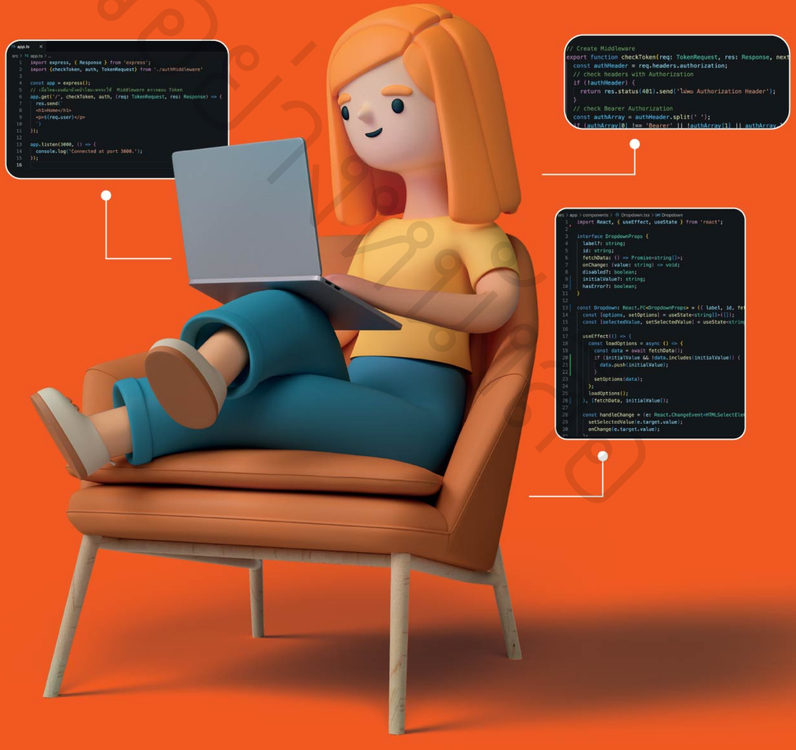
Think Beyond Develop

TypeScript +Node.js

สำหรับ
Full Stack Developer



- พื้นฐาน TypeScript ครบทุกแง่มุม สำหรับนักเรียน นักศึกษา และโปรแกรมเมอร์
- การใช้งาน TypeScript ขั้นสูง เช่น Generics, Decorators และ Metadata
- สร้างเว็บแอปพลิเคชันในฝั่ง Backend ด้วย Node.js และ Express
- พัฒนาทักษะและก้าวสู่การเป็น Full Stack Developer ในแบบมืออาชีพ
- วิธีใช้งานแพ็คเกจยอดนิยม เช่น axios, cookie-session, lodash, class-validator และอื่น ๆ

พิเศษ !
ดาวน์โหลดโค้ดตัวอย่าง
เพื่อใช้ประกอบการเรียน

จัดจำหน่ายโดย

จีราวุธ วารินทร์

สารบัญ

Part 1 TypeScript Basic

Part 2 TypeScript Beyond Basic

Part 3 Node.js with TypeScript

Part 01

TypeScript Basic

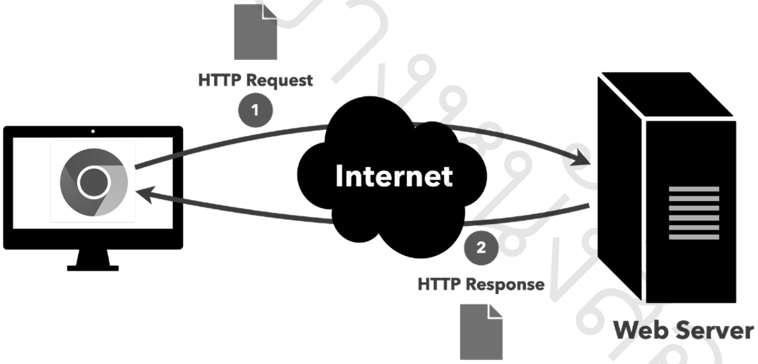
01

แนะนำ TypeScript

TypeScript (ไทป์สคริปต์) เป็นภาษาคอมพิวเตอร์ที่ถูกพัฒนาโดย Microsoft เปิดตัวครั้งแรกในเดือนตุลาคม 2012 เป็นภาษาคอมพิวเตอร์ที่สร้างขึ้นจาก JavaScript โดยได้เพิ่มคุณสมบัติต่าง ๆ ช่วยให้นักพัฒนาสามารถเขียนโค้ด JavaScript ที่มีโครงสร้างที่ชัดเจนและมีความปลอดภัยมากขึ้น เช่น มีระบบตรวจสอบชนิดข้อมูล (Type Checking) และสนับสนุนการใช้งานของ Object-Oriented Programming (OOP) เต็มรูปแบบ เป็นต้น

เว็บเพจและ JavaScript

เว็บเพจ (Webpage หรือ Web page) คือ เอกสารที่ถูกใช้งานบนอินเทอร์เน็ต และถูกแสดงผลบนเบราว์เซอร์ เมื่อผู้ใช้ต้องการเปิดดูข้อมูลจากอินเทอร์เน็ต จะใช้เว็บเบราว์เซอร์ไปยังที่อยู่ของเว็บเพจ (ใช้ IP Address) เพื่อส่งการร้องขอข้อมูลไปยังเว็บเซิร์ฟเวอร์ (เรียกว่า HTTP Request) เมื่อเว็บเซิร์ฟเวอร์ได้รับการร้องขอ ก็จะส่งผลลัพธ์กลับมายังเบราว์เซอร์ (เรียกว่า HTTP Response) เพื่อนำมาแสดงผลต่อไป



▲ รูปแสดง HTTP Request และ HTTP Response

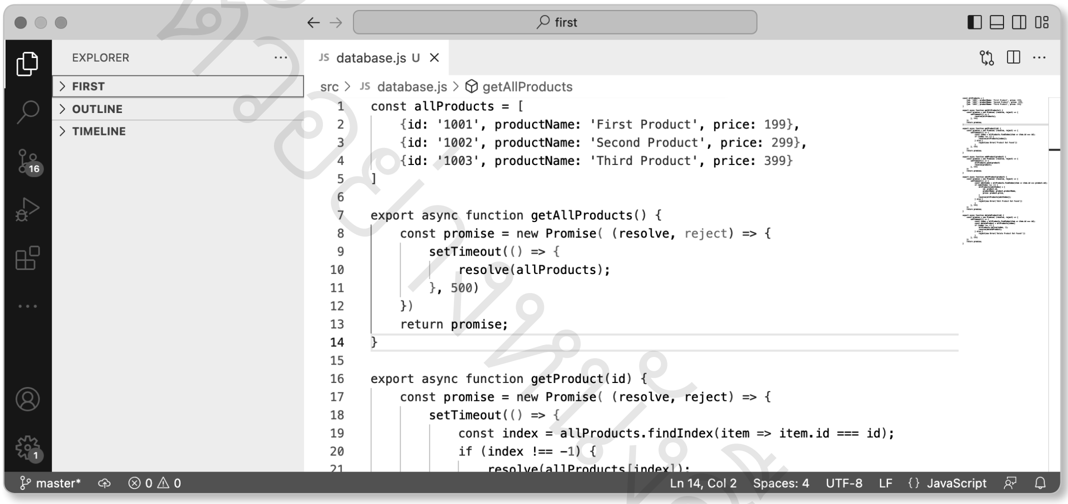
เนื้อหาที่อยู่ในเว็บเพจอาจประกอบด้วยข้อความ ลิงก์ รูปภาพ หรือมีเดียประเภทต่าง ๆ ดังนั้น ข้อมูลเว็บเพจที่ถูกนำมาแสดงผลบนเบราว์เซอร์จึงประกอบไปด้วยไฟล์ประเภทต่าง ๆ มากมาย แต่โดยพื้นฐานแล้ว เว็บเพจจะประกอบด้วยไฟล์หลักอยู่ 3 ประเภท ได้แก่ HTML, CSS และ JavaScript

- **HTML** คือ ไฟล์ที่ใช้กำหนดโครงสร้างของเว็บเพจ บอกให้ทราบว่าภายในเว็บเพจจะมีเนื้อหา และมีจัดเรียงลำดับอย่างไร ไฟล์ HTML จะมีนามสกุลไฟล์เป็น .htm หรือ .html
- **CSS** คือ ไฟล์ที่ใช้ตกแต่งหน้าตาเว็บเพจ เช่น กำหนดขนาดและสีตัวอักษร, กำหนดขนาดรูปภาพ, จัดตำแหน่งการแสดงผลบนอุปกรณ์ที่มีขนาดหน้าจอต่างกัน ฯลฯ ไฟล์ CSS จะมีนามสกุลไฟล์เป็น .css
- **JavaScript** คือ ไฟล์ซึ่งทำให้เว็บเพจสามารถโต้ตอบกับผู้ใช้ได้ หรือทำให้เว็บเพจมีความสามารถบางอย่างเพิ่มขึ้นมา ซึ่งไฟล์ JavaScript จะมีนามสกุลไฟล์เป็น .js

การเขียนโค้ด JavaScript จะมีความสัมพันธ์โดยตรงกับโค้ด HTML และ CSS ที่อยู่ในเว็บเพจ เช่น เขียนโค้ด JavaScript เพื่อแสดงหรือซ่อนเนื้อหาที่อยู่ใน HTML หรือเขียนโค้ด JavaScript เพื่อเลือกใช้งาน CSS สำหรับปรับแต่งหน้าตาเว็บเพจในแบบที่ต้องการ

จุดประสงค์หลักของ JavaScript คือ การทำให้เว็บเพจสามารถโต้ตอบกับผู้ใช้ได้ หรือมีความสามารถบางอย่างเพิ่มขึ้น เช่น ใช้ JavaScript เพื่อขยายรูปเมื่อผู้ใช้เลื่อนเมาส์ไปที่รูปภาพ, ใช้ JavaScript เพื่อสร้างแอนิเมชันเลื่อนข้อความจากซ้ายไปขวา, ใช้ JavaScript เพื่อนำเนื้อหาใหม่มาแทนที่เนื้อหาเดิมโดยไม่ต้องโหลดเว็บเพจใหม่ เป็นต้น

JavaScript เป็นภาษาคอมพิวเตอร์ที่ออกแบบมาสำหรับใช้กับเว็บเพจ ดังนั้น โค้ด JavaScript จึงถูกรันเพื่อใช้งานบนเบราว์เซอร์เป็นหลัก นอกจากนั้น JavaScript ยังสามารถใช้กับสภาพแวดล้อมอื่น ๆ ที่ไม่ใช่เบราว์เซอร์ได้อีกด้วย เช่น Node.js และ Apache CouchDB เป็นต้น



```
const allProducts = [
    {id: '1001', productName: 'First Product', price: 199},
    {id: '1002', productName: 'Second Product', price: 299},
    {id: '1003', productName: 'Third Product', price: 399}
]

export async function getAllProducts() {
    const promise = new Promise( (resolve, reject) => {
        setTimeout(() => {
            resolve(allProducts);
        }, 500)
    })
    return promise;
}

export async function getProduct(id) {
    const promise = new Promise( (resolve, reject) => {
        setTimeout(() => {
            const index = allProducts.findIndex(item => item.id === id);
            if (index !== -1) {
                resolve(allProducts[index]);
```

▲ รูปแสดงตัวอย่างโค้ดคำสั่ง JavaScript

จาก JavaScript สู่ TypeScript

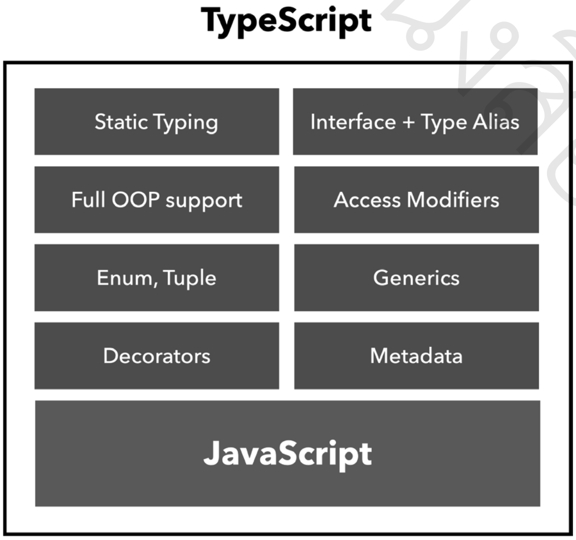
แม้ว่า JavaScript จะเป็นภาษาหลักสำหรับใช้จัดการกับเว็บเพจบนเบราว์เซอร์ แต่การใช้ JavaScript กับแอปพลิเคชันขนาดใหญ่ หรือกับแอปพลิเคชันที่ต้องพัฒนาเป็นทีมนั้นค่อนข้างยุ่งยาก เนื่องจาก JavaScript ไม่มีระบบตรวจสอบชนิดข้อมูล (Type Checking) เมื่อโค้ดมีปัญหามักตรวจสอบข้อผิดพลาดได้ยาก โดยเฉพาะการนำโค้ดจากนักพัฒนาอื่นหรือโค้ดจากภายนอกมาแก้ไขต่อ ก็ยิ่งปวดหัวมากไปกว่าเดิม

เพื่อแก้ปัญหาข้างต้น ภาษาคอมพิวเตอร์ที่ชื่อว่า TypeScript จึงถูกสร้างขึ้น โดยนำ JavaScript มาเพิ่มคุณสมบัติใหม่และทำให้มีประสิทธิภาพยิ่งขึ้น เช่น ข้อมูลจะต้องมีชนิดข้อมูลที่แน่นอน (Type Safety) สามารถตรวจสอบชนิดข้อมูลโดยอัตโนมัติได้ (Type Checking) และรองรับ OOP (Object Oriented Programming) เต็มรูปแบบ เป็นต้น

ต่อไปนี้เป็นตารางเปรียบเทียบความแตกต่างระหว่าง TypeScript และ JavaScript

	TypeScript	JavaScript
กำหนดชนิดข้อมูลที่แน่นอน (Type Safety)	มี	ไม่มี
ระบบตรวจสอบชนิดข้อมูล (Type Checking)	มี	ไม่มี
การใช้งาน OOP	สนับสนุนเต็มรูปแบบ	มีใช้งาน แต่ไม่ได้สนับสนุนเต็มรูปแบบ
สนับสนุน Refactoring	สนับสนุน และปลอดภัย	สนับสนุน แต่ไม่มีระบบตรวจสอบความปลอดภัย
การใช้งาน JavaScript ไลบรารี	จะต้องมี Type Definition Files เพื่อกำหนดชนิดข้อมูลที่อยู่ในไลบรารีนั้น ๆ	รองรับ
นามสกุลไฟล์	.ts	.js

จากตารางข้างต้น สามารถสรุปได้ว่า TypeScript คือ การนำ JavaScript มาเพิ่มระบบตรวจสอบชนิดข้อมูล (Type System) รวมถึงคุณสมบัติอื่น ๆ ที่ช่วยให้การเขียน JavaScript มีประสิทธิภาพและปลอดภัยยิ่งขึ้น กล่าวได้ว่า TypeScript ก็คือ Super Set ของ JavaScript นั่นเอง

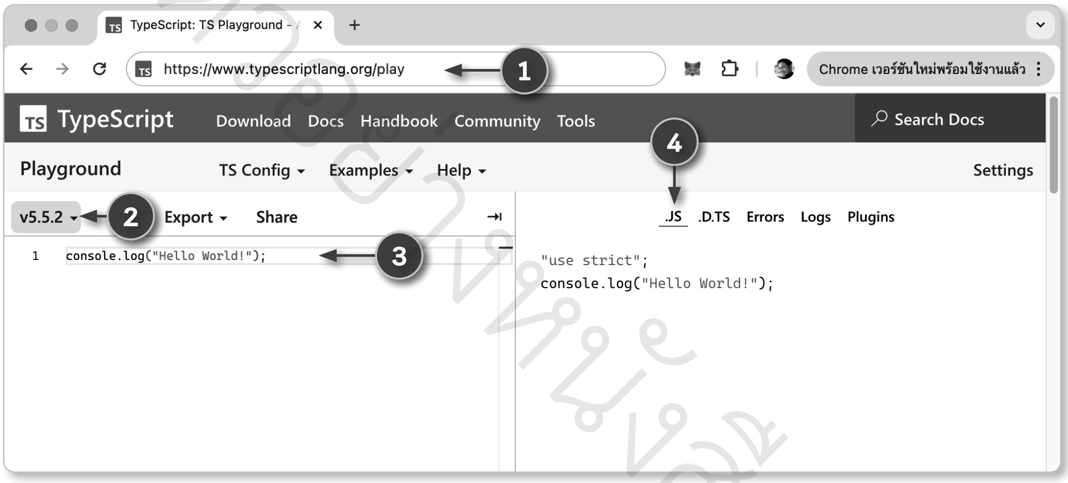


▲ รูปแสดงตัวอย่างคุณสมบัติ TypeScript ที่มากกว่า JavaScript

ทดสอบ TypeScript ผ่านเว็บไซต์

การศึกษา TypeScript สามารถทำได้หลายช่องทาง วิธีหนึ่งที่นิยมเพิ่มขึ้นมากในปัจจุบัน คือ การศึกษา TypeScript ผ่านทาง Playground ซึ่งอยู่ที่เว็บเพจ https://typescriptlang.org/play ผู้อ่านสามารถกรอกชุดคำสั่ง TypeScript ลงในเบราว์เซอร์ ผลลัพธ์การรันชุดคำสั่งก็จะปรากฏบนเบราว์เซอร์ ดังตัวอย่าง

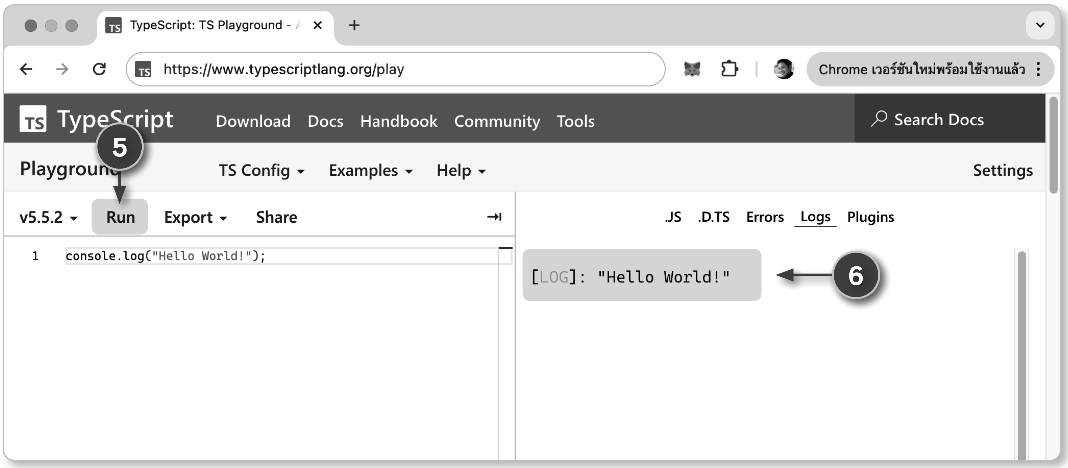
1. เปิดเบราว์เซอร์และไปยังแอดเดรส https://typescriptlang.org/play
2. เลือกเวอร์ชันของ TypeScript แนะนำให้เลือกเป็นเวอร์ชันล่าสุด
3. กรอกโค้ดคำสั่ง TypeScript โดยในตัวอย่างเป็นการพิมพ์ข้อความ Hello World! ออกมาที่หน้าต่างคอนโซล
4. คลิกเมนู .JS เพื่อตรวจสอบผลลัพธ์ที่เป็น JavaScript โดยโค้ด TypeScript ที่อยู่ในหน้าต่างด้านซ้ายจะถูกคอมไพลเลอร์แปลงเป็น JavaScript



```
console.log("Hello World!");
```

```
"use strict";
console.log("Hello World!");
```

5. คลิกเมนู Run เพื่อรันโค้ด
6. สังเกตผลลัพธ์ข้อความ “Hello World!” ที่ได้ในหน้าต่างด้านขวามือ



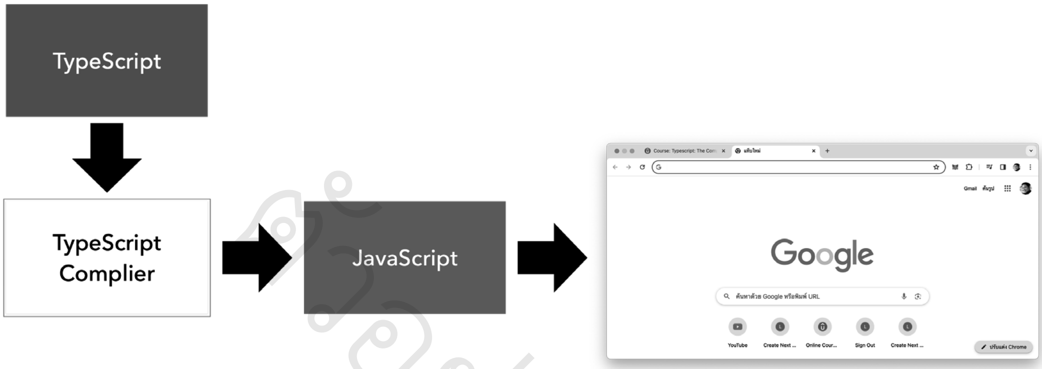
```
console.log("Hello World!");
```

```
[LOG]: "Hello World!"
```

เบราว์เซอร์ไม่รู้จัก TypeScript

เนื่องจากเบราว์เซอร์รู้จัก JavaScript แต่ไม่รู้จัก TypeScript ดังนั้น คำสั่ง TypeScript จึงไม่สามารถใช้งานกับเบราว์เซอร์ได้โดยตรง

เพื่อให้เบราว์เซอร์เข้าใจคำสั่งต่าง ๆ ที่เขียนด้วย TypeScript เราจะต้องติดตั้ง TypeScript คอมไพเลอร์ลงในเครื่องคอมพิวเตอร์ เพื่อทำหน้าที่แปลงโค้ด TypeScript ให้กลายเป็น JavaScript เสียก่อน



▲ รูปแสดงการคอมไพล์ TypeScript ให้เป็น JavaScript

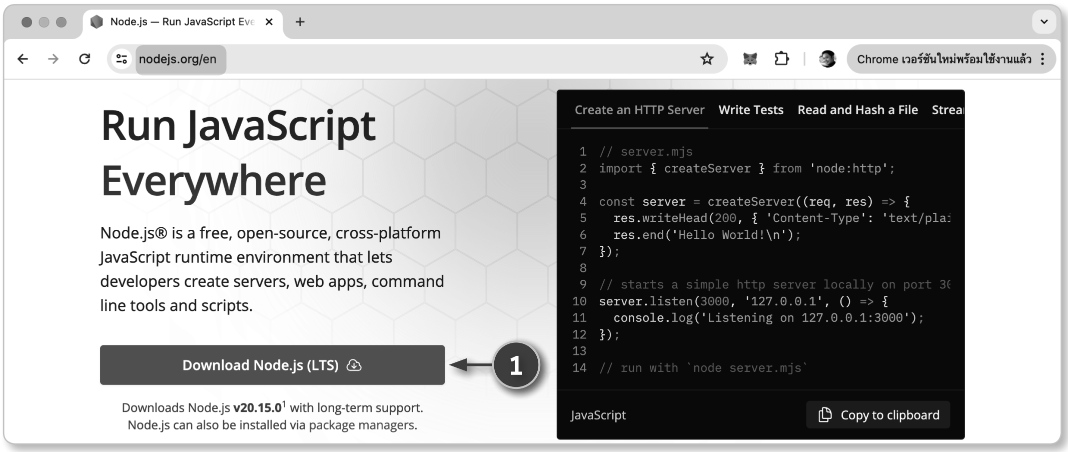
ติดตั้ง TypeScript

การติดตั้ง TypeScript ลงในคอมพิวเตอร์สามารถแบ่งได้เป็น 3 ขั้นตอน ดังนี้

ขั้นตอนที่ 1 ดาวน์โหลดและติดตั้ง Node.js

Node.js เป็นสภาพแวดล้อมที่ช่วยให้นักพัฒนาสามารถใช้ JavaScript พัฒนาแอปพลิเคชันแยกเป็นอิสระจากเบราว์เซอร์ได้ (JavaScript Runtime Environment) โดยหลังจากติดตั้ง Node.js เรียบร้อย จะได้ npm ซึ่งเป็นเครื่องมือสำหรับจัดการกับแพ็คเกจมาใช้งาน

1. ไปยังเว็บไซต์ https://nodejs.org จากนั้นคลิกปุ่มดาวน์โหลด Node.js



Run JavaScript Everywhere

Node.js® is a free, open-source, cross-platform JavaScript runtime environment that lets developers create servers, web apps, command line tools and scripts.

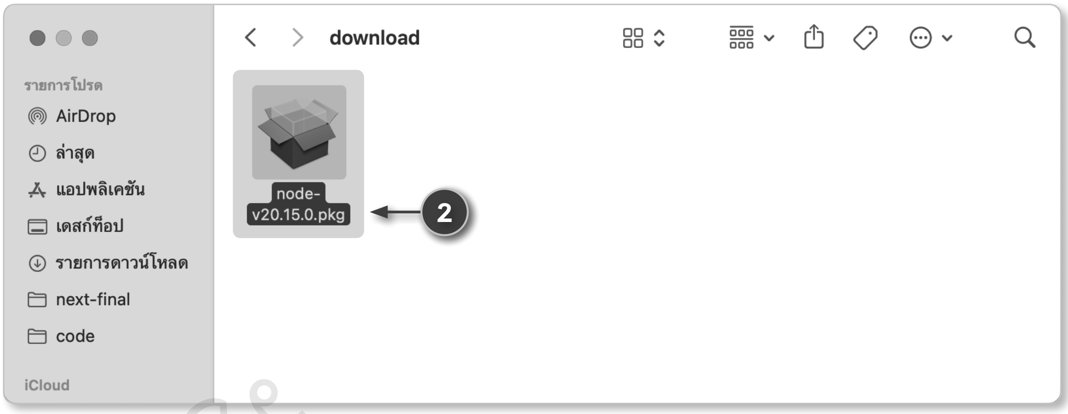
Download Node.js (LTS)

Downloads Node.js **v20.15.0**^{1} with long-term support.
Node.js can also be installed via package managers.

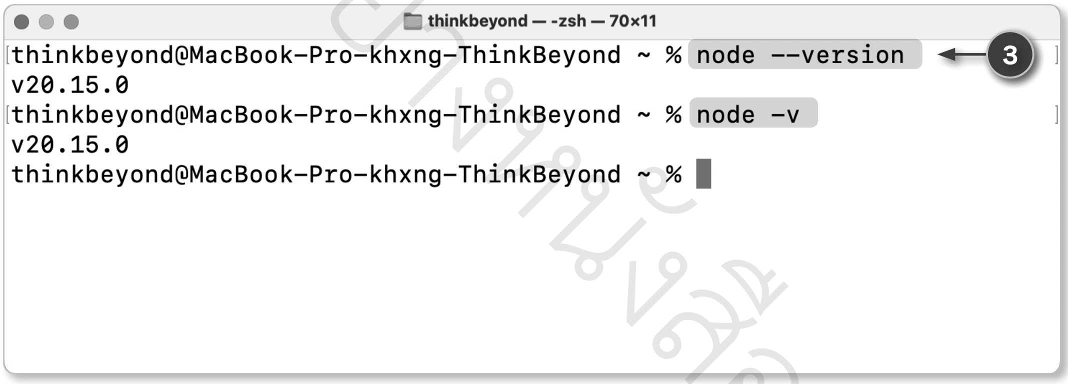
```
// server.mjs
import { createServer } from 'node:http';

const server = createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'text/pla
  res.end('Hello World!\n');
});

// starts a simple http server locally on port 3
server.listen(3000, '127.0.0.1', () => {
  console.log('Listening on 127.0.0.1:3000');
});

// run with `node server.mjs`
```

JavaScript   Copy to clipboard

2. ดับเบิลคลิกไฟล์ เพื่อติดตั้ง Node.js ไปยังคอมพิวเตอร์



3. หลังจากติดตั้งเรียบร้อย ให้ทดสอบว่าการติดตั้งนั้นถูกต้องหรือไม่ โดยเปิดหน้าต่าง Terminal (หรือหน้าต่าง Command Prompt) จากนั้นพิมพ์คำสั่ง node --version หรือ node -v เพื่อดูหมายเลขเวอร์ชัน หากพบหมายเลขเวอร์ชัน การติดตั้ง Node.js ถือว่าสำเร็จเรียบร้อย

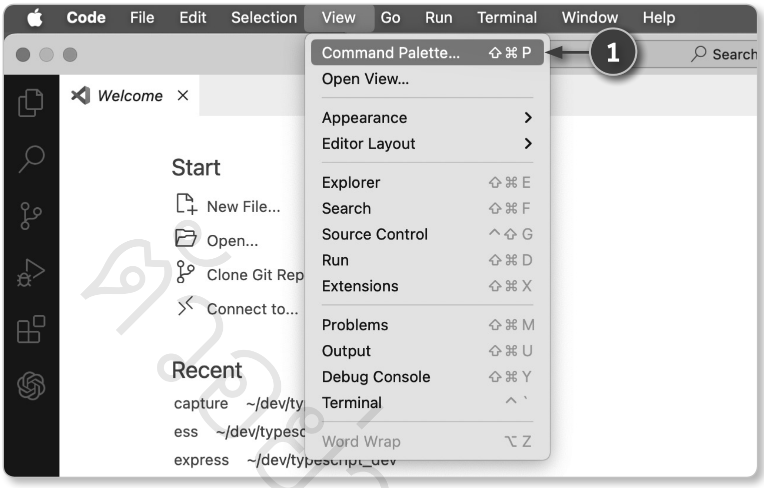


```
thinkbeyond@MacBook-Pro-khxng-ThinkBeyond ~ % node --version
v20.15.0
thinkbeyond@MacBook-Pro-khxng-ThinkBeyond ~ % node -v
v20.15.0
thinkbeyond@MacBook-Pro-khxng-ThinkBeyond ~ % 
```

กำหนด code path ให้กับ VS Code

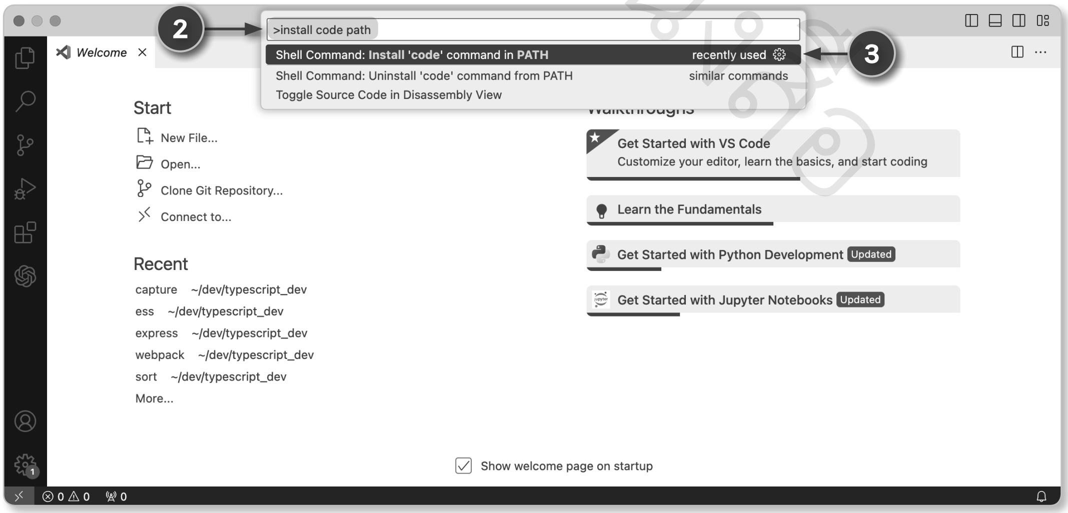
เพื่อให้สามารถใช้คำสั่ง code . เป็นทางลัดในการเปิด VS Code จากหน้าต่าง Terminal จะต้องกำหนดพาธด้วยขั้นตอน ดังนี้

1. เปิดโปรแกรม VS Code แล้วคลิกเมนู View > Command Palette



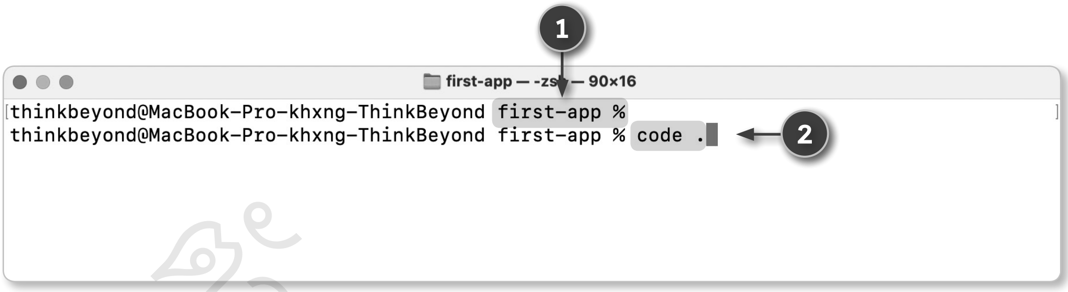
2. ค้นหาคำสั่งที่เกี่ยวกับการติดตั้งพาธ เช่น พิมพ์ install code path

3. จะปรากฏคำสั่งที่เกี่ยวข้อง ให้คลิกที่ Shell Command: Install ‘code’ command in PATH (ในระบบปฏิบัติการแมคโอเอส จะต้องยืนยันว่าเป็นผู้ดูแลระบบด้วยการกรอกรหัสผ่านอีกครั้ง)



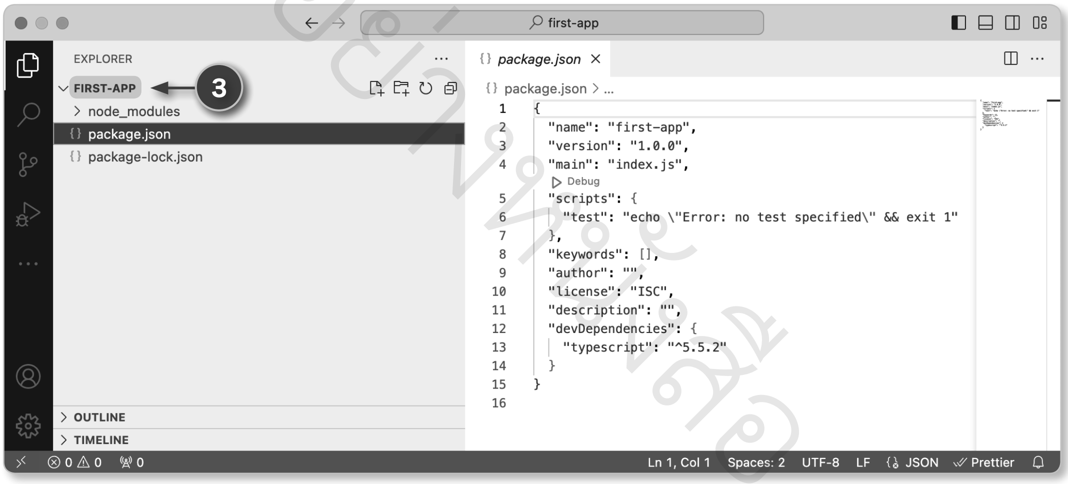
หลังจากทำตามขั้นตอนข้างต้นแล้ว เราสามารถเปิดโฟลเดอร์ หรือโปรเจ็กต์ โดยใช้คำสั่ง code . ได้ ดังตัวอย่าง

1. เปิดหน้าต่าง Terminal (ในแมคโอเอส) หรือหน้าต่าง Command Line (ในวินโดวส์) เข้าไปยังโฟลเดอร์ที่ต้องการ เช่น พิมพ์คำสั่ง cd first-app เพื่อเข้าไปยังโฟลเดอร์ first-app
2. พิมพ์คำสั่ง code . และกดปุ่ม <Enter> เพื่อสั่งให้เปิดโฟลเดอร์ปัจจุบันด้วยโปรแกรม Visual Studio Code (ในที่นี้คือโฟลเดอร์ first-app)



```
thinkbeyond@MacBook-Pro-khxng-ThinkBeyond first-app %
thinkbeyond@MacBook-Pro-khxng-ThinkBeyond first-app % code .
```

3. ผลลัพธ์ VS Code ถูกเรียกใช้งาน และเปิดโฟลเดอร์ first-app โดยอัตโนมัติ



```
{
  "name": "first-app",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "devDependencies": {
    "typescript": "^5.5.2"
  }
}
```

คอมไพล์จาก TypeScript เป็น JavaScript

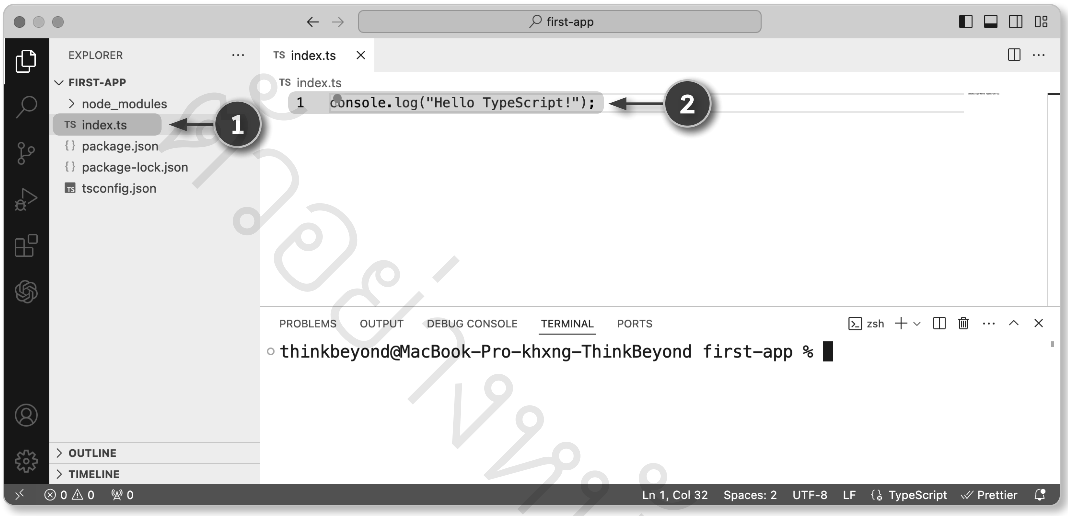
หลังจากติดตั้ง TypeScript ลงในโปรเจ็กต์ เราสามารถใช้คำสั่ง tsc เพื่อคอมไพล์โค้ดในไฟล์ TypeScript ให้กลายเป็น JavaScript

1. เปิดโปรเจ็กต์ด้วย Visual Studio Code สร้างไฟล์ index.ts ไว้ในโปรเจ็กต์ (ไฟล์นามสกุล .ts หมายถึง TypeScript)
2. กรอกโค้ด TypeScript ลงไปในไฟล์ index.ts เช่น console.log(“Hello World!”);

นำโค้ด TypeScript ไปใช้กับเว็บเพจ

เนื่องจากเบราว์เซอร์รู้จัก JavaScript แต่ไม่รู้จัก TypeScript ดังนั้น หากต้องการใช้ TypeScript เพื่อควบคุมการทำงานบนเว็บเพจ เราจะต้องคอมไพล์จาก TypeScript ให้กลายเป็น JavaScript ก่อน แล้วนำ JavaScript ที่ได้ไปกำหนดยังแท็ก <script></script> ของเว็บเพจ ดังตัวอย่าง

1. สร้างไฟล์ index.ts สำหรับเก็บโค้ด TypeScript โดยหลังจากใช้คำสั่ง tsc เราก็จะได้ index.js ที่เป็น JavaScript สำหรับใช้กับเบราว์เซอร์
2. แก้ไขโค้ด index.ts โดยกรอกโค้ด TypeScript ลงไป เช่น console.log(“Hello TypeScript!”);

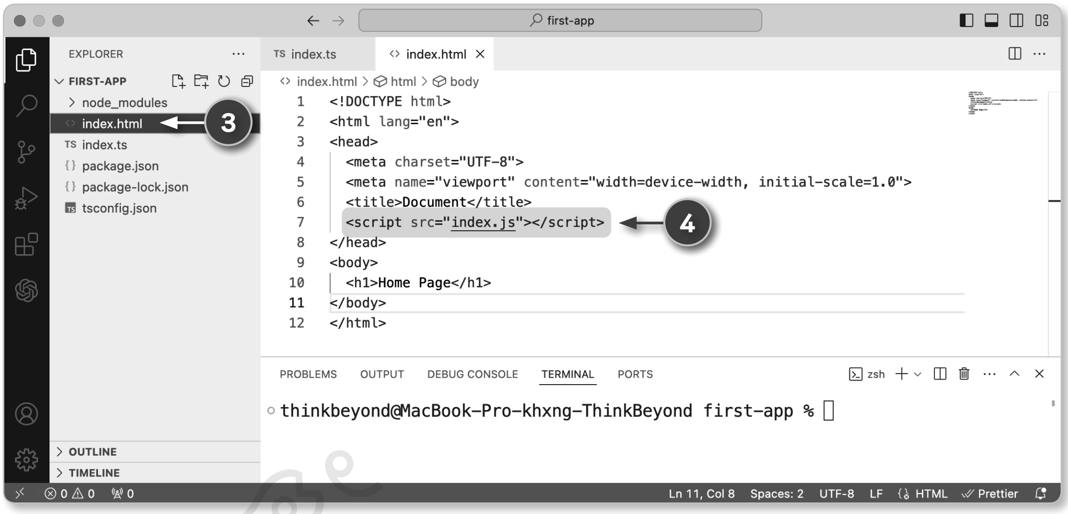


```
console.log("Hello TypeScript!");
```

```
thinkbeyond@MacBook-Pro-khxng-ThinkBeyond first-app %
```

3. สร้างไฟล์ index.html เป็นหน้าเว็บเพจที่ต้องการรันโค้ด JavaScript
4. แก้ไข index.html โดยกำหนดไฟล์ JavaScript (index.js) ไปยังแท็ก <script></script>

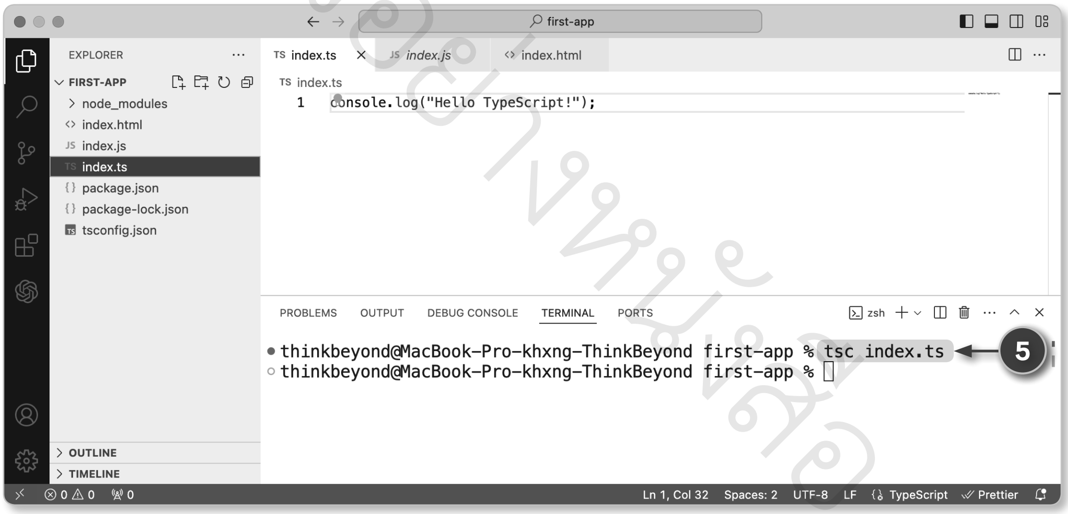
```
<!DOCTYPE html>
<html lang=”en”>
<head>
  <meta charset=”UTF-8”>
  <meta name=”viewport” content=”width=device-width, initial-scale=1.0”>
  <title>Document</title>
  <script src=”index.js”></script>
</head>
<body>
  <h1>Home Page</h1>
</body>
</html>
```



```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
  <script src="index.js"></script>
</head>
<body>
  <h1>Home Page</h1>
</body>
</html>
```

```
thinkbeyond@MacBook-Pro-khxng-ThinkBeyond first-app %
```

5. ใช้คำสั่ง tsc index.ts เพื่อคอมไพล์ เราจะได้ไฟล์ index.js ที่เป็น JavaScript มาใช้งาน



```typescript
console.log("Hello TypeScript!");
```

```
thinkbeyond@MacBook-Pro-khxng-ThinkBeyond first-app % tsc index.ts
thinkbeyond@MacBook-Pro-khxng-ThinkBeyond first-app %
```

6. เปิดไฟล์ index.html บนเบราว์เซอร์ เช่น เปิดใน Google Chrome

7. เปิดหน้าต่างคอนโซลของ Google Chrome โดยกดปุ่ม <Ctrl+Shift+J> (หรือ กดปุ่ม <Option+Cmd+J> ในแมคโอเอส) ก็จะพบผลลัพธ์ Hello TypeScript! ที่หน้าต่างคอนโซล

02

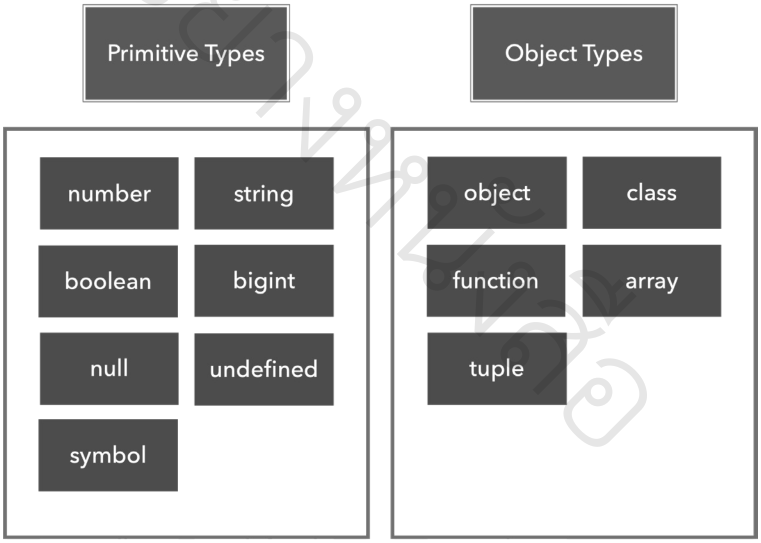
ตัวแปรและชนิดข้อมูล

TypeScript คือ การนำ JavaScript มาเพิ่มคุณสมบัติต่าง ๆ เพื่อให้สามารถนำไปใช้งานได้ปลอดภัย เช่น ให้ความสำคัญต่อการใช้ชนิดข้อมูล (Types) มีการปรับปรุงเกี่ยวกับการเขียนข้อมูลเชิงวัตถุ (OOP) ฯลฯ กล่าวได้ว่า TypeScript ก็คือ Super Set ที่ครอบ JavaScript เอาไว้นั่นเอง

ในบทนี้จะเป็นพื้นฐานเกี่ยวกับการประกาศตัวแปรและวิธีใช้งานชนิดข้อมูลพื้นฐานของ TypeScript เช่น ตัวเลข (number), ข้อความ (string), ค่าความจริงทางตรรกะ (boolean), any, null และ undefined

ชนิดข้อมูลใน TypeScript

TypeScript ได้แบ่งชนิดข้อมูลได้เป็น 2 กลุ่ม ได้แก่ Primitive Types (ชนิดข้อมูลพื้นฐาน) และ Object Types (ชนิดข้อมูลออบเจ็กต์)



- Primitive Types คือ ชนิดข้อมูลแบบพื้นฐาน ได้แก่ number, string, boolean, symbol, bigint, null และ undefined เป็นต้น
- Object Types คือ ชนิดข้อมูลในแบบออบเจ็กต์ ได้แก่ object, array, class, function และ tuple เป็นต้น

ชนิดข้อมูลพื้นฐาน (Primitive)

ชนิดข้อมูล (Data Type หรือ Type) ใช้เพื่อบอกให้ทราบว่า ข้อมูลที่ใช้งานอยู่นั้นมีคุณสมบัติอย่างไร และสามารถทำอะไรได้บ้าง เช่น ถ้าเป็นข้อมูลชนิดตัวเลขก็จะสามารถนำมาคำนวณทางคณิตศาสตร์ บวก ลบ คูณ หารได้ แต่ถ้าเป็นข้อความก็จะนำมาคำนวณไม่ได้ เป็นต้น

ชนิดข้อมูลที่ใช้เป็นพื้นฐาน (Primitive) ของ TypeScript มีอยู่ด้วยกันหลายชนิด ดังนี้

- string คือ ชนิดข้อมูลที่ใช้เก็บข้อความ (ชุดของตัวอักษรที่เรียงต่อกันเป็นลำดับ)
- number คือ ชนิดของข้อมูลที่ใช้เก็บเลข ได้แก่ เลขจำนวนเต็ม เลขทศนิยม ค่าบวก และค่าติดลบ เช่น -3.4, -1, 0, 1, 1.0, 5.98
- boolean คือ ชนิดข้อมูลที่ใช้เก็บค่าความจริงทางตรรกะ ซึ่งเก็บค่าเพียง 2 ค่า ได้แก่ true หรือ false
- undefined คือ ชนิดข้อมูลที่บอกให้ทราบว่า ยังไม่ได้กำหนดค่า หรือค่าที่กำหนดนั้นไม่มีอยู่ เช่น การประกาศตัวแปรแต่ยังไม่ได้กำหนดค่าเริ่มต้น ค่าที่กำหนดให้กับตัวแปรจะมีค่าเป็น undefined เสมอ ส่วนการใช้อินเด็กซ์เพื่ออ้างอิงไปยังตำแหน่งของสตริงที่ไม่ถูกต้อง ก็จะได้ผลลัพธ์กลับมาเป็น undefined เช่นเดียวกัน
- null คือ ชนิดข้อมูลที่บอกให้ทราบว่า ได้กำหนดข้อมูลไปยังตัวแปรแล้ว แต่ไม่มีค่า (no value) หรืออ้างอิงไปยังออบเจ็กต์ที่ไม่มีอยู่

Note

ชนิดข้อมูลที่ใช้เป็นพื้นฐาน (Primitive) คือ ชนิดข้อมูลที่ไม่ใช่ออบเจ็กต์ ดังนั้น นอกจากชนิดข้อมูลที่เป็นพื้นฐานของ TypeScript ตามข้างต้นแล้ว ยังมีข้อมูลรูปแบบอื่น ๆ ที่จำเป็นต้องทราบเพิ่มเติม ได้แก่ ออบเจ็กต์, อาร์เรย์, ฟังก์ชัน, Enum, Tuple ฯลฯ ซึ่งจะกล่าวถึงโดยละเอียดในบทที่เกี่ยวข้องต่อไป

ประกาศตัวแปรโดยใช้คีย์เวิร์ด var, let และ const

เมื่อต้องการกำหนดตัวแปรขึ้นใช้งาน จะใช้คีย์เวิร์ด var, let หรือ const ตามด้วยชื่อตัวแปร และกำหนดค่าเริ่มต้นที่ต้องการให้กับตัวแปร

```
var name = value;
```

หรือ

```
let name = value;
```

หรือ

```
const name = value;
```

- **name** คือ ชื่อตัวแปร
- **value** คือ ค่าที่ต้องการกำหนดให้กับตัวแปร

การใช้คีย์เวิร์ด var, let และ const เพื่อประกาศตัวแปร มีความแตกต่างกัน ดังนี้

- ใช้คีย์เวิร์ด **var** สำหรับการประกาศตัวแปร (ไม่นิยมใช้ในปัจจุบัน)
- ใช้คีย์เวิร์ด **let** สำหรับการประกาศตัวแปรที่สามารถเปลี่ยนค่าได้ เช่น เดิมเก็บตัวเลข 10 สามารถเปลี่ยนไปเก็บตัวเลข 20 ได้
- ใช้คีย์เวิร์ด **const** สำหรับการประกาศตัวแปรซึ่งมีค่าคงที่ เมื่อกำหนดค่าเริ่มต้นไปยังตัวแปรแล้วไม่สามารถเปลี่ยนแปลงค่าได้ (read-only)

Note

ตัวแปรที่ใช้คีย์เวิร์ด var, const และ let มีขอบเขตการใช้งานตัวแปร (scope) ที่แตกต่างกัน ดังนี้

- ตัวแปรที่ใช้คีย์เวิร์ด var จะมีขอบเขตเฉพาะภายในฟังก์ชันซึ่งตัวแปรได้ถูกประกาศเอาไว้ (function scope) หรือถ้าตัวแปรถูกประกาศอยู่นอกฟังก์ชันก็จะมีลักษณะเป็นโกลบอล (global scope) ซึ่งสามารถเข้าถึงได้จากทุก ๆ ส่วนของโค้ด
- ตัวแปรที่ใช้คีย์เวิร์ด let หรือ const จะมีขอบเขตเฉพาะภายในบล็อกที่ตัวแปรได้ประกาศไว้เท่านั้น เช่น ภายในลูป while, ภายใน if, ภายใน for หรือภายในฟังก์ชัน

การประกาศตัวแปรโดยตรวจสอบชนิดข้อมูลอัตโนมัติ

TypeScript สามารถกำหนดชนิดข้อมูลอัตโนมัติ โดยพิจารณาจากค่าเริ่มต้นที่กำหนดให้กับตัวแปรในครั้งแรก เช่น เมื่อกำหนดค่าเริ่มต้นด้วยตัวเลข ตัวแปรจะมีชนิดข้อมูลเป็นแบบ number โดยอัตโนมัติ แต่ถ้ากำหนดค่าเริ่มต้นให้กับตัวแปรด้วยข้อความ ก็จะมีชนิดข้อมูลเป็นแบบ string เป็นต้น

เราเรียกความสามารถในการตรวจสอบชนิดข้อมูลจากค่าเริ่มต้นที่กำหนดให้กับตัวแปรเช่นนี้ว่า Type Inference

```
1  let counter = 10;               // มีชนิดข้อมูลเป็นแบบ number
2  let myText = ‘Hello World!’;    // มีชนิดข้อมูลเป็นแบบ string
3  let flag = true;                // มีชนิดข้อมูลเป็นแบบ boolean
```

- บรรทัดที่ 1 กำหนดค่าให้กับตัวแปรด้วยตัวเลข ตัวแปรจึงมีชนิดข้อมูลเป็นแบบ number
- บรรทัดที่ 2 กำหนดข้อความให้กับตัวแปร ก็จะมีชนิดข้อมูลเป็นแบบ string
- บรรทัดที่ 3 กำหนดค่าเป็น true ตัวแปรก็จะมีชนิดข้อมูลเป็นแบบ boolean

Note

ถ้าประกาศตัวแปรโดยไม่มีการกำหนดชนิดข้อมูล TypeScript จะกำหนดให้มีชนิดข้อมูลเป็นแบบ any โดยอัตโนมัติ ดังตัวอย่าง

```
let data;      // ไม่ได้กำหนดชนิดข้อมูลจะได้เป็น data: any
data = 10;
```

ในทางปฏิบัติเราจะหลีกเลี่ยงการใช้งาน any เนื่องจาก TypeScript จะไม่สามารถตรวจสอบชนิดข้อมูลจริง ๆ ได้ ซึ่งอาจทำให้การตรวจสอบโค้ดทำได้ยาก

ประกาศตัวแปรโดยระบุชนิดข้อมูลด้วยตนเอง

นอกจากวิธีกำหนดชนิดข้อมูลอัตโนมัติ (Type Inference) ตามค่าเริ่มต้นแล้ว เรายังสามารถระบุชนิดข้อมูลให้กับตัวแปรด้วยตนเองได้ (เรียกว่า Type Annotation) เริ่มจากกำหนดชื่อตัวแปร จากนั้นจึงใส่เครื่องหมาย : ตามด้วยชนิดข้อมูลที่ต้องการ

```
1 let first: number;
2 let second: string;
```

- บรรทัดที่ 1 ประกาศตัวแปร first โดยมีชนิดข้อมูลเป็นแบบ number
- บรรทัดที่ 2 ประกาศตัวแปร second โดยมีชนิดข้อมูลเป็นแบบ string

ค่าที่กำหนดให้กับตัวแปรจะต้องมีชนิดข้อมูลตรงกับชนิดข้อมูลที่ได้ประกาศไว้ เช่น ถ้ากำหนดชนิดข้อมูลเป็น number ค่าที่กำหนดให้กับตัวแปรจะต้องเป็นตัวเลข แต่ถ้ากำหนดชนิดข้อมูลเป็น string ค่าที่กำหนดจะต้องเป็นตัวอักษรหรือข้อความ เป็นต้น

```
const first: number = 1;
const second: string = “สอง”;
```

การประกาศตัวแปรโดยระบุชนิดข้อมูล ค่าที่จะกำหนดให้กับตัวแปรจะต้องเป็นชนิดข้อมูลแบบเดียวกันเสมอ หากไม่ตรงกันก็จะเกิด Error ขึ้น ดังตัวอย่าง

```
1 let second: string;
2 second = 2                 // Error
```

- บรรทัดที่ 1 ประกาศตัวแปร second ด้วยชนิดข้อมูลแบบ string
- บรรทัดที่ 2 เกิด Error เนื่องจากได้นำตัวเลข 2 ที่มีชนิดข้อมูลเป็นแบบ number ไปกำหนดให้กับตัวแปร second ที่มีชนิดข้อมูลเป็นแบบ string

ใช้ typeof ตรวจสอบชนิดข้อมูล

หากต้องการทราบว่าตัวแปรที่สนใจอยู่นั้นเก็บข้อมูลชนิดใดอยู่ จะใช้ typeof โอเปอเรเตอร์ตามด้วยข้อมูลที่ต้องการตรวจสอบ

```
typeof dataType
```

- typeof คือ โอเปอเรเตอร์สำหรับใช้ตรวจสอบชนิดข้อมูลของตัวแปร หรือค่าที่สนใจ
- dataType คือ ตัวแปร (หรือค่า) ที่ต้องการตรวจสอบชนิดข้อมูล

ต่อไปนี้คือตัวอย่าง การใช้งาน typeof สำหรับตรวจสอบชนิดข้อมูลแบบต่าง ๆ

```
1  let isLoading = true;
2  let num = 10.0;
3  console.log(typeof isLoading);     // boolean
4  console.log(typeof num);           // number
```

- บรรทัดที่ 1 ประกาศตัวแปร isLoading เก็บค่าความจริง true
- บรรทัดที่ 2 ประกาศตัวแปร num เก็บเลขทศนิยม
- บรรทัดที่ 3 ใช้ typeof ตรวจสอบชนิดข้อมูลที่อยู่ในตัวแปร isLoading จากนั้นแสดงผลลัพธ์ออกมาที่หน้าจอคอนโซล
- บรรทัดที่ 4 ใช้ typeof ตรวจสอบชนิดข้อมูลที่อยู่ในตัวแปร num จากนั้นแสดงผลลัพธ์ออกมาที่หน้าจอคอนโซล

ต่อไปนี้คือตารางแสดงชนิดข้อมูล และผลลัพธ์ที่ได้จากการใช้ typeof โอเปอเรเตอร์

ชนิดข้อมูล	ผลลัพธ์ที่ได้จาก typeof โอเปอเรเตอร์
string	“string”
number	“number”
boolean	“boolean”
undefined	“undefined”
null	“object”
Function หรือ Class	“function”
ออบเจ็กต์อื่น ๆ	“object”

Part

03

Node.js with TypeScript

17

สร้างแอปพลิเคชันในฝั่ง Backend ด้วย Node.js

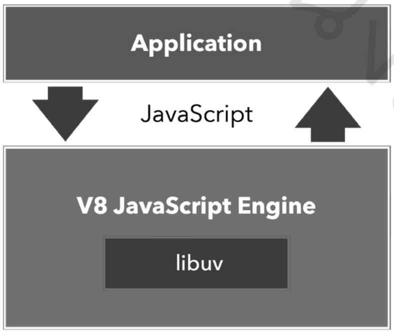
Node.js เป็นแพล็ตฟอร์มที่ถูกสร้างบนสภาพแวดล้อม ที่สามารถรัน JavaScript แอปพลิเคชันโดยไม่ต้องใช้เบราว์เซอร์ได้ การใช้ TypeScript กับ Node.js สามารถทำได้หลายวิธี แต่วิธีที่สะดวกและนิยมใช้งานอย่างกว้างคือ การใช้งานผ่านไลบรารี ts-node

ทบทวนเกี่ยวกับ Node.js

Node.js ได้ถูกพัฒนาขึ้นในปี ค.ศ. 2009 โดยนักพัฒนาที่ชื่อว่า Ryan Dahl โดย Node.js เป็นการกำหนดสภาพแวดล้อมเพื่อให้สามารถรัน JavaScript ในคอมพิวเตอร์โดยไม่ต้องใช้เบราว์เซอร์ (JavaScript Runtime Environment)

จุดเด่นของ Node.js ช่วยให้นักพัฒนาสามารถใช้ JavaScript สร้างแอปพลิเคชันในฝั่งเซิร์ฟเวอร์, สามารถอ่านเขียนไฟล์ในคอมพิวเตอร์ ตลอดจนสามารถควบคุมการรับ ส่งข้อมูลผ่านเครือข่ายได้ และ Node.js ยังสามารถทำงานข้ามแพลตฟอร์มได้ เช่น สามารถรันบนวินโดวส์, แมคโอเอส หรือลินุกซ์

ภายใน Node.js จะประกอบด้วย JavaScript Engine V8 ของ Google Chrome ทำหน้าที่แปลชุดคำสั่ง JavaScript และมีโค้ดคำสั่งภาษา C++ เพื่อกำหนดสภาพแวดล้อมต่าง ๆ ที่จำเป็น เช่น กำหนดไลบรารี libuv สำหรับรันชุดคำสั่งที่ใช้เวลาในการประมวลผล หรือใช้เพื่อติดต่อกับระบบปฏิบัติการ



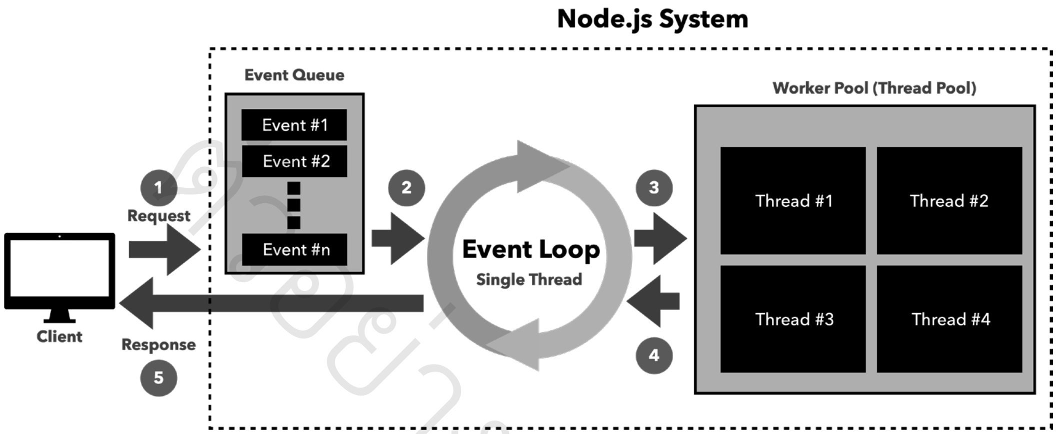
Node.js ถูกออกแบบให้สามารถทำงานได้ด้วยหน่วยประมวลผลเพียงชุดเดียว (Single Thread), รองรับชุดคำสั่งในแบบที่ต้องใช้เวลาในการประมวลผล (Asynchronous) และสามารถทำงานหลายอย่างได้พร้อมกันได้ต่อเนื่อง โดยไม่ขัดจังหวะในการทำงานอื่น (non-blocking I/O)

ด้วยเหตุผลดังกล่าว แอปพลิเคชันที่สร้างจาก Node.js จึงสามารถทำงานได้รวดเร็วและมีประสิทธิภาพมาก ปัจจุบัน Node.js ถูกนำมาใช้งานอย่างกว้างขวาง โดยเฉพาะการสร้างเน็ตเวิร์คแอปพลิเคชัน

การทำงานของ Node.js

Node.js ถูกออกแบบให้ทำงานตามเหตุการณ์ที่เกิดขึ้น (เรียกว่า Event-Driven Architecture) เช่น โปรแกรมจะทำงานเมื่อผู้ใช้คลิกปุ่มที่กำหนด หรือโปรแกรมทำงานเมื่อผู้ใช้ส่งการร้องขอข้อมูลมายังเซิร์ฟเวอร์ เป็นต้น

Node.js ได้กำหนด Event Loop สำหรับรันคำสั่ง JavaScript ทั่วไป และได้กำหนด Worker Pool (หรือ Thread Pool) สำหรับรันคำสั่งที่ต้องใช้เวลาหรือพลังซีพียูจำนวนมาก (เรียกว่า Expensive Tasks) เช่น การอ่านเขียนไฟล์จากฮาร์ดดิสก์ การบีบอัดไฟล์ หรือการเข้ารหัสไฟล์ ฯลฯ



จากรูป เมื่อมีอีเวนต์ (Event) เกิดขึ้น ลำดับของอีเวนต์ที่เกิดขึ้นจะถูกนำไปเรียงไว้ใน Event Queue เพื่อสั่งให้ทำงานบางอย่างซึ่งผูกไว้กับอีเวนต์นั้น ๆ โดย Event Loop จะตรวจสอบว่าควรทำงานนั้นภายใน Event Loop หรือส่งต่องานไปให้กับ Worker Pool ทำงานแทน

1. เมื่อไคลเอนต์ต่าง ๆ ส่ง Request มายังเซิร์ฟเวอร์ เช่น ไคลเอนต์ A ขอให้เซิร์ฟเวอร์เข้ารหัสไฟล์, ไคลเอนต์ B ขอให้เซิร์ฟเวอร์ค้นหารายชื่อสินค้าในฐานข้อมูล, ไคลเอนต์ C ขอข้อมูลเว็บเพจไปแสดงผลบนเบราว์เซอร์ ฯลฯ Node.js จะกำหนดอีเวนต์และนำอีเวนต์มาเรียงไว้ใน Event Queue เพื่อจัดลำดับคิวงาน
2. หากเป็นคำสั่งทั่วไป (เช่น คำนวณพื้นที่วงกลม คำนวณยอดเงินในบัญชี) คำสั่งเหล่านั้นจะถูกนำมารันใน Event Loop หลังจากคำนวณค่าเรียบร้อยก็จะส่ง Response กลับไปให้กับไคลเอนต์
3. หาก Event Loop พบว่างานที่ทำนั้นเป็นงานที่ใช้เวลา หรือใช้พลังซีพียูจำนวนมาก (เช่น การเข้ารหัสและการถอดรหัสข้อมูล) ก็จะต่อส่งงานต่อไปให้กับ Worker Pool ช่วยทำงานแทน
4. หลังจากที่ Worker Pool ทำงานเสร็จแล้ว (เช่น หลังจากเข้ารหัสไฟล์เรียบร้อยแล้ว) ข้อมูลจะถูกส่งกลับไปยัง Event Loop เพื่อนำข้อมูลไปใช้งาน
5. เมื่อ Event Loop ทำงานตามที่กำหนดไว้เสร็จแล้วจะส่ง Response กลับไปให้กับไคลเอนต์

Note

Node.js ถูกออกแบบให้ใช้เธรด (Thread) จำนวน 1 เธรด สำหรับรันคำสั่งใน Event Loop ซึ่งเหมาะสำหรับการทำงานโดยทั่วไป หรือเป็นคำสั่งที่ไม่บล็อกกระบวนการทำงาน (non-blocking operations) แต่ถ้า Node.js พบว่างานนั้นต้องใช้เวลา หรือกินพลังของซีพียู (Expensive Task) ก็จะใช้เธรดเพิ่มเติมจาก Worker Pools

การใช้ TypeScript กับ Node.js

การสร้าง Node.js แอปพลิเคชัน โดยใช้ TypeScript สามารถแบ่งได้เป็น 6 ขั้นตอน ดังนี้

- ขั้นตอนที่ 1 ติดตั้ง Node.js หลังจากติดตั้งเราจะได้ npm ซึ่งเป็นเครื่องมือสำหรับใช้จัดการกับแพ็คเกจมาใช้โดยอัตโนมัติ
- ขั้นตอนที่ 2 สร้างโฟลเดอร์สำหรับเก็บข้อมูลโปรเจ็กต์ทั้งหมด และสร้างไฟล์ package.json ไว้ภายในโปรเจ็กต์ เพื่อเก็บรายละเอียดพื้นฐานของแอปพลิเคชัน เช่น ชื่อผู้พัฒนา, หมายเลขเวอร์ชันของแอปพลิเคชัน, ชนิดของไลเซนต์ของแอปพลิเคชัน, รายชื่อโมดูลที่ติดตั้งในโปรเจ็กต์ ฯลฯ
- ขั้นตอนที่ 3 ติดตั้งแพ็คเกจต่าง ๆ ที่จำเป็นต้องใช้งานในโปรเจ็กต์ เช่น typescript, ts-node, nodemon และ express เป็นต้น
- ขั้นตอนที่ 4 สร้างไฟล์สำหรับใช้ตั้งค่า เช่น สร้างไฟล์ tsconfig.json จากนั้นแก้ไขการตั้งค่าสำหรับการใช้งานกับ TypeScript เช่น เลือกโฟลเดอร์ที่ใช้เป็นโฟลเดอร์หลัก เลือกเวอร์ชันของ JavaScript ตลอดจนเลือกเปิดหรือปิดคุณสมบัติที่ต้องการใช้งาน ฯลฯ นอกจากนั้นเรายังสามารถเขียนสคริปต์ในไฟล์ package.json สำหรับใช้รันแอปพลิเคชันด้วยรูปแบบต่าง ๆ ได้
- ขั้นตอนที่ 5 สร้างโฟลเดอร์และไฟล์ TypeScript (ไฟล์นามสกุล .ts) ไว้ภายในโปรเจ็กต์ โดยชื่อโฟลเดอร์หลักและโฟลเดอร์ที่ใช้เก็บผลลัพธ์จากการคอมไพล์ จะต้องตรงกับที่ได้ตั้งค่าเอาไว้ในขั้นตอนที่ 4
- ขั้นตอนที่ 6 ทดสอบการทำงานของแอปพลิเคชัน โดยใช้คำสั่ง ts-node ตามด้วยชื่อไฟล์ TypeScript ที่ต้องการรัน หรือรันแอปพลิเคชันโดยใช้สคริปต์ที่เขียนขึ้นก็ได้ เช่น ใช้คำสั่ง npm run dev สำหรับรัน TypeScript พร้อมกับติดตามการเปลี่ยนโค้ด เมื่อมีการแก้ไขโค้ดก็จะสั่งรีสตาร์ต เพื่อนำผลลัพธ์ล่าสุดมาแสดงผลโดยอัตโนมัติ

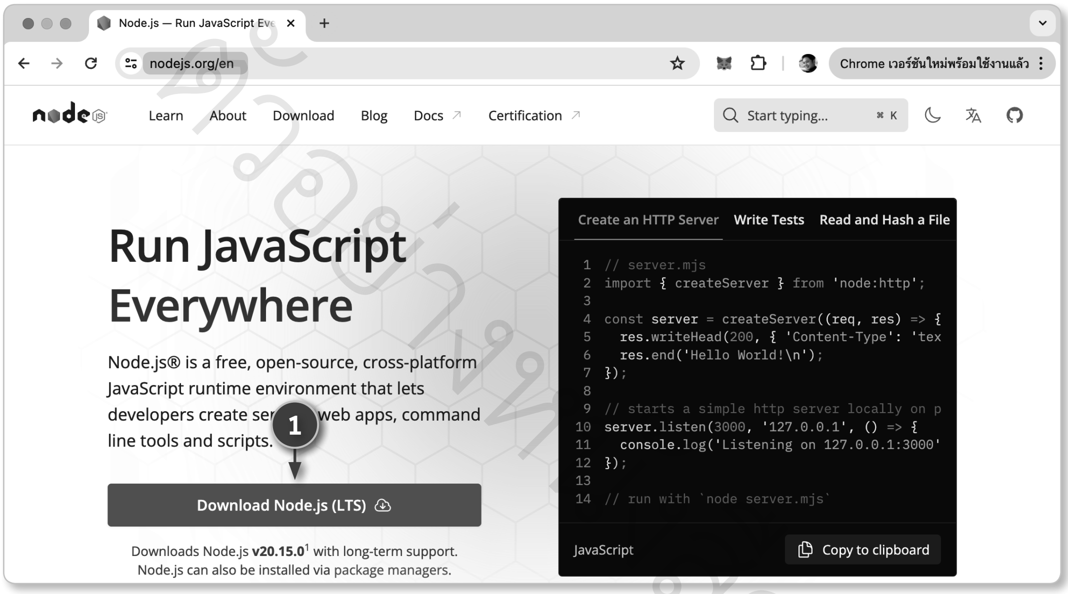
สร้างโปรเจ็กต์สำหรับใช้ TypeScript กับ Node.js

การสร้างโปรเจ็กต์ TypeScript สำหรับ Node.js จะต้องติดตั้งแพ็คเกจที่จำเป็น ได้แก่ typescript, nodemon, ts-node, express และอื่น ๆ โดยสามารถแบ่งได้เป็น 6 ขั้นตอน ดังนี้

ขั้นตอนที่ 1 ติดตั้ง Node.js

สามารถดาวน์โหลด Node.js ได้จากเว็บไซต์ https://nodejs.org

1. ไปที่เว็บไซต์ https://nodejs.org/ คลิกเพื่อดาวน์โหลด Node.js เวอร์ชันที่เป็น LTS (Long Term Support) ซึ่งหมายถึงเวอร์ชันที่ถูกใช้งานอย่างแพร่หลายโดยไม่พบปัญหา



Run JavaScript Everywhere

Node.js® is a free, open-source, cross-platform JavaScript runtime environment that lets developers create servers, web apps, command line tools and scripts.

Downloads Node.js **v20.15.0**¹ with long-term support.
Node.js can also be installed via package managers.

```
// server.mjs
import { createServer } from 'node:http';

const server = createServer((req, res) => {
  res.writeHead(200, { 'Content-Type': 'tex
  res.end('Hello World!\n');
});

// starts a simple http server locally on p
server.listen(3000, '127.0.0.1', () => {
  console.log('Listening on 127.0.0.1:3000'
});

// run with `node server.mjs`
```

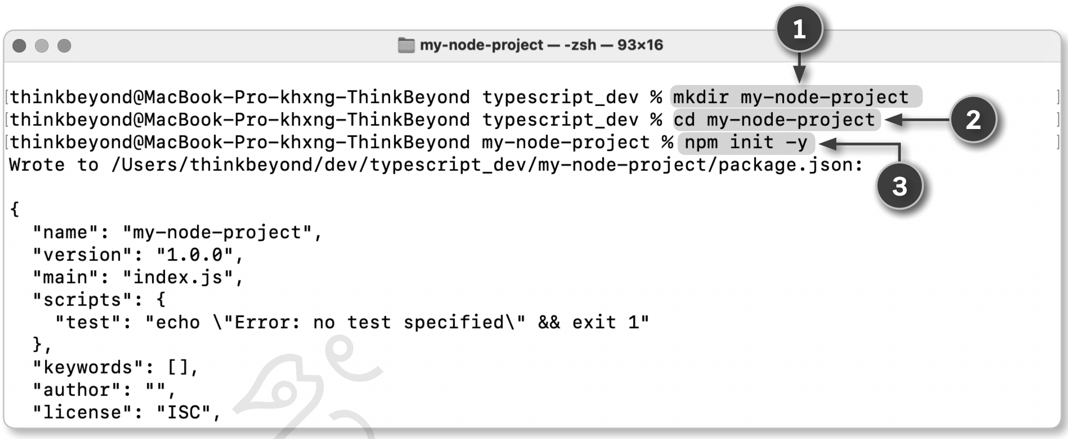
2. หลังจากดาวน์โหลดไฟล์สำหรับติดตั้งเรียบร้อยแล้ว ให้ดับเบิลคลิกเพื่อติดตั้ง Node.js จากนั้นทำตามขั้นตอนที่ปรากฏบนหน้าจนกระทั่งติดตั้งเรียบร้อย

ขั้นตอนที่ 2 สร้างโปรเจ็กต์

ให้สร้างโฟลเดอร์สำหรับเก็บข้อมูลโปรเจ็กต์ เช่น สร้างโฟลเดอร์ชื่อว่า my-node-project จากนั้นจึงเข้าไปยังโฟลเดอร์ my-node-project เพื่อตั้งค่าเริ่มต้นต่าง ๆ ให้กับโปรเจ็กต์

1. สร้างโฟลเดอร์ที่ใช้เก็บโปรเจ็กต์ เช่น สร้างโฟลเดอร์ my-node-project โดยใช้คำสั่ง mkdir my-node-project
2. เข้าไปยังโฟลเดอร์ที่เพิ่งสร้างขึ้น เช่น เข้าไปยังโฟลเดอร์ my-node-project โดยพิมพ์คำสั่ง cd my-typescript-project

3. สร้างไฟล์ package.json ซึ่งใช้เก็บการตั้งค่าต่าง ๆ สำหรับ Node.js ให้พิมพ์คำสั่ง npm init -y เพื่อสร้างไฟล์ package.json พร้อมกำหนดค่าเริ่มต้นโดยอัตโนมัติ

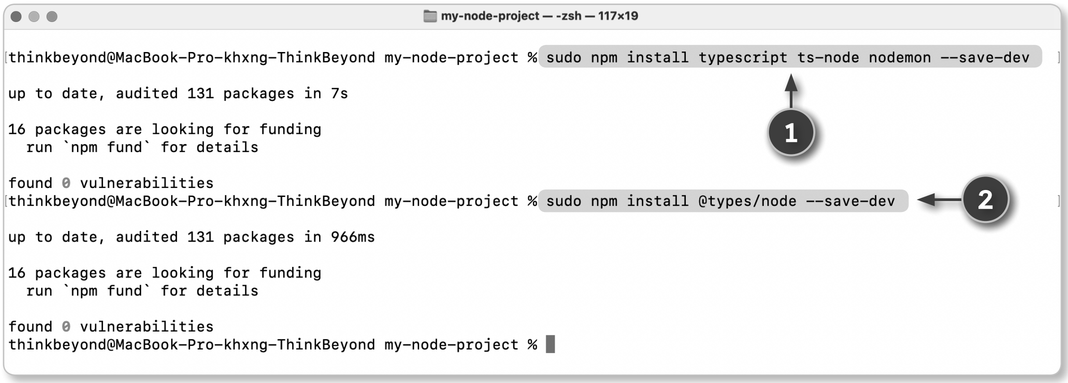


```
[thinkbeyond@MacBook-Pro-khxng-ThinkBeyond typescript_dev % mkdir my-node-project
[thinkbeyond@MacBook-Pro-khxng-ThinkBeyond typescript_dev % cd my-node-project
[thinkbeyond@MacBook-Pro-khxng-ThinkBeyond my-node-project % npm init -y
Wrote to /Users/thinkbeyond/dev/typescript_dev/my-node-project/package.json:

{
  "name": "my-node-project",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
```

ขั้นตอนที่ 3 ติดตั้งแพ็คเกจที่จะใช้งานในโปรเจ็กต์

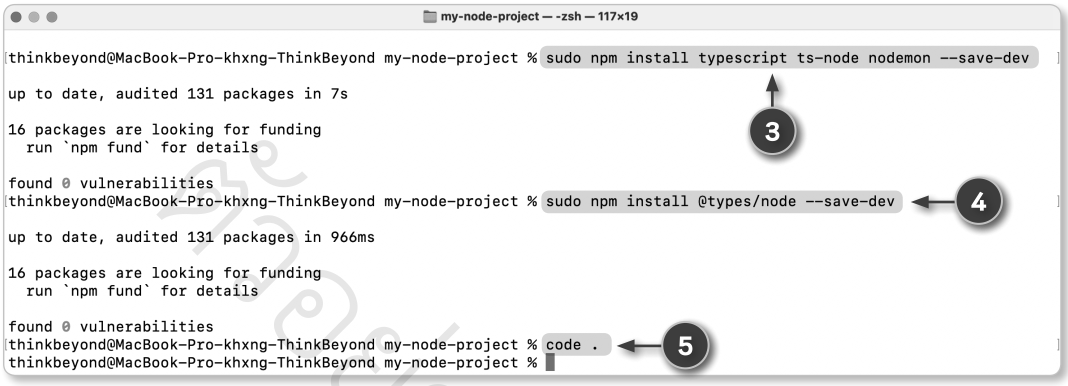
การใช้ TypeScript เพื่อพัฒนา Node.js จะต้องติดตั้งแพ็คเกจเพิ่มเข้าไปยังโปรเจ็กต์ โดยทั่วไปแล้วจะต้องติดตั้งแพ็คเกจดังต่อไปนี้

1. ติดตั้งแพ็คเกจพื้นฐานสำหรับการใช้ TypeScript เพื่อพัฒนา Node.js ได้แก่ typescript, ts-node, และ nodemon โดยใช้คำสั่ง npm install typescript ts-node nodemon --save-dev (สำหรับแมคโอเอสให้ใช้คำสั่ง sudo npm install typescript ts-node nodemon --save-dev จากนั้นกรอกรหัสผ่านของผู้ดูแลระบบ)
2. หากต้องการใช้งานโมดูลต่าง ๆ ที่ติดมากับ Node.js เช่น โมดูล fs, โมดูล path ฯลฯ จะต้องติดตั้ง Type Definition File สำหรับใช้อธิบายชนิดข้อมูลที่อยู่โมดูลที่มาพร้อมกับ Node.js โดยพิมพ์คำสั่ง npm install @types/node --save-dev (สำหรับแมคโอเอสให้ใช้คำสั่ง sudo npm install @types/node --save-dev)



```
[thinkbeyond@MacBook-Pro-khxng-ThinkBeyond my-node-project % sudo npm install typescript ts-node nodemon --save-dev

up to date, audited 131 packages in 7s

16 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
[thinkbeyond@MacBook-Pro-khxng-ThinkBeyond my-node-project % sudo npm install @types/node --save-dev

up to date, audited 131 packages in 966ms

16 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
thinkbeyond@MacBook-Pro-khxng-ThinkBeyond my-node-project % 
```

3. ติดตั้งแพ็คเกจที่ใช้ในโปรเจ็กต์ เช่น หากต้องการใช้ Express สำหรับสร้างเว็บ API ก็ให้ใช้คำสั่ง npm install express (สำหรับแมคโอเอสให้ใช้คำสั่ง sudo npm install express)
4. ติดตั้ง Type Definition File สำหรับอธิบายชนิดข้อมูลที่อยู่ในโมดูล Express ด้วยคำสั่ง npm install @types/express --save-dev (สำหรับแมคโอเอสให้ใช้คำสั่ง sudo npm install @types/express --save-dev)
5. ใช้คำสั่ง code . เพื่อเปิดโปรเจ็กต์โดยใช้ Visual Studio Code

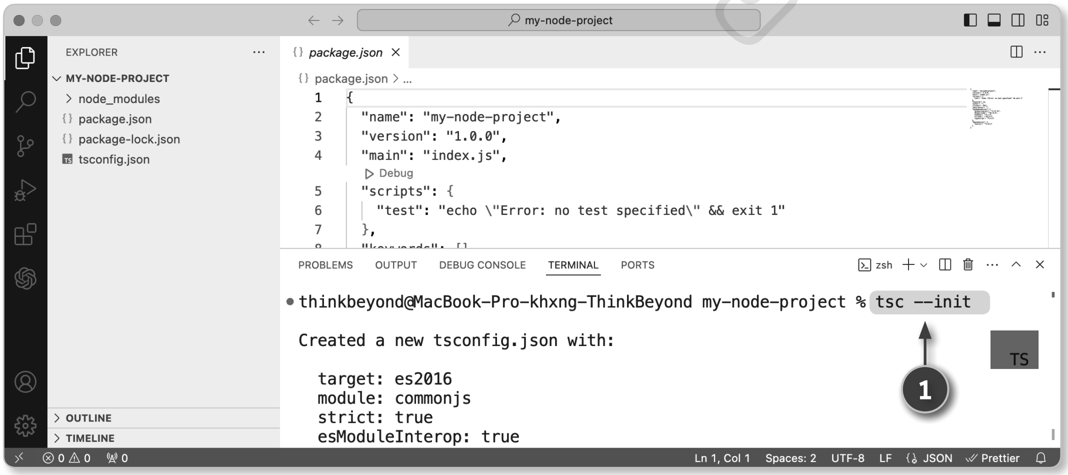


```
thinkbeyond@MacBook-Pro-khxng-ThinkBeyond my-node-project % sudo npm install typescript ts-node nodemon --save-dev

up to date, audited 131 packages in 7s

16 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
thinkbeyond@MacBook-Pro-khxng-ThinkBeyond my-node-project % sudo npm install @types/node --save-dev

up to date, audited 131 packages in 966ms

16 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
thinkbeyond@MacBook-Pro-khxng-ThinkBeyond my-node-project % code .
thinkbeyond@MacBook-Pro-khxng-ThinkBeyond my-node-project %
```

ขั้นตอนที่ 4 สร้างไฟล์สำหรับใช้ตั้งค่าและปรับแต่งการตั้งค่าต่าง ๆ

การใช้งาน TypeScript ใน Node.js จะต้องสร้างไฟล์ tsconfig.json สำหรับกำหนดการตั้งค่าต่าง ๆ ที่เกี่ยวข้องกับการใช้งาน TypeScript เช่น กำหนดโฟลเดอร์ที่เป็นจุดเริ่มต้นของโปรเจ็กต์ กำหนดโฟลเดอร์ที่ใช้เก็บไฟล์ JavaScript ที่ได้จากการคอมไพล์ หรือกำหนดเวอร์ชันของ JavaScript ที่จะใช้งาน ฯลฯ

1. ใน Visual Studio Code ให้เปิดหน้าต่าง Terminal (ด้วยเมนู View>Terminal) จากนั้นใช้คำสั่ง tsc --init เพื่อสร้างไฟล์ tsconfig.json (เป็นไฟล์ที่ใช้สำหรับการตั้งค่า TypeScript)



```
{
  "name": "my-node-project",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
```

```
thinkbeyond@MacBook-Pro-khxng-ThinkBeyond my-node-project % tsc --init

Created a new tsconfig.json with:

  target: es2016
  module: commonjs
  strict: true
  esModuleInterop: true
```

2. ปรับแต่งการตั้งค่าในไฟล์ tsconfig.json ตามต้องการ เช่น กำหนดโฟลเดอร์สำหรับใช้เป็นรูท (rootDir) และโฟลเดอร์สำหรับเก็บผลลัพธ์จากการคอมไพล์ (outDir)

```
 1  {
 2    “compilerOptions”: {
 3      “target”: “ES2016”,
 4      “module”: “commonjs”,
 5      “rootDir”: “./src”,
 6      “moduleResolution”: “node”,
 7      “outDir”: “./dist”,
 8      “esModuleInterop”: true,
 9      “forceConsistentCasingInFileNames”: true,
10      “strict”: true,
11      “skipLibCheck”: true
12    }
13  }
```

- บรรทัดที่ 5 ที่ “rootDir” กำหนดรูทโฟลเดอร์สำหรับเก็บไฟล์ TypeScript ซึ่งใช้เป็นจุดเริ่มต้นในการรันแอปพลิเคชัน โดยในตัวอย่างใช้เป็นโฟลเดอร์ src
- บรรทัดที่ 7 ที่ “outDir” กำหนดโฟลเดอร์สำหรับเก็บไฟล์ JavaScript ที่เป็นผลลัพธ์จากการคอมไพล์จาก TypeScript ให้กลายเป็นไฟล์ JavaScript โดยในตัวอย่างใช้เป็นโฟลเดอร์ dist

3. ปรับแต่งไฟล์ package.json เพื่อเพิ่มสคริปต์สำหรับใช้รันแอปพลิเคชัน

```
{
  "name": "my-node-project",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "tsc",
    "start": "npm run build && node dist/app.js",
    "dev": "nodemon --exec ts-node src/app.ts"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "devDependencies": {
```

จากรูปได้เพิ่มสคริปต์คำสั่ง “build” สำหรับคอมไพล์ TypeScript ให้กลายเป็น JavaScript, เพิ่มสคริปต์ “start” สำหรับคอมไพล์และรันโค้ดในไฟล์ app.ts และเพิ่มสคริป “dev” สำหรับทดสอบ TypeScript ผ่านทาง ts-node เป็นต้น

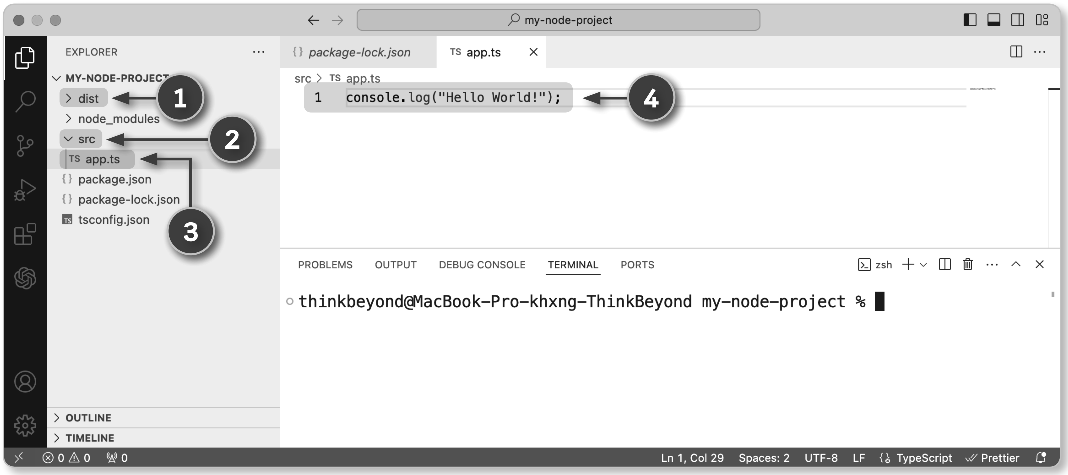
```
1 {
2   “scripts”: {
3     “test”: “echo \”Error: no test specified\” && exit 1”,
4     “build”: “tsc”,
5     “start”: “npm run build && node dist/app.js”,
6     “dev”: “nodemon --exec ts-node src/app.ts”
7   },
8 }
```

- บรรทัดที่ 4 คำสั่ง “build”:“tsc” เป็นสคริปต์ที่ใช้คอมไพล์ไฟล์ TypeScript ให้กลายเป็น JavaScript
- บรรทัดที่ 5 คำสั่ง “start”: “npm run build && node dist/app.js” เป็นสคริปต์ที่ใช้คอมไพล์ไฟล์ TypeScript ให้กลายเป็น JavaScript จากนั้นใช้ node เพื่อรันชุดคำสั่งที่อยู่ในไฟล์ app.js ซึ่งเป็นไฟล์ JavaScript
- บรรทัดที่ 6 คำสั่ง “dev”: “nodemon --exec ts-node src/app.ts” เป็นสคริปที่ใช้ ts-node เพื่อรัน TypeScript ที่อยู่ในไฟล์ src/app.ts โดยตรง พร้อมกับสั่งให้ติดตามการเปลี่ยนแปลงโค้ดโดยใช้ nodemon ทุกครั้งที่มีการแก้ไขไฟล์ก็จะสั่งให้รีสตาร์ตเพื่อนำโค้ดที่อัปเดตล่าสุดมารันโดยอัตโนมัติ

ขั้นตอนที่ 5 สร้างโฟลเดอร์และไฟล์ TypeScript

การกำหนดโฟลเดอร์ภายในโปรเจ็กต์ จะต้องสัมพันธ์กับการตั้งค่าที่อยู่ในไฟล์ tsconfig.json เช่น กำหนดโฟลเดอร์ src เก็บไฟล์ TypeScript กำหนดไฟล์ app.ts เป็นไฟล์เริ่มต้นแอปพลิเคชัน และกำหนดโฟลเดอร์ dist สำหรับเก็บไฟล์ JavaScript ที่เป็นผลลัพธ์จากการคอมไพล์ เป็นต้น

1. สร้างโฟลเดอร์ dist ลงในโปรเจ็กต์สำหรับเก็บไฟล์ JavaScript ที่ได้จากการคอมไพล์
2. สร้างโฟลเดอร์ src ลงในโปรเจ็กตสำหรับเก็บไฟล์ TypeScript
3. สร้างไฟล์ app.ts ลงในโฟลเดอร์ src สำหรับใช้เป็นจุดเริ่มต้นของแอปพลิเคชัน
4. แก้ไขไฟล์ app.ts โดยกรอกโค้ดคำสั่ง TypeScript ลงไป เช่น console.log(“Hello World!”);



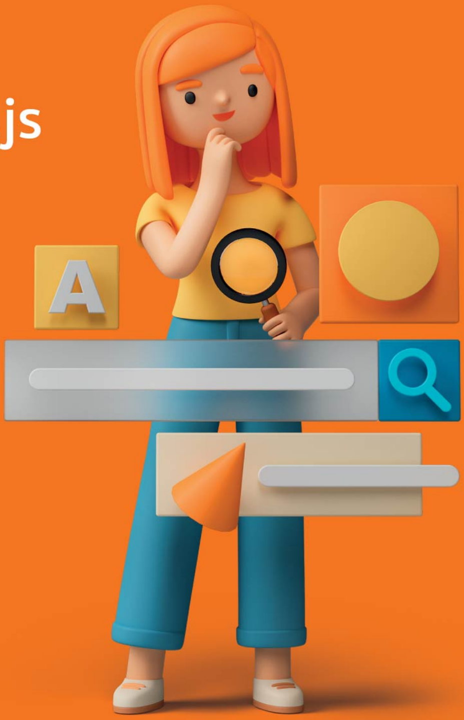
```
1   console.log("Hello World!");
```

```
thinkbeyond@MacBook-Pro-khxng-ThinkBeyond my-node-project %
```

Think Beyond Develop

ใช้ TypeScript ร่วมกับ Node.js

สร้างได้ทั้ง Frontend และ Backend



ด้วยเนื้อหาที่แน่นและกระชับ

ปรับแต่งให้เข้าใจได้ง่าย

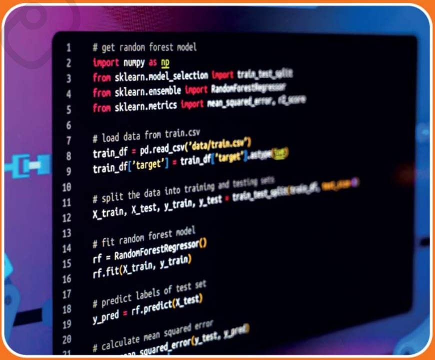
อ่านและทำตามได้แบบ

Step-By-Step

เก่งได้ แม้ไม่มีพื้นฐานมาก่อน

- สรุปพื้นฐาน TypeScript ทั้งหมดอย่างเป็นระบบ
- อธิบาย TypeScript ด้วยตัวอย่าง พร้อมคำอธิบายในทุกขั้นตอน
- การใช้งาน Enum และ Tuple
- วิธีใช้งานอาร์เรย์และอาร์เรย์เมธอดโดยละเอียด
- กำหนดโครงสร้างข้อมูลโดยใช้ออบเจ็กต์ อินเตอร์เฟส และคลาส
- การใช้งาน Optional Property และ Optional Parameter
- การใช้งาน TypeScript ขั้นสูง เช่น Generics, Decorators และ Metadata

- การทำงานกับ async/await และ Promise อย่างมีประสิทธิภาพ
- วิธีสร้างโปรเจ็กต์ด้วย TypeScript และ Node.js
- ตัวอย่างการอ่านเขียนไฟล์ใน Node.js
- การใช้งานไลบรารีและการติดตั้ง Type Definition Files
- พื้นฐานเกี่ยวกับ API, Middleware และ Router
- การสร้าง API ด้วย Express และ TypeScript
- วิธีจัดการ Authentication และ Authorization ในแอปพลิเคชัน



```
# get random forest model
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score

# load data from train.csv
train_df = pd.read_csv('data/train.csv')
train_df['target'] = train_df['target'].astype([illegible])

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(train_df, test_size[illegible]

# fit random forest model
rf = RandomForestRegressor()
rf.fit(X_train, y_train)

# predict labels of test set
y_pred = rf.predict(X_test)

# calculate mean squared error
```

9 786164 495371

ราคา 480 บาท