01076105, 01075106

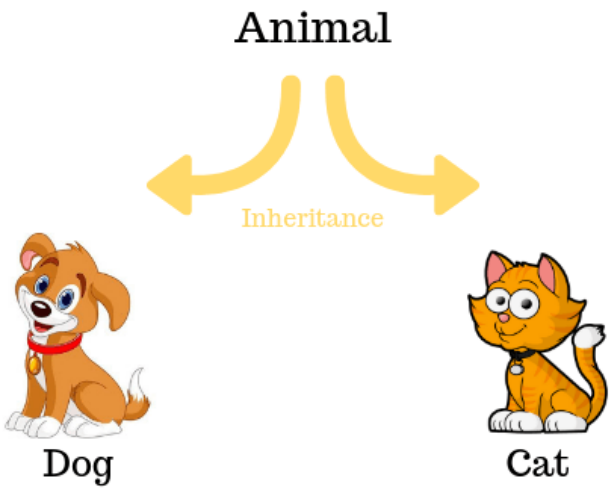
Object Oriented Programming

Object Oriented Programming Project

Inheritance

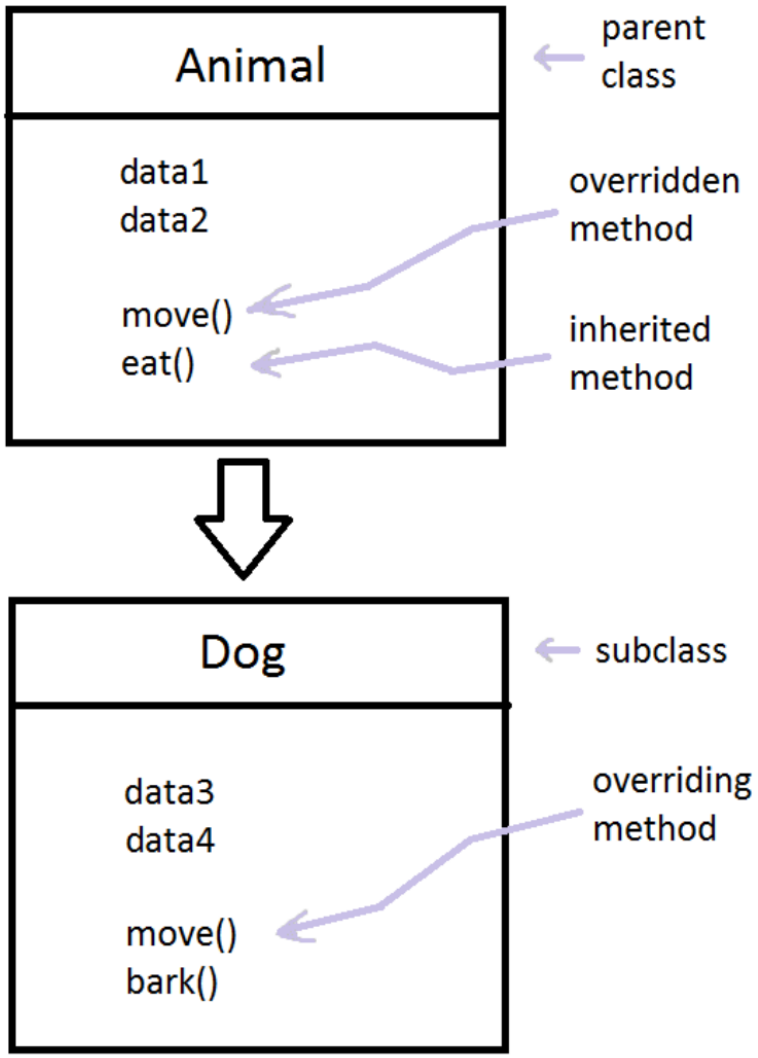
Inheritance

- Inheritance เป็น 1 ใน 3 คุณสมบัติหลักของ Object Oriented Programming (encapsulation, inheritance, polymorphism)
- Inheritance คือ ความสามารถในการสืบทอดคุณสมบัติจาก Class อื่น (เรียก Class ที่สืบทอดว่า Super Class และเรียกตัวเองว่า Subclass)



Inheritance

- ประโยชน์ของ Inheritance
  - ลดความซ้ำซ้อนของ Code (หลักการเขียนโปรแกรม คือ เมื่อมี code ที่ซ้ำกันหรือคล้ายกัน ให้หาทางลด)
  - Reuse Code
  - ทำให้ Code อ่านได้ง่ายขึ้น
- จากรูป ถ้าเพิ่มสัตว์ชนิดอื่นๆ ก็จะทำได้ง่าย และกำหนดเฉพาะคุณลักษณะที่เพิ่มเติม



Inheritance

- Class ที่จะ Inherit จาก Class อื่น มีหลักดังนี้
  - ต้องเป็นส่วนย่อยของ Super Class เช่น ถ้า Super Class คือ Car แล้ว Subclass คือ Trunk ได้ เพราะรถบรรทุก “เป็น” รถยนต์ประเภทหนึ่ง
  - มีการกำหนดลักษณะเฉพาะเพิ่มเติม เช่น รถบรรทุก อาจมี นน. บรรทุก พูดโดยรวม คือ Super Class จะมีลักษณะ “ทั่วไป” แต่ Subclass มีลักษณะ “เฉพาะ”
- Class หนึ่ง สามารถ Inherit จากหลาย Class ได้ เรียกว่า Multiple Inheritance (บางภาษาไม่มีคุณลักษณะนี้) และ Class ก็ถูก Inherit จากหลาย Class ได้เช่นกัน

Inheritance

- Inheritance ช่วย Reuse Code อย่างไร (จากรูป พบ Code ซ้ำซ้อน มาก)

```
class Programmer(object):
    salary = 100000
    monthly_bonus = 500

    def __init__(self, name, age, address, phone, programming_languages):
        self.name = name
        self.age = age
        self.address = address
        self.phone = phone
        self.programming_languages = programming_languages


class Engineer(object):
    salary = 100000
    monthly_bonus = 500

    def __init__(self, name, age, address, phone, bilingual):
        self.name = name
        self.age = age
        self.address = address
        self.phone = phone
        self.bilingual = bilingual
```

Inheritance

- หลังจากใช้ Inheritance

```python
class Employee:
    salary = 100000
    monthly_bonus = 500

    def __init__(self, name, age, address, phone):
        self.name = name
        self.age = age
        self.address = address
        self.phone = phone


class Programmer(Employee):
    def __init__(self, name, age, address, phone, programming_languages):
        Employee.__init__(self, name, age, address, phone)
        self.programming_languages = programming_languages

    # Inherits from Employee


class Assistant(Employee):
    def __init__(self, name, age, address, phone, bilingual):
        Employee.__init__(self, name, age, address, phone)
        self.bilingual = bilingual
```

Inheritance

- รูปแบบการใช้งาน Inheritance

```
class Superclass:
    pass

class Subclass(Superclass)
    pass
```

- เมื่อจะ Inherit มาจาก Class ใด ให้ใส่วงเล็บต่อท้ายเอาไว้
- ใน python เวอร์ชันเก่า ทุก Class จะ Inherit มาจาก Object จะวงเล็บ Object หมดทุก Class แต่ในเวอร์ชันหลังๆ ได้ตัดออก เพื่อให้ดูง่าย

Inheritance

- ตัวอย่าง

```
class Polygon:
    pass


class Triangle(Polygon):
    pass
```

- เพิ่ม Class

```
class Square(Polygon):
    pass
```

- เราสามารถตรวจสอบว่า Class เป็น Subclass ของ Class ใดหรือไม่
- ใช้ฟังก์ชัน issubclass เช่น issubclass(Square, Polygon)

Inheritance

- ตัวอย่างจะเห็นว่า ในการสร้าง Triangle ต้องใส่ parameter เพราะ Inherit มา

```python
class Polygon:

    def __init__(self, num_sides, color):
        self.num_sides = num_sides
        self.color = color


class Triangle(Polygon):
    pass


my_triangle = Triangle(3, "Blue")

print(my_triangle.num_sides)
print(my_triangle.color)
```

Inheritance

- แต่ในกรณีที่ Subclass มี __init__ ของตนเอง จะไม่ Inherit จาก Super Class โดยอัตโนมัติ

```python
class Polygon:

    def __init__(self, num_sides, color):
        self.num_sides = num_sides
        self.color = color


class Triangle(Polygon):

    def __init__(self, base, height):
        self.base = base
        self.height = height
```

```python
my_triangle = Triangle(3, "Blue")

print(my_triangle.num_sides)    # Error
print(my_triangle.color)        # Error
```

Inheritance

- วิธีการเขียนกรณีมี __init__ ของตนเอง

```python
class Polygon:

    def __init__(self, num_sides, color):
        self.num_sides = num_sides
        self.color = color


class Triangle(Polygon):

    NUM_SIDES = 3

    def __init__(self, base, height, color):
        Polygon.__init__(self, Triangle.NUM_SIDES, color)
        self.base = base
        self.height = height
```

Inheritance : Quiz

- ให้ list Instance Attribute ทั้งหมด ของ Enemy Class

```
1  class Sprite:
2
3      def __init__(self, x, y, speed, direction):
4          self.x = x
5          self.y= y
6          self.speed = speed
7          self.direction = direction
8
9
10 class Enemy(Sprite):
11
12     def __init__(self, x, y, speed, direction, num_lives):
13         Sprite.__init__(self, x, y, speed, direction)
14           self.num_lives = lives
```

Inheritance : Quiz

- ใน Instance ของ Puppy Class จะมี Attribute อะไรบ้าง

```
1  class Dog(object):
2
3      def __init__(self, name, age, breed):
4          self.name = name
5          self.age = age
6          self.breed = breed
7
8  class Puppy(Dog):
9
10     def __init__(self, is_vaccinated):
11         self.is_vaccinated = is_vaccinated
```

Inheritance

- เราสามารถใช้ super() ในการแทน Super Class ที่ขึ้นไป 1 ชั้น (ไม่มี self)

```python
class Polygon:

    def __init__(self, num_sides, color):
        self.num_sides = num_sides
        self.color = color


class Triangle(Polygon):

    NUM_SIDES = 3

    def __init__(self, base, height, color):
        super().__init__(Triangle.NUM_SIDES, color)
        self.base = base
        self.height = height
```

Inheritance

- ตัวอย่าง

```python
class Employee:

    def __init__(self, full_name, salary):
        self.full_name = full_name
        self.salary = salary


class Programmer(Employee):

    def __init__(self, full_name, salary, programming_language):
        super().__init__(full_name, salary)
        self.programming_language = programming_language
```

Inheritance : exercise

- จาก Class Mammal ที่กำหนดให้ ให้สร้าง Class Panda ให้เพิ่มข้อมูลดังนี้
  - Class Attribute is_dangered = True
  - Instance Attribute code
- ทดลองสร้าง Instance my_panda แล้วทดสอบ

```
class Mammal:
    def __init__(self, name, age, health, num_offspring, years_in_captivity):
        self.name = name
        self.age = age
        self.health = health
        self.num_offspring = num_offspring
        self.years_in_captivity = years_in_captivity
```

Inheritance

- Multiple Inheritance

```
class Rectangle:
    def __init__(self, length, width, color):
        self.length = length
        self.width = width
        self.color = color


class GUIElement:
    def click(self):
        print("The object was clicked...")


class Button(Rectangle, GUIElement):
    def __init__(self, length, width, color, text):
        Rectangle.__init__(self, length, width, color)
        self.text = text
```

Inheritance : Exercise

- จาก Class Pizza

```python
class Pizza:
    def __init__(self, size, toppings, price, rating):
        self.size = size          # "Small", "Medium", or "Large"
        self.toppings = toppings  # A list of toppings
        self.price = price
        self.rating = rating      # Scale from 1 to 5
```

- สร้าง Class PizzaMargherita โดยเพิ่ม Instance Attribute has_extra_cheese โดย Inherit มาจาก Pizza
- สร้าง Class PizzaMarinara Instance Attribute has_extra_basil โดย Inherit มาจาก Pizza

Inheritance : Quiz

```
        ElectronicDevice
             /   \
       Computer    TV
         /  \
  Desktop Laptop
```

`TV` is a **(1)** _____ of `ElectronicDevice`.

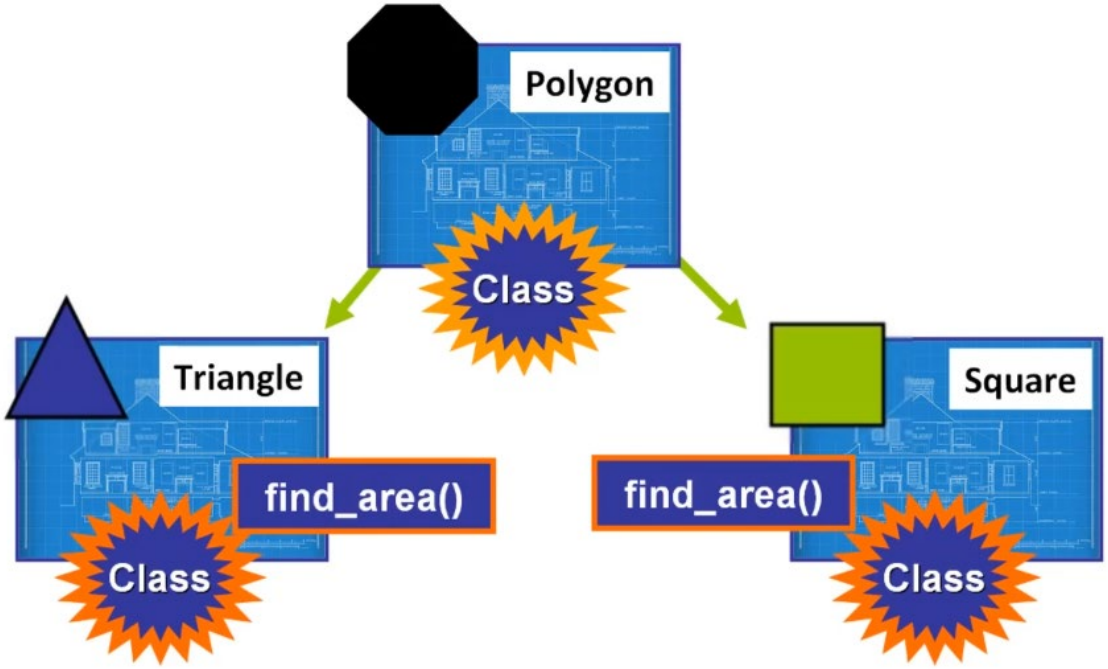
`Computer` is the **(2)** _____ of `Desktop` and `Laptop`.

`ElectronicDevice` is the **(3)** _____ of `Computer` and `TV`.

- ○ (1) Subclass ; (2) Superclass ; (3) Superclass
- ○ (1) Superclass ; (2) Subclass ; (3) Subclass
- ○ (1) Subclass ; (2) Superclass ; (3) Subclass

Inheritance

- ที่ผ่านมากล่าวถึงเฉพาะ Attribute แต่ในส่วนของ Method ก็จะมีลักษณะเช่นเดียวกัน คือ สามารถอ้างถึง Method ได้ทั้ง Super Class และ Class ตัวเอง
- ใน Sub Class จะมี Method ที่มีวิธีการทำงานต่างกัน (แม้จะชื่อเดียวกัน)

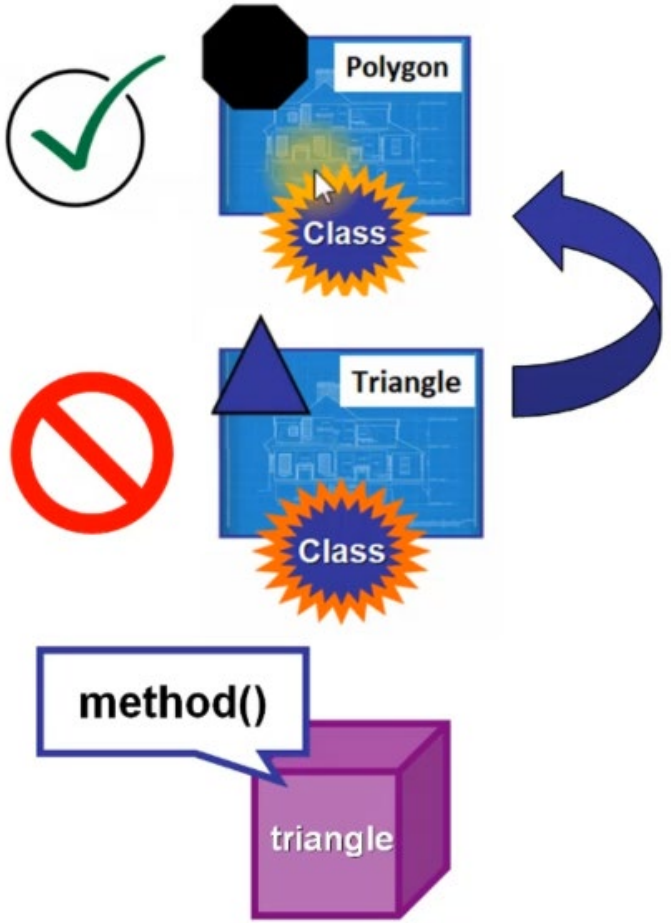


Inheritance

```python
class Triangle(Polygon):

    NUM_SIDES = 3

    def __init__(self, base, height, color):
        Polygon.__init__(self, Triangle.NUM_SIDES, color)
        self.base = base
        self.height = height

    def find_area(self):
        return (self.base * self.height) / 2


class Square(Polygon):

    NUM_SIDES = 4

    def __init__(self, side_length, color):
        Polygon.__init__(self, Square.NUM_SIDES, color)
        self.side_length = side_length

    def find_area(self):
        return self.side_length ** 2
```

Inheritance

- การค้นหา Method จะใช้หลักการตามรูป คือ จะค้นจากจาก Class ลำดับขึ้นไป 1 ชั้น
- ถ้าไม่พบ จึงจะหาใน Class ลำดับเหนือขึ้นไปเรื่อยๆ
- ดังนั้น Method ใน Super Class และ Subclass อาจมีชื่อเดียวกันได้ โดย Method ใน Class ล่างกว่าจะถูกเรียกใช้ก่อน



Inheritance

- การกำหนด Method ใหม่ทับ Method เดิมเรียกว่า Method Overriding

```python
class Teacher:

    def __init__(self, full_name, teacher_id):
        self.full_name = full_name
        self.teacher_id = teacher_id

    def welcome_students(self):
        print(f"Welcome to class!, I'm your teacher. My name is {self.full_name}")


class ScienceTeacher(Teacher):

    def welcome_students(self):
        print(f"Science is amazing.")
        print(f"Welcome to class. I'm your teacher: {self.full_name}")
```

Inheritance

- ในกรณีที่ใน Sub Class ต้องการเรียก Method ของ Super Class

```python
class Teacher:

    def __init__(self, full_name, teacher_id):
        self.full_name = full_name
        self.teacher_id = teacher_id

    def welcome_students(self):
        print(f"Welcome to class!, I'm your teacher. My name is {self.full_name}")


class ScienceTeacher(Teacher):

    def welcome_students(self):
        print(f"Science is amazing.")
        Teacher.welcome_students(self)
```

Inheritance

- ในกรณีที่ใน Sub Class ต้องการเรียก Method ของ Super Class (อีกแบบ)

```python
class Teacher:

    def __init__(self, full_name, teacher_id):
        self.full_name = full_name
        self.teacher_id = teacher_id

    def welcome_students(self):
        print(f"Welcome to class!, I'm your teacher. My name is {self.full_name}")


class ScienceTeacher(Teacher):

    def welcome_students(self):
        print(f"Science is amazing.")
        super().welcome_students()
```

Inheritance : Quiz

- จาก Code ต่อไปนี้ ถ้าเรียก greet_students ใน ScienceProfessor จะแสดง?

```python
class Professor:

    def __init__(self, name, age, course):
        self.name = name
        self.age = age
        self.course = course

    def greet_students(self):
        print("Welcome, dear students")

class ScienceProfessor(Professor):

    def __init__(self, name, age, course):
        Professor.__init__(self, name, age, course)

    def greet_students(self):
        print("Welcome to our Science class, dear students!")
```

Inheritance : Quiz

```
1  class A:
2
3      def x(self):
4          print("Class A")
5
6  class B(A):
7
8      def x(self):
9          print("Class B")
10
11 a = A()
12 b = B()
13
14 # Output?
15 a.x()
16 b.x()
```

- Code นี้จะแสดงอะไร

○
```
1  Class A
2  Class A
```

○
```
1  Class A
2  Class B
```

○
```
1  Class B
2  Class B
```

Inheritance : Example

- กำหนดให้ class Contact ทำหน้าที่เก็บข้อมูลการติดต่อ (ประกอบด้วยชื่อกับ E-Mail ตามรูป)

```python
class Contact:
    all_contacts = []

    def __init__(self, name, email):
        self.name = name
        self.email = email
        Contact.all_contacts.append(self)


c = Contact("Some body", "somebody@example.net")
s = Contact("Sup Plier", "supplier@example.net")
```

Inheritance : Example

- กำหนดเพิ่มเติมว่าประเภทย่อย Supplier สามารถรับ order ได้ จึงได้เพิ่ม class Supplier โดย inherit มาจาก Contact

```
class Supplier(Contact):
    def order(self, order):
        print("Order : ", order)


c = Contact("Some body", "somebody@example.net")
s = Supplier("Sup Plier", "supplier@example.net")
print(c.name, c.email)
print(s.name, s.email)

# c.order("order")
s.order("order")
```

Inheritance : Example

- แต่เนื่องจาก list ของ python ไม่มีความสามารถในการ search ดังนั้นจึง Inherit class List และเพิ่มความสามารถ search เข้าไป (Extending build-in)

```
class ContactList(list):
    def search(self, name):
        matching_contacts = []
        for contact in self:
            if name in contact.name:
                matching_contacts.append(contact)
        return matching_contacts
```

Inheritance : Example

- โดยมีการแก้ไข class Contact ใหม่ ทำให้ค้นหาได้

```
class Contact:
    all_contacts = ContactList()

    def __init__(self, name, email):
        self.name = name
        self.email = email
        self.all_contacts.append(self)


c1 = Contact("John A", "johna@example.net")
c2 = Contact("John B", "johnb@example.net")
c3 = Contact("John C", "johnc@example.net")

print([c.name for c in Contact.all_contacts.search("John")])
```

Inheritance : Example

- ในกรณีที่เป็นเพื่อน อาจมีการเพิ่ม Attribute phone เข้าไป โดยการ Inherit จาก Contact มาได้เช่นกัน

```python
from contact import Contact


class Friend(Contact):
    def __init__(self, name, email, phone):
        super().__init__(name, email)
        self.phone = phone
```

Inheritance : Example

- สมมติว่า ในระบบมีการส่ง Mail เราอาจสร้างเป็น class MailSender ขึ้นมาเพื่อทำหน้าที่ส่ง mail (เขียนแบบย่อๆ)

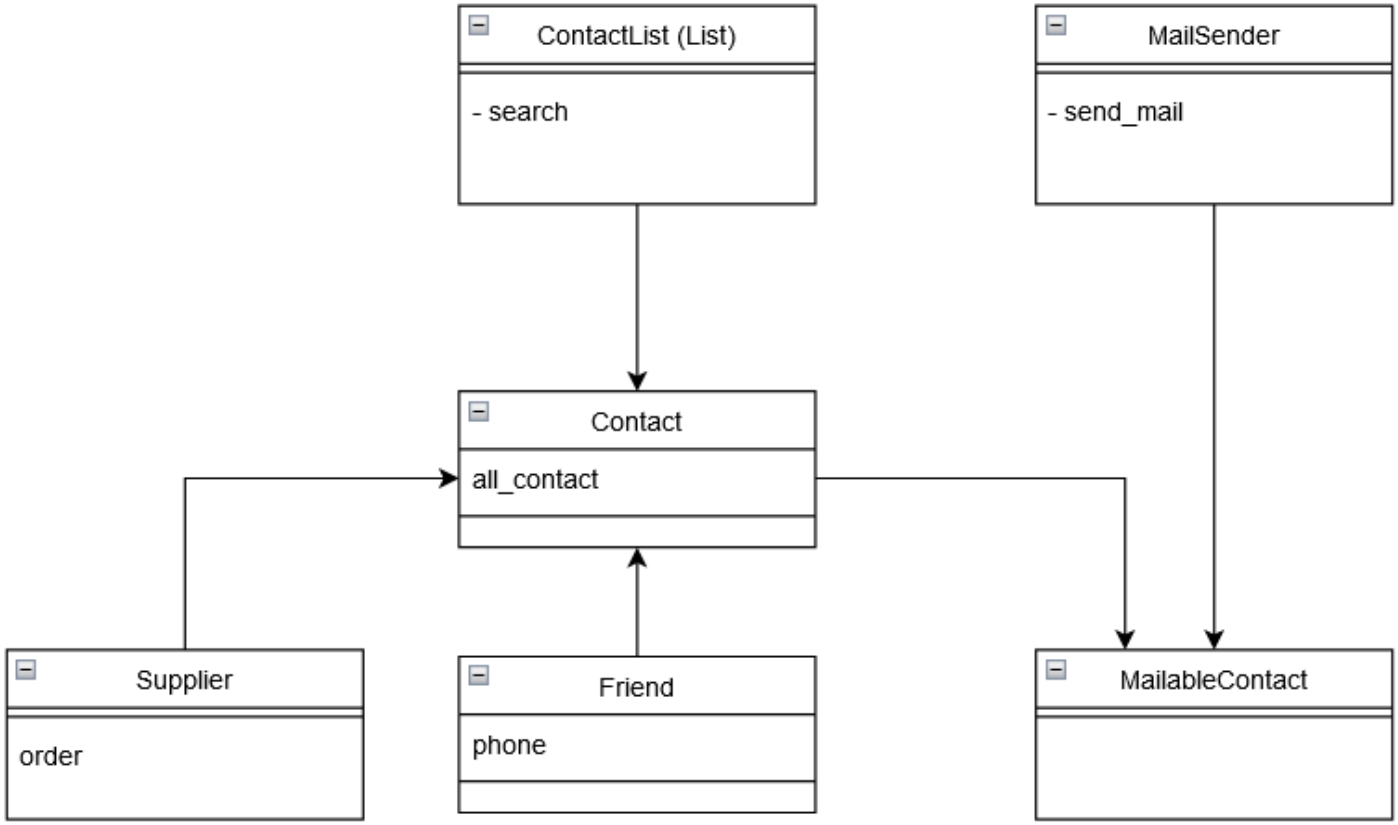
```
class MailSender:
    def send_mail(self, message):
        print("Sending mail to " + self.email)
```

- จากนั้นเราจะสร้าง Contact ที่ส่ง mail ได้

```
class MailableContact(Contact, MailSender):
    pass


e = MailableContact("John Smith", "jsmith@example.net")
e.send_mail("Hello, test e-mail.")
```

Inheritance : Example

- จาก Slide ที่ผ่านมา จะได้ความสัมพันธ์ของ class ดังนี้

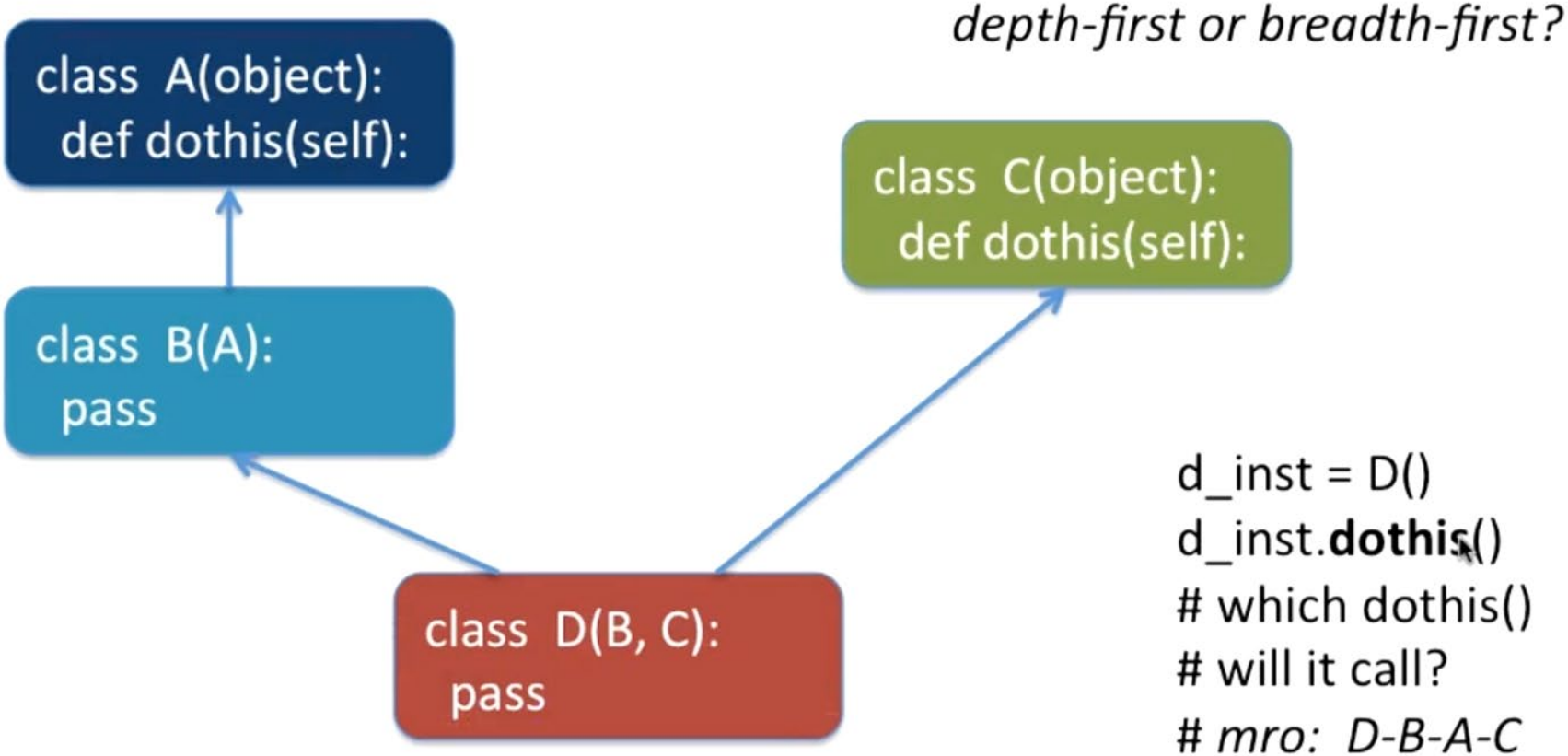


Method resolution order

- เป็นวิธีการของการค้นหา method เพื่อ execute ในกรณีที่มีการ Inherit มาจากหลายๆ Class มีลำดับดังนี้
  - ค้นหาใน Class ที่ถูกเรียกก่อน (หรือ Class ของตัวเอง)

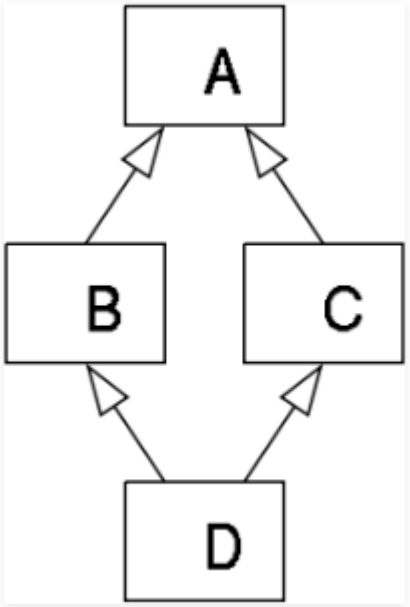
```
1  # Python program showing
2  # how MRO works
3
4  class A:
5      def rk(self):
6          print(" In class A")
7  class B(A):
8      def rk(self):
9          print(" In class B")
10
11 r = B()
12 r.rk()
13
```

Inheritance : depth–first search



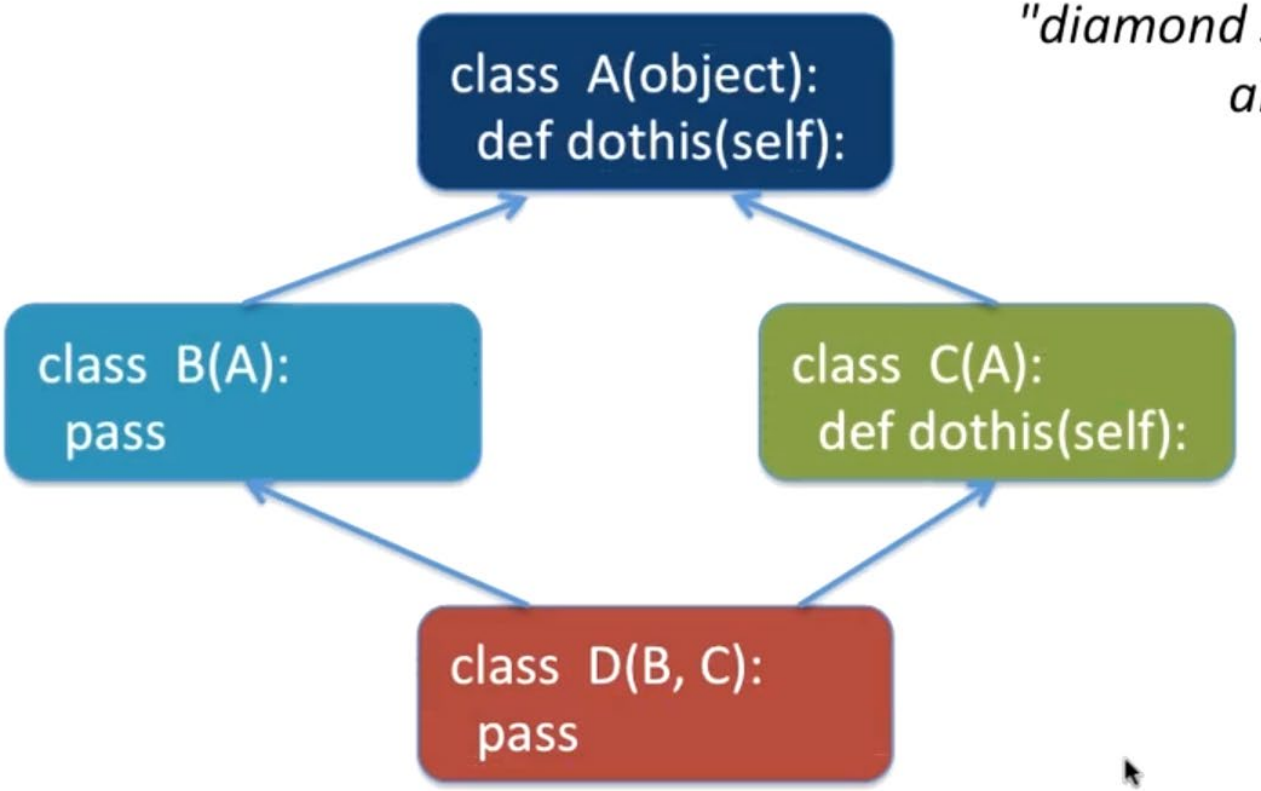
Method resolution order

- คลาส D Inherit จาก B, C ตามรูป
- Diamond Problem
- จะแสดงผลอะไร




```python
# Python program showing
# how MRO works

class A:
    def rk(self):
        print(" In class A")
class B(A):
    def rk(self):
        print(" In class B")
class C(A):
    def rk(self):
        print("In class C")

# classes ordering
class D(B, C):
    pass

r = D()
r.rk()

```

Inheritance : diamond problem



"diamond shape" inheritance: ambiguous!

```
d_inst = D()
d_inst.dothis()
# which dothis()
# will it call?
# mro:  D-B-C-A
```

Method resolution order

- เราสามารถตรวจสอบ MRO ได้

```
# Python program to show the order
# in which methods are resolved

class A:
    def rk(self):
        print(" In class A")
class B:
    def rk(self):
        print(" In class B")

# classes ordering
class C(A, B):
    def __init__(self):
        print("Constructor C")

r = C()

# it prints the lookup order
print(C.__mro__)
print(C.mro())
```

```
Constructor C
(<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>, <class 'object'>)
[<class '__main__.C'>, <class '__main__.A'>, <class '__main__.B'>, <class 'object'>]
```

Inheritance : Example

- โดย MRO ทำให้เราทราบว่า inheritance trees จะทำงานอย่างไร
- แต่การมี Class ใน Tree มากๆ จะทำให้ inheritance trees มีความซับซ้อนและยากจะจัดการ
- ภาษา Python จึงมีองค์ประกอบหนึ่งเรียกว่า Mixin ซึ่งเป็น Class เล็กๆ ที่ทำหน้าที่เฉพาะ โดยจะไม่เข้าไปอยู่ใน inheritance trees เพียงทำหน้าที่เสริมให้กับ Class เท่านั้น
- จากตัวอย่างใน Slide 34 Class MailSender เป็น Mixins
- การใช้ Mixins เพื่อหลีกเลี่ยงการเกิด Diamond Problem
- ควรหลีกเลี่ยงการใช้ multiple inheritance ให้มากที่สุด

Mixin classes in Python

- จะเห็นว่า ResizableMixin ไม่ได้ Inherit จาก GraphicalEntity

```python
class GraphicalEntity:
    def __init__(self, pos_x, pos_y, size_x, size_y):
        self.pos_x = pos_x
        self.pos_y = pos_y
        self.size_x = size_x
        self.size_y = size_y


class ResizableMixin:
    def resize(self, size_x, size_y):
        self.size_x = size_x
        self.size_y = size_y


class ResizableGraphicalEntity(GraphicalEntity, ResizableMixin):
    pass
```

Inheritance : Example

- class MailSender สร้างขึ้นมาเพื่อทำหน้าที่ส่ง mail (เขียนแบบย่อๆ)

```
class MailSender:
    def send_mail(self, message):
        print("Sending mail to " + self.email)
```

- จากนั้นเราจะสร้าง Contact ที่ส่ง mail ได้

```
class MailableContact(Contact, MailSender):
    pass


e = MailableContact("John Smith", "jsmith@example.net")
e.send_mail("Hello, test e-mail.")
```

Inheritance : Example

- กรณีของ send_mail ในตัวอย่าง เรามีแนวทางในการใช้งาน
  - อาจใช้ single inheritance และเพิ่ม send_mail ลงใน subclass แต่ก็จะทำให้เกิด duplicate code ในทุกครั้งที่มีการเพิ่ม send_mail
  - สร้าง function send_mail ขึ้นมาใช้งาน โดยไม่ใช้ class
  - ใช้รูปแบบ Composition (กล่าวถึงภายหลัง)
  - ใช้เทคนิค monkey-patch (กล่าวถึงภายหลัง)

Inheritance : Example

- มาสำรวจปัญหาของ multiple inheritance จาก Class Friend
- สมมติว่าต้องการเพิ่ม address จะทำอย่างไรได้บ้าง
- เช่น เพิ่ม attribute เข้าไปใน class โดยอาจใส่ใน Tuple หรือ Dic

```
from contact import Contact


class Friend(Contact):
    def __init__(self, name, email, phone):
        super().__init__(name, email)
        self.phone = phone
```

Inheritance : Example

- อีกวิธี คือ สร้าง Class ขึ้นมาเก็บที่อยู่ (ข้อดี คือ มี method ได้)

```
class AddressHolder:
def __init__(self, street, city, state, code):
    self.street = street
    self.city = city
    self.state = state
    self.code = code
```

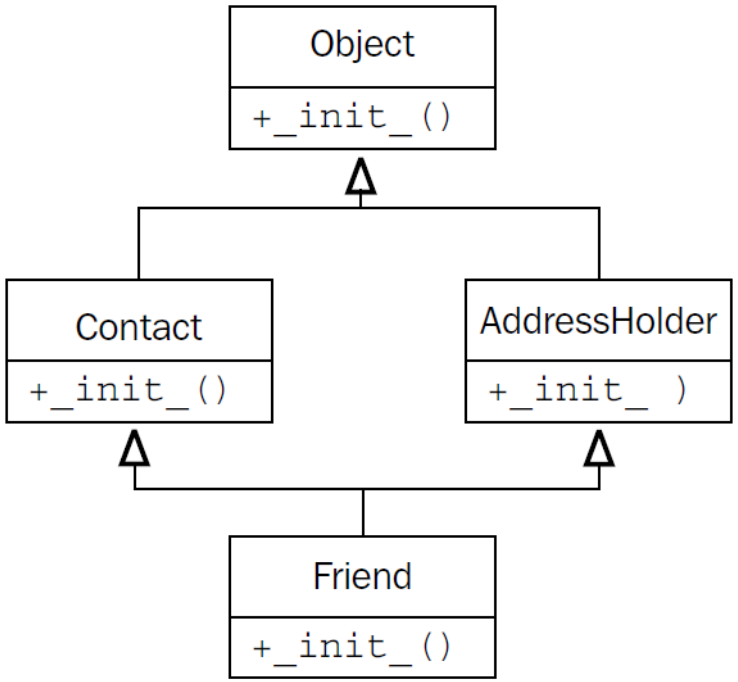
Inheritance : Example

- จะเห็นว่าสามารถ ใส่ที่อยู่ได้ โดยส่วนของ Contact ก็จะส่งต่อไปที่ Super Class : Contact และส่วนของที่อยู่ ก็จะส่งต่อไปที่ Super Class : AddressHolder (สังเกตว่าเราไม่ตั้งชื่อว่า Address เฉยๆ เพราะเป็นที่อยู่ “ของ” มนุษย์)

```
class Friend(Contact, AddressHolder):
    def __init__(self, name, email, phone, street, city, state, code):
        Contact.__init__(self, name, email)
        AddressHolder.__init__(self, street, city, state, code)
        self.phone = phone
```

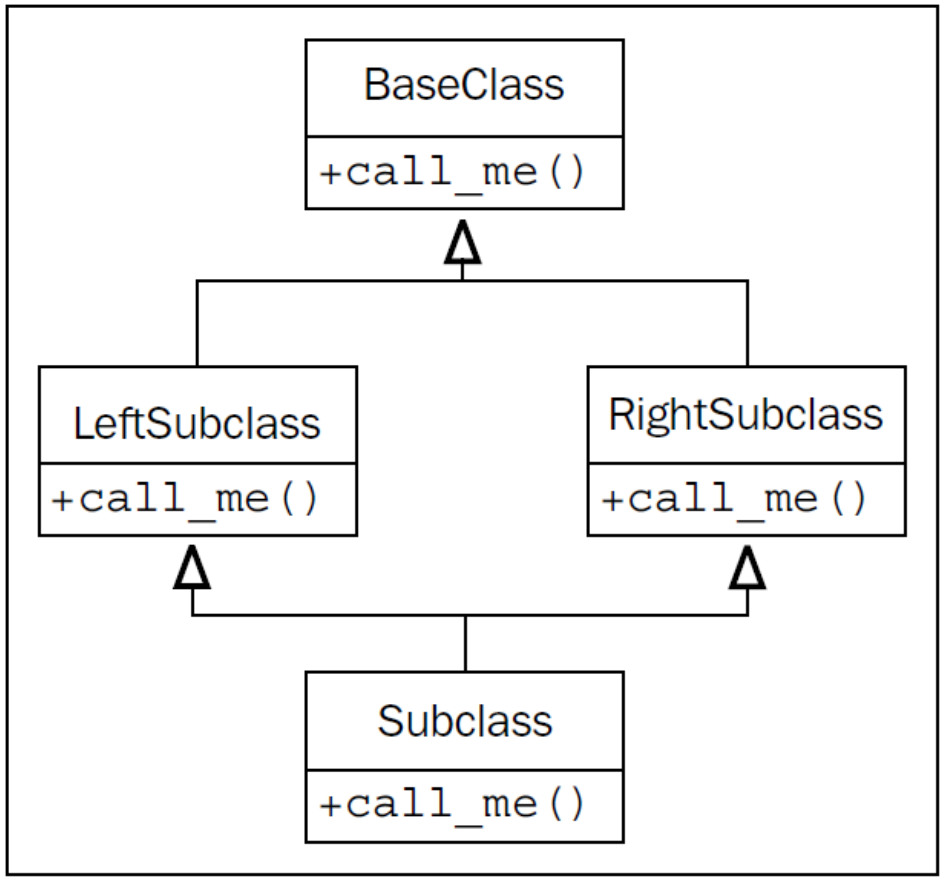
Inheritance : diamond problem

- ปัญหาหนึ่งที่อาจเกิดขึ้นใน Inheritance เรียกว่า diamond problem
- จากรูปเนื่องจากทุก Class Inherit มาจาก Class Object ซึ่งแปลว่า Top Super Class จะ Init 2 ครั้ง แต่กรณีนี้ไม่เป็นปัญหาเพราะใน Object ไม่มีอะไร



Inheritance : diamond problem

- ลองดูตัวอย่างต่อไปนี้ ทุก Class มี method call_me()



Inheritance : diamond problem

```python
class BaseClass:
    num_base_calls = 0

    def call_me(self):
        print("Calling method on Base Class")
        self.num_base_calls += 1

class LeftSubclass(BaseClass):
    num_left_calls = 0

    def call_me(self):
        BaseClass.call_me(self)
        print("Calling method on Left Subclass")
        self.num_left_calls += 1
```

Inheritance : diamond problem

```
class RightSubclass(BaseClass):
    num_right_calls = 0

    def call_me(self):
        BaseClass.call_me(self)
        print("Calling method on Right Subclass")
        self.num_right_calls += 1

class Subclass(LeftSubclass, RightSubclass):
    num_sub_calls = 0

    def call_me(self):
        LeftSubclass.call_me(self)
        RightSubclass.call_me(self)
        print("Calling method on Subclass")
        self.num_sub_calls += 1
```

Inheritance : diamond problem

- ทุก Class มี method call_me ()
- ถ้ามีการเรียกใช้ดังนี้

```
s = Subclass()
s.call_me()
print(s.num_sub_calls, s.num_left_calls,
      s.num_right_calls, s.num_base_calls)
```

- จะมีการทำงานลำดับอย่างไร

Inheritance : diamond problem

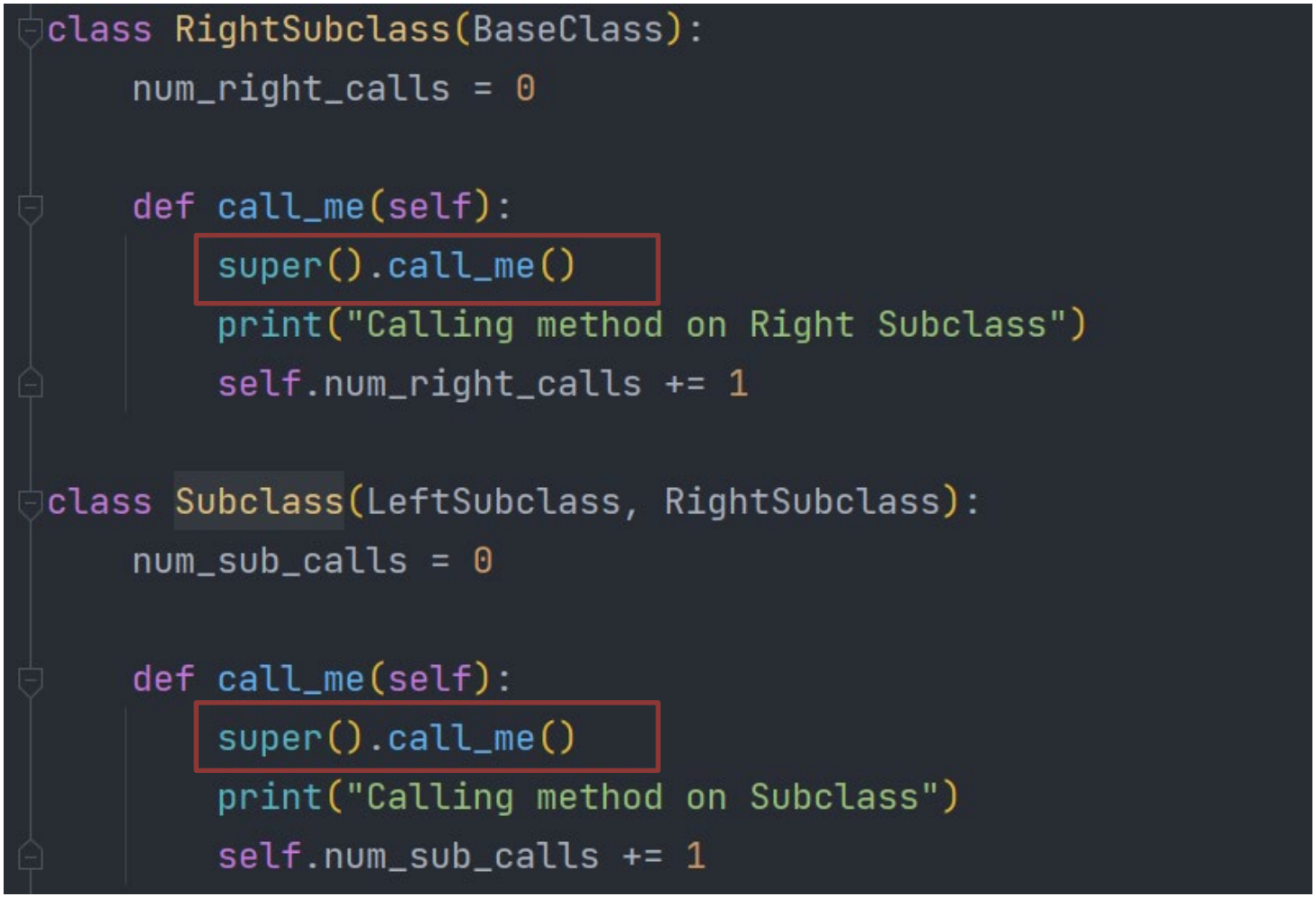
- จะเห็นได้ว่า Base Class มีการเรียกใช้ 2 ครั้ง ซึ่งเป็นสิ่งที่ต้องระวัง เพราะอาจผลที่ไม่พึงประสงค์ได้
- พึงระลึกว่า Method ที่ถูกเรียกนั้น จะค้นหาจาก Next Method ไม่ใช่ Parent Method

```
Calling method on Base Class
Calling method on Left Subclass
Calling method on Base Class
Calling method on Right Subclass
Calling method on Subclass
1 1 1 2
```

Inheritance : diamond problem

```python
class BaseClass:
    num_base_calls = 0

    def call_me(self):
        print("Calling method on Base Class")
        self.num_base_calls += 1

class LeftSubclass(BaseClass):
    num_left_calls = 0

    def call_me(self):
        super().call_me()
        print("Calling method on Left Subclass")
        self.num_left_calls += 1
```

Inheritance : diamond problem



```
class RightSubclass(BaseClass):
    num_right_calls = 0

    def call_me(self):
        super().call_me()
        print("Calling method on Right Subclass")
        self.num_right_calls += 1

class Subclass(LeftSubclass, RightSubclass):
    num_sub_calls = 0

    def call_me(self):
        super().call_me()
        print("Calling method on Subclass")
        self.num_sub_calls += 1
```

Inheritance : diamond problem

- จะเห็นว่าในกรณีนี้ Base Class มีการเรียกใช้ครั้งเดียว
- แต่ลำดับการเรียกใช้จะเห็นว่า มีการเรียกใช้จาก Subclass ไปยัง Left Subclass ไปยัง Right Subclass แล้วจึงไปยัง Base Class ทั้งที่ Base Class เป็น super() ของ Left Subclass

```
Calling method on Base Class
Calling method on Right Subclass
Calling method on Left Subclass
Calling method on Subclass
1 1 1 1
```

Inheritance : Difference set of argument

- กลับมาที่โจทย์เดิม (Friend)

```
class Friend(Contact, AddressHolder):
    def __init__(self, name, email, phone, street, city, state, code):
        Contact.__init__(self, name, email)
        AddressHolder.__init__(self, street, city, state, code)
        self.phone = phone
```

- จะเห็นว่า 2 class มีการใช้ argument ที่ไม่ซ้ำกัน
- คำถาม คือ จะใช้ super() แทนได้หรือไม่
- คำตอบ คือ ได้ แต่เนื่องจากใช้ argument list ต่างกัน ก็ต้องส่ง argument ทั้งหมด แล้วให้แต่ละ Class เลือกเฉพาะที่ตนเองใช้

Inheritance : Difference set of argument

- ในกรณีนี้ทุก Class จะมีการเพิ่ม `super().__init__(**kwargs)` เข้าไปเพื่อส่ง argument list ขึ้นไปยัง Super Class ของตนไปเรื่อยๆ
- แต่ละ Class จะเลือกข้อมูลของตน แต่เนื่องจาก python ไม่สามารถตรวจสอบจำนวน argument จึงต้องกำหนด default argument ด้วย

```python
class Contact:
    all_contacts = []

    def __init__(self, name='', email='', **kwargs):
        super().__init__(**kwargs)
        self.name = name
        self.email = email
        Contact.all_contacts.append(self)
```

Inheritance : Difference set of argument

- จะเห็นว่าใน Friend มีการเรียก `super().__init__(**kwargs)` ครั้งเดียว

```python
class AddressHolder:
    def __init__(self, street='', city='', state='', code='', **kwargs):
        super().__init__(**kwargs)
        self.street = street
        self.city = city
        self.state = state
        self.code = code


class Friend(Contact, AddressHolder):
    def __init__(self, phone='', **kwargs):
        super().__init__(**kwargs)
        self.phone = phone
```

Inheritance : Difference set of argument

- วิธีการข้างต้นจะมีปัญหา 1 จุด คือ ใน argument list จะไม่มี phone ดังนั้นหากใน Super Class ใดๆ มีการใช้ phone จะมีปัญหา
- วิธีแก้ 1 : ให้รวม phone ใน **kwargs และให้ class Friend ค้นหาใน dictionary : kwargs[‘phone’]
- วิธีแก้ 2 : เพิ่ม phone เข้าไปภายหลัง kwargs[‘phone’] = phone
- วิธีแก้ 3 : เพิ่ม phone เข้าไปโดยใช้ kwargs.update
- วิธีแก้ 4 : ส่ง phone ใน __init__(phone=phone, **kwargs)

For your attention