INT151 EXAM-01 — EXAM-READY KNOWLEDGE BASE

▼ table of contents

- table of contents
  - Deep Content Analysis จากเอกสารจริงของวิชา (ยังไม่ใช่ข้อสอบ)
  - PHASE 1 (รวมกับ Final Output #1): EXAM CONTEXT ANALYSIS
  - PHASE 2 (Final Output #2): DOCUMENT ANALYSIS
    - ภาพรวมไฟล์ที่ได้รับ
    - หลักฐานความซ้ำ (Evidence of Emphasis จาก Phase 1 rule)
    - ข้อความยืนยันขอบเขตข้อสอบโดยตรงจากอาจารย์ (หลักฐานสำคัญที่สุดในเอกสารทั้งหมด)
  - PHASE 3 (Final Output #3): COMPLETE INT151 TOPIC MAP
  - PHASE 4 (Final Output #4): DEEP KNOWLEDGE BASE
    - A1–A2. ตัวแปร ชนิดข้อมูล และ Truthy/Falsy/Equality
    - A3. Conditionals (if/else if/else, switch, guard clause)
    - A4. Loop (for, while, do...while, for...of, for...in, break/continue)
    - A5. Function (Declaration/Expression/Arrow, parameter/return, scope)
    - A6. Array & Array Methods
    - A7–A8. Object & Reference vs Copy
    - A9–A10. Destructuring & Spread/Rest
    - A11. JavaScript Date
    - A12. Regular Expression (Regex)
    - A13. Exception Handling (try/catch/finally, throw, custom Error)
    - A14. JavaScript Module (CommonJS / ES Module)
    - A15. การอ่าน Error Message / Debugging
    - B. Web Development Fundamentals
    - C. DOM (Document Object Model)
  - PHASE 5 (Final Output #5): CONCEPT RELATIONSHIPS & DEPENDENCY MAP
  - PHASE 6 (Final Output #6): COMMON CONFUSIONS & MISTAKES
  - PHASE 7 (Final Output #7): PROFESSOR EMPHASIS ANALYSIS
  - PHASE 8 (Final Output #8): THEORY EXAM RELEVANCE (Part 1 — Closed Book)
  - PHASE 9 (Final Output #9): PRACTICAL EXAM RELEVANCE (Part 2 — VSCode + MDN)
  - PHASE 10 (Final Output #10): THEORY vs PRACTICAL MATRIX
  - PHASE 11 (Final Output #11): TIER 1–4 PRIORITY
    - 🔴 TIER 1 — MUST MASTER (ถ้าไม่รู้ มีผลต่อคะแนนอย่างมาก ทั้ง Theory และ Practical)
    - 🟠 TIER 2 — HIGH PRIORITY (ควรเข้าใจดี)
    - 🟡 TIER 3 — MEDIUM (ควรทบทวน)
    - 🟢 TIER 4 — LOW (อ่านเมื่อมีเวลา / มีหลักฐานว่าไม่เน้น)
  - PHASE 12 (Final Output #12): EXAM-READY CHECKLIST
  - PHASE 13 (Final Output #13): FINAL MASTER SUMMARY
  - ข้อจำกัดของการวิเคราะห์นี้ (ตามกฎ "ถ้าไม่มีหลักฐานเพียงพอ ต้องระบุว่าไม่สามารถยืนยันได้")

Deep Content Analysis จากเอกสารจริงของวิชา (ยังไม่ใช่ข้อสอบ)

> **หมายเหตุความน่าเชื่อถือของเอกสาร:** วิเคราะห์จาก 2 ไฟล์ที่ได้รับ — ทั้งสองไฟล์เป็น PDF ที่มีทั้งส่วนอ่านได้และอ่านไม่ได้ (ดูละเอียดใน Phase 2) ข้อสรุปทั้งหมดในเอกสารนี้อ้างอิงเฉพาะจากส่วนที่อ่านได้จริงเท่านั้น ส่วนใดที่เป็นความรู้เสริมนอกเอกสารจะ

> ระบุว่า **"Additional Knowledge"** กำกับไว้ชัดเจน ส่วนใดที่ไม่มีหลักฐานเพียงพอจะระบุว่า **"ไม่สามารถยืนยันได้จากเอกสาร"**

PHASE 1 (รวมกับ Final Output #1): EXAM CONTEXT ANALYSIS

รายการ	รายละเอียดที่ทราบแน่นอน (จากโจทย์ผู้ใช้)
รหัสวิชา/ข้อสอบ	INT151 Exam-01
คะแนนรวม	30 คะแนน
Part 1 — Theory	15 คะแนน, Closed Book, 1 ชั่วโมง, MCQ + Multiple Correct, สอบผ่าน LEB2, ห้าม Editor/ตำรา/เว็บ/AI/แอปอื่น
Part 2 — Practical	15 คะแนน, 2 ชั่วโมง, สอบผ่าน Classroom50 ด้วย VSCode, Restricted Resources (เปิดได้เฉพาะ Google Translate, MDN, เอกสารการสอนบน LEB2)

ผลต่อการวิเคราะห์เนื้อหา:

- **Part 1 (Closed Book)** → เนื้อหาที่เหมาะสมต้องเป็นสิ่งที่ "จำได้ / เข้าใจได้โดยไม่ต้องเปิดอ้างอิง" เช่น concept, พฤติกรรมของภาษา, กฎพิเศษ (เช่น `NaN !== NaN`), ความแตกต่างระหว่างของสองสิ่งที่คล้ายกัน, การอ่าน output จาก code สั้น ๆ
- **Part 2 (Restricted Resources)** → นักศึกษาเปิด MDN ได้ ดังนั้น **Syntax ปลีกย่อยที่เปิดค้นได้ไม่ใช่จุดตาย** แต่ **Concept ที่ MDN ช่วยไม่ได้ถ้าไม่เข้าใจ** (เช่น reference vs copy, closure, scope, immutability, error-message reading) คือของที่ต้องแม่นจริง เพราะ MDN บอกแค่ "syntax คืออะไร" แต่ไม่บอก "ทำไม code นี้ให้ผลลัพธ์แบบนี้"
- ข้อสอบมีลักษณะ **Perfect Match / Multiple Correct** (ยืนยันจากเอกสารทบทวน บทที่ 12 ที่ฝึกรูปแบบนี้โดยเฉพาะ) → ต้องตัดสินใจทีละตัวเลือกอย่างเป็นอิสระ ไม่ใช่หา "ข้อที่ถูกที่สุด"

PHASE 2 (Final Output #2): DOCUMENT ANALYSIS

ภาพรวมไฟล์ที่ได้รับ

ไฟล์	หน้ารวม	ส่วนที่อ่านได้	ส่วนที่อ่านไม่ได้	ลักษณะเนื้อหา
`class1-5.pdf`	517	หน้า 347–517 (171 หน้า)	หน้า 1–346 (เสียหายฟอนต์ไทยแสดงเป็นกล่อง ทั้ง text-layer และการ render ด้วยตา — ยืนยันด้วย 2 renderer คนละตัว)	หน้า 347 = Syllabus สัปดาห์ 1; หน้า 348–403 = สไลด์ "INT151-Basic Frontend Development" (Variable/Condition/Loop → Function → Array → Object → Spread/Destructuring → Date → Regex → Exception → Module); หน้า 404–414 = สไลด์ "Web Development Fundamentals" (Architecture, Client-Server, HTTP, REST); หน้า 415–462 = สไลด์ DOM (โดย ผศ.ดร.อุมาพร สุภสิทธิเมธี); หน้า 463–517 = เอกสารทบทวนเว็บ "ทบทวน JavaScript" 12 บท
`ถอดเสียง_สไลด์.pdf`	373	เกือบทั้งหมด (fonts ไม่เสีย)	ไม่มี (อ่านได้ตลอด)	หน้า 1–~319 = **Transcript คำพูดสอนจริง** จาก 5 การอัดเสียง: `20260805`, `20260811(ALL)`, `20260819`, `20260826`, `20260902`; หน้า ~320–373 = สไลด์ชุดเดียวกับใน `class1-5.pdf` (Conditionals/Loops → … → Regex) ซ้อนทับเนื้อหาเดิม

หลักฐานความซ้ำ (Evidence of Emphasis จาก Phase 1 rule)

Topic ที่ปรากฏ **ทั้งใน (ก) transcript คำสอนจริง 5 ครั้ง + (ข) สไลด์ + (ค) เอกสารทบทวน 12 บท** ถือเป็นหลักฐานเน้นย้ำสูงสุด ได้แก่: **Reference vs Copy (Object/Array), Function (parameter/return/scope), Conditionals, Loop (for/for..of/for..in), Array methods, Destructuring/Spread, Regex, try/catch/Exception**

ข้อความยืนยันขอบเขตข้อสอบโดยตรงจากอาจารย์ (หลักฐานสำคัญที่สุดในเอกสารทั้งหมด)

จาก transcript คำสอนจริง พบข้อความอาจารย์พูดตรง ๆ เกี่ยวกับขอบเขตข้อสอบ 3 จุด:

1. **"...แต่ว่าข้อสอบยังไม่ออก Temporal นะคะ มันเพิ่งออกมา..." → Temporal API ไม่อยู่ในขอบเขต Exam-01** (ยืนยันชัดเจน)
2. **"ตัวซิสเต็มไดอะล็อกไม่เป็นไรค่ะ อันนี้ไม่ได้ออกข้อสอบแล้วก็ไม่ได้สำคัญมาก..." → System Dialogs (alert/confirm/prompt) ไม่อยู่ในขอบเขต Exam-01 และความสำคัญต่ำ**
3. **"...มีอะไรที่ยังไม่เก็บให้เราอีก ที่อาจารย์สอนวันนี้ออกข้อสอบหมดเลยนะคะ..."** → เนื้อหาที่สอนในคาบเรียนนั้น (ครอบคลุม DOM/Node/Selector) มีแนวโน้มสูงมากที่จะออกสอบ
4. ยืนยันรูปแบบสอบ: **"ทฤษฎีก็เป็น choice multiple collect เนอะ แล้วก็แล็บก็เป็นสอบแลป"** — ตรงกับ spec ที่ให้มา

> ข้อ 1–2 คือ **Hard Evidence** ระดับสูงสุดที่พบในเอกสาร ควรใช้เป็นเกณฑ์ตัดเนื้อหาออกจาก Tier สูง (ดู Phase 14)

PHASE 3 (Final Output #3): COMPLETE INT151 TOPIC MAP

```
INT151 FRONT-END (ขอบเขตที่มีหลักฐานจากเอกสาร)
│
├── A. JAVASCRIPT FUNDAMENTALS (สัปดาห์ 1 – ทบทวนพื้นฐาน)
│   ├── A1. Variable & Data Type (let/const/var, primitive vs object, typeof)
│   ├── A2. Truthy/Falsy & Equality (==, ===, NaN, falsy 6 ค่า)
│   ├── A3. Conditionals (if/else if/else, switch, guard clause, && vs ||)
│   ├── A4. Loop (for, while, do...while, for...of, for...in, break/continue, loop ซ้อน)
│   ├── A5. Function (Declaration/Expression/Arrow, parameter/return, scope, callback, default parameter, higher-order function)
│   ├── A6. Array (index, .length, mutating vs non-mutating methods: push/pop/shift/unshift/sort/splice, non-mutating: map/filter/r
educe/find/concat)
│   ├── A7. Object (dot vs bracket notation, property/method, this, reference ไม่ใช่ copy, in / optional chaining `?.`)
│   ├── A8. Reference vs Copy / Primitive vs Object (ตัวแปรชี้ที่อยู่ ไม่ใช่ถือค่า, shallow copy, structuredClone)
│   ├── A9. Destructuring (Array-ตามตำแหน่ง, Object-ตามชื่อ key, rename, default, nested, destructure ใน parameter)
│   ├── A10. Spread (...) & Rest (...) (กระจาย vs รวบรวม, merge object/array, key ชนกัน)
│   ├── A11. JavaScript Date (new Date() หลายรูปแบบ, getters, toLocaleString, คำนวณผลต่างวัน, Date.now())
│   ├── A12. Regular Expression (pattern, flag g/i, \d \w \s ^ $, test/match/replace/split)
│   ├── A13. Exception Handling (try/catch/finally, throw, custom Error class extends Error, instanceof)
│   ├── A14. JavaScript Module (CommonJS require/module.exports, ES Module named/default export, import รูปแบบต่าง ๆ)
│   └── A15. Debugging / อ่าน Error Message (undefined 3 ทาง, อ่านบรรทัดบนสุดของ error, เช็ก 3 จุดก่อนส่ง)
│
├── B. WEB DEVELOPMENT FUNDAMENTALS
│   ├── B1. Web Architecture (3-Tier: Presentation–Application–Data, ปัจจัยด้านคุณภาพ: Speed/Security/Scalability/Maintainability)
│   ├── B2. Client-Server Model & องค์ประกอบเว็บแอป (Browser, Web Server, App Server, DB Server)
│   ├── B3. HTTP Methods (GET/POST/PUT/DELETE)
│   ├── B4. HTTP Status Code (2xx/3xx/4xx/5xx)
│   ├── B5. HTTP Stateless (นิยาม, ข้อดี, การจำสถานะด้วย Cookie/Token)
│   └── B6. REST (นิยามเบื้องต้น – เอกสารมีแค่หัวข้อ ไม่มีเนื้อหาละเอียดในส่วนที่อ่านได้)
│
├── C. DOM (Document Object Model) – สอนโดย ผศ.ดร.อุมาพร สุภสิทธิเมธี
│   ├── C1. DOM คืออะไร (tree of nodes, cross-platform, language-independent)
│   ├── C2. `<script>` Tag Placement (head vs ท้าย body, ผลต่อ render blocking)
│   ├── C3. window / document object, documentElement, document.head/body
│   ├── C4. Node Types (nodeName/nodeType/nodeValue, 12 numeric constants, ELEMENT_NODE ฯลฯ)
│   ├── C5. Element, Attr (createElement, attributes, ownerElement)
│   ├── C6. innerHTML vs innerText vs textContent
│   ├── C7. Traversing Nodes (firstChild/lastChild/childNodes, Element Traversal API: firstElementChild/lastElementChild/previousEl
ementSibling/nextElementSibling/childElementCount)
│   ├── C8. Selecting Nodes (getElementById/getElementsByTagName/getElementsByClassName/querySelector/querySelectorAll/closest)
│   ├── C9. HTMLCollection (live) vs NodeList (static)
│   ├── C10. Manipulating Nodes (appendChild/insertBefore/replaceChild/removeChild/createElement/createTextNode/createAttribute)
│   ├── C11. DocumentFragment (ลด Reflow/Repaint)
│   ├── C12. dataset (data-* attribute, DOMStringMap)
│   ├── C13. classList (add/remove/toggle/contains/replace)
│   └── C14. System Dialogs (alert/confirm/prompt) – **มีหลักฐานตรงว่าไม่ออกสอบ/ไม่สำคัญมาก**
│
└── D. (Additional Knowledge – ไม่อยู่ในเอกสารที่อ่านได้ แต่ถูกกล่าวถึงในโครง syllabus)
    └── Temporal API – **มีหลักฐานตรงว่าไม่ออกสอบ Exam-01** (เพิ่งออก, อาจารย์บอกชัดเจน)
```

> **หมายเหตุ:** โครงสร้างนี้สร้างจากเอกสารจริงทั้งหมด ไม่ได้ตัดหัวข้อใดออกแม้จะดูรอง (เช่น REST, System Dialogs) ตามกฎที่กำหนด แต่จะถูกจัดลำดับความสำคัญต่ำใน Phase 14 หากมีหลักฐานสนับสนุน

PHASE 4 (Final Output #4): DEEP KNOWLEDGE BASE

> โครงสร้างต่อไปนี้ยึด 13 หัวข้อที่ขอ (Definition/Purpose/Core Concept/How it works/Syntax/Rules/Code/Interpretation/Mistakes/Related/Prerequisite/Next/Required Knowledge Level) แบบย่อรวมเป็นหมวดให้อ่านง่าย โดยไม่ตัดเนื้อหาออก อ้างอิงจากสไลด์ "INT151-Basic Frontend Development" + เอกสารทบทวน "ทบทวน JavaScript" (12 บท) + transcript คำสอนจริง

A1–A2. ตัวแปร ชนิดข้อมูล และ Truthy/Falsy/Equality

- **Definition/Purpose:** `let` / `const` / `var` ใช้ประกาศตัวแปร; ชนิดข้อมูลพื้นฐาน (primitive) คือ String, Number, Boolean, null, undefined; object (รวม array, function) เป็นอีกกลุ่ม
- **Core Concept (จุดสำคัญที่สุดของบทนี้ ตามเอกสารทบทวนบทที่ 1): primitive vs object คือความต่างเรื่อง "ค่า" กับ "ที่อยู่"** — primitive ก็อปปี้ใส่ตัวแปรใหม่ได้ค่าอิสระ ส่วน object ก็อปปี้ได้แค่ "ที่อยู่" ตัวแปรสองตัวเลยชี้ไปก้อนเดียวกัน (**เป็น prerequisite ของทั้ง Array/Object/Destructuring/Spread ทั้งหมด**)
- **How it works:** `const` ล็อกที่ "ป้ายชื่อ" (ตัวแปร) เท่านั้น ไม่ได้ล็อก property ข้างในของ object — จึงยังแก้ property ของ object ที่ประกาศด้วย `const` ได้

- **Syntax ที่ต้องรู้:** `typeof x`, `let` / `const` / `var`, ค่า falsy 6 ตัว (ตามเอกสาร: `false, 0, "", null, undefined, NaN`), `[]` **และ** `{}` **เป็น truthy เสมอแม้ว่างเปล่า** (จุดหลอกตาที่เอกสารเน้นเป็นพิเศษ)
- **Important Rules:**
  - ใช้ `===` เป็นค่าเริ่มต้นเสมอ (ไม่ใช่ `==`)
  - `NaN !== NaN` แม้ใช้ `===` — ต้องเช็คด้วย `Number.isNaN(x)` เท่านั้น ห้ามเขียน `x === NaN` เด็ดขาด
  - `a == x == b` **ไม่ใช่** syntax เทียบช่วงแบบคณิตศาสตร์ — ตัวที่สองเทียบกับผล boolean ของตัวแรก ต้องแยกเขียนด้วย `&&`
- **Common Mistakes:** เข้าใจผิดว่า `ค่า === NaN` เช็ค NaN ได้; เข้าใจผิดว่า `a == x == b` แปลว่า "x อยู่ระหว่าง a กับ b"
- **Related Concepts:** Reference vs Copy (A8), Conditionals (A3)
- **Prerequisites:** ไม่มี (จุดเริ่มต้นของวิชา)
- **Next Concepts:** Conditionals, Function, ทุกหัวข้อที่ใช้ค่า
- **Required Knowledge Level:** จำ (falsy list, `===` vs `==`) + เข้าใจ (primitive vs object) — **เหมาะกับ Theory มาก**

A3. Conditionals (if/else if/else, switch, guard clause)

- **Definition/Purpose:** ควบคุมทางเดินโปรแกรมตามเงื่อนไข
- **Core Concept:** โปรแกรมเช็คจากบนลงล่างเสมอ เจอทางแรกที่จริงจะเข้าทันที (ไม่เช็คทางที่เหลือ) — ถ้าเรียงเงื่อนไข "แคบ" ไว้หลัง "กว้าง" เคสพิเศษจะหลุดเข้าไปในทางที่ผิด
- **Syntax:** `if/else if/else`, `switch(x){case:...break; default:}` — switch ใช้ **strict equality (===)** เทียบเสมอ, `break` กัน fall-through, `default` มีหรือไม่มีก็ได้
- **Important Rules:**
  - Switch เหมาะกับ: เทียบตัวแปรเดียวกับค่าคงที่หลายค่า (enum-like), case ≥ 3–4 กรณี
  - if/else เหมาะกับ: เงื่อนไขช่วง (range), เทียบหลายตัวแปรร่วมกัน (`&&`, `||`), เงื่อนไขซับซ้อน
  - `&&` (AND) ต้องจริงทุกข้อ, `||` (OR) จริงข้อเดียวพอ — สับสนบ่อยเพราะภาษาไทยใช้ "และ/หรือ" กำกวม
- **Code Interpretation ตัวอย่างจากสไลด์:** `getOrderStatusLabel(status)` ใช้ switch คืน label ตามสถานะคำสั่งซื้อ
- **Common Mistakes:** ลืมเรียงเงื่อนไขแคบ→กว้าง; ลืม `break` ทำให้ fall-through; เช็คค่า "ว่าง" ไม่ตรงกับที่โจทย์ถาม (length vs สตริงว่างเปล่า vs null/undefined เป็นคนละเรื่องกัน)
- **Related:** Loop (break/continue คำเดียวกันแต่บริบทต่าง), Truthy/Falsy
- **Prerequisites:** A1–A2
- **Next:** Loop, Function
- **Required Knowledge Level:** เข้าใจ + เขียนได้ — **ออกได้ทั้ง Theory (อ่าน output) และ Practical (เขียนจริง)**

A4. Loop (for, while, do...while, for...of, for...in, break/continue)

- **Definition/Purpose:** ทำงานซ้ำตามจำนวนรอบ/เงื่อนไข
- **Core Concept (จุดพลาดบ่อยที่สุดของบทนี้ ตามเอกสารทบทวนบทที่ 3):** สับสนระหว่าง **"สมาชิกตัวที่กำลังวนอยู่" (เช่น `item`)** กับ **"array ทั้งก้อน" (เช่น `items`)** — พิมพ์ `cartItems.price` แทน `item.price` ใน `for...of` จะได้ `undefined` ทุกรอบ **โดยไม่ error**
- **How it works:** for มี 3 ส่วนทำงานคนละจังหวะ — initialization (ครั้งเดียว) → condition (เช็คก่อนทุกรอบ) → increment (หลังจบแต่ละรอบ)
- **Syntax:**

ชนิด	ใช้เมื่อ	Syntax
`for`	รู้จำนวนรอบแน่นอน	`for(let i=0;i<5;i++){}`
`while`	ไม่รู้จำนวนรอบ รู้เงื่อนไขจบ	`while(cond){}`
`do...while`	ต้องทำอย่างน้อย 1 ครั้งก่อนเช็ค	`do{}while(cond);`
`for...of`	วนอ่าน "ค่า" ใน Array/String/Map/Set	`for(const v of arr){}`
`for...in`	วนอ่าน "key" ใน Object	`for(const k in obj){}`

- **Important Rules:**
  - **ห้ามใช้ `for...in` กับ Array** เพราะจะได้ index เป็น **string** (`"0","1"`) ไม่ใช่ตัวเลข — ใช้ `for...of` แทน

- `while(false ตั้งแต่แรก)` จะไม่ทำงานเลย แต่ `do...while` ทำงานอย่างน้อย 1 ครั้งเสมอ
- ตัวแปรสะสมค่า (accumulator) ต้องประกาศ**นอกลูป** ไม่งั้นถูกรีเซ็ตทุกรอบ
- `break` ออกจากลูปทันที, `continue` แค่ข้ามรอบปัจจุบัน

- **Common Mistakes:** ใช้ชื่อ array ทั้งก้อนแทนตัวแปรที่วนอยู่; ประกาศตัวแปรสะสมไว้ในลูป
- **Related:** Array methods (forEach เป็นทางเลือกแทน loop), Conditionals
- **Prerequisites:** A1–A3
- **Next:** Function, Array
- **Required Knowledge Level:** จำ (syntax ของ 5 ชนิด) + เข้าใจ (for...in กับ array) + Debug (หา bug จากการสลับ item/items) — **ออกได้ทั้ง Theory และ Practical**

A5. Function (Declaration/Expression/Arrow, parameter/return, scope)

- **Definition/Purpose:** กลุ่มคำสั่งที่ตั้งชื่อไว้ ทำงานเมื่อถูกเรียก รับค่าเข้า (parameter) คืนค่าออก (return)
- **Core Concept (จากเอกสารทบทวนบทที่ 4):** ฟังก์ชันคือ "สัญญา" ระหว่างคนเรียกกับคนทำงาน — ผิดทางส่งค่าออกนิดเดียว (เช่น `console.log` แทน `return`) งานที่คำนวณถูกทุกตัวอักษรก็เป็นศูนย์คะแนนได้
- **Syntax — 3 รูปแบบ (ให้ผลลัพธ์เหมือนกัน):**

```
function add(a,b){ return a+b }              // Declaration — hoisting ได้ (เรียกก่อนประกาศได้)const add2 = function(a,b){ return a+b }      // Expressionconst add3 = (a,b) => a+b // Arrow — นิยมในโค้ดสมัยใหม่
```

- **Important Rules:**
  - `console.log` ≠ `return` — พิมพ์อย่างเดียวไม่ส่งค่าออกไปให้คนเรียกใช้ต่อได้
  - ฟังก์ชันไม่มี `return` จะคืน `undefined`
  - **Local scope** (ในฟังก์ชัน) vs **Global scope** (นอกฟังก์ชัน)
  - **Default parameter ทำงานเฉพาะตอนได้ `undefined` เท่านั้น** — ส่ง `null` / `""` / `0` เข้าไปจะ **ไม่** ทำให้ default เข้ามาแทนที่ (จุดพลาดบ่อยที่สุดของหัวข้อนี้)
  - Arrow function ไม่มี `this` / `arguments` ของตัวเอง, เรียกด้วย `new` ไม่ได้, ไม่ hoist
  - **Callback** = การยื่นชื่อฟังก์ชันเปล่า ๆ ไม่ใส่วงเล็บเรียก — ใส่วงเล็บ `()` ต่อท้ายจะกลายเป็นเรียกฟังก์ชันทันที แล้วเอา "ผลลัพธ์" ไปเป็น callback แทน (เช่น `products.filter(isAvailable())` ผิด ต้องเป็น `products.filter(isAvailable)`)
- **Common Mistakes:** ใส่วงเล็บเรียกฟังก์ชันไปพร้อมกับส่งเป็น callback; คิดว่า default parameter รับมือ `null` / `0` ได้ด้วย
- **Related:** Scope เชื่อมกับ Object/Array reference, Array method (map/filter/reduce ล้วนรับ callback)
- **Prerequisites:** A1–A4
- **Next:** Array, Object, Module
- **Required Knowledge Level:** จำ (3 รูปแบบ, default param rule) + เข้าใจ (scope, callback) + เขียน + Debug — **ออกได้ทั้ง 2 ส่วน สูงมาก**

A6. Array & Array Methods

- **Definition/Purpose:** ชุดข้อมูลเรียงลำดับ เข้าถึงด้วย index (เริ่ม 0), เช็คจำนวนด้วย `.length`
- **Core Concept (จากเอกสารทบทวนบทที่ 5 — สำคัญที่สุดของหัวข้อนี้):** ก่อนเรียก method ต้องถามเสมอว่า **"แก้ของเดิม (mutating) หรือคืนของใหม่ (non-mutating)"**
- **ตารางที่ต้องจำ (จากเอกสาร):**

Method	หมวด	คืนอะไร	แก้ต้นฉบับ?
`push(x)`	แก้ต้นฉบับ	ความยาวใหม่ (ตัวเลข)	✅ เปลี่ยน
`pop()`	แก้ต้นฉบับ	สมาชิกตัวสุดท้ายที่ถูกดึงออก	✅ เปลี่ยน
`sort()`	แก้ต้นฉบับ	array เดิม (เรียงแล้ว)	✅ เปลี่ยน
`map()`	คืนของใหม่	array ใหม่	❌ไม่เปลี่ยน
`filter()`	คืนของใหม่	array ใหม่ (กรองแล้ว)	❌ไม่เปลี่ยน

Method	หมวด	คืนอะไร	แก้ต้นฉบับ?
`reduce()`	คืนของใหม่	ค่าสะสม (ชนิดใดก็ได้)	❌ไม่เปลี่ยน
`find()`	คืนของใหม่	สมาชิกแรกที่ตรงเงื่อนไข	❌ไม่เปลี่ยน

- **Important Rules:**
  - `sort()` **ไม่ใส่ comparator จะเรียงแบบ string เสมอ** — ตัวเลขจึงเรียงเพี้ยนได้ง่าย (เช่น `[10,2,1]` → `[1,10,2]`) ต้องใส่ `(a,b)=>a-b` เพื่อเรียงเป็นตัวเลขจริง
  - `reduce()` **ต้องมี initial value เสมอ** — ถ้าไม่ใส่ JS จะหยิบสมาชิกตัวแรกมาตั้งเป็นถังแทนโดยไม่บอก และถ้า array ว่างเปล่าจะ error
  - จัดกลุ่มด้วย reduce ต้องเช็ค `groups[key] ??= []` **ก่อน** `.push()` เสมอ ไม่งั้น push เข้า `undefined` จะพัง
- **Common Mistakes:** push เข้า `groups[key]` ก่อนสร้าง array รองรับ; ไม่ใส่ comparator ใน sort ตัวเลข; ไม่ใส่ initial value ใน reduce
- **Related:** Loop (for...of เป็นทางเลือก), Function (callback ของ map/filter/reduce), Destructuring
- **Prerequisites:** A1, A5
- **Next:** Object, Destructuring/Spread
- **Required Knowledge Level:** จำ (ตารางแก้ต้นฉบับ/ไม่แก้) + เข้าใจ + เขียน + Debug — **หัวข้อ High-Value ทั้ง Theory และ Practical**

A7–A8. Object & Reference vs Copy

- **Definition/Purpose:** ชุดข้อมูลแบบ key-value; **property** = ข้อมูล, **method** = ฟังก์ชันในอ็อบเจกต์, `this` อ้างถึงตัวอ็อบเจกต์นั้นเอง
- **Core Concept (จากเอกสารทบทวนบทที่ 6 — เชื่อมกับ A1-A2 โดยตรง): ตัวแปรที่เก็บ object ไม่ได้ "ถือ" ข้อมูลไว้ — มันแค่ "ชี้" ไปหาก้อนข้อมูลอีกที** → `const b = a` (a เป็น object) **ไม่ได้สร้างสำเนาใหม่** ทั้งคู่ยังชี้ก้อนเดียวกัน แก้ทางหนึ่งอีกทางเห็นการเปลี่ยนทันที
- **Syntax:** dot notation `obj.key` (รู้ชื่อ key แน่นอน) vs bracket notation `obj["key"]` (key มาจากตัวแปร หรือมีอักขระพิเศษ) — เขียนชื่อ key ตรง ๆ ใน bracket ต้องมี quote เสมอ; เช็คว่ามี key จริงด้วย `in` หรือ `?.` (optional chaining)
- **Important Rules:**
  - **สะกด key ผิดตอนแก้ค่า → JavaScript ไม่ error แต่สร้าง property ใหม่เงียบ ๆ** ค่าเดิมไม่ถูกแตะเลย (debug ด้วย `Object.keys(obj)` ดูว่ามี key แปลกโผล่มาไหม)
  - วน object ได้ 3 วิธี (for...in, `Object.keys()`, `Object.entries()`) ผลลัพธ์เหมือนกันแม้หน้าตาต่าง
  - `obj?.a?.b` ปลอดภัยถ้า `a` ไม่มีจริง (ไม่ throw error) ต่างจาก `obj.key === undefined` ที่แยกไม่ออกระหว่าง "ไม่มี key นี้เลย" กับ "มี key แต่ค่าเป็น undefined"
- **Common Mistakes:** สะกด key ผิดตอนแก้ค่า (ไม่มี error เตือน); เข้าใจผิดว่า `const b = a` copy ให้
- **Related:** Array (เป็น object ชนิดหนึ่ง), Spread (shallow copy), Destructuring
- **Prerequisites:** A1–A2
- **Next:** Spread/Destructuring
- **Required Knowledge Level:** จำ + **เข้าใจลึก (สำคัญที่สุด — MDN ช่วยไม่ได้ถ้าไม่เข้าใจ concept นี้)** — **หัวข้อที่ MDN ช่วยไม่ได้ ต้องแม่นสำหรับทั้ง Theory และ Practical**

A9–A10. Destructuring & Spread/Rest

- **Definition/Purpose:** Destructuring = แกะค่าจาก object/array ออกมาเป็นตัวแปรอย่างรวดเร็ว; Spread (`...`) = กระจายสมาชิกออก, Rest (`...`) = รวบรวมเข้า (**หน้าตาเหมือนกันแต่ทำหน้าที่ตรงข้าม**)
- **Core Concept:** Array destructuring แกะ**ตามตำแหน่ง** ส่วน Object destructuring แกะ**ตามชื่อ key**
- **Syntax:**

```
const [a, , c] = [1,2,3];              // ข้ามตำแหน่งที่ 2 ด้วย ,const [p = 0, q = 0] = [5];
// default ถ้าตำแหน่งว่าง → p=5, q=0const { name: fullName } = user;        // renameconst
{ country = "ไทย" } = user;       // defaultfunction Card({ name, price }) { ... }  //
```

```
destructure ตรง parameterconst merged = { ...defaults, ...userPrefs }; // spread รวม ob
jectfunction sum(...numbers) { ... }        // rest — รับ argument ไม่จำกัดจำนวน
```

- **Important Rules:**
  - **Arrow function ที่ destructure parameter ต้องครอบด้วย `()` เสมอ** เช่น `({name, price}) => ...` ไม่งั้น JS งงว่า `{}` คือ destructuring pattern หรือจุดเริ่ม function body → **SyntaxError ทันที**
  - Spread merge object: **key ที่ซ้ำกัน ค่าจาก object "ตัวหลัง" ทับตัวก่อนหน้าเสมอ**
  - **Spread คัดลอกได้แค่ชั้นบนสุด (shallow copy)** — ถ้า object มีของซ้อนกันหลายชั้น (nested object) การแก้ผ่านสำเนาที่ spread มาจะ**กระทบต้นฉบับด้วย** เพราะชั้นในยังใช้ reference เดิม (ต้องใช้ nested spread หรือ `structuredClone()` ถ้าต้องการ deep copy)
- **Common Mistakes:** ลืมวงเล็บครอบ parameter destructure ใน arrow function; แก้ของซ้อนผ่านสำเนา spread แล้วงงว่าทำไมต้นฉบับเปลี่ยนตาม (shallow copy trap)
- **Related:** A7-A8 (Reference vs Copy คือรากของปัญหานี้), Array/Object
- **Prerequisites:** A6, A7-A8
- **Next:** Date, API response handling
- **Required Knowledge Level:** จำ Syntax + **เข้าใจลึก (shallow copy)** + เขียน — **จุดพลาดยอดฮิตทั้ง Theory (ทายผลลัพธ์) และ Practical**

A11. JavaScript Date

- **Definition/Purpose:** จัดการวันที่/เวลาในหน้าเว็บ
- **Core Concept (จากเอกสารทบทวนบทที่ 8): เบื้องหลัง Date ทั้งก้อนคือตัวเลขตัวเดียว (มิลลิวินาทีนับจาก epoch)** — พอจับหลักนี้ได้ สูตรคำนวณวันที่จะกลายเป็นเลขคณิตธรรมดา
- **Syntax:**Getters: `getFullYear()`, `getMonth()` (0-11), `getDate()` (1-31), `getDay()` (0=อาทิตย์), `getHours()/getMinutes()` แสดงผล: `toLocaleString('th-TH')`, `toLocaleDateString('th-TH')`, `toISOString()`

```
new Date();                     // ปัจจุบันnew Date(2026, 7, 23);          // ปี, เดือน(0-1
1!), วัน — เดือนนับจาก 0new Date("2026-08-23");         // จาก ISO string — ต้องมี quoteDat
e.now();                    // timestamp ตัวเลข ไม่ใช่ Date object
```

- **Important Rules:**
  - **การลบ Date สองตัวได้ผลต่างเป็น "มิลลิวินาที" เสมอ** ต้องหารด้วย `1000*60*60*24` เพื่อแปลงเป็นวัน — **ต้องมีวงเล็บครอบตัวหารทั้งก้อนเสมอ** (`diffMs/(1000*60*60*24)` ไม่ใช่ `diffMs/1000*60*60*24` เพราะ `/` กับ `*` มีลำดับเท่ากันและอ่านซ้ายไปขวา ผลต่างกันเป็นล้านเท่า)
  - `Date.now()` **ไม่ใช่ Date object** เรียก `.getMonth()` ต่อไม่ได้ทันที ต้องห่อด้วย `new Date(Date.now())`
  - Date ไม่มี `addDays()` ตรง ๆ ต้องใช้ `setDate(getDate()+n)`
  - เทียบ Date ด้วย `>` / `<` ได้ แต่ `===` ใช้ไม่ได้ (เพราะเป็น object คนละก้อน)
- **Common Mistakes:** ลืมวงเล็บครอบตัวหาร; คิดว่า `Date.now()` เป็น Date object
- **Related:** Object (Date เป็น object ชนิดหนึ่ง)
- **Prerequisites:** A1
- **Next:** Regex
- **Required Knowledge Level:** จำ (getters, เดือนเริ่ม 0) + เข้าใจ (ms arithmetic) — **Theory สูง (คำนวณ/ทายผลลัพธ์), Practical ปานกลาง (MDN ช่วยเรื่อง method name ได้ แต่สูตรคำนวณต้องเข้าใจเอง)**
- ⚠️ **หมายเหตุพิเศษ: Temporal API (Date เวอร์ชันใหม่) มีหลักฐานตรงจากอาจารย์ว่ายังไม่ออกสอบ Exam-01** — ไม่ต้องเน้น

A12. Regular Expression (Regex)

- **Definition/Purpose:** pattern สำหรับค้นหา/ตรวจสอบ/แทนที่ข้อความ
- **Core Concept (จากเอกสารทบทวนบทที่ 9):** "อ่านออกเสียงก่อนเขียน" — regex เป็นแค่สัญลักษณ์ไม่กี่ตัวประกอบกัน อ่านทีละชิ้นจะไม่น่ากลัว

- **Syntax:** เขียนด้วย `/pattern/flags` หรือ `new RegExp(str, flags)`

สัญลักษณ์	ความหมาย	ตัวอย่าง
`\d`	ตัวเลข 0-9	`/\d+/` → "123"
`\w`	ตัวอักษร/เลข/underscore	`/\w+/`
`\s`	ช่องว่าง	
`^` `$`	ปักหมุดต้น/ท้ายข้อความ	`/^A/` , `/z$/`
`* + ? {n,m}`	จำนวนซ้ำ	`/\d{3}/` → เลข 3 หลักพอดี
flag `g`	หาทุกจุด ไม่ใช่แค่จุดแรก	
flag `i`	ไม่สนตัวพิมพ์เล็ก-ใหญ่	
Method: `regex.test(str)` →boolean, `str.match(regex)` →array/null, `str.replace(regex,newStr)`, `str.split(regex)`		

- **Important Rules:**
  - `match()` **ไม่คืน true/false เหมือน** `test()` — จับได้คืน **array** (แม้จับได้แค่ 1 อัน), จับไม่ได้คืน `null`
  - `.` **ใน regex แปลว่า "ตัวอักษรอะไรก็ได้ 1 ตัว" ไม่ใช่จุดจริง** — ต้อง escape ด้วย `\.` ถ้าต้องการจุดจริง (เช่น ก่อนนามสกุลไฟล์/อีเมล)
- **Common Mistakes:** เข้าใจผิดว่า match คืน boolean; ลืม escape จุด
- **Code Example (จากสไลด์):** highlight คำค้นหาในผลลัพธ์ด้วย `replace()` + regex แบบ global+ไม่สนตัวพิมพ์ ครอบด้วย `<mark>`
- **Related:** String methods, Form validation
- **Prerequisites:** A1
- **Next:** Exception Handling
- **Required Knowledge Level:** จำสัญลักษณ์ + เข้าใจ (test vs match) — **Practical สูง (MDN ช่วยเรื่อง syntax ได้มาก แต่การอ่าน pattern ทีละชิ้นต้องฝึกมาก่อน), Theory ปานกลาง-สูง (หายผลลัพธ์ pattern)**

A13. Exception Handling (try/catch/finally, throw, custom Error)

- **Definition/Purpose:** ดักข้อผิดพลาดไม่ให้แอปพลิเคชันพังทั้งไฟล์
- **Core Concept (จากเอกสารทบทวนบทที่ 10):** ข้อมูลจากผู้ใช้ไม่เคยสวยเป๊ะเหมือนตัวอย่างในห้องเรียน — try/catch คือด่านสุดท้ายกันไม่ให้ error เล็ก ๆ ลากทั้งไฟล์ล้ม
- **Syntax:**

```
try { /* โค้ดที่อาจ error */ }catch (error) { /* ทำงานเมื่อเกิด error ใน try เท่านั้น */ }finally { /* ทำงานเสมอ ไม่ว่าจะ error หรือไม่ */ }class ApiError extends Error {  constructor(message, status) {    super(message);           // ต้องเรียกก่อนเสมอ    this.name = "ApiError";    this.statusCode = status;  }}
```

- **Important Rules:**
  - **เขียนได้แบบเดียวที่ไม่มี** `catch` : `try{}finally{}` (ไม่ดัก error แต่ finally ยังการันตีว่าทำงานแน่ ๆ) — เขียน `try` เดี่ยว ๆ ไม่มีทั้ง catch/finally, หรือ `catch` เดี่ยว ๆ ไม่มี try นำหน้า **ล้วนเป็น syntax ผิด**
  - `catch` **จับทุก error รวมถึง bug ของเราเองด้วย** (เช่น พิมพ์ชื่อตัวแปรผิดในบรรทัดที่อยู่ใน try ก็โดนจับเงียบ ๆ เหมือนกัน) — ต้องระวังไม่ให้ catch "กลืน" บั๊กจริงจนหาไม่เจอ
  - `throw` หยุดทำงานทันทีส่งไปให้ catch จับ, `return` แค่คืนค่าเฉย ๆ — **คนละอย่างกัน**
  - Custom Error class ต้องเรียก `super(message)` เสมอ แล้วใช้ `instanceof` แยกชนิด error
- **Common Mistakes:** เขียนโครง try/catch/finally ผิด syntax; คิดว่า catch จับเฉพาะ error ที่ตั้งใจดัก
- **Related:** Debugging (A15), Function

- **Prerequisites:** A5, A7
- **Next:** Debugging
- **Required Knowledge Level:** จำโครง syntax (เขียนได้โดยไม่เปิดตัวอย่าง) + เข้าใจ — **ออกได้ทั้ง 2 ส่วน**

A14. JavaScript Module (CommonJS / ES Module)

- **Definition/Purpose:** แยกโค้ดเป็นไฟล์ย่อยเพื่อดูแลง่าย ทดสอบง่าย ใช้ซ้ำได้; แต่ละไฟล์ `.js` คือ 1 module มีขอบเขตตัวแปรของตัวเอง
- **Syntax:**

```
// CommonJSmodule.exports = { greeting, greetWords }const { greeting } = require('./greeting.js')// ES Module — Named Export (ได้หลายค่า/ไฟล์)export function formatPrice(n){...}export const TAX_RATE = 0.07;import { formatPrice, TAX_RATE } from "./utils/formatPrice.js";// ES Module — Default Export (ได้ 1 ค่า/ไฟล์)export default function Button({label}){...}import Button from "./components/Button.js";   // ตั้งชื่ออะไรก็ได้ ไม่ต้องมี {}// รูปแบบ import ที่ใช้บ่อยimport { formatPrice as toBaht } from "...";     // ตั้งชื่อใหม่ (as)import * as priceUtils from "...";              // namespace importimport Button, { ButtonGroup } from "...";       // ผสม default + named
```

- **Important Rules:**
  - Named import **ต้องใช้ชื่อเดียวกัน** กับตอน export เสมอ (อยู่ใน `{}`)
  - Default export **มีได้แค่ 1 ค่า/ไฟล์** ตั้งชื่อตอน import เป็นอะไรก็ได้ (ไม่ต้อง `{}`)
  - 1 ไฟล์มีทั้ง default + named export ผสมกันได้
- **Common Mistakes:** สะกดชื่อ named import ไม่ตรงกับ export; ใส่ `{}` กับ default import
- **Related:** Function (สิ่งที่ export ส่วนใหญ่คือ function)
- **Prerequisites:** A5
- **Next:** (จบสาย JS Fundamentals)
- **Required Knowledge Level:** จำ syntax (named vs default) — **Theory สูง (ความต่าง named/default), Practical ปานกลาง (MDN ช่วยเรื่อง syntax ได้เกือบทั้งหมด)**

A15. การอ่าน Error Message / Debugging

- **Core Concept (จากเอกสารทบทวนบทที่ 11):** error message บอกทั้ง **"อะไรผิด"** และ **"ผิดที่ไหน"** (ชื่อไฟล์+เลขบรรทัด) เสมอ — อ่าน**บรรทัดบนสุด**ของข้อความ error ก่อน เพราะอาจมี error ซ้อนกันหลายบรรทัด บรรทัดบนสุดมักตรงประเด็นที่สุด
- **Important Rules:**
  - `undefined` **มาได้ 3 ทาง:** (1) ตัวแปรไม่มีค่า (2) ฟังก์ชันไม่ `return` (3) เข้าถึง property ที่ไม่มี
  - **ผ่านบางเทสไม่ได้แปลว่าใกล้ถูก** — แต่ละเทสตรวจคนละแง่มุม ไม่ได้เรียงน้ำหนักเท่ากัน
  - **เช็ก 3 จุดก่อนส่งคำตอบเสมอ:** ① มี return/พิมพ์ผลหรือยัง ② ข้อความตรง spec เป๊ะไหม (รวมถึงช่องว่าง) ③ สะกดชื่อ (ตัวแปร/property) ตรงไหม
- **Common Mistakes:** คิดว่าผ่าน 2/3 เทส = เกือบถูกแล้ว (เคสที่ยังไม่ผ่านอาจสำคัญที่สุด)
- **Related:** Exception Handling (A13)
- **Prerequisites:** ทุกหัวข้อ A (เป็น meta-skill)
- **Required Knowledge Level: เป็นทักษะ Practical ล้วน ๆ — สำคัญมากตอนสอบแลป เพราะมีเวลาจำกัด 2 ชม.**

B. Web Development Fundamentals

- **B1. Web Architecture:** การกำหนดว่าเว็บแอปพลิเคชันแบ่งส่วน/ทำงานร่วมกันอย่างไร เพื่อคุณภาพ 4 ด้าน: **Speed, Security, Scalability, Maintainability**; **3-Tier Architecture** = Presentation Tier – Application Tier – Data Tier
- **B2. Client-Server:** Client (Browser) โหลด HTML/CSS/JS, render UI, ฟัง event, เรียก Fetch API, อัปเดต DOM / Server (Backend) รับ HTTP request, ทำ business logic+auth, อ่าน/เขียน DB, ส่งกลับเป็น JSON — Client เป็นฝ่ายร้องขอ (Request) เสมอ, แยกพัฒนา/deploy อิสระตราบใดที่ interface เดิม

- **B3. HTTP Methods:** `GET` (ขอข้อมูล/อ่าน ไม่เปลี่ยนแปลง) `POST` (สร้างใหม่) `PUT` (แก้ไขทั้งก้อน) `DELETE` (ลบ)
- **B4. HTTP Status Code:** `2xx` สำเร็จ(200 OK,201 Created) `3xx` เปลี่ยนเส้นทาง(301,304) `4xx` ผิดจาก Client(400,401,404) `5xx` ผิดจาก Server(500,503)
- **B5. Stateless:** Server ไม่จำข้อมูล request ก่อนหน้า — ทุก request ต้องมีข้อมูลครบในตัวเอง ข้อดี: scale ง่าย, request ล้มเหลวไม่กระทบ request อื่น; Login ค้างได้เพราะ Client แนบข้อมูลยืนยันตัวตนเองผ่าน **Cookie (Session ID)** หรือ **Token (เช่น JWT)**
- **B6. REST:** มีแค่หัวข้อในสไลด์ที่อ่านได้ ไม่มีเนื้อหาขยายความ — **ไม่สามารถยืนยันรายละเอียดเพิ่มเติมได้จากเอกสาร**
- **Common Mistakes:** สับสน PUT (แก้ทั้งก้อน) กับการแก้บางส่วน (PATCH — ไม่ปรากฏในเอกสารที่อ่านได้); เข้าใจผิดว่า Stateless แปลว่า login ไม่ได้
- **Required Knowledge Level:** จำ (HTTP method/status code) + เข้าใจ (client-server, stateless) — **เหมาะกับ Theory เป็นหลัก เพราะเป็นความรู้เชิงแนวคิดมากกว่าการเขียนโค้ด**

C. DOM (Document Object Model)

- **C1. DOM คืออะไร:** แทน HTML page เป็น **hierarchical tree of nodes** ให้โปรแกรมเปลี่ยนโครงสร้าง/สไตล์/เนื้อหาได้; cross-platform, language-independent; ไม่ใช่ภาษาโปรแกรม แต่ JS ต้องพึ่งมันจึงจะจัดการหน้าเว็บได้
- **C2.** `<script>` **Tag Placement:** วางใน `<head>` → JS ทั้งหมดต้องโหลด/parse/interpret ก่อนหน้าเริ่ม render (browser จะขาวว่างระหว่างรอ) → วางท้าย `<body>` (แนวทางสมัยใหม่) เพื่อลดการหน่วงการ render
- **C3. window/document:** `document` เป็น property ของ `window`, เป็น root node, read-only, `document.documentElement` = `<html>`, `document.head` / `document.body`
- **C4. Node Types:** ทุก node มี `nodeName`, `nodeValue`, `nodeType` — Node type เป็นค่าคงที่ตัวเลข 12 แบบ (เช่น `Node.ELEMENT_NODE`) ใช้เช็คด้วย `if(node.nodeType === Node.ELEMENT_NODE)`
- **C5. Element/Attr:** `document.createElement(tagName)` สร้าง element ใหม่ (tag name ไม่สนตัวพิมพ์เล็ก-ใหญ่ใน HTML); เพิ่มเข้า tree ด้วย `appendChild()/insertBefore()/replaceChild()/removeChild()` ; **Attr node สืบทอดจาก Node แต่ไม่ถือเป็นส่วนของ document tree** — `parentNode` / `previousSibling` / `nextSibling` เป็น `null` เสมอสำหรับ Attr, ใช้ `ownerElement` แทน; ดู attribute ทั้งหมดด้วย `getAttributeNames()` + `getAttribute(name)`
- **C6. innerHTML vs innerText vs textContent:**
  - `innerHTML` → คืน HTML markup จริง (รวม tag)
  - `innerText` → คืนเฉพาะข้อความที่ "มองเห็นจริง" (ไม่รวมของที่ `display:none`)
  - `textContent` → คืนข้อความทั้งหมดรวมของที่ซ่อนอยู่ด้วย (ต่างจาก innerText)
- **C7. Traversing Nodes:** `someNode.firstChild === someNode.childNodes[0]` เสมอ, `lastChild === childNodes[length-1]` เสมอ; **Element Traversal API** (เวอร์ชัน element-only ไม่รวม text/comment): `childElementCount`, `firstElementChild`, `lastElementChild`, `previousElementSibling`, `nextElementSibling`
- **C8. Selecting Nodes:** `getElementById()`, `getElementsByTagName()` (HTMLCollection, live), `getElementsByClassName()` (live), `querySelector()` (ตัวแรกที่ตรง CSS selector), `querySelectorAll()` (ทั้งหมด, NodeList static), `element.closest(selector)` (ค้นหาขึ้นไปหา parent ที่ตรง selector จนถึง document, คืน `null` ถ้าไม่เจอ)
- **C9. HTMLCollection (live) vs NodeList (static):**

	HTMLCollection	NodeList
ได้จาก	`getElementsByTagName/ClassName`	`querySelectorAll`, `childNodes`
อัปเดตอัตโนมัติเมื่อ DOM เปลี่ยน?	✅ **live**	❌ ส่วนใหญ่ **static**
เก็บ	element nodes เท่านั้น	ทุก node (element/attr/text)

- **C10. Manipulating Nodes:** `appendChild()` (เพิ่มท้าย), `createElement()`, `insertBefore(new, ref)` (new กลายเป็น previous sibling ของ ref), `replaceChild(new, old)`, `removeChild(child)` (คืน node ที่ถูกลบ), `createAttribute()`, `createTextNode()`
- **C11. DocumentFragment:** เป็น "พื้นที่พักชั่วคราว" ในหน่วยความจำ ไม่ใช่ส่วนของหน้าจริง — สร้าง element ทั้งหมดใส่ fragment ก่อน แล้ว appendChild ทีเดียว → **Reflow/Repaint เกิดแค่ครั้งเดียว** แทนที่จะเกิดทุกรอบ loop (performance optimization)
- **C12. dataset:** อ่าน/เขียน custom attribute `data-*` ผ่าน `element.dataset` (คืน `DOMStringMap`); ชื่อ attribute แปลงจาก kebab-case (`data-user-status`) เป็น camelCase (`dataset.userStatus`) อัตโนมัติ

- **C13. classList:** อ่านอย่างเดียว (read-only property) คืน `DOMTokenList`; method: `add()`, `remove()`, `toggle(className, force)`, `contains()`, `replace()`
- **C14. System Dialogs (alert/confirm/prompt):** synchronous + modal (โค้ดหยุดรอจนกว่าจะปิด dialog), ไม่เกี่ยวกับหน้าเว็บ ไม่มี HTML, หน้าตากำหนดโดย OS/browser ไม่ใช่ CSS — `alert()` รับ string อย่างเดียว, `confirm()` คืน true/false, `prompt()` คืนข้อความในกล่อง หรือ `null` ถ้ากด Cancel — ⚠️ **มีหลักฐานตรงจากอาจารย์ว่าหัวข้อนี้ไม่ออกสอบและไม่สำคัญมาก**
- **Common Mistakes (สรุปจากเนื้อหา DOM):** สับสน `innerText` (ไม่รวมของซ่อน) กับ `textContent` (รวมของซ่อน); เข้าใจผิดว่า NodeList ทุกชนิด live เหมือน HTMLCollection; ลืมว่า `Attr` ไม่ได้อยู่ใน document tree จริง
- **Related:** JavaScript Object/Array fundamentals ทั้งหมด (DOM API เป็นแค่การเอา object/method ไปจัดการหน้าเว็บ)
- **Prerequisites:** A1 (Object), A6 (Array-like: HTMLCollection/NodeList), A7 (property/method access)
- **Required Knowledge Level:** จำ method/property ชื่อยาว ๆ (MDN ช่วยได้มากส่วนนี้) + **เข้าใจ concept ที่ MDN ช่วยไม่ได้ (live vs static, tree structure, reflow/repaint) — Practical สูงมาก (มักออกเป็นโจทย์เขียน/แก้โค้ดจัดการหน้าเว็บ), Theory ปานกลาง-สูง (ความต่างของคู่คำที่คล้ายกัน)**

PHASE 5 (Final Output #5): CONCEPT RELATIONSHIPS & DEPENDENCY MAP

```
A1 (Variable/Type, Primitive vs Object)
  │  ← รากฐานของทุกอย่าง ถ้าไม่เข้าใจ primitive-vs-object จะงงทั้ง A7,A8,A9,A10
  ├─→ A2 (Truthy/Falsy/Equality) ─→ A3 (Conditionals)
  │                                  │
  │                                  ├─→ A4 (Loop) ─┬─→ A6 (Array & Methods)
  │                                  │              │        │
  │                                  └─→ A5 (Function) ←──────┤ (callback ใช้ร่วมกับ map/filter/reduce)
  │                                       │                   │
  │                                       ↓                   ↓
  ├─→ A7 (Object) ──────────────────→ A8 (Reference vs Copy) ─→ A9-A10 (Destructuring/Spread)
  │       │                                                          │
  │       └──────────────────────────────────────────────────────────┘ (shallow copy trap ต้องเข้าใจ A8 ก่อน)
  │
  ├─→ A11 (Date) — ใช้ A1 (ตัวเลข/object พื้นฐาน) ไม่ต้องพึ่ง A6-A10
  ├─→ A12 (Regex) — ใช้ A1 (String) เป็นหลัก ไม่ต้องพึ่งหัวข้ออื่นมาก
  ├─→ A13 (Exception) — ใช้ A5 (Function), A7 (Object/class)
  ├─→ A14 (Module) — ใช้ A5 (Function) เป็นของที่ export/import บ่อยที่สุด
  └─→ A15 (Debugging) — เป็น meta-skill ครอบทุกหัวข้อ A (ยิ่งรู้เนื้อหามาก ยิ่งอ่าน error ได้แม่น)

B (Web Fundamentals) — เป็นความรู้อิสระ (conceptual) ไม่ต้องพึ่ง JS syntax
  B1(Architecture) → B2(Client-Server) → B3(HTTP Methods) → B4(Status Code) → B5(Stateless) → B6(REST)

C (DOM) — ต้องพึ่งพื้นฐาน JS ก่อนทั้งหมด
  A1,A7 (Object/property) + A6 (Array-like: HTMLCollection/NodeList) + A4 (Loop, forEach ผ่าน childNodes)
    → C1(DOM tree) → C3(window/document) → C4(Node Types) → C5(Element/Attr)
    → C6(innerHTML/innerText/textContent)
    → C7(Traversing) → C8(Selecting) → C9(HTMLCollection vs NodeList)
    → C10(Manipulating) → C11(DocumentFragment)
    → C12(dataset) / C13(classList) / C14(System Dialogs — priority ต่ำ)
```

High Priority จาก Dependency (กฎ "ถ้าไม่เข้าใจ A จะทำ B ไม่ได้"):

1. **A1 (Primitive vs Object)** — เป็น prerequisite ของ A7,A8,A9,A10 ทั้งหมด และเป็นรากของ error ประเภท "แก้ค่าไม่ได้ผลตามคาด" เกือบทุกเคสในวิชานี้
2. **A5 (Function: parameter/return/scope)** — เป็น prerequisite ของ A6 (callback), A13 (error handling), A14 (module), และเกือบทุกโจทย์ Practical
3. **A8 (Reference vs Copy)** — เป็น prerequisite โดยตรงของ A9-A10 (shallow copy) และเป็นคำอธิบายเบื้องหลัง bug จำนวนมากที่สุดที่เอกสารเน้นย้ำ (ปรากฏทั้งในบทที่ 1, 6, 7 ของเอกสารทบทวน)

PHASE 6 (Final Output #6): COMMON CONFUSIONS & MISTAKES

> ตารางนี้รวบรวมจาก "พลาดกันบ่อย" ของเอกสารทบทวน 12 บท (แหล่งข้อมูลที่ตรงและเชื่อถือได้ที่สุดสำหรับ Phase นี้ เพราะสร้างมาเพื่อเตือนจุดพลาดโดยเฉพาะ)

Concept A	สับสนกับ	จุดต่าง	จุดที่ต้องจำ
`x === NaN`	`Number.isNaN(x)`	`NaN` ไม่เท่ากับตัวมันเองแม้ใช้ `===`	ใช้ `Number.isNaN()` เท่านั้น

Concept A	สับสนกับ	จุดต่าง	จุดที่ต้องจำ
`a == x == b`	เขียนแยก `a<x && x<b`	JS ไม่มี syntax เทียบช่วงต่อเนื่องแบบคณิตศาสตร์	ตัวที่สอง `==` เทียบกับผล boolean ของตัวแรก
`for...in` กับ Array	`for...of` กับ Array	`for...in` ได้ key เป็น **string** ("0","1")	ใช้ `for...of` กับ array เสมอ
ตัวที่วนอยู่ (`item`)	Array ทั้งก้อน (`items`)	พิมพ์ `items.price` ใน loop จะได้ `undefined` (ไม่ error)	ใช้ตัวแปรที่วนอยู่จริงเสมอ
`console.log()`	`return`	log แค่โชว์ผล ไม่ส่งค่าต่อได้	จำไว้ว่าต้องมี `return` ถ้าต้องการใช้ค่าต่อ
callback ใส่ `()`	callback ไม่ใส่ `()`	`.filter(fn())` เรียกทันที ส่งผลลัพธ์ของ fn ไปแทนตัว fn	ส่งชื่อฟังก์ชันเปล่า ๆ ไม่ใส่วงเล็บ
default parameter	ค่า `null` / `0` / `""`	default ทำงานเฉพาะตอนได้ `undefined` เท่านั้น	อย่าคิดว่า default รับมือ falsy ค่าอื่นได้
`sort()` ตัวเลข (ไม่ใส่ comparator)	`sort()` แบบ string	ไม่ใส่ comparator = เรียงแบบ string เสมอ	ใส่ `(a,b)=>a-b` เพื่อเรียงตัวเลขจริง
`reduce()` ไม่ใส่ initial value	`reduce()` ใส่ initial value	ไม่ใส่ = หยิบสมาชิกแรกมาเป็นถังเงียบ ๆ, array ว่าง = error	ใส่ initial value เสมอ
`const b = a` (object)	สำเนาใหม่จริง	`const` ล็อกแค่ป้ายชื่อ ตัวแปรยังชี้ก้อนเดียวกัน	ต้องใช้ spread/structuredClone ถ้าต้องการสำเนา
สะกด key ผิดตอนแก้ค่า	error เตือน	JS ไม่ error แต่สร้าง key ใหม่เงียบ ๆ	debug ด้วย `Object.keys(obj)`
destructure ใน arrow function ไม่มี `()`	มี `()` ครอบ	ไม่ครอบ = SyntaxError ทันที (งงว่า `{}` คือ pattern หรือ function body)	ครอบ `()` เสมอ
spread คัดลอก object ซ้อนกัน	deep copy จริง	spread คัดลอกได้แค่ชั้นบนสุด (shallow) — ของซ้อนยังใช้ reference เดิม	ใช้ `structuredClone()` ถ้าต้องการ deep copy
`Date.now()`	`new Date()`	`Date.now()` คืนตัวเลข ไม่ใช่ Date object	ห่อด้วย `new Date(Date.now())` ถ้าต้องการ methods
หาผลต่างวันไม่ใส่วงเล็บ	ใส่วงเล็บครอบตัวหาร	`/` กับ `*` ลำดับเท่ากัน อ่านซ้ายไปขวา ผลต่างเป็นล้านเท่า	ครอบวงเล็บผลคูณของตัวหารเสมอ
`str.match(regex)`	`regex.test(str)`	match คืน array/null ไม่ใช่ boolean	ใช้ `test()` ถ้าต้องการ boolean
`.` ใน regex	`\.`	`.` = ตัวอักษรอะไรก็ได้ 1 ตัว ไม่ใช่จุดจริง	escape ด้วย `\.` เมื่อต้องการจุดจริง
`catch` จับทุก error	จับเฉพาะที่ตั้งใจดัก	catch จับ bug ของเราเองด้วย (เช่น พิมพ์ตัวแปรผิด)	ระวัง catch กลืนบั๊กจริงจนหาไม่เจอ
`throw`	`return`	throw หยุดทำงานทันที ส่งไปให้ catch, return แค่คืนค่า	คนละพฤติกรรมกัน
ผ่าน 2/3 เทส	ใกล้ถูกแล้ว	แต่ละเทสตรวจคนละแง่มุม น้ำหนักไม่เท่ากัน	เคสที่ไม่ผ่านอาจสำคัญที่สุด
`innerText`	`textContent`	innerText ไม่รวม element ที่ `display:none` , textContent รวม	เลือกใช้ตามต้องการเห็นของซ่อนหรือไม่
HTMLCollection	NodeList	HTMLCollection = live เสมอ, NodeList (จาก querySelectorAll) = static ส่วนใหญ่	เช็คว่าได้จาก method ไหนก่อนสรุปว่า live/static
Attr node	Element node	Attr ไม่อยู่ใน document tree จริง, parentNode/sibling เป็น null	ใช้ `ownerElement` แทน
System Dialogs (alert/confirm/prompt)	DOM manipulation ปกติ	Dialog ไม่มี HTML, sync+modal, กำหนดโดย OS	**มีหลักฐานว่าไม่ออกสอบ — ให้ความสำคัญต่ำ**
"เสมอ/ทุกกรณี/เท่านั้น" ในโจทย์ Perfect Match	ข้อยกเว้นที่ซ่อนอยู่	คำเด็ดขาดในตัวเลือกมักมีข้อยกเว้นซ่อน	หยุดคิดก่อนเชื่อคำเด็ดขาดทันที

PHASE 7 (Final Output #7): PROFESSOR EMPHASIS ANALYSIS

> วิเคราะห์จาก transcript คำสอนจริง 5 คาบ (Aug 5, 11, 19, 26, Sep 2) — พิจารณาความถี่คำสำคัญ, ข้อความเตือน, และหลักฐานประกอบ (Lab/ตัวอย่างซ้ำ/การเปรียบเทียบ) มีแนวโน้มสูงว่าสะท้อนความสำคัญจริง แต่ **ไม่ยืนยันว่าจะออกข้อสอบแน่นอน** เว้นแต่มีคำพูดตรง (ระบุไว้ในวงเล็บ)

ระดับ	หัวข้อ	หลักฐานสนับสนุน
VERY HIGH	Reference vs Copy (Object/Array)	คำว่า "reference"/"copy"/"ก็อป" ปรากฏรวมกัน >100 ครั้งใน transcript; เป็นหัวข้อเดียวที่ปรากฏซ้ำใน 3 บทของเอกสารทบทวน (1, 6, 7); อาจารย์อธิบายซ้ำหลายรอบใน

ระดับ	หัวข้อ	หลักฐานสนับสนุน
		คาบเรียนจริง (มีบทสนทนาถาม-ตอบยาวเรื่องนี้)
VERY HIGH	Function (parameter/return/callback)	คำว่า "ฟังก์ชัน" ปรากฏ 80 ครั้งใน transcript, มีคำเตือนแบบ "fail fast" (เขียนฟังก์ชันแล้วเช็คเงื่อนไข reject ทันทีตั้งแต่ต้น) ปรากฏชัดเจนเป็นปรัชญาการเขียนโค้ดที่อาจารย์เน้น
HIGH	DOM: Node/Selector/Traversal	ตรงกับคำพูด "ที่อาจารย์สอนวันนี้ออกข้อสอบหมดเลยนะคะ" ในคาบที่สอน DOM node/selector
HIGH	Array methods (mutating vs non-mutating)	เอกสารทบทวนให้พื้นที่ทั้งบท (บทที่ 5) พร้อมตารางเปรียบเทียบละเอียด, คำว่า "ระวัง" ปรากฏ 95 ครั้งทั่วทั้ง transcript ส่วนใหญ่เกี่ยวกับ array/object mutation
HIGH	Destructuring/Spread (shallow copy trap)	มีการอธิบายซ้ำเรื่อง "spread กับ merge" และ key ชนกันในคาบเรียนจริง ตรงกับเอกสารทบทวนบทที่ 7
MEDIUM-HIGH	Regex	อาจารย์ถามตรง ๆ ในคาบเรียนว่า "เรียน Regular Expression ไปทำไม ใช้เมื่อไหร่" แล้วอธิบายเหตุผลยาว — สะท้อนว่าอยากให้เข้าใจ "ทำไม" ไม่ใช่แค่จำ syntax
MEDIUM	Exception Handling	มีการเน้นเรื่อง error message และการ debug อย่างเป็นระบบในคาบเรียน
MEDIUM	JavaScript Date	มีการสอนละเอียดเรื่องคำนวณผลต่างวัน แต่อาจารย์พูดชัดว่า **Temporal (เวอร์ชันใหม่กว่า) ยังไม่ออกสอบ**
LOW (มีหลักฐานตรง)	System Dialogs	อาจารย์พูดตรง: **"ไม่ได้ออกข้อสอบแล้วก็ไม่ได้สำคัญมาก"**
ไม่ยืนยัน	REST, JavaScript Module	ปรากฏในสไลด์แต่ไม่พบหลักฐานเน้นย้ำเป็นพิเศษในส่วน transcript ที่อ่านได้

Pattern อื่นที่พบจากอาจารย์ (Additional signals):

- อาจารย์เน้นปรัชญา **"fail fast"** (เขียนฟังก์ชันเช็คเงื่อนไขผิดแล้ว reject/throw ทันที ไม่รอให้พังทีหลัง) — สอดคล้องกับเนื้อหา Exception Handling (A13) และ Guard Clause ใน Conditionals (A3)
- อาจารย์เตือนเรื่อง **"ยกมือถามเมื่อไม่เข้าใจ"** และการฝึกอธิบายเป็นคำพูด (soft skill) — ไม่ใช่เนื้อหาที่จะออกสอบโดยตรง แต่สะท้อนว่าข้อสอบทฤษฎีน่าจะเน้น "อธิบายได้ว่าทำไม" มากกว่าจำอย่างเดียว
- อาจารย์ใช้ AI ช่วยออกแบบแบบฝึกหัด/สไลด์/ข้อสอบ แต่ตรวจสอบทุกครั้ง (ข้อมูลบริบท ไม่กระทบเนื้อหาสอบ)

PHASE 8 (Final Output #8): THEORY EXAM RELEVANCE (Part 1 — Closed Book)

ระดับ	หัวข้อ	เหตุผล
🔴 **VERY HIGH**	Truthy/Falsy, `==` vs `===`, `NaN`	ทายผลลัพธ์ได้จาก concept ล้วน ๆ ไม่ต้องเขียนโค้ด เหมาะกับ MCQ/Perfect Match มาก (มีแบบฝึกหัดรูปแบบ "ทายก่อนเฉลย true/false" ในเอกสารตรงกับรูปแบบนี้)
🔴 **VERY HIGH**	Reference vs Copy (primitive vs object)	เป็นแนวคิดล้วน ทายผลลัพธ์การแก้ค่าได้โดยไม่ต้องรันโค้ด เหมาะกับ MCQ ทายว่า "ค่าไหนเปลี่ยน/ไม่เปลี่ยน"
🔴 **VERY HIGH**	Array methods (mutating vs non-mutating)	ตารางเปรียบเทียบล้วน ๆ ถามว่า method ไหนแก้ต้นฉบับ/คืนของใหม่ ได้โดยไม่ต้องรันจริง
🔴 **VERY HIGH**	for...in กับ Array vs Object	ทายผลลัพธ์ประเภทข้อมูล (string vs number) จาก concept ตรง ๆ
🟠 **HIGH**	Default Parameter (undefined only)	ทายผลลัพธ์จากการส่ง `null` / `0` / `undefined` เข้าฟังก์ชัน — เหมาะกับ MCQ ทายผลลัพธ์
🟠 **HIGH**	Destructuring/Spread syntax + shallow copy	ทายว่า syntax ถูก/ผิด, ทายว่าต้นฉบับเปลี่ยนหรือไม่หลังแก้ของซ้อน
🟠 **HIGH**	Date arithmetic concept (ms, เดือน 0-11)	ทายค่าที่ได้จาก getter ต่าง ๆ โดยไม่ต้องรันจริง
🟠 **HIGH**	Regex: test() vs match(), `.` vs `\.`	ทายผลลัพธ์ (boolean/array/null) จาก pattern ที่กำหนด
🟠 **HIGH**	try/catch/finally execution order	ทายลำดับการทำงานเมื่อเกิด error — เหมาะกับ MCQ มาก
🟡 **MEDIUM**	HTTP Methods/Status Code/Stateless	เป็นความรู้จำล้วน (definition-based) เหมาะกับ MCQ ทั่วไปแต่ไม่มีความซับซ้อนให้ทดสอบเชิงลึกมาก
🟡 **MEDIUM**	HTMLCollection vs NodeList (live/static)	เป็นคู่เปรียบเทียบที่ MCQ ชอบออก แต่ต้องอาศัยจำมากกว่าคิดวิเคราะห์
🟡 **MEDIUM**	Named vs Default Export	จำกฎง่าย ๆ ได้ ออกเป็น MCQ ได้แต่ไม่ซับซ้อนมาก
🟢 **LOW**	Syntax ปลีกย่อยของ DOM method (เช่นชื่อ method ยาว ๆ)	เหมาะกับ Practical ที่เปิด MDN ได้มากกว่า เพราะเป็นการจำชื่อฟังก์ชันล้วน ๆ ไม่มีอะไรให้ "คิด" ในบริบท closed book

ระดับ	หัวข้อ	เหตุผล
🟢 **LOW (มีหลักฐานตรง)**	System Dialogs	อาจารย์ยืนยันไม่ออกสอบ
🟢 **LOW (มีหลักฐานตรง)**	Temporal API	อาจารย์ยืนยันไม่ออกสอบ (ยังไม่ได้สอนลึก)

PHASE 9 (Final Output #9): PRACTICAL EXAM RELEVANCE (Part 2 — VSCode + MDN)

Topic	Practical Skill ที่วัดได้	ความยาก
Variable/Type/Conditionals/Loop	เขียนโปรแกรมพื้นฐานตั้งแต่ต้น (เช่น ตรวจเกรด, จัดกลุ่มข้อมูล)	ง่าย-ปานกลาง
Function	เขียนฟังก์ชันตาม spec ที่กำหนด (parameter/return ต้องตรง) — โจทย์แบบ "เช็ก 3 จุดก่อนส่ง"	ปานกลาง
Array methods	ประยุกต์ map/filter/reduce กับข้อมูลจริง (เช่น ตะกร้าสินค้า, จัดกลุ่มหมวดหมู่)	ปานกลาง-ยาก
Object + Reference	แก้ bug ที่เกิดจาก reference/mutation โดยไม่ได้ตั้งใจ, เขียน state update แบบ immutable	ยาก (เพราะ MDN ช่วยไม่ได้ ต้องเข้าใจเอง)
Destructuring/Spread	แกะข้อมูลจาก API response ที่ซ้อนกันหลายชั้น	ปานกลาง-ยาก
Date	คำนวณวันที่ (เช่น นับถอยหลังโปรโมชัน)	ปานกลาง (MDN ช่วยเรื่อง method name ได้)
Regex	เขียน pattern ตรวจ validate form (email/เบอร์โทร/รหัสผ่าน) หรือ highlight คำค้นหา	ปานกลาง-ยาก (ต้องฝึกอ่าน pattern เอง MDN ช่วยแค่ syntax)
Exception Handling	ดักและจัดการ error จาก JSON.parse หรือ custom validation	ปานกลาง
Module	แยกไฟล์ import/export (ถ้าโจทย์ practical มีหลายไฟล์)	ง่าย-ปานกลาง (MDN ช่วยได้มาก)
DOM: Select/Traverse/Manipulate	สร้าง/ลบ/แก้ element ตาม requirement (โจทย์แบบ "สร้าง Front-end จาก Requirement" ตรงกับ spec ข้อสอบ)	ยาก (ปริมาณ method เยอะ แต่ MDN ช่วยเรื่อง syntax ได้มาก)
Debugging (A15)	อ่าน error, หา syntax error, ใช้ console.log ไล่ค่า	**สำคัญที่สุดต่อการทำข้อสอบให้ทันเวลา 2 ชม.**

PHASE 10 (Final Output #10): THEORY vs PRACTICAL MATRIX

Topic	ต้องจำ	ต้องเข้าใจ	ต้องเขียน	Theory	Practical	Priority
Truthy/Falsy, `==` / `===`, NaN	✓	✓	-	🔴	🟡	🔴
Primitive vs Object / Reference vs Copy	✓	✓	✓	🔴	🔴	🔴
Conditionals (if/switch/guard)	✓	✓	✓	🟠	🟠	🟠
Loop (for/while/for-of/for-in)	✓	✓	✓	🔴	🟠	🔴
Function (param/return/scope/callback)	✓	✓	✓	🟠	🔴	🔴
Array methods (mutating vs non)	✓	✓	✓	🔴	🔴	🔴
Object (dot/bracket, property)	✓	✓	✓	🟠	🟠	🟠
Destructuring/Spread (shallow copy)	✓	✓	✓	🟠	🟠	🔴
JavaScript Date (ไม่รวม Temporal)	✓	✓	✓	🟠	🟡	🟠
Regular Expression	✓	✓	✓	🟠	🟠	🟠
Exception Handling (try/catch)	✓	✓	✓	🟠	🟠	🟠
JavaScript Module (import/export)	✓	🟡	🟡	🟡	🟡	🟡
Debugging / อ่าน Error	-	✓	✓	🟡	🔴	🟠
Web Architecture / 3-Tier	✓	🟡	-	🟡	-	🟡
Client-Server / HTTP Methods & Status	✓	✓	-	🟡	🟡	🟡

Topic	ต้องจำ	ต้องเข้าใจ	ต้องเขียน	Theory	Practical	Priority
Stateless / Cookie / Token	✓	✓	-	🟡	-	🟢
REST	✓	🟡	-	🟢	-	🟢 (หลักฐานน้อย)
DOM: Node Types/Tree	✓	✓	🟡	🟡	🔴	🟠
DOM: Selecting (querySelector ฯลฯ)	✓	✓	✓	🟡	🔴	🔴
DOM: HTMLCollection vs NodeList	✓	✓	-	🟠	🟡	🟠
DOM: Traversing	✓	✓	✓	🟡	🟠	🟠
DOM: Manipulating (create/append/remove)	✓	✓	✓	🟡	🔴	🔴
DOM: innerHTML/innerText/textContent	✓	✓	-	🟠	🟠	🟠
DOM: dataset/classList	✓	🟡	✓	🟡	🟠	🟡
DOM: System Dialogs	🟡	-	-	🟢	🟢	🟢 (ยืนยันสอบ)
Temporal API	-	-	-	🟢	🟢	🟢 (ยืนยันสอบ)

PHASE 11 (Final Output #11): TIER 1–4 PRIORITY

🔴 TIER 1 — MUST MASTER (ถ้าไม่รู้ มีผลต่อคะแนนอย่างมาก ทั้ง Theory และ Practical)

1. **Primitive vs Object / Reference vs Copy** — รากฐานของ bug และคำถามเกือบทั้งวิชา
2. **Array methods: mutating vs non-mutating** (push/pop/sort vs map/filter/reduce) + ตารางคืนค่า
3. **Function: parameter/return/scope, default parameter (undefined only), callback (ไม่ใส่วงเล็บ)**
4. **Loop: for...of vs for...in, ตัวที่วนอยู่ vs array ทั้งก้อน**
5. **Truthy/Falsy, `===` vs `==`, `NaN` special rule**
6. **Destructuring/Spread + Shallow Copy Trap**
7. **DOM: Selecting Nodes (querySelector/querySelectorAll/getElementById) + Manipulating Nodes (createElement/appendChild/removeChild)**

🟠 TIER 2 — HIGH PRIORITY (ควรเข้าใจดี)

1. Conditionals (switch vs if/else, guard clause order)
2. Object (dot/bracket, สะกด key ผิด)
3. JavaScript Date (getters, ms arithmetic, ไม่รวม Temporal)
4. Regular Expression (test/match/replace, `.` vs `\.`)
5. Exception Handling (try/catch/finally syntax, throw vs return, custom Error)
6. DOM: HTMLCollection vs NodeList, Traversing (Element Traversal API), innerHTML/innerText/textContent
7. Debugging (อ่าน error, undefined 3 ทาง, เช็ก 3 จุดก่อนส่ง)

🟡 TIER 3 — MEDIUM (ควรทบทวน)

1. JavaScript Module (named vs default export/import)
2. Web Architecture (3-Tier), Client-Server, HTTP Methods/Status Code
3. DOM: Node Types (12 constants), dataset, classList
4. DOM: DocumentFragment (performance concept)

🟢 TIER 4 — LOW (อ่านเมื่อมีเวลา / มีหลักฐานว่าไม่เน้น)

1. **System Dialogs (alert/confirm/prompt)** — อาจารย์ยืนยันตรงว่าไม่ออกสอบ

2. **Temporal API** — อาจารย์ยืนยันตรงว่ายังไม่ออกสอบ
3. REST (รายละเอียดเชิงลึก) — เอกสารมีแค่หัวข้อ ไม่มีเนื้อหาขยายในส่วนที่อ่านได้
4. HTTP Stateless (รายละเอียด Cookie/Token) — เป็นความรู้พื้นหลัง ไม่ปรากฏการเน้นย้ำเป็นพิเศษ

PHASE 12 (Final Output #12): EXAM-READY CHECKLIST

JavaScript Fundamentals

- [ ] อธิบายความต่าง primitive vs object ได้ และบอกได้ว่าทำไม `const b = a` (object) ไม่ได้สำเนาใหม่ให้
- [ ] บอกค่า falsy ได้ครบ 6 ตัว และรู้ว่า `[]` / `{}` เป็น truthy เสมอ
- [ ] เช็ค NaN ด้วย `Number.isNaN()` ไม่ใช่ `=== NaN`
- [ ] เรียงเงื่อนไข guard แบบแคบก่อนกว้างได้ถูกต้อง
- [ ] เลือกใช้ `for`, `for...of`, `for...in`, `while` ได้ถูกสถานการณ์ และอธิบายได้ว่าทำไม `for...in` กับ array ได้ key เป็น string
- [ ] แยก `break` กับ `continue` ได้ชัดเจน
- [ ] แยก `return` กับ `console.log` ได้ว่าอันไหนส่งค่าออกจากฟังก์ชันจริง
- [ ] รู้ว่า default parameter ทำงานเฉพาะตอนได้ `undefined` เท่านั้น
- [ ] ส่ง callback แบบไม่ใส่วงเล็บเรียกได้ถูกต้อง
- [ ] บอกได้ว่า array method ตัวไหนแก้ต้นฉบับ ตัวไหนคืนของใหม่ (push/pop/sort vs map/filter/reduce)
- [ ] ใส่ initial value ใน `reduce()` เสมอ และเช็ค `groups[key] ??= []` ก่อน push ตอนจัดกลุ่ม
- [ ] ใส่ comparator ใน `sort()` เมื่อเรียงตัวเลข
- [ ] เข้าถึง property ด้วย dot/bracket ได้ถูกต้อง และรู้ว่า bracket ต้องมี quote เมื่อไหร่
- [ ] เขียน object/array destructuring พร้อม rename และ default ได้
- [ ] destructure ใน arrow function parameter ได้ถูก syntax (ต้องครอบ `()`)
- [ ] บอกได้ว่า spread สอง object คีย์ซ้ำกัน ค่าไหนชนะ (ตัวหลังทับตัวก่อน)
- [ ] อธิบายได้ว่าทำไมแก้ของซ้อนผ่านสำเนาที่ spread มาแล้วยังกระทบต้นฉบับ (shallow copy)
- [ ] รู้ว่า `Date` เป็นตัวเลข ms และคำนวณผลต่างวันโดยใส่วงเล็บครอบตัวหารถูกต้อง
- [ ] แยก `Date.now()` (ตัวเลข) กับ `new Date()` (object) ได้
- [ ] อ่าน pattern regex ทีละชิ้นด้วยตารางสัญลักษณ์พิเศษได้ (`\d \w \s ^ $ * + ?`)
- [ ] แยก `test()` (boolean) กับ `match()` (array/null) ได้
- [ ] เขียนโครง `try/catch/finally` ด้วยมือได้โดยไม่ต้องเปิดตัวอย่าง
- [ ] แยก `throw` กับ `return` ได้
- [ ] เขียน custom Error class ด้วย `extends Error` + `super(message)` และเช็คด้วย `instanceof`
- [ ] แปลความหมายของ error message ที่เจอบ่อยได้ และรู้ว่า undefined มาจาก 3 ทางไหนได้บ้าง
- [ ] แยก Named Export กับ Default Export ได้ถูกต้อง

Web Fundamentals

- [ ] อธิบาย 3-Tier Architecture ได้ (Presentation/Application/Data)
- [ ] บอกหน้าที่ Client vs Server ได้
- [ ] จำ HTTP Methods (GET/POST/PUT/DELETE) และ Status Code กลุ่ม 2xx/3xx/4xx/5xx ได้
- [ ] อธิบาย Stateless และวิธีที่ Client จำสถานะได้เอง (Cookie/Token)

DOM

- [ ] อธิบายโครงสร้าง DOM เป็น tree of nodes ได้
- [ ] บอกความต่าง `innerHTML` / `innerText` / `textContent` ได้
- [ ] แยก HTMLCollection (live) กับ NodeList (static) ได้ และรู้ method ไหนคืนอะไร

- [ ] ใช้ `querySelector` / `querySelectorAll` / `getElementById` / `closest()` ได้ถูกสถานการณ์
- [ ] สร้าง/เพิ่ม/ลบ/แทนที่ element ด้วย `createElement/appendChild/removeChild/replaceChild/insertBefore` ได้
- [ ] อธิบายประโยชน์ของ `DocumentFragment` ต่อ Reflow/Repaint ได้
- [ ] ใช้ `dataset` และ `classList` (add/remove/toggle/contains) ได้
- [ ] รู้ว่า System Dialogs (alert/confirm/prompt) มีความสำคัญต่ำต่อข้อสอบ (แต่ยังควรรู้พฤติกรรมพื้นฐานไว้)

PHASE 13 (Final Output #13): FINAL MASTER SUMMARY

Topic	Core Knowledge	จำ	เข้าใจ	เขียน	Debug	The
Primitive vs Object / Reference-Copy	ตัวแปรชี้ที่อยู่ ไม่ใช่ถือค่า	✓	✓	✓	✓	🔴
Truthy/Falsy/Equality	falsy 6 ค่า, `===`, NaN	✓	✓	-	✓	🔴
Conditionals	if/switch, guard order	✓	✓	✓	✓	🟠
Loop	for/while/for-of/for-in	✓	✓	✓	✓	🔴
Function	param/return/scope/callback	✓	✓	✓	✓	🟠
Array Methods	mutating vs non-mutating	✓	✓	✓	✓	🔴
Object	dot/bracket, property	✓	✓	✓	✓	🟠
Destructuring/Spread	shallow copy trap	✓	✓	✓	✓	🟠
JS Date (ไม่รวม Temporal)	ms arithmetic	✓	✓	✓	✓	🟠
Regular Expression	pattern, test/match	✓	✓	✓	✓	🟠
Exception Handling	try/catch/finally, throw	✓	✓	✓	✓	🟠
JS Module	named/default export	✓	🟡	🟡	-	🟡
Debugging	อ่าน error, undefined 3 ทาง	-	✓	✓	✓	🟡
Web Architecture / HTTP	3-Tier, method, status	✓	✓	-	-	🟡
DOM Core (Node/Select/Manipulate)	tree, querySelector, appendChild	✓	✓	✓	✓	🟡
DOM HTMLCollection vs NodeList	live vs static	✓	✓	-	-	🟠
DOM dataset/classList	data-* attr, class ops	✓	🟡	✓	-	🟡
System Dialogs	alert/confirm/prompt	🟡	-	-	-	🟢
Temporal API	Date เวอร์ชันใหม่	-	-	-	-	🟢

ข้อจำกัดของการวิเคราะห์นี้ (ตามกฎ "ถ้าไม่มีหลักฐานเพียงพอ ต้องระบุว่าไม่สามารถยืนยันได้")

- **ไม่สามารถยืนยันได้:** เนื้อหา REST โดยละเอียด (มีแค่หัวข้อสไลด์ ไม่มีเนื้อหาขยายในส่วนที่อ่านได้จริง)
- **ไม่สามารถยืนยันได้:** เนื้อหาของหน้า 1–346 ใน `class1-5.pdf` เนื่องจากไฟล์เสียหาย — หากมีเนื้อหาสำคัญอื่นอยู่ในช่วงนั้น (เช่น HTML/CSS พื้นฐานที่อาจสอนก่อนหน้า) จะไม่ปรากฏในการวิเคราะห์นี้
- **ไม่สามารถยืนยันได้:** เนื้อหา Lab Sheet / Assignment / Quiz / ข้อสอบเก่าจริง เนื่องจากยังไม่ได้รับเอกสารเหล่านี้ (ผู้ใช้แจ้งว่าจะส่งเพิ่มแต่ยังไม่ได้อัปโหลด)
- **Additional Knowledge ที่แทรกในเอกสารนี้:** มีเพียงเล็กน้อย ส่วนใหญ่เป็นการอธิบายขยายความจาก concept ที่มีอยู่แล้วในเอกสาร ไม่ได้เพิ่มเนื้อหาใหม่นอกขอบเขต — จุดที่เป็น Additional Knowledge ชัดเจนคือคำว่า "PATCH" ที่กล่าวถึงในหมายเหตุ HTTP Methods (ไม่ปรากฏในเอกสารต้นฉบับ)

จบ Knowledge Base — ยังไม่มีการสร้างข้อสอบ 30 ข้อตามที่ระบุไว้ในคำสั่งเดิม รอขั้นตอนถัดไป