Assignment 5: The 24 Game

เกม 24 (https://en.wikipedia.org/wiki/24_Game) เป็นเกมทางคณิตศาสตร์ที่ผู้เล่นได้รับเลข 4 จำนวน เพื่อนำมาจัดการบวก ลบ คูณ หรือ หาร จำนวนเหล่านี้ในลำดับใด ๆ สักหนึ่งลำดับ เพื่อให้ได้ค่า 24 เช่น ให้จำนวนทั้งสี่คือ 14, 17, 2 และ 9 จะได้ว่า วิธีหนึ่งที่คำนวณได้ค่า 24 คือ ((14 + 2) – 9) + 17 แรกเริ่มเกมนี้ใช้สำรับไพ่เฉพาะเลขตั้งแต่ 1 ถึง 10 โดยมีไพ่เอซ (Ace) แทนตัวเลข 1 ถัดมามีการใช้ไพ่แจ็ค (Jack) แหม่ม (Queen) และคิง (King) แทนเลข 11, 12 และ 13 ตามลำดับ

โจทย์ข้อนี้เกี่ยวกับการสร้างโปรแกรมเพื่อรับจำนวนเต็มใด ๆ 4 จำนวน เพื่อหาคำตอบของเกม 24

ก่อนอื่นมาดูจำนวนรูปแบบที่ต้องคำนวณ จำนวนทั้งสี่และตัวดำเนินการ (operator) จะถูกวางเป็นลำดับในรูปแบบ

$$\text{number}_1 \quad \text{operator}_1 \quad \text{number}_2 \quad \text{operator}_2 \quad \text{number}_3 \quad \text{operator}_3 \quad \text{number}_4$$

- หากจำนวนที่รับไม่เหมือนกันเลย จะมีวิธีการเรียงสับเปลี่ยนจำนวนทั้งสี่ได้ 4! หรือ 24 แบบ (เหมือนกับชื่อเกม)
- ตัวดำเนินการมี 4 แบบ คือ + – * / (ใช้ซ้ำได้) วางได้ 3 ตำแหน่งระหว่างจำนวน จึงวางได้ 4^3 = 64 แบบ
- ดังนั้น รูปแบบการวางจำนวนและตัวดำเนินการทั้งหมดมีอย่างมาก 24 x 64 = 1,536 แบบ
- ในแต่ละรูปแบบ เช่น `5 + 3 * 2 - 1` ต้องระบุลำดับการคำนวณด้วย ซึ่งมีรูปแบบละอีกอย่างมาก 5 แบบย่อย คือ

```
( ( 5 + 3 ) * 2 ) - 1
( 5 + ( 3 * 2 ) ) - 1
( 5 + 3 ) * ( 2 - 1 )
5 + ( ( 3 * 2 ) - 1 )
5 + ( 3 * ( 2 - 1 ) )
```

- สรุปว่า ต้องพิจารณาการคำนวณทั้งหมดอย่างมาก 4! x 4^3 x 5 = 7,680 แบบ

การหาคำตอบในโจทย์นี้ จะทำแบบลุยคำนวณทุกแบบที่เป็นไปได้ แต่**ให้แสดงผลออกมาเพียงรูปแบบเดียวก็พอ** (ที่ได้ค่า 24)

สิ่งที่ต้องทำ

ในโปรแกรมต้นแบบมีฟังก์ชัน **`generate_all_combinations( num_list, operators )`** ซึ่งมี

- **`num_list`** เป็นลิสต์ เก็บจำนวนต่าง ๆ ที่ต้องการพิจารณา
- **`operators`** เป็นสตริง เก็บเครื่องหมายตัวดำเนินการที่เป็นไปได้
- ฟังก์ชันนี้คืน ลิสต์ของรูปแบบการวางจำนวนและตัวดำเนินการที่เป็นได้ทั้งหมด (ที่ยังไม่ได้ระบุลำดับการคำนวณ)

เช่น คำสั่ง **`C = generate_all_combinations( [1,2,3,4], '+-*/' )`** จะได้ค่าที่เก็บใน **`C`** เป็นลิสต์ ดังนี้

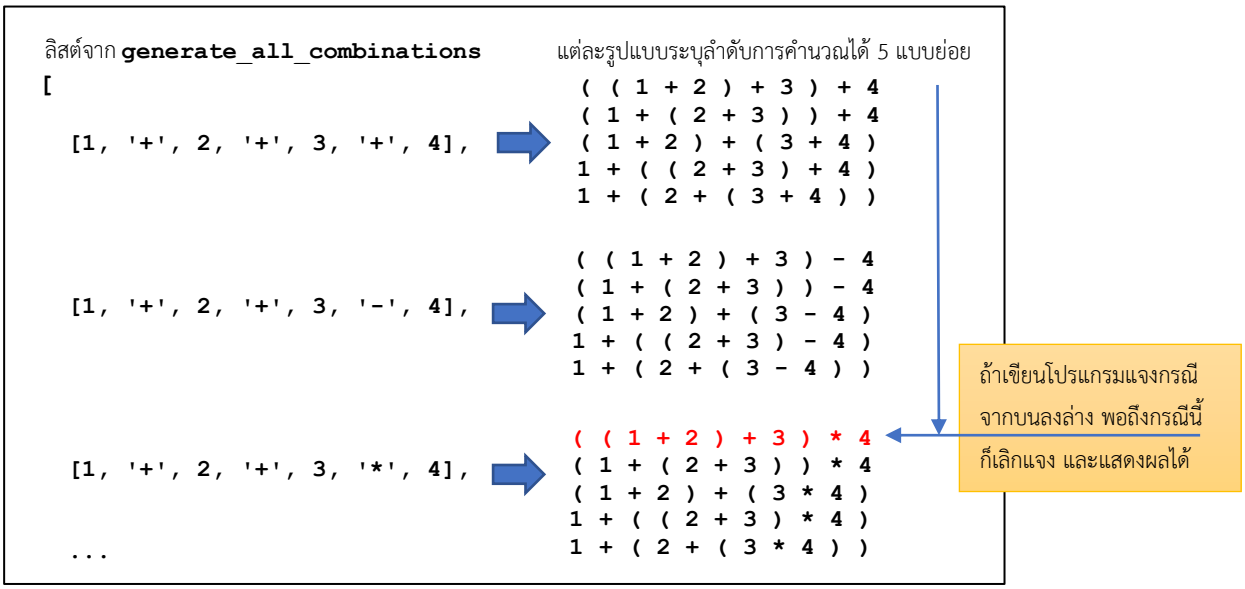
```
[
    [1, '+', 2, '+', 3, '+', 4],
    [1, '+', 2, '+', 3, '-', 4],
    [1, '+', 2, '+', 3, '*', 4],
    [1, '+', 2, '+', 3, '/', 4],
    [1, '+', 2, '-', 3, '+', 4],
    [1, '+', 2, '-', 3, '-', 4],
    [1, '+', 2, '-', 3, '*', 4],
    [1, '+', 2, '-', 3, '/', 4],
    [1, '+', 2, '*', 3, '+', 4],
    [1, '+', 2, '*', 3, '-', 4],
    [1, '+', 2, '*', 3, '*', 4],
    [1, '+', 2, '*', 3, '/', 4],
    [1, '+', 2, '/', 3, '+', 4],
    [1, '+', 2, '/', 3, '-', 4],
    [1, '+', 2, '/', 3, '*', 4],
    [1, '+', 2, '/', 3, '/', 4],
    [1, '-', 2, '+', 3, '+', 4],
    [1, '-', 2, '+', 3, '-', 4],
    [1, '-', 2, '+', 3, '*', 4],
    [1, '-', 2, '+', 3, '/', 4],

    ... # ยังมีอีกมากมาย

    [4, '/', 3, '/', 2, '*', 1],
    [4, '/', 3, '/', 2, '/', 1]
]
```

ขั้นตอนการทำงานของโปรแกรมที่ต้องเขียนเป็นดังนี้

- เริ่มด้วยการรับรายการของจำนวนเต็ม (4 จำนวน ในบรรทัดเดียวกัน แต่ละจำนวนคั่นด้วยช่องว่าง)
- ใช้ฟังก์ชัน **generate_all_combinations** เพื่อสร้างลิสต์ที่เก็บทุกรูปแบบ (ที่ยังไม่ได้ระบุลำดับการคำนวณ)
- **นำแต่ละรูปแบบจากลิสต์ในขั้นตอนที่แล้ว** ไปหาว่า รูปแบบใดที่เมื่อพิจารณาลำดับการคำนวณซึ่งมีทั้งหมด 5 แบบย่อย แล้วมีแบบย่อยใดได้ค่า 24 ดังแสดงในรูปข้างล่างนี้



- ถ้าพบแบบที่ได้ค่า 24 ให้แสดงผลที่หาได้ในรูปแบบข้างล่างนี้ เช่น ถ้ารับ **1 2 3 4** จะแสดงผลลัพธ์คือ

  ((1 + 2) + 3) * 4 = 24

  - ต้องแสดงในรูปแบบข้างบนนี้ คือ
    - มีวงเล็บ เปิด ปิด ให้ครบ (มีเปิด 2 ตัว ปิด 2 ตัว)
    - ทุกอย่างคั่นด้วยช่องว่างหนึ่งช่อง
    - มี **= 24** อยู่ทางขวา
  - จะแสดงผลลัพธ์อื่นก็ได้ ขอให้คำนวณแล้วได้ค่า 24 เช่น **4 * (1 + (3 + 2)) = 24**
  - ในกรณีที่ลองครบทุกแบบแล้ว ก็ไม่ได้ค่า 24 ให้แสดงข้อความว่า **No Solutions**

ข้อแนะนำ

- โปรแกรมอาจจะกะทัดรัดขึ้น ถ้าเขียนฟังก์ชันเสริม **calc(num1, op, num2)** เพื่อคืนผลที่ได้จากการนำจำนวน **num1** กับ **num2** มาคำนวณกันด้วยตัวดำเนินการ **op** ที่ให้มา (ซึ่งมีได้ 4 กรณีคือ **+**, **-**, ***** หรือ **/** แทน บวก ลบ คูณ หรือ หารตามลำดับ) เช่น เขียนว่า **r = calc(2, '*', 3)** จะได้ **r** มีค่า 6

```
def calc(num1, op, num2):
    if operation == '+':
        r = num1 + num2
    elif ...
        ...

    return r
```

- (หากจำเป็น) สามารถสร้างฟังก์ชันอื่น ๆ ที่ใช้เองในบริเวณเขียวที่เตรียมไว้ให้ได้
- ตอนเขียนโปรแกรมจริง ๆ คงไม่จำเป็นต้องมาใส่วงเล็บเพื่อนำไปคำนวณ (ที่แสดงวงเล็บให้ดูในตัวอย่างนั้น เป็นเพราะให้อ่านเข้าใจได้ง่าย) แต่ต้องแสดงวงเล็บเปิดปิดให้ถูกต้องตอนแสดงผลลัพธ์สุดท้าย

โปรแกรมต้นฉบับ

```
# Prog-05: The 24 Game
# 6????????21 Name ?
```

ใส่เลขประจำตัว ชื่อ นามสกุล

download code นี้ได้

```
from itertools import permutations, product
import math

def generate_all_combinations(num_list, operators):
    all_combi = []
    for n,o in product(sorted(set(permutations(num_list))),
                       product(operators, repeat=3)):
        x = [None]*(len(n)+len(o))
        x[::2] = n
        x[1::2] = o
        all_combi.append(x)
    return all_combi
#------------------------------------------------------------
```

โปรแกรมที่ส่ง ห้ามเปลี่ยนโค้ดส่วนที่เป็นสีแดงโดยเด็ดขาด เปลี่ยนได้เฉพาะส่วนที่มีพื้นหลังเป็นสีเขียวเท่านั้น

เขียนฟังก์ชันอื่นที่ต้องการได้ ในบริเวณนี้

```
#------------------------------------------------------------
nums = input('Enter 4 integers: ')
...
...
cases = generate_all_combinations( ????,   '+-*/' )
...
```

ต้องนำผลที่ได้ใน **cases** ไปประมวลผลต่อ

ตัวอย่างการใช้งานและการแสดงผล

```
>>> %Run the24game.py
Enter 4 integers: 5 5 9 5
( ( 5 + 5 ) + 5 ) + 9 = 24
```

```
>>> %Run the24game.py
Enter 4 integers: 13 2 13 13
2 * ( 13 - ( 13 / 13 ) ) = 24
```

```
>>> %Run the24game.py
Enter 4 integers: 1 1 2 7
( 1 + 2 ) * ( 1 + 7 ) = 24
```

```
>>> %Run the24game.py
Enter 4 integers: 200 -120 10 3
( ( -120 + 200 ) * 3 ) / 10 = 24
```

```
>>> %Run the24game.py
Enter 4 integers: 1 1 1 9
No Solutions
```

รูปแบบการแสดงในบริเวณพื้นหลังสีเหลือง ต้องเป็นไปตามตัวอย่าง ไม่ขาด ไม่เกิน **ตัวสีแดง** คือจำนวนที่ผู้ใช้ป้อนให้โปรแกรม

ผลที่ได้ ไม่จำเป็นต้องเหมือนที่แสดงในนี้ก็ได้ แค่ขอให้คำนวณแล้วได้ค่า 24 ก็ถือว่าถูกต้อง

คำเตือน

เพื่อลดขั้นตอนในการสร้างข้อมูลทดสอบในขั้นตอนการให้คะแนน ระบบตรวจอาจเปลี่ยนแปลงการทำงานของฟังก์ชัน **generate_all_combinations** เพื่อสร้างลิสต์ที่เก็บรูปแบบ ไม่ครบทุกกรณี ดังนั้น จึงขอเน้นตรงนี้อีกครั้งว่า ต้องนำผลที่ได้จากฟังก์ชัน **generate_all_combinations** ไปตรวจสอบต่อ ตามที่ได้เขียนไว้ในโปรแกรมต้นฉบับ