กำหนดส่ง

Lab	
HW	
Until	

กระบวนวิชา 204111

การบ้านปฏิบัติการ 15

Problem Solving and Algorithm Practice (20 คะแนน)

ข้อกำหนด

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

1) **4 คะแนน** (`Lab15_1_6XXXXXXXX.py`) ให้เขียนฟังก์ชันแบบ `sum_nested_list(list_a: list)` เพื่อคืนค่าผลรวมของจำนวนเต็มทั้งหมดใน *list_a* คูณกับค่าความลึก (Depth) ของจำนวนนั้น ๆ โดยแต่ละสมาชิกของ *list_a* มีชนิดข้อมูลที่เป็นไปได้ 2 ประเภท คือ เป็นจำนวนเต็ม (`int`) หรือเป็น `list` โดย `list` ที่เป็นสมาชิกดังกล่าว ก็สามารถมีสมาชิกเป็นจำนวนเต็มและ `list` ได้เช่นกัน และกำหนดค่าความลึกคือจำนวนชั้นของลิสต์ที่ซ้อนกันก่อนที่จะเจอจำนวนนั้นๆ

ตัวอย่างเช่น

```
[1, 2, [[3, 0], 4], 8]
```

จากตัวอย่างเป็น `list` ที่มี 4 สมาชิก โดยสมาชิก ที่ `0`, และ `1` และ `3` ของ `list` มีชนิดเป็นจำนวนเต็ม ในขณะที่สมาชิกที่ 2 มีชนิดเป็น `list`: `[[3, 0], 4]` และผลรวมของจำนวนเต็มทั้งหมดจะมีค่า $1 + 2 + (3 \times 3) + (0 \times 3) + (4 \times 2) + 8 = 28$ เนื่องจาก 4 อยู่ในลิสต์ชั้นที่ 2 ในขณะที่ 3 และ 0 อยู่ในลิสต์ชั้นที่ 3

Hint:

- สามารถเรียกใช้ฟังก์ชัน `isinstance(object, classinfo)` เพื่อตรวจสอบชนิดของสมาชิก เช่น `isinstance([3], list)` จะคืนค่าเป็น **True**

Input	Output
`[1, 2, [[3, [[4], 5]], [6, 7]]]`	91
`[1, [2, [3]]]`	14
`[1, [[2, [3]], 4, [5]], [6, [7]]]`	75
`[9, [[8, [7]], 6, [5, [44, [33]]]]]`	429

2) **4 คะแนน** (`Lab15_2_6XXXXXXXX.py`) ทุกเดือนตุลาคมที่โรงงานช็อกโกแลตเม็ดหลากสียี่ห้อ N&N จะมีการออกผลิตภัณฑ์ Editon พิเศษเพื่อเป็นการฉลองความหลากหลายและความ Woke ประจำปี โดยทำเป็นกล่อง N&N ขนาดมินิบรรจุช็อกโกแลตโดยตั้งใจให้มีสีซ้ำกันน้อยที่สุด และเนื่องจากการบรรจุกล่องใช้น้ำหนักเป็นเกณฑ์ทำให้อาจมีจำนวนต่างกันเล็กน้อยในแต่ละกล่อง คุณจะต้องช่วยโรงงานตรวจสอบว่าแต่ละกล่องมีสีอะไรบ้างเพื่อจะพิมพ์ฉลากเท่าที่จำเป็น โดยคุณจะได้รับ List ของ Tuple แสดงสีของช็อกโกแลตในแต่ละกล่อง เช่น `[('red', 'green',`

'blue', 'blue'), ('blue', 'green', 'red', 'red')] ซึ่งจริงๆ แล้วมีสีเหมือนกันเพียงแต่บรรจุในลำดับที่ต่างกัน

หน้าที่ของคุณคือให้เขียนฟังก์ชัน *unique_combo*(*heaps*: list[tuple[str]]) -> set[tuple[str]] เพื่อคืนค่า Set ของ Tuple ที่มีสีที่แตกต่างจริงๆ จาก *heaps* ที่เป็น List ของ Tuple แสดงสีช็อกโกแลตในแต่ละกล่องโดยไม่พิจารณาลำดับ และในคำตอบจะเรียงลำดับสมาชิกของ Tuple ที่คืนค่าอย่างไรก็ได้

Input

```
[('red', 'green', 'blue', 'blue'),
 ('blue', 'green', 'red'),
 ('green', 'blue', 'red'),
 ('pink', 'purple', 'orange', 'pink'),
 ('orange', 'purple', 'pink')]
```

Output

```
{('red', 'green', 'blue'),
 ('purple', 'pink', 'orange')}
```

3) **4 คะแนน** (HW15_1_6XXXXXXXX.py) น้องออนิวเป็นนักสะสมหนังสือการ์ตูนมังงะ (Manga) ในชั้นวางหนังสือของเขาที่บ้านเกิดจังหวัดเชียงใหม่ ออนิวเรียงหนังสือไว้อย่างเรียบร้อยตามวิสัยนักสะสม Manga ทั่วไป โดยจะเรียงตามชื่อเรื่อง และ เลขประจำเล่ม และเมื่อเขาซื้อหนังสือ Manga มาเพิ่มเขาจะต้องนำไปเรียงแทรกในตำแหน่งที่ถูกต้องเสมอ โชคร้ายที่หนังสือบางส่วนเสียหายจากความชื้นจากพายุฝนลูกเห็บที่มาพร้อมผู้นำที่มาเยือนจังหวัดเชียงใหม่เมื่อไม่นานมานี้ เขาจึงต้องทยอยหาเล่มใหม่มาใส่ในชั้นคืนจากหลากหลายแหล่งที่มา ซึ่งจะส่งมาที่บ้านในลำดับและชื่อเรื่องที่คละกันไป จากคอร์ส Python ที่เขาเรียนออนไลน์ เขาพบว่าเขาสามารถใช้ Binary Search ช่วยเรียงหนังสือเข้าชั้นวางได้เร็วกว่าวิธีไล่เรียงจากเล่มแรกมาแบบที่เขาเคยใช้

หน้าที่ของคุณคือให้เขียนฟังก์ชัน *manga_add*(*manga_shelf*: list[tuple[str, int]], *new_m*: tuple[str, int], *show_steps*: bool = False) -> None เพื่อเปลี่ยนแปลง ชั้นวางหนังสือ *manga_shelf* เมื่อมีการเพิ่มหนังสือ Manga เล่มใหม่ *new_m* โดย *new_m* ที่เป็น Tuple ของ (*title*, *num*) เมื่อ *title* คือ String แทนชื่อเรื่องในภาษาอังกฤษ (สามารถมีอักขระว่าง หรือเครื่องหมายต่าง ๆ หรือตัวเลข) และ *num* คือ Integer แทนเลขประจำเล่ม และ *manga_shelf* เป็น List ของ Tuple ของหนังสือในรูป (*title*, *num*) ที่อาจเป็นชั้นเปล่า หรือเป็นชั้นที่มีหนังสือที่เรียงลำดับไว้แล้ว โดยมี Optional Parameter *show_step* เพื่อแสดงตำแหน่ง Index ที่ต้องทำการเปรียบเทียบในแต่ละรอบของการทำการวนซ้ำภายใน Binary Search ทั้งนี้ให้ถือว่าจะไม่มีเล่มไหนซ้ำกัน ชั้นวางหนังสือมีความยาวไม่จำกัด และกำหนดให้ฟังก์ชันทำงานแบบ **Destructive**

Function Call

```
shelf = [('Bleach', 10), ('Naruto', 5), ('One Piece', 24)]
new = ('Naruto', 18)
manga_add(shelf, new, True)
print('--')
print(shelf)
```

Output

```
[1] ('Naruto', 5)
[2] ('One Piece', 24)
--
[('Bleach', 10), ('Naruto', 5), ('Naruto', 18), ('One Piece', 24)]
```

Function Call

```
shelf = [('Bleach', 100), ('Bleach', 10000)]
new = ('Bleach', 99)
manga_add(shelf, new, True)
print('--')
print(shelf)
```

Output

```
[0] ('Bleach', 100)
--
[('Bleach', 99), ('Bleach', 100), ('Bleach', 10000)]
```

Function Call

```
shelf = [('Bleach', 1), ('Bleach', 3), ('Bleach', 4)]
new = ('Bleach', 2)
manga_add(shelf, new, True)
print('--')
print(shelf)
```

Output

```
[1] ('Bleach', 3)
[0] ('Bleach', 1)
--
[('Bleach', 1), ('Bleach', 2), ('Bleach', 3), ('Bleach', 4)]
```

4) **4 คะแนน** (HW15_1_6XXXXXXXX.py) ให้เขียนฟังก์ชัน *histogram*(*scores*: list[int]) -> None เพื่อแสดงผลแผนภูมิ histogram ของคะแนนรายวิชาโปรแกรมมิ่ง 101 ณ สถาบันแห่งหนึ่งทางภาคเหนือ โดยให้คำนวณความถี่จากตัวแปร *scores* ที่อยู่ในรูป Tuple ความยาว n ($n > 0$) ซึ่งคะแนนของนักศึกษาแต่ละคนจะเป็นจำนวนเต็มตั้งแต่ 0 - 100

ในการแจกแจงความถี่ กำหนดให้ bin size มีขนาด 10 เสมอ (ยกเว้นช่วงคะแนนสุดท้าย) โดยให้ bin แรกสุดสำหรับคะแนน 0 - 9 คะแนน และ bin ถัดไปสำหรับคะแนน 10 - 19 ดังนี้ จนไปถึง bin สุดท้ายสำหรับคะแนน 90 - 100 คะแนน (bin size ขนาด 11) ทั้งนี้ในการแสดงผลเครื่องหมาย '*****' หนึ่งแถวในแนวนอนจะแทนคะแนน 5 คะแนน โดยจะแสดงผลแบบปัดขึ้น ดังนั้นความถี่ที่ 48 คน จะแสดงผลด้วย '*****' 10 แถวเป็นต้น

Hint: เราสามารถสร้าง string สำหรับแต่ละแท่งแทนช่วงความถี่แยกกันแ แล้วนำมารวมด้วย string method ต่าง ๆ

Input

```
(19, 39, 59, 42, 42, 42, 100)
```

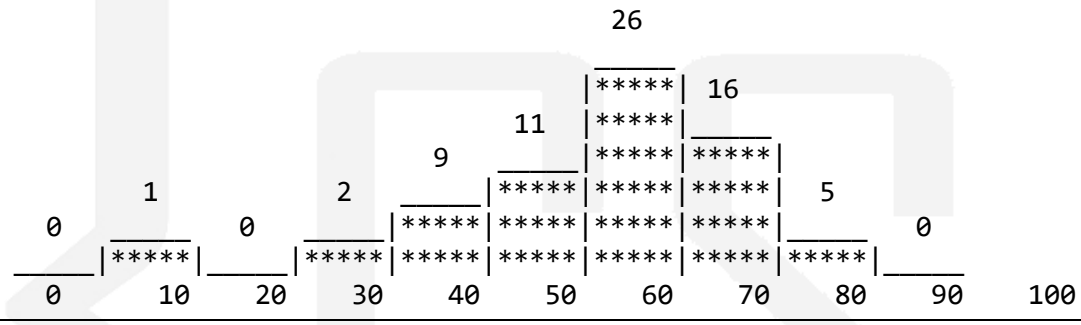
Output

```
         1           1     3     1                       1
   0   _____     0  _____ _____ _____     0     0     0  _____
_____|*****|_____|*****|*****|*****|_____ _____ _____|*****|
0      10    20    30    40    50    60    70    80    90    100
```

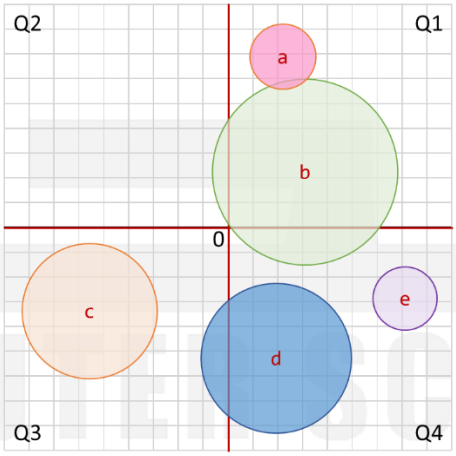
Input

```
(62, 49, 75, 86, 71, 63, 74, 42, 57, 75,
 56, 58, 67, 78, 63, 73, 60, 49, 66, 77,
 47, 69, 74, 63, 65, 64, 55, 52, 52, 57,
 86, 75, 68, 70, 34, 34, 68, 46, 60, 56,
 60, 65, 66, 70, 64, 84, 61, 46, 60, 76,
 59, 64, 68, 69, 68, 47, 72, 80, 11, 44,
 53, 70, 50, 79, 81, 68, 75, 48, 62, 68)
```

Output



5) **4 คะแนน** (HW15_3_6XXXXXXXX.py) ให้เขียนฟังก์ชัน `count_segment(list_a: list[tuple[float]]) -> tuple[int]` เพื่อคืนค่าจำนวนส่วนของวงกลม (Segment) ที่อยู่ใน Quadrant ต่างๆ ที่ระบุด้วย *list_a* โดย *list_a* จะเป็น List ของ `tuple` ที่อยู่ใน รูป $(px,\ py,\ r)$ เมื่อ px และ py คือพิกัดในแนวแกน x และแกน y ตามลำดับ และ r คือ รัศมีวงกลม $(r > 0)$ โดยฟังก์ชันจะคืนค่า `tuple` แทนจำนวนวงกลม หรือส่วนของวงกลม ที่อยู่ใน Quadrant 1, 2, 3 และ 4 ตามลำดับ



เช่นจากรูปด้านบน ฟังก์ชันจะคืนค่า `(2, 1, 2, 3)`

Input

```
[(2, 7, 1.5),          # a
 (3.2, 2.5, 4.06),     # b
 (-5.5, -4.5, 2.5),    # c
 (2, -5.2, 3),         # d
 (7.2, -2.8, 1.2)]     # e
```

Output

```
(2, 1, 2, 3)
```