กำหนดส่ง

Lab	
HW	
Until	

กระบวนวิชา 204111

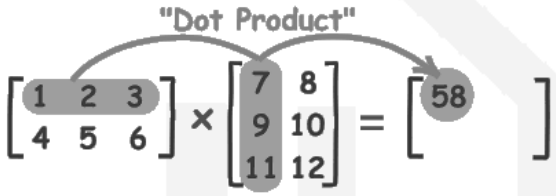
การบ้านปฏิบัติการ 13

n-dimensional Lists and Nested Collections (20 คะแนน)

ข้อกำหนด

การเรียกใช้ฟังก์ชันเพื่อการทดสอบ ต้องอยู่ภายใต้เงื่อนไข `if __name__ == '__main__' :` เพื่อให้สามารถ `import` ไปเรียกใช้งานจาก Script อื่น ๆ ได้อย่างเป็นมาตรฐาน

1) **4 คะแนน** (`Lab13_1_5XXXXXXXX.py`) **[Attachment]** ให้เขียนฟังก์ชัน *`matrix_mult`*`(`$m1$`: list[list[int]], `$m2$`: list[list[int]]) -> list[list[int]]` เพื่อคืนค่าผลคูณของเมทริกซ์ $m1$ และ เมทริกซ์ $m2$ (wikipedia: https://goo.gl/S0DDZv) ทั้งนี้หากไม่สามารถหาผลคูณได้ให้คืนค่า None



Hint: ฟังก์ชันทำงานแบบ Non-destructive

Input	**Output**

`[[1, 2, 3],`
` [4, 5, 6]]`
`[[7, 8],`
` [9, 10],`
` [11,12]]`
	`[[58, 64],`
	` [139, 154]]`

`[[1, 2, 3],`
` [4, 5, 6]]`
`[[7, 8, 5, 9, 3],`
` [9, 10, -3, 7, 13],`
` [11, 12, 6, 2, 9]]`
	`[[58, 64, 17, 29, 56],`
	` [139, 154, 41, 83, 131]]`

2) **4 คะแนน** (`Lab13_2_6XXXXXXXX.py`) **[Attachment]** ให้เขียนฟังก์ชัน *`square_matrix`*`(`*`list_x`*`: list[list[int]])` เพื่อทำให้ List 2 มิติ *list_x* ที่มีสมาชิกเป็นจำนวนเต็มกลายเป็น matrix จัตุรัสโดยเติม `0` เพื่อให้มีขนาด row และ column เท่ากัน โดยจะต้องคงทุก element ใน List เดิมไว้ และจำนวน `0` ที่เติมต้องเป็นจำนวนที่น้อยที่สุดที่เป็นไปได้ ทั้งนี้กำหนดให้ฟังก์ชันทำงานแบบ Destructive และแต่ละ row หรือ column จะต้องไม่เป็น alias ซึ่งกันและกัน

Input	Output

`[[2, 3, 4],`
` [1, 2, 3]]`
	`[[2, 3, 4],`
	` [1, 2, 3],`
	` [0, 0, 0]]`

`[[1, 2],`
` [1, 2, 3],`
` [1, 2],`
` [1, 2],`
` [1]]`
	`[[1, 2, 0, 0, 0],`
	` [1, 2, 3, 0, 0],`
	` [1, 2, 0, 0, 0],`
	` [1, 2, 0, 0, 0],`
	` [1, 0, 0, 0, 0]]`

3) **4 คะแนน** (HW13_1_6XXXXXXXX.py) บริษัท Autopilot ต้องการทดสอบโดรนช้อปปิ้ง โดยผู้ซื้อจะทำการเลือกซื้อสินค้าต่างๆ จาก List ของข้อมูลสินค้าภายในร้าน การซื้อสินค้าจะต้องซื้อแบบยกชิ้นไม่มีการแบ่งขายและซื้อได้มากที่สุด 1 ชิ้นต่อชนิด ด้วยงบประมาณรวม (budget) ที่แทนด้วยตัวแปร *budget* มีชนิดข้อมูลเป็น **float** ซึ่งมีค่า 0 < *budget* <= 10,000 และโดรนช้อปปิ้งจะต้องนำสินค้าที่ซื้อทั้งหมดนำไปส่งให้ที่บ้านของลูกค้า ดังนั้นสินค้าที่ซื้อทั้งหมดจะต้องมีน้ำหนักไม่เกินค่าที่กำหนดไว้ (allowed weight) ซึ่งแทนด้วยค่า *allowed_w* ที่มีชนิดข้อมูลเป็น **float** โดยที่ 5 <= *allowed_w* <= 1000)

ข้อมูลสินค้าแต่ละชิ้นในร้านค้าแทนด้วย *p_list* ซึ่งอยู่ในรูปแบบของ Dictionary มีความยาวไม่เกิน 20 รายการ ประกอบด้วย

key แทนชื่อสินค้าที่เป็น string

value แทนน้ำหนักของสินค้า (w) และราคาสินค้า (p) เป็น float เก็บอยู่ในรูปแบบของ Tuple

โดยจะไม่มีสินค้าใดที่มีราคาหรือน้ำหนักเท่ากัน

ตัวอย่างของข้อมูลสินค้าในร้านค้า

```
{"table": (6, 120), "banana": (3.0, 35.50), "cucumber": (4.0, 42.00)}
```

หน้าที่ของคุณคือให้เขียนฟังก์ชัน *product_shopping*(*p_list*: dict[str, tuple[float, float]], *allowed_w*: float, *budget*: float) -> dict[str, float] เพื่อคืนค่า Dictionary แทนรายการสินค้าที่โดรนช้อปปิ้งเลือกซื้อทั้งหมด โดย Dictionary นั้นประกอบด้วย

key แทนชื่อสินค้าที่เป็น string

value แทนน้ำหนักของสินค้าที่เป็น float

โดยเงื่อนไขการเลือกซื้อของโดรนช้อปปิ้งมีดังนี้

1) ซื้อสินค้าให้ได้จำนวนชิ้นมากที่สุด
2) สินค้าที่ซื้อทั้งหมดจะต้องมีน้ำหนักรวมไม่เกินที่กำหนด
3) ราคารวมของสินค้าทั้งหมดจะต้องไม่เกินงบประมาณที่กำหนด
4) หากได้ผลลัพธ์ที่มีจำนวนชิ้นของสินค้าเท่ากันให้เลือกผลลัพธ์ที่มีน้ำหนักรวมน้อยกว่า และหากยังได้น้ำหนักรวมเท่ากันให้เลือกผลลัพธ์ที่มีราคารวมน้อยที่สุด
5) จากข้อ 4) หากยังได้ผลลัพธ์มากกว่า 1 คำตอบ ให้เลือกตอบเพียงคำตอบเดียวเท่านั้น

รายละเอียด Test case

- test cases 1-10: จะให้คะแนนโดยตรวจสอบความยาวของ dictionary ที่คืนค่าเท่านั้น
- test cases 11-20: จะให้คะแนนโดยตรวจสอบความถูกต้องของรายการสินค้าใน dictionary ที่คืนค่า

Function Call	**Output**

`p_list = {"table": (5, 900.), "chair": (0.4, 450.),`
`"pillow": (3.5, 1200), "stool": (0.3, 300.0)}`
`allowed_w = 25.0`
`budget = 2500.00`
 	
`print(product_shopping(p_list, allowed_w, budget))`
	`{'stool': 0.3,`
	`'chair': 0.4,`
	`'pillow': 3.5}`

`p_list = {"chair": (0.4, 450.0), "pillow": (3.5, 315.0),`
`"stool": (0.3, 300.0), "closet": (2.5, 700.0)}`
`allowed_w = 15.0`
`budget = 1450.00`
 	
`print(product_shopping(p_list, allowed_w, budget))`
	`{'stool': 0.3,`
	`'chair': 0.4,`
	`'closet': 2.5}`

4) **4 คะแนน** (HW13_2_6XXXXXXXX.py) **[Attachment]** ให้เขียนฟังก์ชัน *count_vote*(*pref_matrix*: `list[list[str]]) -> list[tuple[str, int]]` เพื่อคืนค่าคะแนนโหวตของ Pokémon ที่ได้จากการลงคะแนน Twitter-wide Favorite Pokémon แบบจัดลำดับ

การลงคะแนนแบบจัดลำดับ (อังกฤษ: ranked voting) หรือเรียกอีกอย่างว่า การลงคะแนนตามลำดับความชอบ (อังกฤษ: Ranked-choice Voting) หรือ การลงคะแนนตามความชอบ (อังกฤษ: Preferential Voting) เป็นระบบการลงคะแนนใดๆ ที่ผู้ลงคะแนนเสียงใช้การจัดลำดับผู้สมัคร (หรือลำดับความชอบ) ในบัตรลงคะแนนเพื่อเลือกผู้สมัครมากกว่าหนึ่งรายขึ้นไป และเพื่อเรียงลำดับตัวเลือกผู้สมัครทั้งหมดเป็นลำดับที่หนึ่ง สอง สาม ไปจนครบ (Wikipedia)

ในตัวแปร *pref_matrix* แต่ละ Row จะแทนการเลือกของ Voter แต่ละคน และ จำนวน Column ทั้งหมดแทนตัวเลือกที่เลือกได้ โดยการคำนวนคะแนนจะให้น้ำหนักคะแนนที่สูงที่สุดแก่ตัวเลือกอันดับแรกเช่น กรณีเลือกได้ 4 ตัวเลือก ตัวเลือกแรกจะได้น้ำหนักคะแนน 4 ตัวเลือกที่ 2 จะได้น้ำหนักคะแนน 3 ลดหลั่นกันไป จนตัวเลือกสุดท้ายจะมีน้ำหนักคะแนนเท่ากับ 1 ในกรณีที่เลือกได้ n ตัวเลือก อันดับที่ 1 ก็จะได้น้ำหนักคะแนนเท่ากับ n แทน เช่นในตัวอย่างด้านล่าง คะแนนของ Pikachu จะเท่ากับ 2 + 1 + 3 + 2 = 8

ฟังก์ชันจะคืนค่า `list` ของ `tuple` ที่ประกอบด้วยชื่อ Pokémon ทั้งหมดที่มีผู้ vote ให้ และคะแนนที่ได้ เรียงตามลำดับคะแนนจากมากไปน้อย และลำดับตัวอักษรในพจนานุกรมภาษาอังกฤษกรณีที่คะแนนเท่ากัน ทั้งนี้กำหนดให้ทุก row มีความยาวเท่ากัน

Input	**Output**:

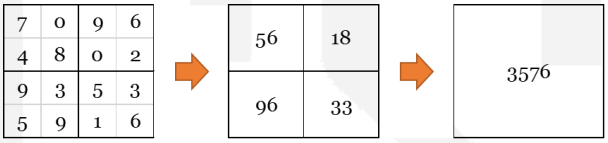
`[['Mewtwo', 'Pikachu', 'Suicune'],`
`['Mewtwo', 'Suicune', 'Pikachu'],`
`['Pikachu', 'Rayquaza', 'Charizard'],`
`['Suicune', 'Pikachu', 'Charizard']]`
	`[('Pikachu', 8),`
	`('Mewtwo', 6),`
	`('Suicune', 6),`
	`('Charizard', 2),`
	`('Rayquaza', 2)]`

5) **4 คะแนน** (HW13_3_6XXXXXXXX.py) **[Attachment]** ให้เขียนฟังก์ชัน *sum_d_product*(*m*: list[list[int]]) -> int เพื่อคืนค่าผลบวกของผลคูณทแยง (Sum of Diagonal Product) ใน matrix m ที่มีขนาด $n \times n$ เมื่อ n สามารถเขียนในรูปของ 2^x (x เป็นจำนวนเต็มบวก)

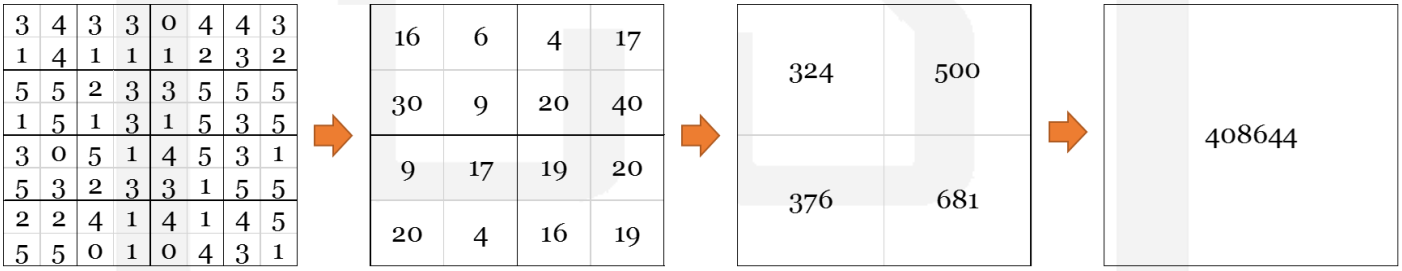
โดยกรณี matrix m ขนาด 2×2 เช่น $\begin{array}{|c|c|}\hline a & b \\ \hline c & d \\ \hline \end{array}$ สามารถหาผลลัพธ์ได้จากสูตร $a \times d + c \times b$

ดังนั้น matrix $\begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array}$ จะมี Sum of Diagonal Product = $(1 \times 4) + (3 \times 2) = 10$

กรณีต้องการหา Sum of Diagonal Product ของ matrix ขนาดใหญ่กว่า 2 × 2 ทำได้โดยการหา sum_d_product() ของ matrix ย่อย ขนาด 2 × 2 ก่อน แล้วหา sum_d_product() ของ matrix ผลลัพธ์อีกที เช่นกรณี matrix ขนาด 4 × 4 จะมีขั้นตอนดังนี้



หรือกรณี matrix 8 × 8



Input	**Output**
[[3, 3, 3, 2], [2, 0, 3, 1], [2, 1, 2, 3], [1, 0, 2, -1]	33
[[1, 1, 5, -1], [12, 2, -2, 0], [4, 8, 8, 12], [4, 12, 12, 15]]	3856
[[0, -1, -1, 3, 2, 3, -1, 3], [3, -1, -1, 2, 0, -1, 2, 1], [3, 0, 1, 2, 3, 1, 3, 1], [2, 2, 1, -1, -1, 2, 0, 3], [1, 3, 2, 1, 3, 2, 2, 1], [1, 2, 2, 1, 3, 3, 1, 3], [2, 2, 2, 2, 2, 2, 3, 3], [1, 3, 2, 3, 1, 1, 2, 2]]	-6290