

หลักสูตร

นักวิทยาศาสตร์ข้อมูล Data Scientist

ภายใต้โครงการความเป็นเลิศด้านวิทยาศาสตร์ข้อมูล: ปลดล็อกศักยภาพ สร้างอาชีพ



Efficient Programming for Data Scientists

ខ្សោយកើតឡើងនូវការងារ

ការខ្សោយកើតឡើងនូវការងារដែលសម្រេចបានបាតិវិជ្ជាសាស្ត្រប៉ូមូល - ហេតុកិចនិងការងារការងារដែលអាចផ្តល់បន្ថែមទៅការងារបាន។



ហាមីរីំងប៉ូលសំគាល់?

បញ្ជាក់ថាអ្នកបានប្រើប្រាស់ការងារប៉ូលខ្លាងខ្លួន។

"គ្រប់គ្រងការងារ" ≠ "គ្រប់គ្រង"

ការងារបានប្រើប្រាស់ការងារប៉ូលបានប្រើប្រាស់ការងារប៉ូល។ ពេលវេលាយករាយការងារប៉ូលត្រូវបានប្រើប្រាស់ការងារប៉ូល។

វេលាកែវតាមការងារប៉ូល។

ការងារប៉ូលត្រូវបានប្រើប្រាស់ការងារប៉ូល។ ពេលវេលាយករាយការងារប៉ូលត្រូវបានប្រើប្រាស់ការងារប៉ូល។

Performance is a feature - ការងារប៉ូលត្រូវបានប្រើប្រាស់ការងារប៉ូល។ ពេលវេលាយករាយការងារប៉ូលត្រូវបានប្រើប្រាស់ការងារប៉ូល។



เป้าหมายและกำหนดการ

สิ่งที่จะได้เรียนรู้ในวันนี้

01

เข้าใจถูกต้อง (Why)

ทำความเข้าใจหลักการพื้นฐานของการเขียนโค้ดที่มีประสิทธิภาพ Big O Notation และการเลือกใช้ Data Structure ที่เหมาะสม

03

นำไปใช้ได้จริง (Action)

ได้ Checklist และแนวทางต่อ�อดที่สามารถนำไปประยุกต์ใช้ในงานจริงได้ทันที

- ❑ **กำหนดการวันนี้:** Part 1: ถูกต้องเบื้องหลังความเร็ว → Part 2: Workshop ปรับโค้ดให้เร็วขึ้น 100 เท่า! → Part 3: สรุปและถาม-ตอบ

Learning Path





ประสิทธิภาพคืออะไร?

มิติของประสิทธิภาพที่ต้องรู้

Time Complexity

ความเร็วในการทำงาน

- โค้ดใช้เวลาทำงานนานแค่ไหน?
- เมื่อข้อมูลเพิ่มขึ้น เวลาเพิ่มขึ้นอย่างไร?
- เป็นสิ่งที่ผู้ใช้สัมผัสได้โดยตรง

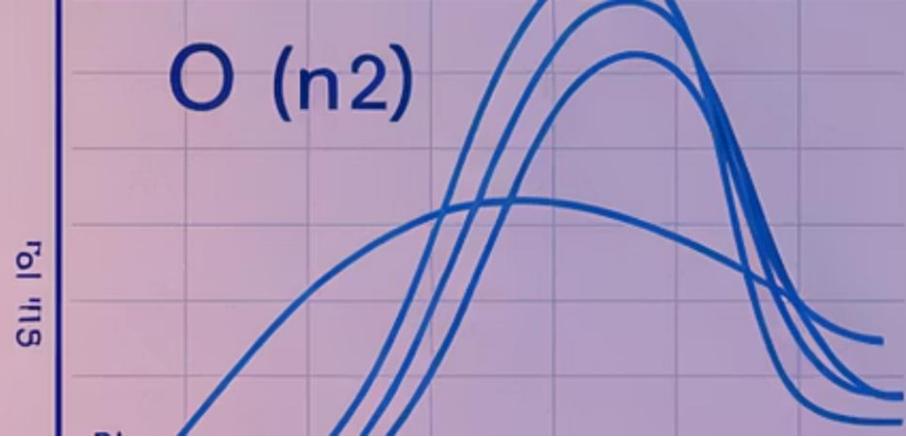
Space Complexity

ปริมาณหน่วยความจำที่ใช้

- โค้ดใช้ Memory เท่าไหร?
- การจัดการหน่วยความจำอย่างมีประสิทธิภาพ
- สำคัญเมื่อกำหนด Big Data

วันนี้เราจะเน้นที่

Time Complexity - การทำให้โค้ดทำงานเร็วขึ้นอย่างเร็วได้ชัด



รู้จักกับ Big O Notation

ภาษาສากลสำหรับวัด Performance

Big O ใช้อธิบาย "อัตราการเติบโต" ของเวลาที่ใช้ในการทำงาน เมื่อข้อมูล (n) เพิ่มขึ้น

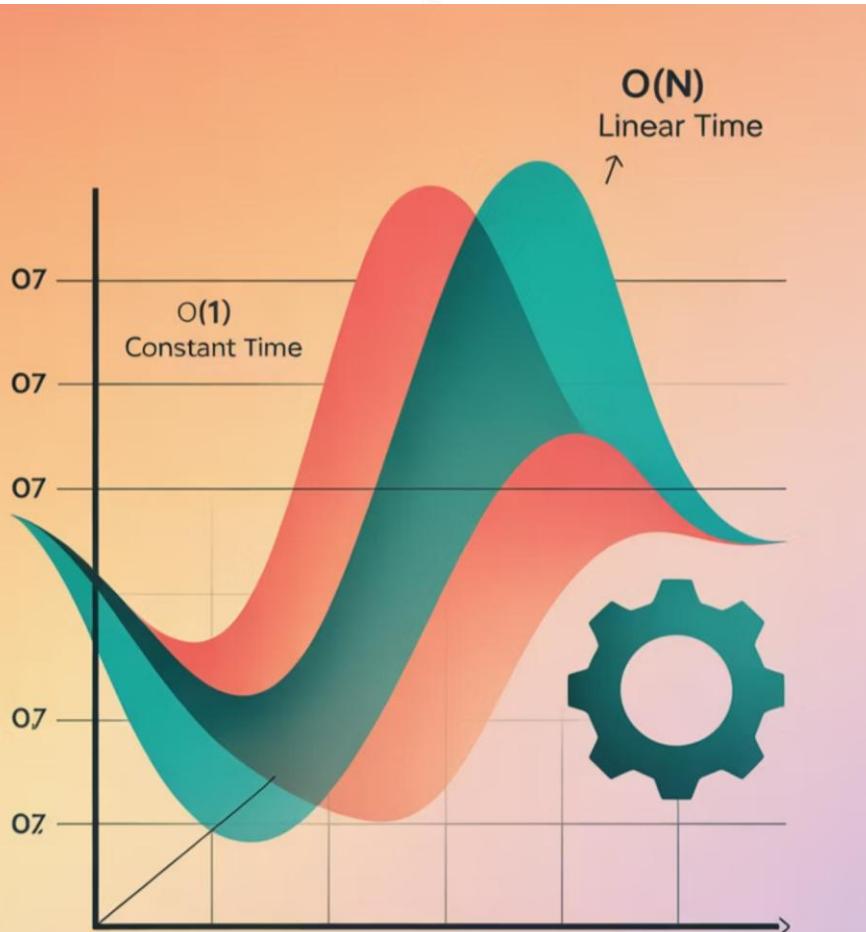
ไม่ใช่การจับเวลาจริง

Big O ไม่ได้บอกว่าโค้ดใช้เวลาเกินกว่า แต่เป็นการบอก "แนวโน้ม" ว่าเมื่อข้อมูลเพิ่มขึ้น 10 เท่า เวลาจะเพิ่มขึ้นอย่างไร

เป็นเครื่องมือประเมิน

ช่วยให้เราสามารถเปรียบเทียบ Algorithm ต่างๆ และคาดการณ์ได้ว่าโค้ดจะทำงานอย่างไรเมื่อข้อมูลโตขึ้น

Big O คือ "คำกำหนด" ไม่ใช่ "ผลลัพธ์" - แต่เป็นคำกำหนดที่แม่นยำมาก



Algorithm Efficiency

Big O កំពុបែរយ

$O(1)$ vs $O(n)$ - แบบเร็วคงที่ vs. เร็วตามจำนวน

O(1) - Constant Time

เร็วท่าเดิมเสมอ

ไม่ว่าข้อมูลจะมีเท่าไหร่ เวลาที่ใช้ยังคงเท่าเดิม

- `dict['key']` - การค้นหาใน Dictionary
 - `array[index]` - การเข้าถึงด้วย Index
 - `len(list)` - การหาความยาว
 - `for item in list` - การวน Loop
 - `item in list` - การค้นหาใน List
 - `sum(list)` - การรวมค่าทั้งหมด

เคล็ดลับ: พยายามใช้ O(1) operations ให้มากที่สุด โดยเฉพาะเมื่อทำงานกับข้อมูลขนาดใหญ่

Big O เขตอันตราย: $O(n^2)$

ตัวการของความช้าที่ต้องระวัง

O(n^2) - Quadratic Time

ចាន់បន្ទាន់របស់ខ្លួន!

ข้อมูลเพิ่มขึ้น 2 เท่า เวลาเพิ่มขึ้น 4 เท่า ข้อมูลเพิ่มขึ้น 10 เท่า เวลาเพิ่มขึ้น 100 เท่า

ຕັວອຍ່າງກີ່ພບບ້ອຍ:

```
# Loop ຂອນ Loop - O(n2)
for i in range(n):
    for j in range(n):
        # do something
```

1K

ข้อมูล 1,000 รายการ ໃຫ້ເວສາ 1 ວັນທີ

10K

ข้อมูล 10,000 รายการ ໃຫ້ເວລາ **100 ວິນາທີ!**

100K

ข้อมูล 100,000 รายการ
ใช้เวลา **10,000** วินาที!

Key Idea: គរតតុការឡើយ $O(n^2)$ ឲ្យមាតកថែស្ថិតប៉ូន្មានលីខណាតុយ

គេចរៀនប័ណ្ណការងារសំខាន់សំខាន់របស់អ្នក

List

អេមាត់កំបែ: ការរាយការការងារក្នុងក្រឡាយការងារ

អេមាត់កំបែ: ការការងារក្នុងក្រឡាយការងារ

- កំណត់តម្លៃការងារក្នុងក្រឡាយការងារ
- ត្រូវការងារក្នុងក្រឡាយការងារ

Set

អេមាត់កំបែ: ការការងារក្នុងក្រឡាយការងារ

អេមាត់កំបែ: ការការងារក្នុងក្រឡាយការងារ

- ត្រូវការងារក្នុងក្រឡាយការងារ
- ត្រូវការងារក្នុងក្រឡាយការងារ

Dictionary

អេមាត់កំបែ: ការការងារក្នុងក្រឡាយការងារ

អេមាត់កំបែ: ការការងារក្នុងក្រឡាយការងារ

- ការការងារក្នុងក្រឡាយការងារ
- ការការងារក្នុងក្រឡាយការងារ

គេចរៀនប័ណ្ណការងារសំខាន់សំខាន់របស់អ្នក!



Workshop - มาเริ่มกันเลย!

โจทย์: ยกกำลังสองตัวเลข 1,000,000 ตัว

เป้าหมายของ Workshop

เปรียบเทียบความเร็วระหว่างโค้ด 2 แบบ และถ่วงเวลาการเลือกใช้เทคนิคที่ถูกต้องสามารถส่งผลต่อประสิทธิภาพได้มากแค่ไหน

01

วิธีที่ 1: The Slow Way

ใช้ For Loop แบบดั้งเดิม - วิธีที่ตรงไปตรงมาที่นักเขียนโค้ดใหม่มักใช้

02

วิธีที่ 2: The Fast Way

ใช้ Vectorization ด้วย NumPy - เทคนิคขั้นสูงที่จะเปลี่ยนเกม

03

การวัดผลและเปรียบเทียบ

ใช้เครื่องมือ Profiling เพื่อวัดเวลาจري่และเปรียบเทียบผลลัพธ์

พร้อมที่จะเห็นผลลัพธ์ที่อาจทำให้คุณประหลาดใจแล้วหรือยัง?



ວິທີທີ່ 1: The Slow Way

ວິທີກໍຕຽນໄປຕຽນມາ (For Loop)

นี่คือโค้ดที่นักเขียนโปรแกรมใหม่มักเขียน - เข้าใจง่าย ตรงไปตรงมา แต่...



```
# The Slow Way - Using For Loop
import timedef
slow_square_numbers(numbers):
    result = []
    for num in numbers:
        result.append(num ** 2)
    return result

# สร้างข้อมูลทดสอบ
data = list(range(1_000_000))
# ทดสอบความเร็ว
start_time = time.time()
result = slow_square_numbers(da
end_time = time.time()
print(f"Time taken: {end_time -
```

ວິເຄຣະກໍ Big O
ໂຄດນີ້ມີ Big O = **O(n)** ເພຣະຕ້ອງວນທຸກຕັ້ງໃນຂ້ອມູລ 1 ຮອບ
ຈຸດອ່ອນ
Python Loop ກໍາງານຫ້າ ແລະຕ້ອງສຽງ List ໃໝ່ກີລະຕັ້ງ

ອຍ່າເດາ, ໃຫ້ວັດຜາ!

การวัดประสิทธิภาพด้วย %timeit

การเดาประสิทธิภาพโค้ดมักจะผิด! วิธีที่ถูกต้องคือการใช้เครื่องมือ Profiling เพื่อวัดผลจริง

In Jupyter Notebook

```
%timeit -n 5 -r  
3slow_square_numbers(data)
```

-n 5 = รับ 5 รอบต่อการทดสอบ -r 3 = ทำการทดสอบ 3 ครั้ง

Python Script

```
import timeit
result = timeit.timeit('slow_square_numbers(data)',  
                      globals=globals(), number=5)
```

ผลลัพธ์ (ตัวอย่าง)

150 ms ± 5 ms per loop

គឺជាការប្រកបដីលើករាជការនៃប្រទេសកម្ពុជា ដែលបានចាប់ឡើងពីថ្ងៃទី 150 មិថុនា ឆ្នាំ 1993 ដោយសារតម្លៃ 1 លានពាន់

- การวัดผลที่ถูกต้องต้องทำหลายรอบ เพราะ CPU อาจมีงานอื่นกำ(SIGINT)อยู่ด้วย

วิธีที่ 2: The Fast Way

เทคนิคเปลี่ยนเกมด้วย NumPy

ແນວຄົດ: Vectorization

คือการสั่งงานกับข้อมูล "กึ่งชุด" ในคำสั่งเดียว แทนการวนทีละตัว



ทำไม NumPy ถึงเร็ว?

- เขียนด้วยภาษา C ที่เร็วกว่า Python
 - ใช้ SIMD (Single Instruction, Multiple Data)
 - จัดการหน่วยความจำอย่างมีประสิทธิภาพ
 - หลีกเลี่ยง Python Loop ที่ซ้ำ

NumPy ไม่ได้เป็นแค่ Library ธรรมดา แต่เป็นการปฏิวัติวิธีการประมวลผลข้อมูลในภาษา Python

ផលការប្រើប្រាស់ប៉ូល

Side-by-Side Comparison

Method	Time	Speedup
For Loop (Traditional)	~150 ms	1x (baseline)
NumPy Vectorization	~1.5 ms	~100x Faster!

100x

ឱ្យឱ្យ

ការឱ្យឱ្យកើតឡើងកើតឡើង 100 ពេលវេលា!

2.5

បាតី

វេលាកំពង់រាយដែលត្រួតពិនិត្យ 1 ក្រឹង

42

ចំពោះ

វេលាកំពង់រាយដែលត្រួតពិនិត្យ 1,000 ក្រឹង

ការឱ្យឱ្យកើតឡើងមិនត្រូវបានធ្វើឡើង សារណ៍បានប្រើប្រាស់កើតឡើង 2.5 បាតី ឱ្យការងារបានប្រើប្រាស់កើតឡើង 1.5 បាតី

ប៉ូលជាប្រព័ន្ធឌីជីថល ដែលបានប្រើប្រាស់កើតឡើង 1.5 បាតី ឱ្យការងារបានប្រើប្រាស់កើតឡើង 1.5 បាតី

អត្ថបទ ណាកិវិយាណាគាស់ប៉ូល Data Scientist

ការងារក្នុងការគាំទ្ររបស់អ្នកជាប៉ូល: ប្រព័ន្ធឌីតាគិភាព សរាយអាជីវ

```
# The Fast Way - Using NumPy Vectorization
import numpy as np
import time

def fast_square_numbers(numbers):
    # ផែលជា NumPy array
    np_array = np.array(numbers)
    # យកការលែងសង្គមចុះនៅក្នុងការសំឡើង
    return np_array ** 2

# សរាយខ្លួនុយលទសោប៊បែន NumPy តឹងតែតាម
data = np.arange(1_000_000)
# ទទួលភារណ៍ថ្មី
start_time = time.time()
result = data ** 2
# បែនភ្លាមៗទៀត!
end_time = time.time()
print(f"Time taken: {end_time - start_time:.3f} seconds")
```

សំណង់ការណែនាំ

- មិនមែន For Loop ឡើ!
- គឺជាការសំណង់លើមករណ៍
- អាចធ្វើបានភ្លាមៗបាន
- ក្នុងការការណែនាំការងារក្នុងការគាំទ្រ

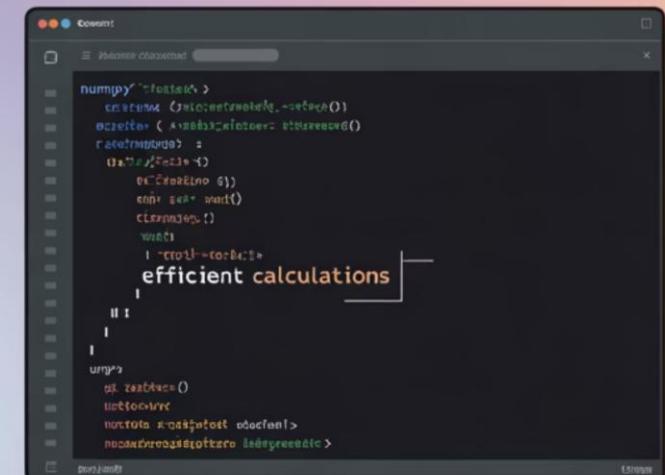
ធនធាន (តើវិញ)

1.5 ms ± 50 µs per loop

ឱ្យឱ្យមាត្រាខ្សែស្រួល ដើម្បីការងារក្នុងការគាំទ្រ!

គឺជាការងារក្នុងការគាំទ្រ

តឹងតែតាមការសំឡើង





สรุปจาก Workshop ทำใน Vectorization ดีงเร็วกว่า?



Python Loop

แต่ละครั้งที่วน Loop Python ต้อง:

- ตรวจสอบ Type ของข้อมูล
- เรียก Function ยกกำลังสอง
- สร้าง Object ใหม่
- จัดการ Memory

คำตอว: NumPy เรียกใช้โค้ดเบื้องหลังที่เขียนด้วยภาษา C ที่ทำงานได้เร็วกว่า Python Loop มาตรฐานมาก รวมถึงการใช้เทคนิค SIMD ที่ประมวลผลข้อมูลหลายตัวพร้อมกัน



NumPy Vectorization

ส่งงานทั้งหมดให้ C code ที่:

- รู้ Type ของข้อมูลล่วงหน้า
- ประมวลผลแบบ Batch
- ใช้ CPU instructions พิเศษ
- จัดการ Memory อย่างมีประสิทธิภาพ

นี่คือเหตุผลที่ Libraries อย่าง Pandas, Scikit-learn, และ TensorFlow ล้วนใช้ NumPy เป็นพื้นฐาน

หลักสูตร นักวิทยาศาสตร์ข้อมูล Data Scientist

ภายใต้โครงการความเป็นเลิศด้านวิทยาศาสตร์ข้อมูล: ปลดล็อกคุณภาพ สร้างอาชีพ

หลักการสำคัญที่ต้องจำ

1 วัดผล อย่าเดา

ใช้ Profiler เช่น `%timeit`, `cProfile` เพื่อหาจุดที่โค้ดช้าจริงๆ อย่าไปปรับจุดที่เดาเอาเอง

2 เลือกเครื่องมือให้ถูก

ใช้ Vectorization ด้วย NumPy/Pandas แทน Python loops เมื่อไหร่ก็ตามที่เป็นไปได้

3 เข้าใจกุญแจ

ใช้ Big O Notation เพื่อประเมินและเปรียบเทียบ Algorithm ก่อนเขียนโค้ด



เลือก Data Structure

Set/Dict สำหรับการค้นหา, List สำหรับการวน Loop ตามลำดับ



จัดการ Memory

หลีกเลี่ยงการสร้าง Object ใหม่ไม่จำเป็น ใช้ In-place operations เมื่อเป็นไปได้



ใช้ Libraries ที่เหมาะสม

NumPy, Pandas, Numba, Dask - เลือกใช้เหมาะสมกับลักษณะงาน

Checklist เพื่อโค้ดที่มีประสิทธิภาพ



ចំណាំ: "Premature Optimization"

ការប្រាប់ចុនកើតឡើងក្នុងកើតឡើង គឺជាការងារទិន្នន័យ

"Premature optimization is the root of all evil"-

Donald Knuth, នាកុវិកម្មសាស្ត្រគណនិភូវតែរដ្ឋីយំនឹង

01

ឱ្យឈ្មោះតួនាទី

កើតឡើងក្នុងកើតឡើងនិងផ្តាល់ Test cases មានការងារទិន្នន័យ

02

ឱ្យឈ្មោះងាយ

កើតឡើងក្នុងកើតឡើង maintain ងាយ និង debug ងាយ តាមការងារទិន្នន័យ

03

អាជីវកម្មសាស្ត្រ

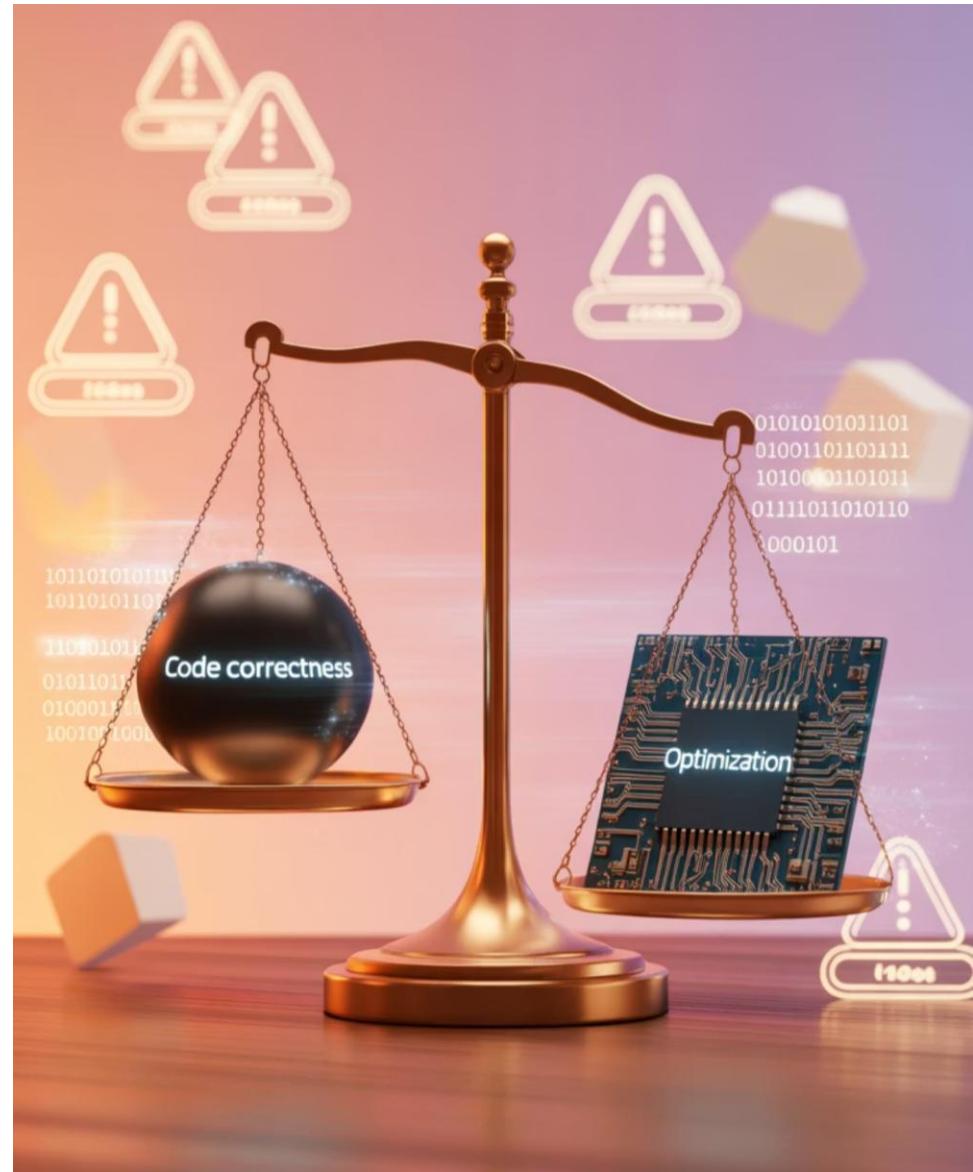
ឱ្យ Profiler អាជីវកម្មសាស្ត្រកើតឡើងក្នុងកើតឡើង និងការងារទិន្នន័យ

04

ប្រាប់ចុនយ៉ាងមិនមេ

ប្រាប់ទេរាជីតុលក្នុងកើតឡើងក្នុងកើតឡើង

- **គុណភាព:** គុណភាពកើតឡើងក្នុងកើតឡើងនិងកើតឡើងក្នុងកើតឡើង តាមការងារទិន្នន័យ



ឯកសារការងារគ្មានៗរបស់ខ្លួន



Memory Management

រឿងនៃការរចនាអនុវត្តន៍យករាយការងារ ដើម្បីបង្កើតរឹងរាយការងារ។



Just-In-Time (JIT) Compilation

ការងារ Numba library ដែលអាចរចនាបញ្ហានៃការងារ Python functions ជាអាជីវការ។



Parallel Computing

រឿងនៃ Dask, multiprocessing, និង concurrent.futures ដើម្បីបង្កើតរឹងរាយការងារប្រចាំបាន។

Resources បណ្ឌិត

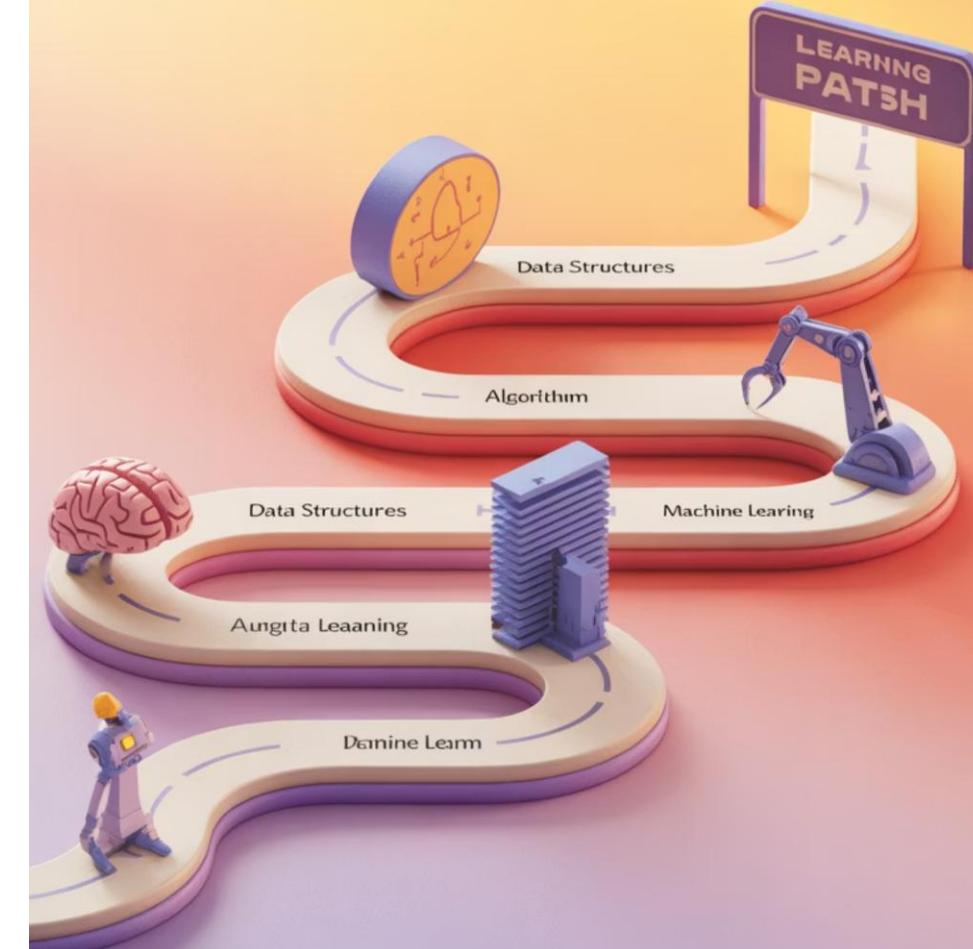
- High Performance Python (ឱសស៊ូ)
- NumPy User Guide
- Pandas Performance Tips
- Dask Tutorial

Tools បណ្ឌិត

- cProfile - Python profiler
- line_profiler - វិធានការងារ
- memory_profiler - វិធានប្រឈមប្រឈម
- py-spy - Real-time profiling

ការបង្កើតរឹងរាយការងារគ្មានៗរបស់ខ្លួន ត្រូវបានរចនាបញ្ហានៃការងារប្រចាំបាន និងការងារថ្មី ដើម្បីបង្កើតរឹងរាយការងារប្រចាំបាន។

ការងារគ្មានៗរបស់ខ្លួន





ถาม-ตอบ Q&A



ແບ່ງປັນປະສົບກາຣດີ

ມີໃຄຣເຄຍເຈອປ້າຍຫາໂຄດ້າ ຂໍ້ອມເທັນຄົດເຖິງ ອຢາກແບ່ງປັນກັນບ້າງໄໝນ?

អត្ថបទ នักវិទ្យាការសេចក្តីថ្លែង Data Scientist

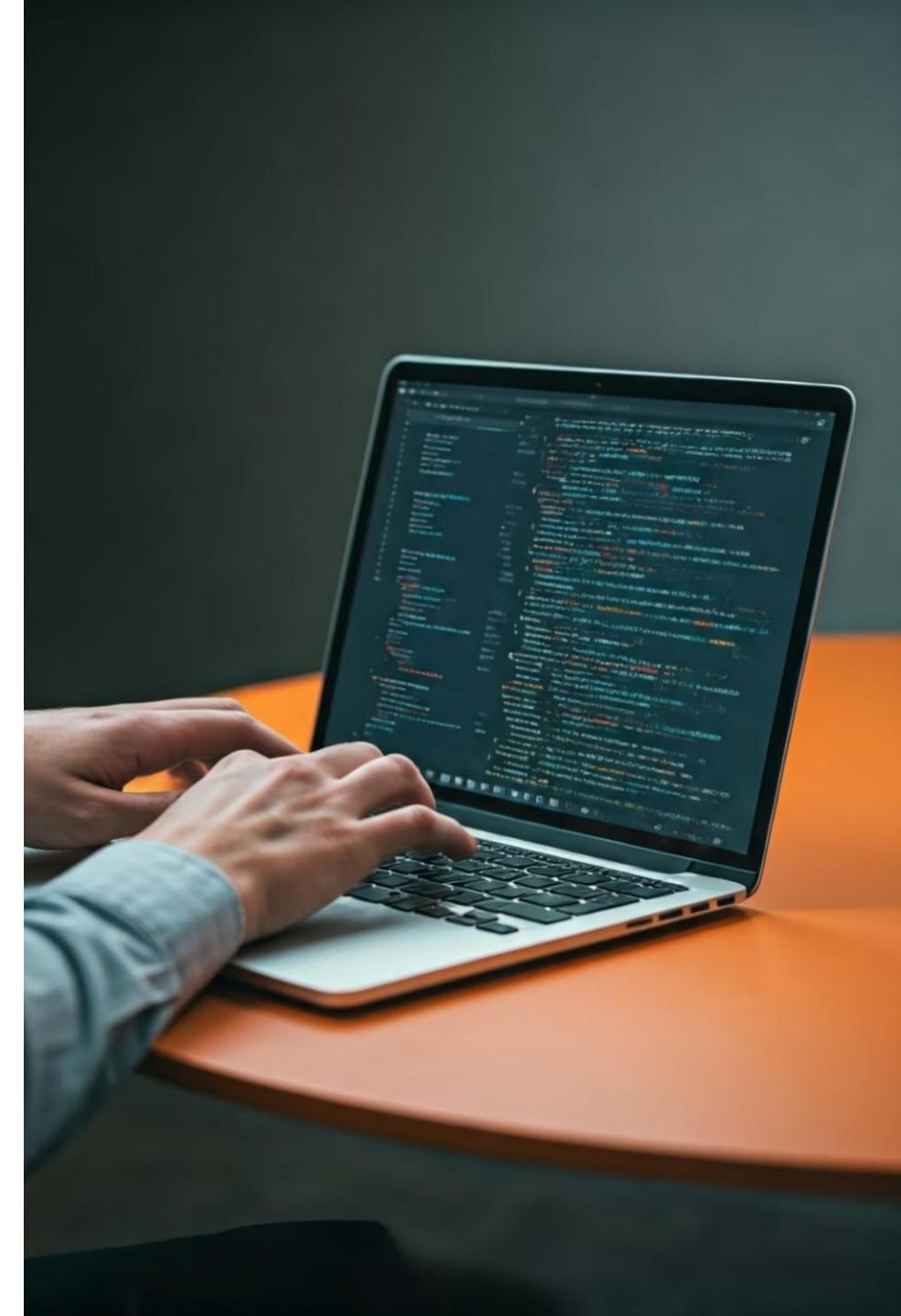
ภายใต้โครงการความเป็นเลิศด้านวิทยาศาสตร์ข้อมูล: ปลดล็อกคักษณภาพ สร้างอาชีพ

หวังว่าเทคโนโลยีและความรู้ที่แบ่งปันวันนี้
จะช่วยให้คุณเขียนโค้ดที่เร็วและมีประสิทธิภาพมากขึ้น

ສິ່ງສຳຄັນທີ່ໄດ້ເຮັດວຽກ

- Performance is a feature ที่สำคัญ
 - เลือกใช้เครื่องมือที่เหมาะสม
 - วัดผลอย่างมีระบบ อย่าเดา
 - Vectorization เป็นการเปลี่ยนแปลง

"The best time to optimize was yesterday.
The second best time is now."

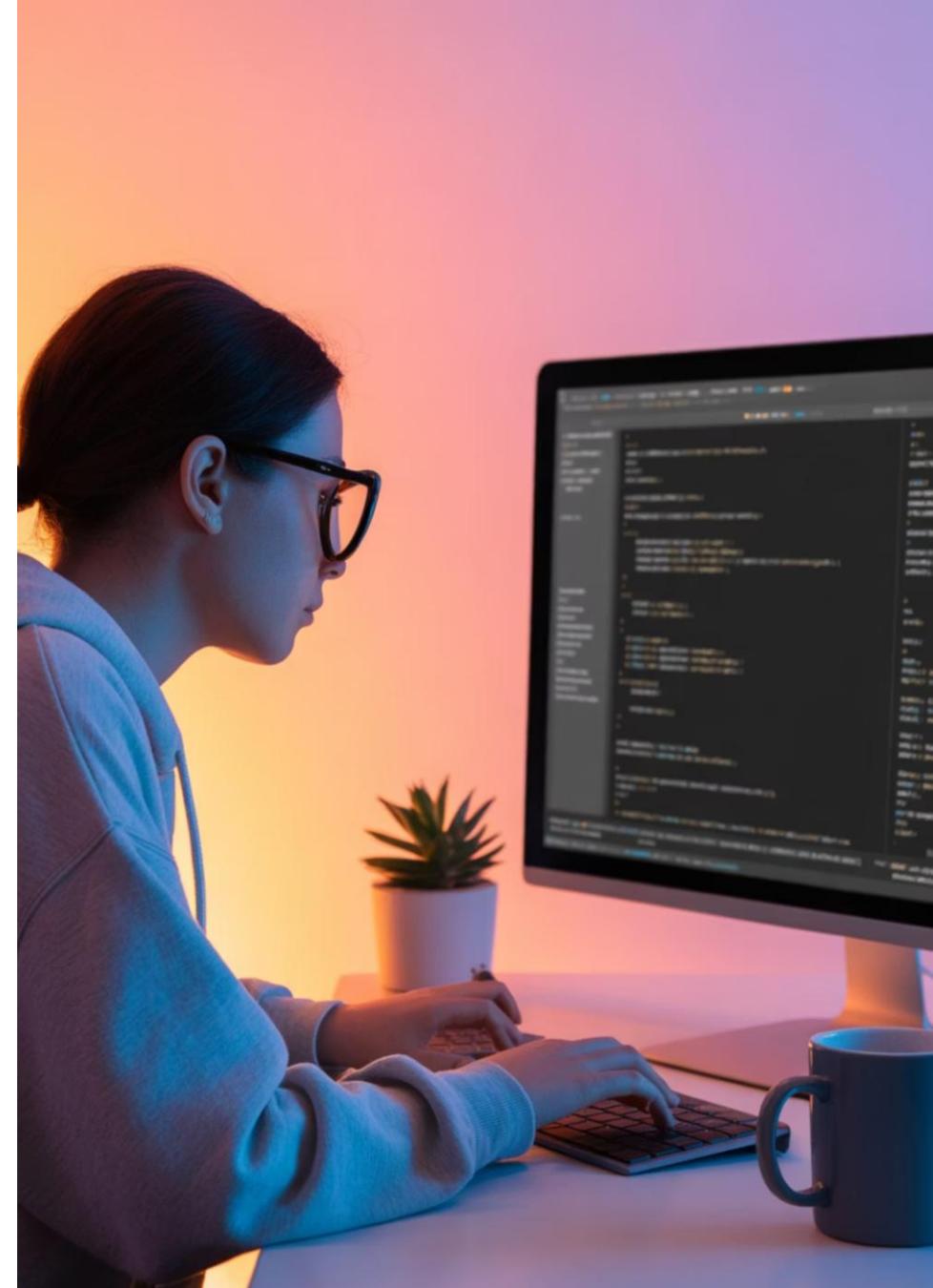


អាស៊ុត្រ បាកវិកយាតាសេតុលីមូល Data Scientist

ការណើតក្នុងការគ្រប់គ្រងការងារទិន្នន័យ និងការងារសេដ្ឋកិច្ច ដែលមានសំណង់ខ្ពស់

ការប្រើប្រាស់ បច្ចេកទិន្នន័យ

គ្រប់គ្រងការងារទិន្នន័យ និងការងារសេដ្ឋកិច្ច ដែលមានសំណង់ខ្ពស់



វត្ថុការសំគាល់សំខាន់របាយការប្រព័ន្ធទៅក្នុងការប្រើប្រាស់បច្ចុប្បន្ន

1 វិភាគផលកំណត់សំណួល (Profiling First!)

ឯកសារនេះផ្តល់ព័ត៌មានលម្អិតប្រព័ន្ធប្រចាំថ្ងៃ និងប្រចាំសប្តាហ៍ ដើម្បីរួចរាល់ពីការងារដែលត្រូវបានបង្កើតឡើង។

2 ប្រព័ន្ធទៅក្នុងការប្រើប្រាស់បច្ចុប្បន្ន

ការងារនេះផ្តល់ព័ត៌មានលម្អិតប្រព័ន្ធប្រចាំថ្ងៃ និងប្រចាំសប្តាហ៍ ដើម្បីរួចរាល់ពីការងារដែលត្រូវបានបង្កើតឡើង។

3 ិច្ចាគ្រឹះសំណួល (Using Libraries)

ិច្ចាគ្រឹះសំណួល និងការប្រើប្រាស់បច្ចុប្បន្ន នូវការប្រព័ន្ធប្រចាំថ្ងៃ និងប្រចាំសប្តាហ៍ ដើម្បីរួចរាល់ពីការងារដែលត្រូវបានបង្កើតឡើង។

4 ការប្រើប្រាស់បច្ចុប្បន្ន (Parallelism)

ការប្រើប្រាស់បច្ចុប្បន្ន និងការប្រើប្រាស់បច្ចុប្បន្ន នូវការប្រព័ន្ធប្រចាំថ្ងៃ និងប្រចាំសប្តាហ៍ ដើម្បីរួចរាល់ពីការងារដែលត្រូវបានបង្កើតឡើង។



🏆 กฏแห่งข้อแรก: วัดผลก่อนเลือก

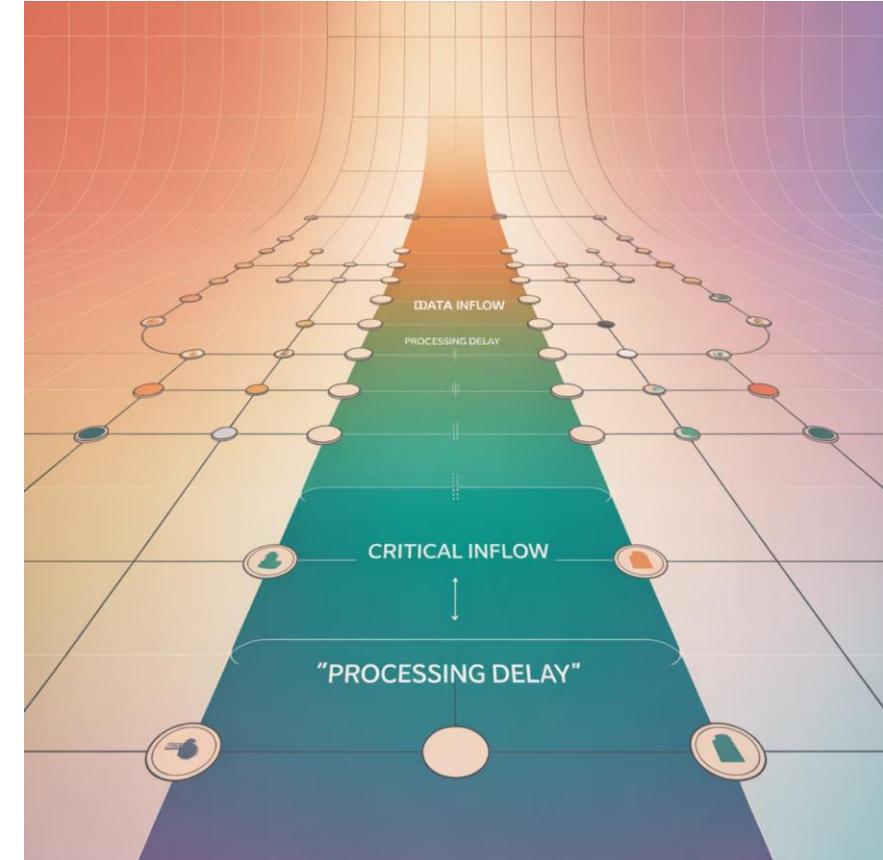
"Don't optimize without measuring" - การเดาผิดเรื่องจุดคอขวดเป็นข้อผิดพลาดที่พบบ่อยที่สุด

การปรับแต่งประสิทธิภาพที่ประสบความสำเร็จต้องเริ่มด้วยการวัดผลอย่างถูกต้อง การเดาใจว่าส่วนไหนของโปรแกรมทำงานช้ามากจะผิดพลาด ทำให้เสียเวลาไปปรับแต่งในจุดที่ไม่จำเป็น

Profiling គឺចือរណ៍ដី?

Profiling គឺជាពេទ្យលេខាមួលដែលបានប្រើប្រាស់ដើម្បីបង្ហាញការងាររបស់កុំពូកដែលមានព័ត៌មានខ្លួនឈាន់ខ្លួន (Time) ឬអេឡិចត្រូនិក (Memory) ធំបាន។ វាអាចបង្ហាញការងារដែលមានការងារខ្លួនឈាន់ខ្លួន (Bottleneck) ដែលបានប្រើប្រាស់បន្ទាត់បាន។

ការ Profile កើតឡើងជាផលរឹងរាល់របស់កុំពូក ដែលបានប្រើប្រាស់ដើម្បីបង្ហាញការងារដែលមានការងារខ្លួនឈាន់ខ្លួន (Bottleneck) ដែលបានប្រើប្រាស់បន្ទាត់បាន។



គោលការ 80/20 ໃນការប្រើប្រាស់បច្ចេកវិទ្យា

80%

វេលាកំឲ្យ

កម្រិតប្រើប្រាស់ 80% នៃការងារ និង 20% នៃការងារ

20%

គោលការ 20%

គោលការ 20% នៃការងារ និង 80% នៃការងារ

គោលការនេះជាប្រព័ន្ធដែលបានបង្ហាញថាអ្នកអាជីវកម្មត្រូវបានប្រើប្រាស់បច្ចេកវិទ្យាបាន និងរក្សាទុកដាក់ការងារ សរាប់អាជីវកម្ម។



เครื่องมือ Profiling ในภาษา Python

timeit

เหมาะสำหรับวัดเวลาโค้ดสั้นๆ และเปรียบเทียบประสิทธิภาพของวิธีการต่างๆ

```
import timeit
time = timeit.timeit('"-".join(str(n) for n in range(100))', number=10000)
```

cProfile

ให้ภาพรวมการทำงานทั้งหมดของโปรแกรม และแสดงเวลาที่ใช้ในแต่ละฟังก์ชัน

```
python -m cProfile -s cumulative your_script.py
```



🕒 ង់ីជុំនកសារប្របាប់ពេះ
អ៊ុកអូរីកំមនុយនកសារប្របាប់ពេះ

អ៊ុកអូរីកំម = ង់ីជុំ

នៀតិ៍គឺសំគាល់ក្នុងការប្រើប្រាស់បច្ចេកទេស និងការប្រើប្រាស់បច្ចេកទេស ដើម្បីបង្កើតការងារស្ថាប័ន និងការងារស្ថាប័ន ដែលផ្តល់ការងារស្ថាប័ន និងការងារស្ថាប័ន

ກຳຄວາມເຂົ້າໃຈ Big O Notation



$O(n^2)$ - ช้ามาก

การใช้ Loop ซ้อน Loop จะช้าลงอีก

ມາຄາລເນື້ອຂ້ອມູລອຍອະຫຸນ ເຊັ່ນ Bubble Sort



O(n) - ເຮືນພອໃຊ້

การวน Loop ใน List หนึ่งรอบ เวลาเพิ่มขึ้นตามจำนวนข้อมูลแบบเส้นตรง



O(1) - ເຮືອສຸດ

การเข้าถึงข้อมูลใน Dictionary/Set ด้วย Key ชั่งเร็วคงที่เสมอ

ເລືອກໂຄຮງສຽງຂ້ອມູລໃຫ້ດູກທານ

✗ ວິທີກໍ່ເຈົາ ($O(n)$)

```
# ข้า - การค้นหาใน List  
fruits_list = ["apple", "orange", "banana"]  
if "banana" in fruits_list:  
    print("Found!")
```

ວຽກີ່ເຮືອ (O(1))

```
# เร็ว - การค้นหาใน Set  
fruits_set = {"apple", "orange", "banana"}  
if "banana" in fruits_set:  
    print("Found!")
```

การใช้ Set หรือ Dictionary แทน List สำหรับการค้นหาจะเร็วกว่าหลายเท่าตัวเมื่อข้อมูลมีขนาดใหญ่ นี่คือการเปลี่ยนแปลงelixka ที่ส่งผลใหญ่มาก



⚡ การเขียนโค้ดเชิงประสิทธิภาพ

เทคโนโลยีนี้ช่วยปรับปรุงประสิทธิภาพในระดับการเขียนโค้ด โดยไม่ต้องเปลี่ยนอัลกอริทึมหลัก แต่สามารถสร้างความแตกต่างได้อย่างมีนัยสำคัญ

Vectorization: ពេលវេលាដំឡើងការងារ Data Science

ការិច្ចាល់ប្រាក់យ៉ាង NumPy ឬ Pandas ដើម្បីប្រើប្រាស់ផលិតផលបញ្ហាផលកំណើននៃការងារ។

✗ ខ្សោយ - Python Loop

```
result = []
for i in range(len(data)):
    result.append(data[i] * 2)
```

✓ ត្រួវ - NumPy Vectorization

```
import numpy as np
data = np.array(data)
result = data * 2
```

ប្រាក់យ៉ាងនេះត្រូវបានរៀបចំឡើងជាបញ្ជីរបស់ខ្លួន។

Loop Optimization In AIA

นำการคำนวณออกจาก Loop

หลักเลี้ยงการคำนวณที่ไม่จำเป็น
ซ้ำๆ ภายใน Loop โดยคำนวณไว้
ข้างนอกแล้วเก็บค่าไว้ใช้

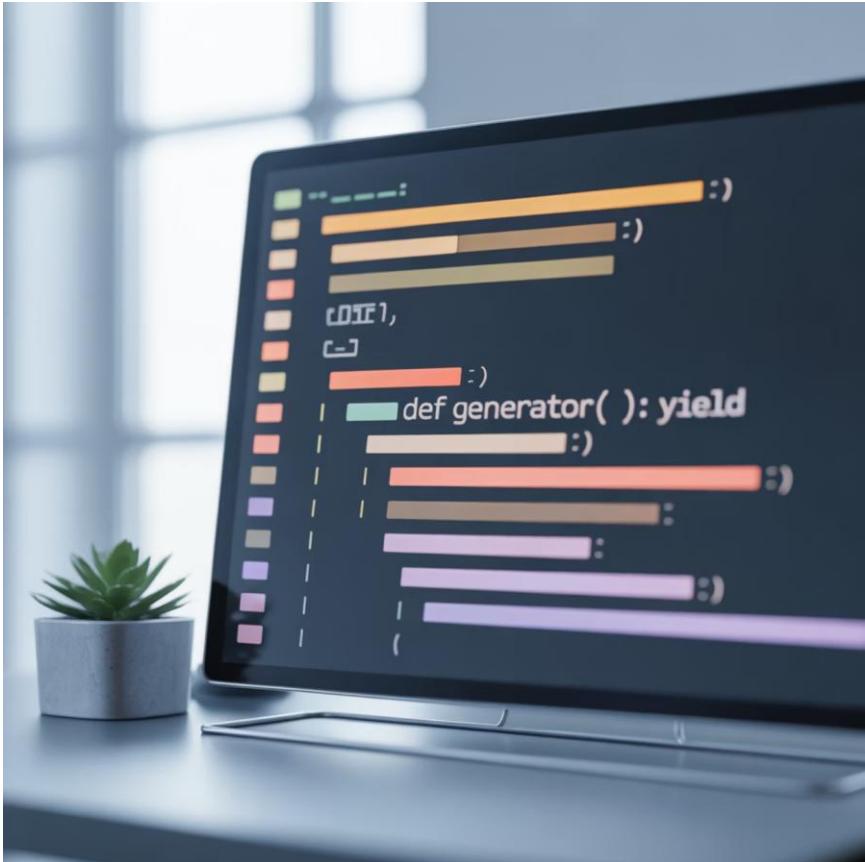
คลีกเลี้ยงการเรียนพิงก์ชันช์

ไม่เรียกฟังก์ชันภายใน Loop ที่ให้ผลลัพธ์เหมือนเดิม แต่เก็บค่าไว้ข้างนอกแล้วใช้ซ้ำ

କେବୁଁ List Comprehension

List comprehension ມັກຈະເຮືອ
ກວ່າ for loop ແບບປົກຕິໃນ
Python

Lazy Evaluation & Generators



ใช้ Generators (yield) เพื่อสร้างข้อมูลອอกมาทีละตัวเมื่อต้องการใช้เท่านั้น แทนการสร้าง List ขนาดใหญ่เก็บไว้ในหน่วยความจำทั้งหมด

```
# Generator - ឧបនគណ៌ Memory
def large_data_generator():
    for i in range(1000000):
        yield process_data(i)
    # ឯកសារនេះមិនត្រូវបានធ្វើឡើង
    for item in large_data_generator():
        if condition_met(item):
            break
```

✿ ការណើតវិទ្យាភាសាស៊តុលីមូល និង Caching

Caching = គោរព

ការណើតវិទ្យាភាសាស៊តុលីមូល ជាប្រព័ន្ធឌីជីថាមពល ដែលបានរចនាបានដឹងត្រូវ និងបានប្រើប្រាស់ជាអនុវត្តន៍យកការងារ ដើម្បីបង្កើតការងារដែលត្រួតពិនិត្យ។



Caching ||at Memoization

คือการเก็บผลลัพธ์ของการคำนวณที่ใช้บอยไว้ และนำกลับมาใช้ใหม่แทนการคำนวณซ้ำ

```
from functools import lru_cache
@lru_cache(maxsize=None)
def heavy_calculation(x, y):
    # การคำนวณที่ใช้เวลานาน
    result = complex_math_operation(x, y)
    return result
# ครั้งแรกจะซ้ำ แต่ครั้งต่อไปที่เรียกด้วย x, y เดิมจะเร็วมาก
heavy_calculation(2, 3)
# คำนวณจริง
heavy_calculation(2, 3)
# ได้ผลลัพธ์จาก Cache ทันที
```

- ❑ **ข้อควรระวัง:** ระวังการใช้ Cache มากเกินไป อาจทำให้หน่วงความจำหนด ควรกำหนด maxsize ที่เหมาะสม



គេរីថែងមីនុយបេជ្ជរីក់ខេមាម

បានក្រ៉ែងទៅមិនត្រូវបានបញ្ជីកិច្ចការណ៍ឡើង ព័ត៌មានអាជីវកម្មនៃការងារគិតជាបន្ទាន់។



Numba (JIT)

ផលិតកម្មកិច្ចការណ៍នៃការងារគិតជាបន្ទាន់ ដែលបានក្រោមការងារគិតជាបន្ទាន់។



NumPy/SciPy

ការងារគិតជាបន្ទាន់នៃការងារគិតជាបន្ទាន់ ដែលបានក្រោមការងារគិតជាបន្ទាន់។



Scikit-learn

ការងារគិតជាបន្ទាន់នៃការងារគិតជាបន្ទាន់ ដែលបានក្រោមការងារគិតជាបន្ទាន់។

Just-In-Time (JIT) Compilers ដោយ Numba

គឺជា

```
def slow_function(data):
    result = 0
    for i in range(len(data)):
        result += data[i] * data[i]
    return result
```

គឺជាដែលបានបញ្ចប់ដោយ Numba

```
from numba import jit@jit

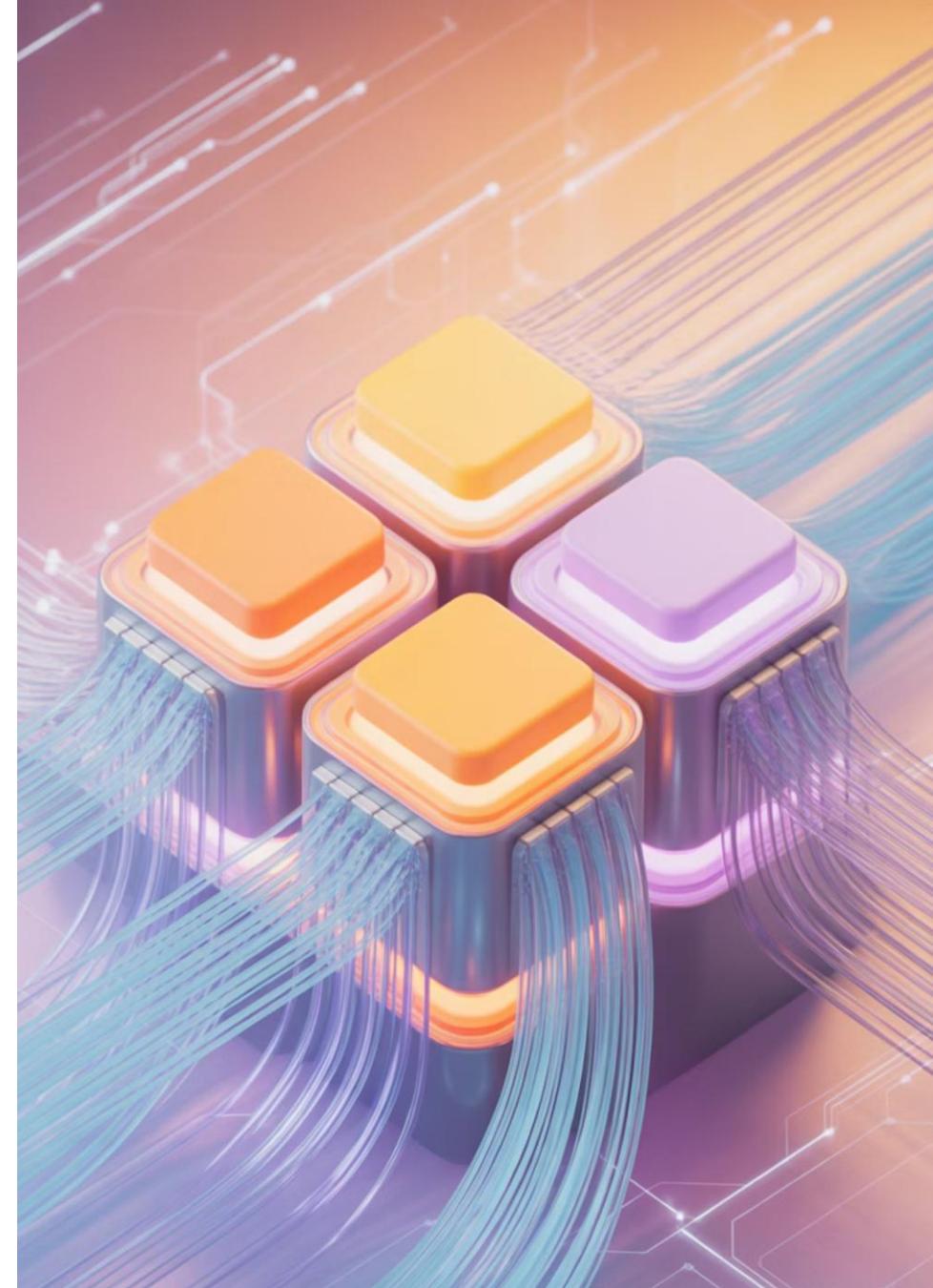
def fast_function(data):
    result = 0
    for i in range(len(data)):
        result += data[i] * data[i]
    return result
```

ការប្រើប្រាស់ @jit decorator ធ្វើឱ្យគឺជាដែលបានបញ្ចប់ដោយ Numba ការងារគិតជាកម្មភាពរបស់អ្នកសារពីវិទ្យាសាស្ត្រប៉ូល: បណ្តុះការការងារស្ថាប័ន



การทำงานพร้อมกัน (Concurrency & Parallelism)

สำหรับโปรแกรมที่ต้องทำงานหนักและมี CPU หลาย Core การทำงานพร้อมกันจะช่วยลดเวลาโดยรวมได้อย่างมีนัยสำคัญ



Multiprocessing vs Multithreading



Multiprocessing

การรับโค้ดบนหน่วย CPU Core พร้อมกัน เหมาะสำหรับงาน CPU-Bound ที่สามารถแบ่งเป็นส่วนย่อยๆ ที่เป็นอิสระต่อกันได้

- การคำนวณทางคณิตศาสตร์
 - การประเมินผลภาพ
 - Machine Learning Training

Multithreading

การรับโค้ดหมายส่วนใน Core เดียวกันแล้วไปมา หมายสำหรับงาน I/O-Bound ที่ต้องรอการตอบสนองจากภายนอก

- การดาวน์โหลดไฟล์
 - การเรียก API
 - การอ่าน/เขียนฐานข้อมูล

สรุปและขั้นตอนการวัดผล

01

Profile โปรแกรมก่อน

ใช้ cProfile หรือ bottleneck จริง อย่าเดา

02

ปรับอัลกอริทึมก่อน

เปลี่ยนโครงสร้างข้อมูล
และอัลกอริทึมก่อนเทคนิคอื่น

03

ใช้เครื่องมือช่วย

ใช้ Numba, NumPy, หรือไลบรารีที่เหมาะสม

04

วัดผลอีกครั้ง

ตรวจสอบว่าการปรับแต่งใช้ผลจริง

05

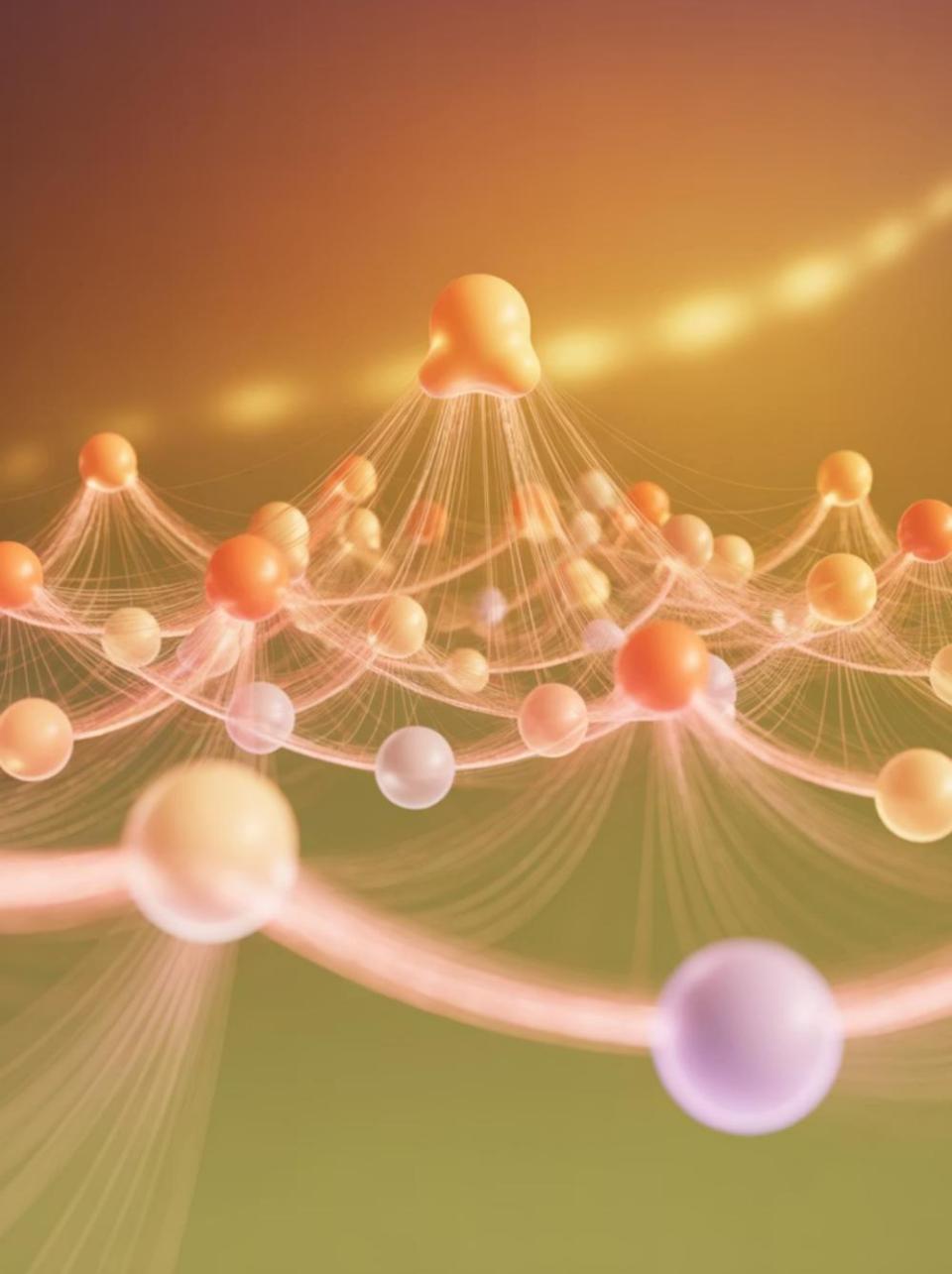
พิจารณา Parallelism

ใช้ multiprocessing/threading เมื่อจำเป็น

▣ **จำไว้:** การปรับแต่งประสิทธิภาพต้องเป็นกระบวนการที่มีการวัดผลเสมอ อย่าปรับแต่งโดยไม่มีข้อมูล!

អត្ថបទ នាក់វិទ្យាសាស្ត្រខែបុល Data Scientist

ការងារដែលធ្វើឡើងជាដំណឹងសារព័ត៌មាន និងការបង្កើតរបាយការ សរាប់អាជីវកម្ម



Design Patterns និងទេគនិក ខ្លួនស្តុងសំខាន់បាន Data Science

ទេគនិកការអក្សដប់កែដខ្លួនស្តុងកំពុងការផ្តល់នូវការការងារ និងការងារបង្កើតរបាយការ សរាប់អាជីវកម្ម។

សើងកីឡាឌាច់បានរូបរាងនេះ

01

Design Patterns សំគាល់

Factory, Singleton, Observer และ Strategy Pattern ដែលត្រូវបានប្រើប្រាស់នៅក្នុងការងារជាន់ខ្ពស់នៃវិភាគយាតាសេតុលីប៊ូល។

02

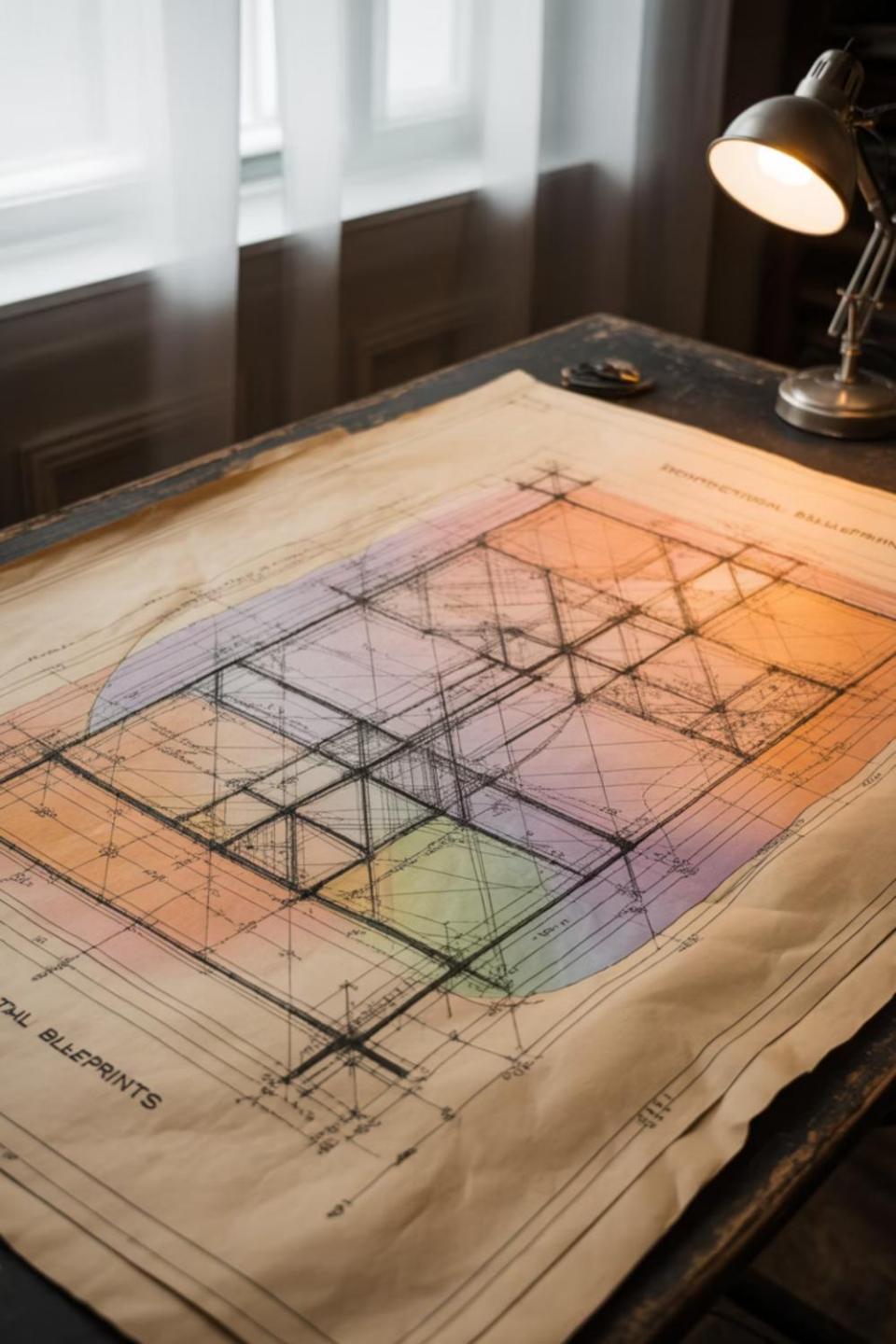
Metaclasses និង Dynamic Code

ការប្រើប្រាស់ Metaclasses និង Dynamic Code ជាការងារស្ថាប់ការសរុបបញ្ជីការងារជាន់ខ្ពស់នៃវិភាគយាតាសេតុលីប៊ូល។

03

ការរៀបចំការងារជាន់ខ្ពស់នៃវិភាគយាតាសេតុលីប៊ូល

វិធីការរៀបចំការងារជាន់ខ្ពស់នៃវិភាគយាតាសេតុលីប៊ូល ដើម្បីបង្កើតការងារជាន់ខ្ពស់នៃវិភាគយាតាសេតុលីប៊ូល។



Design Patterns

ແນ່ແບບກາຣແກ້ປັງຫາທີ່ມີປະສິກົດກາພ

Design Patterns ຄືອແນ່ແບບກາຣແກ້ປັງຫາທີ່ເກີດຂຶ້ນບ່ອຍໆ ໃນກາຣອອກແບບໂຄົດໄວ້ ສໍາຮັບຈານ Data Science ແພກເກີຣນເໜ່ານ໌ຊ່ວຍໃຫ້ໂຄົດມີຄວາມຢັດຍຸນ ຈັດກາຣຈ່າຍ ແລະນຳກລັບມາໃຊ້ໃໝ່ໄດ້ສະດວກ ພຣອມກັ້ນທີ່ໃຫ້ກາຣັບປິດໄວ້ ແລະຮັບປິດໄວ້ເຄຣະຮັບປິດໄວ້

Factory Pattern: สร้าง Object ได้หลากหลาย

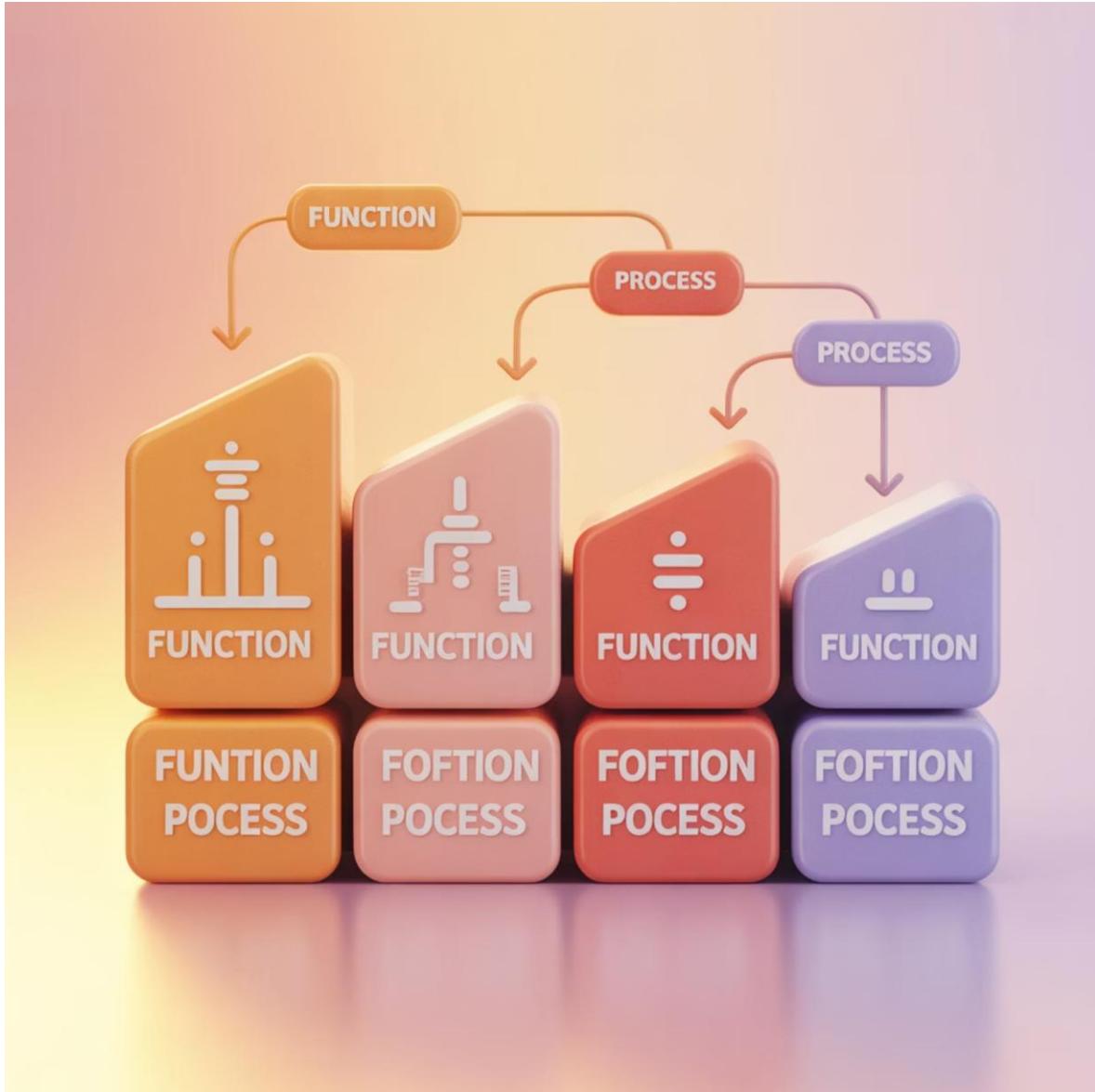
គ្រែកការងារ

สร้าง Object โดยไม่ต้องระบุคลาสที่แน่นอน เหมาะสำหรับการสร้าง "โนเดล" หรือ "ตัวໂຄດຂ້ອງມູລ" ที่แตกต่างกันไปตามสถานการณ์

ចំណាំ

- ลดการ couple ระหว่าง code กับ concrete classes
 - ง่ายต่อการเพิ่มโมเดลใหม่
 - จัดการการสร้าง object ที่ซับซ้อนได้ดี

```
def get_model(model_name):
    if model_name == "logistic":
        return LogisticRegression()
    elif model_name == "random_forest":
        return RandomForest()
    elif model_name == "xgboost":
        return XGBoost()
    else:
        raise ValueError("Unknown model")
```





Factory Pattern ในงานจริง

ตัวอย่างการใช้งาน

สร้างฟังก์ชัน `get_model(model_name)` ที่จะคืน Object ของ `LogisticRegression`, `RandomForest`, หรือ `XGBoost` ตาม `model_name` ที่รับเข้ามา ทำให้เราสลับโมเดลทดสอบได้ง่ายๆ โดยไม่ต้องแก้โค้ดหลัก

Model Selection

เลือกโมเดลได้แบบใดนา米ก ตามพารามิเตอร์ที่ส่งเข้ามา

Easy Experimentation

ทดลองโมเดลต่างๆ ได้สะดวก โดยเปลี่ยนแค่ชื่อโมเดล

Maintainable Code

โค้ดหลักไม่ต้องเปลี่ยน เพิ่มโมเดลใหม่ได้ง่าย

Singleton Pattern: ຮັບສ່ວນເຊີຍວິນໂລກ

ຮັບປະກັນວ່າຈະມີ Object ຂອງຄລາສນີ້ຢູ່ເພີຍງ **ຊື່ເດືອວ** ກ່ຽວກັ້ງໂປຣແກຣມ
ເປັນແພຳເຖິງນີ້ມາກໃນການຈັດການກຣັບພາກຮ່ວມມືກັນ
ເຊັ່ນ ການເຊື່ອມຕ່ອງຈານຂ້ອນນູລ ການໂຄດໄຟສ configuration ແລ້ວການ
ຈັດການ logging

"การນີ້ Object ເພີຍງຕັວເດີຍວ່າຍິ້ງກັບພຍາກຮູກໃຫຍ້ຢ່າງມີປະສົກອີກາພ
ແລະໄມ່ເກີດການຈ້າງຊ້ອນ"



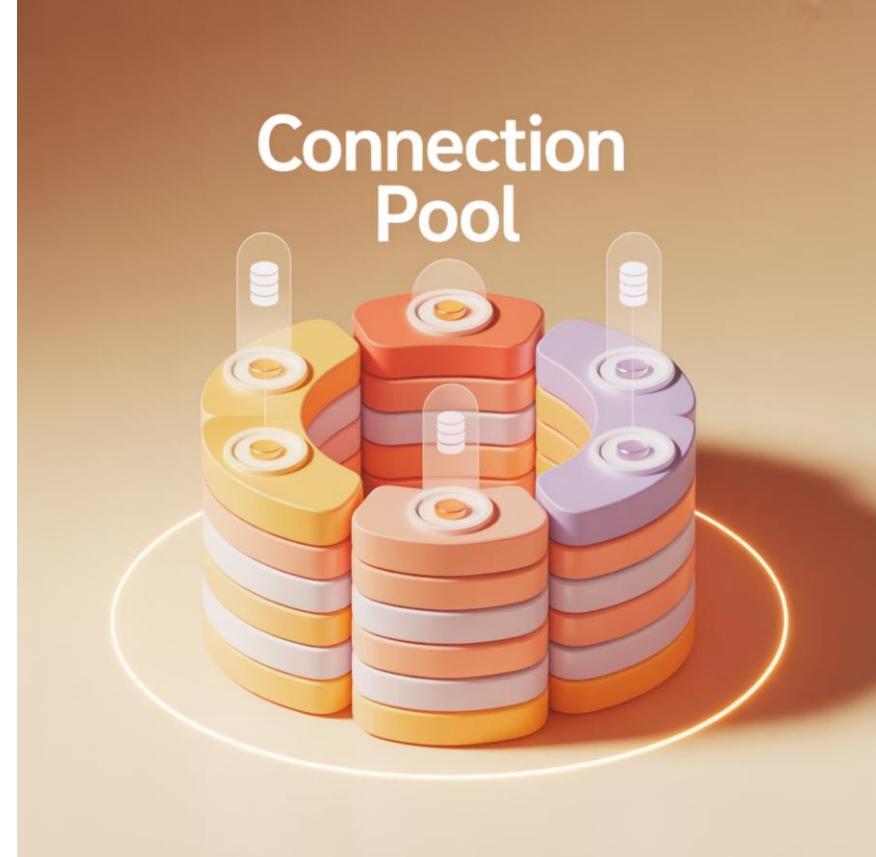
Singleton Pattern: ตัวอย่างการใช้งาน

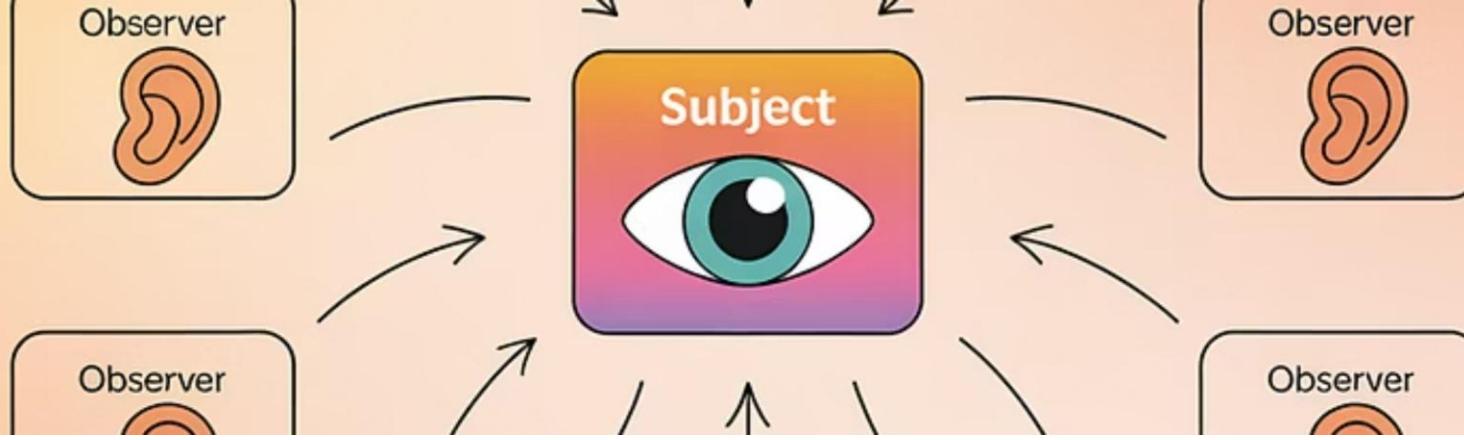
การจัดการ Database Connection

ใช้จัดการการเชื่อมต่อฐานข้อมูล (Database Connection) หรือการโหลดไฟล์ Config เพียงครั้งเดียว เพื่อให้ทุกส่วนของโปรแกรมใช้ทรัพยากรที่แชร์ร่วมกันนี้ ได้อย่างมีประสิทธิภาพและไม่ซ้ำซ้อน

របៀបចាប់

- ประ helyัดกรพยากรณ์ Memory และ CPU
 - การ Configuration ที่สมบูรณ์แบบของระบบ
 - ป้องกันปัญหา Race Condition
 - จัดการ State ส่วนกลางได้ดี





Observer Pattern: แจ้งเตือนอัตโนมัติ

เมื่อ Object หนึ่ง (Subject) มีการเปลี่ยนแปลงสถานะ Object อื่นๆ (Observers) ที่คือ "สังเกตการณ์" จะได้รับแจ้งเตือนและอัปเดตตัวเองโดยอัตโนมัติ นี่คือแพทเทิร์นที่ทรงพลังมากในการสร้างระบบที่ตอบสนองต่อการเปลี่ยนแปลงได้กันที

1

Subject เปลี่ยน State
Object หลักมีการเปลี่ยนแปลง

2

แจ้งเตือน Observers
ส่งสัญญาณไปยังผู้สังเกตการณ์

3

อัปเดตอัตโนมัติ
Observers ทำงานตามที่กำหนด



Observer Pattern ในการเทรนโมเดล ตัวอย่างการใช้งานจริง

ในระหว่างการเทรนโมเดล (Subject) เราสามารถมี Observers หลายตัว เช่น ตัวที่ค่อยบันทึก Log, ตัวที่พล็อตกราฟ Loss, และตัวที่ค่อยเช็คเงื่อนไขเพื่อกำ Early Stopping เมื่อ Loss ไม่ลดลงแล้ว ทุกตัวจะทำงานกันทีก่อนเดลเทรนครบแต่ละรอบ (Epoch)

Logger Observer

บันทึกข้อมูลการเทรน metrics และ parameters ลงในไฟล์ log

Plotter Observer

สร้างกราฟแสดงการเปลี่ยนแปลงของ Loss และ Accuracy แบบ real-time

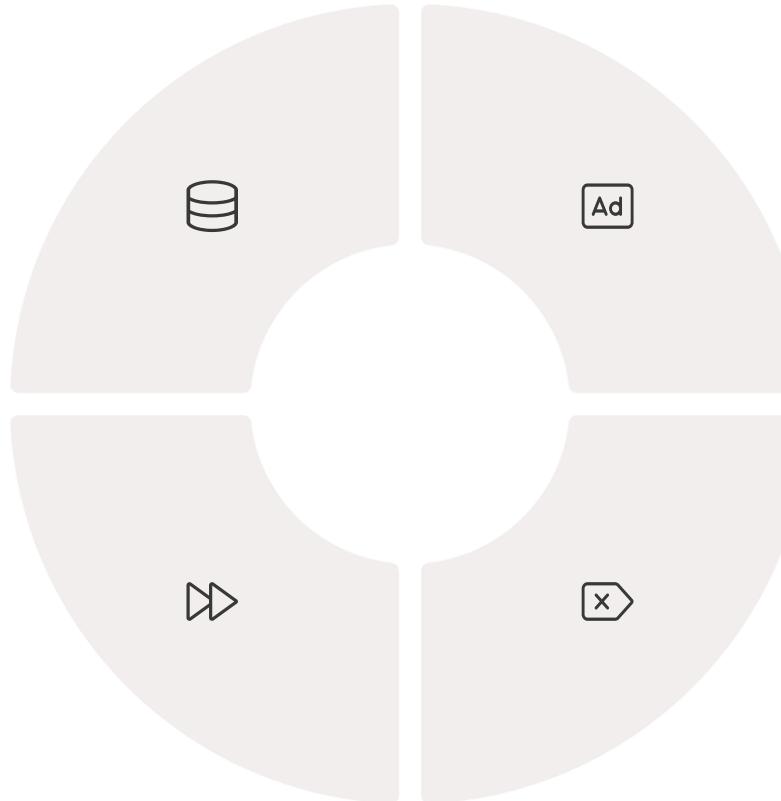
Early Stopping Observer

ตรวจสอบและหยุดการเทรนเมื่อโมเดลไม่มีการปรับปรุงแล้ว

Strategy Pattern: សល់កលូយុធ៍ដែលកំណត់

ការប្រើប្រាស់សល់កលូយុធ៍ ឬ "អេឡិចត្រូនិក" នៅក្នុងការងារបានបង្កើតឡើងដោយមិនត้องផ្តល់ការងារ និងការងារនៃការងារទៅក្នុងគ្រប់គ្រងការសរសៃរបស់អ្នកជាប្រធានបទ។

Mean Imputation
ពិនិត្យនិងគិតថាទីតាំងនៃការបង្ហាញនៃការងារ។



Median Imputation
ពិនិត្យនិងគិតថាទីតាំងនៃការបង្ហាញនៃការងារ។

Forward Fill
ពិនិត្យនិងគិតថាទីតាំងនៃការបង្ហាញនៃការងារ។

Drop NA
លួចលាចការងារដែលមិនមែនតម្លៃតាមរយៈការងារ។

Strategy Pattern: ព័ត៌មានយោងការិច្ចាប់

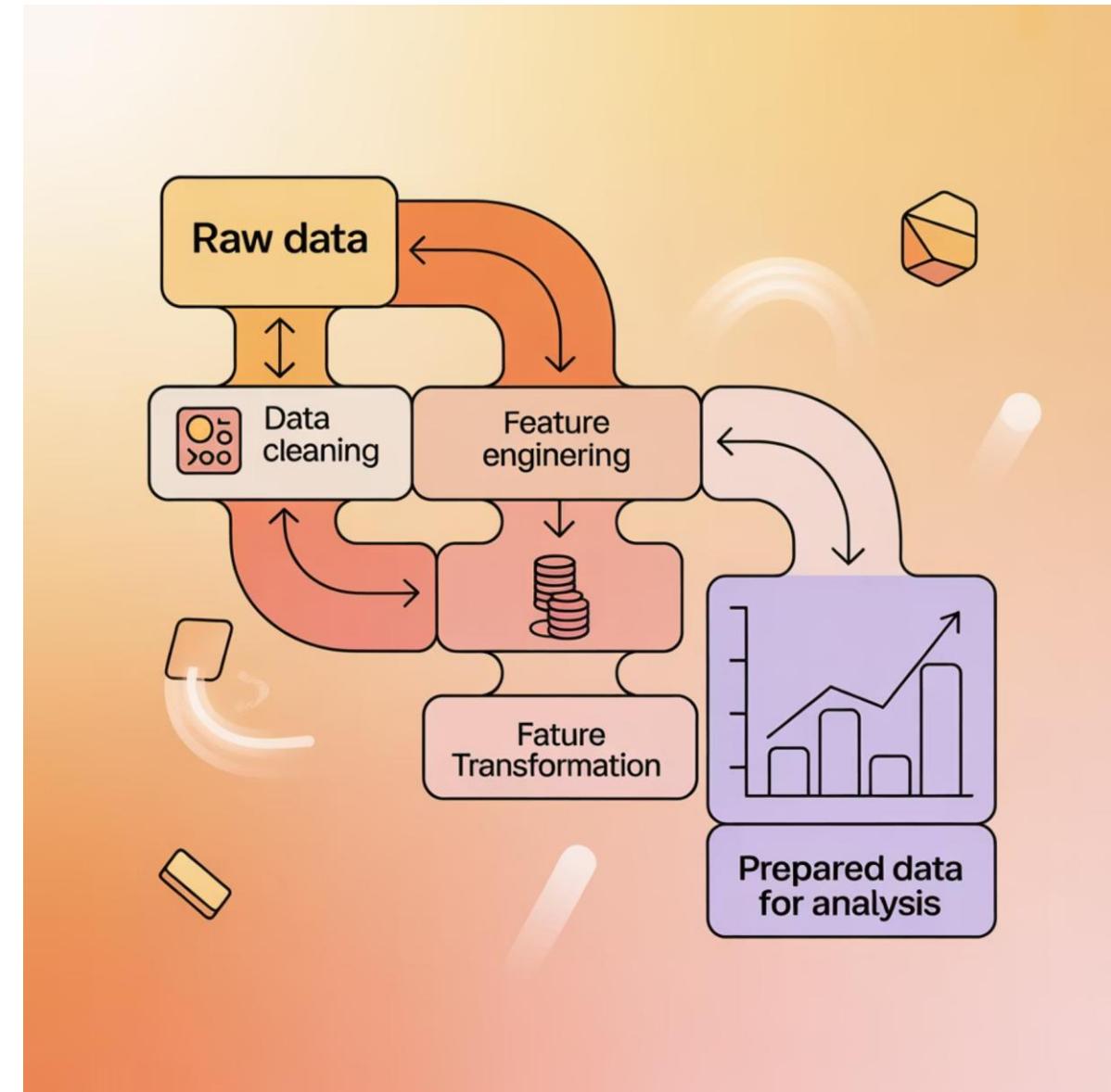
DataPreprocessor Class

រាយការណ៍ DataPreprocessor ដែលមានលក្ខណៈថា ត្រូវបានសម្រេចឡើងនៅក្នុងការបង្កើតការងារ។ នៅក្នុងការងារនេះ ត្រូវបានបង្កើតឡើងជាអំពីរបាយការណ៍ដែលត្រូវបានបង្កើតឡើង។

ខ្លួនខ្លួន Strategy Pattern

- ផ្តល់នូវការងារដែលត្រូវបានបង្កើតឡើងនៅក្នុង client code
- ការងារនេះត្រូវបានបង្កើតឡើងដោយសារការងារគ្រប់គ្រងទិន្នន័យ
- ការងារនេះត្រូវបានបង្កើតឡើងដោយសារការងារគ្រប់គ្រងទិន្នន័យ

```
class DataPreprocessor:  
    def __init__(self, strategy):  
        self.strategy = strategy  
    def handle_missing(self, data):  
        return self.strategy.process(data)  
    # ផ្តល់នូវការងារដែលត្រូវបានបង្កើតឡើងនៅក្នុង Runtime  
processor = DataPreprocessor(MeanImputation())  
clean_data = processor.handle_missing(raw_data)
```



Metaclasses

Dynamic Code Generation

เทคโนโลยีที่ช่วยให้เราสามารถสร้างหรือปรับเปลี่ยนพฤติกรรมของ "คลาส" ได้แบบไม่ต้องเขียนโค้ดเพิ่ม คือความสามารถที่ทำให้เราสามารถกำหนดรูปแบบของคลาสได้โดยไม่ต้องเขียนโค้ดในคลาสเดิม



การสร้างคลาสแบบไดนามิก

เราสามารถสร้างคลาสขึ้นมาใหม่ตอนโปรแกรมทำงานได้โดยใช้ type() เมื่อจะทำการสร้างคลาสที่โครงสร้างขึ้นอยู่กับข้อมูลที่รับเข้ามา เทคนิคนี้มีประโยชน์มากเมื่อเราต้องการสร้างโครงสร้างข้อมูลที่ยืดหยุ่นตามข้อมูลจริง

อ่านข้อมูล CSV

โหลดไฟล์ข้อมูลและดึงชื่อคอลัมน์ออกมานะ

สร้าง Attributes

แปลงชื่อคอลัมน์เป็น attributes ของคลาส

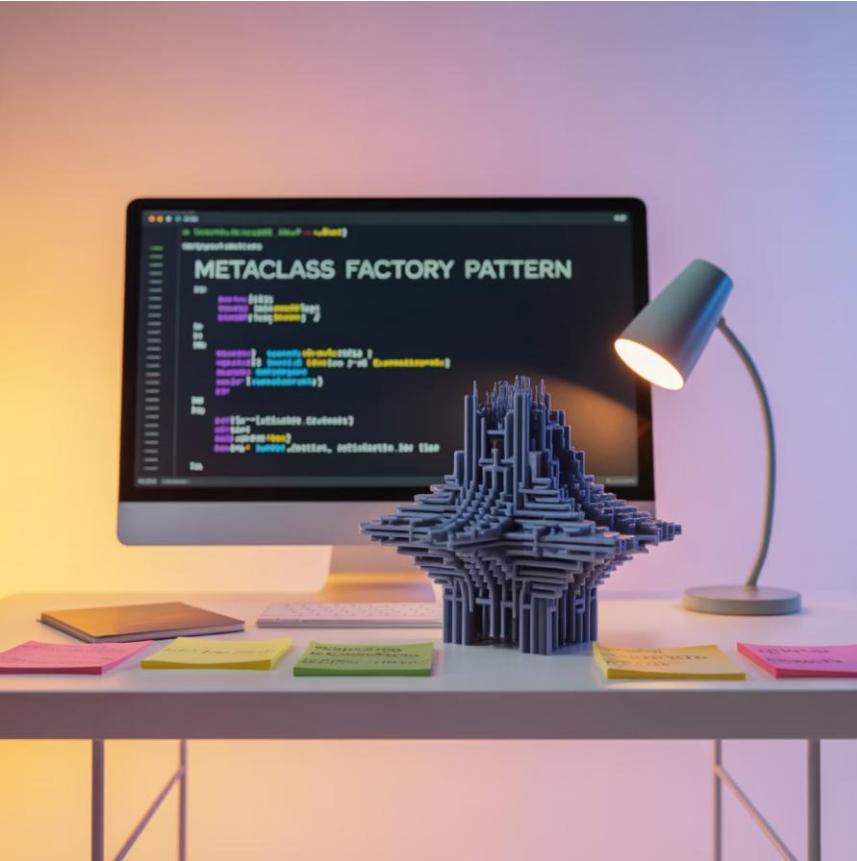
Generate Class

ใช้ type() สร้างคลาสใหม่ด้วย attributes ที่กำหนด

ใช้งานกันตี

ได้คลาสที่มี structure ตรงกับข้อมูลชุดนั้นๆ

Metaclasses: คลาสที่สร้างคลาส



Class Factory

ප්‍රසාදය

- ควบคุมการสร้างคลาสได้ทุกขั้นตอน
 - เพิ่ม functionality ให้คลาสอัตโนมัติ
 - จัดการ registration และ validation
 - สร้าง DSL (Domain Specific Language)

Metaclass: Model Registry

ព័ត៌មានលម្អិត

ឱ្យ Metaclass ដឹងពី "លក្ខណៈបើយោប់" គម្រោងណាមួយដែលបានបង្កើតឡើងដោយខ្លួន ដើម្បីរក្សាទុកដាក់លក្ខណៈបើយោប់។ ក្នុងការបង្កើតគម្រោង MyAwesomeModel នៅក្នុងការងារគិតថ្លែងការសាស្ត្រខ្ពស់បែងចាយរបស់ខ្លួន ត្រូវបានរក្សាទុកដាក់លក្ខណៈបើយោប់។



Auto Registration

ក្នុងការបង្កើតគម្រោងណាមួយ ត្រូវបានរក្សាទុកដាក់លក្ខណៈបើយោប់។



Validation

ត្រូវបានរក្សាទុកដាក់លក្ខណៈបើយោប់។



Systematic Management

ត្រូវបានរក្សាទុកដាក់លក្ខណៈបើយោប់។





Memory Management

การจัดการหน่วยความจำใน Python

การเข้าใจวิธีที่ Python จัดการหน่วยความจำเป็นสิ่งสำคัญเมื่อต้องทำงานกับข้อมูลขนาดใหญ่ การออกแบบระบบที่มีประสิทธิภาพต้องคำนึงถึงการใช้หน่วยความจำอย่างรอบคอบ

Garbage Collection ใน Python

Reference Counting + Cyclic GC

Python จัดการหน่วยความจำอัตโนมัติโดยใช้ระบบ **Reference Counting** เป็นหลัก เมื่อไม่มีตัวแปรใดๆ ซื้อไปยัง Object นั้นแล้ว มันจะถูกกลบออกจากหน่วยความจำทันที นอกจากนี้ยังมีระบบ **Cyclic GC** ที่คอยจัดการกับ Object ที่อ้างอิงกันเป็นวงกลม (Cyclic References) ซึ่ง Reference Counting อย่างเดียวจัดการไม่ได้

1 Object Creation

สร้าง Object ใหม่ Reference count = 1

2 Reference Changes

เพิ่ม/ลด reference count ตามการใช้งาน

3 Count = 0

เมื่อไม่มี reference จะถูกลบกันที

4 Cyclic Detection

GC จัดการ cyclic references

Python Memory Reference counting



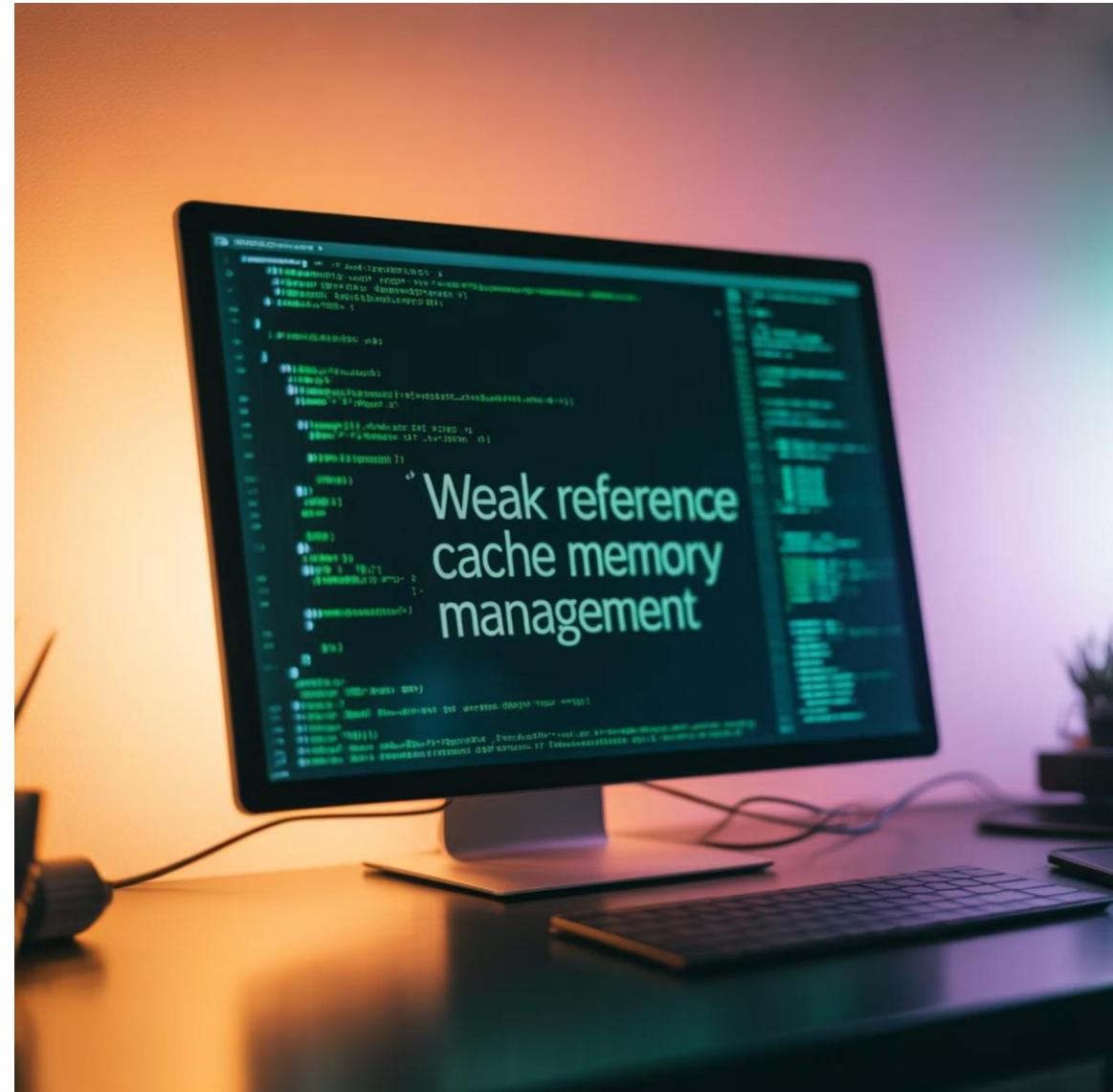
Weak References: ວ້າງອົງແບບໄມ່ເພີ່ມຕົວນັບ ຮັກກາຣກຳງານ

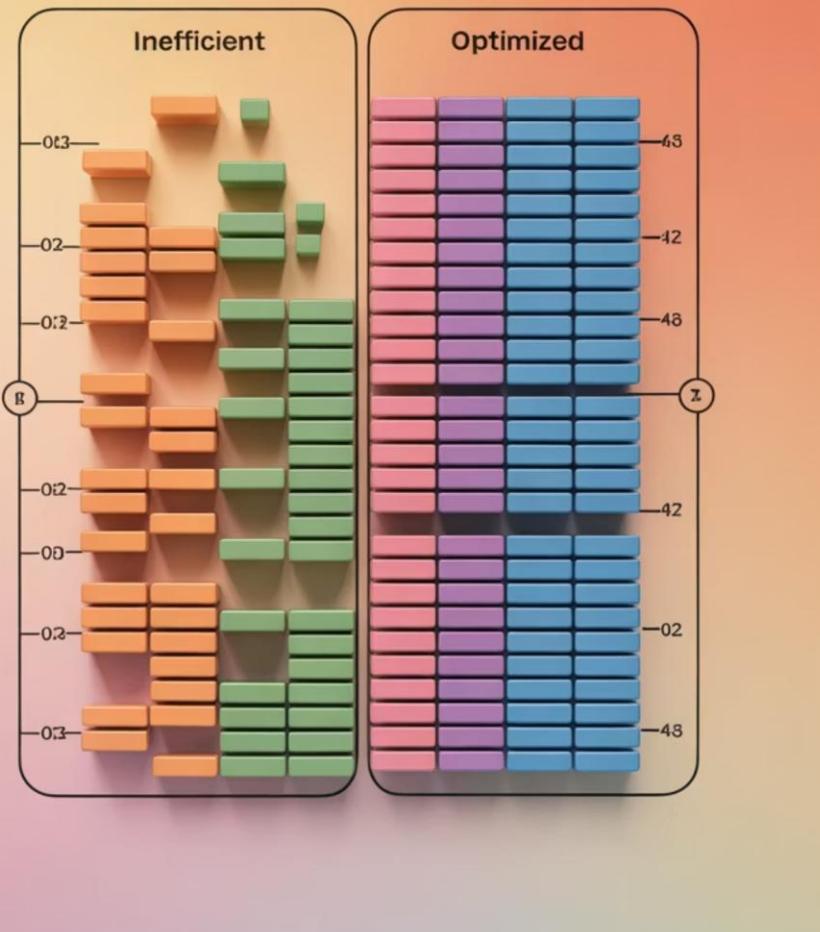
គេការចាយចងកើង Object ទៅ ឲ្យធំ គា Reference Count ហាងក់ Object បានចូរកលប់ទៅ GC តាមប្រពិនិត្យមី Weak Reference ខ្លួយក្នុងពាណិជ្ជកម្ម

Cache กីឡាយការណ៍

ເໝາະສຳເຮັດກາຮຽນ **Cache** ເກີບຂອ້ມລູບນາດໃຫຍ່ ເຊັ່ນ ພຸລັພິບຈາກການຄໍານວນ Feature ທີ່ ຊັບຊັອນ ເຮັດເກີບ Weak Reference ໄວ ດ້ວຍເນັ້ນວຍຄວາມຈຳໄມ່ພອ Python ກີ່ສາມາດໂຄບຂອ້ມລູບໃນ Cache ກັ້ນໄປໄດ້ໂດຍໄປກະທຸກກັບສ່ວນອື່ນຂອງໂປຣແກຣມ

- ▢ **ข้อดีของ Weak References:** ป้องกัน Memory Leaks, ทำให้ Cache มีความยืดหยุ่น และไม่ขัดขวางการทำงานของ Garbage Collector





slots: ប្រអយៗ
បង្កើយគាន់ជាសុទេរ

โดยปกติ Object ใน Python จะมี `__dict__` เพื่อเก็บ Attributes ต่างๆ ซึ่งใช้หน่วยความจำค่อนข้างเยอะ แต่ถ้าเราระบุ `__slots__` ที่แน่นอนของคลาสไว้ใน `__slots__` ตัวแปร `__dict__` จะไม่ถูกสร้างขึ้น

40-50% 2x

Memory Savings Faster Access

ลดการใช้หน่วยความจำ การเข้าถึง attribute เร็ว ได้มาก

1M+

Small Objects

ເນັ້ນາກັບກາຮ່າງ
object ຈຳນວນນຳກຳ

ຕົວອຍ່າງການໃຊ້ງານ

หากเราต้องสร้าง Object ขนาดเล็กเป็นล้านๆ ชิ้น (เช่น Object เก็บข้อมูลจุดพิกัด x, y, z) การใช้ `__slots__` จะช่วยลดการใช้หน่วยความจำลงได้มาก ทำให้ประมวลผลข้อมูลจำนวนมากได้โดยไม่เกิด Memory Error

សរុប: Design Patterns & តេកិច្ចខ្ពស់

Design Patterns

Factory, Singleton, Observer, Strategy ជំនួយដែលអាចបង្កើតរឿងរាល់បាន និងការងារសាស្ត្រប៊ូលឌីជីថល។

Metaclasses

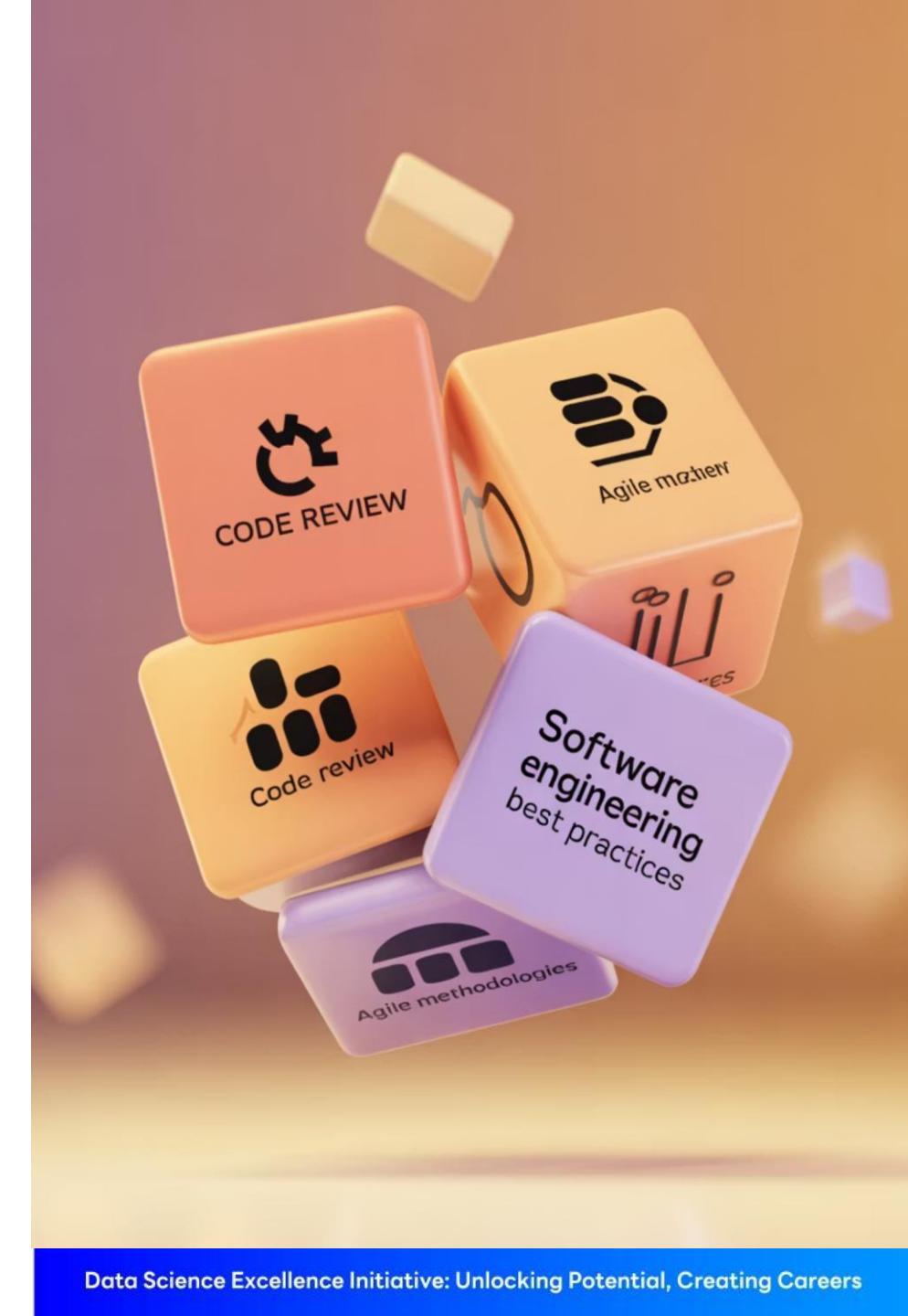
តេកិច្ចខ្ពស់ដែលអាចរួមចំណាំការសរោប់ការបង្កើតរឿងរាល់បាន និងការការពារការងារសាស្ត្រប៊ូលឌីជីថល។

Memory Management

ការបង្ហាញ GC, Weak References និង `__slots__` ជាការងារសាស្ត្រប៊ូលឌីជីថលដែលអាចបង្កើតរឿងរាល់បាន និងការការពារការងារសាស្ត្រប៊ូលឌីជីថល។

ខ្ពស់បាន

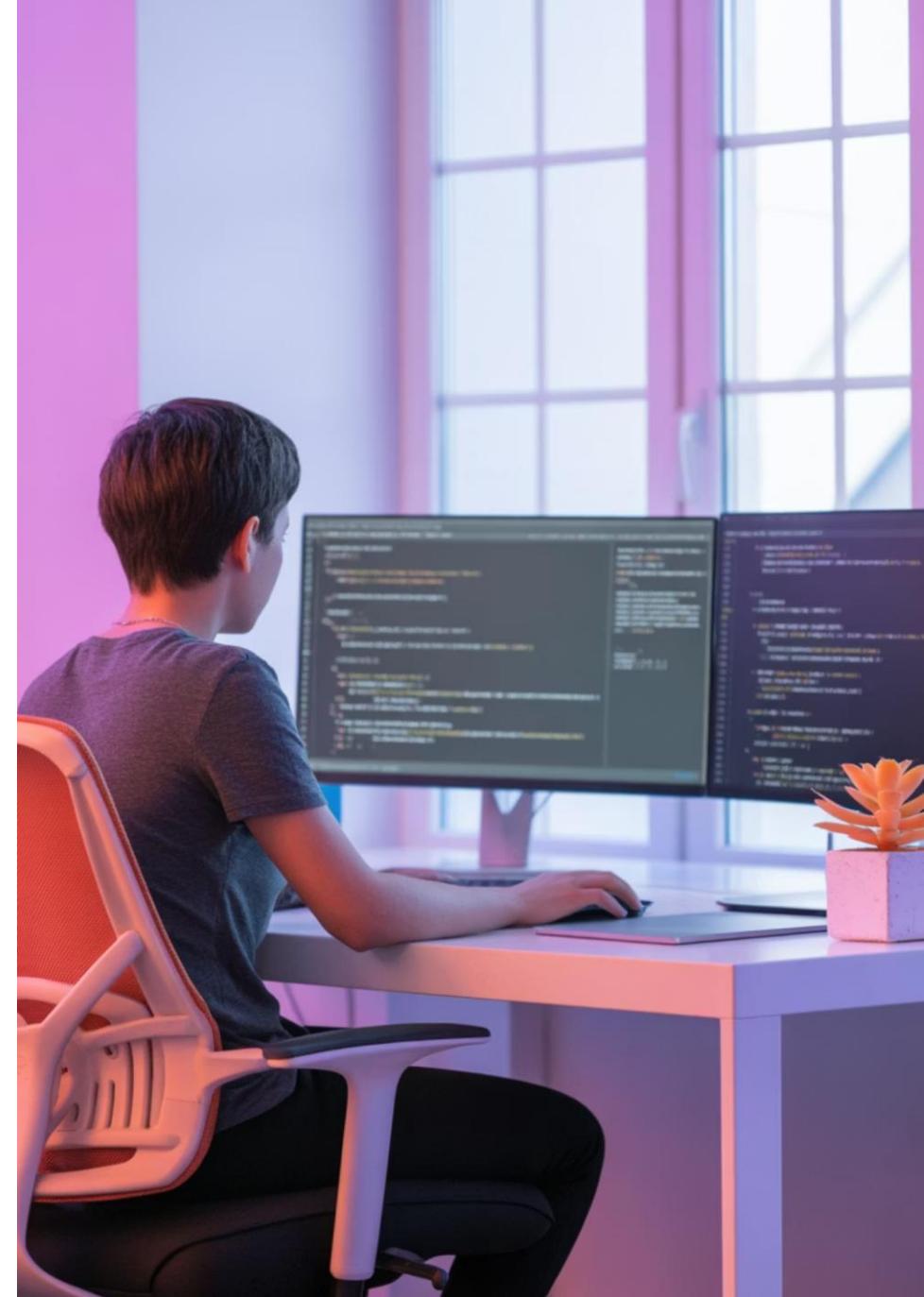
លើកទីនេះ និងការងារសាស្ត្រប៊ូលឌីជីថល ត្រូវបានបង្កើតឡើង និងការការពារការងារសាស្ត្រប៊ូលឌីជីថល ដើម្បីបង្កើតរឿងរាល់បាន និងការការពារការងារសាស្ត្រប៊ូលឌីជីថល។



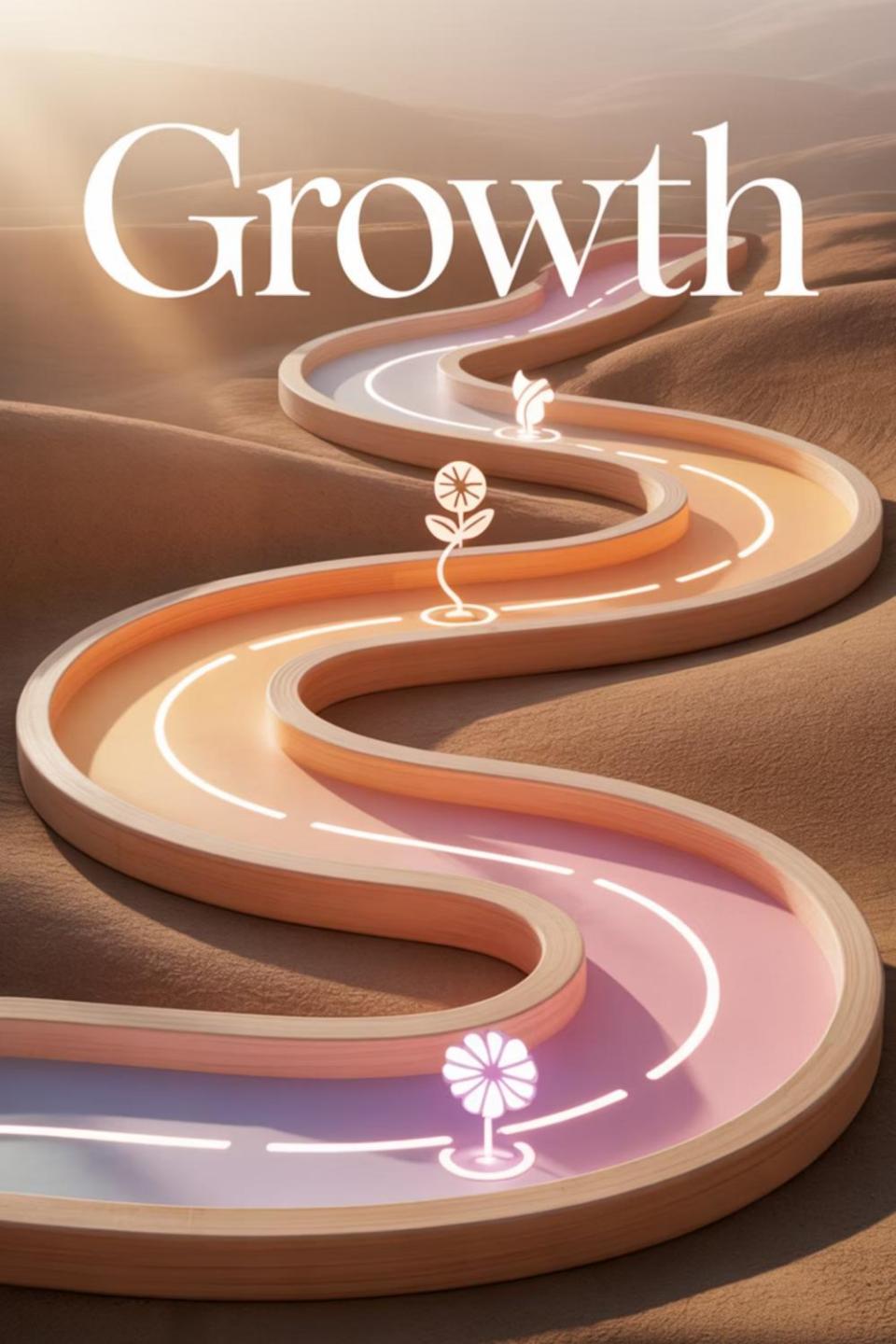
ຮັກກາຣເຂີຍນໂຄດແບບມືອອາຊີວ

Clean Code & Best Practices

จะลึกหลักการเขียนโค้ดที่มีคุณภาพสูง ตั้งแต่ Clean Code, Exception Handling ไปจนถึง Modular Development ที่จะยกระดับทักษะการพัฒนาซอฟต์แวร์ของคุณ มาตรฐานระดับมืออาชีพ



Growth



เนื้อหาที่จะได้เรียนรู้

01

หลักการ Clean Code

การเขียนโค้ดที่อ่านง่าย เข้าใจได้ และดูแลรักษาง่าย

02

Exception Handling

การจัดการข้อผิดพลาดอย่างมีประสิทธิภาพ และปลอดภัย

03

Modular Development

การพัฒนาโค้ดแบบโมดูลที่นำกลับมาใช้ใหม่ได้

Clean Code

គឺជាកួតុដែលបានរចនាទៅក្នុងកិច្ចការប្រើប្រាស់បន្ថែមទៀត។

"គឺជាកួតុដែលបានរចនាទៅក្នុងកិច្ចការប្រើប្រាស់បន្ថែមទៀត។"

Clean Code គឺជាកួតុដែលបានរចនាទៅក្នុងកិច្ចការប្រើប្រាស់បន្ថែមទៀត។



```
xe օթ սց տեմպւր )  
xa xl clapnaչւ շտենտեմպվել ")  
xe տընդեսիէր) e խ=  
renþvesryartere ))=)  
mmptra ))  
xe tttuotecəprig [enmpar))  
xə Jcoecestensf e] teen|c սնուր.- Իմ'eo))  
t heդյօրւնցօլ) այսվլե");
```

```
— գումակ  
— productprice_cuiicun)  
— avie  
— adovl)ascddագեciciceցeucctice)  
— pdtԵՎՈղեաice)  
— add_առծօդօաdediciւեցէ  
— artrԵՎլdeioe  
— auut(իստիցdipede_cvficon)
```

การตั้งชื่อที่สื่อความหมาย

ชื่อตัวแปร (Variables)

ดี: customer_age, raw_dataframe

ไม่ดี: x, df, temp

- ใช้คำนามที่สื่อถึงสิ่งที่เก็บ
- หลีกเลี่ยงชื่อสับๆ ที่ไม่มีความหมาย
- ใช้ snake_case สำหรับ Python

ชื่อฟังก์ชัน (Functions)

ดี: calculate_gross_sales(), validate_input_data()

ไม่ดี: process_data(), do_stuff()

- ใช้คำกริยาที่บอกว่าฟังก์ชัน "ทำอะไร"
- เจาะจงและชัดเจน
- หลีกเลี่ยงชื่อที่กวนว้างเกินไป

អត្ថបទ នកវិភាគការពេទ្យលេខាមួល Data Scientist

Single Responsibility

អនុវត្តន៍ការងារដែលត្រូវរាយការណ៍ក្នុងការងារមួយនៅក្នុងការងារមួយ។

- ស្រួលរាយការងារក្នុងការងារមួយ
- ក្នុងការងារមួយត្រូវរាយការណ៍ក្នុងការងារមួយ
- ក្នុងការងារមួយត្រូវរាយការណ៍ក្នុងការងារមួយ

តាមរយៈការបញ្ចូលការងារមួយ

- If-else ឬ for-loop ត្រូវបានក្លាយក្នុងការងារមួយ។
- ប្រើប្រាស់ការបញ្ចូលការងារមួយ
 - ប្រើប្រាស់ការបញ្ចូលការងារមួយ
 - ប្រើប្រាស់ការបញ្ចូលការងារមួយ

ធនការក្នុងការងារមួយ

ធនការក្នុងការងារមួយត្រូវបានក្លាយក្នុងការងារមួយ។

- ប្រើប្រាស់ការបញ្ចូលការងារមួយ
- ប្រើប្រាស់ការបញ្ចូលការងារមួយ
- ប្រើប្រាស់ការបញ្ចូលការងារមួយ

គិតិយាកាសតែរបៀប

ឱ្យមិនបានស្ថាបន



ឱ្យមិនបានស្ថាបន ដើម្បីធ្វើការដែលល្អជាពេលវេលាដែលស្ថាបន។ គិតិយាកាសតែរបៀប នឹងធ្វើការដែលល្អជាពេលវេលាដែលស្ថាបន។

ឱ្យមិនបានស្ថាបន

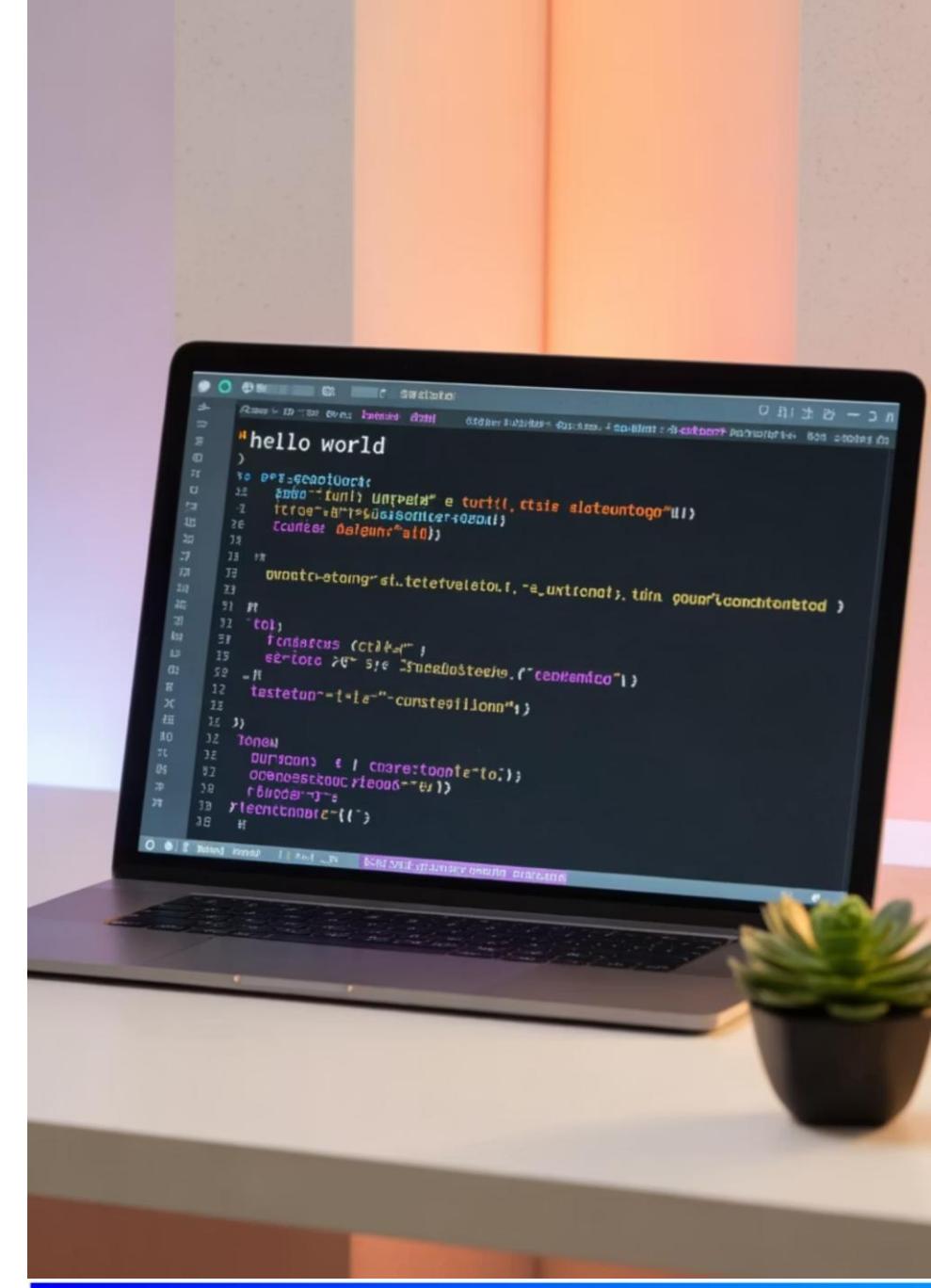


ឱ្យមិនបានស្ថាបន ដើម្បីធ្វើការដែលល្អជាពេលវេលាដែលស្ថាបន។ គិតិយាកាសតែរបៀប នឹងធ្វើការដែលល្អជាពេលវេលាដែលស្ថាបន។

គិតិយាកាសតែរបៀប



ឱ្យមិនបានស្ថាបន ដើម្បីធ្វើការដែលល្អជាពេលវេលាដែលស្ថាបន។ គិតិយាកាសតែរបៀប នឹងធ្វើការដែលល្អជាពេលវេលាដែលស្ថាបន។



Exception Handling

การจัดการข้อผิดพลาดอย่างมืออาชีพ

การจัดการข้อผิดพลาดที่ดีทำให้โปรแกรมไม่หยุดทำงานกลางคัน เมื่อเจอปัญหาที่ไม่คาดคิด และช่วยให้แก้ไขข้อบกพร่อง (Debug) ได้ง่าย ระบบที่มั่นคงต้องقادการณ์และจัดการกับสถานการณ์ผิดปกติได้อย่างเหมาะสม

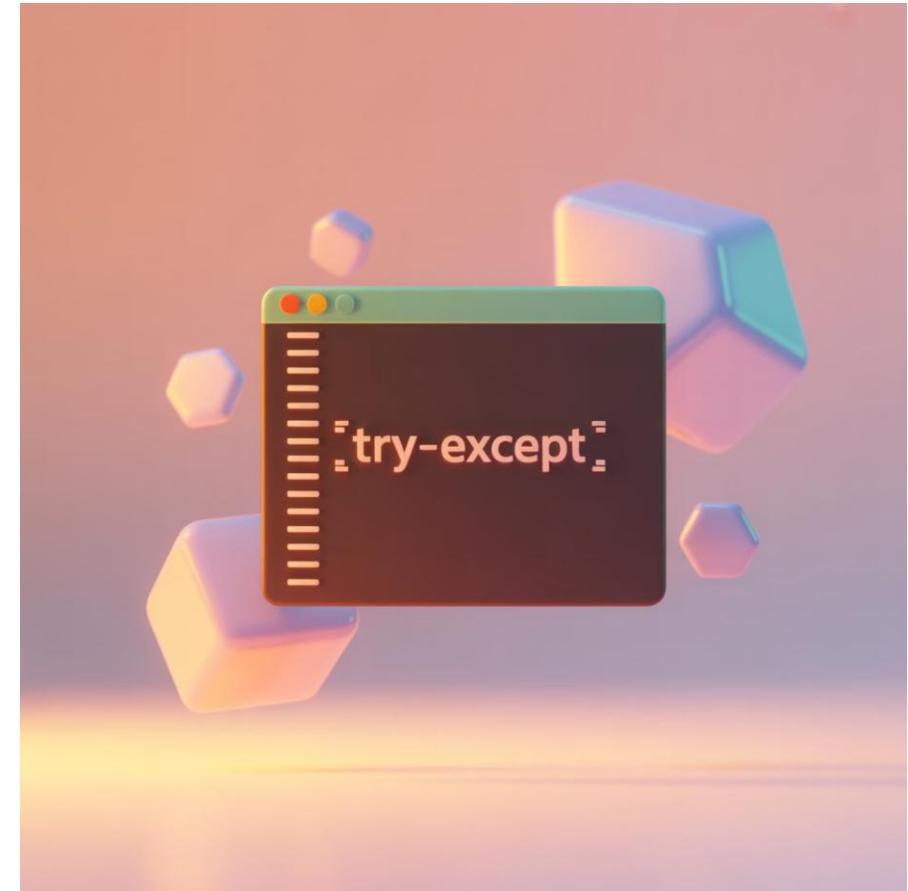


Best Practices สำหรับ Try-Except

เจาะจง Exception ที่จะดักจับ

```
# ไม่ดี
try:
    process_data()
except:
    print("Error occurred")
# ดี
try:
    process_data()
except FileNotFoundError:
    print("File not found")
except ValueError:
    print("Invalid value")
```

- ระบุ Exception ที่เดพะเจาะจง
 - ช่วยให้รู้สาเหตุที่แท้จริง
 - จัดการแต่ละกรณีได้เหมาะสม



- ▣ **เคล็ดลับ:** การเจาะจง Exception ช่วยให้เราจัดการกับแต่ละสถานการณ์ได้อย่างเหมาะสม และไม่ปิดบังข้อผิดพลาดที่ไม่คาดคิด

Finally และ Else ໃນ Exception Handling

1

Finally Block

គឺជាបុលិក finally ដែលរាយការណ៍ នៅក្នុងការបង្កើតរបាយការ ឬការងាររបាយការ។

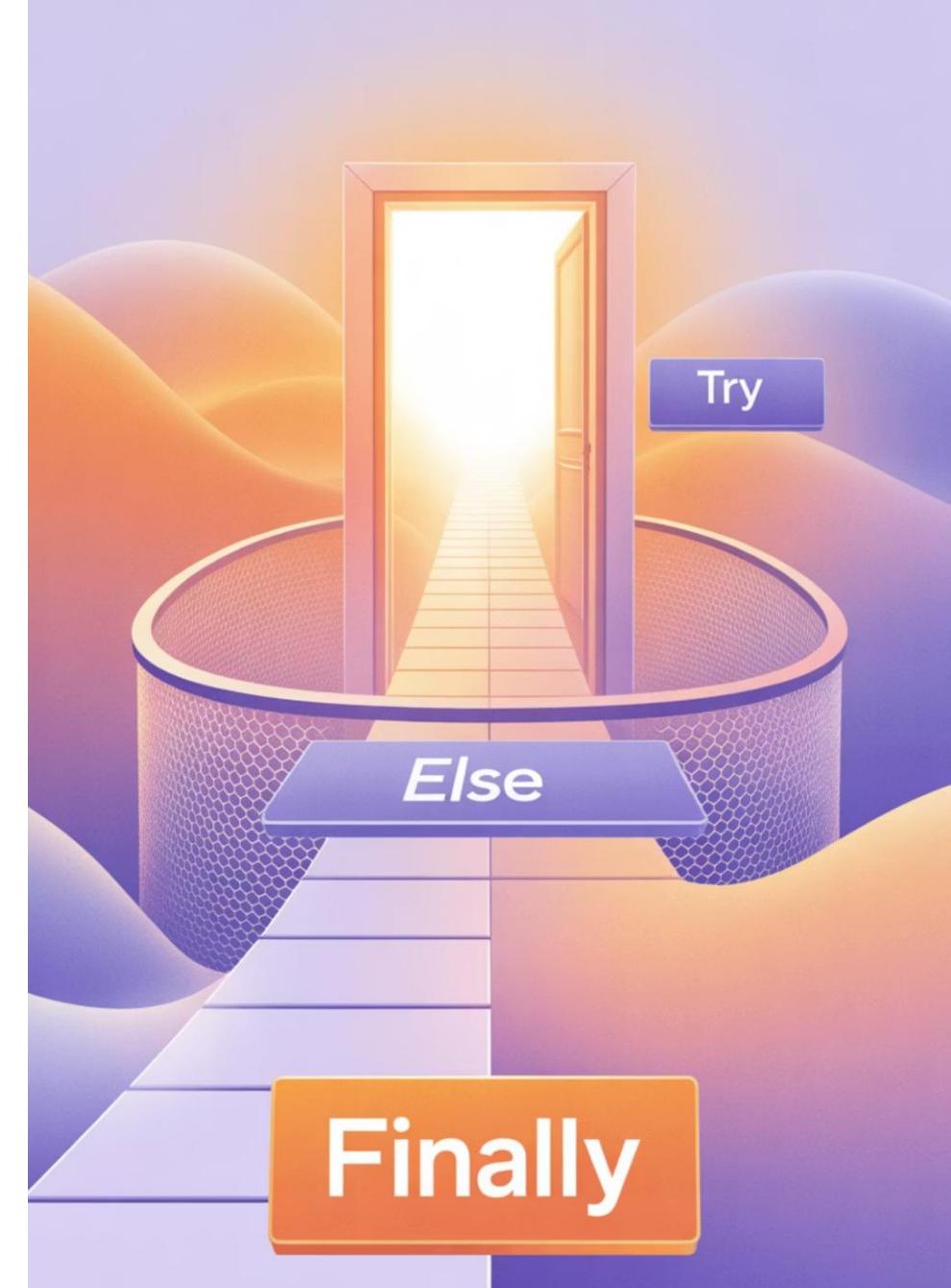
- ឱ្យបង្កើតការងារការបង្កើតរបាយការ
- ឱ្យបង្កើតការងារការបង្កើតរបាយការ
- ឱ្យបង្កើតការងារការបង្កើតរបាយការ

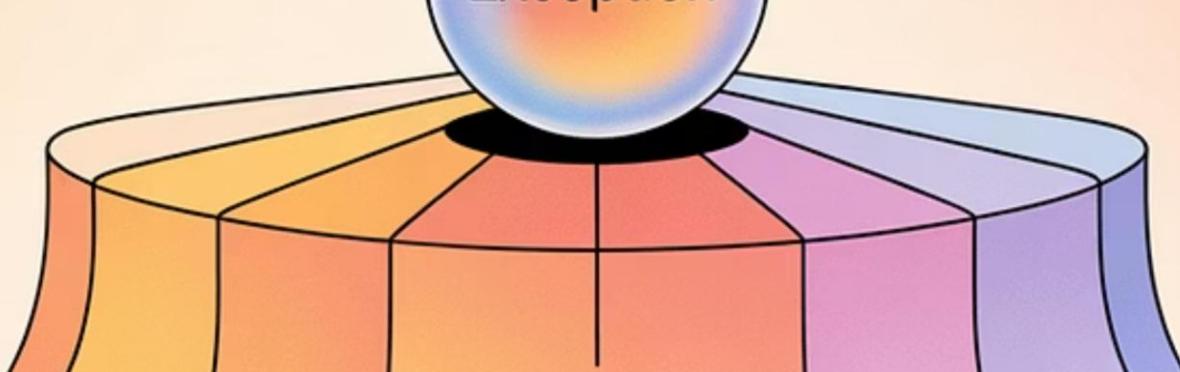
2

Else Block

គឺជាបុលិក else ដែលរាយការណ៍ នៅក្នុងការបង្កើតរបាយការ ឬការងាររបាយការ។

- ឱ្យបង្កើតការងារការបង្កើតរបាយការ
- ឱ្យបង្កើតការងារការបង្កើតរបាយការ
- ឱ្យបង្កើតការងារការបង្កើតរបាយការ





การสร้าง Custom Exception

```
# สร้าง Custom Exception ของเราเอง
class DataValidationError(Exception):
    pass
def validate_age(age):
    if age < 0:
        # ส่ง Exception ที่เราสร้างขึ้นมาเอง
        raise DataValidationError("Age cannot be negative.")
```

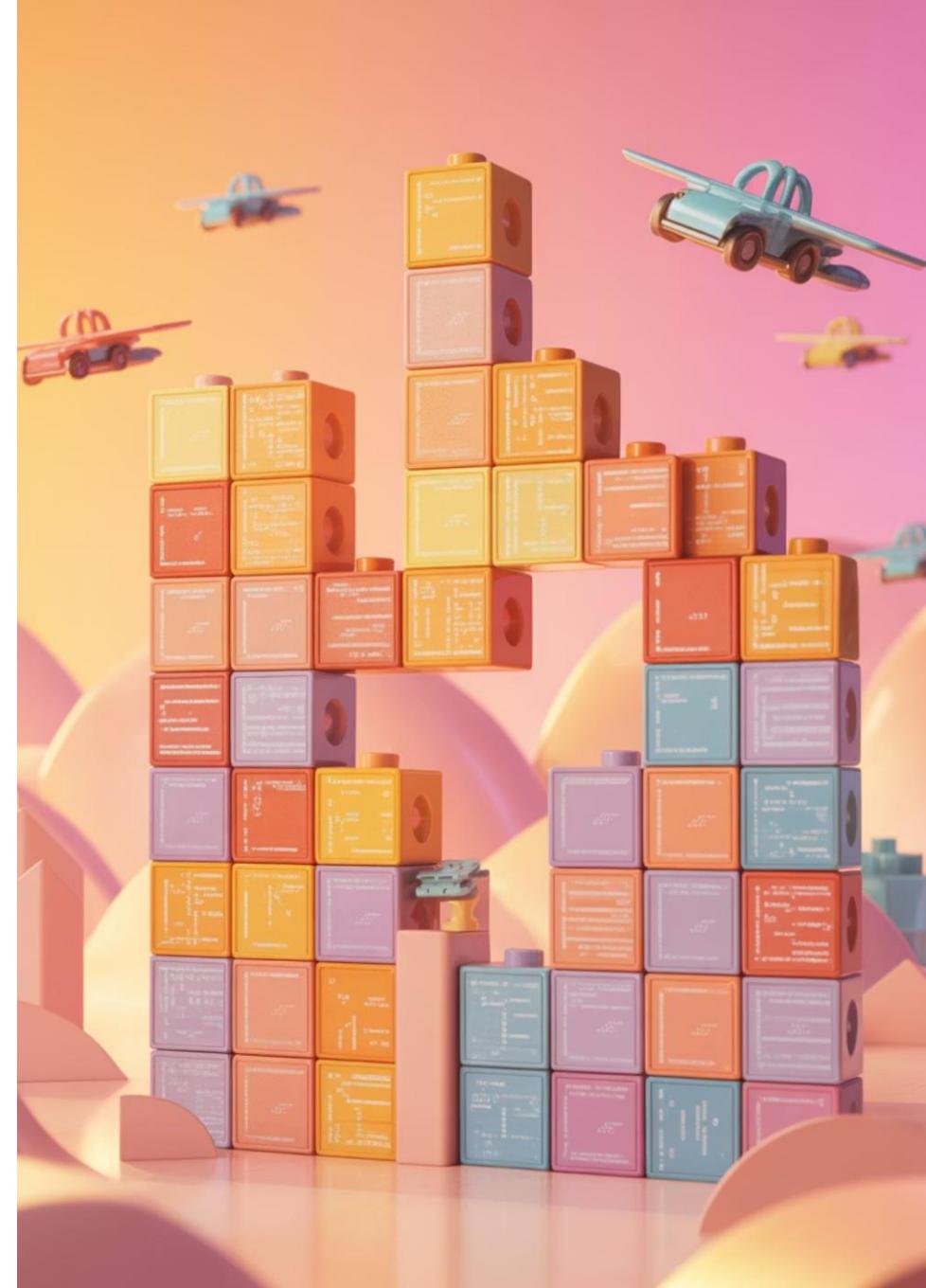
การสร้าง Custom Exception ช่วยให้เราจัดการกับข้อผิดพลาดที่เฉพาะเจาะจงกับโปรแกรมของเรา ทำให้โค้ดสื่อความหมายได้ดีขึ้น และแยกแยะข้อผิดพลาดทางธุรกิจออกจากข้อผิดพลาดทั่วไปได้อย่างชัดเจน

- ▢ **ประโยชน์:** Custom Exception ทำให้เราสามารถจัดกลุ่มข้อผิดพลาดตามประเภทได้ และใช้ข้อมูลที่เฉพาะเจาะจงมากขึ้น

Modular Development

การพัฒนาโค้ดแบบโมดูล

การเขียนคัดแบบโนดลคือการแบ่งโปรแกรมขนาดใหญ่ออกเป็นส่วนเล็กๆ ที่ทำงานเป็นอิสระต่อกัน ทำให้คัดนำกลับมาใช้ใหม่ได้ง่าย ไม่กระทบกัน และสามารถพัฒนาแยกส่วนได้



การแยกโค้ดเป็นโมดูล (Modularization)



db_connector.py

ส่วนที่เกี่ยวกับการเชื่อมต่อฐานข้อมูล การ query และการจัดการ connection pool



data_processor.py

ส่วนที่เกี่ยวกับการประมวลผลข้อมูล การทำความสะอาด และการแปลงข้อมูล



api_handler.py

ส่วนที่จัดการกับ API requests, responses และการพิสูจน์ตัวตน



validators.py

ส่วนที่ตรวจสอบความถูกต้องของข้อมูลต่างๆ ตาม business rules

การแยกคัดตามหน้าที่ทำให้โครงสร้างプロジェクトเป็นระเบียบ หาคัดซ้ำ และสูญเสียในทีมสามารถดำเนินการแยกส่วนได้โดยไม่ซับกัน



Dependency Injection (DI)

แนวคิดของ Dependency Injection

คลาสหรือฟังก์ชัน "ไม่สร้าง" dependency ของตัวเอง แต่จะ "รับเข้ามา" จากภายนอก

- ทดสอบง่ายขึ้นมาก
- ยืดหยุ่นต่อการเปลี่ยนแปลง
- ลด coupling ระหว่างคลาส
- สามารถ mock dependency ได้

ตัวอย่างการใช้งาน

```
# ไม่ดี - สร้าง dependency เอง
class DataProcessor:
    def __init__(self):
        self.db = DatabaseConnection()

# ดี - รับ dependency เข้ามา
class DataProcessor:
    def __init__(self, db_connection):
        self.db = db_connection
```

Unit Testing - พื้นฐานของการทดสอบ

Unit Testing คือการเขียนโค้ดเพื่อทดสอบการทำงานของโค้ดส่วนเล็กที่สุด (Unit) เช่น ฟังก์ชันหรือเมธอด เพื่อให้มั่นใจว่าทำงานได้ถูกต้องตามที่คาดหวัง

ประโยชน์ของ Unit Testing

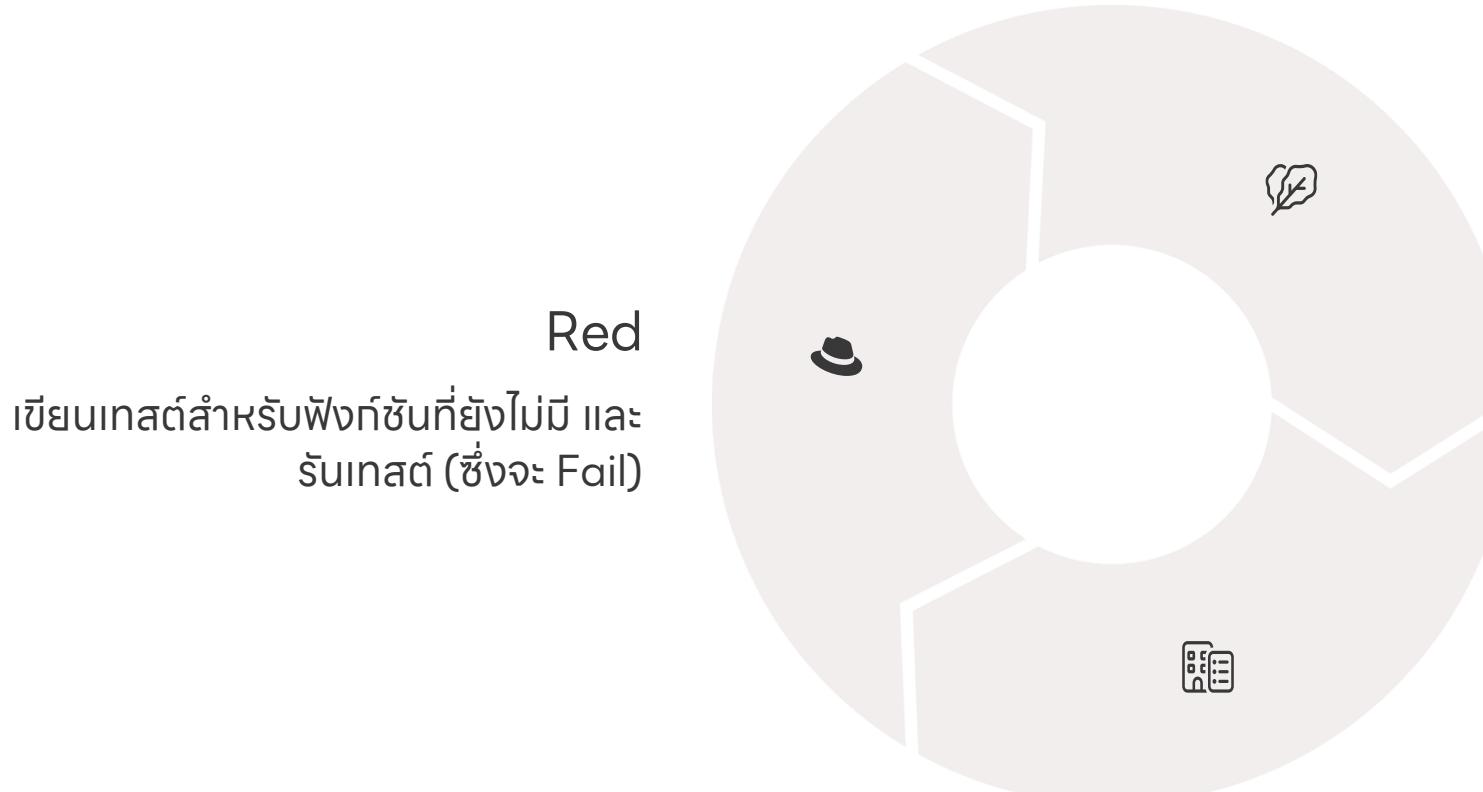
- ตรวจสอบข้อผิดพลาดได้เร็ว
- เป็นเอกสารอธิบายว่าใช้ฟังก์ชัน
- ป้องกัน regression bugs
- ให้ความมั่นใจในการ refactor

หลักการเขียน Test ที่ดี

- ทดสอบ case ปกติและ edge cases
- แต่ละ test ควรทดสอบเพียงสิ่งเดียว
- ชื่อ test ควรอธิบายสิ่งที่ทดสอบ
- Test ต้องรับเร็วและเป็นอิสระ



Test-Driven Development (TDD)



TDD ช่วยให้เราได้โค้ดที่ผ่านการทดสอบและยังช่วยซึ่นนำการออกแบบให้ดีขึ้นด้วยการเขียนเทสต์ก่อนทำให้เราคิดถึง interface และ API ของฟังก์ชันให้ชัดเจน

Green

ເຂົ້າໃຈຕົວຢ່າງທີ່ສຸດເກົ່າທີ່ຈຳເປັນ
ເພື່ອໃຫ້ເກສົງນັ້ນຜ່ານ

Refactor

ปรับปรุงโค้ดจริงให้ดีขึ้นโดยที่เทสต์ยังคงผ่าน

បន្ទាន់នៃ Clean Code

80%

គុណភាពការពេទ្យ

គុណភាពការពេទ្យ
ដែលត្រូវបានបញ្ចប់
ដើម្បីបង្កើតការងារ

60%

ការងារស្ថាប់អាជីវកម្ម

ការងារស្ថាប់អាជីវកម្ម
ដែលត្រូវបានបញ្ចប់
ដើម្បីបង្កើតការងារ

90%

គុណភាពការពេទ្យ

គុណភាពការពេទ្យ
ដែលត្រូវបានបញ្ចប់
ដើម្បីបង្កើតការងារ



គេរែងមីនឹងប៉ូលនៃការបើយន Clean Code

Linters & Formatters

គេរែងមីនឹងយោង pylint, flake8, និង black ដើម្បី
ពារិនិត្យការសរសៃរបស់បាតវិកយាតាសាត់ប៉ូល
និងការសរសៃរបស់បាតវិកយាតាសាត់ប៉ូល។

IDE & Text Editors

គេរែងមីនឹងយោង PyCharm, VSCode ដើម្បី
បង្កើតការសរសៃរបស់បាតវិកយាតាសាត់ប៉ូល
និងការសរសៃរបស់បាតវិកយាតាសាត់ប៉ូល។

Code Reviews

ការ review គឺជាការសរសៃរបស់បាតវិកយាតាសាត់ប៉ូល
និងការសរសៃរបស់បាតវិកយាតាសាត់ប៉ូល។

Common Code Smells ที่ควรหลีกเลี่ยง

Long Methods

ເມືອດທີ່ຢາວເກີນໄປ ຄວຣແຍກເປັນເມືອດຍ່ອຍຖາ

Duplicate Code

Large Classes

คลาสที่มีหน้าที่มากเกินไป ละเมิด Single Responsibility Principle

Feature Envy

ເມືອດກີ່ໃຊ້ຂ້ອມຸລຈາກຄລາສອົນນາກກວ່າ
ຄລາສຕັວອ່ອງ

Dead Code

គឺជាការបង្ហាញពីការរំលែករំលែកដែលមានការចាប់ផ្តើមនៅក្នុងបណ្តុះបណ្តាលទិន្នន័យ។

Magic Numbers

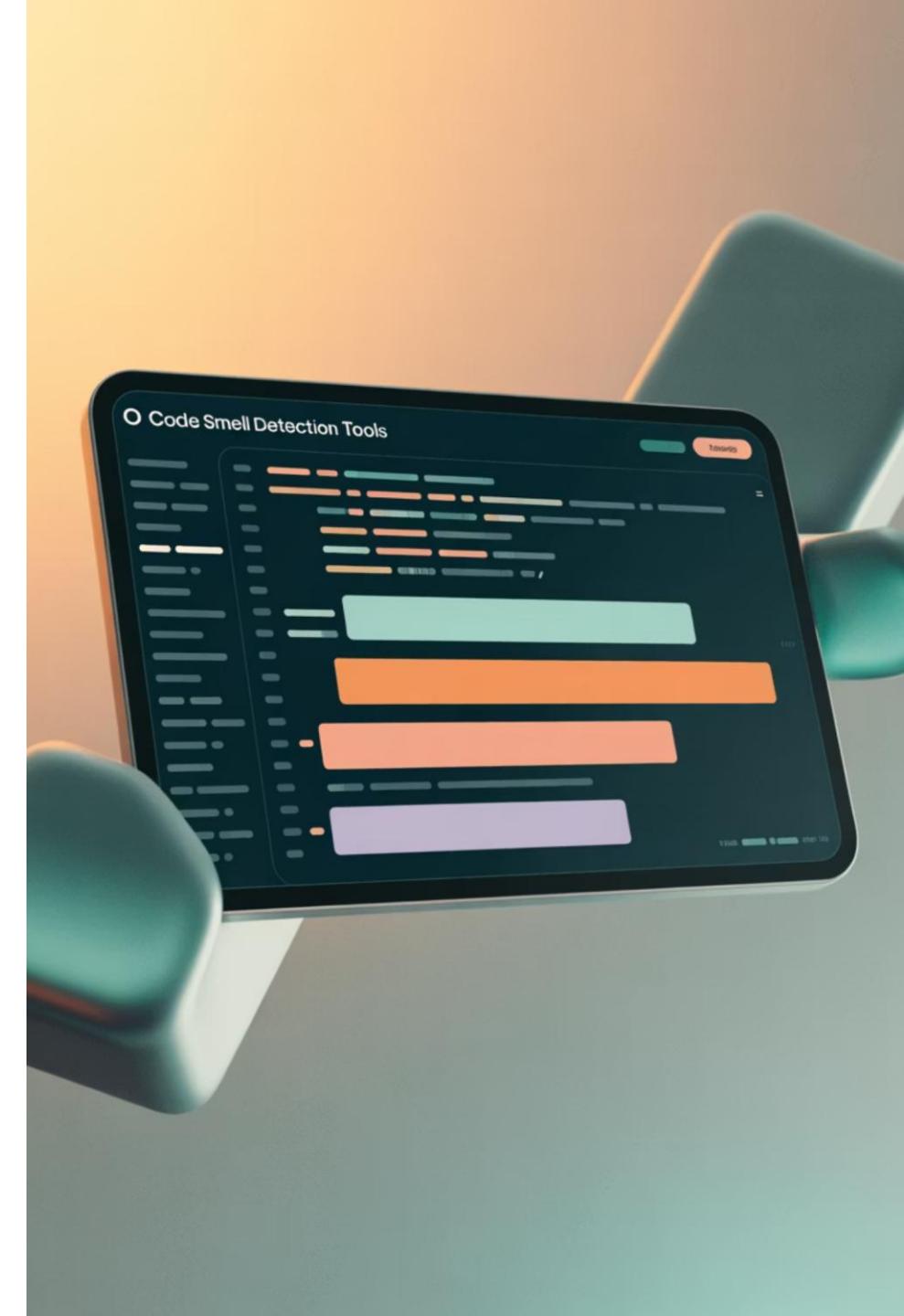
ตัวเลขที่ไม่มีความหมายซัดเจน ควร
กำหนดเป็น constants

God Object

คลาสที่รู้หรือทำทุกอย่างในระบบ ควรแยก ออกเป็นหลายคลาส

Shotgun Surgery

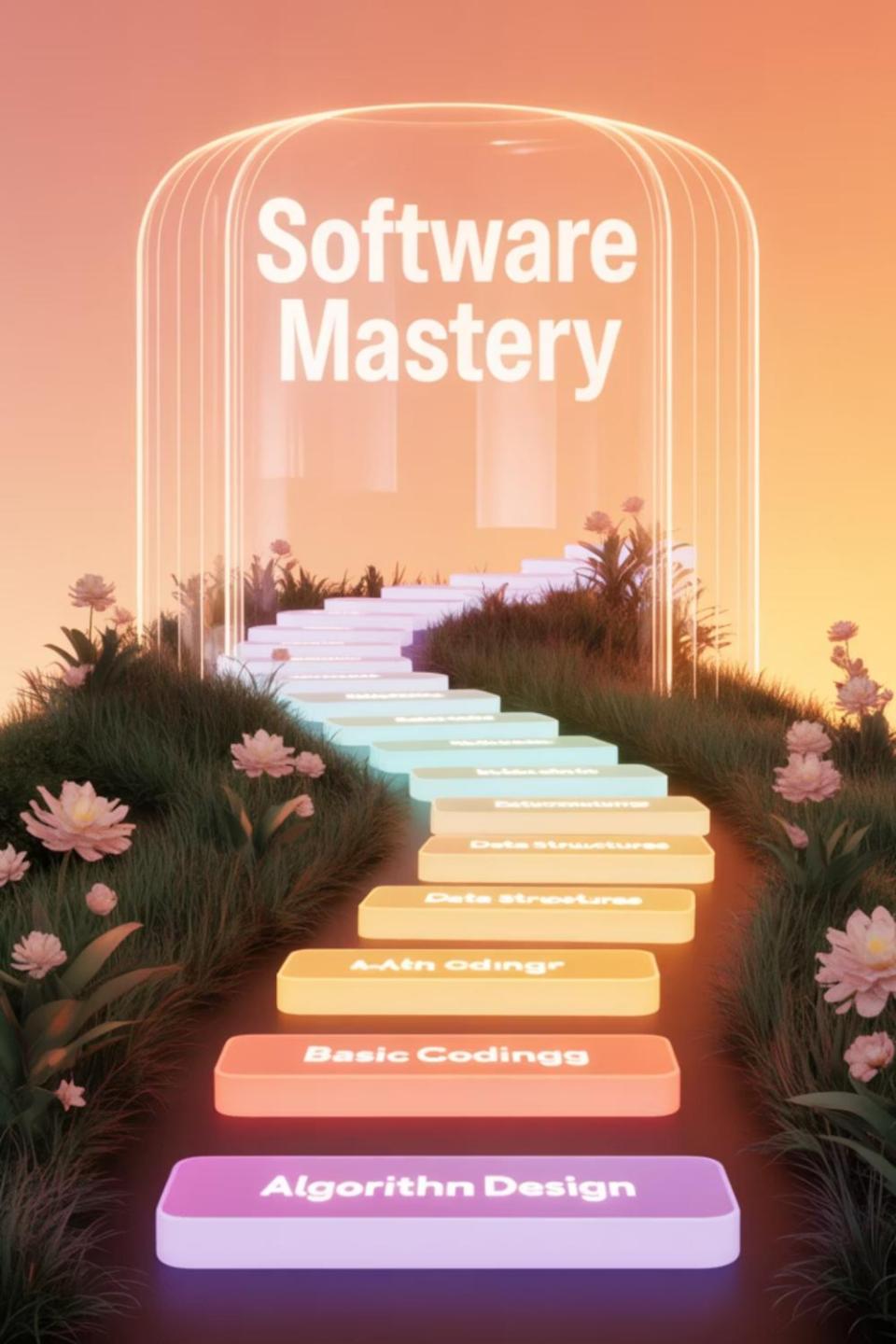
การแก้ไขเล็กน้อยต้องแก้หลายไฟล์ แสดง ว่าโครงสร้างไม่ดี





การนำ Agile ไปประยุกต์ใช้ในโปรเจกต์จริง

- 1 เริ่มจากโปรเจกต์ใหม่
กำหนด coding standards, ตั้งค่า linting tools, และสร้าง project structure ที่เป็นระเบียบ
- 2 Refactor โค้ดเก่าทีละส่วน
ไม่ต้อง refactor ก็งมงດพร้อมกัน เริ่มจากส่วนที่มี impact มากที่สุดหรือ แก้ไขบ่อย
- 3 เขียน Unit Tests
เพิ่ม test coverage ทีละส่วน เริ่มจากฟังก์ชันสำคัญที่มีโอกาสมีข้อผิดพลาดสูง
- 4 ทบทวนและปรับปรุงอย่างต่อเนื่อง
ทำ code review เป็นประจำ วิเคราะห์ metrics และปรับปรุงกระบวนการพัฒนา



สรุปและขั้นตอนต่อไป

เริ่มที่พื้นฐาน

ฝึกการตั้งชื่อที่ดี เขียนฟังก์ชันสั้นๆ และใช้ style guide อย่างสม่ำเสมอ

เรียนรู้การจัดการ Error

เข้าใจและใช้ Exception Handling อย่างมีประสิทธิภาพในโปรเจกต์ของคุณ

พัฒนาแบบ Modular

แบ่งโค้ดเป็นโมดูล เขียน Unit Tests และใช้ TDD ในการพัฒนาฟีเจอร์ใหม่

การเขียนโค้ดแบบมืออาชีพไม่ใช่เรื่องที่ทำได้ในวันเดียว แต่เป็นกักสะสมพัฒนาขึ้นมาจากการฝึกฝนอย่างต่อเนื่อง เริ่มต้นจากโปรเจกต์เล็กๆ และค่อยๆ ประยุกต์ใช้หลักการเหล่านี้ในงานของคุณ

"การเขียนโค้ดที่ดีไม่ใช่แค่การทำให้โปรแกรมทำงาน แต่เป็นการสร้างงานศิลปะที่คนอื่นสามารถเข้าใจและต่อยอดได้"



การอุปกรณ์ระบบวิเคราะห์ ข้อมูลที่ซับซ้อน

การพัฒนาระบบวิเคราะห์ข้อมูลที่มีประสิทธิภาพและปรับขยายได้ สำหรับ
นักพัฒนาซอฟต์แวร์ วิศวกรข้อมูล และสถาปนิกระบบ

ផលិតផលទីការណ៍ទូទៅ

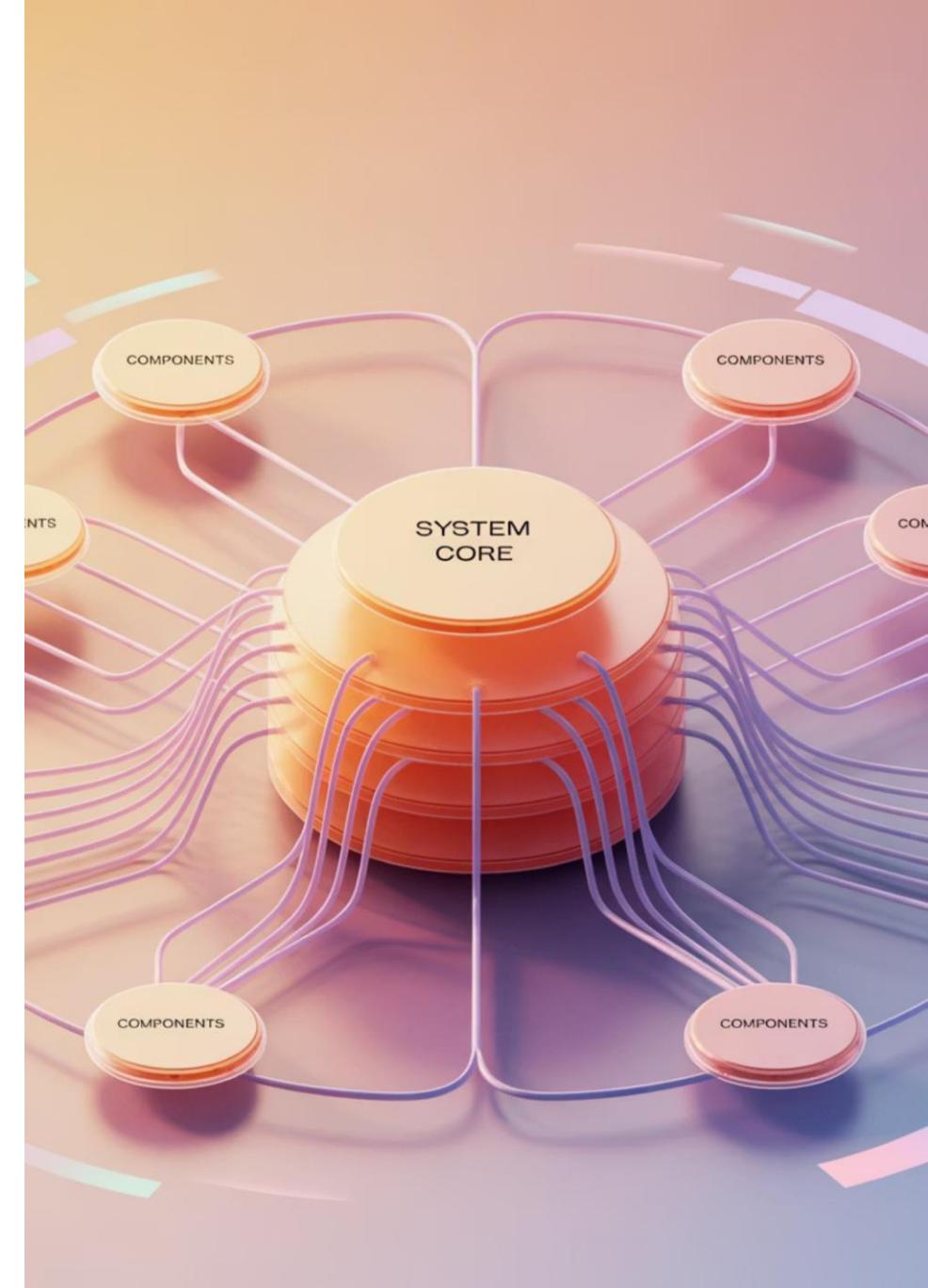
1 សរាងស្ថាប្រព័ន្ធឌីជីថាមពលទីការណ៍ទូទៅ ដែលបានរចនាបានដើម្បីបង្កើតរឹងរាល់ការងារ។

2 ការបង្កើតរឹងរាល់ការងារ ដែលបានរចនាបានដើម្បីបង្កើតរឹងរាល់ការងារ។

3 ការបង្កើតរឹងរាល់ការងារ ដែលបានរចនាបានដើម្បីបង្កើតរឹងរាល់ការងារ។

សំគាល់ 1

ការរចនាប្រព័ន្ធទិន្នន័យ ដើម្បីបង្កើតការងារ



ຮັກການສໍາຄັญຂອງສາປະລິກອມທີ່ດີ

การวางแผนสร้างที่ดินเจน

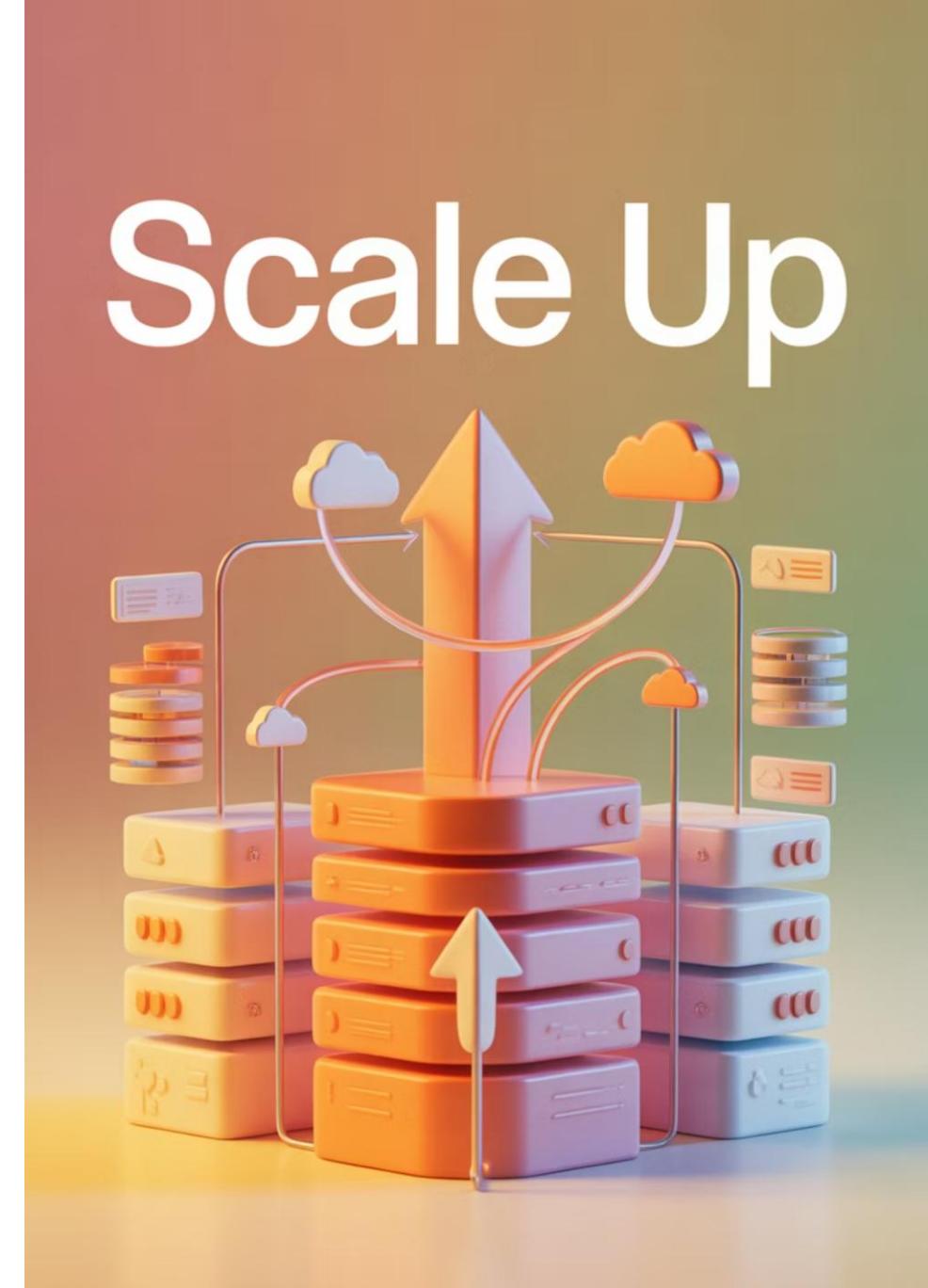
การออกแบบระบบที่ดีต้องเริ่มต้นจากการวางแผนสร้างที่มีหลักการชัดเจน แยกหน้าที่ของแต่ละส่วนอย่างเด็ดขาด

- ลดความซับซ้อนในการพัฒนา
 - เพิ่มความยืดหยุ่นในการดูแลรักษา
 - ง่ายต่อการทดสอบและแก้ไข

การเตรียมพร้อมสำหรับอนาคต

ระบบที่ออกแบบมาอย่างดีจะสามารถรองรับการเติบโตและการเปลี่ยนแปลงของความต้องการในอนาคตได้

- รองรับการเพิ่มฟีเจอร์ใหม่
 - ปรับเปลี่ยนโนเดลได้ง่าย
 - ขยายขนาดระบบได้ตามต้องการ



Layered Architecture: ការបែងចាន់ហើយ

Data Layer

មូលដ្ឋានដែលរួមចំណាំពីការបញ្ចូនធមូលដ្ឋាន, ការបញ្ចូនធមូលដ្ឋាន, និងការបញ្ចូនធមូលដ្ឋាន។

- ការបញ្ចូនធមូលដ្ឋាន
- ការបញ្ចូនធមូលដ្ឋាន
- ការបញ្ចូនធមូលដ្ឋាន

Modeling Layer

មូលដ្ឋានដែលរួមចំណាំពីការបញ្ចូនធមូលដ្ឋាន, ការបញ្ចូនធមូលដ្ឋាន, និងការបញ្ចូនធមូលដ្ឋាន។

- ការបញ្ចូនធមូលដ្ឋាន
- ការបញ្ចូនធមូលដ្ឋាន
- ការបញ្ចូនធមូលដ្ឋាន

Processing Layer

មូលដ្ឋានដែលរួមចំណាំពីការបញ្ចូនធមូលដ្ឋាន, ការបញ្ចូនធមូលដ្ឋាន, និងការបញ្ចូនធមូលដ្ឋាន។

- ការបញ្ចូនធមូលដ្ឋាន
- ការបញ្ចូនធមូលដ្ឋាន
- ការបញ្ចូនធមូលដ្ឋាន

Application/API Layer

មូលដ្ឋានដែលរួមចំណាំពីការបញ្ចូនធមូលដ្ឋាន, ការបញ្ចូនធមូលដ្ឋាន, និងការបញ្ចូនធមូលដ្ឋាន។

- ការបញ្ចូនធមូលដ្ឋាន
- ការបញ្ចូនធមូលដ្ឋាន
- ការបញ្ចូនធមូលដ្ឋាន

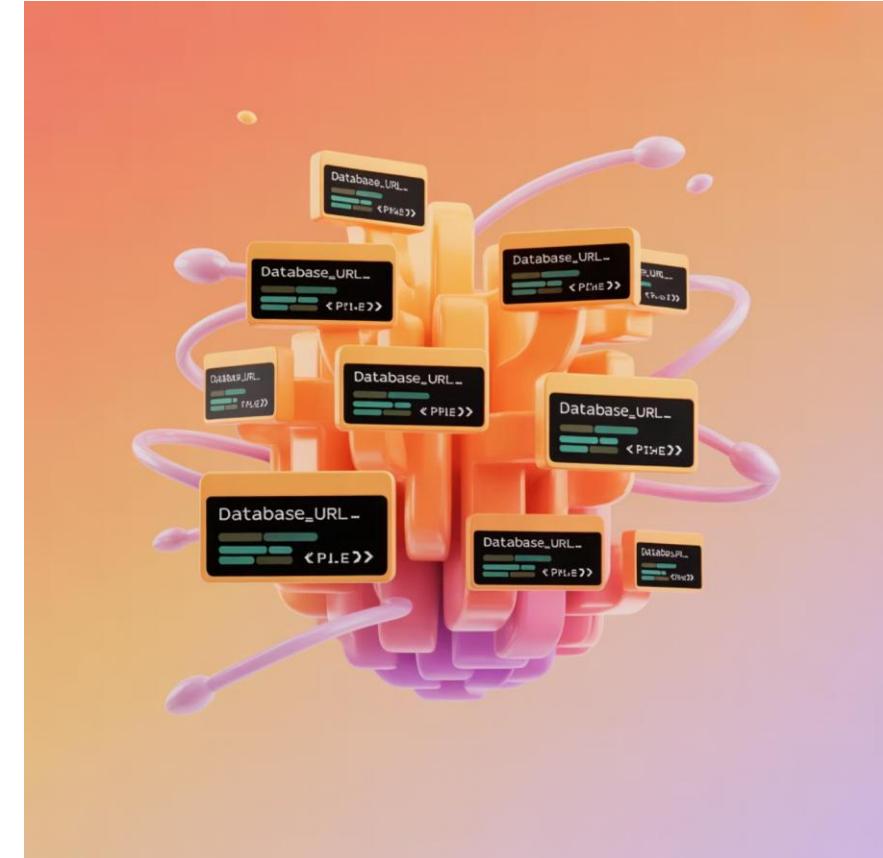
ការបង្កើត Configuration Files

ព្រមទាំងការបង្កើត Configuration Files

ការបង្កើតការងារគឺជាផ្លូវការបង្កើតការងារដែលត្រូវបានបង្កើតឡើង។ ការបង្កើតការងារគឺជាផ្លូវការបង្កើតការងារដែលត្រូវបានបង្កើតឡើង។

- ការបង្កើតការងារគឺជាផ្លូវការបង្កើតការងារដែលត្រូវបានបង្កើតឡើង។
- ការបង្កើតការងារគឺជាផ្លូវការបង្កើតការងារដែលត្រូវបានបង្កើតឡើង។
- ការបង្កើតការងារគឺជាផ្លូវការបង្កើតការងារដែលត្រូវបានបង្កើតឡើង។
- ការបង្កើតការងារគឺជាផ្លូវការបង្កើតការងារដែលត្រូវបានបង្កើតឡើង។

រូបរាងការងារ: YAML និង JSON គឺជាហ៍ប្រព័ន្ធឌីជីថលដែលបានបង្កើតឡើង។



Object-Oriented Programming II at Design Patterns



การสร้างคลาสหลัก

ออกแบบคลาสที่มีหน้าที่ชัดเจนและแยกส่วนกันอย่างเด็ดขาด เช่น DataLoader, DataPreprocessor, ModelTrainer เพื่อความง่ายในการพัฒนาและดูแลรักษา



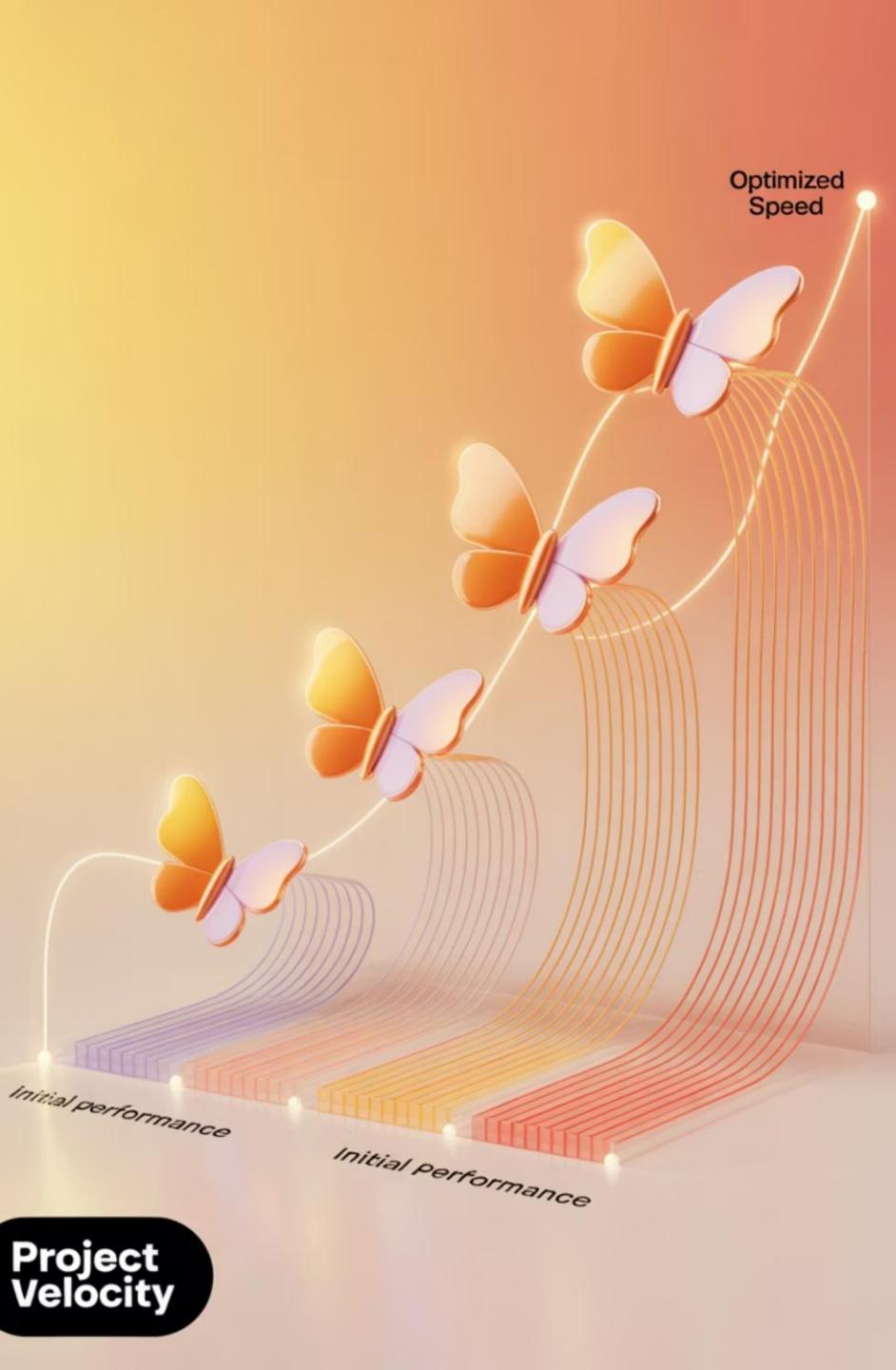
Factory Pattern

สร้างฟังก์ชัน `create_model(config)` ที่สามารถ
สร้างโมเดลต่างๆ เช่น RandomForest, LightGBM
ได้ตามการกำหนดค่าใน `config` ทำให้การทดลองง่าย
โดยเป็นเรื่องง่าย



Strategy Pattern

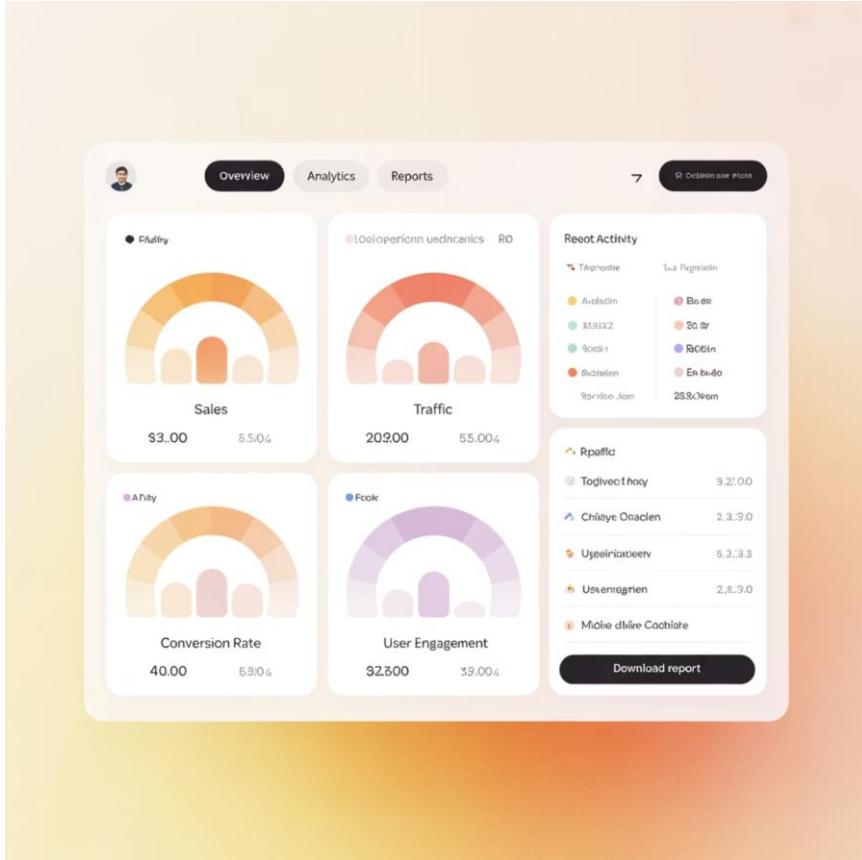
ใช้ในคลาส DataPreprocessor เพื่อสลับวิธีการจัดการ Missing Value หรือการทำ Scaling ได้อย่างอิสระ ทำให้มีความยืดหยุ่นในการเลือกใช้วัลอกอริทึม



ส่วนที่ 2

การปรับปรุงประสิทธิภาพระบบ

ក្នុងការងារ: វិធានកំណើនសំខាន់



Measure First, Optimize Second

ការប្រើប្រាស់តម្លៃការងារដែលត្រួតពិនិត្យរបស់អ្នកជាជាតិ និងការងារទូទៅរបស់អ្នកជាជាតិ។

- ឱ្យក្រុមហ៊ុនរបស់អ្នកជាជាតិ ឱ្យរបាយការណ៍របស់អ្នកជាជាតិ ដើម្បីរកចំណាំការងារ។
- ឱ្យក្រុមហ៊ុនរបស់អ្នកជាជាតិ ឱ្យរបាយការណ៍របស់អ្នកជាជាតិ ដើម្បីរកចំណាំការងារ។
- ឱ្យក្រុមហ៊ុនរបស់អ្នកជាជាតិ ឱ្យរបាយការណ៍របស់អ្នកជាជាតិ ដើម្បីរកចំណាំការងារ។
- ឱ្យក្រុមហ៊ុនរបស់អ្នកជាជាតិ ឱ្យរបាយការណ៍របស់អ្នកជាជាតិ ដើម្បីរកចំណាំការងារ។

"Premature optimization is the root of all evil" - Donald Knuth

เครื่องมือ Profiler สำหรับ Python

1

cProfile - Built-in Profiler

เครื่องมือ Profiling ที่มาพร้อมกับ Python ใช้สำหรับวิเคราะห์การทำงานของโปรแกรม

- ระบุฟังก์ชันที่ต้องการยืดเวลาอย่างสูง
- คำนวณเวลารวมของแต่ละฟังก์ชัน
- เผนาะสำหรับการหา Bottleneck ในภาพใหญ่
- ใช้งานง่าย: `python -m cProfile script.py`

2

line_profiler - Line-by-Line Analysis

ไลบรารีภาษา Python ที่วิเคราะห์โค้ดแบบตัวต่อตัวในฟังก์ชัน

- ติดตั้ง: `pip install line_profiler`
- วิเคราะห์แบบ Line-by-Line ภายในฟังก์ชัน
- ระบุบรรทัดที่ใช้เวลาทำงานมากที่สุด
- ใช้ `@profile decorator` เพื่อระบุฟังก์ชันที่ต้องการวิเคราะห์



เทคโนโลยีการเพิ่มประสิทธิภาพ



sez Bottleneck

ใช้ **จังหวะ**เพื่อหาส่วนที่ใช้เวลาประมาณผลนานที่สุด เป็นขั้นตอนแรกที่สำคัญที่สุด



Vectorization

เปลี่ยนการใช้ Loop ธรรมดามาเป็นการใช้ NumPy/Pandas Operations ที่เร็วกว่ามาก



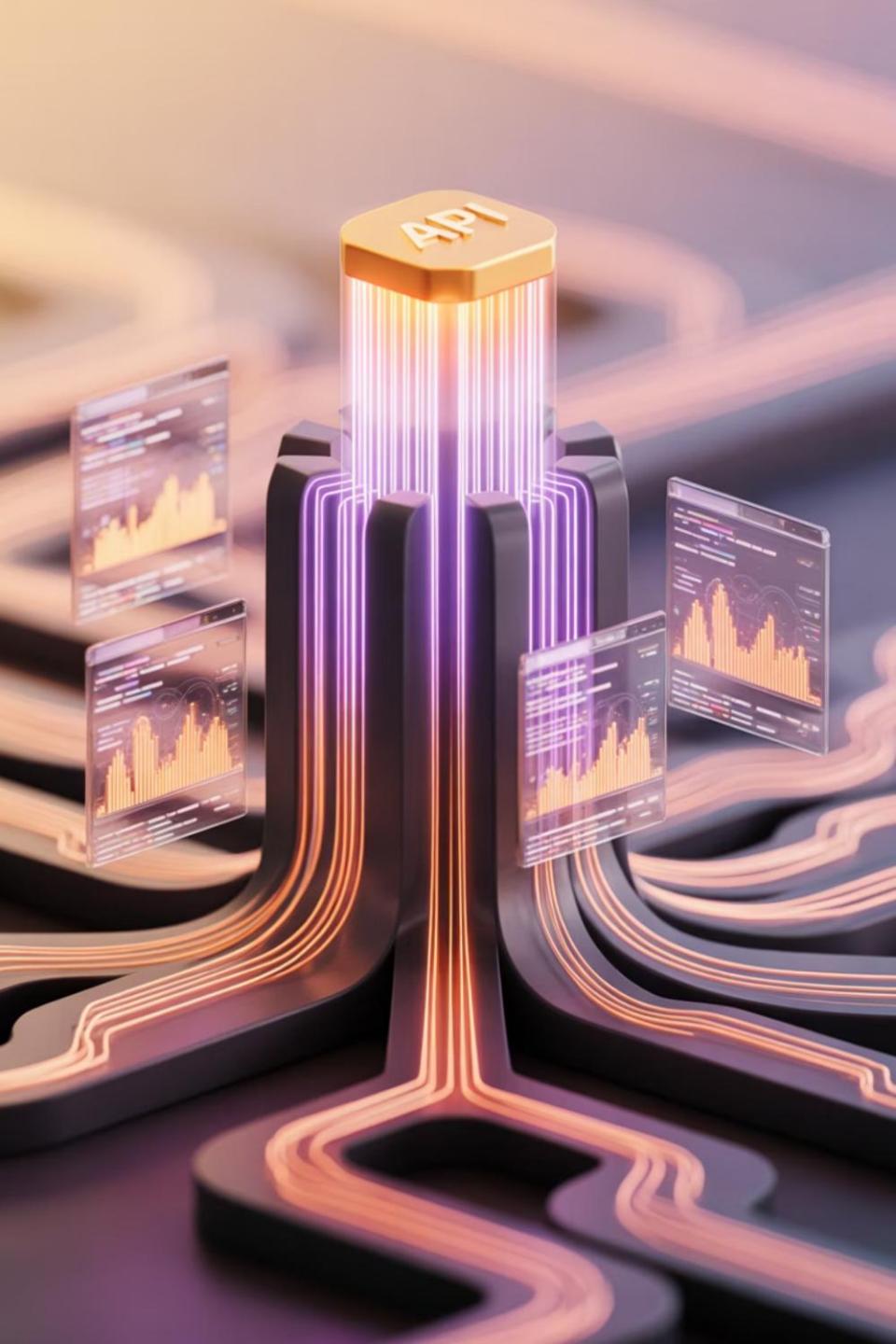
Caching

ใช้ `lru_cache` หรือเทคนิค `Memoization` เพื่อเก็บผลลัพธ์ของการคำนวณที่ซ้ำ



Algorithm Optimization

ปรับปรุงอัลกอริทึมหรือโครงสร้างข้อมูลให้มีประสิทธิภาพดีขึ้น



ส่วนที่ 3

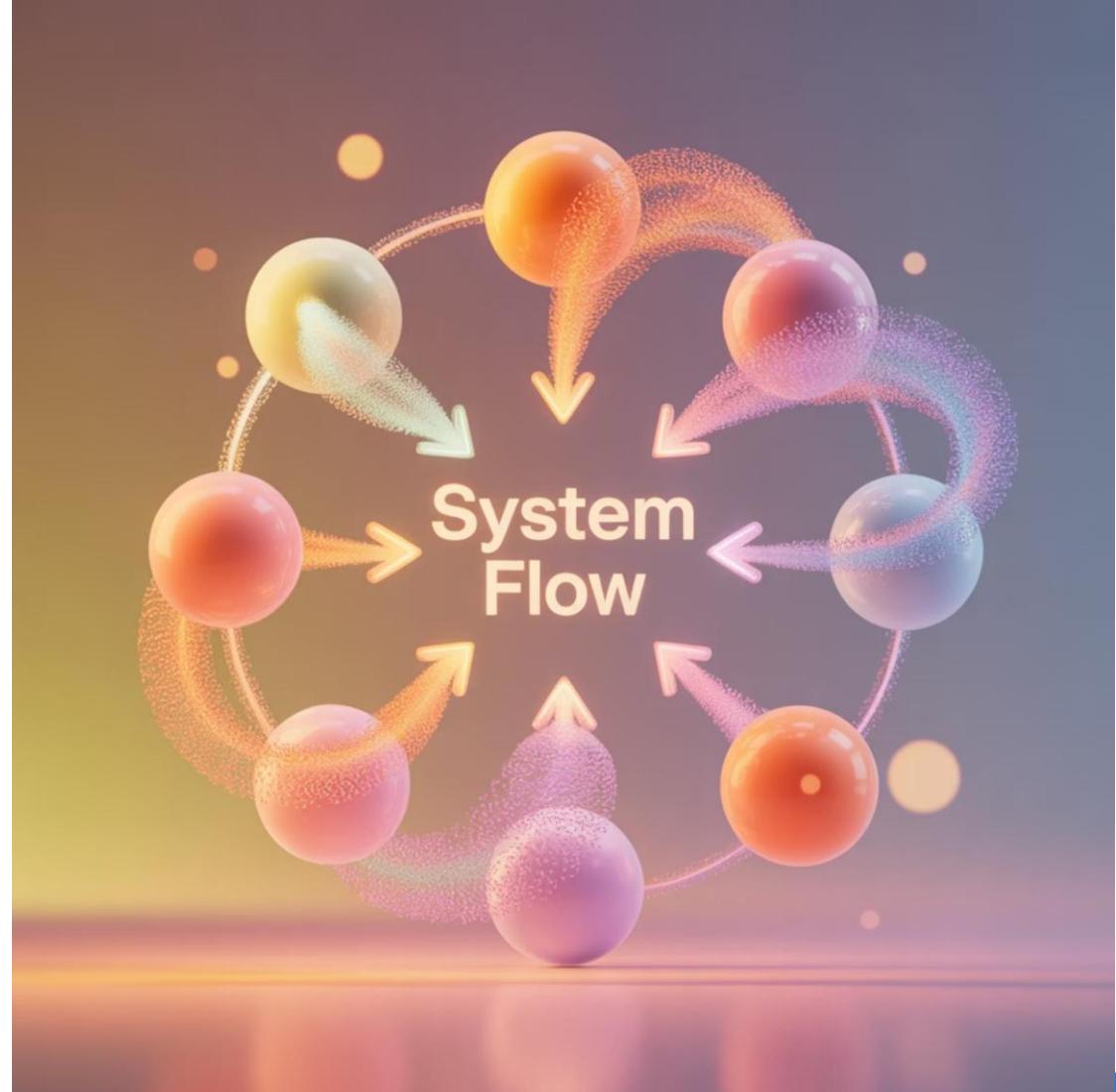
การพัฒนา API สำหรับ Data Science

កាំណីតែង API សំខាន់ Data Science?

ការចែំនៅពេលបច្ចុប្បន្ន

API កាំងការណ៍នឹងការងារប៉ូល Machine Learning នៃយុទ្ធសាស្ត្រ។ បានបង្កើតឡើងដើម្បីការងារប៉ូល ដើម្បីការងារទូទៅ និងការងារទូទៅ។

- យកសំណើនឹង Business Logic នូវការងារប៉ូល
- ទទួលបានការងារប៉ូលដើម្បីការងារទូទៅ
- ងាយស្រួលការងារប៉ូល





FastAPI: Framework สำหรับ Data Science API

ประสิทธิภาพสูง

FastAPI เป็น Framework ที่เร็วที่สุดสำหรับ Python เกียบเคียงได้กับ NodeJS และ Go

พร้อมใช้งาน

มาพร้อมกับ Documentation อัตโนมัติ, Data Validation และ Type Hints

ง่ายต่อการพัฒนา

Syntax ที่เรียบง่าย ใช้เวลาเรียนรู้น้อย เหมาะสำหรับ Data Scientists

รองรับ Async/Await

สามารถจัดการ Request พร้อมกันได้หลายตัว อันเพิ่มประสิทธิภาพของ API

គម្រោងសរាប់បង្កើតផ្តល់នូវ API ដោយប្រើប្រាស់ FastAPI

```
from fastapi import FastAPI
from pydantic import BaseModel
import joblib
# 1. ឡើងការណែនាំនៃកម្រិតបច្ចុប្បន្ន
model = joblib.load('my_model.pkl')
app = FastAPI()
# 2. ការណែនាំកម្រិតបច្ចុប្បន្ន
Inputclass InputData(BaseModel):
    feature1: float
    feature2: int
# 3. ការណែនាំកម្រិតបច្ចុប្បន្ន
@app.post("/predict")
def predict(data: InputData):
    # ពន្លាឯការណែនាំនៃកម្រិតបច្ចុប្បន្ន
    prediction = model.predict(...)
    return {"prediction": prediction.tolist()}
```

គម្រោងនេះបង្កើតផ្តល់នូវ API ដោយប្រើប្រាស់ FastAPI ដើម្បីស្វែងរកសម្រាប់ការងារគិតថ្លែងរបស់អ្នកស្រួលនៃវិទ្យាភាសាអេក្រង់។

ប៉ានពុនការផែនា API

01

ໂខែដូមែល

ໂខែដូមែល Machine Learning ដែលត្រូវបានកំណត់ឡើងដោយ joblib ឬ pickle

03

សរាង Endpoint

ដោយបង្កើតជាផ្លូវការណ៍ និងវិទ្យាបណ្តុះបណ្តាល សាខាអាមេរិក

02

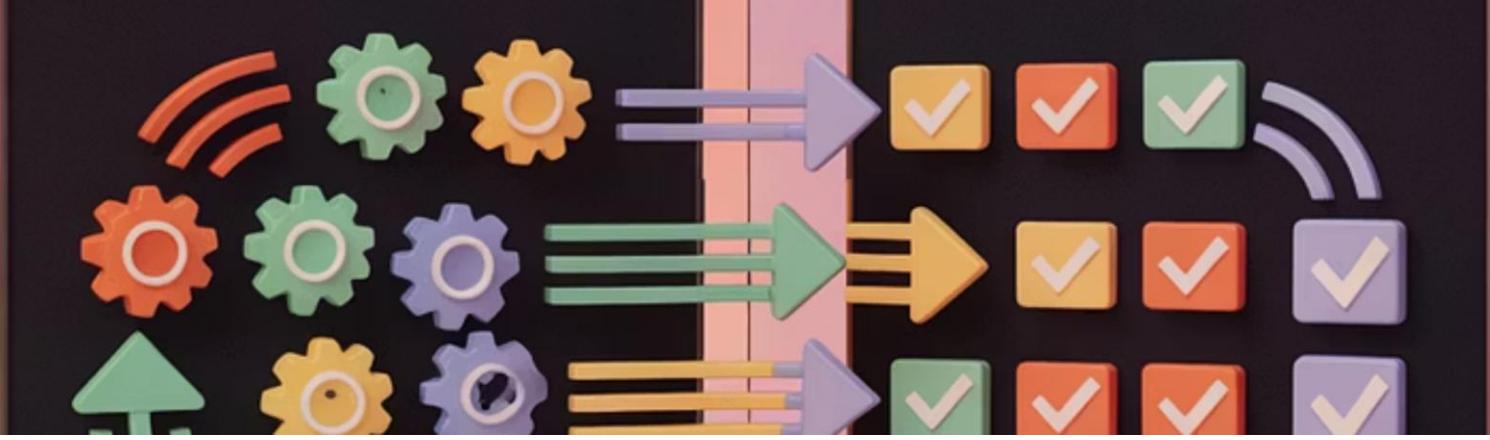
ការណែនាំ Data Schema

ឱ្យ Pydantic ដោះស្រាយការណែនាំរូបរាង និងរាយការណែនាំ Data Schema ដែលត្រូវបានផ្តល់ទៅ API របៀបមួយ

04

ការណែនាំនិង Deploy

ការណែនាំនិង Deploy API ដោយ FastAPI Documentation និង Docker ឬ Server ឬ Cloud



Asynchronous Programming ใน API

ปัญหาของ Synchronous API

API แบบ Synchronous จะจัดการ Request ทีละอันแบบเรียงลำดับ ทำให้เมื่อมี Request ที่ใช้เวลานาน Request อื่นๆ ต้องรอ

- จัดการได้ทีละ Request
- Request อื่นต้องรอให้เสร็จก่อน
- ประสิทธิภาพต่ำเมื่อมี I/O Operations

ข้อดีของ Asynchronous API

API แบบ Async สามารถจัดการหลายๆ Request พร้อมกัน โดยเฉพาะงาน I/O ที่ต้องรอข้อมูลจากภายนอก

- จัดการหลาย Request พร้อมกัน
- ไม่ block เมื่อรอ I/O Operations
- ประสิทธิภาพสูงขึ้นอย่างมาก

ការໃច្ចិញនា Async/Await ໃន FastAPI

```
@app.post("/predict-async")
async def predict_async(data: InputData):
    # សំរាប់រាយការណ៍ទីតាំងទូទៅ I/O ដើម្បី Database Query
    user_data = await get_user_from_db(data.user_id)
    # សំរាប់រាយការណ៍ទីតាំងទូទៅ External API
    external_features = await call_external_api(data)
    # រួមទាំងទូទៅនូវទិន្នន័យ
    features = combine_features(user_data, external_features, data)
    prediction = model.predict(features)
    return {"prediction": prediction.tolist()}
```

តัวอย่างការໃច្ចិញនា Async/Await ໃនរាយការណ៍ទីតាំងទូទៅទិន្នន័យទៅបានពី Database ឬ External API ក្នុងការរាយការណ៍

Best Practices สำหรับ Production API

Error Handling

จัดการข้อผิดพลาดอย่างครบถ้วน ส่งข้อความ Error ที่เป็นประโยชน์กลับไปให้ Client

- ใช้ `HTTPException` สำหรับ Error Response
 - กำหนด Status Code ที่เหมาะสม
 - Log Error สำหรับการ Debug

Data Validation

ใช้ Pydantic Models เพื่อตรวจสอบความถูกต้องของข้อมูลที่รับเข้ามา

- กำหนด Type และ Constraint ที่ชัดเจน
 - ใช้ Field Validators เมื่อจำเป็น
 - ส่ง Clear Error Message เมื่อ Validation ล้มเหลว

Performance Monitoring

ติดตาม Performance และ Resource Usage ของ API อย่างสม่ำเสมอ

- Log Response Time ຂອງໄຕລະ Endpoint
 - Monitor Memory ແລະ CPU Usage
 - ຕັ້ງ Alert ເນື່ອ Performance ອານຸ

Security

ป้องกัน API ด้วยมาตรการความปลอดภัยที่เหมาะสม

- ใช้ HTTPS สำหรับการสื่อสารกับหน้า
 - ใช้ API Keys หรือ JWT สำหรับ Authentication
 - จำกัด Rate Limiting เพื่อป้องกัน Abuse



สรุปและขั้นตอนดัดแปลง

สถาปัตยกรรมที่ดี

เริ่มต้นด้วยการออกแบบ Layered Architecture ที่แยกหน้าที่ชัดเจน ใช้ OOP และ Design Patterns อย่างเหมาะสม

ประสิทธิภาพสูง

ใช้เครื่องมือ Profiling เพื่อระบุและแก้ไขปัญหา Performance ตรงจุด วัดผลก่อนและ

API พร้อมใช้งาน

พัฒนา REST API ด้วย FastAPI พร้อม Async Support สำหรับประสิทธิภาพสูง และรองรับผู้ใช้งานจำนวนมาก

การเรียนรู้ต่อไป

- ทดลองสร้าง API จริงกับโมเดลของตัวเอง
- เรียนรู้การ Deploy API 用 Docker และ Cloud Platforms
- ศึกษา Advanced Topics: Caching, Load Balancing, Microservices