Set

- เหมือนกับเซตทางคณิตศาสตร์ เซตไม่เก็บข้อมูลซ้ำ สามารถ union, intersect ได้
- เหมือนกับ dict แบบที่เก็บแค่ key ไม่มี value (เพราะ key ไม่ซ้ำกัน)
- การค้นด้วย in สามารถทำได้เร็วมาก

การสร้างเซต สามารถสร้างเซตจากข้อมูลใน string, tuple, list, set หรือ dict ได้	set_1 = set() *ได้เซตว่าง* set_2 = {1, 2, 3} *ได้เซตที่มีสมาชิกเป็น* 1, 2, 3 set_3 = set('Hello') *ได้เซต* {'H','e','l','o'} set_4 = set(['oh', 'no']) *ได้เซต* {'oh', 'no'} set_5 = set(set_2) *ได้เซตใหม่มีสมาชิกเหมือนของ* set_2 set_6 = set({1:2, 3:4}) *ได้เซตของ* key *ของ* dict *คือ* {1,3}
บริการ add(e) ใช้เพิ่มข้อมูล 1 ตัว ถ้าเพิ่มตัวที่ซ้ำกับที่มีอยู่ใน set ก็ไม่เพิ่มให้	S = {1} S.add((2, 3)) *ได้* {1, (2, 3)} S.add('Hello') *ได้* {1, (2, 3), 'Hello'}
บริการ update(t) ใช้เพิ่มข้อมูลที่แจกแจงได้จาก t ที่เป็นกลุ่มข้อมูล ซึ่งเป็นได้ทั้ง string, tuple, list, set และ dict (ในกรณีของ dict จะเพิ่ม key) คำสั่ง s.update(t) ทำงานเหมือนคำสั่ง for e in t : s.add(e)	S = {1} S.update([2, 3]) *ได้* {1,2,3} S.update('Hello') *ได้* {1,2,3,'H','e','l','o'} S.update(6) *เกิด* error
การใช้ e in S เพื่อตรวจสอบความเป็นสมาชิกของ e ในเซต S	S = {1, 2, 3, 'H', 'e', 'l', 'o'} x = 1 in S *ได้* x = True y = 'h' in S *ได้* y = False
บริการ remove(e) และ discard(e) เพื่อลบข้อมูล ถ้าไม่มีข้อมูลที่ต้องการจะลบ remove() จะเกิด error ดังนั้นควรใช้ discard()	S = {1, 2, 3, 'H', 'e', 'l', 'o'} S.discard(1) *เหลือ* {2, 3, 'H', 'e', 'l', 'o'} S.discard(4) *เหลือเท่าเดิม ไม่เกิด* error S.remove('e') *เหลือ* {2, 3, 'H', 'l', 'o'} S.remove('h') *เกิด* error *เพราะไม่เจอ* 'h'

CONFIDENTIAL

การดำเนินการของ set union() หรือใช้สัญลักษณ์ | intersection() หรือใช้สัญลักษณ์ & difference() หรือใช้สัญลักษณ์ - (A∪B) - (A∩B) หรือใช้สัญลักษณ์ ^ โดยเซตที่มากระทำกันจะไม่เปลี่ยนแปลง	a = {1, 2, 3} b = {2, 3, 4} c = a.union(b) *หรือ* c = a | b *ได้* c = {1, 2, 3, 4} c = a.intersection(b) *หรือ* c = a & b *ได้* c = {2, 3} c = a.difference(b) *หรือ* c = a - b *ได้* c = {1} c = b.difference(a) *หรือ* c = b - a *ได้* c = {4} c = a ^ b *ได้* c = {1, 4}
บริการ issubset() และ issuperset() ใช้ตรวจสอบความเป็น subset และ superset	a = {1, 2, 3} b = {1, 2} a.issubset(b) *ได้* False b.issubset(a) *ได้* True a.issuperset(b) *ได้* True b.issuperset(a) *ได้* False

CONFIDENTIAL

สรุปการใช้งาน list, tuple, dict, set

- ใช้คำสั่ง `len, sum, max, min, in` ได้ทั้งหมด
- ใช้ `x = sorted(q)` ได้ โดยที่ `q` เป็นได้ทั้ง `list, tuple, dict, set`
  ผลที่ได้เป็น `list` ที่นำข้อมูลที่แจกแจงได้จาก `q` ไปเรียงลำดับ

	list	tuple	dict	set
การใช้งาน	ลำดับมีความหมาย สร้างแล้วแก้ไขได้	ลำดับมีความหมาย สร้างแล้วแก้ไขไม่ได้	เก็บคู่ลำดับ `(key, value)` `key` ไม่ซ้ำ, ไม่สนลำดับ	เก็บข้อมูลไม่ซ้ำ ไม่สนลำดับ สามารถใช้ set operation ได้
ประเภทข้อมูลที่เก็บ	อะไรก็ได้	อะไรก็ได้	`key` เป็น `int, float, str, tuple, bool` ส่วน `value` เป็นอะไรก็ได้	`int, float, str, tuple, bool`
การเข้าถึงข้อมูล	ใช้จำนวนเต็มระบุตำแหน่งหรือช่วง `x[i]` `x[a:b:c]`	ใช้จำนวนเต็มระบุตำแหน่งหรือช่วง `t[i]` `t[a:b:c]`	ใช้ `key` ระบุตำแหน่ง `d[key]` ได้ `value` ที่คู่กัน ไม่มีแบบรับ `value` แล้วได้ `key`	ไม่มี

การแจกแจงข้อมูล	`for e in x` ได้ลำดับจากซ้ายไปขวา	`for e in t` ได้ลำดับจากซ้ายไปขวา	`for k in d`
			`for k in d.keys()`
			`for v in d.values()`
			`for k,v in d.items()` ได้ลำดับไม่แน่นอน
				`for e in s` ได้ลำดับไม่แน่นอน

การค้นด้วย `in, not in`	ค้นจากซ้ายไปขวา (ช้า) ใช้ `x.index(e)` หา index ของ e ใน x	ค้นจากซ้ายไปขวา (ช้า) ใช้ `x.index(e)` หา index ของ e ใน x	ค้น `key` เร็วมาก	ค้นข้อมูลเร็วมาก

CONFIDENTIAL

	list	tuple	dict	set
การสร้าง	x = [1,2,3,4] x = list() x = [] x = list(q) เมื่อ q เป็นสิ่งที่ใช้กับ for in ได้	t = (1, 2, 3, 4) t = (5,) t = tuple() t = () t = tuple(q) เมื่อ q เป็นสิ่งที่ใช้กับ for in ได้	d = {'k1':1,'k2':2} d = dict() d = {}	s = {1, 2, 3, 4} s = set() s = set(q) เมื่อ q เป็นสิ่งที่ใช้กับ for in ได้
	ใช้ x = list(x1) เมื่อ x1 เป็น list ไม่ควรเขียน x = x1 จะเป็น list ตัวเดียวกัน	ใช้ t = t1 ได้ เมื่อ t1 เป็น tuple เพราะ tuple ไม่เปลี่ยนค่า	ใช้ d = dict(d1) เมื่อ d1 เป็น dict ไม่ควรเขียน d = d1 จะเป็น dict ตัวเดียวกัน	ใช้ s = set(s1) เมื่อ s1 เป็น set ไม่ควรเขียน s = s1 จะเป็น set ตัวเดียวกัน
การเพิ่มข้อมูล	x.append(9) x.insert(1,9)	ต้องสร้างตัวใหม่ t=t + (9,) t=t[:1]+(9,)+t[1:]	d['k3'] = 9 d.update({'k3':9})	s.add(9) s.update({9})
การลบข้อมูล	x.pop(2)	ต้องสร้างตัวใหม่ t = t[:2] + t[3:]	d.pop('k3')	s.discard(99)
การแก้ไขข้อมูล	x[2] = 7	ต้องสร้างตัวใหม่ t=t[:2]+(7,)+t[3:]	d['k3'] = 7	ต้องลบแล้วเพิ่ม

การใช้ tuple, dict, set ที่พบบ่อย

ใช้ dict เพื่อจับคู่ข้อมูล หรือสร้างความสัมพันธ์ระหว่างคู่ข้อมูล key กับ value โดยหวังจะขอ value จากค่า key	month = {'JAN':1, 'FEB':2, 'MAR':3} ใช้ month[k] เพื่อขอเลขเดือนจากชื่อย่อเดือนที่เก็บในตัวแปร k num2en = {11:'eleven', 2:'two', 3:'three'} ใช้ num2en[a] เพื่อขอคำภาษาอังกฤษจากจำนวนเต็มในตัวแปร a
ใช้ set เพื่อเก็บกลุ่มข้อมูลที่ไม่ซ้ำ ต้องการค้นข้อมูลว่ามีอยู่หรือไม่ในเซตอย่างรวดเร็ว หรือต้องการบริการ intersection, union, issubset และอื่น ๆ ของ set	ต้องการหาจำนวนประกอบที่มีค่า ระหว่าง 2 ถึง 12 n = 13; c = set() for i in range(2, n//2) : for j in range(2*i, n, i) : c.add(j) # *มีการเพิ่ม* j *ที่ซ้ำกัน แต่* add *ไม่เพิ่มข้อมูลซ้ำ* # c = {4, 6, 8, 9, 10, 12}

CONFIDENTIAL

ใช้ tuple เมื่อต้องการเก็บข้อมูลมีลำดับคล้าย list แต่มั่นใจว่าหลังสร้างแล้วจะไม่เปลี่ยนค่าภายใน tuple ประหยัดหน่วยความจำ
 	
ใช้คำสั่ง t1 = t2 ได้โดยไม่ต้องกังวลเรื่องการใช้ tuple ร่วมกัน (เพราะค่าภายในเปลี่ยนไม่ได้) และสามารถใช้เป็น key ของ dict และเป็นข้อมูลที่เก็บใน set ได้
	มีพิกัด x, y, z จำนวนหนึ่งที่ต้องเก็บ ถ้าต้องการหา z จาก x,y ที่กำหนดให้บ่อย ๆ ก็ไม่ควรเก็บเป็น list of tuples (x,y,z) เช่น
	`d = [ (1,1,3), (2,1,8), (3,1,2) ]`
	แบบนี้การหา z จาก x,y ก็ต้องเป็น
	`for a,b,z in d :`
	`  if x == a and y == b :`
	`  print('z =', z)`
	`  break`
	`else:`
	`  print('Not Found')`
	ควรใช้ dict {(x,y):z} เช่น d={(1,1):3,(2,1):8,(3,1):2}
	แบบนี้ การหา z จาก x,y ก็ง่ายและที่สำคัญคือเร็ว
	`if (x,y) in d :`
	`  print('z =', d[(x,y)])`
	`else :`
	`  print('Not Found')`

การแจกแจงข้อมูลต่าง ๆ ใน dict	`d = {1:7, 2:8, 3:9}`
	`for k in d:`
	`  print(k)`
	`for k in d.keys():`
	`  print(k)`
	`for v in d.values():`
	`  print(v)`
	`for k,v in d.items():`
	`  print(k,v)`

การแจกแจงข้อมูลใน dict เรียงตาม key หรือเรียงตาม value จากน้อยไปมาก	`d = {2:8, 1:8, 3:9}`
	`for k in sorted(d.keys()):`
	`  print(k,d[k])`
	`for v in sorted(d.values()):`
	`  print(v)`

การรับคู่อันดับข้อมูล แล้วเพิ่มลงใน dict	`d = {}`
	`n = int(input())`
	`for i in range(n):`
	`  k,v = input().split()`
	`  d[k] = v`

การรับคู่อันดับข้อมูล แล้วเพิ่มลงใน dict ที่มี value เป็น list หรือ set	`d = {}`
	`n = int(input())`
	`for i in range(n):`
	`  k,v = input().split()`
	`  if k not in d:`
	`  d[k] = [v]  # ถ้า d[k] เป็นเซต, ใช้ d[k] = {v}`
	`  else :`
	`  d[k].append(v)  # ถ้า d[k] เป็นเซต, ใช้ d[k].add(v)`
	
	ข้อควรระวัง : คำสั่ง d[k] = [v] ถ้าเขียนเป็น d[k] = list(v) จะมีปัญหา ถ้า v เป็นสตริง (เพราะอะไร ?)

CONFIDENTIAL

CONFIDENT

สร้างลิสต์ของ tuple เก็บข้อมูลชั่วคราว เพื่อการประมวลผล (เช่น เรียงลำดับข้อมูล)	d เป็น dict ที่ key เป็นรหัสนิสิต ส่วน value เป็นลิสต์ของคะแนน ต้องการแสดงชื่อเรียงลำดับตามคะแนนรวมจากน้อยไปมาก
	`x = [(sum(scores),sid) for sid,scores in d.items()]`
	`t = [sid for sum_scores,sid in sorted(x)]`
	`print('\n'.join(t))`

dict มี key เป็น a, value เป็นเซตของ b ต้องการสร้างอีก dict ที่กลับลักษณะ การจัดเก็บคือมี key เป็น b, value เป็นเซต ของ a	c เป็น dict ที่ key เป็นรหัสนิสิต ส่วน value เป็นเซตของรหัสวิชา จึงใช้ c ตอบคำถามว่ารหัสนิสิต sid เรียนวิชาอะไร ได้อย่างรวดเร็ว ถ้าอยากรู้ด้วยว่า รหัสวิชา cid มีรหัสนิสิตใดเรียนบ้าง ก็ต้องสร้างอีก dict stu เป็น dict ที่ key เป็นรหัสวิชา ส่วน value เป็นเซตของรหัสนิสิต
	`stu = dict()`
	`for (sid,cids) in c.items() :`
	`  for cid in cids :`
	`  if cid not in stu :`
	`  stu[cid] = {sid}`
	`  else :`
	`  stu[cid].add(sid)`

นำชุดข้อมูลที่อาจมีค่าซ้ำกันมาเพิ่มใส่ set แล้วได้ชุดข้อมูลที่ไม่มีค่าซ้ำ	จาก dict c ในข้อที่แล้ว อยากทราบว่ามีรหัสวิชาอะไรบ้างที่มีนิสิตเรียน
	`d = set()`
	`for cids in c.values() :`
	`  d.update( cids )`

เรื่องผิดบ่อย

คำสั่ง a = b โดยที่ b คือ set, dict หรือ list จะทำให้ a กับ b เป็นตัวแปรของที่เก็บข้อมูลที่เดียวกัน ถ้าต้องการเป็นคนละตัว แต่เก็บข้อมูลเหมือนกัน ต้องใช้ a = set(b) หรือ a = dict(b) หรือ a = list(b)	`A = {1:'one', 2:'two', 3:'three', 10:'ten'}`
	`B = A`
	`B[4] = 'four'`
	จะเป็นการแก้ทั้ง A และ B เพราะเป็น dict เดียวกัน
	ถ้าใช้คำสั่ง C = dict(A) จะได้ A และ C เป็นคนละ dict กัน แต่มีข้อมูลเหมือนกัน

ไม่สามารถใช้ d.sort() เมื่อ d เป็น tuple, dict หรือ set ได้ แต่สามารถใช้ sorted(d) ได้ โดย d คือ tuple, dict, set หรือ list	`S = {3, 1, 2}; D = {'A':5, 'C':2, 'B':7}`
	`S.sort()` *ผิด เพราะ set ไม่มีบริการ sort*
	`L = sorted(S)` *ได้* `L = [1, 2, 3]`
	`L = sorted(D)` *ได้* `L = ['A', 'B', 'C']`
	`L = sorted(D.values())` *ได้* `L = [2, 5, 7]`

CONFIDENTIAL

tuple ที่มีตัวเดียวต้องมี comma ต่อท้าย	`my_tuple = (1,)` *ได้ tuple ที่มีตัวเดียว (สังเกตที่ comma)* `my_tuple += (4,)` *ได้ (1, 4)* `my_tuple += 5` *ผิด นำ 5 ไปรวมกับ tuple ไม่ได้* `my_tuple += (5)` *ผิด เขียน (5) ก็เหมือน 5* `not_a_tuple = (1)` *ได้จำนวนเต็มธรรมดา*
tuple แก้ข้อมูลไม่ได้ tuple ไม่มีบริการ append, insert, add, pop, remove, discard	`my_tuple[3] = 'B'` *ผิด* ถ้าจะแก้ข้อมูล ต้องสร้างใหม่ `my_tuple = my_tuple[:3] + ('B',) + my_tuple[4:]`

การอ้างถึงข้อมูลใน dict ที่ไม่มีมาก่อนจะผิด	`D = {'Name':'Tom', 'Age':39}`
	`print(D['Gender'])` ผิดเพราะอ้างได้แค่ 'Name' และ 'Age'
	หรือ ต้องการนับจำนวนตัวอักษรภาษาอังกฤษแต่ละตัวว่าปรากฏกี่ครั้งในสตริง t
	`c = dict()`
	`for e in t :`
	`  c[e] += 1  # ผิด เพราะอาจไม่มี key e ใน t`
	ต้องเปลี่ยนเป็น
	`for e in t :`
	`  if e not in c :`
	`  c[e] = 1`
	`  else :`
	`  c[e] += 1`

key ของ dict ห้ามเป็น list, dict หรือ set	`my_dict = {}` `my_dict[[1,2,3]] = 'list'` *ผิด* *เพราะ [1,2,3] ใช้เป็น key ไม่ได้*

การเก็บคู่อันดับใน dict อาจจะไม่เรียงลำดับตามลำดับการใส่ข้อมูล	`D = {}`
	`D[1] = 1.00; D[2] = 2.00`
	`for k in D.keys():`
	`  # อาจได้ 1 แล้ว 2 หรือ 2 แล้ว 1 ก็ได้ ไม่แน่นอน`

ใช้วงวนหา key เพื่อให้ได้ value ที่คู่กัน ทำแบบนี้ไม่ได้ใช้ความสามารถของ dict เลย	ต้องการหา value ของ key ที่มีค่าเท่ากับ key1 ใน d ไม่ควรเขียนแบบข้างล่างนี้

```
for k,v in d.items() :
  if k == key1 :
    print('value =', v)
    break
else :
  print('Not found')
```

```
for k in d.keys() :
  if k == key1 :
    print('value =', d[k])
    break
else :
  print('Not found')
```

เขียนแบบข้างล่างนี้ง่ายกว่า และเร็วกว่ามาก

```
if key1 in d :
  print( 'value =', d[key1] )
else :
  print('Not found')
```

วงเล็บปีกกาว่าง คือ dict ไม่ใช่ set	= {} *เป็นการสร้าง dict ว่าง ๆ*
การเก็บข้อมูลใน set อาจจะไม่เรียงลำดั ตามลำดับการใส่ข้อมูล	S = {1, 2} print(S) *อาจได้ {1, 2} หรือ {2, 1} ก็ได้ ไม่แน่นอน*
การสร้างเซตจากสตริง ต่างจากการสร้าง เซต จากลิสต์ของสตริง	set_1 = {'Hello'} *ได้เซต {'Hello'}* set_2 = set('Hello') *ได้เซต {'H', 'e', 'l', 'o'}* set_3 = set(['Hello', 'World']) *ได้เซต {'Hello', 'World'}* การเขียน s = set(q) เหมือนการเขียน s = set() for e in q : s.add(e)
ห้ามเก็บ list, dict หรือ set ใน set	my_set = {1} หรือ my_set.add((1,2)) *ทำได้* my_set.add([2,3,4]) *ผิด*
การใช้ตัวดำเนินการของ set ไม่เปลี่ยนค่าในเซตเดิม	a = {1, 2, 3} b = {2, 3, 4} c = a.union(b) ได้ c = {1, 2, 3, 4} แต่ค่าของ a = {1, 2, 3} และ b = {2, 3, 4} ไม่เปลี่ยนแปลง
ใช้ index เพื่อเข้าใช้ข้อมูลใน set ไม่ได้ เพราะข้อมูลใน set ไม่มีลำดับ ไม่มี index	A = {'one', 'two', 'three'} for i in range(len(A)) : print(A[i]) *แบบนี้ผิด* ควรใช้ for...in แทน for e in A : print(e)

แบบฝึกหัด

Problem	Code
ลองคิดดูว่าการจะตอบคำถามแบบนี้ ควรเก็บข้อมูลด้วยอะไร 1) ถามเกรดวิชา comp prog จากรหัสนิสิต 2) มีรายชื่อนิสิตที่ลงทะเบียนเรียนวิชา comp prog เพื่อถามว่านิสิตที่มีรหัส x ลงทะเบียน comp prog หรือไม่ 3) ให้ชื่อภาควิชา แล้วถามว่า มีนิสิตคนใดอยู่ภาควิชานั้นบ้าง 4) ต้องการเก็บข้อมูลรุ่นโทรศัพท์มือถือที่เขาเคยใช้ เรียงจากอดีตจนถึงปัจจุบัน ของนิสิตคนหนึ่ง 5) อยากนับว่าหมายเลขโทรศัพท์ส่วนใหญ่ลงท้ายด้วยเลขอะไร	

CONFIDENTIAL

Problem	Code
Input: รับจำนวนเต็มบวก 1 จำนวนจากแป้นพิมพ์เก็บใน x Process: สร้าง tuple ของจำนวนเต็มคู่ตั้งแต่ 0 และน้อยกว่า x Output: tuple ที่สร้าง เช่น รับค่า `10` ให้แสดงผล `(0, 2, 4, 6, 8)`	
Input: รับจำนวนเต็มบวก 1 จำนวนจากแป้นพิมพ์ เก็บใน x Process: สร้าง tuple ของการแยกหลักของ x Output: tuple ที่สร้าง เช่น รับค่า `12803` ให้แสดงผล `(1, 2, 8, 0, 3)`	
Input: รับสตริงจากแป้นพิมพ์ เก็บใน x Process: สร้าง dict แสดงการนับตัวอักษรของ x Output: dict ที่สร้าง (ไม่สนใจลำดับที่แสดงผล) เช่น รับค่า `book` ให้แสดงผล `{'b':1, 'k':1, 'o':2}`	
Input: รับสตริง 2 บรรทัดจากแป้นพิมพ์ เก็บใน x และ y Process: สร้าง set ของตัวอักษรใน x และ set ของตัวอักษรใน y จากนั้นนำมาหาตัวอักษรที่ปรากฏในทั้งสองสตริง Output: set ของ ตัวอักษรที่ปรากฏในทั้งสองสตริง (ไม่สนใจลำดับที่แสดงผล) เช่น รับค่า `book` และ `bank` ให้แสดงผล `{'b', 'k'}`	

CONFIDENTIAL

ตัวอย่างการแก้โจทย์ปัญหา

ทะเบียนนิสิต

จงเขียนโปรแกรมรับรายการของข้อมูล ซึ่งประกอบด้วยชื่อของนิสิตและคณะที่นิสิตคนนั้นอยู่ จากนั้นจะกำหนดชื่อคณะมาให้จำนวนหนึ่ง ให้ตอบว่านิสิตในคณะเหล่านั้นมีชื่ออะไรบ้าง ถ้ามีชื่อซ้ำกัน ให้ตอบเพียงครั้งเดียว

▶ ข้อมูลนำเข้า

บรรทัดแรกมีจำนวนเต็มบวก n คือจำนวนรายการข้อมูลทั้งหมด
n บรรทัดต่อมา แต่ละบรรทัดประกอบด้วย ชื่อของนิสิตและคณะที่นิสิตอยู่ โดยคั่นด้วยช่องว่าง
บรรทัดสุดท้ายจะเป็นรายชื่อคณะที่ต้องการถาม ถ้ามีหลายชื่อจะคั่นด้วยช่องว่าง

▶ ข้อมูลส่งออก

แสดงชื่อนิสิตในคณะเหล่านั้น โดยเรียงตามตัวอักษร ให้คั่นแต่ละชื่อด้วยช่องว่าง ถ้ามีชื่อซ้ำกัน ให้ตอบเพียงครั้งเดียว

▶ ตัวอย่าง

Input (จากแป้นพิมพ์)	Output (ทางจอภาพ)
3 Tom Engineering Pam Arts Jim Engineering Engineering	Jim Tom
5 Tom Engineering Pam Arts Jim Engineering Tom Arts Jenny Science Engineering Arts	Jim Pam Tom

ตัวอย่างการเขียนโปรแกรม

เนื่องจากโจทย์จะกำหนดชื่อคณะมาให้ และให้เราหาชื่อนิสิตที่อยู่ในคณะเหล่านั้น ดังนั้นเราควรเก็บข้อมูลโดยใช้ dict ซึ่งมี key คือชื่อคณะ และ value เป็น set ของชื่อนิสิต (เพราะในแต่ละคณะ มีนิสิตได้หลายคน) ตัวอย่างการเก็บข้อมูล เช่น

```
{
        'Engineering' : {'Tom', 'Jim'},
        'Arts'        : {'Pam', 'Tom'},
        'Science'     : {'Jenny'}
}
```

CONFIDENTIAL

ขอแบ่งการประมวลผลเป็นขั้นตอนทีละขั้นดังนี้

1. อ่านข้อมูลนำเข้าเก็บเป็น dict
2. รับรายการของชื่อคณะที่ต้องการถาม มาเก็บเป็น list
3. เนื่องจากเราต้องการพิมพ์ชื่อที่ไม่ซ้ำกัน ดังนั้นเราจะใช้ set มาช่วย ให้สร้าง set คำตอบเริ่มต้นเป็นเซตว่าง
4. วนลูปทีละคณะที่ต้องการถาม เข้านำชื่อนิสิตของคณะนั้นไปรวมกับเซตคำตอบที่มีอยู่เดิม
5. พิมพ์คำตอบ โดยเรียงลำดับชื่อตามตัวอักษร

โปรแกรม	คำอธิบาย
ขั้นตอนที่ 1	

`n = int(input())`
`fac2name = {}`
 	
`for i in range(n):`
`  name,fac = input().split()`
`  fac2name[fac] = {name}`
	อ่านค่า n ซึ่งเป็นจำนวนข้อมูลเข้ามา
	สร้าง dict ว่าง ๆ ชื่อ fac2name ซึ่งจะใช้เก็บว่า คณะนี้มีนิสิตชื่ออะไรบ้าง จากนั้นวนลูป n รอบ เพื่อเก็บข้อมูลคู่ (key, value) คือ ชื่อคณะและ set ของชื่อนิสิต แต่ตัวอย่างนี้ผิด เพราะแต่ละ key ของ dict จะเก็บ value ได้แค่ค่าเดียว ถ้าใส่ค่าแบบนี้จะทำให้ค่าที่เก็บใหม่ไปทับค่าเดิม

`n = int(input())`
`fac2name = {}`
 	
`for i in range(n):`
`  name,fac = input().split()`
`  fac2name[fac].add(name)`
	เปลี่ยนมาใช้ add เพิ่มใน value ของ dict แต่การเก็บค่าแบบนี้ยังผิดอยู่ เพราะไม่ได้ตั้งค่าเริ่มต้นของ dict ไว้ก่อน จึงทำให้ add ไม่ได้

`n = int(input())`
`fac2name = {}`
 	
`for i in range(n):`
`  name,fac = input().split()`
`  if fac in fac2name:`
`  fac2name[fac].add(name)`
`  else:`
`  fac2name[fac] = {name}`
	แบบที่ถูกต้อง ต้องมีการตรวจสอบก่อนว่า dict ของเรามี key นี้เก็บไว้หรือยัง ถ้ายังไม่มี ให้สร้าง set ขึ้นมาก่อน แต่ถ้ามีแล้ว จะสามารถให้คำสั่ง add เพิ่มได้

ขั้นตอนที่ 2	
`ask_fac = input().split()`	รับข้อมูลรายชื่อคณะที่ต้องการถาม มาเก็บไว้ใน list ชื่อว่า ask_fac
ขั้นตอนที่ 3	
`ans_set = {}`	สร้างเซตคำตอบเริ่มต้นเป็นเซตว่าง แต่แบบนี้ผิด เพราะจะได้เป็น dict ว่างแทน
`ans_set = set()`	สร้างแบบนี้ถึงจะได้เซตว่างที่ถูกต้อง

CONFIDENTIAL

โปรแกรม	คำอธิบาย
ขั้นตอนที่ 4	
`for f in ask_fac:` `  ans_set.union(fac2name[f])`	นำชื่อนิสิตในคณะมาเพิ่มในเซต `ans_set` แบบ `union` แต่แบบนี้ผิด เพราะการใช้คำสั่ง `union` แบบนี้ไม่ได้ทำให้ค่าของ `ans_set` เปลี่ยนไปด้วย
`for f in ask_fac:` `  ans_set = ans_set.union(fac2name[f])`	การใช้คำสั่ง `union` แบบนี้จะทำให้ค่าใน `ans_set` เปลี่ยน แต่แบบนี้ยังผิดอยู่ เพราะคณะที่ถามมาอาจจะไม่มีอยู่ใน `dict` ของเราก็ได้ ต้องตรวจสอบก่อน

`for f in ask_fac:`
`  if f in fac2name:`
`  ans_set = ans_set.union(fac2name[f])`
	แบบนี้ถูกต้องแล้ว

ขั้นตอนที่ 5	
`print(' '.join(ans_set.sort()))`	พิมพ์ค่าใน `ans_set` โดยเรียงลำดับตามตัวอักษร แต่แบบนี้ผิด เพราะ `sort()` ใช้กับ `set` ไม่ได้
`print(' '.join(list(ans_set).sort()))`	ใช้วิธีแปลงเป็น `list` แล้วค่อยใช้ `sort()` แบบที่แสดงทางซ้ายนี้ก็ไม่ได้ เพราะ `list(ans_sort).sort()` เรียงลำดับได้ แต่ไม่ได้คืนค่าอะไรกลับไปให้ `join`

`ans_list = list(ans_set)`
`ans_list.sort()`
`print(' '.join(ans_list))`
	ต้องแปลงเป็น `list` เก็บใส่ในตัวแปร แล้วค่อยเรียก `sort` กับตัวแปร แล้วส่งตัวแปรนั้นให้ `join` ไปใช้

`print(' '.join(sorted(ans_set)))`	หรือใช้ `sorted` แทน เพราะ `sorted` รับ `set` ได้ และให้ผลเป็นลิสต์ที่เรียงลำดับแล้วกลับคืนมาส่งให้ `join`

CONFIDENTIAL

ตัวอย่างโจทย์ปัญหา

Union & Intersection

เขียนโปรแกรมเพื่อหา union และ intersection ของเซตที่กำหนด

▶ ข้อมูลนำเข้า

บรรทัดแรก ระบุจำนวนเต็ม n แทนจำนวนเซต

n บรรทัดถัดมา ระบุเซตของจำนวนเต็มบรรทัดละ 1 เซต โดยระบุจำนวนเต็มที่อยู่ในเซต คั่นด้วยช่องว่าง

▶ ข้อมูลส่งออก

บรรทัดแรก แสดงขนาดของเซตที่เกิดจากการ union ทุกเซต

บรรทัดที่ 2 แสดงขนาดของเซตที่เกิดจากการ intersect ทุกเซต

▶ ตัวอย่าง

Input (จากแป้นพิมพ์)	Output (ทางจอภาพ)
3 1 2 1 2 3 1 2 1 2 1 2 3 2 5 4 3	5 2
6 100 1000 101 123 200 201 -1 -2 -3	9 0
6 -1 0 1 -1 1 0 0 -1 1 0 1 -1 1 -1 0 1 0 -1	3 3

CONFIDENTIAL

อักษรสองตัวหน้าที่พบมากในประเภทของคำภาษาอังกฤษ

จากข้อมูลคำศัพท์ภาษาอังกฤษแยกตามประเภทของคำ จงเขียนโปรแกรมเพื่อหาตัวอักษรสองตัวแรกยอดฮิตของคำศัพท์ที่เป็นข้อมูลนำเข้า จำนวนคำศัพท์ และรายการคำศัพท์ที่ขึ้นต้นด้วยอักษรสองตัวแรกยอดฮิตนี้

▶ ข้อมูลนำเข้า

บรรทัดแรก บอกจำนวนรายการข้อมูลที่ต้องอ่านเข้ามา

บรรทัดที่เหลือ เป็นรายการข้อมูล โดยข้อมูลแรกเป็นประเภทของคำศัพท์ ข้อมูลที่สองเป็นคำศัพท์ คั่นด้วยเครื่องหมายแท็บ

▶ ข้อมูลส่งออก

ให้พิมพ์อักษรสองตัวหน้ายอดฮิต จำนวนคำศัพท์ และรายการคำศัพท์ที่ขึ้นต้นด้วยตัวอักษรสองตัวหน้ายอดฮิตนี้ พร้อมประเภทของคำศัพท์ คั่นด้วยเว้นวรรค โดยเรียงลำดับคำตามลำดับเดียวกันกับข้อมูลนำเข้า ถ้ามีอักษรสองตัวที่ปรากฏบ่อยมากที่สุดเท่ากัน ให้เลือกอักษรสองตัวแรกที่มาก่อนเรียงตามพจนานุกรม

▶ ตัวอย่าง

Input (จากแป้นพิมพ์)	Output (ทางจอภาพ)
10 Adjective good Noun goose Verb wrap Verb write Noun wrinkle Noun wreck Noun wrangler Noun hall Adjective happy Noun hobby	wr 5 wrap Verb write Verb wrinkle Noun wreck Noun wrangler Noun
6 Adjective good Verb wrap Verb write Noun hall Noun hobby Noun goose	go 2 good Adjective goose Noun

CONFIDENTIAL

ใครเรียนอะไร

ให้อ่านข้อมูลจากแป้นพิมพ์ โดยอ่านข้อมูลเป็นบรรทัด แต่ละบรรทัดมีรหัสนักเรียนคั่นด้วยเว้นวรรค ตามด้วยรหัสวิชา (อาจมีมากกว่า 1 วิชา) เมื่อพบว่า รหัสนักเรียนเป็น -1 ให้หยุดอ่าน จากนั้นให้อ่านรหัสวิชาสองรหัสวิชา แล้วให้แสดงจำนวนนักเรียนที่เรียนทั้งสองวิชานั้น จำนวนนักเรียนที่เรียนวิชาเดียวเท่านั้น และจำนวนนักเรียนทั้งหมดซึ่งเรียนวิชาใดวิชาหนึ่งหรือทั้งสองวิชา

▶ <u>ข้อมูลนำเข้า</u>

รหัสนักเรียนคั่นด้วยเว้นวรรค ตามด้วยรหัสวิชา (อาจมีมากกว่า 1 วิชา คั่นด้วยเว้นวรรค)
รับประกันว่าจะไม่มีนักเรียนที่รหัสซ้ำกัน และนักเรียน 1 คนจะไม่มีรหัสวิชาซ้ำกัน
เมื่อใส่ข้อมูลนักเรียนครบทุกคนแล้ว บรรทัดต่อไปจะเป็นข้อมูล -1
บรรทัดสุดท้ายเป็นรหัสวิชาสองรหัสวิชา คั่นด้วยเว้นวรรค

▶ <u>ข้อมูลส่งออก</u>

มี 3 จำนวนคั่นด้วยเว้นวรรค เรียงลำดับดังนี้
จำนวนนักเรียนที่เรียนทั้งสองวิชานั้น
จำนวนนักเรียนที่เรียนวิชาใดวิชาหนึ่งเท่านั้น
จำนวนนักเรียนซึ่งเรียนวิชาใดวิชาหนึ่งหรือทั้งสองวิชา

▶ <u>ตัวอย่าง</u>

Input (จากแป้นพิมพ์)	Output (ทางจอภาพ)
001 c001 c002 c003 002 c002 c003 c004 003 c003 c005 -1 c002 c003	2 1 3
5931111121 2110101 2109101 5932222221 2109101 -1 2110101 5500101	0 1 1

▶ <u>คำแนะนำ</u> (ถ้ารู้วิธีทำแล้ว ไม่ต้องอ่านก็ได้)

ควรเก็บข้อมูลด้วย dict ที่มี key คือรหัสวิชา และ value เป็นเซตของนักเรียน (ลองคิดดูว่าทำไม) เช่น ตัวอย่างที่ 1 จะได้ dict ดังนี้

```
{
   'c001':{'001'},
   'c002':{'001', '002'},
   'c003':{'001', '002', '003'},
   'c004':{'002'},
   'c005':{'003'}

}
```

CONFIDENTIAL

CONFIDENTIAL

สรุปเนื้อหา

การใช้งานฟังก์ชัน

- อาจเรียกว่า subprogram หรือ subroutine
- เป็นการแยกส่วนคำสั่งที่ซ้ำ ๆ กัน หรือเข้าใจยาก ออกมาจากโปรแกรมหลัก
- ทำให้โปรแกรมอ่านง่าย เข้าใจง่าย
- ทำให้โปรแกรมหลักเรียกใช้ฟังก์ชันได้ โดยไม่ต้องเขียนคำสั่งหลายรอบ
- ฟังก์ชันควรมีหน้าที่การทำงานชัดเจน เช่น ฟังก์ชันหาค่าเฉลี่ยของจำนวนในลิสต์ ฟังก์ชันกลับสตริง เป็นต้น
- ต้องเขียนฟังก์ชันไว้ก่อนส่วนที่จะเรียกใช้

องค์ประกอบของฟังก์ชัน

- ชื่อฟังก์ชัน มีข้อกำหนดเหมือนการตั้งชื่อตัวแปร
- ค่าที่รับเข้ามา หรือ "พารามิเตอร์" (ไม่จำเป็นต้องมีก็ได้)
- การคืนค่าจากฟังก์ชันด้วยคำสั่ง `return` (ไม่จำเป็นต้องมีก็ได้)

```
def average3(x,y,z):
    s = x+y+z
    return s/3
```

การคืนการทำงานจากฟังก์ชันและการคืนค่าจากฟังก์ชัน

- สามารถใช้คำสั่ง `return` ได้หลายที่ในฟังก์ชัน
- เมื่อทำคำสั่ง `return` แล้ว จะหยุดการทำงานของฟังก์ชันทันที และกลับไปทำงานต่อหลังจุดที่เรียกใช้ฟังก์ชัน
- ฟังก์ชันคืนค่าได้ 1 ค่าเท่านั้น ถ้าต้องการคืนหลายค่า ให้ใช้ `tuple` เช่น `return (answer1, answer2)`
- หากคำสั่งสุดท้ายของฟังก์ชันไม่ใช่ `return` ระบบจะเพิ่มคำสั่ง `return` (ไม่คืนค่าใด ๆ) ที่ท้ายฟังก์ชัน
- คำสั่ง `return` ที่ไม่ได้กำหนดให้คืนค่าใด ๆ ระบบจะคืนค่า `None` (`None` เป็นค่าพิเศษในระบบ ไม่ใช่สตริง `'None'`)

ตัวแปรในฟังก์ชัน (local variables)

- ตั้งชื่อซ้ำกับตัวแปรในฟังก์ชันอื่นได้ ถือว่าเป็นคนละตัวแปรกัน
- เรียกใช้ตัวแปรในฟังก์ชันอื่นไม่ได้
- หากส่งตัวแปรประเภท `int, float, string, boolean` เข้ามาในฟังก์ชัน จะถือว่าเป็นคนละตัวกัน
- หากส่งตัวแปรประเภท `list, dict, set` เข้ามาในฟังก์ชัน
  ถ้ามีการแก้ค่าในฟังก์ชัน ตัวแปรเดิมของผู้เรียกฟังก์ชันจะเปลี่ยนค่าด้วย

ฟังก์ชันเวียนเกิด

- ฟังก์ชันเวียนเกิดมี 2 ส่วนคือ ส่วนการคำนวณแบบพื้นฐาน และส่วนที่มีการเรียกซ้ำ

```
def factorial(n):
    if n < 2: return 1
    return n * factorial(n-1)
```

- บางครั้งเขียนง่ายกว่า loop เหมาะกับการทำซ้ำที่ไม่รู้จำนวนรอบ
- ทำงานช้ากว่า loop และใช้หน่วยความจำมากกว่า

การแปลงความสัมพันธ์ทางคณิตศาสตร์เป็นฟังก์ชันเวียนเกิด

- เขียนกรณีพื้นฐานก่อน คือกรณีที่รู้คำตอบทันที ไม่มีการเรียกซ้ำ
- แล้วค่อยเขียนกรณีที่ต้องเรียกซ้ำ

$$f_n = \begin{cases} 0, & n = 0 \\ 1, & 1 \le n \le 2 \\ f_{n-1} + f_{n-2}, & otherwise \end{cases}$$

```
def f(n):
    if n == 0: return 0
    if 1 <= n <= 2: return 1
    return f(n-1) + f(n-2)
```

เรื่องผิดบ่อย

เขียนฟังก์ชันไว้หลังส่วนที่เรียกใช้	`print(median(3,1,2))`	*ผิด เพราะหาฟังก์ชัน* median *ไม่เจอ ต้องย้ายฟังก์ชันขึ้นไปไว้ก่อนบรรทัดนี้*
	`def median(x,y,z):` `  return (x+y+z)-min(x,y,z)-max(x,y,z)`	
ตั้งชื่อฟังก์ชันซ้ำกันเอง หรือซ้ำกับชื่อตัวแปร	`def average(x,y):` `  return (x+y)/2`	average *เป็นฟังก์ชันที่รับ 2 ตัวแปร*
	`def average(x,y,z):` `  return (x+y+z)/3`	average *กลายเป็นฟังก์ชันที่รับ 3 ตัวแปร*
	`print(average(1,2,3))`	*ยังเรียกได้อยู่*
	`print(average(4,5))`	*ผิด เพราะต้องใช้แบบ 3 พารามิเตอร์*
	`average = 0`	average *กลายเป็นตัวแปรแล้ว*
	`print(average(1,2,3))`	*ผิด เพราะ* average *เป็นตัวแปรแล้ว*

CONFIDENTIAL

ถ้าพารามิเตอร์ของฟังก์ชันเป็น `list`, `set` หรือ `dict` การเปลี่ยนแปลงข้อมูลที่เก็บในพารามิเตอร์นี้ จะส่งผลให้ข้อมูลที่เก็บในตัวแปรของผู้เรียกที่ส่งมาให้พารามิเตอร์เปลี่ยนแปลงด้วย (เพราะเป็นที่เก็บตัวเดียวกัน)
`def f(x) :`
`  ...`
`  # ลิสต์ x เปลี่ยนแปลง ลิสต์ y จะเปลี่ยน`
`  x.append(e)`
`  x[i] = e`
`  x[:] = [0,0,0]`
`  x.pop(0)`
`  ...`
`f(y)` ←
	`def sum_double(x):`
	`  for i in range(len(x)):`
	`  x[i] *= 2  # x เป็นที่เก็บเดียวกับตัวแปรที่ส่งให้ x`
	`  return sum(x)`
	
	`y = [1, 2, 3]`
	`print(sum_double(y))  ได้ 12`
	`print(y)  ได้ [2, 4, 6]`

ถ้าตั้งค่าใหม่ให้กับพารามิเตอร์ของฟังก์ชันจะไม่ส่งผลถึงตัวแปรของผู้เรียกที่ส่งมาให้พารามิเตอร์
`def f(x) :`
`  ...`
`  # ตัวแปร x เปลี่ยนค่า ลิสต์ y ไม่เปลี่ยน`
`  x = [1,2,3]`
`  ...`
`f(y)` ←
	`def double(x) :`
	`  x *= 2`
	
	`def sum_double(x):`
	`  x = [2*e for e in x] # x ถูกเปลี่ยนเป็นลิสต์ตัวใหม่แล้ว`
	`  return sum(x)`
	
	`a = 8`
	`double(a)`
	`print(a)  ได้ 8 เหมือนเดิม ไม่ใช่ 16`
	`y = [1, 2, 3]`
	`print(sum_double(y))  ได้ 12`
	`print(y)  จะได้ [1, 2, 3]`

ลืม `return` ผลลัพธ์	`def square(x):`
	`  x = x**2  ลืมคืนค่า จะทำให้ได้ None`
	
	`def clip(x):`
	`  if x > 255 :`
	`  return 255  กรณี x ไม่เกิน 255 จะได้ None`

ลืม ( ) เมื่อเรียกใช้ฟังก์ชัน	`def read_next_positive_int():`
	`  n = int(input())`
	`  while n <= 0 :`
	`  n = int(input())`
	`  return n`
	
	`a = read_next_positive_int  ผิด ต้องมี () หลังชื่อฟังก์ชัน`
	`a = read_next_positive_int()  ถูกต้อง`

CONFIDENTIAL

ส่งค่าหรือตัวแปรที่เก็บค่าผิดประเภทให้กับพารามิเตอร์ของฟังก์ชัน	`def f(n):` *n ต้องเป็นจำนวนเต็ม เพราะใช้ใน* `range`
	`    g = 1`
	`    for k in range(n) :`
	`        g = 1 + 1/(1+g)`
	`    return g`
	
	`a = int(input())`
	`b = f(a/2)` *ส่ง* `float` *ไป จะทำงานผิดในฟังก์ชัน* `f`
	`c = f(10*a)` *ส่งจำนวนเต็ม ทำงานได้ถูกต้อง*

ลืมกรณีพื้นฐานของ recursive	`def factorial(n):` `    return n * factorial(n-1)` *จะเรียกฟังก์ชันวนไปเรื่อย ๆ*

เขียนตรวจสอบกรณีพื้นฐานไว้ทีหลัง	`def factorial(n):`
	`    return n * factorial(n-1)`
	`    if n == 0: return 1` *บรรทัดนี้ไม่เคยทำงานเลย*

ถ้ามีการคำนวณค่า recursive ที่ซ้ำกัน ควรคำนวณทีเดียว แล้วเก็บไว้ในตัวแปร	`def f(n):`
	`    if n == 0: return 1`
	`    return f(n-1) + f(n-1)**2`
	มีการคำนวณซ้ำ โปรแกรมจะช้า
	
	`def f(n):`
	`    if n == 0: return 1`
	`    x = f(n-1)`
	`    return x + x**2`
	คำนวณครั้งเดียว จะทำงานเร็วกว่า

CONFIDENTIAL

แบบฝึกหัด

Problem	Code
ชื่อฟังก์ชัน: `f1` Parameter: รับข้อมูล a เป็นสตริง และ รับ b เป็นจำนวนเต็ม Process: พิมพ์ a ออกทางหน้าจอจำนวน b บรรทัด Return: ไม่ต้องคืนค่า	
ชื่อฟังก์ชัน: `f2` Parameter: รับข้อมูล a เป็นสตริง และ รับ b เป็นจำนวนเต็ม Process: สร้างลิสต์ที่เก็บสตริง a จำนวน b ตัว Return: ลิสต์ที่สร้าง	

Problem	Code
<u>ชื่อฟังก์ชัน</u>: g <u>Parameter</u>: รับจำนวนจริงสี่จำนวน m b n และ c <u>Return</u>: คืนค่าจุดตัดของเส้นตรง y = mx+b และ y = nx+c เป็น tuple ของจุดตัด (x,y) หากเป็นเส้นตรงที่ขนานกัน ให้คืนค่าจำนวนเต็ม 1 หากเป็นเส้นตรงเดียวกัน ให้คืนค่าจำนวนเต็ม 2	
<u>ชื่อฟังก์ชัน</u>: h <u>Parameter</u>: รับลิสต์ของจำนวนเต็ม x <u>Return</u>: คืนลิสต์ใหม่ ที่มีสมาชิกจาก x เฉพาะที่เป็นเลขคู่เท่านั้น โดยห้ามแก้ไขค่าในลิสต์ x	
เขียนฟังก์ชันเวียนเกิดเพื่อคำนวณค่าดังนี้ $a_n = \begin{cases} 1, & n = 0 \\ -2, & n = 1 \\ a_{n-2} \times n, & otherwise \end{cases}$	

เขียนฟังก์ชันเวียนเกิดเพื่อคำนวณค่าดังนี้
`k(2n)  = k(n)+(k(n)%10) if n > 0`
`k(2n+1) = k(n-1)*n  if n > 0`
`k(0)  = 1`
`k(1)  = 2`	

เขียนฟังก์ชันเวียนเกิดเพื่อคำนวณค่าดังนี้ $s_{i,k} = \begin{cases} 0, & i \geq k \\ k + t_{i+1,k}, & i < k \end{cases}$ $t_{j,k} = \begin{cases} 0, & j \geq k \\ j + s_{j,k-1}, & j < k \end{cases}$	
<u>ชื่อฟังก์ชัน</u>: `is_palindrome` <u>Parameter</u>: รับข้อมูล s เป็นสตริง <u>Process:</u> เขียนฟังก์ชันเวียนเกิดเพื่อตรวจสอบว่า s เป็น palindrome (อ่านจากหน้าไปหลังเหมือนหลังไปหน้า) หรือไม่ <u>Return</u>: ถ้า s เป็น palindrome ให้คืนค่า `True` ถ้าไม่เป็น ให้คืนค่า `False`	

CONFIDENTIAL

ตัวอย่างการแก้โจทย์ปัญหา

Function Refactoring

กำหนดโปรแกรมคำนวณจำนวนวันตั้งแต่วันที่ A จนถึงวันที่ B มาให้ จงแยกเป็นฟังก์ชันต่อไปนี้ และเติมคำสั่งให้ครบสมบูรณ์

เส้นประ		ฟังก์ชัน `is_leap_year(y)` คืน `True` เมื่อปีคริสตศักราช `y` เป็นปีอธิกสุรทิน ถ้าไม่เป็นให้คืน `False`
เส้นไข่ปลา		ฟังก์ชัน `day_of_year(d,m,y)` คืนว่าวันที่ `d m y` เป็นวันที่ลำดับที่เท่าใดของปีคริสตศักราช `y`
เส้นประยาว		ฟังก์ชัน `days_in_year(y)` คืนจำนวนวันในปีคริสตศักราช `y`

▶ ข้อมูลนำเข้า

มี 2 บรรทัด บรรทัดแรกเป็นวันที่ A ในรูปแบบ `d1 m1 y1` เป็นปีคริสตศักราช
บรรทัดที่ 2 เป็นวันที่ B ในรูปแบบ `d2 m2 y2` เป็นปีคริสตศักราช
รับประกันว่าวันที่ทั้งสองวันจะเป็นวันที่ที่ถูกต้อง และวันที่ A จะมาก่อนวันที่ B เสมอ และอยู่คนละปีกัน

▶ ข้อมูลส่งออก

จำนวนวันระหว่างวันที่ A และ B โดยรวมวันที่ A และ B ด้วย

▶ ตัวอย่าง

Input (จากแป้นพิมพ์)	Output (ทางจอภาพ)
`1 1 2018` `1 1 2020`	`731`
`25 12 1999` `9 3 2000`	`76`

CONFIDENTIAL

โปรแกรมเริ่มต้น

```
d1,m1,y1 = [int(e) for e in input().split()]
d2,m2,y2 = [int(e) for e in input().split()]

# คำนวณว่าวันที่ A เป็นวันที่เท่าไรของปี
D = [31,28,31,30,31,30,31,31,30,31,30,31]
if y1%4==0 and (y1%100!=0 or y1%400==0):     # ตรวจสอบปีอธิกสุรทิน
    D[1] += 1
A = 0
for i in range(m1-1):
    A += D[i]
A += d1

# คำนวณว่าวันที่ B เป็นวันที่เท่าไรของปี
D = [31,28,31,30,31,30,31,31,30,31,30,31]
if y2%4==0 and (y2%100!=0 or y2%400==0):     # ตรวจสอบปีอธิกสุรทิน
    D[1] += 1
B = 0
for i in range(m2-1):
    B += D[i]
B += d2

# คำนวณจำนวนวันระหว่างปี A และปี B
C = 0
for i in range(y1+1,y2):
    # บวกด้วยจำนวนวันในปีนั้น ๆ
    if i%4==0 and (i%100!=0 or i%400==0):    # ตรวจสอบปีอธิกสุรทิน
        C += 366
    else:
        C += 365

print( ?? )

# (จำนวนวันในปี A) - (ลำดับที่ของวัน A) + 1 + (จำนวนวันระหว่างปีทั้งสอง) + (ลำดับที่ของวัน B)
```

CONFIDENTIAL

ตัวอย่างการเขียนโปรแกรม

โปรแกรม	คำอธิบาย
ฟังก์ชัน `is_leap_year`	

จากโปรแกรมในกล่องเส้นประ สามารถเขียนเป็นฟังก์ชันได้ดังนี้
`def is_leap_year(y):`
`  if y%4==0 and (y%100!=0 or y%400==0):`
`  return True`
`  else:`
`  return False`
	ฟังก์ชัน `is_leap_year` จะตรวจสอบว่าเป็นปีอธิกสุรทินหรือไม่ ตามเงื่อนไขในกล่องเส้นประ ถ้าตรงตามเงื่อนไขให้คืนค่า `True` แต่ถ้าไม่ตรงให้คืนค่า `False`

หรือสามารถเขียนให้สั้นลงได้เป็นแบบนี้
`def is_leap_year(y):`
`  if y%4==0 and (y%100!=0 or y%400==0):`
`  return True`
`  return False`
	ปรับปรุง : ในกรณีนี้ เราสามารถตัดคำว่า `else` ออกได้ เนื่องจากถ้าเงื่อนไขเป็นจริงแล้วจะ `return True` และจบการทำงานฟังก์ชันทันที จะไม่มาทำงานที่บรรทัด `return False` อีก

หรือสามารถเขียนให้สั้นลงได้อีกเป็นแบบนี้
`def is_leap_year(y):`
`  return y%4==0 and (y%100!=0 or y%400==0)`
	ปรับปรุง : เนื่องจากประโยคตรวจสอบเงื่อนไขมีค่าความจริงเป็น `True` หรือ `False` อยู่แล้ว เราสามารถคืนผลการเปรียบเทียบนี้โดยตรงได้เลย

ฟังก์ชัน `day_of_year`	

จากโปรแกรมในกล่องเส้นไข่ปลา สามารถเขียนเป็นฟังก์ชันได้ดังนี้
`def day_of_year(d,m,y):`
`  D = [31,28,31,30,31,30,31,31,30,31,30,31]`
`  if y%4==0 and (y%100!=0 or y%400==0):`
`  D[1] += 1`
`  A = 0`
`  for i in range(m-1)`
`  A += D[i]`
`  A += d`
	คัดลอกโปรแกรมจากกล่องเส้นไข่ปลา อย่าลืมเปลี่ยนชื่อตัวแปรด้วย (`d1,m1,y1` เป็น `d,m,y`) แต่ฟังก์ชันนี้ยังไม่ถูกต้องเพราะลืมคืนค่า

`def day_of_year(d,m,y):`
`  D = [31,28,31,30,31,30,31,31,30,31,30,31]`
`  if y%4==0 and (y%100!=0 or y%400==0):`
`  D[1] += 1`
`  A = 0`
`  for i in range(m-1)`
`  A += D[i]`
`  A += d`
`  return A`
	เติมคำสั่ง `return A` ทำให้ฟังก์ชันทำงานได้ถูกต้องแล้ว

CONFIDENTIAL

โปรแกรม	คำอธิบาย

`def day_of_year(d,m,y):`
`    D = [31,28,31,30,31,30,31,31,30,31,30,31]`
`    if is_leap_year(y):`
`        D[1] += 1`
`    A = 0`
`    for i in range(m-1):`
`        A += D[i]`
`    A += d`
`    return A`
	ปรับปรุง: นำฟังก์ชัน `is_leap_year` ที่เขียนไว้แล้วมาใช้ได้

ฟังก์ชัน `days_in_year(y)`	

จากโปรแกรมในกล่องเส้นประยาว สามารถเขียนเป็นฟังก์ชันได้ดังนี้
`def days_in_year(y):`
`    if y%4==0 and (y%100!=0 or y%400==0):`
`        return 366`
`    else:`
`        return 365`
	คัดลอกโปรแกรมจากกล่องเส้นประยาว อย่าลืมเปลี่ยนชื่อตัวแปร และคืนค่าให้ถูกต้อง

`def days_in_year(y):`
`    if is_leap_year(y):`
`        return 366`
`    else:`
`        return 365`
	ปรับปรุง: นำฟังก์ชัน `is_leap_year` ที่เขียนไว้แล้วมาใช้

`def days_in_year(y):` `    return day_of_year(31,12,y)`	ปรับปรุง: นำฟังก์ชัน `day_of_year` ที่เขียนไว้แล้วมาใช้ โดยขอลำดับของวันที่ 31 ธันวาคม ปี y (กรณีนี้ได้โปรแกรมสั้นลง แต่อาจทำให้ฟังก์ชันเข้าใจยากขึ้น)
ส่วนโปรแกรมหลัก	

`d1,m1,y1 = [int(e) for e in input().split()]`
`d2,m2,y2 = [int(e) for e in input().split()]`
 	
`A = day_of_year(d1,m1,y1)`
`B = day_of_year(d2,m2,y2)`
 	
`C = 0`
`for i in range(y1+1,y2):`
`    C += days_in_year(i)`
 	
`print(days_in_year(y1) - A + 1 + C + B)`
	เปลี่ยนกล่องต่าง ๆ เป็นการเรียกฟังก์ชันและส่งค่าไปให้แต่ละฟังก์ชันให้ถูกต้อง

CONFIDENTIAL

ตัวอย่างโจทย์ปัญหา

Four Functions

จงเขียน 4 ฟังก์ชัน ให้ทำงานตามที่เขียนอธิบายกำกับแต่ละฟังก์ชัน ในโครงของโปรแกรมข้างล่างนี้

```
def make_int_list(x):
    # รับสตริง x มาแยกและแปลงเป็น int เก็บใน list แล้วคืนเป็นผลลัพธ์
    # เช่น x = '12 34 5' ได้ผลเป็น [12, 34, 5]


def is_odd(e):
    # คืนค่าจริงเมื่อ e เป็นจำนวนคี่  ถ้าไม่ใช่ คืนค่าเท็จ


def odd_list(alist):
    # คืน list ที่มีค่าเหมือน alist แต่มีเฉพาะตัวที่เป็นจำนวนคี่
    # เช่น alist = [10, 11, 13, 24, 25] จะได้ [11, 13, 25]


def sum_square(alist):
    # คืนผลรวมของกำลังสองของแต่ละค่าใน alist
    # เช่น alist = [1,3,4] ได้ผลเป็น (1*1 + 3*3 + 4*4) = 26


exec(input().strip())   # ต้องมีบรรทัดนี้เมื่อส่งไป Grader
```

▶ ข้อมูลนำเข้า

คำสั่งในการทดสอบฟังก์ชันที่เขียน

▶ ข้อมูลส่งออก

ผลที่ได้จากคำสั่งที่ป้อนเป็นข้อมูลนำเข้า

▶ ตัวอย่าง

Input (จากแป้นพิมพ์)	Output (ทางจอภาพ)
`print(make_int_list('1 2 3 4 5'))`	`[1, 2, 3, 4, 5]`
`print(is_odd(2222))`	`False`
`print(odd_list([1,2,3,4,5,6,7]))`	`[1, 3, 5, 7]`
`print(sum_square([1,1,2,3]))`	`15`

CONFIDENTIAL

Recursive C(n,k)

เราสามารถหาค่าของจำนวนวิธีในการเรียงสับเปลี่ยน C(n,k) ได้จากสมการด้านล่าง จงใช้สมการต่อไปนี้ในการเขียนโปรแกรมแบบ recursive เพื่อคำนวณค่า C(n,k)

$$C(n,k) = \begin{cases} C(n-1,k) + C(n-1,k-1), & \text{if } 0 < k < n \\ 1, & \text{if } k = 0 \text{ or } n = k \\ 0, & \text{otherwise} \end{cases}$$

▶ <u>ข้อมูลนำเข้า</u>

มี 2 บรรทัด ประกอบด้วยจำนวนเต็ม n และ k

▶ <u>ข้อมูลส่งออก</u>

มี 1 บรรทัด แสดงค่า C(n,k) ที่คำนวณได้

▶ <u>ตัวอย่าง</u>

Input (จากแป้นพิมพ์)	Output (ทางจอภาพ)
6 2	15
10 8	45
0 1	0
3 7	0
10 10	1

CONFIDENTIAL

Recursive SumList

โจทย์ข้อนี้สั้น ๆ จงเขียนฟังก์ชัน sumlist() ของโครงโปรแกรมข้างล่างนี้ sumlist รับ x เป็นลิสต์เก็บจำนวนเต็ม แล้วคืนผลรวมของจำนวนเต็มทุกตัวใน x โดย x เป็นลิสต์ที่ภายในเป็นลิสต์ซ้อนลิสต์กี่ชั้นก็ได้ ดังตัวอย่างที่แสดงข้างล่างนี้

หมายเหตุ : สามารถใช้คำสั่ง if type(x) == list เพื่อทดสอบว่า x เป็นข้อมูลประเภทลิสต์หรือไม่

```
def sumlist( x ):
    # ???





print(eval(input().strip()))  # do not remove this line
```

▶ ข้อมูลนำเข้า

ลิสต์ที่เก็บจำนวนเต็ม (อาจเป็นลิสต์แบบลิสต์ซ้อนลิสต์กี่ชั้นก็ได้)

▶ ข้อมูลส่งออก

ผลรวมของจำนวนเต็มทุกตัวในลิสต์ที่เป็นข้อมูลนำเข้า

▶ ตัวอย่าง

Input (จากแป้นพิมพ์)	Output (ทางจอภาพ)
sumlist([[],[[]]])	0
sumlist([1,1,2,2])	6
sumlist([1,[1],[2,[2],0,[0,0]]])	6
sumlist([0,[1,[2,[3,[4,5],6],7],8],[9,10]])	55

CONFIDENTIAL

CONFIDENT

สรุปเนื้อหา

```
import numpy as np
```

- NumPy เป็นคลังคำสั่งที่ให้บริการมากมายเกี่ยวกับการคำนวณทางวิทยาศาสตร์ โดยมีที่เก็บข้อมูลที่เรียกว่า อาเรย์ n มิติ (ndarray) มีไว้เก็บข้อมูลเพื่อการประมวลผลที่มีประสิทธิภาพมาก ๆ
- อาเรย์มีลักษณะคล้ายลิสต์ แต่สร้างแล้วเปลี่ยนขนาดไม่ได้
- ค่าในอาเรย์ทุกช่องต้องเป็นประเภทเดียวกันทั้งหมด เช่น เป็น `int` ทุกช่อง หรือ `float` ทุกช่อง (ผสมกันไม่ได้ ถ้าเป็นลิสต์ผสมได้)
- เราสร้าง เวกเตอร์ ได้ด้วยอาเรย์ 1 มิติ และสร้าง เมทริกซ์ ได้ด้วยอาเรย์ 2 มิติ

การสร้างอาเรย์ด้วยฟังก์ชัน

สร้างอาเรย์จาก ลิสต์ L	`np.array(L)`	หาขนาดของ อาเรย์ M	`M.shape`

`[1 0 2 3 -1]`	`np.array([1,0,2,3,-1])`	`[[2 3]`
		`[4 1]`
		`[7 5]]`
			`np.array([[2,3],[4,1],[7,5]])`

`[[0. 0. 0.]`
`[0. 0. 0.]]`
	`np.zeros((2,3))`	`[[1 1]`
		`[1 1]`
		`[1 1]]`
			`np.ones((3,2),int)`

`[[1 0 0 0]`
`[0 1 0 0]`
`[0 0 1 0]`
`[0 0 0 1]]`
	`np.identity(4,int)`	`[[1. 0. 0.]`
		`[0. 1. 0.]`
		`[0. 0. 1.]]`
			`np.identity(3)`

สร้างอาเรย์ที่ มีขนาดเท่ากับ อาเรย์ x และเป็น 0 (int) ทุกช่อง	`np.zeros_like(x,int)` (x อาจเก็บ `int` หรือ `float` ก็ได้)	สร้างอาเรย์ที่ มีขนาดเท่ากับ อาเรย์ y และเป็น 1.0 ทุกช่อง	`np.ones_like(y,float)` (y อาจเก็บ `int` หรือ `float` ก็ได้)
`[4 5 6 7 8 9]`	`np.arange(4,10)`	`[8. 7. 6. 5.]`	`np.arange(8.0,4.0,-1.0)` `np.arange(8,4,-1,float)`
`[2.3 2.33 2.36 2.39 2.42 2.45 2.48]`			`np.arange(2.3,2.5,0.03)`

CONFIDENTIAL

Element-wise operations, Broadcasting และ Indexing

- เป็นสามแนวคิดของ NumPy (ที่ต้องรู้ในวิชานี้) ที่ช่วยให้การประมวลผลอาเรย์ทำได้สะดวก
- การดำเนินการอาเรย์กับอาเรย์ (เช่น A+B) การดำเนินการหรือทำฟังก์ชันกับอาเรย์ จะเป็นแบบช่องต่อช่อง (element-wise) เช่น

  `np.array([1,2,3]) + np.array([1,1,1])` ได้ `np.array([2,3,4])`
  `1/(np.array([1,2,4]))` ได้ `np.array([1.0, 0.5, 0.25])`
- การดำเนินการอาเรย์กับอาเรย์ที่มีขนาดไม่เท่ากัน อาจมีการขยายอาเรย์ให้มีขนาดเท่ากันก่อน (broadcasting) โดยพิจารณามิติของทั้งสองอาเรย์จากขวาไปซ้าย ให้เป็นไปตามกฎการ broadcast (ขอนำเสนอด้วยตัวอย่าง)
- `A.shape = (2,3)`, `B.shape = (3,)` ดูจากขวามาซ้าย 3 เท่ากัน จึง broadcast B ให้มีขนาดเท่ากับ A ได้
  เช่น `[[1 2 3]  + [9 8 7]` broadcast แล้วกลายเป็น `[[1 2 3] + [[9 8 7] = [[10 10 10]`
  `[4 5 6]]` `[4 5 6]]  [9 8 7]]  [13 13 13]]`
- มิติไหนขนาดเป็น 1 ถือว่า มิตินั้น broadcast ได้ (แต่มิติอื่นที่ไม่ใช่ 1 ต้องเท่ากัน)
  - `A.shape = (2,3), B.shape = (2,1)` broadcast B ให้เป็น (2,3) มีขนาดเท่ากับ A
  - `A.shape = (1,3), B.shape = (2,1)` broadcast ทั้ง A และ B ให้เป็น (2,3)
    เช่น `[[1 2 3]] + [[6]` broadcast แล้วกลายเป็น `[[1 2 3] + [[6 6 6] = [[ 7  8  9]`
    `[7]]` `[1 2 3]]  [7 7 7]]  [ 8  9 10]]`
  - `A.shape = (2,3), B.shape = (1,)` broadcast B ได้ (ถ้า B เป็นสเกลาร์ ให้ถือว่ามีมิติเป็น (1,))
    เช่น `[[1 2 3]  + 9` broadcast แล้วกลายเป็น `[[1 2 3] + [[9 9 9] = [[10 11 12]`
    `[4 5 6]]` `[4 5 6]]  [9 9 9]]  [13 14 15]]`
  - `A.shape = (3,5), B.shape = (5,1)` broadcast B ไม่ได้ (ปรับ 1 ได้ แต่อีกมิติไม่เท่ากัน เลยทำไม่ได้)
- การอ้างอิงข้อมูลในอาเรย์ (Indexing ของ NumPy)
  - การอ้างอิงข้อมูลในอาเรย์ 1 มิติ
    - การเข้าใช้ข้อมูล
      - `V[k]` เหมือนการใช้ลิสต์ คือเลือกข้อมูลหนึ่งช่อง โดยที่ k เป็นจำนวนเต็ม
      - `V[a:b:c]` เหมือนการใช้ลิสต์ คือเลือกข้อมูลเป็นช่วง ได้ผลเป็นอาเรย์
    - การใส่ค่าใหม่
      - `V[k] = 99` เหมือนกับการใช้ลิสต์
      - `V[a:b:c] = d` แตกต่างจากลิสต์
        - ถ้า d เป็น `int`, `float` หรือลิสต์/อาเรย์ขนาด 1 ช่อง จะ broadcast ให้มีขนาดเท่ากับช่วงของ `a:b:c`
        - ถ้า d เป็นลิสต์/อาเรย์ขนาดมากกว่า 1 ช่อง ต้องมีขนาดเท่ากับขนาดของช่วง `a:b:c` เช่น
          - `V = np.zeros(5,int)`
          - `V[2:4] = 1` ทำได้, V เปลี่ยนเป็น `[0 0 1 1 0]`
          - `V[::2] = 3` ทำได้, V เปลี่ยนเป็น `[3 0 3 1 3]`
          - `V[::2] = [9]` ทำได้, V เปลี่ยนเป็น `[9 0 9 1 9]`
          - `V[::2] = [8,8,8]` ทำได้, V เปลี่ยนเป็น `[8 0 8 1 8]`
          - `V[1:5] = [1,2]` ทำไม่ได้, เพราะขนาดของข้อมูลไม่ตรงกัน
          - `V[7:7] = [1,2]` ไม่เกิดการเปลี่ยนแปลงใด ๆ

- การอ้างอิงข้อมูลในอาเรย์ 2 มิติ
  - สามารถอ้างอิงถึงตัวข้อมูล, อาเรย์ 1 มิติที่แทนแถว (row) ของข้อมูล, อาเรย์ 1 มิติที่แทนหลัก (column) ของข้อมูล หรืออาเรย์ย่อย 2 มิติที่แทนช่วงของแถวและหลัก ได้หลากหลายแบบ ดังนี้
    - `M[r,c]` ข้อมูล ณ แถวที่ r หลักที่ c
    - `M[r,:]` หรือ `M[r]` อาเรย์ 1 มิติของแถวที่ r ทั้งแถว
    - `M[:,c]` อาเรย์ 1 มิติของหลักที่ c ทั้งหลัก (ไม่ใช่อาเรย์ 2 มิติที่มี 1 หลัก)
    - `M[r1:r2,:]` หรือ `M[r1:r2]` อาเรย์ 2 มิติประกอบด้วย แถวที่ r1 ถึง r2-1
    - `M[:,c1:c2]` อาเรย์ 2 มิติประกอบด้วย หลักที่ c1 ถึง c2-1
    - `M[r1:r2,c1:c2]` อาเรย์ย่อย 2 มิติ ในช่วงแถวที่ r1 ถึง r2-1 และช่วงหลักที่ c1 ถึง c2-1
    - `M[r1:r2:a,c1:c2:b]` อาเรย์ย่อย 2 มิติ ในช่วงแถวและหลักที่กำหนดโดย r1:r2:a, c1:c2:b
    - เช่น ให้ `M = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])`

      `M[1:3]` ได้ `np.array([[5,6,7,8],[9,10,11,12]])`
      `M[1:2, 1:3]` ได้ `np.array([[6,7]])`
      `M[:, 2]` ได้ `np.array([3,7,11])`
      `M[1:, :2]` ได้ `np.array([[5,6],[9,10]])`
      `M[0:3:2]` ได้ `np.array([[1,2,3,4],[9,10,11,12]])`
      `M[0:3:2, 1:4:2]` ได้ `np.array([[2,4],[10,12]])`
      `M[::-1]` ได้ `np.array([[9,10,11,12],[5,6,7,8],[1,2,3,4]])`
  - สามารถให้ค่ากับหลาย ๆ ช่องในอาเรย์พร้อมกันได้ เช่น
    - `M[2:4, 1:4] = [[-1, 1, -1], [-1, 1, -1]]` ขนาดเท่ากันพอดี
      `M[2:4, 1:4] = [-1, 1, -1]` ขนาดไม่เท่า ระบบ broadcast ให้
    - `M[:, 0] = [1, 2, 3, 4]` M ต้องมี 4 แถว
    - `M[::2, ::3] = 0` broadcast 0

คำสั่ง dot (ใช้ว่า `x.dot(y)` หรือ `np.dot(x,y)` ก็ได้)

dot ใช้กับเวกเตอร์ คือการหา dot product	dot ใช้กับเมทริกซ์ คล้ายกับการคูณเมทริกซ์

`x = np.array([ 1, 2, 3, 4])`
`y = np.array([ 0, -1, 1, 2])`
`x.dot(y)` ได้ค่าเท่ากับ `y.dot(x)` เท่ากับ
`1*0 + 2*(-1) + 3*1 + 4*2 = 9`
	`x = np.array([[1,2],[3,4],[5,6]])` `x.shape` คือ (3,2)
	`y = np.array([[-2],[3]])` `y.shape` คือ (2,1)
	`z = np.array([[-2,3]])` `z.shape` คือ (1,2)
	`w = np.array([-2,3])` `w.shape` คือ (2,)
	
	`x.dot(y)` ได้ `np.array([[4],[6],[8]])`
	มีมิติเป็น (3,2) กับ (2,1) คูณได้ ได้อาเรย์ขนาด (3,1)
	`y.dot(x)` ทำไม่ได้ เพราะมิติไม่ถูกต้อง (2,1) กับ (3,2)
	`x.dot(z)` ทำไม่ได้ เพราะมิติไม่ถูกต้อง (3,2) กับ (1,2)
	`x.dot(w)` ได้ `np.array([4,6,8])`
	อันนี้แปลก (3,2) กับ (2,) คูณได้ ได้อาเรย์ขนาด (3,)

CONFIDENTIAL

ฟังก์ชันที่ระบุแกนได้, การ transpose และการเปรียบเทียบ (M = np.array([[1,2,3,4],[5,6,7,8]]))

- ฟังก์ชันที่ระบุแกนได้ เช่น sum, max, min, mean, std, argmax, argmin (คืนตำแหน่งตัวมาก/น้อยสุด)
- np.sum(M) ได้ผลรวมของทุกช่อง np.sum(M) ได้ 36
- np.sum(M,axis=0) ได้ผลรวมตามแนวตั้ง np.sum(M,axis=0) ได้ np.array([6,8,10,12])
- np.sum(M,axis=1) ได้ผลรวมตามแนวนอน np.sum(M,axis=1) ได้ np.array([10,26])
- matrix transpose ใช้ว่า M.T M.T ได้ np.array([[1,5],[2,6],[3,7],[4,8]])
- M > 3 ได้ array([[False,False,False,True],[True,True,True,True]])
- ใช้ np.sum นับจำนวน element ที่ตรงตามเงื่อนไขได้ np.sum(M > 3) ได้ 5
- ใช้เงื่อนไขในการเลือกบาง element ของอาเรย์ได้ M[M%2==0] ได้ np.array([2,4,6,8])

CONFIDENTIAL

เรื่องผิดบ่อย

ลืม import numpy	โดยทั่วไปมักใช้ import numpy as np คือการ import คลังคำสั่ง numpy และตั้งชื่อใหม่ว่า np จะได้เขียนสั้น ๆ
ลืมระบุประเภทของข้อมูลที่จะเก็บว่าเป็น int หรือ float	x = np.zeros((3,4)) # *ได้เป็น* float x = np.zeros((3,4),int) # *ได้เป็น* int x = np.zeros_like(y) # *ขึ้นกับประเภทข้อมูลของ* y
ถ้าเก็บค่า float ในเมทริกซ์ที่เก็บ int ระบบจะปัดค่าหลังจุดทศนิยมทิ้ง	x = np.array([1,2,3],int) x[0] = 4.8 # *ได้* x *เท่ากับ* np.array([4,2,3])
ใช้ np.array กับ np.ndarray ผิด	ต้องการสร้างเมทริกซ์ขนาด 4×5 แต่เขียน a = np.array((4,5)) # *ได้* [4,5] ต้องการสร้างเวกเตอร์ [2,3] แต่เขียน a = np.ndarray([2,3]) # *ได้เมทริกซ์ขนาด* 2x3
การทำงานของเวกเตอร์ไม่เหมือนเมทริกซ์ (อาเรย์ 1 มิติ vs 2 มิติ)	x = np.array([1,2,3]) ได้อาเรย์ 1 มิติ 3 ช่อง y = np.array([[1,2,3]]) ได้อาเรย์ 2 มิติ 1 แถว 3 คอลัมน์ x.shape ได้ (3,) y.shape ได้ (1,3) x.T ได้อาเรย์เหมือนกับ x x.T.shape ได้ (3,) y.T ได้อาเรย์ 3 แถว 1 คอลัมน์ y.T.shape ได้ (3,1) print(x.dot(x.T)) ได้ 14 print(x.T.dot(x)) ก็ได้ 14 print(y.dot(y.T)) ได้ array([[14]]) print(y.T.dot(y)) ได้ np.array([[1,2,3],[2,4,6],[3,6,9]])

broadcasting & element-wise operations	`x = np.array([[1,2,3],[4,5,6]])` `y = x+2` # [[3,4,5],[6,7,8]] `y = x+[1,2,3]` # [[2,4,6],[5,7,9]] `y = x+np.array([1,2,3])` # [[2,4,6],[5,7,9]] `y = x+np.array([[1,2,3]])` # [[2,4,6],[5,7,9]] `y = x+np.array([[1],[2],[3]]).T` # [[2,4,6],[5,7,9]] `y = x+np.array([[1],[2],[3]])` # *ผิด* `y = x+np.array([[1,2,3]]).T` # *ผิด* `y = x+np.array([[1,2]]).T` # [[2,3,4],[6,7,8]]

CONFIDENTIAL

แบบฝึกหัด

(พยายามเขียนให้สั้น ๆ และทำงานได้เร็วสุด ๆ)

Problem	Code
สมมติว่ามีอาเรย์ 2 มิติ `M` มาให้ Input: จำนวนเต็มบวก `1` จำนวนจากแป้นพิมพ์ เก็บใน `k` Process: เปลี่ยนค่าของช่องใน `M` ที่หมายเลขแถวและหลักหารด้วย `k` ลงตัวให้มีค่าเป็น 0	
สมมติว่ามีอาเรย์ 2 มิติ `M` มาให้ Input: จำนวนเต็มบวก `1` จำนวนจากแป้นพิมพ์ เก็บใน `k` Process: เปลี่ยนค่าของช่องใน `M` ที่หมายเลขแถวและหลักหารด้วย `k` ลงตัวให้มีค่าเป็น `2` เท่าของค่าเดิม	
สมมติว่ามีอาเรย์ 2 มิติ `M` มาให้ ให้คำนวณหาผลต่างของค่ามากสุดและค่าน้อยสุดในแต่ละหลัก เช่น ถ้า `M = np.array([[3,2],[5,6],[7,1]])` จะได้คำตอบ `A` เท่ากับ `np.array([4,5])` (`4` มาจาก `7-3` และ `5` มาจาก `6-1`)	
กำหนดอาเรย์ `X` เก็บความยาวของด้านประกอบมุมฉากของรูปสามเหลี่ยมมุมฉากหลายรูป เช่น `X = np.array([[3,4],[5,12],[24,7]])` ให้สร้างอาเรย์ `Y` ซึ่งเก็บความยาวของด้านตรงข้ามมุมฉาก เช่น จาก `X` ด้านบน จะได้ `Y` เท่ากับ `array([5.,13.,25.])`	

Input: จำนวนเต็มบวก `1` จำนวนจากแป้นพิมพ์ เก็บใน `k`
Process: สร้างเมทริกซ์ขนาด `k×k` ซึ่งเก็บค่า `0` และ `1` เป็นตารางหมากรุก (มุมซ้ายบนเป็น `0`) เช่น ถ้า `k = 5` จะได้
`C = np.array([[0,1,0,1,0]`
`[1,0,1,0,1]`
`[0,1,0,1,0]`
`[1,0,1,0,1]`
`[0,1,0,1,0]])`	

Problem	Code
Input: จำนวนเต็มบวก 1 จำนวนจากแป้นพิมพ์ เก็บใน k Process: สร้างเมทริกซ์ขนาด k×k ซึ่งเก็บค่าเป็นตารางหมากรุกอีกแบบหนึ่ง (มุมซ้ายบนเป็น 1) โดยแทน 1 ด้วยเลขแถว (ให้เลขแถวเริ่มที่ 1) เช่น ถ้า k = 5 จะได้ C = np.array([[1,0,1,0,1] [0,2,0,2,0] [3,0,3,0,3] [0,4,0,4,0] [5,0,5,0,5]])	

ตัวอย่างการแก้โจทย์ปัญหา

Weighted Score

รายการประกวดร้องเพลงกำลังหาผู้ชนะจากการแข่งขัน โดยคะแนนของผู้เข้าแข่งขันมาจาก 3 ส่วน คือ คะแนนของกรรมการ คะแนนจากผู้ชมในห้องส่ง และคะแนนจากผู้ชมทางบ้าน ทางรายการได้กำหนดน้ำหนักของคะแนนแต่ละส่วนมาให้แล้ว ให้คำนวณคะแนนรวมของผู้เข้าแข่งขันแต่ละคน

▶ ข้อมูลนำเข้า

บรรทัดแรกคือจำนวนผู้เข้าแข่งขัน n เป็นจำนวนเต็ม
n บรรทัดถัดมา รับจำนวนเต็ม 3 จำนวน คือคะแนนของกรรมการ คะแนนจากผู้ชมในห้องส่ง และคะแนนจากผู้ชมทางบ้าน
บรรทัดสุดท้ายคือน้ำหนักของคะแนนแต่ละส่วน เป็นจำนวนทศนิยม 3 จำนวน

▶ ข้อมูลส่งออก

มี n บรรทัด แต่ละบรรทัดแสดงคะแนนของผู้เข้าแข่งขันแต่ละคน

▶ ตัวอย่าง

Input (จากแป้นพิมพ์)	Output (ทางจอภาพ)
4 10 15 10 20 5 15 14 8 7 12 12 12 0.5 0.25 0.25	11.25 15.0 10.75 12.0
2 10 30 20 20 10 30 0.6 0.3 0.1	17.0 18.0

CONFIDENTIAL

ตัวอย่างการเขียนโปรแกรม

โปรแกรม	คำอธิบาย

`n = int(input())`
`S = []`
`for i in range(n):`
`  S.append([int(e) for e in input().split()])`
`W = [float(e) for e in input().split()]`
 	
`for i in range(n):`
`  score = [S[i][j]*W[j] for j in range(3)]`
`  print(sum(score))`
	โปรแกรมส่วนบนเป็นส่วนสำหรับรับข้อมูล ได้ S เป็นลิสต์ซ้อนลิสต์ของคะแนนผู้เข้าแข่งขันแต่ละคน และ W เป็นลิสต์ของน้ำหนัก เช่น
	`S = [[10,30,20],[20,10,30]]`
	`W = [0.6,0.3,0.1]`
	โปรแกรมด้านขวาทำงานถูกต้อง แต่ถ้ามีจำนวนข้อมูลมาก จะช้า เพราะใช้ลิสต์ในการประมวลผล

`n = int(input())`
`S = []`
`for i in range(n):`
`  S.append([int(e) for e in input().split()])`
`W = [float(e) for e in input().split()]`
 	
`score = [sum([S[i][j]*W[j] \`
`  for j in range(3)]) \`
`  for i in range(n)]`
`for i in score:`
`  print(i)`
	ยุบ `for` วงล่างให้เป็น list comprehension ทำงานถูกต้อง แต่ก็ยังช้าอยู่ ลองเปลี่ยนมาใช้ `numpy`

`n = int(input())`
`S = []`
`for i in range(n):`
`  S.append([int(e) for e in input().split()])`
`W = [float(e) for e in input().split()]`
 	
`S = np.array(S)`
`W = np.array(W)`
`for i in range(n):`
`  print(sum(S[i]*W))`
	เปลี่ยนลิสต์ S และ W มาใช้ `numpy array`
	สั่ง run, ใส่ข้อมูลตามตัวอย่างแรก, ผิดที่บรรทัดที่ 7
	`NameError: name 'np' is not defined`
	แปลว่า ระบบไม่รู้จักคำว่า `np` เนื่องจากไม่ได้ `import numpy as np`

`import numpy as np`
`n = int(input())`
`S = []`
`for i in range(n):`
`  S.append([int(e) for e in input().split()])`
`W = [float(e) for e in input().split()]`
 	
`S = np.array(S)`
`W = np.array(W)`
`for i in range(n):`
`  print(sum(S[i]*W))`
	ทำงานได้ถูกต้อง (ใช้การคูณแบบ element-wise) แต่ไม่ได้เร็วกว่าของเดิมนัก เพราะยังใช้คำสั่งของ `numpy` ได้ไม่เต็มที่

CONFIDENTIAL

โปรแกรม	คำอธิบาย

`import numpy as np`
`n = int(input())`
`S = []`
`for i in range(n):`
`  S.append([int(e) for e in input().split()])`
`W = [float(e) for e in input().split()]`
 	
`S = np.array(S)`
`W = np.array(W)`
`S *= W`
`total_score = np.sum(S, axis = 0)`
`for i in total_score:`
`  print(i)`
	เราสามารถเขียน `S *= W` ได้เลย เพราะ numpy จะ broadcast อาเรย์ `W` ให้โดยอัตโนมัติ
	สั่ง run, ใส่ข้อมูลตามตัวอย่างแรก, ได้
	28
	9
	9
	ไม่ตรงกับที่แสดง พบว่าจำนวนตัวเลขไม่ครบ 4 ตัว น่าจะผิดที่ `axis`

`import numpy as np`
`n = int(input())`
`S = []`
`for i in range(n):`
`  S.append([int(e) for e in input().split()])`
`W = [float(e) for e in input().split()]`
 	
`S = np.array(S)`
`W = np.array(W)`
`S *= W`
`total_score = np.sum(S, axis = 1)`
`for i in total_score:`
`  print(i)`
	เปลี่ยนเป็น `axis = 1`
	สั่ง run, ใส่ข้อมูลตามตัวอย่างแรก, ได้
	10
	14
	10
	12
	ไม่ตรงกับที่แสดง พบว่าสิ่งที่แสดงเป็นจำนวนเต็ม แสดงว่าน่าจะมีปัญหาที่การเก็บค่า พบว่าคำสั่ง `S *= W` จะเก็บผลคูณในอาเรย์ `S` ซึ่งเก็บจำนวนเต็ม ดังนั้น ผลลัพธ์จึงเป็นจำนวนเต็ม ซึ่งไม่ถูกต้อง

`import numpy as np`
`n = int(input())`
`S = []`
`for i in range(n):`
`  S.append([int(e) for e in input().split()])`
`W = [float(e) for e in input().split()]`
 	
`S = np.array(S)`
`W = np.array(W)`
`T = S*W`
`total_score = np.sum(T, axis = 1)`
`for i in total_score:`
`  print(i)`
	เปลี่ยนเป็น `T = S*W`
	สั่ง run, ใส่ข้อมูลตามตัวอย่างแรก, ได้
	11.25
	15.0
	10.75
	12.0
	ถูกต้อง

`import numpy as np`
`n = int(input())`
`S = []`
`for i in range(n):`
`  S.append([int(e) for e in input().split()])`
`W = [float(e) for e in input().split()]`
 	
`S = np.array(S)`
`W = np.array(W)`
`for i in S.dot(W):`
`  print(i)`
	อีกวิธีที่ทำได้ และง่ายกว่าคือ ใช้คำสั่ง `S.dot(W)` ซึ่งได้ผลลัพธ์เป็นอาเรย์ 1 มิติ มีข้อมูลแต่ละตัวคือคะแนนรวมที่ถ่วงน้ำหนักแล้ว

CONFIDENTIAL

ตัวอย่างโจทย์ปัญหา

ค่าเช่าหนังสือ

จงเขียนโปรแกรมคำนวณหาราคาค่าเช่าหนังสือของร้านเช่าหนังสือแห่งหนึ่ง ที่มีประเภทหนังสือให้เช่า 4 ประเภท คือ นิยาย สารคดี ท่องเที่ยว และการ์ตูน

เจ้าของร้านเช่าหนังสือต้องการทราบว่าในหนึ่งสัปดาห์ (คิด 5 วันจันทร์ถึงศุกร์) วันใดให้เช่าหนังสือเป็นจำนวนเล่มมากที่สุด เป็นจำนวนกี่เล่ม และค่าเช่าหนังสือรวมทุกประเภทในแต่ละวันเป็นจำนวนเท่าไร

▶ ข้อมูลนำเข้า

บรรทัดที่ 1 เป็นจำนวนเต็ม 4 จำนวนคั่นด้วยช่องว่าง แทนค่าเช่า นิยาย สารคดี ท่องเที่ยว และการ์ตูน
บรรทัดที่ 2 เป็นจำนวนหนังสือนิยายที่ถูกเช่าในวัน จ. อ. พ. พฤ. และ ศ. (คั่นด้วยช่องว่าง)
บรรทัดที่ 3 เป็นจำนวนหนังสือสารคดีที่ถูกเช่าในวัน จ. อ. พ. พฤ. และ ศ. (คั่นด้วยช่องว่าง)
บรรทัดที่ 4 เป็นจำนวนหนังสือท่องเที่ยวที่ถูกเช่าในวัน จ. อ. พ. พฤ. และ ศ. (คั่นด้วยช่องว่าง)
บรรทัดที่ 5 เป็นจำนวนหนังสือการ์ตูนที่ถูกเช่าในวัน จ. อ. พ. พฤ. และ ศ. (คั่นด้วยช่องว่าง)

▶ ข้อมูลส่งออก

บรรทัดแรกแสดง ชื่อวันที่มีจำนวนหนังสือถูกเช่ารวมมากสุด และจำนวนหนังสือรวมนั้น (ให้ถือว่ามีวันเดียวเท่านั้นที่ให้เช่ามากสุด)
บรรทัดที่สองแสดงค่าเช่ารวมของหนังสือทุกประเภทในแต่ละวัน (เรียงตั้งแต่วันจันทร์ถึงศุกร์ คั่นด้วยช่องว่าง)

▶ ตัวอย่าง

Input (จากแป้นพิมพ์)	Output (ทางจอภาพ)
50 30 40 20 20 50 10 15 20 30 40 20 65 35 75 30 42 70 45 40 25 35 22 55	Thu 172 5700 5400 3480 5940 4950

ขั้นตอนการทำงานของโปรแกรม

1. `x` = ลิสต์ที่สร้างจากข้อมูลในบรรทัดแรกจากแป้นพิมพ์ เป็นจำนวนเต็ม 4 จำนวนแทนค่าเช่าหนังสือแต่ละประเภท
2. `rentalrates` = สร้าง `numpy array` ที่มีค่าเริ่มต้นจากลิสต์ `x`
3. `sales` = สร้าง `numpy array` ขนาด 4 แถว 5 คอลัมน์ (แถวแทนประเภทหนังสือ คอลัมน์แทนวัน)
4. วงวนทำข้อ 5 โดยเปลี่ยนค่าของตัวแปร `k = 0,1,2,3` (`k` แทนหมายเลขประเภทหนังสือ)
5. `sales[k,] = list` ที่สร้างจากข้อมูลหนึ่งบรรทัดจากแป้นพิมพ์ เป็นจำนวนเต็ม 5 จำนวน แทนจำนวนหนังสือประเภทที่ `k` ของแต่ละวันที่ขายได้
6. `totalsales` = ผลรวมของจำนวนหนังสือที่ถูกเช่าในแต่ละวันคำนวณจาก `sales` (ในข้อ 3)
7. `d` = ตำแหน่งใน `totalsales` ที่มีค่ามากสุด (`d = 0` แทนวันจันทร์, `1` แทนวันอังคาร, ..., `4` แทนวันศุกร์)
8. หาชื่อย่อวัน จาก `d` และ tuple `('Mon', 'Tue', 'Wed', 'Thu', 'Fri')`
9. แสดง ชื่อย่อวัน ตามด้วย `totalsales[d]`
10. `salesvalues` = ค่าเช่าหนังสือรวมของหนังสือทุกประเภทในแต่ละวัน (นำ `rentalrates` มา `dot` กับ `sales`)
11. แสดงรายการของยอดเงินที่ขายได้จาก `salesvalues` มาแสดง (คั่นด้วยช่องว่าง)

CONFIDENTIAL

BMI

ฟังก์ชัน `read_height_weight()` ข้างล่างนี้ อ่านข้อมูลความสูง (หน่วยเป็นเซนติเมตร) และน้ำหนัก (หน่วยเป็นกิโลกรัม) มาสร้าง `numpy array` แบบสองมิติ ดังตัวอย่างในตารางข้างล่างนี้ (บรรทัดแรกคือจำนวนข้อมูล บรรทัดที่ตามมาคือ ความสูงกับน้ำหนัก)

Input	ผลที่ได้จาก `read_height_weight()`

`4`
`160 60`
`155 62`
`170 54`
`180 55`
	`array([[160,  60],`
	`[155,  62],`
	`[170,  54],`
	`[180,  55]])`

จงเขียนฟังก์ชัน `cm_to_m(x)` และ `cal_bmi(hw)` ในโปรแกรมข้างล่างนี้ ที่มีข้อกำหนดของพารามิเตอร์ และผลลัพธ์ที่ได้ตามตารางนี้

Function	Input parameter	Return value
`cm_to_m(x)`	`array` หนึ่งมิติ เก็บความสูงหน่วยเป็นเซนติเมตร เช่น `array([160, 155, 170, 180])`	`array` หนึ่งมิติเก็บความสูงหน่วยเป็นเมตร เช่น `array([1.6,1.55,1.7,1.8])`

`cal_bmi(hw)`	`array` สองมิติ ขนาด `n` แถว 2 คอลัมน์ แต่ละแถวแทนข้อมูลหนึ่งคู่ คอลัมน์ 0 เก็บความสูง (เซนติเมตร) คอลัมน์ 1 เก็บน้ำหนัก (กิโลกรัม) เช่น
	`array([[160,  60],`
	`[155,  62],`
	`[170,  54],`
	`[180,  55]])`
		`array` หนึ่งมิติเก็บ `bmi` ที่คำนวณจากความสูงและน้ำหนักใน Input parameter ที่ได้รับ เช่น
		`array([23.4375, 25.80645161,`
		`18.68512111, 16.97530864])`

และเขียนคำสั่ง

- หาค่าเฉลี่ยของ `bmi` ทั้งหมดที่คำนวณได้ เก็บใส่ตัวแปร `avg_bmi` และ
- นับจำนวน `bmi` ที่คำนวณได้ที่มีค่าน้อยกว่า `18.5`

CONFIDENTIAL

โปรแกรมเริ่มต้น

```
import numpy as np

def read_height_weight():
    list_hw = []
    for k in range(int(input())) :
        h,w = input().split()
        list_hw.append((int(h),int(w)))
    return np.array(list_hw)

def cm_to_m(x):
    # ???



def cal_bmi(hw):
    # ???




def main():
    hw = read_height_weight()
    bmi = cal_bmi(hw)
    avg_bmi = ______________________________________
    count_underweight = ____________________________
    print('average bmi =', avg_bmi)
    print('#bmi < 18.5 =', count_underweight)

exec(input().strip())
```

▶ ข้อมูลนำเข้า

คำสั่งในการทดสอบฟังก์ชันที่เขียน

▶ ข้อมูลส่งออก

ผลที่ได้จากคำสั่งที่ป้อนเป็นข้อมูลนำเข้า

▶ ตัวอย่าง

Input (จากแป้นพิมพ์)	Output (ทางจอภาพ)
`x=np.array([160,150,140]); print(cm_to_m(x)); print(x)`	`[ 1.6  1.5  1.4]` `[160 150 140]`
`d=np.array([[100,30],[120,36]]); print(cal_bmi(d))`	`[ 30.  25.]`

`main()`
`4`
`160 60`
`155 62`
`170 54`
`180 55`
	`average bmi = 21.2260953405`
	`#bmi < 18.5 = 1`

CONFIDENTIAL

การคำนวณจำนวนฟิโบนักชีโดยใช้การยกกำลังเมทริกซ์อย่างรวดเร็ว

CONFIDENTIAL

0, 1, 1, 2, 3, 5, 8, ... เป็นลำดับของจำนวนฟิโบนักชี (F_0 = 0, F_1 = 1, F_2 = 1, ...) วิธีหนึ่งในการหา F_n คือคำนวณผลการยกกำลัง n ของเมทริกซ์ A ที่แสดงด้านล่าง จะได้ผลเป็นเมทริกซ์ขนาด 2×2 ที่มี F_n อยู่ที่มุมขวาบนของเมทริกซ์ เช่น

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

หา F_3 คำนวณ $A^3 = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^3 = \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix}$ ได้ $F_3 = 2$ หา F_4 คำนวณ $A^4 = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^4 = \begin{bmatrix} 2 & 3 \\ 3 & 5 \end{bmatrix}$ ได้ $F_4 = 3$

ถ้าคิดดูดี ๆ จะพบว่าการหาด้วยวิธีข้างต้นนี้คือการหาค่ายกกำลัง ซึ่งเราก็ไม่น่าหาแบบค่อย ๆ คูณไปทีละครั้ง เช่น การหา A^{10} ก็ไม่น่าใช้วิธีที่เริ่มด้วยเมทริกซ์เอกลักษณ์ I แล้วคูณด้วย A ไป 10 ครั้ง น่าจะใช้วิธีการหา A^5 แล้วจับมาคูณกับตัวเองก็จะได้ A^{10} นั่นคือ

$$A^n = \begin{cases} \mathrm{I} & n = 0 \\ \left(A^{\lfloor n/2 \rfloor}\right)^2 & n \text{ is even} \\ A\left(A^{\lfloor n/2 \rfloor}\right)^2 & n \text{ is odd} \end{cases} \qquad A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \qquad \mathrm{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

จงเขียนฟังก์ชัน `fib(n,k)` เพื่อคำนวณ F_n % k ด้วยวิธีข้างต้นนี้ โดยใช้คำสั่งของ numpy เพื่อคูณเมทริกซ์ (หมายเหตุ: หลังการคูณเมทริกซ์ทุกครั้ง ให้นำผลที่ได้มา % k numpy จะทำ % k แบบ element-wise ในเมทริกซ์)

```
import numpy as np

def fib(n,k):
    # ???

n,k = [int(e) for e in input().split()]
print( fib(n,k) )
```

▶ <u>ข้อมูลนำเข้า</u>

จำนวนเต็ม 2 ค่า n กับ k (0 ≤ n ≤ 10**13, 0 ≤ k ≤ 100000)

▶ <u>ข้อมูลส่งออก</u>

แสดงค่า F_n % k

▶ <u>ตัวอย่าง</u>

Input (จากแป้นพิมพ์)	Output (ทางจอภาพ)
0 10	0
2 10	1
89 10	9
11111 111	55
1234567890 1234	162
10000000000000 999	600

10 : Class

CONFIDENT

สรุปเนื้อหา

- คลาส ใช้สร้างประเภทข้อมูลใหม่ที่ต้องการ โดยเรากำหนดได้ว่า จะให้เก็บข้อมูลย่อยอะไรบ้าง และทำงานอะไรได้บ้าง
- คลาส คือ ประเภทข้อมูล, อ็อบเจกต์ คือ ตัวข้อมูล เช่น `b1 = Book(...)` ได้ว่า `b1` เป็นอ็อบเจกต์ของคลาส `Book`
- เมท็อด คือ ฟังก์ชันซึ่งเป็นบริการของคลาส

ตัวอย่าง: คลาส `Book`

- คลาส `Book` ข้างล่างนี้ ใช้สร้างอ็อบเจกต์ที่เก็บข้อมูลของหนังสือ (ชื่อ ราคา และคำสำคัญต่าง ๆ ของหนังสือ)
- ทุกเมท็อดของคลาสต้องมีตัวแปร `self` เป็นพารามิเตอร์แรก ซึ่งแทนอ็อบเจกต์ที่จะใช้บริการ เมื่อเรียกใช้ตัวแปรประจำอ็อบเจกต์ภายในคลาส จะต้องมี `self.` นำหน้าเสมอ
- เมท็อด `__init__` (เรียกว่า constructor) ใช้สร้างอ็อบเจกต์ของคลาส โดยบอกว่าอ็อบเจกต์จะเก็บข้อมูลอะไรบ้าง และจะถูกเรียกเมื่อมีการสร้างอ็อบเจกต์ โดย `self` จะแทนอ็อบเจกต์ที่เพิ่งสร้าง
- เมท็อด `__str__` คืนค่าสตริงของอ็อบเจกต์ จะถูกเรียกเมื่อใช้งานฟังก์ชัน `str` หรือ `print`
- เมท็อด `__lt__` ใช้สำหรับเปรียบเทียบอ็อบเจกต์ของ `Book` ในที่นี้จะเปรียบเทียบโดยใช้ราคาก่อน ถ้าราคาเท่ากัน จะใช้ชื่อหนังสือเป็นตัวเปรียบเทียบ เมท็อดนี้จะถูกเรียกเมื่อเปรียบเทียบอ็อบเจกต์ด้วย `<` หรือใช้ฟังก์ชัน `sort`
- นอกจากนี้ยังมีบริการ ปรับราคา และขอคำสำคัญร่วมของหนังสือ 2 เล่ม ผ่านเมท็อด `update_price` และ `get_common_keywords` (สามารถเขียนเมท็อดอื่น ๆ เพิ่มได้ตามต้องการ เป็นการเพิ่มความสามารถของอ็อบเจกต์)
- เมท็อดอาจจะคืนค่าหรือไม่ก็ได้ โดยทั่วไปเมท็อดที่ไม่คืนผลลัพธ์มักเป็นเมท็อดที่มีการเปลี่ยนแปลงค่าภายในอ็อบเจกต์

```
class Book:
    def __init__(self, title, price, keywords):
        self.title = title; self.price = price; self.keywords = set(keywords)

    def __lt__(self, rhs):
        if self.price != rhs.price: return self.price < rhs.price
        else: return self.title < rhs.title

    def __str__(self):
        return self.title + ' ($' + str(self.price) + ')'

    def update_price(self, new_price):
        self.price = new_price

    def get_common_keywords(self, other):
        return self.keywords & other.keywords

b1 = Book('Python',    99, ['code','computer'])          # using __init__
b2 = Book('Calculus', 199, ['maths'])
b3 = Book('Physics',   99, ['science','maths'])

b1.update_price(199)
print(Book.get_common_keywords(b2,b3))                   # {'maths'}
if b3 < b2: print(b1)                                    # using __lt__ & __str__
books = [b1,b2,b3]
books.sort()                                             # using __lt__
print(books[0],',',books[1],',',books[2])                # using __str__
# 'Physics ($99) , Calculus ($199) , Python ($199)'
```

- การเรียกเมท็อดทำได้ 2 แบบ
  - เรียกผ่านชื่อคลาส เช่น `Book.get_common_keywords(b2,b3)`
  - เรียกผ่านอ็อบเจกต์ เช่น `b2.get_common_keywords(b3)`

  (b2 จะถูกแทนใน `self` และ b3 จะถูกแทนใน `other` โดยอัตโนมัติ และทั้งสองคำสั่งนี้ทำงานเหมือนกัน)
- `str(b1)` เหมือนกับการเรียก `Book.__str__(b1)` หรือ `b1.__str__()`
- `print(b1)` เหมือนกับการเรียก `print(str(b1))`
- `b1 < b2` เหมือนกับการเรียก `Book.__lt__(b1,b2)` หรือ `b1.__lt__(b2)`
- การเรียก `books.sort()` จะเรียงลำดับอ็อบเจกต์ของ `Book` จากน้อยไปมาก โดยเปรียบเทียบด้วย `__lt__`

CONFIDENTIAL

เรื่องผิดบ่อย

ลืม `self` ในเมท็อด
ไม่ได้ประกาศ `self` เป็นพารามิเตอร์
ไม่มี `self.` นำหน้าตัวแปรของอ็อบเจกต์
	`class A:`
	`  def __init__(self, x, y):`
	`  d = dict()  # แก้เป็น self.d = dict()`
	`  d[x] = y  # แก้เป็น self.d[x] = y`
	`  def total():  # แก้เป็น def total(self):`
	`  return sum(d.values())`
	`  # แก้เป็น return sum(self.d.values())`

การกำหนดค่าอ็อบเจกต์ให้กับตัวแปรด้วยเครื่องหมายเท่ากับ จะทำให้ตัวแปรนั้นชี้ไปที่อ็อบเจกต์เดียวกัน	`class B:`
	`  def __init__(self, b):`
	`  self.b = b`
	
	`b1 = B(10)`
	`b2 = b1  # b1, b2 อ้างอิงอ็อบเจกต์เดียวกัน`
	`b2.b = 5`
	`print(b1.b)  # 5`

indent ผิด	`class C:`
	`  def __init__(self, c):`
	`  self.c = c`
	`def double(self):  # indent ผิด กลายเป็นฟังก์ชันทั่วไป`
	`  self.c *= 2  # เรียก double(c1) ได้`
	`  # แต่จะเรียก c1.double() ไม่ได้`
	`  # เรียก C.double(c1) ก็ไม่ได้`

แบบฝึกหัด

1. เติมเมท็อดของคลาสนิสิตตามคอมเมนต์ที่กำหนดให้สมบูรณ์

```
class Nisit:
    def __init__(self, name, year, faculty):
        # n = Nisit('Krit', 4, 'Engineering')





    def __str__(self):
        # คืนสตริงของนิสิต เช่น 'Krit (year 4) Engineering'





    def __lt__(self, rhs):
        # เรียงลำดับนิสิตด้วยคณะตามพจนานุกรม ถ้าอยู่คณะเดียวกัน ให้เรียงลำดับด้วยชั้นปีจากน้อยไปมาก
        # ถ้าอยู่คณะและชั้นปีเดียวกัน ให้เรียงลำดับด้วยชื่อตามพจนานุกรม
        # เช่น  Nisit('Krit', 4, 'Engineering') < Nisit('Boy',  3, 'Science')
        #       Nisit('Prim', 2, 'Engineering') < Nisit('Krit', 4, 'Engineering')
        #       Nisit('Joey', 2, 'Engineering') < Nisit('Prim', 2, 'Engineering')
```

CONFIDENTIAL

2. เติมเมท็อดของคลาสรถยนต์ตาม comment ที่กำหนดให้ให้สมบูรณ์

```
class Car:
    def __init__(self, license, brand, color):
        # c = Car('AA1234', 'Honda', 'White')
        # มีตัวแปร report สำหรับเก็บข้อมูลประวัติการซ่อมบำรุง โดยกำหนดค่าเริ่มต้นเป็นลิสต์ว่าง




    def __str__(self):
        # คืนสตริงของรถยนต์ เช่น 'AA1234 - White Honda'




    def __lt__(self, rhs):
        # เรียงลำดับรถยนต์โดยเปรียบเทียบป้ายทะเบียนรถแบบสตริง




    def add_report(self, new_report):
        # เพิ่มประวัติการซ่อมบำรุง โดยไม่ต้องคืนค่า
        # ตัวแปร new_report เก็บ tuple (วันที่, คำอธิบาย, ราคา)
        # เช่น c.add_report( ('25 May 2017', 'change tires', 1500) )




    def total_payment(self):
        # คืนค่าใช้จ่ายทั้งหมดที่ใช้ในการซ่อมบำรุงที่ผ่านมา




    def max_payment(self):
        # คืนลิสต์ของประวัติการซ่อมบำรุง (วันที่, คำอธิบาย, ราคา) ทุกรายการ ที่มีค่าใช้จ่ายมากที่สุด
        # กรณีที่รถยนต์ไม่มีประวัติการซ่อมบำรุงเลย ให้คืนค่าลิสต์ว่าง
```

CONFIDENTIAL

3. จาก class Book ให้เติมเมท็อดของ class ShoppingCart สำหรับการซื้อหนังสือผ่านเว็บไซต์ ดังนี้

```
class ShoppingCart:
    def __init__(self, id):
        self.id = id
        self.books = []
        # books เก็บข้อมูลของหนังสือในตะกร้าพร้อมจำนวน เช่น [[b1,2],[b3,7]]

    def add_book(self, book, n):
        # เพิ่มข้อมูลการซื้อหนังสือ book เพิ่มอีก n เล่ม โดยไม่ต้องคืนค่า
        # หากไม่มีหนังสือเล่มนี้ในตะกร้า ให้เพิ่มลิสต์ [book, n] ต่อท้าย books
        # หากเคยมีข้อมูลหนังสือเล่มนี้ในตะกร้าแล้ว ให้เพิ่มจำนวนที่ซื้ออีก n เล่ม
        # เช่น ถ้า books = [[b1,2]] และเราสั่ง add_book(b1,3) จะได้ books = [[b1,5]]









    def delete_book(self, book):
        # ลบข้อมูลการซื้อหนังสือ book ออกจากตะกร้า โดยไม่ต้องคืนค่า
        # ถ้าในตะกร้าไม่มีหนังสือ book ไม่ต้องทำอะไร









    def get_total(self):
        # คืนค่าราคารวมของหนังสือทั้งหมดในตะกร้า









    def __lt__(self, rhs):
        # ตะกร้าที่มีราคารวมของหนังสือน้อยกว่า จะเป็นตะกร้าที่น้อยกว่า
```

CONFIDENTIAL

4. ข้างล่างนี้แสดงคลาส Station และคลาส BTScard (อ่านคำอธิบายของแต่ละคลาสจาก comment ที่เขียน)

สถานีรถไฟฟ้าเป็นอ็อบเจกต์ของคลาส Station และบัตรโดยสารแบบเติมเงินแต่ละใบเป็นอ็อบเจกต์ของคลาส BTScard จงเติมคำสั่งในเมท็อด add_value, enter, leave และ __lt__ ของคลาส BTScard ให้ทำงานตาม comment ที่เขียน (เมท็อดอื่นที่ได้เขียนคำสั่งไว้ ทำงานถูกต้องแล้ว) ดูตัวอย่างการใช้งานข้างล่างนี้ประกอบ

```
s1 = Station(1,'Siam'); s2 = Station(3,'Mo Chit'); s3 = Station(5,'Asok')
c1 = BTScard(123, 5); c2 = BTScard(999, 10)
# ------------------------------------------------------------------
c1.add_value(100)       # c1 มีเงินในบัตร 105 บาท
p = c1.enter(s1)        # p = True
p = c1.enter(s3)        # p = False (แตะเข้าสถานีหลังจากแตะเข้าไปแล้ว)
p = c1.leave(s2)        # c1 เหลือเงินในบัตร 95 บาท โดย p = (95, 0)
# ------------------------------------------------------------------
p = c2.enter(s3)        # p = True
p = c2.leave(s1)        # c2 มีเงินในบัตร 10 บาทไม่พอจ่ายค่าโดยสาร โดย p = (10, -1)
c2.add_value(50)        # c2 มีเงินในบัตร 60 บาท
p = c2.leave(s1)        # c2 เหลือเงินในบัตร 40 บาท โดย p = (40, 0)
p = c2.leave(s2)        # p = (40, -2) (ยังไม่ได้แตะเข้าสถานี จึงไม่มีสถานีต้นทาง)
p = c2.enter(s2)        # p = True
p = c1 < c2             # p = False
```

```
class Station:                          # คลาสของสถานีรถไฟฟ้า
    def __init__(self, id, name):       # สร้างสถานีที่มีหมายเลข (id) และชื่อสถานี (name)
        self.sid = int(id)              # กำหนดให้หมายเลขสถานีเป็นจำนวนเต็ม โดยสถานีที่ติดกัน
        self.name = name                # มีค่าห่างกัน 1

    def get_price(self, other):         # คืนค่าโดยสารระหว่างสถานี self และ other
        return abs(self.sid - other.sid)*5

# ------------------------------------------------------------------

class BTScard:                          # คลาสของบัตรโดยสารแบบเติมเงิน
    def __init__(self, id, value):      # สร้างบัตรโดยสารที่มีเลขบัตร (id) และเงินเริ่มต้น (value)
        self.cid = id                   # self.station เก็บว่าสถานีต้นทางคือสถานีอะไร
        self.value = value              # โดยถ้าบัตรไม่ได้อยู่ระหว่างการเดินทาง จะเก็บค่า
        self.station = ''               # สถานีต้นทางนี้เป็นสตริงว่าง ๆ

    def __str__(self):
        return '('+str(self.cid)+','+str(self.value)+')'

    def add_value(self, x):             # เพิ่มเงินในบัตรโดยสารเท่ากับ x โดยไม่ต้อง return
```

CONFIDENTIAL

```
def enter(self, station):
    # แตะบัตรเพื่อเข้าสู่สถานีรถไฟฟ้า ให้เช็คว่า บัตรนี้ไม่ได้แตะเข้าที่สถานีอื่นมาก่อน
    # ถ้าไม่มีการแตะเข้ามาก่อน ให้เปลี่ยนค่าสถานีต้นทางเป็น station แล้ว return True
    # แต่ถ้ามีการแตะเข้าสถานีอื่นมาก่อน ให้ return False โดยไม่เปลี่ยนข้อมูลสถานีต้นทางของบัตรโดยสาร

def leave(self, station):
    # แตะบัตรเพื่อออกจากสถานีรถไฟฟ้า ให้เช็คว่า บัตรนี้มีข้อมูลสถานีต้นทางอยู่
    # ถ้าไม่มีข้อมูลสถานีต้นทาง ให้ return tuple ของเงินในบัตรและ -2
    # ถ้ามีสถานีต้นทาง แต่จำนวนเงินในบัตรไม่พอจ่ายค่าโดยสาร ให้ return tuple ของเงินในบัตรและ -1
    # ถ้ามีสถานีต้นทาง และจำนวนเงินในบัตรพอจ่ายค่าโดยสาร ให้ลบค่าโดยสารออกจากจำนวนเงินในบัตร
    # เปลี่ยนสถานีต้นทางเป็นสตริงว่าง แล้ว return tuple ของเงินในบัตรหลังหักค่าโดยสารและ 0

def __lt__(self, rhs):          # บัตรโดยสารที่มีเงินในบัตรน้อยกว่า จะถือว่าน้อยกว่า
```

CONFIDENTIAL

ตัวอย่างการแก้โจทย์ปัญหา

CONFIDENTIAL

Bus

ให้เขียนคลาสของรถเมล์ซึ่งมีเมท็อดดังนี้

1. `__init__` สร้างรถเมล์ 1 คัน รับพารามิเตอร์ จำนวนคนบนรถ `people` และค่าโดยสาร `fare`
2. `__str__` คืนค่าสตริงซึ่งบอกจำนวนคนบนรถและค่าโดยสาร
3. `__lt__` เปรียบเทียบรถเมล์โดยพิจารณาค่าโดยสารรวมของรถ (จำนวนคนบนรถคูณค่าโดยสารต่อคน)
4. `people_in` เพิ่มจำนวนคนบนรถ `k` คน ไม่คืนค่า
5. `people_out` ลดจำนวนคนบนรถ `k` คน (หากจำนวนคนน้อยกว่า `0` จะต้องแก้ไขจำนวนคนเป็น `0`) ไม่คืนค่า
6. `change_fare` เปลี่ยนค่าโดยสารเป็นค่าโดยสารใหม่ `new_fare` ไม่คืนค่า

▶ ตัวอย่าง

`b1 = Bus(10, 5)`	`# b1 has 10 people with fare = 5`
`b2 = Bus( 8, 7)`	`# b2 has  8 people with fare = 7`
`if b1 < b2:`	`# b1 < b2 is True (10*5 < 8*7)`
`    print(b1)`	`this bus has 10 people with fare = 5`
`else:`	
`    print(b2)`	
`b1.people_in(3)`	`# b1 has 13 people with fare = 5`
`b1.people_out(6)`	`# b1 has  7 people with fare = 5`
`b1.change_fare(12)`	`# b1 has  7 people with fare = 12`
`print(b1)`	`this bus has 7 people with fare = 12`

ตัวอย่างการเขียนโปรแกรม

โปรแกรม	คำอธิบาย

`class Bus:`
`    def __init__(people, fare):`
`        people = people`
`        fare = fare`
`    def __str__():`
`        return 'this bus has ' + str(people) \`
`        + ' people with fare = ' + str(fare)`
`    def __lt__(rhs):`
`        return people*fare < \`
`                 rhs.people*rhs.fare`
`    def people_in(k):`
`        return people += k`
`    def people_out(k):`
`        return people -= k`
`    def change_fare(new_fare):`
`        return fare = new_fare`
	ทำงานไม่ถูกต้อง มีจุดผิดดังนี้
	- ไม่มีการใช้ `self`
	- เมท็อด `people_in`, `people_out` และ `change_fare` ต้องไม่คืนค่า
	- เมท็อด `people_out` ไม่ได้ตรวจสอบว่าจำนวนคนน้อยกว่าศูนย์หรือไม่

โปรแกรม	คำอธิบาย

`class Bus:`
`  def __init__(self, people, fare):`
`  self.people = people`
`  self.fare = fare`
`  def __str__(self):`
`  return 'this bus has ' + \`
`  str(self.people) + \`
`  ' people with fare = ' + \`
`  str(self.fare)`
`  def __lt__(self, rhs):`
`  return self.people*self.fare < \`
`  rhs.people*rhs.fare`
`  def people_in(self, k):`
`  self.people += k`
`  def people_out(self, k):`
`  self.people = max(0,self.people-k)`
`  def change_fare(self, new_fare):`
`  self.fare = new_fare`
	ทำงานถูกต้อง สังเกตการใช้งาน `self.people = max(0,self.people-k)` ว่าทำการแก้ไขจำนวนคนหากน้อยกว่าศูนย์ให้แล้ว

`class Bus:`
`  def __init__(self, people, fare):`
`  self.fare = fare`
`  self.total = people*fare`
`  def __str__(self):`
`  return 'this bus has ' + \`
`  str(self.total//self.fare) + \`
`  ' people with fare = ' + \`
`  str(self.fare)`
`  def __lt__(self, rhs):`
`  return self.total < rhs.total`
`  def people_in(self, k):`
`  self.total += self.fare*k`
`  def people_out(self, k):`
`  self.total = max(0,self.total- \`
`  self.fare*k)`
`  def change_fare(self, new_fare):`
`  self.total = \`
`  self.total//self.fare*new_fare`
`  self.fare = new_fare`
	เราสามารถเขียนคลาส Bus แบบอื่นได้ เช่น แทนที่จะเก็บจำนวนคนและค่าโดยสาร อาจจะเก็บค่าโดยสารต่อคนและค่าโดยสารรวมก็ได้ แต่ก็ต้องปรับการทำงานของคลาสให้ถูกต้อง (ในเมท็อด `change_fare` ถ้าทำการเปลี่ยนค่า `self.fare` ก่อน จะทำให้เมท็อดทำงานผิดได้)

CONFIDENTIAL

ตัวอย่างโจทย์ปัญหา

เศษส่วน

กำหนดคลาสของเศษส่วน ประกอบด้วยเศษ (numerator) และส่วน (denominator) และมีเมท็อด 4 เมท็อดคือ เมท็อดสำหรับการแสดงผลเป็นสตริง เมท็อดการทำเศษส่วนอย่างต่ำ เมท็อดการบวก และเมท็อดการคูณ ดังนี้ (ขอให้สังเกตการเรียกใช้งานเมท็อด ว่าสามารถเรียกได้หลายแบบ)

```
def gcd(x,y):
    if x%y == 0: return y
    return gcd(y,x%y)

class Fraction:
    def __init__(self,a,b):
        self.numerator   = ___________________
        self.denominator = ___________________

    def __str__(self):
        # ???



    def simplify(self):
        g = gcd(self.numerator,self.denominator)
        return Fraction(self.numerator//g,self.denominator//g)

    def add(self,other):
        # ???





    def multiply(self,other):
        ans_numer = self.numerator * other.numerator
        ans_denom = self.denominator * other.denominator
        return Fraction(ans_numer,ans_denom).simplify()

a,b,c,d = [int(e) for e in input().split()]
fraction1 = ____________________
fraction2 = ____________________

print(fraction1.add(fraction2))
print(Fraction.multiply(fraction1,fraction2))
```

โจทย์ได้เขียนเมท็อดการทำเศษส่วนอย่างต่ำและเมท็อดการคูณมาให้แล้ว ให้เขียนเติมส่วนอื่น ๆ ให้สมบูรณ์ สำหรับการบวก เมื่อบวกแล้วให้ตอบเป็นเศษส่วนอย่างต่ำด้วย สามารถเรียกใช้เมท็อด `simplify` ได้

CONFIDENTIAL

▶ ข้อมูลนำเข้า

มีบรรทัดเดียว เป็นจำนวนเต็มบวก a b c d ซึ่งแทนเศษส่วน a/b และ c/d

▶ ข้อมูลส่งออก

มี 2 บรรทัด แสดงผลบวกและผลคูณของเศษส่วนที่กำหนดให้

▶ ตัวอย่าง

Input (จากแป้นพิมพ์)	Output (ทางจอภาพ)
1 7 3 7	4/7 3/49
1 2 1 3	5/6 1/6
1 8 3 8	1/2 3/64
2 3 1 2	7/6 1/3

CONFIDENTIAL