// 5.2 練習

// 練習 A：MongoDB架構層次

// 1. 基礎題

// 1.1 請說明MongoDB的四個架構層次，並用一個生活中的例子來比喻。

// MongoDB 的四個架構層次:

// Database（資料庫）

// MongoDB 中的最上層結構。

// 一個 MongoDB 伺服器可以包含多個資料庫。

// 每個資料庫彼此獨立，像是不同的資料夾。

// Collection（集合）

// 資料庫中的一組文件（documents）。

// 類似於關聯式資料庫中的「資料表」，但不需要固定的 schema。

// 同一個資料庫可以有多個集合。

// Document（文件）

// MongoDB 的基本資料單位，使用 BSON（類似 JSON）格式儲存。

// 每個文件可以有不同的欄位與資料型別，彈性高。

// Field（欄位）

// 文件中的每個鍵值對（key-value pair）。

// 類似於資料表中的欄位，但每個文件可以有不同的欄位。

// 生活中的比喻：家中的書櫃

// 想像你家裡有一個大書櫃，這個書櫃可以幫助我們理解 MongoDB 的架構：

// MongoDB 層級 比喻  說明

// Database 書櫃  書櫃裡可以放很多本相簿（每本代表一個主題）

// Collection 相簿  每本相簿收集了某一類的照片（例如：旅遊、家庭、寵物）

// Document 一張照片  每張照片記錄了一次事件（例如：2023年去日本旅遊）

// Field  照片上的註記  每張照片上可以寫上日期、地點、人物等資訊

// 1.2 在MongoDB中，如果你要切換到一個名為 shopDB 的資料庫，應該使用什麼指令？

// use 是 MongoDB 的指令，用來切換目前操作的資料庫。

// 如果 shopDB 尚未存在，MongoDB 不會立即建立它，直到你在其中建立集合（collection）或插入文件（document）時才會真正創建。

// 你可以把它想像成走進一間空的房間，只有當你開始擺放家具（資料）時，這個房間才真正成為一個「住家」（資料庫）

// 1.3 請寫出將以下資料插入到 products 集合的指令：

// •  產品名稱：無線滑鼠

// •  價格：350

// •  類別：電腦周邊

// Question 1.3 Answer:

// db.products.insertOne({

//   name: "無線滑鼠",

//   price: 350,

//   category: "電腦周邊"

// })

// 1.4 同一個集合裡的文件是否必須有相同的欄位？

// 不必！在 MongoDB 中，同一個集合（Collection）裡的文件（Document）不需要有相同的欄位，這是 MongoDB 的一大特色

// 1.5 如果你要一次插入三個產品資料到資料庫，應該使用 insertOne() 還是 insertMany()？

// Question 1.5 Answer:

// 你應該使用 insertMany()，因為它是專門用來一次插入多筆文件（documents）到集合（collection）的方法。

// db.products.insertMany([

//   {

//     name: "無線滑鼠",

//     price: 350,

//     category: "電腦周邊"

//   },

//   {

//     name: "機械式鍵盤",

//     price: 1200,

//     category: "電腦周邊"

//   },

//   {

//     name: "USB-C 集線器",

//     price: 650,

//     category: "配件"

//   }

// ])

// 2. 判斷題（對或錯）

// 2.1 在插入第一份文件之前，必須先使用 db.createCollection() 建立集合。（ ）

// Question 2.1 Answer:

// 錯。

// 在 MongoDB 中，不需要先使用 db.createCollection() 來建立集合。你可以直接使用 insertOne() 或 insertMany() 插入文件，MongoDB 會自動建立集合（Collection）和資料庫（Database）（如果它們尚未存在）。

// 2.2 MongoDB會在你第一次使用資料庫時自動建立該資料庫。（ ）

// Question 2.2 Answer:

// 對。

// 在 MongoDB 中，當你第一次使用某個資料庫名稱（例如 use shopDB）時，MongoDB 並不會立刻建立該資料庫。但只要你在該資料庫中執行了實際操作（例如插入資料或建立集合），MongoDB 就會自動建立這個資料庫。

// 2.3 一個資料庫最多只能有10個集合。（ ）

// Question 2.3 Answer:

// 錯。

// 在 MongoDB 中，一個資料庫可以擁有不限數量的集合（collections），並沒有「最多只能有 10 個集合」的限制。

// 2.4 集合的名稱可以包含空格和特殊符號。（ ）

// Question 2.4 Answer:

// 錯。

// 在 MongoDB 中，集合（Collection）的名稱有一些命名限制，不能包含空格或某些特殊符號。

// 2.5 如果一份文件有某個欄位，其他文件也必須有這個欄位。（ ）

// Question 2.5 Answer:

// 錯。

// 在 MongoDB 中，文件（Document）之間不需要有相同的欄位。這是 MongoDB 作為 NoSQL 資料庫的一大特色：結構彈性高，每筆資料可以根據需要儲存不同的欄位與格式。

// 3. Debug 題

// 3.1 以下程式碼有什麼問題？請指出錯誤並修正。

// use myDatabase

// db.users.insertOne(

//   name: "王小明",

//   age: 30

// )

// Question 3.1 answer:

// { } missing.

// 3.2 以下程式碼有什麼問題？請指出錯誤並修正。

// db.createDatabase("shopDB")

// db.shopDB.products.insertOne({ name: "手機" })

// Question 3.2 answer:

//  createDatabase doesn't exist

// 3.3 以下程式碼有什麼問題？請指出錯誤並修正。

// use products

// db.myShop.insertMany({

//   name: "產品A",

//   price: 100

// })

// Question 3.3 answer:

// inserOne instead of insertMany

// 4. 進階題

// 4.1 假設你正在設計一個部落格系統，請列出你會建立哪些集合，每個集合會包含什麼欄位。

// Question 4.1 answer:

// db blogDB

// collections

// post

//  -\_id

//  -title

//  -content

//  -author

//  -category

//  -tags

//  -createdAt

//  -puplishAt

// comment

//  -\_id

//  -author

//  -content

//  -createdAt

// users

//  -\_id

//  -email

//  -password

//  -bio

//  -createdAt

// categories

//  -\_id

//  -name

//  -description

// 4.2 如果你想查詢 users 集合中所有使用者的資料，並以易讀的格式顯示，應該使用什麼指令？

// Question 4.2 Answer:

// db.products.find() or db.products.find().pretty()

// 4.3 請解釋為什麼MongoDB要設計成「用到才建立」資料庫和集合，而不是要求事先建立？

// Question 4.3 answer:

// MongoDB 採用「用到才建立（lazy creation）」的設計，是為了提升開發效率、簡化流程，並符合它作為 NoSQL 資料庫的核心理念。以下是幾個主要原因：

// 1. 提升開發靈活性

// 開發者不需要事先定義資料庫或集合的結構。

// 可以直接插入資料，MongoDB 會自動建立所需的資料庫與集合。

// 這讓原型設計（prototyping）和快速開發變得更容易。

// 2. 符合動態結構的特性

// MongoDB 是 schema-less（無固定結構）的資料庫。

// 文件之間可以有不同欄位，因此不需要像關聯式資料庫那樣事先定義表格結構。

// 自動建立集合更符合這種彈性資料模型。

// 3. 簡化部署與維護

// 不需要額外的初始化腳本或設定流程。

// 對於 DevOps 或自動化部署來說，減少了建立資料庫的步驟，降低出錯機率。

// 4. 節省資源

// MongoDB 只有在實際儲存資料時才分配資源。

// 沒有資料的集合或資料庫不會佔用空間，有助於系統效能與儲存管理。

// 比喻：像是租倉庫

// 想像你在倉儲公司租儲物櫃：

// 你不需要一開始就租 10 個櫃子。

// 當你有東西要存放時，才會開一個櫃子。

// 這樣更省錢、更有效率，也不會浪費空間。

// 4.4 在同一個集合裡，有些文件有 phoneNumber 欄位，有些沒有。當你要查詢並顯示所有使用者的電話號碼時，應該如何處理？請寫出程式碼。

// Question 4.4 answer:

// 你可以使用 MongoDB 的查詢語法來篩選出有 phoneNumber 欄位的文件，並只顯示該欄位。以下是範例程式碼：

// javascript

// db.users.find(

//   { phoneNumber: { $exists: true } }, // 條件：只找有 phoneNumber 欄位的文件

//   { \_id: 0, phoneNumber: 1 }          // 投影：只顯示 phoneNumber 欄位，不顯示 \_id

// )

// 語法說明：

// { phoneNumber: { $exists: true } }：這個查詢條件會找出所有有 phoneNumber 欄位的文件。

// { \_id: 0, phoneNumber: 1 }：這是投影（projection），表示只顯示 phoneNumber 欄位，並隱藏預設的 \_id 欄位。

4 REST Methods

練習 A：REST Methods 基礎

基礎題

1.請問 GET、POST、PUT、DELETE 這四個 HTTP 方法分別用來做什麼？

Question 1 Answer:

GET： 用於從伺服器獲取資源。 此操作是安全的，不會修改伺服器上的任何數據，就像資料庫的 SELECT 操作一樣。 參數通常通過URL 的查詢字串傳遞。

POST： 用於向伺服器提交數據，通常用於創建新資源。 這類似於資料庫的 INSERT 操作。 資料通過請求體（request body）傳遞，因此比GET 更安全，因為參數不會顯示在URL 中。

PUT： 用於更新或創建指定的資源。 如果資源已存在，則進行更新（類似資料庫的 UPDATE 操作）；如果不存在，則創建新資源。

DELETE： 用於從伺服器上刪除指定的資源。 這類似於資料庫的 DELETE 操作。

2.當我們使用 POST 方法時，資料應該放在哪裡？（A）URL 查詢參數 （B）請求主體 （C）路徑參數 （D）Headers

Question 2 Answer: B

3.在 Express 中，如果要解析 JSON 格式的資料，需要使用哪一個中介軟體？

Question 3 Answer:

如何使用 express.json():

在你的Express 應用程式中，於路由定義之前，先用 app.use() 來掛載此中介軟體。

使用 app.post() 或 app.put() 等方法來接收客戶端發送的POST/PUT 請求。

通過 req.body 來存取解析後的JSON 數據。

4.請問 req.params 和 req.query 有什麼區別？

Question 4 Answer:

req.params

用途：: 從URL 的「路徑部分」中獲取動態的參數。

範例URL: http://localhost:3000/user/123

路由定義: /user/:id

存取方式: req.params.id 會得到 '123'。

req.query

用途：: 從URL 中 ? 後面的查詢字串獲取參數。

範例URL: http://localhost:3000/users?name=John&age=30

路由定義: /users

存取方式:

req.query.name 會得到 'John'。

req.query.age 會得到 '30'。

5.當我們用 POST 成功建立新資料時，應該回傳哪一個 HTTP 狀態碼？（A）200 （B）201 （C）204 （D）404

Question 5 Answer: B-201

當使用 POST 方法成功建立新資料時，應回傳的 HTTP 狀態碼是 (B) 201 Created。這是因為 201 狀態碼明確表示伺服器已成功建立新的資源。

(A) 200 OK：表示請求成功處理，但通常用於已存在的資源獲取、更新或執行其他非建立資源的操作。

(B) 201 Created：表示請求成功，並在伺服器上建立了一個新的資源，這正是 POST 方法的目的。

(C) 204 No Content：表示請求已成功處理，但回應中不包含任何內容，常見於成功刪除資源後。

(D) 404 Not Found：表示請求的資源在伺服器上找不到，是一個客戶端錯誤狀態碼。

判斷題

6.（對 / 錯）DELETE 請求通常需要在請求主體中提供要刪除的資料。

Qusetion 6 Answer: 錯

 app.delete('api/people/:id, (req, res) => {

 const {id} = req.params;

 })

7.（對 / 錯）使用 express.urlencoded() 可以解析 HTML 表單資料。

Qusetion 7 Answer: 對

HTML form format --> application/x-www-form-urlencoded

8.（對 / 錯）在使用 await 關鍵字時，函式必須宣告為 async。

Qusetion 8 Answer: 對

9.（對 / 錯）req.params.id 取得的值預設是數字型別。

Qusetion 9 Answer: 對

Debug 題

11.以下程式碼有什麼問題？請指出並修正：

app.post('/api/people', (req, res) => {

  const { name } = req.body;

  if (!name) {

    res.status(400).json({ success: false, msg: '請提供姓名' });

  }

  res.status(201).json({ success: true, person: name });

});

Qestion 11 Answer: No return

12.以下程式碼有什麼問題？請指出並修正：

app.put('/api/people/:id', (req, res) => {

  const { id } = req.params;

  const { name } = req.body;

  const person = people.find((p) => p.id === id);

  person.name = name;

  res.status(200).json({ success: true, data: people });

});

app.put('api/people/:id. (req, res) => {

  const {id} = req.params;

  if (isNaN(id)) {

    return resizeBy,status(400).json({success: false, msg:"ID must be number"});

  }

  //Number(id)

})

Qestion 12 Answer: Number(id) instead of id

進階題

14.請寫一個 Express 路由，實現以下功能：

路徑：GET /api/people/:id

功能：取得指定 ID 的人的資料

如果找不到，回傳 404 錯誤

Question 14 Answer:

app.get('/api/people/:id', (req, res) => {

  const { id } = req.params;

  const person = people.find((p) => p.id === Number(id);

  if (!people){

    return res.status(404).json({success:false, msg:` Cannot find this ${id} person`})

  }

  res.status(200).json({success: true, data: person});

})

  res.status(200).json({ success: true, data: people });

});

如果找到，回傳該人的資料

17.請實現一個 PUT 路由，可以同時更新姓名和年齡。如果只提供其中一個欄位，只更新該欄位；如果兩個都提供，就兩個都更新。

Question 17 Answer:

app.put('/api/people/:id', (req, res) => {

  const { id } = req.params;

  const { name } = req.body;

  const person = people.find((p) => p.id === Number(id);

  if (!people){

    return res.status(404).json({success:false, msg:` Cannot find this ${id} person`})

  }

  if (name) person.name = name;

  if (age) person.age = age;

  res.status(200).json({ success: true, data: person });

});

4.3 Middleware 2

練習 C：內建與第三方 Middleware

9. 填充題

請填寫正確的 middleware：

const express = require('express');

const app = express();

// 解析 JSON 資料

app.use(\_\_\_\_\_\_);  //Answer: app.use(express.json());

// 解析表單資料

app.use(express.\_\_\_\_\_\_({ extended: false })); // Answer: app.use(express.urlencoded({extended:false}));

app.post('/api/users', (req, res) => {

  const { name } = \_\_\_\_\_\_; // 從請求主體取得 name  //Answer: const { name }= await res.json();

  res.send(`歡迎, ${name}`);

});

10. 判斷題

a) express.json() 可以解析 HTML 表單資料。 //Answer : No

b) morgan 是 Express 的內建 middleware。  //Answer: No

c) 如果不使用 express.json()，req.body 會是 undefined。  //Answer: Yes

d) express.urlencoded() 只能處理 GET 請求。  //Answer: No

e) morgan('tiny') 會記錄所有 HTTP 請求。  //Answer: Yes , log all http request

練習 D：GET 方法與路由參數

12. 程式碼補完題

請完成以下程式碼，實作一個書籍 API：

const books = [

  { id: 1, title: 'Node.js 入門', author: '張三' },

  { id: 2, title: 'Express 實戰', author: '李四' },

];

// 取得所有書籍

app.get('/api/books', (req, res) => {

  res.\_\_json\_\_\_\_({ success: true, data: \_\_books\_\_\_\_ });

});

// 取得特定書籍

app.get('/api/books/:id', (req, res) => {

  const { id } = \_\_\_res.params\_\_\_;

  const book = books.find(b => b.id === \_\_Number\_\_\_(id));

  if (!book) {

    return res.status(\_\_\_404\_\_\_).json({ success: false, msg: '找不到該書籍' });

  }

 res.json({ success: true, data: book });

});

13. 進階題（可選）

請實作一個商品搜尋 API，支援以下查詢條件：

category（類別）

minPrice（最低價格）

maxPrice（最高價格）

sort（排序方式：'price-asc' 或 'price-desc'）

const products = [

  { id: 1, name: '手機', price: 8000, category: '電子' },

  { id: 2, name: '電腦', price: 15000, category: '電子' },

  { id: 3, name: '椅子', price: 1500, category: '傢俬' },

];

app.get('/api/search', (req, res) => {

  // 你的程式碼

});

//Answer 13:

const express = require("express");

const app = express();

const morgan = require("morgan");

app.use(express.static(\_\_dirname));

app.use(morgan("tiny"))

const products = [

    {id:1, name: "Phone", price: 8000, category: "Electronic"},

    {id:2, name: "Computer", price: 15000, category: "Electronic"},

    {id:3, name: "Chair", price: 1500, category: "Furniture"},

];

app.get('/api/search', (req, res) => {

    const {category, minPrice, maxPrice, sor} = req.query;

    let results = [...products];

    if(catergory){

        results = results.filter(p => p.category === category);

    }

    if(minPrice) {

        results = results.filter(p => p.price >= Number(minPrice));

    }

    if(maxPrice) {

        results = results.filter(p => p.price <= Number(maxPrice));

    }

    if(sort === "price-asc"){

        results.sort((a,b) => a.price -b.price);

    } else if (sort === "price-desc") {

        results.sort((a,b) => b.price -a.price);

    }

    res.json({success: true, count: results.length, data: results });

})

app.listen(3000, () => {

    console.log("API server is running on port http://localhost:3000");

})

4.1 Middleware 練習

練習 A：Middleware 基礎概念

1. 判斷題：以下說法正確嗎？

a) Middleware 必須在路由處理器之前定義，才會生效。

   正確。 Middleware 會依據在程式碼中定義的順序來執行。

   如果希望middleware 在特定路由處理器之前運行，它必須在該路由定義之前使用 app.use() 或 app.METHOD() 來註冊。

b) 如果 middleware 不送出回應，就一定要呼叫 next()。

正確。 如果middleware 想要將請求繼續傳遞給下一個middleware 或路由處理器，就必須呼叫 next() 函式。

如果沒有呼叫 next()，而又沒有像 res.send() 或 res.end() 這樣結束回應的函式，請求就會被掛起，永不送達下一個處理器。

c) app.use() 只能註冊一個 middleware。

不正確。 app.use() 可以註冊一個或多個middleware，也可以用來註冊一個middleware 陣列。Ex.: app.us(xxx.yyyy,zzzz);

d) Middleware 可以修改 req 和 res 物件。

不正確。 Middleware 可以修改 req 和 res 物件的屬性，但它不能直接改變「物件本身」。

例如，你可以將資料附加到 req 上，但不能將 req 的屬性改成一個全新的物件。

e) 路由參數（req.params）和查詢字串（req.query）的值都是數字格式。

不正確。 req.params 和 req.query 裡的屬性值預設都是字串格式。

在使用時需要手動進行轉換才能使用數字運算。

2. 填充題

請完成以下程式碼，定義一個簡單的 logger middleware：

const logger = (req, res, next) => {

  console.log(`請求方法: ${\_\_\_\_\_\_}`);   //Answer 2: ${req.method}

  console.log(`請求網址: ${\_\_\_\_\_\_}`);  // Answer 2: ${req.url}

  \_\_\_\_\_\_; // 讓請求繼續往下走     // Answer 2: next();

};

app.use(\_\_\_\_\_\_);    //Answer 2: app.use(logger);

3. Debug 題

以下程式碼有問題，請找出錯誤並說明原因：

Answer 3:

const express = require("express");

const app = express();

const checkAuth = (req, res, next) => {

  const { user } = req.query;

  if (user !== 'john') {

    return res.status(401).send('未授權');

  }

  req.user = { name: 'John' };

  next();

    // if (user ==='john'){

    //     req.user = {name:'John'};

    //     next();

    // } else {

    //     res.status(401).send("Not auth");

    // }

};

app.use(checkAuth);

app.get('/', (req, res) => {

  res.send(`歡迎, ${req.user.name}`);

});

app.listen(3000, () =>{

    console.log( "Server is runing at http://localhost:3000");

});

4. 選擇題

以下哪個是正確的 middleware 定義？

A)

const middleware = (req, res) => {

  console.log('Hello');

  next();

};

B)

const middleware = (req, res, next) => {

  console.log('Hello');

};

C)

const middleware = (req, res, next) => {

  console.log('Hello');

  next();

};

D)

const middleware = () => {

  console.log('Hello');

  next();

};

Answer 4 : C

5. 實作題

請撰寫一個 middleware，檢查請求的 query string 中是否有 apiKey 參數，且值為 '12345'。如果沒有或不正確，回傳 403 錯誤；如果正確，讓請求繼續。

const checkApiKey = (req, res, next) => {

  // 你的程式碼

};

/Answer 5:

const express = require("express");

const app = express();

const checkApiKey = (req, res, next) => {

  const {apiKey} = req.query;

    if (apiKey === "12345") {

      next();

    } else {

      res.status(403).send("Not Valid API key");

    }

  }

app.use(checkApiKey);

app.get('/', (req, res) => {

  res.send("歡迎");

});

app.listen(3000, () => {

  console.log("Server is runing at http://localhost:3000");

});

練習 B：串連多個 Middleware

6. 程式碼閱讀題

請閱讀以下程式碼，回答問題：

const logger = (req, res, next) => {

  console.log('Logger');

  next();

};

const authorize = (req, res, next) => {

  console.log('Authorize');

  res.send('授權檢查完成');

};

app.use(logger);

app.get('/', authorize, (req, res) => {

  res.send('首頁');

});

app.listen(3000, () => {

    console.log("Server is runing at http://localhost:3000");

});

問題：6a) 當訪問 / 時，終端機會顯示什麼？

    Answer 6a :  Logger

              Authorize

 6b) 瀏覽器會收到什麼回應？

    Answer 6b: 授權檢查完成

 6c) 為什麼路由處理器中的 res.send('首頁') 沒有執行？

    Answer 6c: 授權檢查完成, 沒有執行('首頁')

7. 排序題

以下是一個 Express 應用程式的程式碼片段（順序打亂）。請重新排列，使其正確執行：

A. app.listen(5000);

B. app.use(logger);

C. const logger = (req, res, next) => { console.log('Log'); next(); };

D. app.get('/', (req, res) => res.send('首頁'));

E. const express = require('express');

F. const app = express();

正確順序: \_\_\_\_\_\_

Answer 7: E, F, C, B, D, A

8. 進階題（可選）

請撰寫兩個 middleware：

addTimestamp: 在 req 物件上加上當前時間戳記（req.timestamp）

logRequest: 在console記錄 req.method, req.url, 和 req.timestamp

然後在一個路由中使用這兩個 middleware。

const addTimestamp = (req, res, next) => {

  // 你的程式碼

};

const logRequest = (req, res, next) => {

  // 你的程式碼

};

app.get('/', addTimestamp, logRequest, (req, res) => {

  res.send('完成');

});

Answer 8:

const express = require("express");

const app = express();

const addTimestamp = (req, res, next) => {

    req.timestamp = new Date().toISOString();

    next();

};

const logRequest = (req, res, next) => {

    console.log(`${req.method} ${req.url} ${req.timestamp}}`);

    next();

}

app.get ("/", addTimestamp, logRequest, (req, res) => {

    res.send('完成');

});

app.listen(3000, () => {

    console.log("Running at http://localhost:3000");

});

4.1練習

練習 A：HTTP Request/Response Cycle 基礎練習

基礎題：

請解釋 HTTP 請求與回應循環的基本概念，並舉出一個日常生活的例子來比喻。

HTTP 請求與回應循環是網際網路通訊的基礎，其概念是「客戶端」發出一個「請求」到「伺服器」，「伺服器」處理後再回傳一個「回應」給「客戶端」。

日常生活中，這就像是您在「雜貨店」（伺服器），「打電話」（發送請求）給店員（伺服器），請他幫您找「一瓶可樂」（您要的資源），店員找到後「打電話」（發送回應）回覆您說「可樂在冰箱裡」（回應內容），您再掛斷電話（循環結束）。

在 Node.js 中，req 和 res 物件分別代表什麼？它們的主要作用是什麼？

在Node.js 中，req 物件代表「請求」，是客戶端發出的要求；res 物件代表「回應」，是用於回傳給客戶端的資料。

req 的主要作用是讓伺服器可以從中獲取請求的資訊，如URL、請求方法和HTTP標頭等，而 res 的主要作用則是讓伺服器能夠建構並結束回應，例如回傳文字、HTML 或JSON 資料

什麼是端口號？為什麼在啟動伺服器時需要指定端口號？

連接埠（Port）是用來區分同一台主機上不同網路服務的識別碼。 雖然每台設備都有唯一的IP 位址，但一台伺服器往往同時提供多種網路服務，如網頁瀏覽、郵件處理、檔案傳輸等，這時就需要透過連接埠來分辨不同的服務。 舉例來說： IP 位址是「地址」：用來找出哪一台主機。

判斷題（對或錯）：

 HTTP 標頭必須在發送回應內容之前設定。（對）

 Express.js 的 res.json() 方法會自動設定正確的 Content-Type。（對）

Debug 題：

找出以下 Express.js 程式碼的錯誤：

const express = require('express');

const app = express();

app.use(express.json());

app.get('/', (req, res) => {

    res.send('Hello World');

    res.json({message: 'Another response'});

});

app.listen(3000);

app.listen(3000, () =>{

    console.log( "Server is running on port http://localhost:3000");

});

Show Hello Word only

res.json({message: 'Another response'}): This line attempts to send another response, this time a JSON object, to the same client request.

However, since the response has already been sent and closed by res.send(), this second attempt will fail.

 以下 JSON 格式有什麼問題？

{

    name: "John",

    age: 30

    hobbies: ['reading', 'swimming'],

    sayHello: function() { return "Hello"; }

}

Json format : "name" and no function & array inside json

進階題：

 設計一個 Express.js 應用程式，提供以下三個路由：

GET / - 返回歡迎訊息

GET /about - 返回公司資訊

GET /api/time - 返回當前時間的 JSON 格式

const express = require("express");

const app = express();

app.get('/', (req, res) => {

    res.send(`<h1>Welcome to Homepage </h1>`);

});

app.get('/about', (req, res) => {

    res.send(`<!DOCTYPE html>

            <html lang="zh-hk">

            <head>

                <meta charset="UTF-8">

                <title> Web Title </title>

            </head>

            <body>

                <p>About us</p><br/>

                <a href="/">Return to Company Information</a><br/>

            </body>

            </html>`);

});

app.get('/contact', (req, res) => {

    res.send("Contact Us");

});

app.get("/api/time", (req, res) => {

    res.json({

        Current\_time: `${new Date().toLocaleString()}`,

            timezone : 'UTC'

    });

});

app.use((req, res)=>{

    res.status(404).send("Page not found");

})

app.listen(3000, () =>{

    console.log( "Server is running on port http://localhost:3000");

});

解釋為什麼現代 Web 應用程式傾向於使用 JSON API 而不是直接返回 HTML 頁面？

現代 Web 應用程式傾向於使用 JSON API 而不是直接返回 HTML 頁面，主要是因為這種做法可以將前端（使用者介面）和後端（資料與業務邏輯）分離開來，帶來更高的靈活性、效率和效能。

以下是使用 JSON API 的幾個主要優勢：

1. 前後端分離與多平台支援

「無頭」架構（Headless Architecture）：JSON API 促成了「無頭」架構的發展，即後端只負責提供資料，不關心前端的呈現方式。這使得開發者可以將內容獨立於呈現方式進行管理。

多裝置與跨平台支援：後端只需要提供一組 API，就能支援多種前端應用程式，例如：

網頁應用程式（Web Apps）

iOS 和 Android 行動應用程式（Mobile Apps）

智慧手錶或智慧電視等物聯網裝置

桌面應用程式

一致的使用體驗：這確保了無論使用者在哪個平台上瀏覽，都能獲得一致的資料來源和體驗，並避免了為不同平台重複開發後端邏輯的冗餘工作

2. 提升效能與使用者體驗

單頁應用程式（SPA）：JSON API 是建構單頁應用程式的基礎。在單頁應用程式中，瀏覽器只在第一次載入時下載一個 HTML 頁面，之後的所有互動都透過非同步請求（如 AJAX）向伺服器取得 JSON 資料，並由瀏覽器上的 JavaScript 動態更新頁面內容。

減少資料傳輸量：與直接傳輸包含大量標籤、樣式和指令碼的完整 HTML 相比，JSON 是一種更輕量級的資料格式。這能減少網路傳輸的頻寬，加快資料載入速度，尤其對行動裝置使用者而言至關重要。

更流暢的互動：因為不必在每次操作時都重新載入整個頁面，使用者介面能夠保持順暢，提供類似原生應用程式的無縫體驗。

3. 加速開發與維護

前後端獨立開發：前後端團隊可以各自獨立開發，並使用 JSON API 作為溝通介面。前端開發者可以專注於使用者介面設計和體驗，後端開發者則專注於業務邏輯和資料處理。

技術棧自由：開發團隊可以自由選擇最適合各自需求的技術。前端可以使用 React、Vue 或 Angular 等框架，後端則可以使用 Node.js、Python 或 Go 等不同語言和框架。

更好的擴展性：將應用程式的各部分解耦，可以更容易地獨立擴展後端服務，以應對高流量或複雜業務邏輯的需求。

4. 易於理解與通用性

人類可讀、機器可解析：JSON 格式簡潔、易於閱讀和撰寫，同時也便於程式語言解析。

語言中立：JSON 是一種獨立於語言的資料格式，幾乎所有現代程式語言都支援。這使得來自不同系統的應用程式可以輕鬆地互相通訊。

標準化：JSON 是 Web API 資料交換的通用標準，這使得與第三方服務的整合變得更加容易。

如何在 Express.js 中處理 404 錯誤（頁面不存在）？請寫出程式碼。clear

app.use((req, res)=>{

    res.status(404).send("Page not found");

})

app.listen(3000, () =>{

    console.log( "Server is running on port http://localhost:3000");

});

3.2 練習

moment-main.js 練習題

練習 A：Moment.js 基礎操作

基礎題

1. 顯示明天的日期（格式：YYYY-MM-DD）

const moment = require("moment");

const tomorrow = moment().add(1, "day");

console.log(" 顯示明天的日期(格式:YYYY-MM-DD)", tomorrow.format("YYYY-MM-DD"));

顯示昨天的時間（格式：HH:mm:ss）

const moment = require("moment");

const  yesterdayTime = moment().subtract(1, "day");

console.log("顯示昨天的時間(格式:HH:mm:ss): ", yesterdayTime.format('YYYY-MM-DD HH:mm:ss'));

// 計算 50 天後是什麼日期

const moment = require("moment");

const dayAfter50 = moment().add(50, "days");

console.log("計算 50 天後是什麼日期: ", dayAfter50.format("YYYY-MM-DD"));

or console.log(moment().add(50, "days").format("YYYY-MM-DD")));

// 顯示當前時間的 24 小時制格式（格式：HH:mm）

const moment = require("moment");

console.log("顯示當前時間的 24 小時制格式(格式:HH:mm): ", now.format("HH:mm"));

or console.log(moment().format("HH:mm")));

// 計算三個月前的日期

const moment = require("moment");

const monthBefore3 = moment().subtract(3, "months");

console.log("計算三個月前的日期: ", monthBefore3.format("YYYY-MM\_DD"));

or console.log(moment().subtract(3, "months").format("YYYY-MM-DD")));

判斷題

moment().subtract(1, "month") 會計算上個月的日期 （是/否）

是

moment().format("dddd") 會顯示星期幾的英文全名 （是/否）

否

Moment.js 可以直接比較兩個日期的大小 （是/否）

是

console.log(moment().isBefore(tomorrow));

Debug 題

找出以下程式碼的錯誤：

const moment = require("moment");

const nextMonth = moment().add(2, "month");

console.log(nextMonth.format("YYYY-MM-DD"));

修正以下程式碼：

const startDate = moment("2025-01-01");

const endDate = moment("2025-12-31");

const duration = endDate.diff(startDate, "days");

console.log("天數差:", duration);

進階題

計算今年還剩多少天

const moment = require("moment");

const startDate = moment("2025-09-26");

const endDate = moment("2025-9-30");

const duration = endDate.diff(startDate, "days");

console.log("顯示當前月份有多少天", duration);

or

const moment = require("moment");

const endOfYear = moment().endOf("year");

const daysLeft = endOfYear.diff(moment(), "days");

console.log(daysLeft);

建立一個函數，判斷給定日期是否為週末

const moment = require("moment");

function isWeekend(dateString){

    const date = moment(dataString);

    const dayOfWeek = date.day();

    if(datOfWeek === 0 || dayOfWeek === 6) {

        return true;

    } else {

        return false;

    }

}

console.log(moment().daysInMonth());

顯示當前月份有多少天

const moment = require("moment");

const startDayOfMonth= moment("2025-09-01");

const endDayOfMonth = moment("2025-9-30");

const duration = endDayOfMonth.diff(startDayOfMonth, "days");

console.log("Start of Month: ", startDayOfMonth.format("YYYY-MM-DD"));

console.log("End of Month: ", endDayOfMonth.format("YYYY-MM-DD"));

console.log("顯示當前月份有多少天: ", duration, 'days');

計算兩個生日之間相差多少個月

const moment = require("moment");

const birthday1= moment("2025-04-06");

const birthday2 = moment("2025-11-28");

const duration = birthday2.diff(birthday1, "days");

console.log("First Birthday: ", birthday1.format("YYYY-MM-DD"));

console.log("Second Birthday: ", birthday2.format("YYYY-MM-DD"));

console.log("計算兩個生日之間相差多少個月: ", duration, 'days');

or

const moment = require("moment");

const bday1 = moment("1990-05-15");

const bday2 = moment("1992-08-20");

console.log(bday2.diff(bday1, "months"));

實際應用題

建立一個會議提醒系統，計算距離會議開始還有多少小時

function meetingReminder(meetingTime) {

  your code here

  return workDays;

}

meetingReminder("2025-09-26 15:30");

const moment = require("moment");

function meetingReminder(meetingTime) {

    const meeting = moment(meetingTime);

    const hoursLeft = meeting.diff(moment(), "hours");

    if(hoursLeft > 0) {

        console.log(`meeting left time ${hoursLeft} hour`);

    } else if (hoursLeft === 0) {

        console.log (`Meeting has been starting`);

    } else {

        console.log(`Meeting finished`)

    }

}

meetingReminder("2025-10-02 15:30");

設計一個員工請假系統，計算請假的工作日數（不包括週末）

function calculateWorkDays(startDate, endDate) {

// your code here

  return workDays;

}

console.log(calculateWorkDays("2025-09-22", "2025-09-26")); // 5 (週一到週五)

function calculateWorkDays(startDate, endDate) {

    const start = moment(startDate);

    const end = moment(endDate);

    let workDays = 0;

    let current = start.clone();

    while(current.isSameOrBefore(end)){

        if(current.day() !==0 && current.day() !==6) {

            workDays++;

        }

        current.add(1, "day");

    }

    return workDays;

}

console.log(calculateWorkDays2("2025-10-02", "2025-10-07"));

or

function calculateWorkDays2(startDate, endDate){

    const start = moment(startDate);

    const end = moment(endDate);

    let workDays = 0;

    const totalDays = end.diff(start, "days") + 1;

    for (let i = 0; i < totalDays; i++) {

        const day = start.clone().add(i, "days");

        if(day.day() !==0 && day.day() !==6){

             workDays++;

        }

    }

    return workDays;

}

console.log(calculateWorkDays2("2025-10-02", "2025-10-07"));

建立一個生日提醒器，顯示朋友的下一個生日是幾天後

function nextBirthday(birthDate) {

// your code here

}

console.log(nextBirthday("1990-12-25")); // 距離聖誕節生日還有 X 天

function nextBirthday(birthDate){

    const today = moment();

    const thisYear = today.year();

    let nextBday = moment(birthDate).year(thisYear);

    if(nextBday.isBefore(today)) {

        nextBday.add(1, "year");

    }

    const dayLeft = nextBday.diff(today, "days") + 1;

    return (dayLeft);

}

console.log(nextBirthday("1990-09-03"));

練習 B：Path 模組操作

基礎題

從路徑 "C:\Users\Student\Desktop\homework.docx" 取得檔案名稱（不含副檔名）

const path = require('path');

const filePath = "/Users/Student/Desktop/homework.docx";

const fileName = path.basename(filePath);

console.log("檔案名稱: ",fileName);     // homework.doc

console.log("檔案名稱（不含副檔名):", path.basename(filePath, path.extname(filePath)));  //homework

檢查檔案 "image.jpg" 的副檔名是否為圖片格式 (提示：includes)

const path = require('path');

const imageFormats = [".jpg", ".jpeg", ".png", ".gif", ".bmp"];

const fileName2 = "image.png";

const ext2 = path.extname(fileName2).toLowerCase();

const isImage = imageFormats.includes(ext2);

console.log("IsImage?", isImage);   // true

建立路徑：將 "photos"、"2025"、"vacation.jpg" 組合成完整路徑

const path = require('path');

const fullPath = path.join('photos', '2025', 'vacation.jpg');

console.log(fullPath);  // photos/2025/vacation.jpg

從 "/var/www/html/index.html" 取得目錄名稱

const path = require('path');

const fullPath = "/var/www/html/index.html";

const folderPath = path.dirname(fullPath);

console.log(folderPath); //   /var/www/html

other example

const path = require('path');

const dirPath = path.dirname("/var/www/html/index.html");

console.log("目錄名稱:", dirPath);   /var/www/html

將相對路徑 "docs/readme.txt" 轉換為絕對路徑

const path = require('path');

const absolutePath = path.resolve("docs/readme.txt");

console.log("絕對路徑:", absolutePath); //絕對路徑: /Users/gtcroom6/Desktop/Henry\_Backend/Day 9 (26-9-2025)/docs/readme.txt

Debug 題

找出錯誤並修正：

const path = require("path");

const fileName = path.basename("/home/user/document.pdf", ".pdf");

console.log("檔名:", fileName.name);

修正以下程式碼：

const fullPath = path.join("C:", "/Users", "/Documents", "file.txt");

console.log(fullPath);

進階題

建立一個函數，判斷檔案是否為 Office 文件（.doc, .docx, .xls, .xlsx, .ppt, .pptx）

const path = require('path');

function isOfficeFile(fileName) {

    const ext = path.extname(fileName).toLowerCase();

    const officeFormats = [".doc", ".docx", ".xls", ".xlsx", ".ppt", ".pptx"];

    return officeFormats.includes(ext);

}

console.log(isOfficeFile("test.jpg"));   //false

比較兩個檔案路徑是否指向同一個目錄

const path = require('path');

function isSameDirectory(path1, path2) {

    const dir1 = path.dirname(path.resolve(path1));

    const dir2 = path.dirname(path.resolve(path2));

    return dir1 === dir2;

}

console.log(isSameDirectory("./docs/file1.txt", "./docs/v1/file2.txt"));  //false

練習 C：檔案系統操作

基礎題

建立一個新檔案 "shopping-list.txt"，內容為 "牛奶\n麵包\n雞蛋"

const fs = require("fs");

const content = "Milk\nBread\nEgg";

fs.writeFile("shopping-list.txt", content, (err)=>{

    if(err) {

        console.log("fail: ", err);

        return;

    }

    console.log("file write done")

})

讀取檔案 "config.txt" 並顯示檔案大小（字元數）

const fs = require("fs");

fs.readFile("config.txt", "utf8",(err, data)=>{

    if(err) {

        console.log("fail:", err);

        return;

    }

    console.log("File content:", data);

    console.log("File size", data.length, "words");

    });

將新內容附加到現有檔案 "log.txt" 的末尾

const fs = require("fs");

const newContent = "\n [" + new Date().toISOString() + " ] New Project";

fs.appendFile("log.txt", newContent,(err)=>{

    if(err){

        console.log("Fail:", err);

        return;

    }

    console.log("append content");

})

檢查檔案 "backup.txt" 是否存在（提示：fs.access, fs.constants.F\_OK）

const fs = require("fs");

fs.access("backup.txt", fs.constants.F\_OK,(err)=>{

    if(err) {

        console.log("file does not exit");

        return;

    } else {

        console.log("file exit");

    }

})

複製檔案 "original.txt" 到 "copy.txt" （提示：copyFile）

const fs = require("fs");

fs.copyFile("orginal.txt", "copy.txt", (err) => {

    if(err) {

        console.log("fail:", err);

        return;

    }

    console.log("file copy sucess");

});

判斷題

fs.appendFile() 會覆蓋原有檔案內容 （是/否）

否

fs.stat() 可以取得檔案的詳細資訊 （是/否）

是

Example

const fs = require("fs");

fs.stat("orginal.txt", (err, stats) => {

    if(err) {

        console.log("fail:", err);

        return;

    }

    console.log("is File?", stats.isFile());

    console.log("is Directory?", stats.isDirectory());

    console.log("is Size", stats.size);

    console.log("is BirthTime?", stats.birthtime);

    console.log("is LastWrite Time?", stats.mtime);

});

讀取不存在的檔案會導致程式崩潰 （是/否）

否

Debug 題

找出錯誤：

const fs = require("fs");

fs.readFile("data.txt", "utf8", (err, data) => {

  if (!err) {

    console.log("內容:", data);

  }

});

進階題

建立一個函數，計算文字檔案的行數（提示：\n）

const fs = require("fs");

function countLines(fileName){

    fs.readFile(fileName, "utf8", (err, data) => {

        if(err) {

            console.log("fail:", err);

            return;

        }

        const Lines = data.split('\n').length;

        console.log(`total ${Lines} line`);

    });

}

countLines("log.txt");

實作一個簡單的檔案搜尋功能，在檔案中搜尋特定文字

const fs = require("fs");

function hasTextInFile(fileName, searchText) {

    fs.readFile(fileName, "utf8", (err, data) => {

        if(err) {

            console.log("fail:", err);

            return;

        }

        if(data.includes(searchText)) {

            console.log("found");

        } else {

            console.log("Not found");

        }

    });

}

hasTextInFile("log.txt", "New Project");

建立一個函數，取得檔案的建立時間和修改時間

const fs = require("fs");

function getFileTimestamps(fileName){

    fs.stat(fileName, (err, stats) => {

        if(err) {

            console.log("fail:", err);

            return;

        }

        console.log("Set Time:", stats.birthtime);

        console.log("Edit time:", stats.mtime);

    });

}

getFileTimestamps("log.txt");

練習 D：事件處理基礎

基礎題

建立一個 "doorbell" 事件，當有人按門鈴時顯示訊息

const EventEmitter = require("events");

const houseEmitter = new EventEmitter();

houseEmitter.on("doorbell", () => {

    console.log("有人按門鈴");

});

houseEmitter.emit("doorbell");

// 建立 "alarm" 事件，可以接收警報類型和時間參數

houseEmitter.emit("alarm", "Flooding", "2025-10-03 09:17");

// 為同一個 "notification" 事件建立三個不同的處理器

houseEmitter.on("notification", (message) => {

    console.log(`Inform 1:${message}`);

});

houseEmitter.on("notification", (message) => {

    console.log(`Inform 2:${message}`);

});

houseEmitter.on("notification", (message) => {

    console.log(`Inform 3:${message}`);

});

houseEmitter.emit("notification", "New Inform");

// 建立一個 "login" 事件，記錄使用者登入時間

const authEmitter = new EventEmitter();

authEmitter.on("login", (username)=> {

    const loginTime = new Date().toISOString();

    console.log(`${loginTime} user ${username} login`);

});

authEmitter.emit("login", "john");

// 建立一個 "error" 事件處理器，顯示錯誤訊息和時間

const appEmitter = new EventEmitter();

appEmitter.on("error", (errorType, errorMessage)=>{

    console.log(`${new Date().toISOString()}

        [ERROR] ${errorType}: ${errorMessage}`);

});

appEmitter.emit("notExist", "NETWORK", "Excess time");

判斷題

EventEmitter 可以同時處理多個不同的事件類型 （是/否）

是

事件監聽器執行的順序是隨機的 （是/否）

否

emit() 方法會回傳布林值表示是否有監聽器處理該事件 （是/否）

否

Debug 題

找出錯誤：

const EventEmitter = require("events");

const emitter = new EventEmitter();  //add new

emitter.on("message", (text) => {

  console.log("收到訊息:", text);

});

emitter.emit("message", "Hello");

修正以下程式碼：

const myEmitter = new EventEmitter();

myEmitter.on("calculate", (a, b) => {

  console.log("結果:", a + b);

});

myEmitter.emit("calculate", "5", "3");  // 5 , 3

練習 E：HTTP 伺服器建立

基礎題

建立一個伺服器，在 "/welcome" 路徑顯示 "歡迎光臨我的網站"

const http = require("http");

const server = http.createServer((req, res)=>{

    if(req.url === "/welcome"){

        res.statusCode = 200;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end("Welcome my homepage");

    } else {

        res.statusCode = 404;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end("Page not exist");

    }

});

server.listen(3000,()=>{

    console.log("Server started at http://localhost:3000");

})

建立 "/date" 路徑，顯示今天是星期幾（提示：moment.locale）

const moment = require("moment");

const http = require("http");

const server = http.createServer((req, res)=>{

    if(req.url === "/welcome"){

        res.statusCode = 200;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end("Welcome my homepage");

    } else if(req.url === "/date"){

        const today = new Date();

        const weekdays = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"];

        const todayWeekday = weekdays[today.getDay(1)];

        res.status = 200;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end(`Today is ${todayWeekday}`);   //Today is Friday

        console.log(`frome moment: `, moment().format("dddd"));

    } else {

        res.statusCode = 404;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end("Page not exist");

    }

});

server.listen(3000,()=>{

    console.log("Server started at http://localhost:3000");

})

建立 "/random" 路徑，顯示 1-100 之間的隨機數字

const moment = require("moment");

const http = require("http");

const server = http.createServer((req, res)=>{

    if(req.url === "/welcome"){

        res.statusCode = 200;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end("Welcome my homepage");

    } else if(req.url === "/date"){

        const today = new Date();

        const weekdays = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"];

        const todayWeekday = weekdays[today.getDay(1)];

        res.status = 200;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end(`Today is ${todayWeekday}`);   //Today is Friday

        console.log(`frome moment: `, moment().format("dddd"));

    } else if(req.url === "/random"){

         res.status = 200;

         res.setHeader("Content-Type", "text/plain; charset=utf8");

         const randomNum = Math.floor(Math.random() \* 100) + 1;

         res.end(`Random Number: ${randomNum}`);

    } else {

        res.statusCode = 404;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end("Page not exist");

    }

});

server.listen(3000,()=>{

    console.log("Server started at http://localhost:3000");

})

在首頁 ("/") 顯示一個簡單的 HTML 清單，包含所有可用的路徑

const moment = require("moment");

const http = require("http");

const server = http.createServer((req, res)=>{

    if(req.url === "/") {

        res.statusCode = 200;

        res.setHeader("Content-Type", "text/html; charset=utf8");

        res.end(`

            <!DOCTYPE html>

            <html lang="zh-hk">

            <head>

                <meta charset="UTF-8">

                <title> Web Title </title>

            </head>

            <body>

                <a href="/welcome">Welcome page</a><br/>

                <a href="/date">Today day</a><br/>

                <a href="/random">Random Number</a><br/>

            </body>

            </html>

            `)

    } else if(req.url === "/welcome"){

        res.statusCode = 200;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end("Welcome my homepage");

    } else if(req.url === "/date"){

        const today = new Date();

        const weekdays = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"];

        const todayWeekday = weekdays[today.getDay(1)];

        res.status = 200;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end(`Today is ${todayWeekday}`);   //Today is Friday

        console.log(`frome moment: `, moment().format("dddd"));

    } else if(req.url === "/random"){

        res.status = 200;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        const randomNum = Math.floor(Math.random() \* 100) + 1;

        res.end(`Random Number: ${randomNum}`);

    } else {

        res.statusCode = 404;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end("Page not exist");

    }

});

server.listen(3000,()=>{

    console.log("Server started at http://localhost:3000");

})

OR

const http = require("http");

const hostname = "127.0.0.1";

const port = 3000;

const server = http.createServer((req, res) => {

  res.statusCode = 200;

  res.setHeader("Content-Type", "text/plain; charset=utf-8");

  if (req.url === "/") {

    res.setHeader("Content-Type", "text/plain; charset=utf-8");

    res.end("歡迎來到我的網站");

    console.log("有訪客");

  } else if (req.url === "/about") {

    res.setHeader("Content-Type", "text/html; charset=utf-8");

    res.end(`

            <!DOCTYPE html>

            <html lang="zh-Hant">

            <head>

                <meta charset="UTF-8">

                <title>關於我們</title>

            </head>

            <body>

                <h1>關於我們</h1>

                <p>這是一個使用 <strong>Node.js</strong> 建立的簡單網站。</p>

                <p>我們專注於教學與分享程式知識。</p>

                <a href="/">回到首頁</a>

            </body>

            </html>

        `);

    // const htmlContent = "<html></html>";

    // res.end(httpContent);

  } else if (req.url === "/contact") {

    res.setHeader("Content-Type", "text/plain; charset=utf-8");

    res.end("聯絡我們");

  } else if (req.url === "/time") {

    res.setHeader("Content-Type", "text/plain; charset=utf-8");

    const now = new Date().toLocaleString();

    res.end(`目前時間：${now}`);

  } else {

    res.setHeader("Content-Type", "text/plain; charset=utf-8");

    res.statusCode = 404;

    res.end("找不到你要的頁面!");

  }

});

server.listen(port, hostname, () => {

  console.log(`伺服器運行在 http://${hostname}:${port}`);

});

判斷題

HTTP 伺服器可以在同一個埠號上處理多個不同的路徑 （是/否）

是，HTTP 伺服器可以在同一個埠號（例如 80）上處理多個不同的路徑。

說明

HTTP 協定本身是基於「請求路徑」（例如 /index.html、/api/users）來區分不同的服務內容。

例如，http://example.com/ 與 http://example.com/about 會同時使用埠號 80，但路徑不同，HTTP 伺服器會根據路徑分別處理與回應對應內容。

這是任何標準 Web 伺服器（如 IIS、Apache、Nginx）都具備的功能，也是網站運作的基本原理。

因此，答案是「是」。一個 HTTP 伺服器完全可以在同一個埠號上同時處理多個不同的路徑。

req.method 可以取得 HTTP 請求方法（GET, POST 等） （是/否）

是的，可以。req.method 能取得 HTTP 請求的方法，如 GET、POST、PUT、DELETE 等。

用法說明

在 Express 或其他 Node.js HTTP 服務中，req.method 是 request 物件的屬性。

它的值是一個字串，代表客戶端所發出的請求方法，例如 "GET" 或 "POST"。

程式範例

javascript

app.use((req, res, next) => {

  console.log(req.method); // 例如 GET、POST

  next();

});

這可以讓伺服器根據不同 HTTP 請求方法作不同處理，是後端開發的基本操作。

不設定 res.statusCode 預設會是 200 （是/否）

是，預設會是 200。當不特別設定 res.statusCode，伺服器回應會自動使用 HTTP 狀態碼 200 代表「OK」。

說明

在 Node.js 或 Express 建立 HTTP 回應時，若沒有自訂狀態碼，res.statusCode 預設值就是 200。

200 是標準的成功狀態碼，表示伺服器已成功處理請求。

此行為在原生 Node.js HTTP 模組與 Express 框架中皆適用。

Debug 題

找出錯誤：

const http = require("http");

const server = http.createServer((req, res) => {

  res.writeHead(200);       //res.setHeader("Content-Type", "text/plain; charset=utf8");

  res.end("Hello World");

});

server.listen(3000);   //console.log("Server started at http://localhost:3000");

修正中文顯示問題：

const server = http.createServer((req, res) => {

  if (req.url === "/chinese") {

    res.writeHead(200, { "Content-Type": "text/html" });   //charset=utf-8

    res.end("<h1>歡迎來到中文網站</h1>");

  }

});

練習 F：作業系統資訊

基礎題

顯示電腦的 CPU 核心數量

const os = require('os');

const cpus = os.cpus();

console.log(`Number of CPU Cores: ${cpus.length}`);

or

const os = require('os');

console.log("CPU Core Number:", os.cpus().length);

計算記憶體使用率（已使用/總容量 \* 100%）

const os = require("os");

const total = (os.totalmem() /1024 /1024 /1024).toFixed(2);

const used = (os.freemem()/1024 /1024 /1024).toFixed(2);

const usage = (used/ total) \* 100;

console.log("記憶體使用率:", usage, "%");

or

const os = require("os");

const totalMem = os.totalmem();

const freeMem = os.freemem();

const usedMem = totalMem - freeMem;

const usagePercent = ((usedMem/totalMem)\*100).toFixed(2);

console.log("Memory use:",usagePercent + "%");

顯示作業系統的版本資訊

const os = require("os");

console.log("OS type:", os.type());

console.log("Operating system", os.release());

console.log("Platform: ", os.platform());

console.log("Arc : ", os.arch());

取得目前登入使用者的使用者名稱

const os = require("os");

const userInfo = os.userInfo();

console.log(`Current User: ${userInfo.username}`);

or

console.log("User name:", OscillatorNode.userInfo().username);

顯示系統的暫存目錄路徑

const os = require("os");

console.log(`Home Directory: ${os.homedir()}`);

or

console.log("temp:", OscillatorNode.tmpdir());

判斷題

os.networkInterfaces(); // 可以取得網路卡資訊 （是/否）

是

const os = require('os');

console.log("network Card Info:", OscillatorNode.networkInterfaces());

// Get network interfaces information

const networkInterfaces = os.networkInterfaces();

console.log('Network Interfaces:');

// Iterate over each network interface

Object.entries(networkInterfaces).forEach(([name, addresses]) => {

  console.log(`\nInterface: ${name}`);

  addresses.forEach((address) => {

    console.log(`- Family: ${address.family}`);

    console.log(` Address: ${address.address}`);

    console.log(` Netmask: ${address.netmask}`);

    console.log(` MAC: ${address.mac || 'N/A'}`);

    console.log(` Internal: ${address.internal}`);

  });

});

// Example: Find the first non-internal IPv4 address

function getLocalIpAddress() {

  const interfaces = os.networkInterfaces();

  for (const name of Object.keys(interfaces)) {

    for (const iface of interfaces[name]) {

      if (iface.family === 'IPv4' && !iface.internal) {

        return iface.address;

      }

    }

  }

  return '127.0.0.1'; // Fallback to localhost

}

const localIp = getLocalIpAddress();

console.log(`\nLocal IP Address: ${localIp}`);

Debug 題

修正記憶體計算錯誤：

const os = require("os");

const total = (os.totalmem() /1024 /1024 /1024).toFixed(2);

const used = (os.freemem()/1024 /1024 /1024).toFixed(2);

// const usedMem = ((os.totalmem() /1024 /1024 /1024).toFixed(2)) - ((os.freemem()/1024 /1024 /1024).toFixed(2));

const usage = (used/ total) \* 100;

console.log("記憶體使用率:", usage.toFixed(2), "%");  //add toFixed(2) 28.63%

找出錯誤：

const os = require("os");

const cpus = os.cpus();

console.log(`CPU 核心數: ${cpus.length}`);   //not count use length

練習 G：URL 解析處理

基礎題

解析 URL "https://shop.example.com/products/electronics?brand=sony&price=500" 並取得商品分類

從查詢參數中取得品牌和價格資訊

檢查 URL 是否使用 HTTPS 協定

建立一個新的 URL，加入 "discount=10" 查詢參數

從 URL 中移除所有查詢參數，只保留基本路徑

const { parse } = require("path");

const url = require("url");

const myUrl = "https://shop.example.com/products/electronics?brand=sony&price=500";

const parsedUrl = url.parse(myUrl, true);

const splitterUrlPath = parsedUrl.pathname.split("/");

console.log(splitterUrlPath[splitterUrlPath.length-1]);

console.log("Brand name:", parsedUrl.query.brand); //並取得商品分類

console.log("Price:", parsedUrl.query.price);    // 從查詢參數中取得品牌和價格資訊

const isHTTPS = parsedUrl.protocol === "https:";

console.log("Is it HTTPS: ", isHTTPS);         //檢查 URL 是否使用 HTTPS 協定

console.log(parsedUrl.search);  //?nrand=sony&price=500

console.log(parsedUrl.query);   //[object: null prototype]

parsedUrl.query.discount = "10";

parsedUrl.search = null;             //delete search

console.log("New URL: ", url.format(parsedUrl));

delete parsedUrl.search;

delete parsedUrl.query;

console.log("Delete all query:", url.format(parsedUrl));

判斷題

URL 物件的 search 屬性包含問號 （是/否）

是

searchParams.has() 可以檢查是否存在指定的查詢參數 （是/否）

是

const { parse } = require("path");

const url = require("url");

const myUrl2 = new URL("https://shop.example.com/products/electronics?brand=sony&price=500");

console.log(myUrl2.searchParams.has("brand")).  //true

url.resolve() 可以合併基礎 URL 和相對路徑 （是/否）

是

console.log(URL.resolve("http://example.com", "/one"));  //http://example.com/one

Debug 題

修正程式碼：

const url = require("url");

const Url = "https://example.com/search?q=nodejs";

// const params = url.parsedUrl.query;

const parsedUrl2 = url.parse("https://shop.example.com/products/electronics?brand=sony&price=500", false);

// console.log("搜尋關鍵字:", params.q);

console.log(parsedUrl2.search);

console.log(parsedUrl2.query);

找出錯誤：

const myURL = new URL("https://example.com/api");

myURL.searchParams.add("token", "abc123");

myURL.searchParams.add("token", "xyz789");

console.log("所有 token:", myURL.searchParams.get("token"));

const url = require("url");

const myUrl3 = new URL("https://example.com/api");

myUrl3.searchParams.append("token", "abc123");

myUrl3.searchParams.append("token", "xyz789");

console.log("所有 token:", myUrl3.searchParams.getAll("token"));

console.log(myUrl3);                //New API (new URL found query but nothing value)

2.2.3練習

練習A：檔案系統基礎操作

基礎題

寫一個程式，使用fs.readFile()讀取名為data.txt的檔案內容並顯示在console。

fs.readFile 是一個非同步（asynchronous）方法：

if (typeof require !== 'undefined'){

    const fs = require('fs');

    // const path = require('path');

    try{

        const data = fs.readFileSync('data1.txt', 'utf8');

        console.log("file data:", data);

    } catch(error){

        console.log("file retrive failed", error.message);

    }

}

fs.readFile 是一個同步方法：

const fs = require('fs');

fs.readFile('data.txt', 'utf8', (error, data) => {

    if(error){

        console.log('file retieve failed', error.message);

        return;

    }

    console.log("Content", data);

});

建立一個程式，將字串「這是測試內容」寫入名為output.txt的檔案。

fs.writeFile 是一個非同步（asynchronous）方法：

if (typeof require !== 'undefined'){

    const fs = require('fs');

    const path = require('path');

    try{

        const data = fs.readFileSync('data.txt', 'utf8');

        console.log("file data:", data);

    } catch(error){

        console.log("write error", error.message);

    }

    const content = '這是測試內容';

    fs.writeFile('output.txt', content, 'utf8', (error) =>{

        if(error){

            console.log('write error', error.message);

            return;

        }

        console.log("Write sucessful");

    });

}

fs.writeFile 是一個同步方法：

const fs = require('fs');

const content =  '這是測試內容';

fs.writeFile('output.txt', content, 'utf8', (error) =>{

    if(error){

        console.log('write error', error.message);

        return;

    }

    console.log("Write sucessful");

});

寫一個程式顯示當前檔案的完整路徑和所在資料夾。

const fs = require('fs');

const path = require('path');

console.log('Directory: ', \_\_dirname);

console.log('File: ', \_\_filename);

const filePath = path.join(\_\_dirname, 'data', 'users.json');

console.log('full directory: ', filePath);

判斷題

4. 以下程式碼是否正確？請說明原因。

const fs = require('fs');

const data = fs.readFileSync('test.txt','utf8');

console.log(data);

這段程式碼是不正確的，因為 fs.readFile 是一個非同步（asynchronous）方法，需要傳入 callback 函數來處理讀取結果。原始寫法會直接得到 undefined，也不會正確顯示檔案內容

fs.readFile 和 fs.readFileSync 的差異在於前者是非同步、後者是同步。fs.readFile 需要傳入 callback，檔案讀取時不會阻塞程式繼續執行；fs.readFileSync 則會阻塞後續程式，直到全部讀取完成並直接回傳內容

if (typeof require !== 'undefined'){

    const fs = require('fs');

    try{

        const data = fs.readFileSync('test.txt', 'utf8');

        console.log("file data:", data);

    } catch(error){

        console.log("file retrive failed", error.message);

    }

}

\_\_dirname和\_\_filename有什麼差別？

\_\_dirname：代表目前檔案所在的資料夾

\_\_filename：代表目前檔案的完整路徑（包含檔名）

Debug題

6. 找出並修正以下程式碼的問題：missing utf-8

fs = require('fs');

fs.readFile('config.txt','utf8', (error, data) => {

    console.log(data);

});

以下程式碼為什麼會出錯？如何修正？

console.log(process.version);   //v22.16.0

console.log(window.location);   //window is not defined

process.version 屬於 Node.js 的 API，只能在 Node.js 執行環境使用。

window.location 屬於瀏覽器的 API，只能在瀏覽器的 JavaScript 環境中存取。

在 Node.js 執行時，沒有 window 這個全域物件，所以會拋出「ReferenceError: window is not defined」錯誤

請根據執行環境，分開使用：

如果在 Node.js 環境，只能使用 process 相關 API：

javascript

console.log(process.version); // 正確

如果在瀏覽器環境，只能使用 window 相關 API：

javascript

console.log(window.location); // 正確

如果需要判斷執行環境：

javascript

if (typeof window !== 'undefined') {

  // 瀏覽器環境

  console.log(window.location);

} else if (typeof process !== 'undefined') {

  // Node.js 環境

  console.log(process.version);

}

這樣可以根據目前環境正確執行程式。

練習B：HTTP伺服器建立

基礎題

16. 建立一個最簡單的HTTP伺服器，在瀏覽器訪問時顯示「Hello Node.js」。

const http = require("http");

const server = http.createServer((request, response) => {

    response.writeHead(200, {

        'Content-Type': 'text/plain; charset=utf-8'

    });

    if (request.url === '/') {

        response.end("Hello Node.js");

    } else if (request.url === '/about') {

        response.end("About");

    } else {

        response.writeHead(404);

        response.end('Not found page');

    }

});

const PORT = 3000;

server.listen(PORT, () => {

    console.log(`Server http:/localhost:${PORT}`);

});

17. 修改上面的伺服器，使其能夠處理首頁（/）和關於頁面（/about）兩個路徑。

18. 建立一個伺服器，能夠顯示當前時間。

const http = require("http");

const server = http.createServer((request, response) => {

    response.writeHead(200, {

        'Content-Type': 'text/html; charset=utf-8'

    });

    if (request.url === '/') {

        response.end(`

          <!DOCTYPE html>

            <html lang="zh-hk">

            <head>

                <meta charset="UTF-8">

                <title> Web Title </title>

            </head>

            <body>

            <p id="time"></p>

            <script>

              function updateTime() {

                let now = new Date();

                let timeString = now.toLocaleTimeString();

                document.getElementById('time').innerHTML = timeString;

              }

              setInterval(updateTime, 1000); // Updates every second

            </script>

            </body>

            </html>

          `);

    } else if (request.url === '/about') {

        response.end("About");

    } else {

        response.writeHead(404);

        response.end('Not found page');

    }

});

const PORT = 3000;

server.listen(PORT, () => {

    console.log(`Server http:/localhost:${PORT}`);

});

OR

const http = require("http");

const hostname = "127.0.0.1";

const port = 3000;

const server = http.createServer((req, res) => {

  res.statusCode = 200;

  res.setHeader("Content-Type", "text/plain; charset=utf-8");

  if (req.url === "/") {

    res.setHeader("Content-Type", "text/plain; charset=utf-8");

    res.end("歡迎來到我的網站");

    console.log("有訪客");

  } else if (req.url === "/about") {

    res.setHeader("Content-Type", "text/html; charset=utf-8");

    res.end(`

            <!DOCTYPE html>

            <html lang="zh-Hant">

            <head>

                <meta charset="UTF-8">

                <title>關於我們</title>

            </head>

            <body>

                <h1>關於我們</h1>

                <p>這是一個使用 <strong>Node.js</strong> 建立的簡單網站。</p>

                <p>我們專注於教學與分享程式知識。</p>

                <a href="/">回到首頁</a>

            </body>

            </html>

        `);

    // const htmlContent = "<html></html>";

    // res.end(httpContent);

  } else if (req.url === "/contact") {

    res.setHeader("Content-Type", "text/plain; charset=utf-8");

    res.end("聯絡我們");

  } else if (req.url === "/time") {

    res.setHeader("Content-Type", "text/plain; charset=utf-8");

    const now = new Date().toLocaleString();

    res.end(`目前時間：${now}`);

  } else {

    res.setHeader("Content-Type", "text/plain; charset=utf-8");

    res.statusCode = 404;

    res.end("找不到你要的頁面!");

  }

});

server.listen(port, hostname, () => {

  console.log(`伺服器運行在 http://${hostname}:${port}`);

});

判斷題

19. 以下程式碼執行後會發生什麼？

const http = require('http');

const server = http.createServer((req, res) => {

  res.writeHead(200);

  res.write('Hello');

  res.end();

});

//  server.listen(3000); port number not declare

const PORT = 3000;

server.listen(PORT, () => {

    console.log(`Server http:/localhost:${PORT}`);

});

在Node.js中，require('http')和瀏覽器中的fetch()有什麼差別？

Node.js 的 require('http') 是一个底层的HTTP 模块，用于创建服务器或发送HTTP 请求，它提供了对HTTP 协议的详细控制。

而浏览器中的 fetch() 是一个现代的、基于Promise 的网络请求API，用于在客户端发起网络请求，操作更简洁、易用，并且是浏览器环境的标准功能。

3.2 練習

moment-main.js 練習題

練習 A：Moment.js 基礎操作

基礎題

1. 顯示明天的日期（格式：YYYY-MM-DD）

const moment = require("moment");

const tomorrow = moment().add(1, "day");

console.log(" 顯示明天的日期(格式:YYYY-MM-DD)", tomorrow.format("YYYY-MM-DD"));

顯示昨天的時間（格式：HH:mm:ss）

const moment = require("moment");

const  yesterdayTime = moment().subtract(1, "day");

console.log("顯示昨天的時間(格式:HH:mm:ss): ", yesterdayTime.format('YYYY-MM-DD HH:mm:ss'));

// 計算 50 天後是什麼日期

const moment = require("moment");

const dayAfter50 = moment().add(50, "days");

console.log("計算 50 天後是什麼日期: ", dayAfter50.format("YYYY-MM-DD"));

or console.log(moment().add(50, "days").format("YYYY-MM-DD")));

// 顯示當前時間的 24 小時制格式（格式：HH:mm）

const moment = require("moment");

console.log("顯示當前時間的 24 小時制格式(格式:HH:mm): ", now.format("HH:mm"));

or console.log(moment().format("HH:mm")));

// 計算三個月前的日期

const moment = require("moment");

const monthBefore3 = moment().subtract(3, "months");

console.log("計算三個月前的日期: ", monthBefore3.format("YYYY-MM\_DD"));

or console.log(moment().subtract(3, "months").format("YYYY-MM-DD")));

判斷題

moment().subtract(1, "month") 會計算上個月的日期 （是/否）

是

moment().format("dddd") 會顯示星期幾的英文全名 （是/否）

否

Moment.js 可以直接比較兩個日期的大小 （是/否）

是

console.log(moment().isBefore(tomorrow));

Debug 題

找出以下程式碼的錯誤：

const moment = require("moment");

const nextMonth = moment().add(2, "month");

console.log(nextMonth.format("YYYY-MM-DD"));

修正以下程式碼：

const startDate = moment("2025-01-01");

const endDate = moment("2025-12-31");

const duration = endDate.diff(startDate, "days");

console.log("天數差:", duration);

進階題

計算今年還剩多少天

const moment = require("moment");

const startDate = moment("2025-09-26");

const endDate = moment("2025-9-30");

const duration = endDate.diff(startDate, "days");

console.log("顯示當前月份有多少天", duration);

or

const moment = require("moment");

const endOfYear = moment().endOf("year");

const daysLeft = endOfYear.diff(moment(), "days");

console.log(daysLeft);

建立一個函數，判斷給定日期是否為週末

const moment = require("moment");

function isWeekend(dateString){

    const date = moment(dataString);

    const dayOfWeek = date.day();

    if(datOfWeek === 0 || dayOfWeek === 6) {

        return true;

    } else {

        return false;

    }

}

console.log(moment().daysInMonth());

顯示當前月份有多少天

const moment = require("moment");

const startDayOfMonth= moment("2025-09-01");

const endDayOfMonth = moment("2025-9-30");

const duration = endDayOfMonth.diff(startDayOfMonth, "days");

console.log("Start of Month: ", startDayOfMonth.format("YYYY-MM-DD"));

console.log("End of Month: ", endDayOfMonth.format("YYYY-MM-DD"));

console.log("顯示當前月份有多少天: ", duration, 'days');

計算兩個生日之間相差多少個月

const moment = require("moment");

const birthday1= moment("2025-04-06");

const birthday2 = moment("2025-11-28");

const duration = birthday2.diff(birthday1, "days");

console.log("First Birthday: ", birthday1.format("YYYY-MM-DD"));

console.log("Second Birthday: ", birthday2.format("YYYY-MM-DD"));

console.log("計算兩個生日之間相差多少個月: ", duration, 'days');

or

const moment = require("moment");

const bday1 = moment("1990-05-15");

const bday2 = moment("1992-08-20");

console.log(bday2.diff(bday1, "months"));

實際應用題

建立一個會議提醒系統，計算距離會議開始還有多少小時

function meetingReminder(meetingTime) {

  your code here

  return workDays;

}

meetingReminder("2025-09-26 15:30");

const moment = require("moment");

function meetingReminder(meetingTime) {

    const meeting = moment(meetingTime);

    const hoursLeft = meeting.diff(moment(), "hours");

    if(hoursLeft > 0) {

        console.log(`meeting left time ${hoursLeft} hour`);

    } else if (hoursLeft === 0) {

        console.log (`Meeting has been starting`);

    } else {

        console.log(`Meeting finished`)

    }

}

meetingReminder("2025-10-02 15:30");

設計一個員工請假系統，計算請假的工作日數（不包括週末）

function calculateWorkDays(startDate, endDate) {

// your code here

  return workDays;

}

console.log(calculateWorkDays("2025-09-22", "2025-09-26")); // 5 (週一到週五)

function calculateWorkDays(startDate, endDate) {

    const start = moment(startDate);

    const end = moment(endDate);

    let workDays = 0;

    let current = start.clone();

    while(current.isSameOrBefore(end)){

        if(current.day() !==0 && current.day() !==6) {

            workDays++;

        }

        current.add(1, "day");

    }

    return workDays;

}

console.log(calculateWorkDays2("2025-10-02", "2025-10-07"));

or

function calculateWorkDays2(startDate, endDate){

    const start = moment(startDate);

    const end = moment(endDate);

    let workDays = 0;

    const totalDays = end.diff(start, "days") + 1;

    for (let i = 0; i < totalDays; i++) {

        const day = start.clone().add(i, "days");

        if(day.day() !==0 && day.day() !==6){

             workDays++;

        }

    }

    return workDays;

}

console.log(calculateWorkDays2("2025-10-02", "2025-10-07"));

建立一個生日提醒器，顯示朋友的下一個生日是幾天後

function nextBirthday(birthDate) {

// your code here

}

console.log(nextBirthday("1990-12-25")); // 距離聖誕節生日還有 X 天

function nextBirthday(birthDate){

    const today = moment();

    const thisYear = today.year();

    let nextBday = moment(birthDate).year(thisYear);

    if(nextBday.isBefore(today)) {

        nextBday.add(1, "year");

    }

    const dayLeft = nextBday.diff(today, "days") + 1;

    return (dayLeft);

}

console.log(nextBirthday("1990-09-03"));

練習 B：Path 模組操作

基礎題

從路徑 "C:\Users\Student\Desktop\homework.docx" 取得檔案名稱（不含副檔名）

const path = require('path');

const filePath = "/Users/Student/Desktop/homework.docx";

const fileName = path.basename(filePath);

console.log("檔案名稱: ",fileName);     // homework.doc

console.log("檔案名稱（不含副檔名):", path.basename(filePath, path.extname(filePath)));  //homework

檢查檔案 "image.jpg" 的副檔名是否為圖片格式 (提示：includes)

const path = require('path');

const imageFormats = [".jpg", ".jpeg", ".png", ".gif", ".bmp"];

const fileName2 = "image.png";

const ext2 = path.extname(fileName2).toLowerCase();

const isImage = imageFormats.includes(ext2);

console.log("IsImage?", isImage);   // true

建立路徑：將 "photos"、"2025"、"vacation.jpg" 組合成完整路徑

const path = require('path');

const fullPath = path.join('photos', '2025', 'vacation.jpg');

console.log(fullPath);  // photos/2025/vacation.jpg

從 "/var/www/html/index.html" 取得目錄名稱

const path = require('path');

const fullPath = "/var/www/html/index.html";

const folderPath = path.dirname(fullPath);

console.log(folderPath); //   /var/www/html

other example

const path = require('path');

const dirPath = path.dirname("/var/www/html/index.html");

console.log("目錄名稱:", dirPath);   /var/www/html

將相對路徑 "docs/readme.txt" 轉換為絕對路徑

const path = require('path');

const absolutePath = path.resolve("docs/readme.txt");

console.log("絕對路徑:", absolutePath); //絕對路徑: /Users/gtcroom6/Desktop/Henry\_Backend/Day 9 (26-9-2025)/docs/readme.txt

Debug 題

找出錯誤並修正：

const path = require("path");

const fileName = path.basename("/home/user/document.pdf", ".pdf");

console.log("檔名:", fileName.name);

修正以下程式碼：

const fullPath = path.join("C:", "/Users", "/Documents", "file.txt");

console.log(fullPath);

進階題

建立一個函數，判斷檔案是否為 Office 文件（.doc, .docx, .xls, .xlsx, .ppt, .pptx）

const path = require('path');

function isOfficeFile(fileName) {

    const ext = path.extname(fileName).toLowerCase();

    const officeFormats = [".doc", ".docx", ".xls", ".xlsx", ".ppt", ".pptx"];

    return officeFormats.includes(ext);

}

console.log(isOfficeFile("test.jpg"));   //false

比較兩個檔案路徑是否指向同一個目錄

const path = require('path');

function isSameDirectory(path1, path2) {

    const dir1 = path.dirname(path.resolve(path1));

    const dir2 = path.dirname(path.resolve(path2));

    return dir1 === dir2;

}

console.log(isSameDirectory("./docs/file1.txt", "./docs/v1/file2.txt"));  //false

練習 C：檔案系統操作

基礎題

建立一個新檔案 "shopping-list.txt"，內容為 "牛奶\n麵包\n雞蛋"

const fs = require("fs");

const content = "Milk\nBread\nEgg";

fs.writeFile("shopping-list.txt", content, (err)=>{

    if(err) {

        console.log("fail: ", err);

        return;

    }

    console.log("file write done")

})

讀取檔案 "config.txt" 並顯示檔案大小（字元數）

const fs = require("fs");

fs.readFile("config.txt", "utf8",(err, data)=>{

    if(err) {

        console.log("fail:", err);

        return;

    }

    console.log("File content:", data);

    console.log("File size", data.length, "words");

    });

將新內容附加到現有檔案 "log.txt" 的末尾

const fs = require("fs");

const newContent = "\n [" + new Date().toISOString() + " ] New Project";

fs.appendFile("log.txt", newContent,(err)=>{

    if(err){

        console.log("Fail:", err);

        return;

    }

    console.log("append content");

})

檢查檔案 "backup.txt" 是否存在（提示：fs.access, fs.constants.F\_OK）

const fs = require("fs");

fs.access("backup.txt", fs.constants.F\_OK,(err)=>{

    if(err) {

        console.log("file does not exit");

        return;

    } else {

        console.log("file exit");

    }

})

複製檔案 "original.txt" 到 "copy.txt" （提示：copyFile）

const fs = require("fs");

fs.copyFile("orginal.txt", "copy.txt", (err) => {

    if(err) {

        console.log("fail:", err);

        return;

    }

    console.log("file copy sucess");

});

判斷題

fs.appendFile() 會覆蓋原有檔案內容 （是/否）

否

fs.stat() 可以取得檔案的詳細資訊 （是/否）

是

Example

const fs = require("fs");

fs.stat("orginal.txt", (err, stats) => {

    if(err) {

        console.log("fail:", err);

        return;

    }

    console.log("is File?", stats.isFile());

    console.log("is Directory?", stats.isDirectory());

    console.log("is Size", stats.size);

    console.log("is BirthTime?", stats.birthtime);

    console.log("is LastWrite Time?", stats.mtime);

});

讀取不存在的檔案會導致程式崩潰 （是/否）

否

Debug 題

找出錯誤：

const fs = require("fs");

fs.readFile("data.txt", "utf8", (err, data) => {

  if (!err) {

    console.log("內容:", data);

  }

});

進階題

建立一個函數，計算文字檔案的行數（提示：\n）

const fs = require("fs");

function countLines(fileName){

    fs.readFile(fileName, "utf8", (err, data) => {

        if(err) {

            console.log("fail:", err);

            return;

        }

        const Lines = data.split('\n').length;

        console.log(`total ${Lines} line`);

    });

}

countLines("log.txt");

實作一個簡單的檔案搜尋功能，在檔案中搜尋特定文字

const fs = require("fs");

function hasTextInFile(fileName, searchText) {

    fs.readFile(fileName, "utf8", (err, data) => {

        if(err) {

            console.log("fail:", err);

            return;

        }

        if(data.includes(searchText)) {

            console.log("found");

        } else {

            console.log("Not found");

        }

    });

}

hasTextInFile("log.txt", "New Project");

建立一個函數，取得檔案的建立時間和修改時間

const fs = require("fs");

function getFileTimestamps(fileName){

    fs.stat(fileName, (err, stats) => {

        if(err) {

            console.log("fail:", err);

            return;

        }

        console.log("Set Time:", stats.birthtime);

        console.log("Edit time:", stats.mtime);

    });

}

getFileTimestamps("log.txt");

練習 D：事件處理基礎

基礎題

建立一個 "doorbell" 事件，當有人按門鈴時顯示訊息

const EventEmitter = require("events");

const houseEmitter = new EventEmitter();

houseEmitter.on("doorbell", () => {

    console.log("有人按門鈴");

});

houseEmitter.emit("doorbell");

// 建立 "alarm" 事件，可以接收警報類型和時間參數

houseEmitter.emit("alarm", "Flooding", "2025-10-03 09:17");

// 為同一個 "notification" 事件建立三個不同的處理器

houseEmitter.on("notification", (message) => {

    console.log(`Inform 1:${message}`);

});

houseEmitter.on("notification", (message) => {

    console.log(`Inform 2:${message}`);

});

houseEmitter.on("notification", (message) => {

    console.log(`Inform 3:${message}`);

});

houseEmitter.emit("notification", "New Inform");

// 建立一個 "login" 事件，記錄使用者登入時間

const authEmitter = new EventEmitter();

authEmitter.on("login", (username)=> {

    const loginTime = new Date().toISOString();

    console.log(`${loginTime} user ${username} login`);

});

authEmitter.emit("login", "john");

// 建立一個 "error" 事件處理器，顯示錯誤訊息和時間

const appEmitter = new EventEmitter();

appEmitter.on("error", (errorType, errorMessage)=>{

    console.log(`${new Date().toISOString()}

        [ERROR] ${errorType}: ${errorMessage}`);

});

appEmitter.emit("notExist", "NETWORK", "Excess time");

判斷題

EventEmitter 可以同時處理多個不同的事件類型 （是/否）

是

事件監聽器執行的順序是隨機的 （是/否）

否

emit() 方法會回傳布林值表示是否有監聽器處理該事件 （是/否）

否

Debug 題

找出錯誤：

const EventEmitter = require("events");

const emitter = new EventEmitter();  //add new

emitter.on("message", (text) => {

  console.log("收到訊息:", text);

});

emitter.emit("message", "Hello");

修正以下程式碼：

const myEmitter = new EventEmitter();

myEmitter.on("calculate", (a, b) => {

  console.log("結果:", a + b);

});

myEmitter.emit("calculate", "5", "3");  // 5 , 3

練習 E：HTTP 伺服器建立

基礎題

建立一個伺服器，在 "/welcome" 路徑顯示 "歡迎光臨我的網站"

const http = require("http");

const server = http.createServer((req, res)=>{

    if(req.url === "/welcome"){

        res.statusCode = 200;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end("Welcome my homepage");

    } else {

        res.statusCode = 404;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end("Page not exist");

    }

});

server.listen(3000,()=>{

    console.log("Server started at http://localhost:3000");

})

建立 "/date" 路徑，顯示今天是星期幾（提示：moment.locale）

const moment = require("moment");

const http = require("http");

const server = http.createServer((req, res)=>{

    if(req.url === "/welcome"){

        res.statusCode = 200;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end("Welcome my homepage");

    } else if(req.url === "/date"){

        const today = new Date();

        const weekdays = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"];

        const todayWeekday = weekdays[today.getDay(1)];

        res.status = 200;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end(`Today is ${todayWeekday}`);   //Today is Friday

        console.log(`frome moment: `, moment().format("dddd"));

    } else {

        res.statusCode = 404;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end("Page not exist");

    }

});

server.listen(3000,()=>{

    console.log("Server started at http://localhost:3000");

})

建立 "/random" 路徑，顯示 1-100 之間的隨機數字

const moment = require("moment");

const http = require("http");

const server = http.createServer((req, res)=>{

    if(req.url === "/welcome"){

        res.statusCode = 200;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end("Welcome my homepage");

    } else if(req.url === "/date"){

        const today = new Date();

        const weekdays = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"];

        const todayWeekday = weekdays[today.getDay(1)];

        res.status = 200;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end(`Today is ${todayWeekday}`);   //Today is Friday

        console.log(`frome moment: `, moment().format("dddd"));

    } else if(req.url === "/random"){

         res.status = 200;

         res.setHeader("Content-Type", "text/plain; charset=utf8");

         const randomNum = Math.floor(Math.random() \* 100) + 1;

         res.end(`Random Number: ${randomNum}`);

    } else {

        res.statusCode = 404;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end("Page not exist");

    }

});

server.listen(3000,()=>{

    console.log("Server started at http://localhost:3000");

})

在首頁 ("/") 顯示一個簡單的 HTML 清單，包含所有可用的路徑

const moment = require("moment");

const http = require("http");

const server = http.createServer((req, res)=>{

    if(req.url === "/") {

        res.statusCode = 200;

        res.setHeader("Content-Type", "text/html; charset=utf8");

        res.end(`

            <!DOCTYPE html>

            <html lang="zh-hk">

            <head>

                <meta charset="UTF-8">

                <title> Web Title </title>

            </head>

            <body>

                <a href="/welcome">Welcome page</a><br/>

                <a href="/date">Today day</a><br/>

                <a href="/random">Random Number</a><br/>

            </body>

            </html>

            `)

    } else if(req.url === "/welcome"){

        res.statusCode = 200;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end("Welcome my homepage");

    } else if(req.url === "/date"){

        const today = new Date();

        const weekdays = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"];

        const todayWeekday = weekdays[today.getDay(1)];

        res.status = 200;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end(`Today is ${todayWeekday}`);   //Today is Friday

        console.log(`frome moment: `, moment().format("dddd"));

    } else if(req.url === "/random"){

        res.status = 200;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        const randomNum = Math.floor(Math.random() \* 100) + 1;

        res.end(`Random Number: ${randomNum}`);

    } else {

        res.statusCode = 404;

        res.setHeader("Content-Type", "text/plain; charset=utf8");

        res.end("Page not exist");

    }

});

server.listen(3000,()=>{

    console.log("Server started at http://localhost:3000");

})

OR

const http = require("http");

const hostname = "127.0.0.1";

const port = 3000;

const server = http.createServer((req, res) => {

  res.statusCode = 200;

  res.setHeader("Content-Type", "text/plain; charset=utf-8");

  if (req.url === "/") {

    res.setHeader("Content-Type", "text/plain; charset=utf-8");

    res.end("歡迎來到我的網站");

    console.log("有訪客");

  } else if (req.url === "/about") {

    res.setHeader("Content-Type", "text/html; charset=utf-8");

    res.end(`

            <!DOCTYPE html>

            <html lang="zh-Hant">

            <head>

                <meta charset="UTF-8">

                <title>關於我們</title>

            </head>

            <body>

                <h1>關於我們</h1>

                <p>這是一個使用 <strong>Node.js</strong> 建立的簡單網站。</p>

                <p>我們專注於教學與分享程式知識。</p>

                <a href="/">回到首頁</a>

            </body>

            </html>

        `);

    // const htmlContent = "<html></html>";

    // res.end(httpContent);

  } else if (req.url === "/contact") {

    res.setHeader("Content-Type", "text/plain; charset=utf-8");

    res.end("聯絡我們");

  } else if (req.url === "/time") {

    res.setHeader("Content-Type", "text/plain; charset=utf-8");

    const now = new Date().toLocaleString();

    res.end(`目前時間：${now}`);

  } else {

    res.setHeader("Content-Type", "text/plain; charset=utf-8");

    res.statusCode = 404;

    res.end("找不到你要的頁面!");

  }

});

server.listen(port, hostname, () => {

  console.log(`伺服器運行在 http://${hostname}:${port}`);

});

判斷題

HTTP 伺服器可以在同一個埠號上處理多個不同的路徑 （是/否）

是，HTTP 伺服器可以在同一個埠號（例如 80）上處理多個不同的路徑。

說明

HTTP 協定本身是基於「請求路徑」（例如 /index.html、/api/users）來區分不同的服務內容。

例如，http://example.com/ 與 http://example.com/about 會同時使用埠號 80，但路徑不同，HTTP 伺服器會根據路徑分別處理與回應對應內容。

這是任何標準 Web 伺服器（如 IIS、Apache、Nginx）都具備的功能，也是網站運作的基本原理。

因此，答案是「是」。一個 HTTP 伺服器完全可以在同一個埠號上同時處理多個不同的路徑。

req.method 可以取得 HTTP 請求方法（GET, POST 等） （是/否）

是的，可以。req.method 能取得 HTTP 請求的方法，如 GET、POST、PUT、DELETE 等。

用法說明

在 Express 或其他 Node.js HTTP 服務中，req.method 是 request 物件的屬性。

它的值是一個字串，代表客戶端所發出的請求方法，例如 "GET" 或 "POST"。

程式範例

javascript

app.use((req, res, next) => {

  console.log(req.method); // 例如 GET、POST

  next();

});

這可以讓伺服器根據不同 HTTP 請求方法作不同處理，是後端開發的基本操作。

不設定 res.statusCode 預設會是 200 （是/否）

是，預設會是 200。當不特別設定 res.statusCode，伺服器回應會自動使用 HTTP 狀態碼 200 代表「OK」。

說明

在 Node.js 或 Express 建立 HTTP 回應時，若沒有自訂狀態碼，res.statusCode 預設值就是 200。

200 是標準的成功狀態碼，表示伺服器已成功處理請求。

此行為在原生 Node.js HTTP 模組與 Express 框架中皆適用。

Debug 題

找出錯誤：

const http = require("http");

const server = http.createServer((req, res) => {

  res.writeHead(200);       //res.setHeader("Content-Type", "text/plain; charset=utf8");

  res.end("Hello World");

});

server.listen(3000);   //console.log("Server started at http://localhost:3000");

修正中文顯示問題：

const server = http.createServer((req, res) => {

  if (req.url === "/chinese") {

    res.writeHead(200, { "Content-Type": "text/html" });   //charset=utf-8

    res.end("<h1>歡迎來到中文網站</h1>");

  }

});

練習 F：作業系統資訊

基礎題

顯示電腦的 CPU 核心數量

const os = require('os');

const cpus = os.cpus();

console.log(`Number of CPU Cores: ${cpus.length}`);

or

const os = require('os');

console.log("CPU Core Number:", os.cpus().length);

計算記憶體使用率（已使用/總容量 \* 100%）

const os = require("os");

const total = (os.totalmem() /1024 /1024 /1024).toFixed(2);

const used = (os.freemem()/1024 /1024 /1024).toFixed(2);

const usage = (used/ total) \* 100;

console.log("記憶體使用率:", usage, "%");

or

const os = require("os");

const totalMem = os.totalmem();

const freeMem = os.freemem();

const usedMem = totalMem - freeMem;

const usagePercent = ((usedMem/totalMem)\*100).toFixed(2);

console.log("Memory use:",usagePercent + "%");

顯示作業系統的版本資訊

const os = require("os");

console.log("OS type:", os.type());

console.log("Operating system", os.release());

console.log("Platform: ", os.platform());

console.log("Arc : ", os.arch());

取得目前登入使用者的使用者名稱

const os = require("os");

const userInfo = os.userInfo();

console.log(`Current User: ${userInfo.username}`);

or

console.log("User name:", OscillatorNode.userInfo().username);

顯示系統的暫存目錄路徑

const os = require("os");

console.log(`Home Directory: ${os.homedir()}`);

or

console.log("temp:", OscillatorNode.tmpdir());

判斷題

os.networkInterfaces(); // 可以取得網路卡資訊 （是/否）

是

const os = require('os');

console.log("network Card Info:", OscillatorNode.networkInterfaces());

// Get network interfaces information

const networkInterfaces = os.networkInterfaces();

console.log('Network Interfaces:');

// Iterate over each network interface

Object.entries(networkInterfaces).forEach(([name, addresses]) => {

  console.log(`\nInterface: ${name}`);

  addresses.forEach((address) => {

    console.log(`- Family: ${address.family}`);

    console.log(` Address: ${address.address}`);

    console.log(` Netmask: ${address.netmask}`);

    console.log(` MAC: ${address.mac || 'N/A'}`);

    console.log(` Internal: ${address.internal}`);

  });

});

// Example: Find the first non-internal IPv4 address

function getLocalIpAddress() {

  const interfaces = os.networkInterfaces();

  for (const name of Object.keys(interfaces)) {

    for (const iface of interfaces[name]) {

      if (iface.family === 'IPv4' && !iface.internal) {

        return iface.address;

      }

    }

  }

  return '127.0.0.1'; // Fallback to localhost

}

const localIp = getLocalIpAddress();

console.log(`\nLocal IP Address: ${localIp}`);

Debug 題

修正記憶體計算錯誤：

const os = require("os");

const total = (os.totalmem() /1024 /1024 /1024).toFixed(2);

const used = (os.freemem()/1024 /1024 /1024).toFixed(2);

// const usedMem = ((os.totalmem() /1024 /1024 /1024).toFixed(2)) - ((os.freemem()/1024 /1024 /1024).toFixed(2));

const usage = (used/ total) \* 100;

console.log("記憶體使用率:", usage.toFixed(2), "%");  //add toFixed(2) 28.63%

找出錯誤：

const os = require("os");

const cpus = os.cpus();

console.log(`CPU 核心數: ${cpus.length}`);   //not count use length

練習 G：URL 解析處理

基礎題

解析 URL "https://shop.example.com/products/electronics?brand=sony&price=500" 並取得商品分類

從查詢參數中取得品牌和價格資訊

檢查 URL 是否使用 HTTPS 協定

建立一個新的 URL，加入 "discount=10" 查詢參數

從 URL 中移除所有查詢參數，只保留基本路徑

const { parse } = require("path");

const url = require("url");

const myUrl = "https://shop.example.com/products/electronics?brand=sony&price=500";

const parsedUrl = url.parse(myUrl, true);

const splitterUrlPath = parsedUrl.pathname.split("/");

console.log(splitterUrlPath[splitterUrlPath.length-1]);

console.log("Brand name:", parsedUrl.query.brand); //並取得商品分類

console.log("Price:", parsedUrl.query.price);    // 從查詢參數中取得品牌和價格資訊

const isHTTPS = parsedUrl.protocol === "https:";

console.log("Is it HTTPS: ", isHTTPS);         //檢查 URL 是否使用 HTTPS 協定

console.log(parsedUrl.search);  //?nrand=sony&price=500

console.log(parsedUrl.query);   //[object: null prototype]

parsedUrl.query.discount = "10";

parsedUrl.search = null;             //delete search

console.log("New URL: ", url.format(parsedUrl));

delete parsedUrl.search;

delete parsedUrl.query;

console.log("Delete all query:", url.format(parsedUrl));

判斷題

URL 物件的 search 屬性包含問號 （是/否）

是

searchParams.has() 可以檢查是否存在指定的查詢參數 （是/否）

是

const { parse } = require("path");

const url = require("url");

const myUrl2 = new URL("https://shop.example.com/products/electronics?brand=sony&price=500");

console.log(myUrl2.searchParams.has("brand")).  //true

url.resolve() 可以合併基礎 URL 和相對路徑 （是/否）

是

console.log(URL.resolve("http://example.com", "/one"));  //http://example.com/one

Debug 題

修正程式碼：

const url = require("url");

const Url = "https://example.com/search?q=nodejs";

// const params = url.parsedUrl.query;

const parsedUrl2 = url.parse("https://shop.example.com/products/electronics?brand=sony&price=500", false);

// console.log("搜尋關鍵字:", params.q);

console.log(parsedUrl2.search);

console.log(parsedUrl2.query);

找出錯誤：

const myURL = new URL("https://example.com/api");

myURL.searchParams.add("token", "abc123");

myURL.searchParams.add("token", "xyz789");

console.log("所有 token:", myURL.searchParams.get("token"));

const url = require("url");

const myUrl3 = new URL("https://example.com/api");

myUrl3.searchParams.append("token", "abc123");

myUrl3.searchParams.append("token", "xyz789");

console.log("所有 token:", myUrl3.searchParams.getAll("token"));

console.log(myUrl3);                //New API (new URL found query but nothing value)

2.2.3練習

練習A：檔案系統基礎操作

基礎題

寫一個程式，使用fs.readFile()讀取名為data.txt的檔案內容並顯示在console。

fs.readFile 是一個非同步（asynchronous）方法：

if (typeof require !== 'undefined'){

    const fs = require('fs');

    // const path = require('path');

    try{

        const data = fs.readFileSync('data1.txt', 'utf8');

        console.log("file data:", data);

    } catch(error){

        console.log("file retrive failed", error.message);

    }

}

fs.readFile 是一個同步方法：

const fs = require('fs');

fs.readFile('data.txt', 'utf8', (error, data) => {

    if(error){

        console.log('file retieve failed', error.message);

        return;

    }

    console.log("Content", data);

});

建立一個程式，將字串「這是測試內容」寫入名為output.txt的檔案。

fs.writeFile 是一個非同步（asynchronous）方法：

if (typeof require !== 'undefined'){

    const fs = require('fs');

    const path = require('path');

    try{

        const data = fs.readFileSync('data.txt', 'utf8');

        console.log("file data:", data);

    } catch(error){

        console.log("write error", error.message);

    }

    const content = '這是測試內容';

    fs.writeFile('output.txt', content, 'utf8', (error) =>{

        if(error){

            console.log('write error', error.message);

            return;

        }

        console.log("Write sucessful");

    });

}

fs.writeFile 是一個同步方法：

const fs = require('fs');

const content =  '這是測試內容';

fs.writeFile('output.txt', content, 'utf8', (error) =>{

    if(error){

        console.log('write error', error.message);

        return;

    }

    console.log("Write sucessful");

});

寫一個程式顯示當前檔案的完整路徑和所在資料夾。

const fs = require('fs');

const path = require('path');

console.log('Directory: ', \_\_dirname);

console.log('File: ', \_\_filename);

const filePath = path.join(\_\_dirname, 'data', 'users.json');

console.log('full directory: ', filePath);

判斷題

4. 以下程式碼是否正確？請說明原因。

const fs = require('fs');

const data = fs.readFileSync('test.txt','utf8');

console.log(data);

這段程式碼是不正確的，因為 fs.readFile 是一個非同步（asynchronous）方法，需要傳入 callback 函數來處理讀取結果。原始寫法會直接得到 undefined，也不會正確顯示檔案內容

fs.readFile 和 fs.readFileSync 的差異在於前者是非同步、後者是同步。fs.readFile 需要傳入 callback，檔案讀取時不會阻塞程式繼續執行；fs.readFileSync 則會阻塞後續程式，直到全部讀取完成並直接回傳內容

if (typeof require !== 'undefined'){

    const fs = require('fs');

    try{

        const data = fs.readFileSync('test.txt', 'utf8');

        console.log("file data:", data);

    } catch(error){

        console.log("file retrive failed", error.message);

    }

}

\_\_dirname和\_\_filename有什麼差別？

\_\_dirname：代表目前檔案所在的資料夾

\_\_filename：代表目前檔案的完整路徑（包含檔名）

Debug題

6. 找出並修正以下程式碼的問題：missing utf-8

fs = require('fs');

fs.readFile('config.txt','utf8', (error, data) => {

    console.log(data);

});

以下程式碼為什麼會出錯？如何修正？

console.log(process.version);   //v22.16.0

console.log(window.location);   //window is not defined

process.version 屬於 Node.js 的 API，只能在 Node.js 執行環境使用。

window.location 屬於瀏覽器的 API，只能在瀏覽器的 JavaScript 環境中存取。

在 Node.js 執行時，沒有 window 這個全域物件，所以會拋出「ReferenceError: window is not defined」錯誤

請根據執行環境，分開使用：

如果在 Node.js 環境，只能使用 process 相關 API：

javascript

console.log(process.version); // 正確

如果在瀏覽器環境，只能使用 window 相關 API：

javascript

console.log(window.location); // 正確

如果需要判斷執行環境：

javascript

if (typeof window !== 'undefined') {

  // 瀏覽器環境

  console.log(window.location);

} else if (typeof process !== 'undefined') {

  // Node.js 環境

  console.log(process.version);

}

這樣可以根據目前環境正確執行程式。

練習B：HTTP伺服器建立

基礎題

16. 建立一個最簡單的HTTP伺服器，在瀏覽器訪問時顯示「Hello Node.js」。

const http = require("http");

const server = http.createServer((request, response) => {

    response.writeHead(200, {

        'Content-Type': 'text/plain; charset=utf-8'

    });

    if (request.url === '/') {

        response.end("Hello Node.js");

    } else if (request.url === '/about') {

        response.end("About");

    } else {

        response.writeHead(404);

        response.end('Not found page');

    }

});

const PORT = 3000;

server.listen(PORT, () => {

    console.log(`Server http:/localhost:${PORT}`);

});

17. 修改上面的伺服器，使其能夠處理首頁（/）和關於頁面（/about）兩個路徑。

18. 建立一個伺服器，能夠顯示當前時間。

const http = require("http");

const server = http.createServer((request, response) => {

    response.writeHead(200, {

        'Content-Type': 'text/html; charset=utf-8'

    });

    if (request.url === '/') {

        response.end(`

          <!DOCTYPE html>

            <html lang="zh-hk">

            <head>

                <meta charset="UTF-8">

                <title> Web Title </title>

            </head>

            <body>

            <p id="time"></p>

            <script>

              function updateTime() {

                let now = new Date();

                let timeString = now.toLocaleTimeString();

                document.getElementById('time').innerHTML = timeString;

              }

              setInterval(updateTime, 1000); // Updates every second

            </script>

            </body>

            </html>

          `);

    } else if (request.url === '/about') {

        response.end("About");

    } else {

        response.writeHead(404);

        response.end('Not found page');

    }

});

const PORT = 3000;

server.listen(PORT, () => {

    console.log(`Server http:/localhost:${PORT}`);

});

OR

const http = require("http");

const hostname = "127.0.0.1";

const port = 3000;

const server = http.createServer((req, res) => {

  res.statusCode = 200;

  res.setHeader("Content-Type", "text/plain; charset=utf-8");

  if (req.url === "/") {

    res.setHeader("Content-Type", "text/plain; charset=utf-8");

    res.end("歡迎來到我的網站");

    console.log("有訪客");

  } else if (req.url === "/about") {

    res.setHeader("Content-Type", "text/html; charset=utf-8");

    res.end(`

            <!DOCTYPE html>

            <html lang="zh-Hant">

            <head>

                <meta charset="UTF-8">

                <title>關於我們</title>

            </head>

            <body>

                <h1>關於我們</h1>

                <p>這是一個使用 <strong>Node.js</strong> 建立的簡單網站。</p>

                <p>我們專注於教學與分享程式知識。</p>

                <a href="/">回到首頁</a>

            </body>

            </html>

        `);

    // const htmlContent = "<html></html>";

    // res.end(httpContent);

  } else if (req.url === "/contact") {

    res.setHeader("Content-Type", "text/plain; charset=utf-8");

    res.end("聯絡我們");

  } else if (req.url === "/time") {

    res.setHeader("Content-Type", "text/plain; charset=utf-8");

    const now = new Date().toLocaleString();

    res.end(`目前時間：${now}`);

  } else {

    res.setHeader("Content-Type", "text/plain; charset=utf-8");

    res.statusCode = 404;

    res.end("找不到你要的頁面!");

  }

});

server.listen(port, hostname, () => {

  console.log(`伺服器運行在 http://${hostname}:${port}`);

});

判斷題

19. 以下程式碼執行後會發生什麼？

const http = require('http');

const server = http.createServer((req, res) => {

  res.writeHead(200);

  res.write('Hello');

  res.end();

});

//  server.listen(3000); port number not declare

const PORT = 3000;

server.listen(PORT, () => {

    console.log(`Server http:/localhost:${PORT}`);

});

在Node.js中，require('http')和瀏覽器中的fetch()有什麼差別？

Node.js 的 require('http') 是一个底层的HTTP 模块，用于创建服务器或发送HTTP 请求，它提供了对HTTP 协议的详细控制。

而浏览器中的 fetch() 是一个现代的、基于Promise 的网络请求API，用于在客户端发起网络请求，操作更简洁、易用，并且是浏览器环境的标准功能。

2.1 additional 2 練習

練習 A：this 指向判斷

練習 1

const car = {

  brand: "Toyota",

  showBrand: function() {

    console.log(this.brand);

  }

};

car.showBrand();

const show = car.showBrand;

show();

第一次同第二次呼叫會輸出咩？點解會唔同？

練習 2

const user = {

  name: "Alice",

  greet() {

    console.log("Hello, " + this.name);

  }

};

user.greet();

呢段程式碼會輸出咩？點解？ arrow function not allow this.name

練習 B：計時器應用

練習 4寫一個函數 createTimer(seconds)，實現以下功能：

每秒顯示剩餘時間

時間到咗顯示 "時間到！"

返回一個物件，包含 start() 同 stop() 方法

function createTimer(seconds) {

  let timerInterval = null; // 用於儲存 setInterval 的 ID，方便後續停止

  let remainingTime = seconds; // 儲存剩餘時間

  function updateDisplay() {

    // 顯示剩餘時間

    console.log(`剩餘時間：${remainingTime} 秒`);

  }

  function timerEnd() {

    // 時間到時的處理

    console.log("時間到！");

    clearInterval(timerInterval); // 停止計時器

    remainingTime = seconds; // 重置剩餘時間

  }

  return {

    start: function() {

      // 清除任何現有計時器，避免重複啟動

      if (timerInterval !== null) {

        clearInterval(timerInterval);

      }

      // 立即更新一次顯示

      updateDisplay();

      // 啟動計時器，每秒執行一次

      timerInterval = setInterval(() => {

        remainingTime--; // 減少一秒

        if (remainingTime >= 0) {

          updateDisplay(); // 顯示更新後的剩餘時間

        } else {

          timerEnd(); // 時間到，執行結束邏輯

        }

      }, 1000); // 1000 毫秒 = 1 秒

    },

    stop: function() {

      if (timerInterval !== null) {

        clearInterval(timerInterval); // 清除計時器

        timerInterval = null; // 重置計時器 ID

        console.log("計時器已停止。");

      } else {

        console.log("計時器未運行。");

      }

    }

  };

}

範例使用：

const myTimer = createTimer(5); // 創建一個 5 秒的計時器

myTimer.start(); // 啟動計時器

稍後停止計時器 (可選)

setTimeout(() => {

  myTimer.stop();

}, 3000);

練習 6寫一個函數 delayedExecution(tasks, delay)，其中 tasks 係一個函數陣列，delay 係延遲時間（毫秒）。要求每隔 delay 時間執行一個任務。

function delayedExecution(tasks, delay) {

  let index = 0; // 追蹤當前執行的任務索引

  const intervalId = setInterval(() => {

    if (index < tasks.length) {

      console.log(`執行的任務: ${index + 1}`);

      tasks[index](); // 執行當前任務

      index++; // 移至下一個任務

    } else {

      console.log("所有任務執行完畢");

      clearInterval(intervalId); // 所有任務執行完畢後清除定時器

    }

  }, delay);

}

const myTasks = [

  () => console.log("任務 1 執行"),

  () => console.log("任務 2 執行"),

  () => console.log("任務 3 執行"),

];

delayedExecution(myTasks, 1000); // 每隔 1 秒執行一個任務

練習 C：判斷題

練習 7判斷以下陳述係咪正確，並解釋原因：

a) 箭頭函數可以用作建構函數

不，箭頭函數（Arrow functions）不能用作建構函數（constructor）。這是因為箭頭函數沒有自己的 this 值，也不能綁定到對象執行上下文，同時也沒有 prototype 屬性，而這些是建構函數所必需的。當嘗試使用 new 操作符調用箭頭函數時，會直接出錯

b) setTimeout(fn, 0) 會立即執行函數

setTimeout(fn, 0) 並不會立即執行函數，而是將該函數加入到瀏覽器的「任務佇列」(task queue) 中，等待當前的同步任務和佇列中已有的其他任務執行完畢後，才在下一次的事件循環中執行。這是一個將任務延後執行、讓出主執行緒的機制，確保了當前程式碼的執行流程，而不是立即執行。

c) 箭頭函數入面嘅 this 永遠指向全域物件

箭頭函數裡面 this 永遠指向全域物件」是錯誤的。箭頭函數的 this 值是從其外層作用域繼承而來，而不是指向全域物件。它會捕獲其所在上下文的 this 值，這與傳統函數的行為不同。

d) setInterval 執行嘅時間間隔係絕對準確嘅

setInterval 的執行時間間隔並非絕對準確，JavaScript 的事件循環機制和瀏覽器/系統的資源競爭會導致時間間隔的延遲。

練習 D：除錯題

練習 9以下程式碼有問題，請指出錯誤並修正： clearInterval(intervalId) missing

function countdown(num) {

  const intervalId = setInterval(() => {

        console.log(num);

        num--;

        if (num < 0) {

        console.log("完成！");

        clearInterval(intervalId);  //stop the interval

        }

    }, 1000);

}

countdown(3);

2.1.10 練習

練習 A：基本物件操作

A1. 建立一個名為 book 的物件，包含以下屬性：

title: "JavaScript 入門"

author: "張老師"

pages: 300

isAvailable: true

const book = {

    title:"JavaScript 入門",

    author: "張老師",

    page: 300,

    isAvailable: true

};

console.log(book);

A2. 使用點記法讀取並輸出 book 的標題與作者。

console.log(book.title);

console.log(book.author);

A3. 使用方括號記法讀取並輸出 book 的頁數。

console.log(book["pages"]);

練習 B：屬性修改與新增

B1. 將 book.isAvailable 修改為 false。B2. 為 book 新增屬性 publisher，值為 "科技出版社"。B3. 為 book 新增屬性 isbn，值為 "978-986-123-456-7"。

book.isAvailable = false;

console.log(book);

book.publisher = "科技出版社";

console.log(book);

book.isbn = "978-986-123-456-7";

console.log(book);

C1. 下列程式碼是否正確？若錯誤，指出原因。

const user = {

  "first-name": "John"

};

console.log(user["first-name"]);

C2. 下面程式碼會輸出什麼？

const person = { name: "Alice" };

console.log(person.height);

undefined

C3. 下列操作會否報錯？說明理由。

const config = { theme: "dark" };

delete config.lanuage;

delete config.theme;

console.log(config);

練習 D：Debug 題

D1. 下列程式碼為何輸出不正確？請修正。

const student = {

  name: "李小明",

  greet() {

    return "你好，我是 " + this.name;

  }

};

console.log(student.greet());

D2. 下列程式碼為何輸出 undefined？請修正。

const calculator = {

  add(a, b) {

   return a + b;

  }

};

console.log(calculator.add(5, 3));

練習 E：進階題目

E1. 建立 car 物件，包含 brand、model、year，並撰寫 getAge() 回傳當前年份減去 year 的結果。

const car = {

    brand: "Toyota",

    model: "Camry",

    year: 2020,

    getAge(){

        return new Date().getFullYear() - this.year;

    }

};

console.log(car.getAge());

console.log(Date());   //Thu Sep 18 2025 14:44:18 GMT+0800 (Hong Kong Standard Time)

//

E3. 使用物件解構，從以下物件提取 name、email，並將 age 重新命名為 userAge：

const userData = {

  id: 1, name: "王小華", age: 28, email: "wang@example.com", city: "台北"

};

const { name, email, age:userAge} = userData;

console.log(userData.name);

console.log(userData.email);

console.log(userData.age);

E4. 撰寫 createUser({ name, age, city })，參數以解構接收，並為 city 設預設值 "未知"。

const createUser = {

  name:"John",

  age: 20,

  city: "未知"

}

const {name, age, city} = createUser;

function createUser ({name, age, city: "Not know"}){

  return {name, age, city};

}

console.log(createUser());

E5. 由以下巢狀物件解構出 name 與 city：

const userProfile = {

  personal: { name: "陳大明", age: 35 },

  contact: { address: { city: "高雄", district: "前鎮區" } }

};

const {

  personal: {name},

  contact: {address: {city}}

} = userProfile;

console.log(name);

console.log(city);

2.1.8 練習

練習A：基礎操作題

題目1：建立和存取陣列

建立一個包含你最喜歡的3種食物的陣列，然後顯示第二個食物。

 const myFavouriteFood = ["Apple", "Banana", " Orangle"];

 console.log(myFavouriteFood[1]);

題目2：陣列長度

有一個陣列 const numbers = [10, 20, 30, 40, 50]，請寫程式碼顯示陣列的長度和最後一個元素。

const numbers = [10, 20, 30, 40, 50];

console.log(numbers.length, numbers[numbers.length-1]);

題目3：修改陣列元素

將上題的陣列中第三個元素（索引2）改為99，然後顯示整個陣列。

const numbers = [10, 20, 30, 40, 50];

numbers[2] = 99;

console.log(numbers);

練習B：陣列方法

題目4：使用push和pop

建立一個空陣列，依次加入"星期一"、"星期二"、"星期三"，然後移除最後一個元素，顯示最終陣列。

const days = [];

days.pus("星期一");

days.pus("星期二");

days.pus("星期三");

days.pop();

console.log(days);

題目5：使用shift和unshift

有陣列 ["二", "三", "四"]，在開頭加入"一"，在結尾加入"五"，然後移除第一個元素，顯示結果。

let week = ["2", "3", "4"];

week.unshift("1");

week.push("5");

week.shift();

console.log(week);

題目6：使用slice

有陣列 [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]，取出第3到第6個元素（包括第3個，不包括第7個），顯示結果，並確認原陣列沒有改變。

const numbers6 = [1, 2,3,4,5, 6,7,8,9,10];

const sliced = numbers6.slice(2,6);

console.log("sliced array:" + sliced);

console.log("original array:" + numbers6);

練習C：判斷題

題目7： 陣列的第一個元素索引是1。（對/錯）

錯

題目8： push()方法會返回新的陣列。（對/錯）

錯

題目9： slice(1, 4)會返回索引1到3的元素。（對/錯）

對

題目10： splice()方法不會改變原陣列。（對/錯）

錯

練習D：Debug題

題目11：找出錯誤

const fruits = ["蘋果", "香蕉", "橙"];

console.log(fruits[3]); // 學生期望顯示某個水果

undefined

題目13：找出錯誤

const colors = ["紅", "綠", "藍"];

colors.splice(1,2,"藍"); // 學生只想刪除"綠"

console.log(colors);

練習E：進階題目

題目16：陣列解構

有一個陣列 [100, 200, 300, 400, 500]，用解構賦值的方式：

將第一個和第二個值賦給變數a和b

將剩餘的值賦給變數rest

顯示三個變數的值

const numbers16 = [100, 200, 300, 400, 500];

const a = numbers16[0];

const b = numbers16[1];

const [a, b, d, e] = numbers16;

console.log("a:" + a);

console.log("d:" + d)

const [a, b, ...rest] = numbers16;

console.log("a:" + a);

console.log("b:" + b);

console.log("rest:" + rest);

// 2.1.6練習

// 練習 A：基本函數操作

// A1. 寫一個名為 calculateTax 嘅函數，接收價錢作為參數，返回加咗 10% 稅後嘅總價。

// function calculateTax(money) {

//    total = money + (money \* 0.10);

//    console.log(total);

// }

// calculateTax(100);

// function calaculateTax(price){

//     return price \* 1.1;

// }

// A2. 建立一個 greetCustomer 函數，接收客人姓名，如果冇提供姓名就用「客人」作預設值。

// function greetCustomer(name="Guest"){

//         console.log(name);

// }

// greetCustomer();

// A3. 用箭頭函數寫一個 isEven 函數，檢查一個數字係咪偶數。

const isEven = number => number % 2  === 0;

console.log(isEven(7));

// 練習 B：判斷題

// B1. 以下代碼嘅輸出係咩？

// function test() {

//     return 5;

// }

// console.log(test());

// B2. 箭頭函數可以用作建構函數嗎？（是/否）

//  Yes

// B3. 以下兩個函數係咪完全一樣？

// const func1 = function(x) { return x \* 2; };

// const func2 = (x) => x \* 2;

// Yes

// 練習 C：Debug 題

// C1. 搵出以下代碼嘅錯誤並修正：

// function add(a, b) {

//    return a + b;

// }

// const result = add(3, 5);

// console.log(result);

// C2. 修正以下物件方法：

// const user = {

//     name: "Tom",

//     getName: function () {   // getName(){

//         return this.name

//     }

//  };

// console.log(user.getName());

// 練習 D：進階題目

// D3. 用閉包建立一個簡單嘅銀行戶口，有存款、提款同查詢餘額功能。

// const person3 = {

//     accountName: "Peter",

//     accountAmount: 2000

// }

// console.log(person3.accountAmount);

// function createBankAccount(initialBalance){

//     let balance = initialBalance;

//     function getBalance() {

//         return balance;

//     }

//     function deposit(amount){

//             balance += amount;

//     }

//     function withdraw(amount){

//         if(amount <= balance) balance -= amount;

//     }

//     return {

//         getBalance: getBalance,

//         deposit: deposit,

//         withdraw: withdraw

//     };

// }

// const acc1 = createBankAccount(1000);

// acc1.deposit(1500);

// acc1.withdraw(500);

// console.log(acc1.getBalance());

//const acc2 = createBankAccount(500);

//console.log(acc2.getBalance());

// D4. 寫一個函數檢查參數係咪函數，如果係就執行佢，如果唔係就顯示錯誤訊息。

function gree99() {

    console.log("hi");

}

function gree88() {

    return "hi88";

}

function safeExecute(fn) {

    if (typeof fn === "function"){

        fn();

    } else {

        console.log("not function");

    }

}

safeExecute(gree99);

safeExecute("123");

/\*

2.1.4練習

練習 A：基本條件判斷

1. 寫一個程式，檢查一個數字係咪正數。如果係正數就輸出「正數」，否則輸出「負數或零」。

\*/

let x = -5

if (x>0) {

    console.log("Positive number");

} else {

    console.log("Negative number");

}

// 練習 B：多重條件判斷

// 4. 根據分數給予等級：90分或以上係A，80-89係B，70-79係C，60-69係D，60分以下係F。

let score = 67;

if (score >= 90){

    console.log("Grade A");

} else if (score >= 80) {

    console.log("Grade B");

} else if (score >= 70) {

    console.log("Grade C");

} else if (score >= 60) {

    console.log("Grade D");

} else {

    console.log("Grade F");

}

let grade = score >= 90 ? "Grade A" :

            score >= 80 ? "Grade B" :

            score >= 70 ? "Grade C" :

            score >= 60 ? "Grade D" : "Grade F";

            console.log(grade);

// 5. 建立一個程式判斷季節：11-2月係冬天，3-4月係春天，5-8月係夏天，9-10月係秋天。

// 練習 C：邏輯運算子

// let month = 8

// let session = month >= 3 ? "Spring" :

//               month >= 8 ? "Summer" :

//               month >= 9 ? "Autum" :

//               month >=12 ? "Winter":

//             console.log(session);

const month = -2;

// if (month >= 3 && month <=4){

//     console.log("Spring");

// } else if (month > 4 && month <= 8) {

//     console.log("Summer");

// } else if (month > 8 && month <= 11) {

//     console.log("Autumn");

// } else {

//     console.log("Winter");

// }

if (month <=12 && month >= 11  || month >=1 && month <=2){

    console.log("Winter");

} else if (month >=3 && month <= 4) {

    console.log("Spring");

} else if (month >=5 && month <= 8) {

    console.log("Summer");

} else if (month>=0 && month <=10){

    console.log("Autumn");

} else {

    console.log("No this month");

}

// 練習 C：邏輯運算子

// 6. 檢查一個人係咪可以借書：年齡要18歲或以上，同時要有有效嘅圖書證。

const personAge = 19;

const hasValidLibraryCard = true;

if (personAge >=18 && hasValidLibraryCard ) {

    console.log("Age over 18 and has valid libruary card")

} else {

    console.log("Age under 18 or not valid libray Card")

}

// 7. 寫一個程式檢查係咪適合戶外活動：天氣要好（唔落雨）同埋溫度要喺15-30度之間。

const isNotRaining = true;

const temp = 31

if (isNotRaining && temp >=15 && temp <=30 ) {

    console.log("Good weather and has temperature between 15 and 30 degree celes")

} else {

    console.log("Not good weather and temperature not between 15 and 30 degree celes")

}

// 練習 D：三元運算子

// 8. 用三元運算子檢查一個數字係咪偶數，係就返回「偶數」，唔係就返回「奇數」。

const number = 8

const result = number % 2 == 0 ? "Even number" : "Odd number";

console.log(result);

// 9. 用三元運算子根據時間決定問候語：如果係12點之前就返回「早晨」，否則返回「午安」。

const hour = 14

const greeting = hour >12 ? "Good afternoon" : "Good morning";

console.log(greeting);

// 練習 E：Debug 題目

// 10. 以下程式碼有乜嘢問題？請指出同修正：

// const score = 85;

// if (score = 90) {

//     console.log("滿分");

let score1 = 80;

if (score1 === 90) {

    console.log("Full Mark");

}

// 11. 以下程式碼有乜嘢問題？請指出同修正：

// const age = 25;

// if (age >= 60)

//     console.log("長者折扣");

//     console.log("歡迎光臨");

const age = 25;

if (age >= 60) {

    console.log("長者折扣");

} else {

    console.log("歡迎光臨");

}

// 12. 以下程式碼有乜嘢問題？請指出同修正：

// const score = 75;

// if (score >= 60) {

//     console.log("及格");

// } else if (score >= 80) {

//     console.log("良好");

// }

const score4 = 59;

if (score4 >= 80) {

    console.log("良好");

} else if (score4 >= 60) {

    console.log("及格");

} else {

    console.log("Failed");

}

// 練習 F：進階題目

// 13. 寫一個程式判斷閏年：能被4整除但唔能被100整除，或者能被400整除。

let year = 2024;

if (year %4 === 0 && year%100 !==0 || year%400==0){

    console.log(`${year}閏年`);

} else {

    console.log(`${year}不閏年`);

}

// 14. 建立一個簡單嘅計算機，根據運算子（+、-、\*、/）進行計算，如果運算子無效就顯示錯誤訊息。

const num1 = 10;

const num2 = 5;

let operator = "/";

if(operator === "+") {

    console.log(num1 + num2);

} else if(operator === "-") {

    console.log(num1 - num2);

} else if(operator ==="\*"){

    console.log(num1\*num2);

} else if (operator ==="/") {

    if(num2 !== 0) {

         console.log(num1 /num2);

    } else {

        console.log("not divide by 0");

    }

} else {

        console.log("not calculation")

}

// 15. 寫一個程式檢查密碼強度：至少8個字符、包含大楷同細楷字母、包含數字。

const password = "AAA123";

const hasMinLength = password.length >=8;

const hasUpperCase = /[A-Z]/.test(password);

const hasLowerCase = /[a-z]/.test(password);

const hasNumber = /[0-9]/.test(password);

const hasCharacter = /[A-z]/.test(password);

console.log(`${password} \n ${hasMinLength} \nUpperae: ${hasUpperCase} \nLowercase: ${hasLowerCase} \nnumber: ${hasNumber} \nCharacter: ${hasCharacter}`)

// 16. 根據 BMI 值判斷體重狀況：低於18.5係過輕，18.5-24.9係正常，25-29.9係過重，30或以上係肥胖。

const weight = 100;  // (kg)

const height = 1.7;  // (m)

const bmi = weight/(height\*\*2);

console.log(bmi);

if (bmi < 18.5) {

    console.log("less than 18.5 less weight");

} else if (bmi <25) {

    console.log("18.5-24.9 normal");

} else if (bmi <30) {

    console.log("25-29.9 over weight");

} else {

    console.log("over 30 fat");

}

// 17. 寫一個程式判斷三角形類型：等邊（三邊相等）、等腰（兩邊相等）、普通三角形或唔係三角形。

const side1 = 5;

const side2 = 8;

const side3 = 8;

const isValidTrangle = side1 + side2 > side3 && side1 + side3 > side2 && side2 + side3 > side1;

if(isValidTrangle){

    if(side1 == side2 && side2 === side3){

        console.log("equal side trangle")

    }else if (side1 === side2 || side2 === side3 || side1 === side3){

        console.log("equal length tranigle")

    } else {

        console.log("common triangle")

    }

} else {

    console.log(" Not Triangle");

}

// 練習 G：實際應用題目

// 18. 建立一個簡單嘅購物車折扣系統：購買金額超過500元有9折，超過1000元有8折，超過2000元有7折。

let spent = 1200;

if (spent >=500 && spent <=990) {

    console.log(spent \* 0.9)

} else if (spent >=1000 && spent <= 1999){

    console.log(spent \* 0.8)

} else if (spent >2000) {

    console.log(spent \* 0.7)

}

const amount = 1500;

let finalAmount= amount;

if(amount >2000){

    finalAmount \*= 0.7;

    console.log(`Original amount:$${amount}, discount $${finalAmount}`)

} else if (amount >1000){

    finalAmount \*= 0.8;

    console.log(`Original amount:$${amount}, discount $${finalAmount}`)

} else if (amount >500) {

    finalAmount \*= 0.9;

    console.log(`Original amount:$${amount}, discount $${finalAmount}`)

} else {

    console.log(`Original amount:$${amount}, no discount`)

}

// 19. 寫一個程式計算停車費：首小時15元，之後每小時10元，超過8小時係一日票60元。

let taxiTime = 9

if (taxiTime <= 1) {

    console.log("less than 1 hour fee $150")

} else if (taxiTime >1 && taxiTime <8){

    console.log("after 1 hour fee $10")

} else if (taxiTime > 8) {

    console.log("greater than 6 hours fee $60")

}

//

const hours = 1;

let fee;

if (hours >8) {

    fee = 60;

    console.log(`${hours} hour, dayfee$${fee}`);

} else {

    fee = 15 + (hours-1) \*10;

    console.log(`$${hours} hour, fee $${fee}`);

}

// 20. 建立一個學生成績管理系統：根據多科成績計算平均分，然後判斷係咪可以升班（平均分要60或以上，同時冇科目低於40分）

let studentResult = 60;

if (studentResult >= 60) {

    console.log("Passed");

} else if (studentResult <60) {

    console.log("Failed")

}

const math = 75;

const english = 80;

const science = 65;

const average = (math + english + science) / 3;

const hasFailingGrade = math < 40 || english < 40 || english < 40 || science < 40;

if(average >=60 && !hasFailingGrade){

    console.log("Up level");

} else {

    console.log("not level")

}

2.1.3練習

練習 A：基本比較運算子

請寫出以下比較運算的結果（true 或 false）：

console.log(5 == "5");  //true

console.log(5 === "5"); //false

console.log(0 == false);  //true

console.log(0 === false); //false

console.log("" == 0);     //true

請寫出以下表達式的結果：

console.log (null == undefined); //true

console.log(null === undefined);  //false

console.log(NaN == NaN);  //false

console.log(NaN === NaN);  //false

練習 B：判斷題

請判斷以下說法是否正確（對或錯）：

使用 == 比使用 === 更安全  //錯

NaN 等於它自己  //對

typeof null 的結果是 "null"  //錯 (object)

undefined 表示變數未被賦值     //對

null 和 undefined 在寬鬆比較中是相等的 //錯

// 2.1.2練習

// 練習 A：基礎資料型別識別

// 題目 1： 請使用 typeof 運算子檢查以下變數的資料型別：

// const score = 95;

// const studentName = "李小明";

// const isPassed = true;

// let grade;

// const scholarship = null;

const score = 95;

console.log(typeof score);   //number

const studentName = "Li Siu Ming";

console.log(typeof studentName);  // string

const isPassed = true;

console.log(typeof isPassed); //boolean

let grade;

console.log(typeof grade);  //undefined

const scholarship = null;

console.log(typeof scholarship);  // object

// 題目 3： 以下程式碼會輸出什麼？請寫出預期結果：

// console.log(typeof 42);

// console.log(typeof "42");

// console.log(typeof true);

// console.log(typeof undefined);

// console.log(typeof null);

console.log(typeof 42);   // number

console.log(typeof "42");  //string

console.log(typeof true);   // boolean

console.log(typeof undefined); // undefined

console.log(typeof null); // object

// 練習 B：型別轉換操作

// 題目 4： 請將以下字串轉換為數字：

// const str1 = "100";

// const str2 = "3.14";

// const str3 = "0";

const str1 = "100";

console.log(Number(str1));  //100

const str2 = "3.14";

console.log(Number(str2));  //3.14

const str3 = "0";

console.log(Number(str3));  //0

// 題目 5： 請將以下數字轉換為字串：

// const num1 = 42;

// const num2 = 0;

// const num3 = -15;

const num1 = "42";

console.log(typeof num1);

const num2 = "0";

console.log(typeof num2);

const num3 = "-15";

console.log(typeof num3);

const num4 = 10;

let tmp = String(num4);

console.log(typeof tmp);

console.log(tmp);

// 題目 6： 請判斷以下值轉換為布林值後的結果：

// Boolean(1)

// Boolean(0)

// Boolean("")

// Boolean("hello")

// Boolean(null)

// Boolean(undefined)

console.log("\_\_\_\_\_\_\_\_\_\_\_");

Boolean(1);

console.log(Boolean(1));  // true

Boolean("hello")

console.log(Boolean("hello"));  //true

// Falsy

Boolean(0)

console.log(Boolean(0));  //false

Boolean("")

console.log(Boolean("")); //false

Boolean(null)

console.log(Boolean(null)); //false

Boolean(undefined)

console.log(Boolean(undefined));  //false

Boolean(NaN)

console.log(Boolean(NaN));  //false

Boolean(false)

console.log(Boolean(false));  //false

Boolean(-0)

console.log(Boolean(-0));  //false

// 題目 7： 以下程式碼有什麼問題？請修正：

// const price = "50";

// const tax = 5;

// const total = price + tax;

// console.log("總價：" + total);

const price = "50";

const tax = 5;

total = Number(price) + tax;

console.log("總價：" + total);

// 練習 D：進階題目

// 題目 9： 建立一個函數，接收任何型別的參數，並回傳該參數的資料型別名稱。

function getDataType(param){

    return typeof param;

}

console.log(getDataType(123));      // "number"

console.log(getDataType("Hello"));  // "string"

console.log(getDataType(true));      // "boolean"

console.log(getDataType({ a: 1 }));  // "object"

console.log(getDataType([1, 2]));    // "object"

console.log(getDataType(null));      // "object"

console.log(getDataType(undefined)); // "undefined"

console.log(getDataType(function() {})); // "function"

// Sir Answer

function getType(value){

  return typeof value;

}

// 題目 10： 寫一個程式，檢查使用者輸入的年齡是否為有效數字（不是 NaN），並假設最大年齡是150歲。

function validateAge(ageInput) {

  // 1. 嘗試將輸入轉換為數字

  const age = Number(ageInput); // <<!nav>>Number()<<!/nav>> 可以處理數字字串和字串

  // 2. 檢查是否為 NaN（非數字）

  if (isNaN(age)) {

    return "年齡不是一個有效的數字。"; // 處理非數字輸入

  }

  // 3. 檢查年齡是否在有效範圍內（0 到 150）

  if (age < 0 || age > 150) {

    return "年齡必須在 0 到 150 之間。"; // 處理超出範圍的數字

  }

  // 4. 如果通過所有檢查，則為有效年齡

  return "年齡有效。";

}

// 範例使用方法：

let userInput1 = "25"; // 有效數字

let userInput2 = "abc"; // 非數字

let userInput3 = "170"; // 超出最大年齡

let userInput4 = "-10"; // 小於最小年齡

let userInput5 = "0";   // 有效數字

console.log(`輸入「${userInput1}」的結果：` + validateAge(userInput1)); // 預期：年齡有效。

console.log(`輸入「${userInput2}」的結果：` + validateAge(userInput2)); // 預期：年齡不是一個有效的數字。

console.log(`輸入「${userInput3}」的結果：` + validateAge(userInput3)); // 預期：年齡必須在 0 到 150 之間。

console.log(`輸入「${userInput4}」的結果：` + validateAge(userInput4)); // 預期：年齡必須在 0 到 150 之間。

console.log(`輸入「${userInput5}」的結果：` + validateAge(userInput5)); // 預期：年齡有效。

//Sir Answer

// age = 18

function isValidAge(age) {

  const n = Number(age);

  const checkIsNan = Number.isNaN(n);

  const isBiggerThan0 = n >=0;

  const isSmallerThan150 = n <= 150;

  return !checkIsNan && isBiggerThan0 && isSmallerThan150;

}

console.log(isValidAge(160));

// 題目 11： 建立一個安全的加法函數，確保兩個參數都是數字再進行相加。

function safeAdd(a, b) {

  // 檢查 a 和 b 是否皆為數字

  if (typeof a === 'number' && typeof b === 'number') {

    return a+b // 若皆為數字，則進行加法運算

  } else {

    return "錯誤：參數必須皆為數字"; // 若非數字，則回傳錯誤訊息

  }

}

//Sir answer

// a = "1" b= 2

function safeAdd(a, b) {

  //1. check first value is valid number

  const x = Number("1");

  //2. check second value is valid number

  const y = Number(2);

  if (Number.isNaN(x) || Number.isNaN(y)){

    return "Invalid number";

  }

  return x + y;

}

let a = 10;

let b = 3

console.log(a%b);

console.log(a \*\* b);

// && ---> AND

console.log(true && true);

console.log(true && false);

console.log(false && true);

console.log(false && false);

// ||  --> OR

console.log(true || true);

console.log(true || false);

console.log(false || true);

console.log(false || false);

 // ! ---> NOT

console.log(!true);

console.log(!false);

let result = 11/3;

console.log(result);  // 3.6666666666666665

console.log(Math.floor(result));  // 3

console.log(Math.ceil(result));  // 4

console.log(Math.round(result));  // 4

console.log(result.toFixed(2));  //3.67

console.log("9" > "10");

// 2.1.1練習

// 練習 A：基礎變數宣告

// 1. 用 const 宣告一個變數叫 schoolName，值係「香港大學」

const schoolName = '香港大學';

// 2. 用 let 宣告一個變數叫 studentCount，值係 1000

let studentCount = 1000;

// 3. 將 studentCount 嘅值改為 1200

studentCount = 1200;

// 用 console.log 輸出兩個變數的值

console.log(schoolName, studentCount);

//練習 B：變數類型判斷

// 1. 一個人嘅出生日期

// 2. 銀行戶口餘額

// 3. 圓周率（π）

// 4. 網頁嘅標題

// 5. 購物車入面商品嘅數量

// 1. const

// 2. let

// 3. const

// 4. let

// 5. let

//練習 C：錯誤偵測

// 以下程式碼有咩問題？請指出錯誤並提供正確嘅寫法：

// 題目 1

// const price = 100;

// price = 120;

// console.log(price);

// 題目 2

// let user name = "John";

// console.log(user name);

// 題目 3

// let count = 10;

// const count = 15;

// console.log(count);

// 題目 4

// if (true) {

//     let message = "Hello";

// }

// console.log(message);

// 1. const change to let

// 2. userName instead of user name

// 3. const delete

// 4. if (message = true)

// 練習 E：進階題目

// 題目 1：複雜作用域

// 請預測每行 console.log 嘅輸出，然後複製到vs code裡試試效果。

// const name = "外層";

// function outer() {

//     const name = "外函數";

//     function inner() {

//         const name = "內函數";

//         console.log(name);

//     }

//     inner();

//     console.log(name);

// }

// outer();

// console.log(name);

// const name = '外層';

// function outer() {

//     const name = "外函數";

//     function inner(){

//         const name = "內函數";

//         console.log(name);          // 內函數

//     }

//     inner();

//     console.log(name);          // 外函數

// }

// outer();

// console.log(name);  //外層

/\*

1 -》內函數

2 -》外函數

3 -》外層

\*/

// 練習 F：實際應用

// 應用題 1：學生管理系統

// 假設你要寫一個簡單嘅學生管理系統，請宣告適當嘅變數：

// 學校名稱（唔會變）

// 當前學期（可能會變）

// 學生總數（會變）

// 每班最多學生數（唔會變）

// 系統是否已啟動（會變）

const schoolName = '香港大學';

let currentSemester = "2024 Spring";

let totalStudents = 30;

const maxStudentsPerClass = 10;

let IsSystemactive = true;

// 應用題 2：購物車計算

// 寫一個購物車計算程式：

// 宣告商品價格（假設 $299）

// 宣告購買數量（可以改變）

// 計算總價

// 如果超過 $500，打 9 折

let productPrice = 299;

let quality = 5;

let totalPrice = productPrice \* quality;

console.log(totalPrice);

if (totalPrice > 500) {

    totalPrice = totalPrice \* 0.9

}

console.log("total price:" + totalPrice);

// 應用題 3：溫度轉換

// 寫一個溫度轉換程式：

// 宣告攝氏溫度變數

// 轉換為華氏溫度（公式：華氏 = 攝氏 × 9/5 + 32）

// 用適當嘅變數類型儲存結果

const celsiusTemp = 30;

let fahrenheitTemp = (celsiusTemp \*9) / 5 + 32;

console.log("攝氏溫度 :"+ celsiusTemp + " = " + "華氏溫度:" + fahrenheitTemp );