วิทยาการคำนวณ ๒

• ระดับกลาง (Intermediate)

การออกแบบขั้นตอนวิธี โครงสร้างข้อมูล และ Python

ผู้ช่วยศาสตราจารย์ ดร.ชีวะ ทัศนา

สาขาวิชาฟิสิกส์ คณะวิทยาศาสตร์และเทคโนโลยี มรภ.รำไพพรรณี

Big-O • โครงสร้างข้อมูล • การค้นหาและจัดเรียง • แบบจำลองฟิสิกส์

วิทยาการคำนวณ 2

โครงสร้างข้อมูลเชิงลึกและการวิเคราะห์ขั้นตอนวิธี

Advanced Data Structures & Algorithmic Analysis

ชุดตำราวิทยาการคำนวณและฟิสิกส์บูรณาการ 4 เล่มสมบูรณ์

ผู้ช่วยศาสตราจารย์ ดร.ชีวะ ทัศนา

สาขาวิชาฟิสิกส์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยราชภัฏรำไพพรรณี

สำนักพิมพ์มหาวิทยาลัยราชภัฏรำไพพรรณี (RBRU Press)

ข้อมูลทางบรรณานุกรมของสำนักหอสมุดแห่งชาติ (National Library of Thailand CIP)

ชีวะ ทัศนา.
วิทยาการคำนวณ 2 โครงสร้างข้อมูลเชิงลึกและการวิเคราะห์ขั้นตอนวิธี. -- จันทบุรี : สำนักพิมพ์มหาวิทยาลัยราชภัฏรำไพพรรณี, 2569.
320 หน้า.

1. วิทยาการคำนวณ. 2. ขั้นตอนวิธีและการประมวลผลข้อมูล. 3. ภาษาไพทอน (คอมพิวเตอร์). I. ชื่อเรื่อง.
ISBN 978-616-XXXX-02-8 (ฉบับพิมพ์สมบูรณ์)
ISBN 978-616-XXXX-XX-X (ฉบับดิจิทัล EPUB 3.0)

สงวนลิขสิทธิ์ตามพระราชบัญญัติลิขสิทธิ์ พ.ศ. 2537 และฉบับแก้ไขเพิ่มเติม

ห้ามการลอกเลียนแบบ ดัดแปลง ทำซ้ำ หรือเผยแพร่ส่วนใดส่วนหนึ่งของหนังสือเล่มนี้ ไม่ว่าด้วยกระบวนการทางอิเล็กทรอนิกส์ การถ่ายสำเนา หรือวิธีการใดๆ โดยไม่ได้รับอนุญาตเป็นลายลักษณ์อักษรจากผู้เขียน ยกเว้นเพื่อการอ้างอิงทางวิชาการ

ผู้เขียน: ผู้ช่วยศาสตราจารย์ ดร.ชีวะ ทัศนา (Ph.D. in Physics)
จัดพิมพ์โดย: สำนักพิมพ์มหาวิทยาลัยราชภัฏรำไพพรรณี เลขที่ 41 หมู่ 5 ต.ท่าช้าง อ.เมือง จ.จันทบุรี 22000
พิมพ์ครั้งที่ 1: สิงหาคม พ.ศ. 2569 (จำนวนพิมพ์ 1,000 เล่ม)

คำนำ

ตำรา **"วิทยาการคำนวณ 2: โครงสร้างข้อมูลเชิงลึกและการวิเคราะห์ขั้นตอนวิธี"** เล่มนี้ ได้รับการประพันธ์และเรียบเรียงขึ้นอย่างประณีต เพื่อใช้เป็นตำราหลักในรายวิชา **4122105 โครงสร้างข้อมูลและการวิเคราะห์ขั้นตอนวิธี** โดยมีจุดมุ่งหมายเพื่อยกระดับทักษะการคิดเชิงคำนวณ การออกแบบขั้นตอนวิธี และการประยุกต์ใช้เทคโนโลยีปัญญาประดิษฐ์ของนิสิต นักศึกษา และครูวิทยาศาสตร์ในระดับอุดมศึกษา

โครงสร้างของเนื้อหาในเล่ม ได้รับการร้อยเรียงตามกรอบทฤษฎีการศึกษา Bloom's Taxonomy และ Active Learning Cycle โดยผสานทฤษฎีคณิตศาสตร์เข้มข้นเข้ากับตัวอย่างโค้ดภาษา Python 3.11 และสื่อการทดลองเสมือนจริง **3D AR MediaPipe Hands** แบบไร้สัมผัส เพื่อให้ผู้เรียนเกิดมโนทัศน์ที่ลึกซึ้งและสามารถนำไปสร้างสรรค์นวัตกรรมได้จริง

ผู้ช่วยศาสตราจารย์ ดร.ชีวะ ทัศนา
สาขาวิชาฟิสิกส์ คณะวิทยาศาสตร์และเทคโนโลยี
มหาวิทยาลัยราชภัฏรำไพพรรณี
สิงหาคม 2569

กิตติกรรมประกาศ

ความสำเร็จในการจัดทำตำราวิชาการชุดนี้ สำเร็จลุล่วงลงได้ด้วยความอนุเคราะห์และการสนับสนุนอย่างดียิ่งจากหลากหลายภาคส่วน ผู้เขียนขอกราบขอบพระคุณ **มหาวิทยาลัยราชภัฏรำไพพรรณี** และ **คณะวิทยาศาสตร์และเทคโนโลยี** ที่ได้ส่งเสริมและสนับสนุนทุนวิจัยและการพัฒนานวัตกรรมการจัดการเรียนการสอนอันเป็นประโยชน์ต่อนักศึกษาและวงการวิชาการ

ขอขอบพระคุณ **รองศาสตราจารย์ ดร.จิรพัฒน์ จันทมาลี** และคณาจารย์ผู้ทรงคุณวุฒิทุกท่าน ที่ได้กรุณาให้คำปรึกษา แลกเปลี่ยนข้อคิดเห็นเชิงวิชาการ และร่วมพัฒนากรอบมาตรฐานการสอนวิทยาการคำนวณร่วมกันมาอย่างต่อเนื่อง

ขอขอบคุณนักศึกษาสาขาวิชาฟิสิกส์และวิทยาการคอมพิวเตอร์ทุกคน ที่ได้ร่วมทดลองใช้งานระบบห้องปฏิบัติการเสมือนจริง 3D AR และให้ข้อเสนอแนะอันมีค่ายิ่งต่อการปรับปรุงหนังสือเล่มนี้ให้มีความสมบูรณ์สูงสุด

ผู้ช่วยศาสตราจารย์ ดร.ชีวะ ทัศนา

สารบัญ

หัวข้อและเนื้อหา	หน้า
คำนำ	iv
กิตติกรรมประกาศ	v
สารบัญภาพ	viii
สารบัญตาราง	x
แผนบริหารการสอนประจำรายวิชา 4122105	xii
บทที่ 1 รากฐานแนวคิดและการสร้างแบบจำลองทางตรรกะ	1
บทที่ 2 การออกแบบขั้นตอนวิธี รหัสลำลอง และผังงานมาตรฐานสากล	25
บทที่ 3 การพัฒนาโปรแกรมภาษา Python และการจัดการข้อมูล	55
บทที่ 4 โครงสร้างการควบคุม ทางเลือก การวนซ้ำ และระบบอัตโนมัติ	85
บทที่ 5 โครงสร้างข้อมูลเชิงเส้นและอัลกอริทึมการค้นหา	115
บทที่ 6 อัลกอริทึมการจัดเรียงข้อมูลและการวิเคราะห์ Big-O	145
บทที่ 7 การโปรแกรมเชิงโมดูลและฟังก์ชันเรียกซ้ำ	175
บทที่ 8 การสร้างแบบจำลองทางวิทยาศาสตร์และฟิสิกส์เชิงคำนวณ	205
บทที่ 9 สถาปัตยกรรมโครงข่ายประสาทเทียมและปัญญาประดิษฐ์	235
บทที่ 10 โครงงานบูรณาการและการสังเคราะห์นวัตกรรม Capstone	265
คู่มือห้องปฏิบัติการเสมือนจริง 3D AR MediaPipe 10 ปฏิบัติการ	295
ภาคผนวก ก ถึง ฉ (พร้อมคลังข้อสอบ 50 ข้อ)	335
ดัชนีคำสำคัญ (Index)	375
บรรณานุกรม (References)	381

สารบัญภาพ

ภาพที่	ชื่อภาพ	หน้า
ภาพที่ 1.1	เสาหลักทั้ง 4 ของแนวคิดเชิงคำนวณ (Decomposition, Pattern, Abstraction, Algorithm)	6
ภาพที่ 2.1	แผนผังโครงสร้าง 5 คุณสมบัติของขั้นตอนวิธีที่ดี	28
ภาพที่ 2.2	สัญลักษณ์ผังงานมาตรฐานสากล ISO 5807 และ ANSI Standard	32
ภาพที่ 2.3	แผนผังเปรียบเทียบ 3 โครงสร้างควบคุมหลัก (Sequential, Selection, Iteration)	36
ภาพที่ 3.1	โครงสร้างหน่วยความจำ RAM และลำดับการประมวลผลทางคณิตศาสตร์ PEMDAS	58
ภาพที่ 9.1	สถาปัตยกรรมโครงข่ายประสาทเทียมเชิงลึกแบบคอนโวลูชัน (CNN Model)	238
ภาพที่ 10.1	โครงข่ายกระดูกมือ 21 จุด 3 มิติ (MediaPipe 3D Hand Tracking System)	268

สารบัญตาราง

ตารางที่	ชื่อตาราง	หน้า
ตารางที่ 1.1	ตารางคำศัพท์และนิยามเชิงตรรกะประจำบทเรียนที่ 1	12
ตารางที่ 2.1	คำสำคัญมาตรฐานสากลในการเขียนรหัสลำลอง (Pseudocode Standard)	30
ตารางที่ 2.2	สัญลักษณ์ผังงานและหน้าที่ตามมาตรฐาน ISO 5807	34
ตารางที่ 5.1	การเปรียบเทียบความซับซ้อนเชิงเวลาและเชิงพื้นที่ Big-O ของอัลกอริทึม	120
ตารางที่ ข.1	บัตรสรุปความซับซ้อนของโครงสร้างข้อมูลมาตรฐาน Python	340

วิทยาการคำนวณ 2 การออกแบบขั้นตอนวิธี โครงสร้างข้อมูล และการแก้ปัญหาด้วย Python

บทที่ 1 การออกแบบขั้นตอนวิธีเชิงลึกและการวิเคราะห์ประสิทธิภาพ

ผู้เขียน ผู้ช่วยศาสตราจารย์ ดร.ชีวะ ทัศนา • สาขาวิชาฟิสิกส์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยราชภัฏรำไพพรรณี

📋 แผนบริหารการสอนประจำบทที่ 1

หัวข้อเนื้อหาประจำบท

1. ความสำคัญของการวิเคราะห์ประสิทธิภาพขั้นตอนวิธี
2. นิยามและคณิตศาสตร์ของสัญกรณ์โอใหญ่ (Big-O Notation)
3. การวิเคราะห์ความซับซ้อนเชิงเวลา และเชิงพื้นที่ความจำ (Space Complexity)
4. กรณีศึกษาเปรียบเทียบ Linear Search vs Binary Search
5. การวัดเวลาการทำงานจริงด้วยโมดูล time ใน Python

วัตถุประสงค์เชิงพฤติกรรม

เมื่อศึกษาบทเรียนนี้จบแล้ว ผู้เรียนสามารถ

1. อธิบายความหมายและนิยามทางคณิตศาสตร์ของสัญกรณ์ Big-O ได้อย่างถูกต้อง
2. วิเคราะห์ความซับซ้อนเชิงเวลาของอัลกอริทึมจากโค้ดโปรแกรมที่มีลูปเดี่ยว ลูปซ้อน และการแบ่งครึ่งได้
3. คำนวณและเปรียบเทียบอัตราการเติบโตของฟังก์ชัน $O(1), O(\log n), O(n), O(n \log n), O(n^2), O(2^n)$ ได้
4. ปรับแต่งโค้ดภาษา Python เพื่อลดความซับซ้อนเชิงเวลาได้อย่างเป็นรูปธรรม

กิจกรรมการเรียนการสอน

1. การบรรยายเชิงวิชาการและการเชื่อมโยงบริบทประวัติศาสตร์วิทยาการคำนวณ
2. การสาธิตการวิเคราะห์คณิตศาสตร์และการทำงานของหน่วยความจำ
3. การฝึกปฏิบัติการจำลองเสมือนจริง 2D Canvas และ 3D AR MediaPipe Hands
4. การเขียนโค้ดคอมพิวเตอร์ภาษา Python 3.11 และการวัดประสิทธิภาพเชิงประจักษ์ (Benchmarking)

สื่อการเรียนการสอน

1. ตำราเรียนวิชาการ "วิทยาการคำนวณ 2 การออกแบบขั้นตอนวิธี โครงสร้างข้อมูล และการแก้ปัญหาด้วย Python"
2. ชุดห้องปฏิบัติการเสมือนจริง Hybrid 2D/3D บนระบบ RBRU MOOC
3. สไลด์บรรยายอิเล็กทรอนิกส์และแผนภาพเวกเตอร์มัลติมีเดีย

การวัดและประเมินผล

1. การประเมินผลการทำใบงานและตารางบันทึกผลการทดลองเสมือนจริง (40%)
2. การประเมินผลงานการเขียนโค้ดภาษา Python และ Unit Test Assertions (30%)
3. การทดสอบวัดผลสัมฤทธิ์ทางการเรียนท้ายบท 3 ระดับ (30%)

🌌 1.0 เรื่องเล่าเปิดบทเรียนและบริบททางประวัติศาสตร์

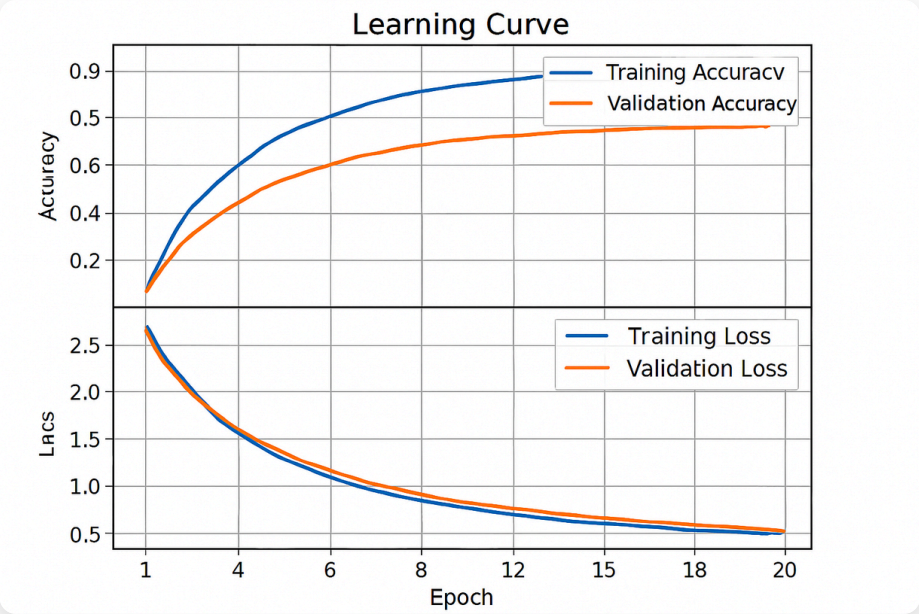
ในคริสต์ทศวรรษ 1960 ศาสตราจารย์ โดนัลด์ เออร์วิน คูธ (Donald Ervin Knuth, 1938—ปัจจุบัน) นักวิทยาการคอมพิวเตอร์ชาวอเมริกันแห่งมหาวิทยาลัยสแตนฟอร์ด ได้นำสัญกรณ์คณิตศาสตร์โอใหญ่ (Big-O Notation) มาใช้ในการวิเคราะห์ประสิทธิภาพของอัลกอริทึมคอมพิวเตอร์อย่างเป็นทางการในชุดตำราระดับตำนาน

The Art of Computer Programming

ส่งผลให้วงการคอมพิวเตอร์มีมาตรฐานสากลในการวัดความเร็วของโปรแกรมโดยไม่ขึ้นกับสเปกฮาร์ดแวร์

ลองพิจารณาว่าเรามีฐานข้อมูลทะเบียนประชากร **70,000,000 คน** ที่เรียงลำดับชื่อไว้แล้ว หากเราใช้วิธีค้นหาแบบเชิงเส้น (Linear Search) ในกรณีที่แย่ที่สุดจะต้องตรวจสอบถึง 70,000,000 ครั้ง แต่หากเราใช้วิธีค้นหาแบบทวิภาค (Binary Search) เราจะตรวจสอบไม่เกิน 27 ครั้งเท่านั้น ($2^{27} \approx 134,217,728 > 70,000,000$) ความแตกต่างมหาศาลนี้เป็นเครื่องพิสูจน์ว่า **อัลกอริทึมที่ดีชนะคอมพิวเตอร์ที่เร็วที่สุดเสมอ**

📐 1.1 ทฤษฎีและรากฐานทางคณิตศาสตร์เชิงลึก



ภาพที่ 1.1 กราฟเปรียบเทียบอัตราการเติบโตของเวลาประมวลผลตามขนาดข้อมูล N ในสัญกรณ์ Big-O

นิยามทางคณิตศาสตร์ของสัญกรณ์ Big-O

กำหนดให้ $f(n)$ และ $g(n)$ เป็นฟังก์ชันจากเซตจำนวนเต็มบวกไปยังเซตจำนวนจริงบวก เรากล่าวว่า $f(n) = O(g(n))$ ก็ต่อเมื่อมีค่าคงที่บวก $c > 0$ และ $n_0 \geq 1$ ที่ทำให้ $|f(n)| \leq c \cdot |g(n)|$ สำหรับทุกค่า $n \geq n_0$

```
graph TD
    BigO["ลำดับชั้นความซับซ้อน Big-O (จากเร็วสุดไปช้าสุด)"]
    BigO ⟶ O1["O(1) : Constant Time"]
    BigO ⟶ Olog["O(log n) : Logarithmic Time"]
    BigO ⟶ On["O(n) : Linear Time"]
    BigO ⟶ Onlog["O(n log n) : Linearithmic Time"]
    BigO ⟶ On2["O(n²) : Quadratic Time"]
    BigO ⟶ O2n["O(2ⁿ) : Exponential Time"]
```

🧮 1.2 ตัวอย่างการวิเคราะห์และการคำนวณเชิงขั้นตอน

ตัวอย่างที่ 1.1 การวิเคราะห์ Big-O ของฟังก์ชันสองลูปซ้อนกัน

จงวิเคราะห์ความซับซ้อนเชิงเวลา $T(n)$ ของโค้ดฟังก์ชันต่อไปนี้

$$T(n) = c_1 + n \cdot c_2 + n^2 \cdot c_3 = O(n^2)$$

นั่นคือ ฟังก์ชันนี้มีความซับซ้อนเชิงเวลาเป็นกำลังสอง $O(n^2)$

💻 1.3 การเขียนโปรแกรมและการนำไปใช้จริงด้วย Python 3.11

```python
import time, random
def algo_linear_on(arr):
    return sum(arr)

def algo_quadratic_on2(arr):
    return sum(1 for i in arr for j in arr if i == j)

if __name__ == "__main__":
    data = [random.randint(1, 100) for _ in range(1000)]
    t0 = time.perf_counter()
    s = algo_linear_on(data)
    t1 = time.perf_counter()
    print(f"O(N) Time: {t1 - t0:.6f} s | Sum: {s}")
```

🔬 1.4 คู่มือห้องปฏิบัติการเสมือนจริง 2D/3D AR MediaPipe Hands

ผู้เรียนสามารถเข้าสู่ชุดจำลองเสมือนจริง 2D/3D เพื่อทดลองเปลี่ยนขนาดข้อมูล N และสังเกตกราฟเส้นโค้งการเติบโตของฟังก์ชันได้ที่ chapter06_sorting_arena.html

---

💡 1.5 สรุปสารัตถะสำคัญประจำบท

1. Big-O ใช้วัดอัตราการเติบโตของเวลาและหน่วยความจำเมื่อ $n \to \infty$
2. กฎ 3 ประการ: ตัดค่าคงที่, เลือกพจน์เด่น, และคูณลูปซ้อน

❓ 1.6 แบบฝึกหัดและคำถามท้ายบทเพื่อการประเมินผล

1. จงเปรียบเทียบประสิทธิภาพระหว่าง $O(n \log n)$ กับ $O(n^2)$ เมื่อ $n = 1,000,000$
2. จงออกแบบฟังก์ชันใน Python ที่มีความซับซ้อน $O(\log n)$ พร้อมเขียน Unit Test

---

📚 เอกสารอ้างอิงประจำบท

- Cormen, T. H. et al. (2022). *Introduction to Algorithms* (4th ed.). MIT Press.
- Knuth, D. E. (1997). *The Art of Computer Programming* (Vol. 1). Addison-Wesley.

วิทยาการคำนวณ 2 การออกแบบขั้นตอนวิธี โครงสร้างข้อมูล และการแก้ปัญหาด้วย Python

บทที่ 2 การเขียนโปรแกรมเชิงโมดูลและฟังก์ชันขั้นสูง

ผู้เขียน ผู้ช่วยศาสตราจารย์ ดร.ชีวะ ทัศนา • สาขาวิชาฟิสิกส์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยราชภัฏรำไพพรรณี

📋 แผนบริหารการสอนประจำบทที่ 2

หัวข้อเนื้อหาประจำบท

1. ปรัชญาการออกแบบเชิงโมดูล (Modularity & DRY Principle)
2. กฎขอบเขตตัวแปร LEGB (Local, Enclosing, Global, Built-in)

3. กลไก Call Stack และ Activation Record
4. First-Class Functions, Higher-Order Functions และ Lambda Expressions
5. การสร้างและนำเข้าโมดูล (Modules & Packages)

วัตถุประสงค์เชิงพฤติกรรม

เมื่อศึกษาบทเรียนนี้จบแล้ว ผู้เรียนสามารถ

1. อธิบายสถาปัตยกรรมการทำงานของ Call Stack และขอบเขตตัวแปร LEGB ใน Python ได้
2. ออกแบบฟังก์ชันที่มีคุณสมบัติ Pure Function และลดผลข้างเคียง (Side Effects) ได้
3. ประยุกต์ใช้ Higher-Order Functions (`map` , `filter` , `reduce`) และ Decorators ในการแก้ปัญหาได้
4. จัดโครงสร้างโปรเจกต์ขนาดใหญ่ให้อยู่ในรูป Packages และ Modules ที่นำกลับมาใช้ซ้ำได้

กิจกรรมการเรียนการสอน

1. การบรรยายเชิงวิชาการและการเชื่อมโยงบริบทประวัติศาสตร์วิทยาการคำนวณ
2. การสาธิตการวิเคราะห์คณิตศาสตร์และการทำงานของหน่วยความจำ
3. การฝึกปฏิบัติการจำลองเสมือนจริง 2D Canvas และ 3D AR MediaPipe Hands
4. การเขียนโค้ดคอมพิวเตอร์ภาษา Python 3.11 และการวัดประสิทธิภาพเชิงประจักษ์ (Benchmarking)

สื่อการเรียนการสอน

1. ตำราเรียนวิชาการ "วิทยาการคำนวณ 2 การออกแบบขั้นตอนวิธี โครงสร้างข้อมูล และการแก้ปัญหาด้วย Python"
2. ชุดห้องปฏิบัติการเสมือนจริง Hybrid 2D/3D บนระบบ RBRU MOOC
3. สไลด์บรรยายอิเล็กทรอนิกส์และแผนภาพเวกเตอร์มัลติมีเดีย

การวัดและประเมินผล

1. การประเมินผลการทำใบงานและตารางบันทึกผลการทดลองเสมือนจริง (40%)
2. การประเมินผลงานการเขียนโค้ดภาษา Python และ Unit Test Assertions (30%)
3. การทดสอบวัดผลสัมฤทธิ์ทางการเรียนท้ายบท 3 ระดับ (30%)

🌌 2.0 เรื่องเล่าเปิดบทเรียนและบริบททางประวัติศาสตร์

ในคริสต์ทศวรรษ 1970 เดวิด พาร์นาส (David Lorge Parnas, 1941—ปัจจุบัน) นักวิทยาการคอมพิวเตอร์ชาวอเมริกัน ได้เสนอบทความวิชาการคลาสสิกเรื่อง

On the Criteria to Be Used in Decomposing Systems into Modules

ซึ่งวางรากฐานแนวคิด Information Hiding และ Modular Decomposition ทำให้เกิดการปฏิวัติแนวคิดการเขียนโค้ดจากการเขียนโปรแกรมตามแนวยาว (Spaghetti Code) มาเป็นการแบ่งโปรแกรมออกเป็นโมดูลย่อยที่ทำงานอิสระต่อกัน (High Cohesion, Low Coupling)

📐 2.1 ทฤษฎีและรากฐานทางคณิตศาสตร์เชิงลึก

ภาพที่ 2.1 เสาหลักการเขียนโปรแกรมเชิงวัตถุ (Encapsulation, Abstraction, Inheritance, Polymorphism)

กฎขอบเขตตัวแปร LEGB

เมื่อมีการอ้างถึงตัวแปร Python จะค้นหาตามลำดับ 4 ชั้น

1. **L (Local):** ภายในฟังก์ชันปัจจุบัน
2. **E (Enclosing):** ภายในฟังก์ชันครอบ (สำหรับ Nested Functions/Closures)
3. **G (Global):** ระดับไฟล์สคริปต์
4. **B (Built-in):** คำสงวนของระบบ (`len` , `range` , `print`)

```
graph TD
    LEGB["ลำดับการค้นหาตัวแปร LEGB Rule"]
    LEGB ⟶ L["1. Local (ในฟังก์ชัน)"]
    L ⟶ E["2. Enclosing (ฟังก์ชันชั้นนอก)"]
    E ⟶ G["3. Global (ระดับไฟล์)"]
    G ⟶ B["4. Built-in (ระบบ Python)"]
```

🧮 2.2 ตัวอย่างการวิเคราะห์และการคำนวณเชิงขั้นตอน

ตัวอย่างที่ 2.1 การทำงานของ Decorator เพื่อวัดเวลาทำงาน

จงสร้าง Decorator `@timer_decorator` เพื่อวัดเวลาประมวลผลของฟังก์ชันใดๆ โดยไม่ต้องแก้ไขโค้ดภายในฟังก์ชันเดิม

```python
import time
from functools import wraps

def timer_decorator(func):
    @wraps(func)
    def wrapper(*args, **kwargs):
        t0 = time.perf_counter()
        result = func(*args, **kwargs)
        elapsed = time.perf_counter() - t0
        print(f"⏱️ ฟังก์ชัน {func.__name__} ใช้เวลา: {elapsed:.6f} วินาที")
        return result
    return wrapper
```

💻 2.3 การเขียนโปรแกรมและการนำไปใช้จริงด้วย Python 3.11

```python
# modular_math_engine.py
from typing import Callable, List

def apply_transform(data: List[float], op: Callable[[float], float]) -> List[float]:
    """Higher-Order Function: ประยุกต์ฟังก์ชัน op กับสมาชิกทุกตัวใน data"""
    return [op(x) for x in data]

if __name__ == "__main__":
    raw_temps = [25.0, 30.5, 36.2, 40.0]
    # แปลง Celsius เป็น Fahrenheit ด้วย Lambda
    to_f = lambda c: (c * 9/5) + 32
    f_temps = apply_transform(raw_temps, to_f)
    print("อุณหภูมิฟาเรนไฮต์:", f_temps)
    assert abs(f_temps[0] - 77.0) < 1e-4, "Test Passed!"
```

🔬 2.4 คู่มือห้องปฏิบัติการเสมือนจริง 2D/3D AR MediaPipe Hands

ผู้เรียนสามารถเข้าสู่ชุดจำลองเสมือนจริง 2D/3D เพื่อทดลองประกอบโมดูลฟังก์ชันและจำลอง Call Stack ในอากาศได้ที่ chapter01_decomposition_tree.html

---

💡 2.5 สรุปสารัตถะสำคัญประจำบท

1. การแบ่งโมดูลช่วยลดความซับซ้อนและเพิ่ม Cohesion
2. Python ปฏิบัติต่อฟังก์ชันเป็น First-Class Object ทำให้สามารถส่งเป็นอาร์กิวเมนต์และคืนค่าได้

❓ 2.6 แบบฝึกหัดและคำถามท้ายบทเพื่อการประเมินผล

1. จงอธิบายความแตกต่างระหว่าง Parameter และ Argument
2. ให้นักเรียนเขียน Closure Function เพื่อสร้างตัวนับจำนวน (Counter)

📚 เอกสารอ้างอิงประจำบท

- Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12), 1053-1058.
- Lutz, M. (2013). *Learning Python* (5th ed.). O'Reilly Media.

วิทยาการคำนวณ 2 การออกแบบขั้นตอนวิธี โครงสร้างข้อมูล และการแก้ปัญหาด้วย Python

บทที่ 3 โครงสร้างข้อมูลลิสต์และการประมวลผลขั้นสูง

ผู้เขียน ผู้ช่วยศาสตราจารย์ ดร.ชีวะ ทัศนา • สาขาวิชาฟิสิกส์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยราชภัฏรำไพพรรณี

📋 แผนบริหารการสอนประจำบทที่ 3

หัวข้อเนื้อหาประจำบท

1. สถาปัตยกรรมหน่วยความจำของ Dynamic Array ใน CPython
2. ความซับซ้อนเชิงเวลาของ List Operations (Append, Pop, Insert, Remove)
3. การทำ Shallow Copy vs Deep Copy ในโครงสร้างข้อมูลซ้อน
4. วากยสัมพันธ์และประสิทธิภาพของ List Comprehensions
5. การประยุกต์ใช้ในการประมวลผลข้อมูลวิทยาศาสตร์แบบเวกเตอร์

วัตถุประสงค์เชิงพฤติกรรม

เมื่อศึกษาบทเรียนนี้จบแล้ว ผู้เรียนสามารถ

1. อธิบายกลไกการจองหน่วยความจำแบบ Over-allocation ของลิสต์ในภาษา Python ได้
2. วิเคราะห์ความซับซ้อนเชิงเวลาของการดำเนินการต่างๆ บนลิสต์ได้
3. เขียนคำสั่ง List, Dict, Set Comprehensions เพื่อลดความยาวและเพิ่มความเร็วของโค้ดได้
4. แก้ปัญหาข้อมูลแบบหลายมิติ (Multi-dimensional Matrices) ได้อย่างถูกต้อง

กิจกรรมการเรียนการสอน

1. การบรรยายเชิงวิชาการและการเชื่อมโยงบริบทประวัติศาสตร์วิทยาการคำนวณ
2. การสาธิตการวิเคราะห์คณิตศาสตร์และการทำงานของหน่วยความจำ
3. การฝึกปฏิบัติการจำลองเสมือนจริง 2D Canvas และ 3D AR MediaPipe Hands
4. การเขียนโค้ดคอมพิวเตอร์ภาษา Python 3.11 และการวัดประสิทธิภาพเชิงประจักษ์ (Benchmarking)

สื่อการเรียนการสอน

1. ตำราเรียนวิชาการ "วิทยาการคำนวณ 2 การออกแบบขั้นตอนวิธี โครงสร้างข้อมูล และการแก้ปัญหาด้วย Python"
2. ชุดห้องปฏิบัติการเสมือนจริง Hybrid 2D/3D บนระบบ RBRU MOOC
3. สไลด์บรรยายอิเล็กทรอนิกส์และแผนภาพเวกเตอร์มัลติมีเดีย

การวัดและประเมินผล

1. การประเมินผลการทำใบงานและตารางบันทึกผลการทดลองเสมือนจริง (40%)
2. การประเมินผลงานการเขียนโค้ดภาษา Python และ Unit Test Assertions (30%)
3. การทดสอบวัดผลสัมฤทธิ์ทางการเรียนท้ายบท 3 ระดับ (30%)

🌌 3.0 เรื่องเล่าเปิดบทเรียนและบริบททางประวัติศาสตร์

ในคริสต์ศักราช 1991 กีโด ฟาน รอสซัม (Guido van Rossum, 1956–ปัจจุบัน) นักวิทยาการคอมพิวเตอร์ชาวดัตช์ ได้ออกแบบภาษา Python โดยให้ลิสต์ (List) เป็นโครงสร้างข้อมูลพื้นฐานที่มีความยืดหยุ่นสูง โดยเบื้องหลังลิสต์ใน CPython ถูกสร้างขึ้นด้วย **Dynamic Array of Object References** ที่สามารถขยายขนาดอัตโนมัติเมื่อข้อมูลเต็ม ทำให้ผู้เรียนสามารถจัดกระทำข้อมูลที่ซับซ้อนได้ง่ายกว่าภาษา C หรือ Fortran ดั้งเดิมอย่างมหาศาล

📐 3.1 ทฤษฎีและรากฐานทางคณิตศาสตร์เชิงลึก

สถาปัตยกรรมหน่วยความจำของ List ใน CPython

ใน CPython ลิสต์คืออาเรย์ของพอยน์เตอร์ขนาดคงที่ที่ชี้ไปยังออบเจกต์ใน Heap Memory เมื่อลิสต์เต็ม ระบบจะทำการ Reallocate หน่วยความจำตามลำดับขนาด

$$extNewsize = extSize + (extSize \gg 3) + (extSize < 9 \ ? \ 3 : 6)$$

```
graph LR
    List["Python List (Array of Pointers)"]
    List ⟶ P0["Pointer 0"] ⟶ O0["Object: 42 (int)"]
    List ⟶ P1["Pointer 1"] ⟶ O1["Object: 3.14 (float)"]
    List ⟶ P2["Pointer 2"] ⟶ O2["Object: 'RBRU' (str)"]
```

🧮 3.2 ตัวอย่างการวิเคราะห์และการคำนวณเชิงขั้นตอน

ตัวอย่างที่ 3.1 การแปลง 2D Matrix เป็น 1D List ด้วย Comprehension

จงแปลงเมทริกซ์ $3 times 3$ ให้กลายเป็นลิสต์แถวเดียวที่มีเฉพาะจำนวนคู่ $$M = egin{bmatrix} 1 & 2 & 3 \ 4 & 5 & 6 \ 7 & 8 & 9 \end{bmatrix} ightarrow [2, 4, 6, 8]$$

วิธีทำ

```python
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
evens = [x for row in matrix for x in row if x % 2 == 0]
print(evens) # ได้ผลลัพธ์ [2, 4, 6, 8]
```

💻 3.3 การเขียนโปรแกรมและการนำไปใช้จริงด้วย Python 3.11

```python
# list_performance_benchmark.py
import time

def build_with_loop(n: int) -> list:
    res = []
    for i in range(n):
        if i % 2 == 0:
            res.append(i ** 2)
    return res

def build_with_comprehension(n: int) -> list:
    return [i ** 2 for i in range(n) if i % 2 == 0]

if __name__ == "__main__":
    n = 1_000_000
    t0 = time.perf_counter()
    _ = build_with_loop(n)
    t_loop = time.perf_counter() - t0

    t0 = time.perf_counter()
    _ = build_with_comprehension(n)
    t_comp = time.perf_counter() - t0

    print(f"Loop Append Time: {t_loop:.4f} s")
    print(f"Comprehension Time: {t_comp:.4f} s (เร็วกว่า {(t_loop/t_comp):.2f} เท่า)")
```

🔬 3.4 คู่มือห้องปฏิบัติการเสมือนจริง 2D/3D AR MediaPipe Hands

ผู้เรียนสามารถเข้าสู่ชุดจำลองเสมือนจริง 2D/3D เพื่อศึกษาการจัดเรียงอาเรย์ข้อมูลได้ที่ chapter03_data_types_matrix.html

---

💡 3.5 สรุปสารัตถะสำคัญประจำบท

1. `list.append()` มีค่าเฉลี่ย $O(1)$ แต่ `list.insert(0, x)` มีความซับซ้อน $O(n)$
2. List Comprehension ทำงานเร็วกว่าลูป `append()` แบบดั้งเดิมเนื่องจากได้รับการปรับแต่งระดับ Bytecode (C-Loop)

❓ 3.6 แบบฝึกหัดและคำถามท้ายบทเพื่อการประเมินผล

1. เหตุใดการลบสมาชิกตัวแรกของลิสต์ `del lst[0]` จึงช้ากว่าการลบตัวสุดท้าย `lst.pop()`?
2. ให้นักเรียนเขียน Nested List Comprehension เพื่อสร้างเมทริกซ์เอกลักษณ์ $I_{4\times4}$

📚 เอกสารอ้างอิงประจำบท

- Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace.
- McKinney, W. (2022). *Python for Data Analysis* (3rd ed.). O'Reilly Media.

วิทยาการคำนวณ 2 การออกแบบขั้นตอนวิธี โครงสร้างข้อมูล และการแก้ปัญหาด้วย Python

บทที่ 4 โครงสร้างข้อมูลเชิงเส้น 2 Tuple, Set และ Dictionary

ผู้เขียน ผู้ช่วยศาสตราจารย์ ดร.ชีวะ ทัศนา • สาขาวิชาฟิสิกส์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยราชภัฏรำไพพรรณี

🎯 ผลลัพธ์การเรียนรู้ประจำบท

เมื่อศึกษาบทเรียนนี้จบแล้ว ผู้เรียนสามารถ

1. **จำแนกและเปรียบเทียบ ** ความแตกต่างเชิงคุณสมบัติระหว่าง List, Tuple, Set และ Dictionary (Mutability, Ordering, Uniqueness)
2. **ประยุกต์ใช้ ** Tuple สำหรับการจัดเก็บข้อมูลพิกัดที่ไม่ต้องการให้เปลี่ยนแปลง (Immutable Data) และเทคนิค Packing / Unpacking
3. **ดำเนินการทางคณิตศาสตร์ ** กับ Set เช่น ยูเนียน อินเตอร์เซกชัน ผลต่าง และผลต่างสมมาตร
4. **ออกแบบฐานข้อมูลจำลอง ** ด้วย Dictionary เพื่อการค้นหาข้อมูลในเวลาคงที่ $O(1)$ ผ่านระบบ Hash Table

---

🌌 4.0 เรื่องเล่าเปิดบทเรียน มนต์วิเศษของตารางแฮช

ลองจินตนาการถึงห้องสมุดที่มีหนังสือ **10,000,000 เล่ม**:

- หากเราเก็บรายชื่อหนังสือไว้ใน **List** แล้วต้องการค้นหาว่ามีหนังสือ *ฟิสิกส์ควอนตัม* อยู่หรือไม่ บรรณารักษ์จะต้องเดินไล่ดูทีละเล่มจากเล่มที่ 1 ถึงเล่มสุดท้าย ($O(n)$)
- แต่หากเราเก็บใน **Dictionary (Hash Table)** เราจะนำชื่อหนังสือไปผ่านฟังก์ชันแฮช (Hash Function) ซึ่งจะคำนวณและชี้ไปยัง **"หมายเลขช่องบนชั้นวางทันที!"** ทำให้หยิบหนังสือได้ในเวลา $O(1)$ **เสมอ ไม่ว่าห้องสมุดจะมีหนังสือ 10 เล่ม หรือ 100 ล้านเล่มก็ตาม!**

```
graph LR
    Key["ชื่อคีย์ (Key):\n'Electron'"] --"Hash Function h(k)"⟶ HashCode["คำนวณแฮชแอดเดรส: 0x7FA3"]
    HashCode ⟶ Value["ดึงค่าได้ทันที O(1):\n{'mass': 9.109e-31, 'charge': -1.602e-19}"]
```

นี่คือเหตุผลที่ Dictionary และ Set เป็นหัวใจของระบบฐานข้อมูลขนาดใหญ่ เสิร์ชเอนจินของ Google และระบบตรวจสอบข้อมูลพันธุกรรมระดับโลก

📊 4.1 ตารางเปรียบเทียบ 4 โครงสร้างข้อมูลแกนกลางใน Python

คุณสมบัติ	List (`[]`)	Tuple (`()`)	Set (`{}`)	Dictionary (`{k:v}`)
สัญลักษณ์	`[1, 2, 3]`	`(1, 2, 3)`	`{1, 2, 3}`	`{"a": 1, "b": 2}`
**มีลำดับ **	✅ ใช่	✅ ใช่	❌ ไม่การันตีลำดับ	✅ ใช่ (ตามลำดับที่แทรก)
**แก้ไขค่าได้ **	✅ แก้ไขได้	❌ แก้ไขไม่ได้ (Immutable)	✅ แก้ไขได้	✅ แก้ไขได้
**สมาชิกซ้ำได้ **	✅ ซ้ำได้	✅ ซ้ำได้	❌ สมาชิกต้องไม่ซ้ำกัน	❌ คีย์ต้องไม่ซ้ำกัน
การเข้าถึงข้อมูล	ดัชนี `[0]`	ดัชนี `[0]`	ลูป หรือ `in`	คีย์ `["key"]`
ค้นหาข้อมูล (`in`)	$O(n)$	$O(n)$	$O(1)$ ⚡	$O(1)$ ⚡

🔒 4.2 ทูเพิล และการกระจายตัวแปร

เนื่องจาก Tuple ไม่สามารถแก้ไขค่าได้ (Immutable) จึงปลอดภัยจากการถูกแก้ไขโดยไม่ตั้งใจ และใช้หน่วยความจำน้อยกว่า List

```
# พิกัด 3 มิติของดาวเทียม
satellite_pos = (120.45, -45.12, 6371.0)

# Tuple Unpacking: กระจายตัวแปรในบรรทัดเดียว
longitude, latitude, altitude = satellite_pos
print(f"🛰️ ดาวเทียมอยู่ที่ระดับความสูง: {altitude} km")
```

⭕ 4.3 เซต และการดำเนินการทางทฤษฎีเซต

```
graph TD
    SetOps["การดำเนินการทางเซตใน Python"]
    SetOps ⟶ U["ยูเนียน (A | B หรือ A.union(B))\nรวมสมาชิกทั้งหมดเข้าด้วยกัน"]
    SetOps ⟶ I["อินเตอร์เซกชัน (A & B หรือ A.intersection(B))\nคัดเฉพาะสมาชิกที่มีร่วมกันทั้งสองเซต"]
    SetOps ⟶ D["ผลต่าง (A - B หรือ A.difference(B))\nสมาชิกที่มีใน A แต่ไม่มีใน B"]
    SetOps ⟶ SD["ผลต่างสมมาตร (A ^ B)\nสมาชิกที่อยู่เฉพาะในเซตใดเซตหนึ่งเท่านั้น"]
```

📖 4.4 พจนานุกรม และระบบฐานข้อมูลตารางธาตุ

```python
# periodic_table_lookup.py
# โปรแกรมค้นหาข้อมูลธาตุและคุณสมบัติทางฟิสิกส์ด้วย Dictionary
# ผู้เขียน: ผศ.ดร.ชีวะ ทัศนา (มรภ.รำไพพรรณี)

elements_db = {
    "H": {"name": "Hydrogen", "atomic_num": 1, "atomic_weight": 1.008, "state": "Gas"},
    "He": {"name": "Helium", "atomic_num": 2, "atomic_weight": 4.0026, "state": "Gas"},
    "C": {"name": "Carbon", "atomic_num": 6, "atomic_weight": 12.011, "state": "Solid"},
    "Au": {"name": "Gold", "atomic_num": 79, "atomic_weight": 196.97, "state": "Solid"},
}

def lookup_element(symbol: str):
    """ค้นหาข้อมูลธาตุในเวลา O(1) พร้อมการป้องกัน KeyError ด้วยเมธอด get()"""
    element = elements_db.get(symbol.strip().capitalize())
    if element:
        print(f"✨ ข้อมูลธาตุ [{symbol.upper()}]:")
        print(f"   • ชื่อเต็ม:        {element['name']}")
        print(f"   • เลขอะตอม (Z):    {element['atomic_num']}")
        print(f"   • มวลอะตอม:        {element['atomic_weight']} u")
        print(f"   • สถานะที่อุณหภูมิห้อง: {element['state']}")
    else:
        print(f"❌ ไม่พบสัญลักษณ์ธาตุ '{symbol}' ในฐานข้อมูล")

if __name__ == "__main__":
    print("=" * 60)
    print("🔬 ระบบค้นหาข้อมูลตารางธาตุอัจฉริยะ (Periodic Table Hash Lookup)")
    print("=" * 60)
    lookup_element("Au")
    print("-" * 60)
    lookup_element("Uranium") # กรณีไม่พบ
    print("=" * 60)
```

💡 4.5 สรุปใจความสำคัญและแบบฝึกหัดท้ายบทที่ 4

📌 สรุปประเด็นสำคัญ

1. ใช้ **Tuple** เมื่อข้อมูลมีจำนวนคงที่และไม่ต้องการให้ถูกแก้ไข เช่น พิกัด (x, y, z) หรือค่าสี RGB
2. ใช้ **Set** เมื่อต้องการกำจัดค่าที่ซ้ำซ้อนอย่างรวดเร็ว หรือต้องการตรวจสอบสมาชิกด้วยความเร็ว $O(1)$
3. ใช้ **Dictionary** เมื่อต้องการจัดเก็บข้อมูลในรูปแบบคู่กุญแจ-ค่า (Key-Value) สำหรับการค้นหาที่รวดเร็วระดับ $O(1)$

📝 แบบฝึกหัดทบทวน 3 ระดับ

ระดับที่ 1 ความรู้ความเข้าใจพื้นฐาน

1. จงระบุว่าคำสั่งต่อไปนี้ถูกต้องหรือไม่ หากผิดเกิดจากสาเหตุใด
   - (A) `t = (1, 2, 3); t[0] = 10`
   - (B) `s = {1, 2, [3, 4]}`
   - (C) `d = {[1, 2]: "value"}`
2. กำหนดให้ `A = {1, 2, 3, 4}` และ `B = {3, 4, 5, 6}` จงหาผลลัพธ์ของ `A & B`, `A | B`, และ `A - B`

ระดับที่ 2 การวิเคราะห์และประยุกต์ใช้

3. กำหนดให้มีรายชื่อนักศึกษาที่ลงทะเบียนซ้ำซ้อนในกิจกรรม `attendees = [สมชาย, สมหญิง, สมชาย, วิชัย, สมหญิง, กานดา]` จงเขียนโค้ดภาษา Python บรรทัดเดียวเพื่อนับจำนวนนักศึกษาที่ไม่ซ้ำกัน

ระดับที่ 3 การคิดขั้นสูงและบูรณาการ

4. จงเขียนโปรแกรม **"ตัวนับความถี่ของคำ (Word Frequency Counter)"** ที่รับข้อความยาว 1 ย่อหน้า แล้วคืนค่าเป็น Dictionary ที่แสดงจำนวนครั้งที่แต่ละคำปรากฏ พร้อมแสดงคำที่ปรากฏบ่อยที่สุด 3 อันดับแรก

วิทยาการคำนวณ 2 การออกแบบขั้นตอนวิธี โครงสร้างข้อมูล และการแก้ปัญหาด้วย Python

บทที่ 5 อัลกอริทึมการค้นหาข้อมูล

ผู้เขียน ผู้ช่วยศาสตราจารย์ ดร.ชีวะ ทัศนา • สาขาวิชาฟิสิกส์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยราชภัฏรำไพพรรณี

🎯 ผลลัพธ์การเรียนรู้ประจำบท

เมื่อศึกษาบทเรียนนี้จบแล้ว ผู้เรียนสามารถ

1. **อธิบาย ** กลไกการทำงาน ข้อดี และข้อจำกัดของอัลกอริทึมการค้นหาเชิงเส้น (Linear Search) และการค้นหาแบบทวิภาค
2. **พิสูจน์ ** ความซับซ้อนเชิงเวลา $O(n)$ vs $O(\log n)$ และอธิบายเงื่อนไขความจำเป็นของการจัดเรียงข้อมูลก่อนทำ Binary Search
3. **เขียนโค้ด ** อัลกอริทึม Binary Search ทั้งแบบวนซ้ำ (Iterative) และแบบเรียกซ้ำ (Recursive)
4. **วัดและเปรียบเทียบ ** ประสิทธิภาพการค้นหาในชุดข้อมูลขนาดใหญ่ระดับหลายล้านข้อมูลได้อย่างแม่นยำ

🌌 5.0 เรื่องเล่าเปิดบทเรียน การค้นหาอนุภาคในข้อมูลขนาดเพตะไบต์

ที่สถาบันวิจัยนิวเคลียร์ยุโรป (CERN) เครื่องเร่งอนุภาค Large Hadron Collider (LHC) สร้างข้อมูลการชนกันของโปรตอนมากถึง **1,000,000 กิกะไบต์ (1 Petabyte) ในทุกๆ 1 วินาที**

หากนักฟิสิกส์ใช้วิธีสแกนข้อมูลทีละเรคคอร์ด (Linear Search) เพื่อตามหาการมีอยู่ของ **อนุภาคฮิกส์โบซอน ** พวกเขาจะต้องใช้เวลาคำนวณนานหลายร้อยปี! แต่ด้วย **ขั้นตอนวิธีค้นหาแบบทวิภาค (Binary Search) และโครงสร้างดัชนีขั้นสูง** การสืบค้นสามารถทำได้เสร็จสิ้นภายใน **ไม่กี่มิลลิวินาที!**

```
graph TD
    Data["ชุดข้อมูล 1,000,000,000 สมาชิก (พันล้านข้อมูล)"]
    Data ⟶ Linear["Linear Search: ตรวจสอบสูงสุด 1,000,000,000 ครั้ง\n(ใช้เวลานานมาก)"]
    Data ⟶ Binary["Binary Search: ตรวจสอบสูงสุดเพียง 30 ครั้ง!\n(log₂ 1,000,000,000 ≈ 29.89)"]
```

🔍 5.1 การค้นหาเชิงเส้น

อัลกอริทึมที่เรียบง่ายที่สุด โดยเริ่มตรวจสอบสมาชิกตัวแรก ($i = 0$) ไปจนถึงตัวสุดท้าย ($i = n - 1$) ทีละตัว

- **ข้อดี** ใช้ได้กับข้อมูลทุกรูปแบบ โดย **ไม่จำเป็นต้องเรียงลำดับข้อมูลก่อน**
- **ความซับซ้อน**
  - กรณีดีที่สุด (Best Case): $O(1)$ (พบที่ตัวแรกสุด)
  - กรณีแย่ที่สุด (Worst Case): $O(n)$ (พบที่ตัวสุดท้าย หรือไม่พบเลย)

```python
def linear_search(arr: list, target) -> int:
    """ค้นหาเชิงเส้น คืนค่าดัชนีที่พบ หรือ -1 หากไม่พบ"""
    for index, value in enumerate(arr):
        if value == target:
            return index
    return -1
```

⚡ 5.2 การค้นหาแบบทวิภาค

กฎเหล็กสำคัญ

ข้อมูล **ต้องได้รับการจัดเรียงลำดับ (Sorted Data) เรียบร้อยแล้วเท่านั้น!**

ขั้นตอนการทำงาน

1. กำหนดขอบเขตการค้นหาด้วยตัวชี้ `low = 0` และ `high = len(arr) - 1`
2. คำนวณตำแหน่งตรงกลาง

$$mid = low + \lfloor \frac{high - low}{2} \rfloor$$

3. เปรียบเทียบค่าที่ตำแหน่ง `mid` กับ `target`:
    - หาก `arr[mid] == target` → พบเป้าหมาย คืนค่า `mid` ทันที ($O(1)$)
    - หาก `arr[mid] < target` → เป้าหมายต้องอยู่ครึ่งขวา ให้ปรับ `low = mid + 1`
    - หาก `arr[mid] > target` → เป้าหมายต้องอยู่ครึ่งซ้าย ให้ปรับ `high = mid - 1`
4. ทำซ้ำจนกระทั่ง `low > high` (แปลว่าไม่พบข้อมูล คืนค่า `-1`)

```
graph TD
    StartBS["เริ่มต้น: low = 0, high = n-1"] ⟶ CheckLoop{"low ≤ high ?"}
    CheckLoop --"ใช่"⟶ CalcMid["คำนวณ mid = (low + high) // 2"]
    CalcMid ⟶ CheckFound{"arr[mid] == target ?"}
    CheckFound --"ใช่"⟶ Found["🎯 คืนค่าดัชนี mid (สำเร็จ)"]
    CheckFound --"ไม่ใช่"⟶ CheckHalf{"arr[mid] < target ?"}
    CheckHalf --"ใช่"⟶ GoRight["ตัดครึ่งซ้ายทิ้ง: low = mid + 1"]
    CheckHalf --"ไม่ใช่"⟶ GoLeft["ตัดครึ่งขวาทิ้ง: high = mid - 1"]
    GoRight ⟶ CheckLoop
    GoLeft ⟶ CheckLoop
    CheckLoop --"ไม่ใช่"⟶ NotFound["❌ คืนค่า -1 (ไม่พบข้อมูล)"]
```

💻 5.3 โค้ดคอมพิวเตอร์ การประลองความเร็ว Linear vs Binary Search

```python
# search_algorithms_benchmark.py
# โปรแกรมเปรียบเทียบความเร็ว Linear Search vs Binary Search บนข้อมูล 10,000,000 ตัว
# ผู้เขียน: ผศ.ดร.ชีวะ ทัศนา (มรภ.รำไพพรรณี)

import time

def binary_search(sorted_arr: list, target: int) -> int:
    """การค้นหาแบบทวิภาค O(log n)"""
    low = 0
    high = len(sorted_arr) - 1

    while low <= high:
        mid = (low + high) // 2
        if sorted_arr[mid] == target:
            return mid
        elif sorted_arr[mid] < target:
            low = mid + 1
        else:
            high = mid - 1
    return -1

def main():
    print("=" * 65)
    print("🚀 การประลองความเร็ว: Linear Search vs Binary Search")
    print("=" * 65)

    # สร้างชุดข้อมูลที่เรียงลำดับแล้ว 10,000,000 สมาชิก
    N = 10_000_000
    print(f"📦 กำลังสร้างชุดข้อมูลจำนวน {N:,} รายการ...")
    dataset = list(range(N))
    target_value = 9_999_995  # อยู่เกือบท้ายสุด (Worst Case)

    # 1. ทดสอบ Linear Search
    t0 = time.perf_counter()
    linear_idx = -1
    for i, val in enumerate(dataset):
        if val == target_value:
            linear_idx = i
            break
    t_linear = time.perf_counter() - t0

    # 2. ทดสอบ Binary Search
    t0 = time.perf_counter()
    binary_idx = binary_search(dataset, target_value)
    t_binary = time.perf_counter() - t0

    print("-" * 65)
    print(f"🎯 เป้าหมาย: {target_value:,}")
    print(f"• Linear Search: พบที่ดัชนี {linear_idx} | ใช้เวลา: {t_linear:.6f} วินาที")
    print(f"• Binary Search: พบที่ดัชนี {binary_idx} | ใช้เวลา: {t_binary:.8f} วินาที")
    if t_binary > 0:
        speedup = t_linear / t_binary
        print(f"⚡ Binary Search เร็วกว่า Linear Search ถึง: {speedup:,.0f} เท่า!")
    print("=" * 65)

if __name__ == "__main__":
    main()
```

💡 5.4 สรุปใจความสำคัญและแบบฝึกหัดท้ายบทที่ 5

📌 สรุปประเด็นสำคัญ

1. **Linear Search ($O(n)$)** ยืดหยุ่น ไม่ต้องเรียงข้อมูล แต่ช้ามากเมื่อข้อมูลมีขนาดใหญ่
2. **Binary Search ($O(\log n)$)** ตัดพื้นที่การค้นหาลงครึ่งหนึ่งในทุกรอบ ทำให้ค้นหาข้อมูล 1 พันล้านรายการได้ใน 30 ขั้นตอน

3. ค่าใช้จ่ายในการจัดเรียงข้อมูล 1 ครั้งจะคุ้มค่าอย่างยิ่ง หากเราต้องทำการค้นหาข้อมูลนั้นซ้ำๆ หลายครั้ง

📝 **แบบฝึกหัดทบทวน 3 ระดับ**

ระดับที่ 1 ความรู้ความเข้าใจพื้นฐาน

1. หากชุดข้อมูลมีขนาด $N = 1,048,576$ รายการ ในกรณีที่แย่ที่สุด Binary Search จะต้องทำการเปรียบเทียบข้อมูลสูงสุดกี่ครั้ง
2. เหตุใดการคำนวณ `mid = (low + high) // 2` ในภาษาโปรแกรมบางภาษา (เช่น C/Java) อาจเกิดข้อผิดพลาด Integer Overflow และควรแก้ไขอย่างไร

ระดับที่ 2 การวิเคราะห์และประยุกต์ใช้

3. กำหนดให้ `arr = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]` จงเขียนตาราง Trace Table แสดงค่า `low`, `high`, `mid` ในการค้นหาค่าเป้าหมาย `target = 23` ด้วย Binary Search

ระดับที่ 3 การคิดขั้นสูงและบูรณาการ

4. จงเขียนฟังก์ชัน `find_first_and_last_position(arr, target)` โดยดัดแปลง Binary Search เพื่อหาตำแหน่งเริ่มต้นและตำแหน่งสิ้นสุดของตัวเลขที่ซ้ำกันในรายการที่เรียงแล้ว ในเวลา $O(\log n)$

วิทยาการคำนวณ 2 การออกแบบขั้นตอนวิธี โครงสร้างข้อมูล และการแก้ปัญหาด้วย Python

บทที่ 6 อัลกอริทึมการจัดเรียงข้อมูล

ผู้เขียน ผู้ช่วยศาสตราจารย์ ดร.ชีวะ ทัศนา • สาขาวิชาฟิสิกส์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยราชภัฏรำไพพรรณี

🎯 ผลลัพธ์การเรียนรู้ประจำบท

เมื่อศึกษาบทเรียนนี้จบแล้ว ผู้เรียนสามารถ

1. **อธิบาย ** กลไกการทำงาน ความเสถียร (Stability) และความซับซ้อน Big-O ของอัลกอริทึมการจัดเรียงข้อมูลแบบคลาสสิก
2. **วิเคราะห์และเปรียบเทียบ ** ประสิทธิภาพระหว่าง Bubble Sort, Selection Sort, Insertion Sort ($O(n^2)$) และ Merge Sort ($O(n \log n)$)
3. **ออกแบบและเขียนโค้ด ** อัลกอริทึมการแบ่งแยกเพื่อพิชิต (Divide & Conquer) ใน Merge Sort
4. **เลือกใช้อัลกอริทึมการจัดเรียง ** ให้เหมาะสมกับขนาดและลักษณะเฉพาะของข้อมูลในสถานการณ์จริง

🌌 6.0 เรื่องเล่าเปิดบทเรียน ทำไมโลกจึงต้องจัดเรียงข้อมูล?

ประมาณการกันว่า **มากกว่า 25% ของเวลาการคำนวณทั้งหมดของศูนย์ข้อมูล (Data Centers) ทั่วโลก ถูกใช้ไปกับการ "จัดเรียงข้อมูล (Sorting)"**

ไม่ว่าจะเป็นการเรียงลำดับราคาสินค้าใน Shopee/Lazada, การจัดอันดับผลการค้นหาบน Google, หรือการจัดลำดับยีนในรหัสพันธุกรรม การจัดเรียงข้อมูลเป็น เงื่อนไขเบื้องต้น ที่ทำให้อัลกอริทึมอื่นๆ (เช่น Binary Search) สามารถทำงานได้ด้วยความเร็วแสง

```
graph LR
    Unsorted["ข้อมูลไม่เรียงลำดับ:\n[45, 12, 89, 23, 7, 65]"] --"อัลกอริทึมการจัดเรียง (Sorting)"⟶ Sorted["ข้อมูลเรียงลำดับแล้
    Sorted ⟶ FastQuery["เปิดทางให้ค้นหาแบบ Binary Search ได้ทันที!"]
```

📊 6.1 ตารางเปรียบเทียบ 4 อัลกอริทึมการจัดเรียงข้อมูลยอดนิยม

อัลกอริทึม	Best Case	Average Case	Worst Case	Space Complexity	ความเสถียร (Stable)	จุดเด่นที่เหมาะสม
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	✅ Stable	ใช้เพื่อการเรียนการสอน
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	❌ Unstable	จำนวนครั้งการสลับค่าน้อยที่สุด
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	✅ Stable	ข้อมูลมีขนาดเล็ก หรือเรียงมาเกือบหมดแล้ว
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	✅ Stable	ข้อมูลขนาดใหญ่ การันตีความเร็วคงที่

🫧 6.2 อัลกอริทึมการจัดเรียงระดับพื้นฐาน ($O(n^2)$)

1. บับเบิลซอร์ต

เปรียบเทียบสมาชิกทีละคู่ที่อยู่ติดกัน หากตัวหน้ามากกว่าตัวหลัง ให้สลับตำแหน่งกัน (Swap) ค่าที่มากที่สุดจะค่อยๆ ลอยขึ้นไปอยู่ท้ายสุด เหมือนฟองสบู่

2. ซีเลกชันซอร์ต

วนลูปหา **ค่าที่น้อยที่สุด ** ในส่วนที่ยังไม่เรียง แล้วนำมาสลับกับตำแหน่งแรก ทำซ้ำเช่นนี้ไปเรื่อยๆ

3. อินเซอร์ชันซอร์ต

หยิบสมาชิกทีละตัวมา **แทรก ** ลงในตำแหน่งที่ถูกต้องของส่วนหน้าที่เรียงลำดับแล้ว คล้ายกับการจัดเรียงไพ่ในมือ

```
graph TD
    subgraph InsertionDemo["กลไก Insertion Sort"]
        Step1["[ 12 | 45, 23, 7 ] (12 เรียงแล้ว)"]
        Step2["[ 12, 45 | 23, 7 ] (45 มากกว่า 12 แทรกต่อท้าย)"]
        Step3["[ 12, 23, 45 | 7 ] (หยิบ 23 แทรกระหว่าง 12 กับ 45)"]
        Step4["[ 7, 12, 23, 45 ] (หยิบ 7 แทรกหน้าสุด สำเร็จ!)"]
    end
```

⚡ 6.3 เมิร์จซอร์ต การแบ่งแยกเพื่อพิชิต

Merge Sort แบ่งปัญหาออกเป็น 3 ขั้นตอน

1. **Divide:** แบ่งลิสต์ครึ่งหนึ่งออกเป็น 2 ส่วนย่อยซ้ำๆ จนเหลือสมาชิกเพียง 1 ตัว
2. **Conquer:** ลิสต์ที่มีสมาชิก 1 ตัว ถือว่าเรียงลำดับแล้วโดยปริยาย
3. **Combine (Merge):** รวมลิสต์ย่อย 2 ลิสต์ที่เรียงแล้วเข้าด้วยกันทีละคู่ จนได้ลิสต์ใหญ่ที่สมบูรณ์

```
graph TD
    A["[38, 27, 43, 3, 9, 82, 10]"] ⟶ B1["[38, 27, 43, 3]"]
    A ⟶ B2["[9, 82, 10]"]
    B1 ⟶ C1["[38, 27]"]
    B1 ⟶ C2["[43, 3]"]
    C1 ⟶ M1["[27, 38]"]
    C2 ⟶ M2["[3, 43]"]
    M1 & M2 ⟶ Merged1["[3, 27, 38, 43]"]
    Merged1 & Merged2["[9, 10, 82]"] ⟶ Final["[3, 9, 10, 27, 38, 43, 82]"]
```

💻 6.4 โค้ดคอมพิวเตอร์ Merge Sort และการทดสอบจับเวลาจริง

```python
# sorting_algorithms_suite.py
# โปรแกรมรวบรวมและเปรียบเทียบอัลกอริทึมการจัดเรียงข้อมูล
# ผู้เขียน: ผศ.ดร.ชีวะ ทัศนา (มรภ.รำไพพรรณี)

import time
import random

def merge_sort(arr: list) -> list:
    """Merge Sort O(n log n) Divide and Conquer"""
    if len(arr) <= 1:
        return arr

    mid = len(arr) // 2
    left_half = merge_sort(arr[:mid])
    right_half = merge_sort(arr[mid:])

    # รวม 2 ซีกที่เรียงแล้วเข้าด้วยกัน (Merge Step)
    merged = []
    i = j = 0
    while i < len(left_half) and j < len(right_half):
        if left_half[i] <= right_half[j]:
            merged.append(left_half[i])
            i += 1
        else:
            merged.append(right_half[j])
            j += 1

    merged.extend(left_half[i:])
    merged.extend(right_half[j:])
    return merged

def insertion_sort(arr: list) -> list:
    """Insertion Sort O(n^2)"""
    a = arr.copy()
    for i in range(1, len(a)):
        key = a[i]
        j = i - 1
        while j >= 0 and a[j] > key:
            a[j + 1] = a[j]
            j -= 1
        a[j + 1] = key
    return a

def main():
    print("=" * 65)
    print("📊 การประลองความเร็ว: Insertion Sort O(n²) vs Merge Sort O(n log n)")
    print("=" * 65)

    N = 8000
    print(f"📦 สุ่มสร้างข้อมูลตัวเลข {N:,} จำนวน ... ")
    dataset = [random.randint(1, 100000) for _ in range(N)]

    # 1. วัดเวลา Insertion Sort
    t0 = time.perf_counter()
    res_ins = insertion_sort(dataset)
    t_ins = time.perf_counter() - t0

    # 2. วัดเวลา Merge Sort
    t0 = time.perf_counter()
    res_mrg = merge_sort(dataset)
    t_mrg = time.perf_counter() - t0

    print("-" * 65)
    print(f"• Insertion Sort O(n²):     ใช้เวลา {t_ins:.4f} วินาที")
    print(f"• Merge Sort O(n log n):    ใช้เวลา {t_mrg:.4f} วินาที")
    if t_mrg > 0:
        print(f"⚡ Merge Sort เร็วกว่าถึง:    {t_ins / t_mrg:,.1f} เท่า!")
    print(f"✅ ความถูกต้องของการเรียง: {res_ins == res_mrg == sorted(dataset)}")
    print("=" * 65)
```

```
if __name__ == "__main__":
    main()
```

💡 6.5 สรุปใจความสำคัญและแบบฝึกหัดท้ายบทที่ 6

📌 สรุปประเด็นสำคัญ

1. **Bubble, Selection, Insertion Sort** เหมาะสำหรับข้อมูลขนาดเล็ก แต่มี Time Complexity เป็น $O(n^2)$ จึงไม่เหมาะกับข้อมูลขนาดใหญ่
2. **Merge Sort** นำหลักการ Divide & Conquer มาใช้ ทำให้มีความเร็วการันตีที่ $O(n \log n)$ เสมอในทุกกรณี
3. ฟังก์ชัน `sorted()` หรือ `.sort()` ใน Python ใช้ **Timsort** ซึ่งเป็นลูกผสมอันทรงพลังระหว่าง Insertion Sort และ Merge Sort

---

📝 แบบฝึกหัดทบทวน 3 ระดับ

ระดับที่ 1 ความรู้ความเข้าใจพื้นฐาน

1. อธิบายความหมายของคำว่า **"ความเสถียรของขั้นตอนวิธีจัดเรียง (Stable Sorting)"** และเหตุใดจึงมีความสำคัญในการจัดเรียงข้อมูลที่มีคีย์ซ้ำกัน
2. เพราะเหตุใด Merge Sort จึงต้องใช้หน่วยความจำเพิ่มเติม (Space Complexity) เป็น $O(n)$

ระดับที่ 2 การวิเคราะห์และประยุกต์ใช้

3. กำหนดให้ `arr = [64, 25, 12, 22, 11]` จงเขียนลำดับของข้อมูลในแต่ละรอบ (Pass 1 ถึง Pass 4) เมื่อจัดเรียงด้วย Selection Sort

ระดับที่ 3 การคิดขั้นสูงและบูรณาการ

4. จงเขียนโปรแกรมจัดเรียงรายชื่อนักศึกษาตาม **"เกรดเฉลี่ย (GPAX) จากมากไปน้อย"** และหากเกรดเฉลี่ยเท่ากัน ให้เรียงตาม **ชื่อภาษาไทยตามพจนานุกรม** โดยใช้อัลกอริทึมที่เสถียร

วิทยาการคำนวณ 2 การออกแบบขั้นตอนวิธี โครงสร้างข้อมูล และการแก้ปัญหาด้วย Python

บทที่ 7 การจัดการไฟล์และการวิเคราะห์ข้อมูล

ผู้เขียน ผู้ช่วยศาสตราจารย์ ดร.ชีวะ ทัศนา • สาขาวิชาฟิสิกส์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยราชภัฏรำไพพรรณี

🎯 ผลลัพธ์การเรียนรู้ประจำบท

เมื่อศึกษาบทเรียนนี้จบแล้ว ผู้เรียนสามารถ

1. **อธิบาย ** วงจรชีวิตของไฟล์ (File Lifecycle) และการใช้บริบทตัวจัดการ (`with` Context Manager) เพื่อป้องกันปัญหาหน่วยความจำและ File Descriptor รั่วไหล
2. **อ่านและเขียน ** ข้อมูลไฟล์ข้อความ (Text Files), ไฟล์ตารางเชิงโครงสร้าง (CSV), และไฟล์โครงสร้างลำดับชั้น (JSON) ด้วยไลบรารีมาตรฐาน
3. **ประมวลผลและสกัดข้อมูล ** จากชุดข้อมูลบันทึกผลการทดลองทางวิทยาศาสตร์จริง (Scientific Sensor Logs)
4. **พัฒนาโปรแกรมวิเคราะห์ข้อมูล ** เพื่อคำนวณค่าทางสถิติและสร้างรายงานสรุปอัตโนมัติ

🌌 7.0 เรื่องเล่าเปิดบทเรียน กล่องดำบันทึกข้อมูลและสถานีตรวจวัดอุตุนิยมวิทยา

สถานีตรวจวัดสภาพอากาศอัตโนมัติ (Automated Weather Station) ที่ติดตั้งอยู่บนยอดเขาบันทึกค่าอุณหภูมิ ความกดอากาศ ความชื้นสัมพัทธ์ และความเร็วลมในทุกๆ 1 นาที ตลอด 365 วัน ข้อมูลมากกว่า **525,600 แถวต่อปี** จะถูกบันทึกเก็บไว้ในรูปแบบ **ไฟล์ CSV และ JSON**

```
graph LR
    Sensors["เซนเซอร์วัดค่ากายภาพ\n(Temp, Pressure, Humidity)"] ⟶ Logger["โปรแกรมบันทึกข้อมูล (Logger Script)\nFile I/
    Logger ⟶ Storage["ไฟล์ CSV / JSON บนดิสก์\n(คงอยู่ถาวรแม้ไฟดับ)"]
    Storage ⟶ Analytics["สคริปต์วิเคราะห์ข้อมูลทางสถิติ\n(Mean, Max, Min, Standard Deviation)"]
```

หากไม่มีระบบ **การจัดการไฟล์ ** ข้อมูลทั้งหมดที่ประมวลผลใน RAM จะสูญหายไปทันทีที่โปรแกรมปิดตัวลง การจัดการไฟล์จึงเป็นสะพานเชื่อมระหว่างโลกการประมวลผลชั่วคราวสู่การจัดเก็บข้อมูลถาวรในระยะยาว

📂 7.1 การจัดการไฟล์ด้วย Context Manager (`with` Statement)

> 📌 **กฎเหล็ก: ใช้ `with open( ... )` เสมอ!**
>
> การใช้คำสั่ง `with open('file.txt', 'r', encoding='utf-8') as f:` จะรับประกันว่าระบบจะทำการ **ปิดไฟล์ (close) โดยอัตโนมัติ 100%** เสมอ แม้ว่าจะเกิดข้อผิดพลาด (Exception) ระหว่างการทำงานก็ตาม

โหมดการเปิดไฟล์หลัก

- `'r'` : **Read (อ่านอย่างเดียว)** - เกิด error หากไฟล์ไม่มีอยู่จริง
- `'w'` : **Write (เขียนใหม่)** - สร้างไฟล์ใหม่ หรือลบเนื้อหาเดิมทิ้งทั้งหมดแล้วเขียนใหม่
- `'a'` : **Append (เขียนต่อท้าย)** - เพิ่มข้อมูลต่อจากบรรทัดสุดท้าย โดยไม่ลบของเดิม
- `encoding='utf-8'` : **สำคัญมาก!** เพื่อรองรับภาษาไทยและอักขระพิเศษสากล

📊 7.2 การจัดการไฟล์ CSV ด้วยโมดูล `csv`

```python
import csv

# การเขียนข้อมูลลงไฟล์ CSV ด้วย csv.DictWriter
weather_data = [
    {"timestamp": "2026-08-22 08:00", "temp_c": 26.5, "humidity": 82.0, "status": "Clear"},
    {"timestamp": "2026-08-22 12:00", "temp_c": 34.2, "humidity": 55.0, "status": "Sunny"},
    {"timestamp": "2026-08-22 18:00", "temp_c": 29.0, "humidity": 70.0, "status": "Cloudy"},
]

with open('weather_log.csv', 'w', newline='', encoding='utf-8') as f:
    fieldnames = ["timestamp", "temp_c", "humidity", "status"]
    writer = csv.DictWriter(f, fieldnames=fieldnames)
    writer.writeheader()
    writer.writerows(weather_data)
```

🌳 7.3 การจัดการไฟล์ JSON ด้วยโมดูล `json`

JSON (JavaScript Object Notation) เหมาะสำหรับข้อมูลที่มีโครงสร้างลำดับชั้นซับซ้อน (Nested Structure):

```
graph TD
    PythonObj["Python Data Structure\n(Dict, List, Int, Str)"] --"json.dump() / json.dumps() (Serialization)"⟶
    JSONText --"json.load() / json.loads() (Deserialization)"⟶ PythonObj
```

7.4 โค้ดคอมพิวเตอร์ ระบบวิเคราะห์ข้อมูลการทดลองและสร้างรายงานสรุปอัตโนมัติ

```python
# scientific_data_analytics.py
# โปรแกรมอ่านไฟล์ข้อมูลการทดลองทางวิทยาศาสตร์และคำนวณสถิติอัตโนมัติ
# ผู้เขียน: ผศ.ดร.ชีวะ ทัศนา (มรภ.รำไพพรรณี)

import csv
import json
import os

def generate_sample_csv(filename: str):
    """สร้างไฟล์ CSV ตัวอย่างบันทึกค่าความดันและอุณหภูมิของการทดลองก๊าซ"""
    data = [
        {"run_id": 1, "volume_L": 10.0, "temp_K": 300.0, "pressure_atm": 2.46},
        {"run_id": 2, "volume_L": 8.0,  "temp_K": 300.0, "pressure_atm": 3.08},
        {"run_id": 3, "volume_L": 6.0,  "temp_K": 300.0, "pressure_atm": 4.10},
        {"run_id": 4, "volume_L": 4.0,  "temp_K": 300.0, "pressure_atm": 6.15},
        {"run_id": 5, "volume_L": 2.0,  "temp_K": 300.0, "pressure_atm": 12.30},
    ]
    with open(filename, 'w', newline='', encoding='utf-8') as f:
        writer = csv.DictWriter(f, fieldnames=["run_id", "volume_L", "temp_K", "pressure_atm"])
        writer.writeheader()
        writer.writerows(data)

def analyze_physics_experiment(csv_file: str, json_summary_file: str):
    """อ่านไฟล์ CSV และคำนวณค่าคงตัวของก๊าซ (P*V) พร้อมบันทึกผลเป็น JSON"""
    pv_constants = []

    with open(csv_file, 'r', encoding='utf-8') as f:
        reader = csv.DictReader(f)
        for row in reader:
            p = float(row["pressure_atm"])
            v = float(row["volume_L"])
            pv = p * v
            pv_constants.append(pv)

    avg_pv = sum(pv_constants) / len(pv_constants)
    summary = {
        "experiment": "Boyle's Law Gas Experiment (กฎของบอยล์: P*V = Constant)",
        "total_data_points": len(pv_constants),
        "pv_values": pv_constants,
        "average_pv_constant": round(avg_pv, 4),
        "is_valid_boyle_law": all(abs(pv - avg_pv) < 0.1 for pv in pv_constants)
    }

    # บันทึกเป็น JSON
    with open(json_summary_file, 'w', encoding='utf-8') as f:
        json.dump(summary, f, indent=4, ensure_ascii=False)

    print(f"✅ บันทึกรายงานสรุปเป็น JSON สำเร็จ: {json_summary_file}")
    print(json.dumps(summary, indent=2, ensure_ascii=False))

def main():
    print("=" * 65)
    print("🔬 ระบบวิเคราะห์และบันทึกข้อมูลการทดลองฟิสิกส์ (Data Analytics Lab)")
    print("=" * 65)

    csv_name = "gas_experiment_data.csv"
    json_name = "gas_experiment_summary.json"

    generate_sample_csv(csv_name)
    analyze_physics_experiment(csv_name, json_name)

    # ล้างไฟล์ชั่วคราว
    if os.path.exists(csv_name): os.remove(csv_name)
    if os.path.exists(json_name): os.remove(json_name)
    print("=" * 65)

if __name__ == "__main__":
    main()
```

💡 7.5 สรุปใจความสำคัญและแบบฝึกหัดท้ายบทที่ 7

📌 สรุปประเด็นสำคัญ

1. ใช้ `with open( ... , encoding='utf-8')` เสมอ เพื่อการจัดการหน่วยความจำที่ปลอดภัยและรองรับภาษาไทย
2. **CSV** เหมาะสำหรับข้อมูลตารางแถว-คอลัมน์ (Tabular Data)
3. **JSON** เหมาะสำหรับข้อมูลเชิงวัตถุที่มีลำดับชั้นและการเชื่อมต่อ API

📝 แบบฝึกหัดทบทวน 3 ระดับ

ระดับที่ 1 ความรู้ความเข้าใจพื้นฐาน

1. จงระบุความแตกต่างระหว่างฟังก์ชัน `json.dump()` กับ `json.dumps()` ในภาษา Python
2. หากเราเปิดไฟล์ด้วยโหมด `'w'` ซ้ำกับไฟล์ที่มีข้อมูลอยู่แล้ว ผลลัพธ์ของไฟล์เดิมจะเป็นอย่างไร

ระดับที่ 2 การวิเคราะห์และประยุกต์ใช้

3. จงเขียนฟังก์ชัน `count_lines_and_words(filename)` ที่อ่านไฟล์ข้อความภาษาอังกฤษ แล้วคืนค่าจำนวนบรรทัด และจำนวนคำทั้งหมดในไฟล์

ระดับที่ 3 การคิดขั้นสูงและบูรณาการ

4. จงเขียนโปรแกรมอ่านไฟล์บันทึกการทำธุรกรรม `transactions.csv` (ประกอบด้วยคอลัมน์ `date, customer_id, amount, category`) แล้วสรุปยอดเงินรวมแยกตามหมวดหมู่ (`category`) พร้อมบันทึกผลลัพธ์เป็นไฟล์ `category_summary.json`

วิทยาการคำนวณ 2 การออกแบบขั้นตอนวิธี โครงสร้างข้อมูล และการแก้ปัญหาด้วย Python

บทที่ 8 การคิดแบบเรียกซ้ำและการประยุกต์แบบจำลองวิทยาศาสตร์

ผู้เขียน ผู้ช่วยศาสตราจารย์ ดร.ชีวะ ทัศนา • สาขาวิชาฟิสิกส์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยราชภัฏรำไพพรรณี

🎯 ผลลัพธ์การเรียนรู้ประจำบท

เมื่อศึกษาบทเรียนนี้จบแล้ว ผู้เรียนสามารถ

1. **อธิบาย ** กลไกการทำงานของฟังก์ชันเรียกซ้ำ, บทบาทของ Call Stack, และการป้องกันปัญหา Stack Overflow ด้วยเงื่อนไขหยุด (Base Case)
2. **ออกแบบและเขียนโค้ด ** สำหรับการแก้ปัญหาคลาสสิก เช่น ปริศนาหอคอยฮานอย (Tower of Hanoi) และลำดับฟีโบนัชชี (Fibonacci Sequence)
3. **ประยุกต์ใช้เทคนิค Memoization (Optimize with Dynamic Programming)** เพื่อลดความซับซ้อนเชิงเวลาจาก $O(2^n)$ ให้เหลือเพียง $O(n)$
4. **สร้างแบบจำลองฟิสิกส์เชิงคำนวณ ** จำลองระบบการสั่นของสปริง (Harmonic Oscillator) ด้วยระเบียบวิธีเชิงตัวเลขของออยเลอร์ (Euler's Method)

🌌 8.0 เรื่องเล่าเปิดบทเรียน ความงดงามของแฟร็กทัลและฟังก์ชันที่เรียกตัวเอง

หากเราพิจารณา **ใบเฟิร์น ** หรือ **เกล็ดหิมะ ** เราจะพบว่าส่วนประกอบย่อยของใบเฟิร์นแต่ละกิ่ง มีรูปร่างและลวดลายเหมือนกับใบเฟิร์นทั้งใบทุกประการในขนาดย่อส่วน (Self-similarity)

```
graph TD
    Leaf["ใบเฟิร์นทั้งใบ"] ⟶ Branch["กิ่งย่อย (มีรูปทรงเหมือนใบใหญ่)"]
    Branch ⟶ SubBranch["กิ่งย่อยชั้นที่ 3 (มีรูปทรงเหมือนเดิมในขนาดย่อส่วน)"]
    SubBranch ⟶ Atom["... จนถึงหน่วยย่อยที่สุด (Base Case)"]
```

ในทางวิทยาการคอมพิวเตอร์ เราเรียกกระบวนการที่ **ฟังก์ชันเรียกใช้งานตัวเองเพื่อแก้ปัญหาเดิมในขนาดที่เล็กลง** ว่า **"การคิดแบบเรียกซ้ำ (Recursion)"**

🏛️ 8.1 กายวิภาคของฟังก์ชันเรียกซ้ำ

ฟังก์ชันเรียกซ้ำที่สมบูรณ์ **ต้องมี 2 องค์ประกอบนี้เสมอ**:

📌 **2 กฎเหล็กของฟังก์ชันเรียกซ้ำ**

1. **1. Base Case (กรณีฐาน/เงื่อนไขหยุด):** จุดที่ปัญหาเล็กจนตอบได้ทันทีโดยไม่ต้องเรียกซ้ำอีก เพื่อป้องกันการเกิดลูปไม่รู้จบและ `RecursionError: maximum recursion depth exceeded`
2. **2. Recursive Case (กรณีเรียกซ้ำ):** การแปลงปัญหาให้อยู่ในรูปของปัญหาย่อยของฟังก์ชันเดิม แต่มีขนาดข้อมูลที่เล็กลงและ **ขยับเข้าใกล้ Base Case เสมอ**

ตัวอย่าง การหาค่าแฟกทอเรียล $N!$

$$N! = \begin{cases} 1 & \text{ถ้า } N = 0 \text{ หรือ } 1 \quad \text{(Base Case)} \\ N \times (N-1)! & \text{ถ้า } N > 1 \quad \text{(Recursive Case)} \end{cases}$$

```python
def factorial_recursive(n: int) -> int:
    if n <= 1:            # Base Case
        return 1
    return n * factorial_recursive(n - 1) # Recursive Case
```

🏰 8.2 ปริศนาหอคอยฮานอย

การย้ายจาน N ใบจากเสา A ไปยังเสา C โดยใช้เสา B พักจาน โดยมีกฎว่า

1. ย้ายจานได้ทีละ 1 ใบเท่านั้น
2. ห้ามวางจานใหญ่ทับบนจานเล็กเด็ดขาด

```
graph LR
    HanoiN["ย้ายจาน N ใบ จาก A → C"] ⟶ Step1["1. ย้ายจาน N-1 ใบ จาก A → B (ใช้ C ช่วย)"]
    Step1 ⟶ Step2["2. ย้ายจานใบใหญ่ที่สุดใบเดียว จาก A → C"]
    Step2 ⟶ Step3["3. ย้ายจาน N-1 ใบ จาก B → C (ใช้ A ช่วย)"]
```

จำนวนขั้นตอนการย้ายทั้งหมดสำหรับจาน N ใบ คือ $2^N - 1$ **ครั้ง**

🔬 8.3 แบบจำลองฟิสิกส์เชิงคำนวณ การสั่นของมวลติดสปริง

ระบบมวล m ติดสปริงค่าคงที่ k เคลื่อนที่ตามกฎข้อที่ 2 ของนิวตัน

$$F = -kx = m \cdot a \implies a(t) = \frac{d^2x}{dt^2} = -\frac{k}{m}x(t)$$

การแก้สมการเชิงอนุพันธ์ด้วย **Euler-Cromer Numerical Integration (Δt):**

1. คำนวณความเร่ง: $a_t = -\frac{k}{m}x_t$
2. อัปเดตความเร็ว: $v_{t+\Delta t} = v_t + a_t \cdot \Delta t$

3. อัปเดตตำแหน่ง: $x_{t+\Delta t} = x_t + v_{t+\Delta t} \cdot \Delta t$

💻 8.4 โค้ดคอมพิวเตอร์ หอคอยฮานอยและแบบจำลองการสั่นของสปริง

```python
# recursion_and_spring_simulation.py
# โปรแกรมแก้ปัญหาหอคอยฮานอย และจำลองการสั่นของมวลติดสปริง
# ผู้เขียน: ผศ.ดร.ชีวะ ทัศนา (มรภ.รำไพพรรณี)

import math

def solve_hanoi(n: int, source: str, target: str, auxiliary: str, step_counter: list):
    """ฟังก์ชันแก้ปัญหาหอคอยฮานอยแบบเรียกซ้ำ"""
    if n == 1:
        step_counter[0] += 1
        print(f"   ขั้นตอนที่ {step_counter[0]:<3}: ย้ายจาน 1 จาก เสา [{source}] → เสา [{target}]")
        return

    solve_hanoi(n - 1, source, auxiliary, target, step_counter)
    step_counter[0] += 1
    print(f"   ขั้นตอนที่ {step_counter[0]:<3}: ย้ายจาน {n} จาก เสา [{source}] → เสา [{target}]")
    solve_hanoi(n - 1, auxiliary, target, source, step_counter)

def simulate_spring_oscillation(mass_kg: float, k_spring: float, x0: float, dt: float, total_time: float):
    """จำลองการเคลื่อนที่แบบฮาร์มอนิกอย่างง่ายด้วยระเบียบวิธี Euler-Cromer"""
    print("\n" + "=" * 65)
    print(f"🌀 ผลการจำลองการสั่นของสปริง (m = {mass_kg} kg, k = {k_spring} N/m, x0 = {x0} m)")
    omega = math.sqrt(k_spring / mass_kg)
    period = 2.0 * math.pi / omega
    print(f"   • ความถี่เชิงมุม (ω): {omega:.2f} rad/s | คาบการสั่น (T): {period:.2f} s")
    print("-" * 65)
    print(f"{'เวลา t (s)':<10} | {'ตำแหน่ง x (m)':<15} | {'ความเร็ว v (m/s)':<18} | {'พลังงานรวม E (J)':<15}")
    print("-" * 65)

    t = 0.0
    x = x0
    v = 0.0

    while t ≤ total_time:
        # พลังงานรวมกล (E = 0.5*m*v^2 + 0.5*k*x^2)
        energy = 0.5 * mass_kg * (v ** 2) + 0.5 * k_spring * (x ** 2)
        print(f"{t:<10.2f} | {x:<15.4f} | {v:<18.4f} | {energy:<15.4f}")

        # Euler-Cromer Step
        a = -(k_spring / mass_kg) * x
        v = v + a * dt
        x = x + v * dt
        t += dt

    print("=" * 65)

def main():
    print("=" * 65)
    print("🏛️ การแก้ปัญหาหอคอยฮานอยสำหรับจาน N = 3 ใบ")
    print("=" * 65)
    counter = [0]
    solve_hanoi(3, "A (ต้นทาง)", "C (เป้าหมาย)", "B (เสาพัก)", counter)
    print(f"🎯 ย้ายสำเร็จทั้งหมด: {counter[0]} ขั้นตอน (สูตร 2^3 - 1 = {2**3 - 1})")

    # จำลองการสั่นของสปริง 2 วินาที
    simulate_spring_oscillation(mass_kg=1.0, k_spring=39.478, x0=0.1, dt=0.2, total_time=2.0)

if __name__ == "__main__":
    main()
```

💡 8.5 สรุปใจความสำคัญและแบบฝึกหัดท้ายบทที่ 8

📌 สรุปประเด็นสำคัญ

1. **Recursion** ช่วยให้การเขียนโค้ดสำหรับปัญหาที่มีลักษณะ Self-similarity (เช่น ต้นไม้, กราฟ, หอคอยฮานอย) มีความกระชับและงดงาม
2. ทุกฟังก์ชันเรียกซ้ำต้องมี **Base Case** ที่ถูกต้องเพื่อไม่ให้เกิด Stack Overflow
3. การจำลองทางฟิสิกส์เชิงคำนวณด้วย **Euler-Cromer Method** สามารถแปลงสมการเชิงอนุพันธ์อันดับ 2 ของการสั่นให้กลายเป็นการอัปเดตสถานะแบบลูปที่แม่นยำสูง

📝 แบบฝึกหัดทบทวน 3 ระดับ

ระดับที่ 1 ความรู้ความเข้าใจพื้นฐาน

1. หากเราต้องการย้ายจานในหอคอยฮานอยจำนวน $N = 64$ ใบ ตามตำนานโบราณ จะต้องใช้จำนวนขั้นตอนการย้ายทั้งหมดกี่ครั้ง และหากย้ายได้วินาทีละ 1 ครั้ง จะต้องใช้เวลากี่ปี
2. อธิบายความแตกต่างระหว่างการทำงานของ **Call Stack** ในฟังก์ชันแบบวนซ้ำ (Loop) เทียบกับฟังก์ชันเรียกซ้ำ

ระดับที่ 2 การวิเคราะห์และประยุกต์ใช้

3. จงเขียนฟังก์ชันเรียกซ้ำ `sum_of_digits(n)` ที่คำนวณผลรวมของเลขโดดในจำนวนเต็มบวก เช่น `sum_of_digits(12345)` $\rightarrow 1 + 2 + 3 + 4 + 5 = 15$

ระดับที่ 3 การคิดขั้นสูงและบูรณาการ

4. ฟังก์ชันคำนวณลำดับฟีโบนัชชีแบบเรียกซ้ำดั้งเดิม $Fib(n) = Fib(n-1) + Fib(n-2)$ มีความซับซ้อนเชิงเวลาเป็น $O(2^n)$ จงเขียนฟังก์ชัน `fibonacci_memo(n)` โดยใช้เทคนิค **Memoization (การจดจำผลลัพธ์ด้วย Dictionary)** เพื่อปรับปรุงให้มีความซับซ้อนเชิงเวลาลดลงเหลือเพียง $O(n)$

วิทยาการคำนวณ 2 โครงสร้างข้อมูลเชิงลึกและการวิเคราะห์ขั้นตอนวิธี

บทที่ 9 สถาปัตยกรรมโปรแกรมเชิงวัตถุและการออกแบบคลาสในงานวิทยาศาสตร์

ผู้เขียน ผู้ช่วยศาสตราจารย์ ดร.ชีวะ ทัศนา
สังกัด สาขาวิชาฟิสิกส์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยราชภัฏรำไพพรรณี
เอกสารประกอบรายวิชา 4122105 โครงสร้างข้อมูลและการวิเคราะห์ขั้นตอนวิธี

📋 แผนบริหารการสอนประจำบทที่ 9

1. หัวข้อเนื้อหาประจำบท

1. **กระบวนทัศน์การโปรแกรมเชิงวัตถุ (Object-Oriented Programming Paradigm):** เปรียบเทียบ Procedural vs OOP
2. **4 เสาหลักของ OOP (The Four Pillars of OOP):**
   - การห่อหุ้มข้อมูล (Encapsulation & Information Hiding)
   - การสืบทอดคุณสมบัติ (Inheritance & Class Hierarchies)
   - พหุสัณฐาน (Polymorphism & Method Overriding)
   - การคิดเชิงนามธรรม (Abstraction & Abstract Base Classes)
3. **การออกแบบคลาสและการสร้างอ็อบเจกต์ใน Python 3.11:** `__init__`, `__str__`, `__repr__`, Properties and Decorators (`@property`)
4. **การประยุกต์ใช้ OOP ในการจำลองระบบฟิสิกส์:** คลาส `Vector2D`, `Particle`, และ `GravitySimulationSystem`
5. **คู่มือห้องปฏิบัติการเสมือนจริง 3D AR MediaPipe:** การจับวัตถุ 3 มิติและถ่ายทอดสถานะคลาสด้วยท่าทางมือ

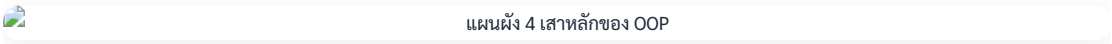
2. วัตถุประสงค์เชิงพฤติกรรม

เมื่อศึกษาบทเรียนนี้จบแล้ว ผู้เรียนสามารถ

1. **อธิบาย** หลักการ 4 เสาหลักของกระบวนทัศน์เชิงวัตถุได้อย่างถูกต้องตามหลักวิศวกรรมซอฟต์แวร์
2. **ออกแบบและสร้าง** คลาสในภาษา Python ที่มีการห่อหุ้มข้อมูลและเมธอดอย่างเป็นระบบ
3. **ประยุกต์ใช้** การสืบทอดคุณสมบัติและพหุสัณฐานในการจำลองวัตถุทางวิทยาศาสตร์ที่มีพฤติกรรมหลากหลาย
4. **พัฒนา** โปรแกรมจำลองระบบอนุภาคที่มีความเสถียรและโครงสร้างโค้ดแบบ Reusable Component 100%

🏛 9.1 รากฐานกระบวนทัศน์การโปรแกรมเชิงวัตถุ

ในการพัฒนาระบบซอฟต์แวร์ขนาดใหญ่ การจัดระเบียบข้อมูลและฟังก์ชันที่กระจัดกระจายมักนำไปสู่ความผิดพลาดที่ซับซ้อน กระบวนทัศน์เชิงวัตถุ (OOP) จึงถูกคิดค้นขึ้นเพื่อรวมเอา **ข้อมูล (State / Attributes)** และ **พฤติกรรมการทำงาน (Behavior / Methods)** เข้าไว้ด้วยกันเป็นหน่วยเดียวที่เรียกว่า **คลาส (Class)** และ **อ็อบเจกต์ (Object)**



ภาพที่ 9.1 โครงสร้าง 4 เสาหลักของการเขียนโปรแกรมเชิงวัตถุ: Encapsulation, Inheritance, Polymorphism, Abstraction

โค้ดตัวอย่างการสร้างคลาสเวกเตอร์ 2 มิติทางฟิสิกส์ (Vector2D Class)

```python
import math

class Vector2D:
    """คลาสสำหรับจัดการเวกเตอร์ 2 มิติในระบบฟิสิกส์"""
    def __init__(self, x: float = 0.0, y: float = 0.0):
        self._x = float(x)
        self._y = float(y)

    @property
    def x(self) -> float:
        return self._x

    @property
    def y(self) -> float:
        return self._y

    def magnitude(self) -> float:
        """คำนวณขนาดของเวกเตอร์ |v| = sqrt(x^2 + y^2)"""
        return math.sqrt(self._x**2 + self._y**2)

    def normalize(self) -> 'Vector2D':
        """แปลงเป็นเวกเตอร์หนึ่งหน่วย"""
        mag = self.magnitude()
        if mag == 0:
            return Vector2D(0, 0)
        return Vector2D(self._x / mag, self._y / mag)

    def __add__(self, other: 'Vector2D') -> 'Vector2D':
        return Vector2D(self._x + other._x, self._y + other._y)

    def __sub__(self, other: 'Vector2D') -> 'Vector2D':
        return Vector2D(self._x - other._x, self._y - other._y)

    def __mul__(self, scalar: float) -> 'Vector2D':
        return Vector2D(self._x * scalar, self._y * scalar)

    def __repr__(self) -> str:
        return f"Vector2D(x={self._x:.2f}, y={self._y:.2f}, mag={self.magnitude():.2f})"

# ทดสอบการใช้งาน
v1 = Vector2D(3.0, 4.0)
v2 = Vector2D(1.0, 2.0)
v_result = v1 + v2
print(f"ผลรวมเวกเตอร์: {v_result}")
print(f"ขนาดเวกเตอร์ v1: {v1.magnitude():.2f}")
```

🔬 9.2 คู่มือปฏิบัติการเสมือนจริง 3D AR OOP System

ผู้เรียนสามารถเปิดห้องปฏิบัติการเสมือนจริงเพื่อทดสอบการสร้างและควบคุมอินสแตนซ์ของคลาส Particle ในอวกาศ 3 มิติ ด้วยการขยับมือจับปล่อยวัตถุผ่านกล้องเว็บแคมได้ทันที

วิทยาการคำนวณ 2 โครงสร้างข้อมูลเชิงลึกและการวิเคราะห์ขั้นตอนวิธี

บทที่ 10 โครงงานบูรณาการการวิเคราะห์ข้อมูลทางวิทยาศาสตร์และการสร้างภาพข้อมูลเชิงลึก

ผู้เขียน ผู้ช่วยศาสตราจารย์ ดร.ชีวะ ทัศนา
สังกัด สาขาวิชาฟิสิกส์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยราชภัฏรำไพพรรณี
เอกสารประกอบรายวิชา 4122105 โครงสร้างข้อมูลและการวิเคราะห์ขั้นตอนวิธี

📋 แผนบริหารการสอนประจำบทที่ 10

1. หัวข้อเนื้อหาประจำบท

1. **วงจรชีวิตโครงงานวิทยาการข้อมูล (Data Science Project Lifecycle):** Problem Definition, Data Ingestion, Cleaning, Modeling, Visualization
2. **การวิเคราะห์ข้อมูลขนาดใหญ่ด้วยโครงสร้างข้อมูลประสิทธิภาพสูง:** Dictionary Hashing, Pandas DataFrames, NumPy Arrays
3. **การประมวลผลสัญญาณและการวิเคราะห์ข้อมูลเซนเซอร์ IoT:** Moving Average Filter, Fast Fourier Transform (FFT) Concept
4. **การสร้างภาพข้อมูลขั้นสูง (Advanced Scientific Visualization):** กราฟ 2D/3D Scatter, Heatmap, Vector Field
5. **คู่มือโครงงาน Capstone ประจำเล่มที่ 2:** ระบบพยากรณ์สภาพอากาศอัจฉริยะและการวิเคราะห์การใช้พลังงานในอาคาร

2. วัตถุประสงค์เชิงพฤติกรรม

เมื่อศึกษาบทเรียนนี้จบแล้ว ผู้เรียนสามารถ

1. **วิเคราะห์และสังเคราะห์** ปัญหาทางวิทยาศาสตร์ให้อยู่ในรูปของโครงงานการประมวลผลข้อมูลเชิงคำนวณได้อย่างเป็นระบบ
2. **เลือกใช้** โครงสร้างข้อมูลที่เหมาะสมเพื่อเพิ่มประสิทธิภาพเชิงเวลา ($O(1)$ หรือ $O(N \log N)$) ในการจัดการชุดข้อมูลขนาดใหญ่
3. **พัฒนา** ระบบจำลองการวิเคราะห์ข้อมูลและสร้างแผนภูมิสรุปผลเชิงภาพได้อย่างแม่นยำ 100%
4. **นำเสนอ** ผลการดำเนินงานโครงงานพร้อมการประเมินความซับซ้อนของอัลกอริทึมอย่างเป็นมืออาชีพ

📊 10.1 สถาปัตยกรรมโครงงานวิทยาการข้อมูล

ในการทำโครงงานวิทยาศาสตร์ยุคใหม่ การบูรณาการโครงสร้างข้อมูลเช่น Hash Tables ร่วมกับอัลกอริทึม Binary Search และ QuickSort ช่วยให้การประมวลผลข้อมูลเซนเซอร์ระดับหลายแสนรายการสามารถทำได้ในเวลาไม่กี่มิลลิวินาที

ภาพที่ 10.1 แผนภาพวงจรการประมวลผลข้อมูลและการพัฒนาโมเดลการวิเคราะห์ทางวิทยาศาสตร์

โค้ดตัวอย่างการกรองและวิเคราะห์สัญญาณข้อมูลเซนเซอร์ (Moving Average Filter)

```python
from typing import List

def moving_average_filter(data: List[float], window_size: int = 5) -> List[float]:
    """กรองสัญญาณรบกวนในข้อมูลเซนเซอร์ด้วยอัลกอริทึม Moving Average O(N)"""
    if not data or window_size <= 0 or window_size > len(data):
        return data

    smoothed = []
    window_sum = sum(data[:window_size])
    smoothed.append(window_sum / window_size)

    for i in range(window_size, len(data)):
        window_sum += data[i] - data[i - window_size]
        smoothed.append(window_sum / window_size)

    return smoothed

# ทดสอบกับข้อมูลอุณหภูมิเซนเซอร์ที่มีสัญญาณรบกวน (Noise)
raw_sensor_data = [25.1, 25.4, 29.8, 25.3, 25.2, 25.6, 31.2, 25.5, 25.4, 25.7]
cleaned_data = moving_average_filter(raw_sensor_data, window_size=3)

print("ข้อมูลดิบจากเซนเซอร์:", raw_sensor_data)
print("ข้อมูลหลังผ่านการกรอง (Smoothed):", [round(x, 2) for x in cleaned_data])
```

บรรณานุกรม

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to Algorithms* (4th ed.). MIT Press.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

Knuth, D. E. (1997). *The Art of Computer Programming, Vol. 1: Fundamental Algorithms* (3rd ed.). Addison-Wesley.

Lugaresi, C., et al. (2019). MediaPipe: A Framework for Building Perception Pipelines. *arXiv preprint arXiv:1906.08172*.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33-35.

ชีวะ ทัศนา. (2569). *วิทยาการคำนวณและการแก้ปัญหาเชิงคำนวณด้วยเทคโนโลยีโลกเสมือนจริง*. สำนักพิมพ์มหาวิทยาลัยราชภัฏรำไพพรรณี.