปรถูกต้อง ผู้เขียนเปลี่ยนวิธีสร้างตัวแปรจากคำสั่งเดิม var เป็นคำสั่ง let ส่งผลให้ตัวแปรที่ได้มามีขอบเขตแบบ Local หรือมีขอบเขตตามพื้นที่ที่ตัวเองเกิดขึ้นมา ดังสคริปต์ต่อไปนี้

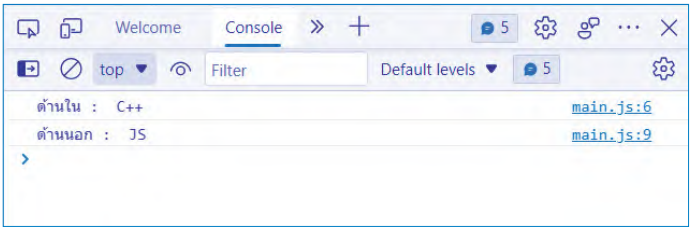
\js\main.js

```
let data = "JS";
let point = 10;

if (point===10){
    let data = "C++";
    console.log("ด้านใน : ", data);
}

console.log("ด้านนอก : ", data)
```

เมื่อรันสคริปต์จะได้ผลลัพธ์ดังรูป



รูปที่ 2-11 กรณีตัวแปรสร้างด้วยคำสั่ง let

จากรูปที่ 2-11 ค่าของตัวแปร data ที่ถูกสร้างจากคำสั่ง let แสดงค่าได้ถูกต้องตามพื้นที่ที่ตัวเองเกิดขึ้นมา กล่าวคือ

- **ตัวแปร data ด้านในบล็อกคำสั่ง if** มีขอบเขตในระยะบล็อกของปีกกา { } เท่านั้น เก็บค่า "C++"
- **ตัวแปร data ด้านนอก** เก็บข้อความ "JS" ตั้งแต่เกิด ไม่มีใครมายุ่งเกี่ยวกับค่านี้

จากสคริปต์ชุดนี้แสดงให้เห็นว่า เราควรสร้างตัวแปรด้วยคำสั่ง let ดีกว่าคำสั่ง var เพื่อควบคุมขอบเขตการเข้าถึงตัวแปรให้ถูกต้องนั่นเอง

Note

การใช้งาน JavaScript สมัยใหม่ ไม่แนะนำให้สร้างตัวแปรด้วยคำสั่ง var อีกต่อไปแล้ว เพราะเกิดจากขอบเขตที่กว้างมากจนเกินไปนั่นเอง

ค่าคงที่ (คำสั่ง const)

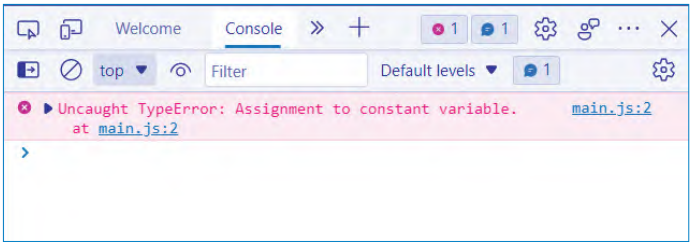
ค่าคงที่ (Constant) ทำหน้าที่เช่นเดียวกับตัวแปร กล่าวคือ ทำหน้าที่เก็บข้อมูลชั่วคราว มีข้อแตกต่างที่สำคัญคือ แก้ไขค่าในค่าคงที่ไม่ได้ เช่น สร้างค่าคงที่ที่ชื่อว่า price ขึ้นมา ทำหน้าที่เก็บตัวเลขจำนวนเต็ม 500 จากนั้นให้ผู้อ่านทดสอบแก้ไขค่าใหม่เป็นตัวเลข 1000 ส่งผลให้เกิดข้อผิดพลาด ดังรูปที่ 2-12

\js\main.js

```
const price = 500;
price = 1000;

console.log("ราคา : ", price);
```

เมื่อรันสคริปต์จะได้ผลลัพธ์ดังรูป



รูปที่ 2-12 แสดงข้อผิดพลาดที่เกิดจากการแก้ไขค่าคงที่

จากรูปที่ 2-12 ถ้ากล่าวตามความรู้สึกของเรา ความหมายของค่าคงที่ชัดเจนในตัวเองอยู่แล้วว่า เราแก้ไขค่าไม่ได้ แต่ใน JavaScript สามารถแก้ไขได้ ถ้าค่าคงที่นั้นๆ เป็นออบเจ็กต์ เช่น

สร้างค่าคงที่ที่ชื่อว่า data ภายในประกอบด้วยข้อมูล 2 ส่วนคือ

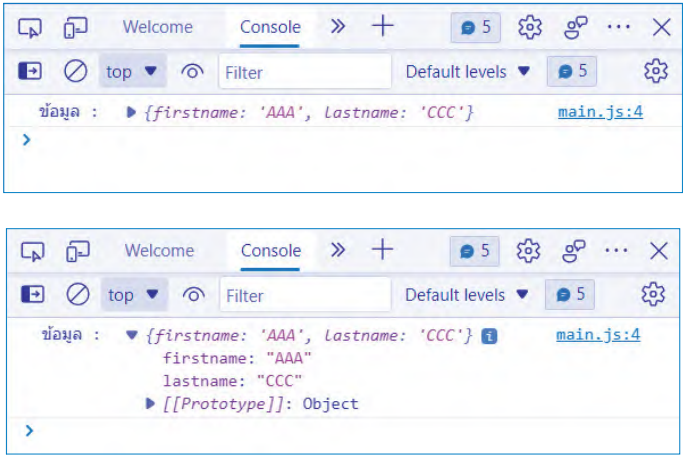
- **ส่วน firstname** ทำหน้าที่เก็บชื่อ "AAA"
- **ส่วน lastname** ทำหน้าที่เก็บนามสกุล "BBB"

จากนั้นเปลี่ยนนามสกุลจากค่าเดิม "BBB" เป็นค่าใหม่ "CCC" พบว่าสามารถแก้ไขค่าคงที่ data ได้ ดังรูปที่ 2-13

\js\main.js

```
const data = { firstname:"AAA", lastname:"BBB" };
data.lastname = "CCC";

console.log("ข้อมูล : ", data);
```



รูปที่ 2-13 แสดงการแก้ไขค่าคงที่ในออบเจ็กต์ data