

Programming Fundamentals I Lab.

10. Class Inheritance

ชื่อ _____ รหัสนิสิต _____

ในปฏิบัติการนี้ คุณจะรู้จักการสืบทอดคลาส และการเขียนโปรแกรมติดต่อระหว่างวัตถุ

10.1 การสืบทอด

สร้างไฟล์ .py ใหม่ให้มันได้ดังนี้

```
class Person:
    def __init__(self, name):
        self.name = name
        self.power = 100

    def say(self, to_whom, something):
        print(self.name + ' says to ' + to_whom.name + ': ' + something)

    def move(self):
        print(self.name + ' is walking.')

a = Person('Alice')
b = Person('Bob')

a.say(b, 'Hi, ' + b.name)
a.move()
b.move()
```

รันโปรแกรม บันทึกผลที่ได้

แก้ไขไฟล์ .py เป็นดังนี้

```
class Person:
    def __init__(self, name):
        self.name = name
        self.power = 100

    def say(self, to_whom, something):
        print(self.name + ' says to ' + to_whom.name + ': ' + something)

    def move(self):
        print(self.name + ' is walking.')

class KUPerson(Person):
    number_of_members = 0

    def __init__(self, name):
        Person.__init__(self, name)
        KUPerson.number_of_members += 1
        self.member_id = KUPerson.number_of_members

a = Person('Alice')
b = KUPerson('Bob')
c = KUPerson('Carl')
```

รันโปรแกรม หลังจากนั้น ทดลองพิมพ์คำสั่งต่อไปนี้ใน Shell ตามลำดับ และบันทึกผล

a.power	
b.power	
c.power	
a.member_id	
b.member_id	
c.member_id	
b.say(a, 'Welcome to KU.')	
c.move()	

จากการทดลอง คุณคิดว่าในการประกาศ `class KUPerson(Person)` การใส่คลาส `Person` ในวงเล็บ มีผลอย่างไร

คุณคิดว่าคำสั่ง `Person.__init__(self, name)` ทำหน้าที่อะไร

ทำไมค่าของ `b.member_id` และ `c.member_id` จึงมีค่าไม่เท่ากัน

ทดลองแก้ไขคลาส `KUPerson` โดยเพิ่มเมธอด `move()` ดังนี้

```
def move(self):
    print(self.name + ' is riding a motorcycle.')
```

โดยให้คลาส `Person` และโค้ดในการสร้างตัวแปร `a`, `b`, และ `c` อยู่เหมือนเดิม รันโปรแกรมใหม่และลองสั่ง `c.move()` บันทึกผลที่ได้

ในไฟล์ .py ให้ลบคำสั่งในการสร้างตัวแปร a, b, และ c ออก และเพิ่มคลาสดังโค้ดต่อไปนี้

```
class Prof(KUPerson):
    def __init__(self, name, rank):
        KUPerson.__init__(self, name)
        self.rank = rank
        self.teaching = []

    def add_teaching(self, subj):
        self.teaching.append(subj)

class Student(KUPerson):
    def __init__(self, name):
        KUPerson.__init__(self, name)
        self.year = 1

    def set_year(self, year):
        self.year = year
```

จงแสดงรายชื่อเมธอด, instant variable, และ class variable ทั้งหมดที่สามารถเข้าถึงได้ในวัตถุของคลาส Prof

จงแสดงรายชื่อเมธอด, instant variable, และ class variable ทั้งหมดที่สามารถเข้าถึงได้ในวัตถุของคลาส Student

10.2 Setter และ getter

สังเกตดูในคลาส `Student` จะเห็นว่าเราสามารถเรียกเมธอด `set_year()` เพื่อแก้ไขค่าของตัวแปร `year` ในวัตถุแต่ละตัวของคลาสได้อย่างอิสระ นั่นหมายความว่าเราอาจแก้ไขค่าของตัวแปร `year` ให้มีค่าเป็นเลขลบได้ ซึ่งไม่เหมาะสมเพราะไม่สื่อความหมายตามที่เราต้องการ

เราสามารถเขียนโค้ดเพื่อป้องกันการแก้ไขค่าของตัวแปรต่าง ๆ เพื่อให้มีค่าที่เหมาะสมอยู่เสมอ โดยเพิ่มโค้ดในการพิจารณาค่าที่รับมาก่อนแก้ไขในตัวแปรจริง ตัวอย่างเช่น

ให้คุณแก้ไขเมธอด `set_year()` ในคลาส `Student` เป็นดังนี้

```
def set_year(self, year):
    if year > 0:
        self.year = year
    else:
        self.year = 1
```

ทดลองรันโปรแกรม และทดสอบด้วยชุดคำสั่งต่อไปนี้

```
>>> a = Student('Alice')
>>> print(a.year)
>>> a.set_year(3)
>>> print(a.year)
>>> a.set_year(-7)
>>> print(a.year)
```

บันทึกผลที่ได้

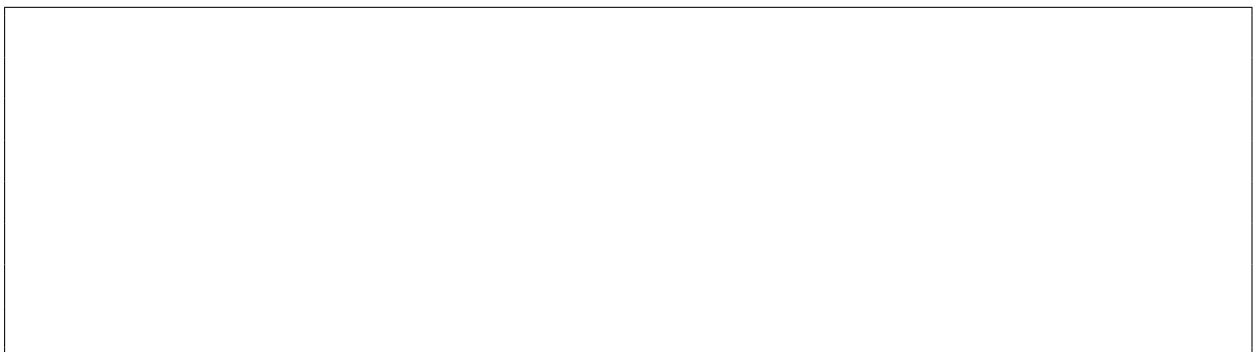
จงแก้ไขเมธอด `add_teaching()` ในคลาส `Prof` เพื่อตรวจสอบว่า วัตถุของคลาส `Prof` แต่ละตัวจะมีจำนวนสมาชิกในลิสต์ `teaching` ไม่เกิน 3 เสมอ โดยหากวัตถุของ `Prof` มีจำนวนสมาชิกในลิสต์ `teaching` เป็น 3 อยู่แล้ว เมื่อถูกสั่ง `add_teaching()` ก็จะไม่มีการเพิ่มสมาชิกเข้าไปในลิสต์ `teaching` อีก บันทึกโค้ดของเมธอดหลังการแก้ไข



เมธอดที่ทำหน้าที่กรองเงื่อนไขก่อนแก้ค่าของตัวแปรของวัตถุเช่นนี้ เราเรียกว่า `setter`

นอกจากนี้ ยังมีเมธอดอีกพวกหนึ่งที่ทำหน้าที่ดึงข้อมูลบางอย่างออกมาแสดง ซึ่งอาจเป็นข้อมูลที่มีการบันทึกในตัวแปรหรือเป็นข้อมูลที่ต้องคำนวณจากข้อมูลอื่น ๆ ก็ได้ เราเรียกเมธอดเหล่านี้ว่า `getter`

ให้คุณเพิ่มเมธอดชื่อ `is_alive(self)` ในคลาส `Person` โดยให้คืนค่าเป็น `True` ถ้าวัตถุนั้นมีค่าของตัวแปร `power` มากกว่า 0 ไม่เช่นนั้นให้คืนค่าเป็น `False` บันทึกโค้ดของเมธอดที่สร้างขึ้น



10.3 การสืบทอดหลายทาง

สร้างคลาสเพิ่มในไฟล์ `.py` ดังนี้

```
class Runner(Person):
    def move(self):
        print(self.name + 'is running.')
```

```
class RunningStudent(Runner, Student):
    def __init__(self, name):
        Student.__init__(self, name)
```

รันและสร้างตัวแปรที่เป็นวัตถุของคลาส `RunningStudent` ทดลองเรียกใช้เมธอดต่าง ๆ บันทึกว่าสามารถเรียกใช้เมธอดอะไรได้บ้าง และมีตัวแปรอะไรที่เข้าถึงได้บ้าง

หากเราสั่งให้ตัวแปรวัตถุของคลาส `RunningStudent` กระทำคำสั่ง `move()` การทำงานที่ได้จะเป็นการทำงานที่สืบทอดมาจากคลาสใด

ทดลองแก้ไขคลาส `RunningStudent` เป็นดังนี้

```
class RunningStudent(Student, Runner):
    def __init__(self, name):
        Student.__init__(self, name)
```

หากเราสั่งให้ตัวแปรวัตถุของคลาส `RunningStudent` กระทำคำสั่ง `move()` การทำงานที่ได้จะเป็นการทำงานที่สืบทอดมาจากคลาสใด

จงเขียนแผนภาพแสดงความสัมพันธ์การสืบทอดระหว่างคลาสทั้งหมดที่มีในปฏิบัติการนี้

10.4 โจทย์ปัญหา

1. จงสร้างคลาสชื่อ `Department` ทำหน้าที่แทนภาควิชาใด ๆ โดยรองรับการเรียกด้วยคำสั่งต่อไปนี้
 - `add(who)` ทำการเพิ่ม `who` เข้าไปในภาควิชาหาก `who` เป็น object ของคลาส `Prof` และยังไม่อยู่ในภาควิชา นั้น หากไม่ตรงตามเงื่อนไขจะไม่เกิดอะไรขึ้น
 - `get_all_subj()` คืนค่าเป็นลิสต์ของทุปเปิล (`prof`, `subj`) เมื่อ `prof` เป็นสมาชิกในภาควิชา นั้น และ `subj` เป็นวิชาที่อยู่ในลิสต์ `teaching` ของ `prof`
 - `name` เป็น instant variable เก็บชื่อภาควิชา

ตัวอย่างการใช้งานเช่น

```
>>> p1 = Prof('Poonna', 'Full')
>>> p2 = Prof('Anan', 'Full')
>>> p1.add_teaching('Java')
>>> p1.add_teaching('F#')
>>> p2.add_teaching('AI')
>>> cpe = Department('Computer Engineering')
>>> cpe.add(p1)
>>> cpe.add(p2)
>>> cpe.get_all_subj()
[('Poonna', 'Java'), ('Poonna', 'F#'), ('Anan', 'AI')]
```

2. จงสร้างคลาสชื่อ `Queue` ให้ทำหน้าที่เป็นคลังเก็บจำนวนเต็มที่รองรับการเรียกคำสั่งต่อไปนี้

- `push(k)` เพิ่มจำนวนเต็ม `k` เข้าไปในคลัง
- `pop()` ดึงข้อมูลตัวที่อยู่ในคลังมานานที่สุดออก โดยคืนค่าข้อมูลตัวนั้นออกมา และลบข้อมูลนั้นออกจากคลัง
- `is_empty()` คืนค่าเป็น `True` ถ้าคลังข้อมูลไม่เหลือจำนวนเต็มใดเก็บอยู่เลย ไม่เช่นนั้นคืนค่าเป็น `False`

โดยให้เขียนโค้ดป้องกันกรณีที่ผู้ใช้เรียก `pop()` ในขณะที่คลังข้อมูลไม่มีข้อมูลอยู่ด้วย โดยกำหนดให้ หากคลังข้อมูลว่างอยู่และถูกเรียก `pop()` ก็ไม่ต้องมีการทำงานใดเกิดขึ้น

ตัวอย่างการใช้งานเป็นดังนี้

```
>>> q = Queue()
>>> q.push(5)
>>> q.push(3)
>>> q.push(19)
>>> q.push(10)
>>> while not q.is_empty():
        print(q.pop())
5
3
19
10
```

3. จงสร้างคลาสชื่อ `PriorityQueue` ให้สืบทอดจากคลาส `Queue` ในข้อที่แล้ว ให้รองรับการเรียกคำสั่งเช่นเดียวกัน แต่ให้คำสั่ง `pop()` นั้นทำการคืนค่าและลบจำนวนเต็มที่มีค่าน้อยที่สุดในคลังออกมาแทน

ตัวอย่างการใช้งานเป็นดังนี้

```
>>> q = PriorityQueue()
>>> q.push(5)
>>> q.push(3)
>>> q.push(19)
>>> q.push(10)
>>> while not q.is_empty():
        print(q.pop())
3
5
10
19
```