01076105, 01076106

Object Oriented Programming

Object Oriented Programming Project

Encapsulation and Abstraction

Class Attribute

- Class Attribute เป็น Attribute ของ Class ไม่ใช่ Attribute ของ Instance
- ทุก Instance ที่สร้างมาจาก Class เดียวกัน จะใช้ Class Attribute ร่วมกัน โดยทุก Instance จะเห็นค่าเดียวกัน
- ปกติเราจะกำหนด Class Attribute ไว้เหนือ __init__()

```
class ClassName:

    # Class Attributes

    # __init__()

    # Methods
```

Class Attribute

- การกำหนด Class Attribute จะเหมือนกับกำหนดตัวแปร

```
<class_attribute> = <value>
```

- ตัวอย่าง

```
class Backpack:

    max_num_items = 10

    def __init__(self):
        self.items = []
```

Quiz

- จากตัวอย่างนี้ อะไร คือ Class Attribute

```
1  class Bird:
2
3      num_wings = 2
4      flies = True
5
6      def __init__(self, name, species, color, sings):
7          self.name = name
8          self.species = species
9          self.color = color
10         self.sings = sings
```

Class Attribute

- การเข้าถึง Class Attribute ใช้ dot notation โดยใช้ชื่อ class แล้วตามด้วย Class Attribute

```
<ClassName>.<class_attribute>
```

- เช่น จาก class Backpack

```
print(Backpack.max_num_items)
```

- การเข้าถึงใน Class ก็ใช้วิธีเดียวกัน
- สามารถใช้ <Object>.<Class Attribute> ได้เช่นกัน ได้ค่าเดียวกัน

Class Attribute

- เราสามารถใช้ Class Attribute ในการนับจำนวน Object ได้ เช่น

```python
class Movie:

    id_counter = 1

    def __init__(self, title, rating):
        self.id = Movie.id_counter
        self.title = title
        self.rating = rating

        Movie.id_counter += 1

my_movie = Movie("Inception", 8.8)
your_movie = Movie("Legends of the fall", 7.5)
print(Movie.id_counter)
```

Class Attribute : Quiz

- จากรูป ข้อใดถูก

```
1  class Rainbow(object):
2
3      num_colors = 5
4
5      def __init__(self, location, latitude, longitude, rating):
6          self.location = location
7          self.latitude = latitude
8          self.longitude = longitude
```

- ○ `Rainbow_num_colors = 6`
- ○ `num_colors = 6`
- ○ `Rainbow.num_colors = 6`

Class Attribute : Quiz

- ความแตกต่างระหว่าง class attributes และ instance attributes คือ
  1. **class attributes** ใช้ร่วมกันในทุก instance ของ class ดังนั้นจึงมีค่าเดียวกัน ในขณะที่ **instance attributes** เป็นของ instance จึงมีค่าเป็นอิสระต่อกัน
  2. **instance attributes** ใช้ร่วมกันในทุก instance ของ class ดังนั้นจึงมีค่าเดียวกัน ในขณะที่ **class attributes** เป็นของ instance จึงมีค่าเป็นอิสระต่อกัน

Class Attribute : Quiz

- การกำหนดค่า class attributes ถูกต้องหรือไม่

```
1  class A:
2
3      self.attr1 = 5
4
5      def __init__(self, param1, param2):
6          self.param1 = param1
7          self.param2 = param2
8
9      # Methods
```

Class Attribute : Quiz

- หลังจากโปรแกรมทำงานแล้ว ค่าของ class attributes attr1 เป็นเท่าไร

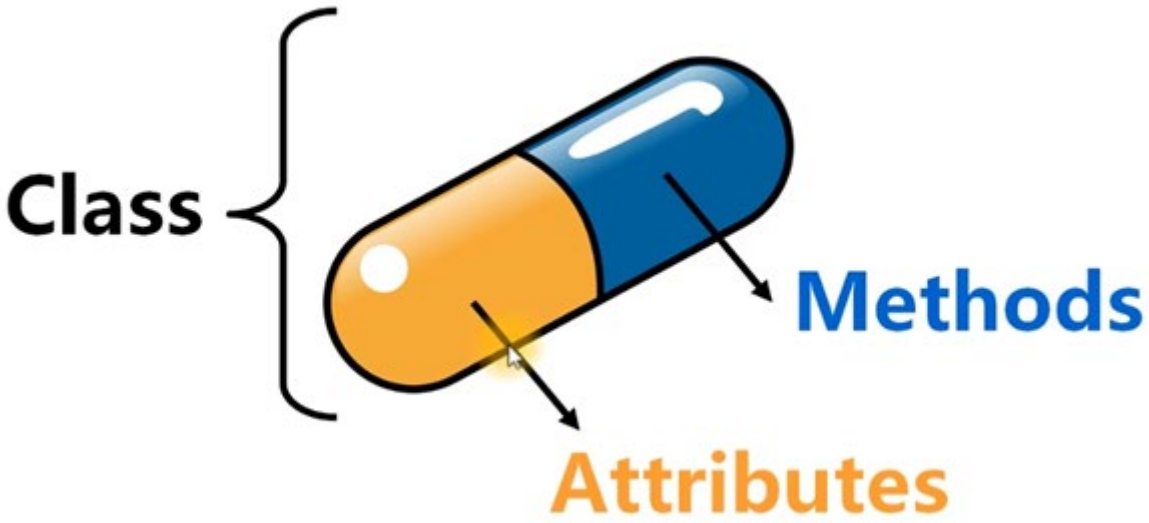
```
1  class A:
2
3      attr1 = 5
4
5      def __init__(self):
6          A.attr1 += 1
7
8  a1 = A()
9  a2 = A()
10 A.attr1 = 26
11 a3 = A()
12 print(A.attr1) # Output?
```

Activity

- ให้ออกแบบ Class สำหรับระบบจ่ายเงินเดือน โดยให้มี class attributes ที่เหมาะสม
- ให้โหลดไฟล์ payroll.py และแก้ไขให้ทำงานได้

Encapsulation

- Encapsulation เป็นคุณสมบัติที่สำคัญของ Object Oriented (มาจากคำว่า capsule: เป็นชิ้นเดียวกัน กับ en = ทำให้) โดยความหมาย คือ การนำ Data กับ Method หรือ State กับ Behavior มารวมไว้ด้วยกัน



Encapsulation

- อีกคำหนึ่งที่เป็นผลมาจาก encapsulation คือ information hiding ซึ่งหมายถึงการจำกัดการเข้าถึง “ข้อมูล” จากภายนอก Object
- ใน Object Oriented จะมีการแบ่งข้อมูลภายใน Object เป็น 2 ส่วน คือ ส่วนที่เป็น Non-Public ซึ่งไม่สามารถเข้าถึงได้จากภายนอก และ Public ซึ่งสามารถเข้าถึงได้จากภายนอก
- ดังนั้นการเข้าถึง attribute ของ Object โดยใช้ dot notation

```
<object>.<attribute> = <new_value>
```

- จึงขัดกับหลักการของ information hiding

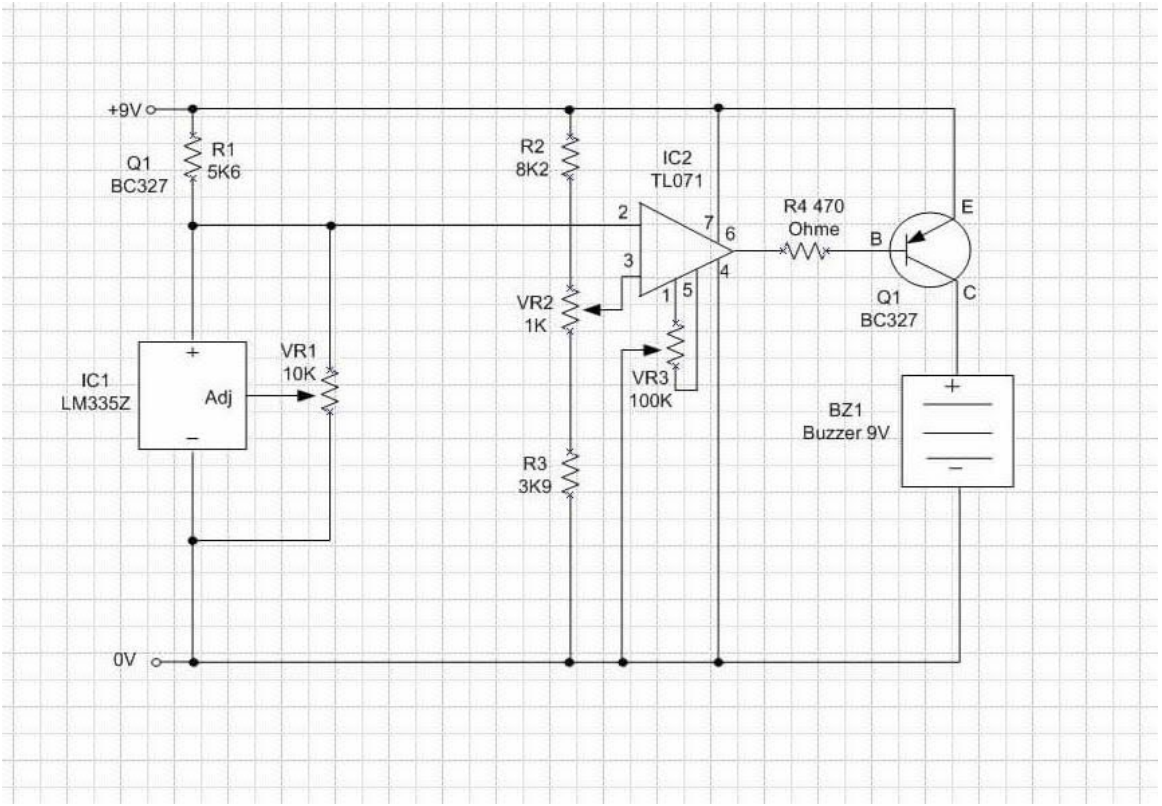
Encapsulation

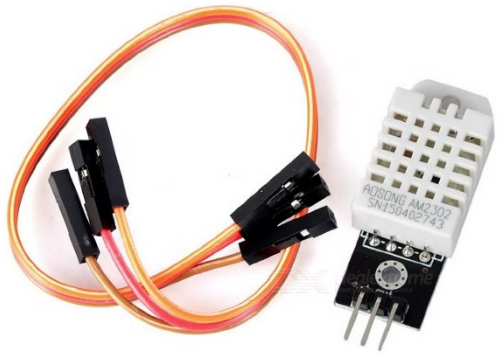
- ยกตัวอย่าง เช่น ในรถยนต์ อาจจะมีข้อมูลบางอย่างที่ให้ผู้ซื้อทราบ แต่ข้อมูลบางอย่างก็อาจจะทราบได้เฉพาะผู้ผลิต
- ดังนั้นในแต่ละ Class จึงจำเป็นต้องกำหนดว่า ข้อมูลใด ที่เป็น Public และข้อมูลใดที่เป็น Non-Public
- ในภาษาบางภาษา จะ strict กับ Public และ Non-Public มาก เช่น ภาษา C++ และ Java
- แต่ในบางภาษา เช่น Python หรือ JavaScript จะไม่ strict ดังนั้นจึงสามารถเข้าถึง Attribute ได้ อย่างไรก็ตามในภาษา Python จะมีวิธีจัดการเป็นพิเศษ ซึ่งจะกล่าวถึงต่อไป

Abstraction

- คำว่า Abstraction มาจากรากศัพท์ Latin 2 คำคือ abs หมายถึง away from และ trahere หมายถึง draw ซึ่งเมื่อแปลรวมกันจะได้ เป็นกรรมวิธีที่กำจัดหรือเอาลักษณะเฉพาะบางประการออกไป เพื่อทำให้เหลือเฉพาะลักษณะเฉพาะเท่าที่จำเป็นเท่านั้น
- ในทาง OOP การทำ Abstraction คือการซ่อน attribute และ behavior ทุกอย่างที่ไม่เกี่ยวข้องกับ object เพื่อช่วยลดความซับซ้อน และเพิ่มประสิทธิภาพในการใช้งาน
- สรุป Abstraction คือการแสดงถึงคุณลักษณะและพฤติกรรมของ object เท่าที่จำเป็นต้องรับรู้และใช้งาน โดยซ่อนส่วนที่เหลือเอาไว้เพื่อไม่ให้เกิดความสับสน (data hiding และ encapsulation) วิธีการสร้างให้ระบบเรามีคุณสมบัติ Abstraction คือใช้หลักของ Abstract Class หรือ Interface

Abstraction

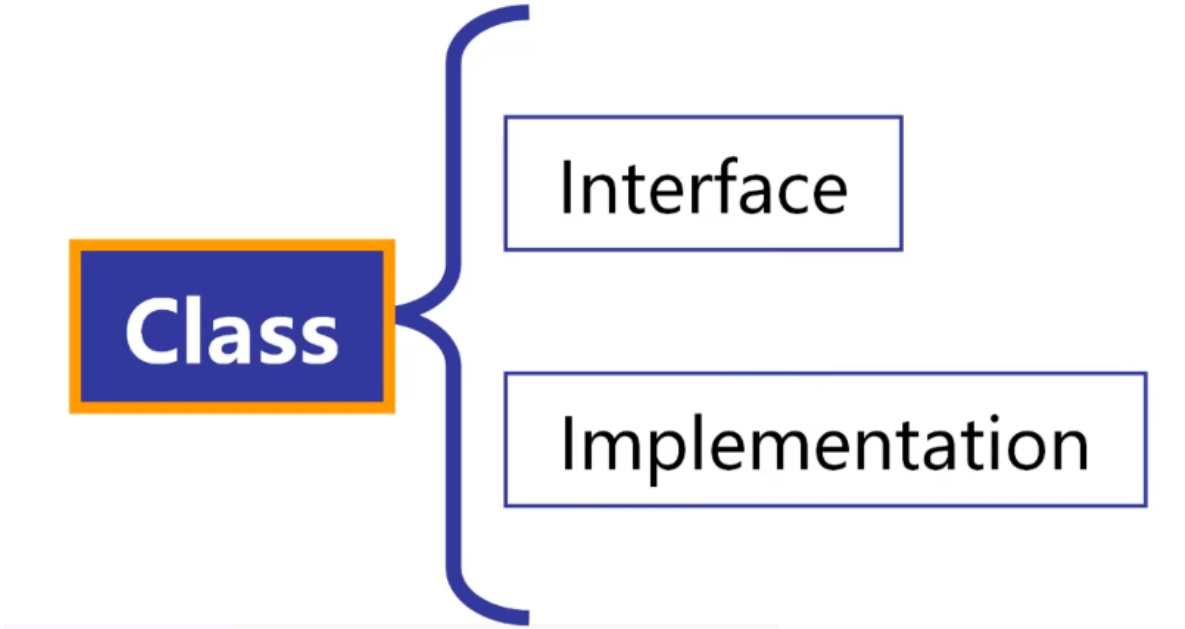




วงจรวัดอุณหภูมิ

Abstraction

- Interface คือ ส่วนของ Class ที่มองเห็นได้จากภายนอก
- Implementation คือ รายละเอียดการทำงานหรือฟังก์ชันภายใน Class



Abstraction

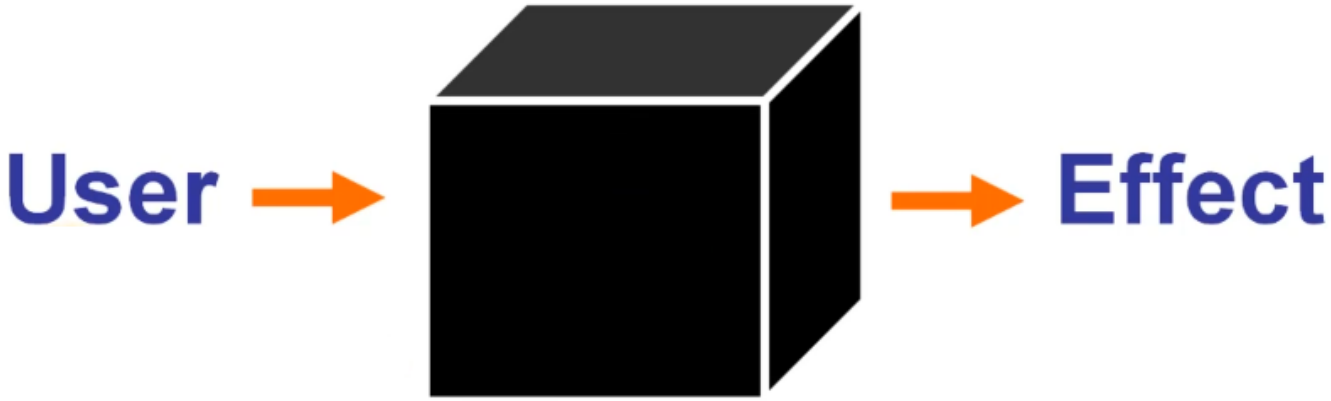
- ตัวอย่างของ Abstraction

Interface

Implementation

Abstraction

- เราอาจมอง Class หรือ Object เป็น กล่องดำ ที่เห็นเฉพาะผลการทำงาน เมื่อใส่ Input หรือ message เข้าไป
- นี่เป็นหลักการที่สำคัญของ OOP คือ Information Hiding ซึ่งทำให้เกิด Abstraction

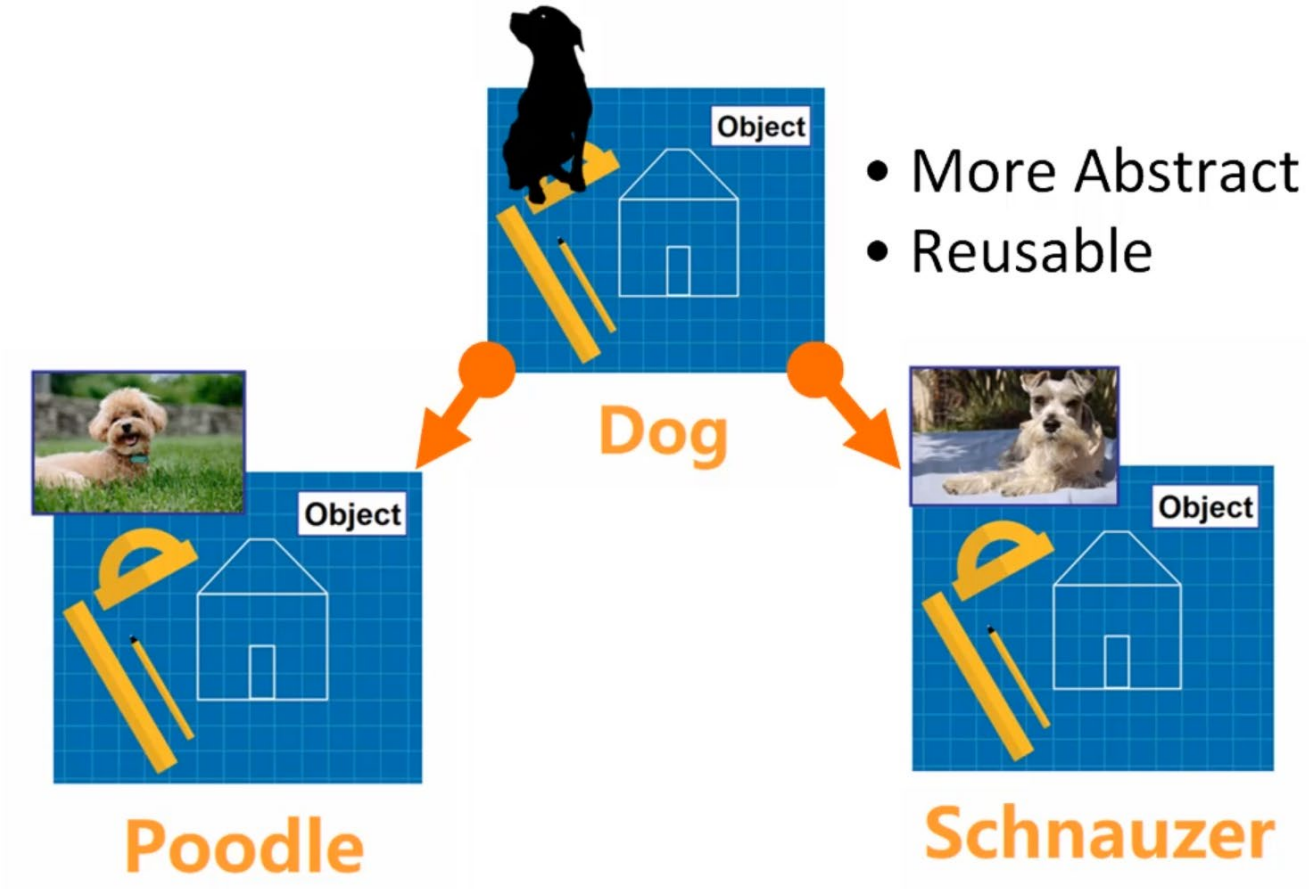


Abstraction

- ซึ่งทำให้เราสามารถเปลี่ยนแปลงการทำงานภายใน โดยไม่กระทบกับ Interface

Abstraction

- คำว่า Abstraction ยังหมายถึง การนำสิ่งที่คล้ายกันมารวมกัน เพื่อลดความซ้ำซ้อน



Public and Non-Public Attribute

- จากหลักการของ Information Hiding และ Abstraction ทำให้การที่ยอมให้สามารถเข้าถึง Attribute ทั้งหมดของ Object จึงไม่ควรทำ
- ดังนั้นเราจึงต้องหาทางจำกัดการเข้าถึงข้อมูล

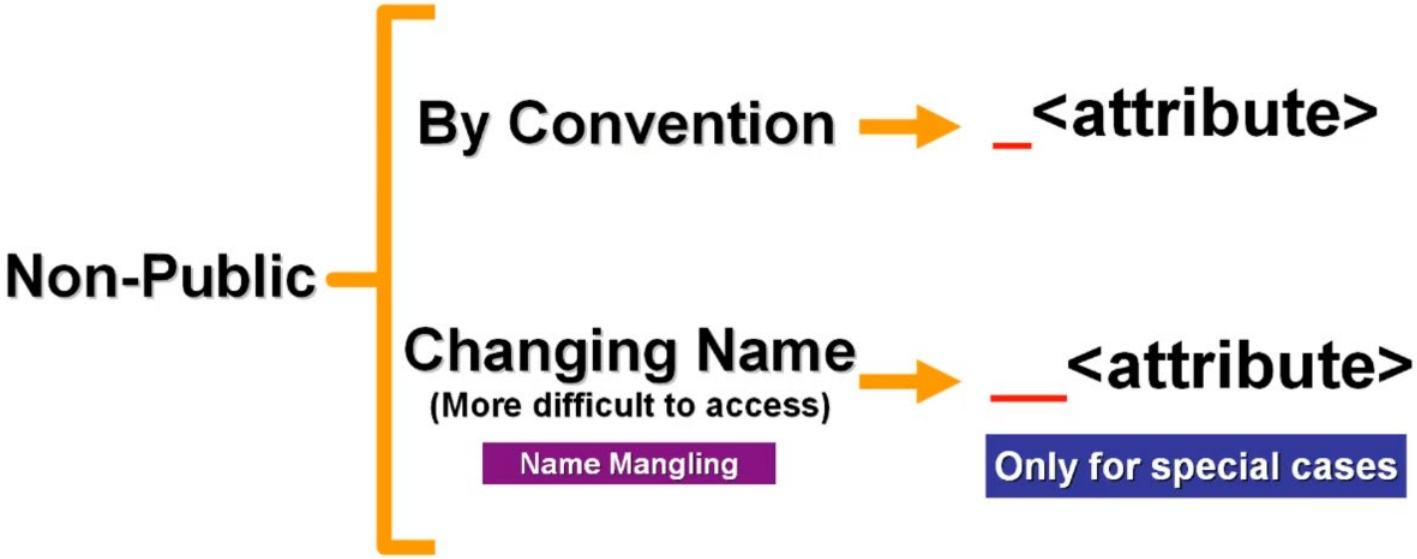
Public and Non-Public Attribute

- Public Attribute คือ Attribute ที่สามารถเข้าถึง หรือ แก้ไข จากภายนอก Object ได้

```python
class Car:

    def __init__(self, brand, model, year):
        self.brand = brand
        self.model = model
        self._year = year


my_car = Car("Porsche", "911 Carrera", 2020)

print(my_car._year)
my_car._year = 5600
print(my_car._year)
```

Public and Non-Public Attribute

- Non-Public Attribute คือ Attribute ที่ไม่สามารถเข้าถึงจากภายนอก Object
- แต่ Python ไม่ได้ทำแบบนั้น
- คำถาม คือ จะทำอย่างไร เพราะ Python ไม่มี Private เหมือนกับภาษา JAVA
- มี 2 วิธี ดังนี้



Public and Non-Public Attribute

- วิธีแรก คือ By Convention คือ ตัวแปรใดที่เป็น Non Public ก็ตั้งโดยใช้เครื่องหมาย “_” นำหน้า แล้วก็อย่าไปเขียนเพื่ออ้างถึงจากภายนอก

```python
class Car:

    def __init__(self, brand, model, year):
        self.brand = brand
        self.model = model
        self._year = year


my_car = Car("Porsche", "911 Carrera", 2020)

print(my_car.year) # Can't be accessed

print(my_car._year)
```

Public and Non-Public Attribute

- ตัวอย่างของ Non-Public Attribute

```python
class Movie:

    id_counter = 1

    def __init__(self, title, year, language, rating):
        self._id = Movie.id_counter
        self.title = title
        self.year = year
        self.language = language
        self.rating = rating

        Movie.id_counter += 1


my_movie = Movie("Pride and Prejudice", 2005, "English", 4.7)
your_movie = Movie("Sense and Sensibility", 1995, "English", 4.6)

print(my_movie.id)    # Throws an error for both instances.
print(your_movie.id)

print(my_movie._id)    # Can be accessed but it shouldn't be.
print(your_movie._id)
```

Public and Non-Public Attribute

- วิธีที่ 2 เรียกว่า Name Mangling คือ ตัวแปรใดที่เป็น Non Public ก็ตั้งโดยใช้เครื่องหมาย “__” นำหน้า ซึ่งเมื่อเรียกใช้นอก Object จะ Error

```python
class Backpack:

    def __init__(self):
        self.__items = [] # Two underscores


my_backpack = Backpack()

print(my_backpack.items) # Can't be accessed.

print(my_backpack._items) # Can't be accessed with this name.

print(my_backpack.__items) # Can't be accessed with this name.

print(my_backpack._Backpack__items) # Can be accessed but it shouldn't be.
                                    # It should only be used for special cases.
```

Activity

- จาก Class book จงทำให้ attribute ต่อไปนี้เป็น Non Public
  - num_pages
  - ISBN
  - publisher

```
class Book:

    def __init__(self, title, author, num_pages, ISBN, publisher):
        self.title = title
        self.author = author
        self.num_pages = num_pages
        self.ISBN = ISBN
        self.publisher = publisher
```

Setter and Getter

- จากที่ผ่านมา เราได้แยกระหว่าง Public กับ Non_Public
- ดังนั้นในการเข้าถึง Non_Public เราจะใช้ Setter และ Getter

Getters → **Get** the value of an attribute.

Setters → **Set** the value of an attribute.

Setter and Getter

- getter เป็น method สำหรับอ่านค่าจาก Object มักใช้คำว่า get และ “_” จากนั้นตามด้วยชื่อ Attribute

get_ + <attribute>

get_name

get_address

get_color

get_age

get_id

Setter and Getter

```python
class Movie:

    def __init__(self, title, rating):
        self._title = title
        self.rating = rating

    def get_title(self):
        return self._title


my_movie = Movie("The Godfather", 4.8)

print(my_movie.title) # Throws an error

print(my_movie.get_title())
print("My favorite movie is:", my_movie.get_title())
```

Setter and Getter

- setter เป็น method สำหรับกำหนดค่าให้กับ Attribute ใน Object มักใช้คำว่า set และ “_” จากนั้นตามด้วยชื่อ Attribute
- setter มีหน้าที่ validate ข้อมูลก่อนจะกำหนดค่าด้วย

set_ + <attribute>

set_name **set_address** **set_color**

set_age **set_id**

Setter and Getter

```python
class Dog:

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

    def get_name(self):
        return self._name

    def set_name(self, new_name):
        if isinstance(new_name, str) and new_name.isalpha():
            self._name = new_name
        else:
            print("Please enter a valid name.")

my_dog = Dog("Nora", 8)
print("My dog is:", my_dog.get_name())
my_dog.set_name("Norita")
print("Her new name is:", my_dog.get_name())
```

Setter and Getter

```python
class Backpack:

    def __init__(self):
        self._items = []

    def get_items(self):
        return self._items

    def set_items(self, new_items):
        if isinstance(new_items, list):
            self._items = new_items
        else:
            print("Please enter a valid list of items.")


my_backpack = Backpack()
print(my_backpack.get_items())

my_backpack.set_items("Hello, World!") # Invalid value
my_backpack.set_items(["Water Bottle", "Sleeping Bag", "First Aid Kit"])

print(my_backpack.get_items())
```

Setter and Getter

```python
class Circle:

    def __init__(self, radius):
        self._radius = radius

    def get_radius(self):
        return self._radius

    def set_radius(self, new_radius):
        if isinstance(new_radius, float) and new_radius > 0:
            self._radius = new_radius
        else:
            print("Please enter a valid value for the radius.")


my_circle = Circle(5.0)
print(my_circle.get_radius())

my_circle.set_radius(0) # This will not change the value.
print(my_circle.get_radius())

my_circle.set_radius("Hello, World!") # This will not change the value.
print(my_circle.get_radius())

my_circle.set_radius(10.5) # This will change the value.
print(my_circle.get_radius())
```

Properties in Python

- ที่ผ่านมาได้กล่าวถึง Python ที่ใช้ Getter และ Setter เพื่อให้เกิด Encapsulation และ Abstraction ตามแนวคิดของ OOP
- แต่ข้อเสียคือ แทนที่จะให้ความรู้สึกของการเข้าถึง attribute แบบเดิม กลับต้องทำผ่าน method ซึ่งทำให้โปรแกรมดูยุ่งยาก ไม่เหมือนกับการเข้าถึง attribute
- อย่างไรก็ตาม Python ได้ให้ฟังก์ชัน Property ไว้ เพื่อทำให้การเรียก getter และ setter เป็นไปโดยอัตโนมัติ

```
<property_name> = property(<getter>, <setter>)
```

Properties in Python

```python
class Dog:

    def __init__(self, age):
        self._age = age

    def get_age(self):
        return self._age

    def set_age(self, new_age):
        if isinstance(age, int) and 0 < age < 30:
            self._age = new_age
        else:
            print("Please enter a valid age.")

    age = property(get_age, set_age)


my_dog = Dog(8)
print(f"My dog is {my_dog.age} years old.")
print("One year later...")

my_dog.age += 1
print(f"My dog is now {my_dog.age} years old.")
```

Properties in Python

```python
class Dog:

    def __init__(self, age):
        self._age = age

    def get_age(self):
        print("Calling the Getter...")
        return self._age

    def set_age(self, new_age):
        print("Calling the Setter...")
        if isinstance(age, int) and 0 < age < 30:
            self._age = new_age
        else:
            print("Please enter a valid age.")

    age = property(get_age, set_age)


my_dog = Dog(8)
print(f"My dog is {my_dog.age} years old.")
print("One year later...")

my_dog.age += 1
print(f"Now my dog is {my_dog.age} years old.")
```

Properties in Python : Quiz

- จาก Class ต่อไปนี้ property value กำหนดถูกต้องหรือไม่

```
1   class Card:
2
3       def __init__(self, value):
4           self._value = value
5
6       def get_value(self):
7           return self._value
8
9       def set_value(self, value):
10          if 0 < value < 10:
11              self._value = value
12
13      value = property(set_value, get_value)
```

Properties in Python

- จะเห็นว่าการใช้งาน สามารถอ้างถึง Attribute **age** ได้คล้ายกับไม่มี getter และ setter
- แต่มีข้อดีมากกว่า เพราะสามารถ **validate** ข้อมูลได้ด้วย
- แต่ก็สร้างปัญหาใหม่ เพราะเท่ากับว่าสามารถอ้างถึง Attribute ได้ ถึง 2 วิธี คือ ใช้ **set_age(8)** ก็ได้ หรือ **my_dog.age = 8** ก็ได้
- วิธีแก้วิธีแรก คือ กำหนด _age ให้เป็น __age ก็จะทำให้การอ้างถึงเหลือแค่วิธีเดียว แต่ดูไม่ค่อยดีเท่าไร
- วิธีที่ 2 คือใช้ Decorator

First Class Function

- First Class Function เป็นสิ่งที่ใช้เรียก Programming Language ที่สามารถมอง Functions เป็นเหมือน Object โดยมีคุณสมบัติดังนี้
  - Function เป็น instance ของ Object
  - เราสามารถเก็บ function ไว้ใน variable.
  - เราสามารถ pass function เป็น parameter ไปยังอีก function ได้
  - เราสามารถ return function จาก function ได้
  - เราสามารถเก็บ function ไว้ใน data structures เช่น lists, dictionary, ...

First Class Function

- Function เป็น instance ของ Object และเก็บในตัวแปรได้
  - จากโปรแกรมตัวอย่าง ในส่วนแรกจะเป็นการเรียก Fn และคืนค่ามาในตัวแปร แต่ส่วนที่ 2 นั้น my_cube เป็น Instance ของ Object ของ Fn จึงสามารถใช้งานได้เช่นเดียวกับ Fn

```
1  def cube(x):
2      return x*x*x
3
4  res = cube(5)
5  print(res)
6
7  my_cube = cube #The my_cube is same as the cube method
8  res = my_cube(5)
9  print(res)
10
```

```
$python main.py
125
125
```

First Class Function

- pass function เป็น parameter ไปยังอีก function ได้
  - จากโปรแกรมจะเห็นว่าเราสามารถใช้ cube ซึ่งเป็น Fn เป็น parameter ของ my_map ได้

```
1  def cube(x):
2      return x*x*x
3
4  def my_map(method, argument_list):
5      result = list()
6      for item in argument_list:
7          result.append(method(item))
8      return result
9  my_list = my_map(cube, [1, 2, 3, 4, 5, 6, 7, 8]) #Pass the function as argument
10 print(my_list)
11
```

```
$python main.py
[1, 8, 27, 64, 125, 216, 343, 512]
```

First Class Function

- return function จาก function ได้
  - จะเห็นว่าฟังก์ชัน create_logger ส่งค่า Function log กลับคืนมา อยู่ในตัวแปร my_logger (ทำให้มันเป็น instance ของ object function) และเรียกใช้งานได้

```
1  def create_logger(message):
2      def log():
3          print('Log Message: ' + message)
4      return log #Return a function
5
6  my_logger = create_logger('Hello World')
7  my_logger()
8
```

```
$python main.py
Log Message: Hello World
```

Closures

- Closures เป็นอีกคุณสมบัติที่มีในภาษา Programming สมัยใหม่อย่าง Python และ Javascript และเป็นคุณสมบัติที่มาจาก first class function
- จะเห็นว่าเมื่อเรียก outer_func() จะมีการส่งค่า inner_func() กลับมาและทำงาน ซึ่งสามารถเข้าถึง message ได้ (กรณีนี้เรียกว่า free variable เพราะไม่อยู่ใน inner_func()

```
1  def outer_func():
2      message = 'Hi'
3
4      def inner_func():
5          print(message)
6
7      return inner_func()
8
9  outer_func()
10
```

```
$python main.py
Hi
```

Closures

- ลองดูอีกตัวอย่าง ตัวอย่างนี้เมื่อรัน จะไม่แสดงผลอะไร

```
1  def outer_func():
2      message = 'Hi'
3
4      def inner_func():
5          print(message)
6
7      return inner_func
8
9  myfunc = outer_func()
10
```

- ถ้าเราลอง print(my_func) จะแสดงว่าเป็น inner_func() แสดงว่าเป็นเพียง inner_func() แต่ถ้าสั่งทำงานพบว่าจะยังสามารถเข้าถึงตัวแปร message อยู่ ความสามารถนี้เรียกว่า Closures

```
$python main.py
<function inner_func at 0x7f7edbaa0a50>
```

Closures

- เอาความสามารถนี้ไปทำอะไรได้ ลองดูตัวอย่าง จะเห็นว่าเราสามารถสร้าง function ที่ทำงานต่างกันเล็กน้อยได้ จาก source code ชุดเดียวกัน

```
1  def outer_func(msg):
2      message = msg
3
4      def inner_func():
5          print(message)
6
7      return inner_func
8
9  hi_func = outer_func('Hi')
10 hello_func = outer_func('Hello')
11 hi_func()
12 hello_func()
```

```
$python main.py
Hi
Hello
```

Decorator

- ลองดูตัวอย่างต่อไปนี้ เมื่อทำงานจะแสดงผลอย่างไร

```
1  def decorator_function(original_function):
2
3      def wrapper_function():
4          return original_function()
5
6      return wrapper_function
7
8  def display():
9      print('display function')
10
11 decorated_display = decorator_function(display)
12
13 decorated_display()
14
```

Decorator

- เราสามารถเขียนแบบนี้แทนได้ ความหมาย คือ ให้นำฟังก์ชันต่อไปเป็น พารามิเตอร์ของ decorator_function ในชื่อเดิม (ไม่ใช่การเรียก display ตรงๆ)
- ผลที่เกิดขึ้น คือ เราสามารถแต่งเติม (decorate) ฟังก์ชันใดๆ ได้หมด

```python
def decorator_function(original_function):

    def wrapper_function():
        print('before original fucntion')
        return original_function()

    return wrapper_function

@decorator_function
def display():
    print('display function')

display()
```

$python main.py

before original fucntion
display function

@property Decorator

```python
class Backpack:

    def __init__(self):
        self._items = []

    @property
    def items(self):
        return self._items

    @items.setter
    def items(self, new_items):
        if isinstance(new_items, list):
            self._items = new_items
        else:
            print("Please enter a valid list of items.")


my_backpack = Backpack()
my_backpack.items = ["Water Bottle", "Sleeping Bag"]
print(my_backpack.items)
```

@property Decorator

- Getter

```
@property
def property_name(self):
    return self._property_name
```

- Setter

```
@property_name.setter
def property_name(self, new_value):
    self._property_name = new_value
```

@property Decorator

- นอกเหนือจาก getter กับ setter ยังมี deleter ด้วย (กรณีที่มีการลบ)

```
1  class Bus:
2
3          def __init__(self, color):
4                  self._color = color
5
6          @property
7          def color(self):
8                  return self._color
9
10         @color.setter
11         def color(self, new_color):
12                 self._color = new_color
13
14         @color.deleter
15         def color(self):
16                 del self._color
```

@property Decorator : Quiz

- เลือกข้อที่ถูกต้อง
  1. ในการกำหนด attribute ที่ read-only ควรกำหนด getter โดยใช้ @property
  2. ควรจะกำหนดทั้ง getter และ setter เสมอ ไม่ควรกำหนดอย่างใดอย่างหนึ่ง
  3. ไม่ควรใช้ @property ในภาษา python

@property Decorator : Quiz

- ถ้าจะสร้าง property สำหรับ self._code เราควรกำหนด getter ด้วย _____ และ setter ด้วย _____

  1. @property; @property.setter
  2. @code; @code.setter
  3. @property; @code.setter

@property Decorator : Quiz

- การกำหนด property สำหรับ value ถูกต้องหรือไม่

```python
class Card:

    def __init__(self, value):
        self._value = value

    @property
    def get_value(self):
        return self._value

    @get_value.setter
    def set_value(self, new_value):
        if 0 < new_value < 10:
            self._value = new_value
        else:
            print("Please enter a valid value")
```

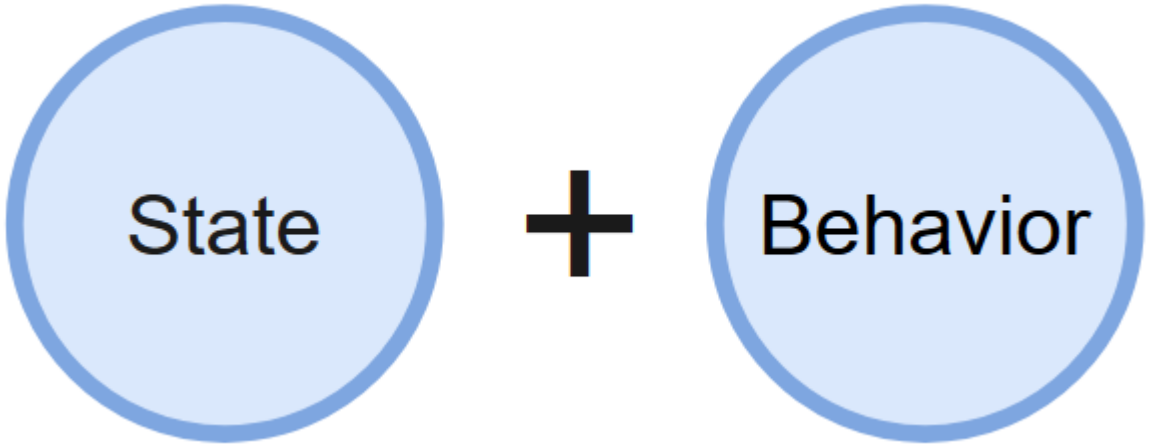
@property Decorator : Assignment

- จาก code ที่ให้ ให้แก้ไขให้ใช้ @ property อย่างถูกต้อง

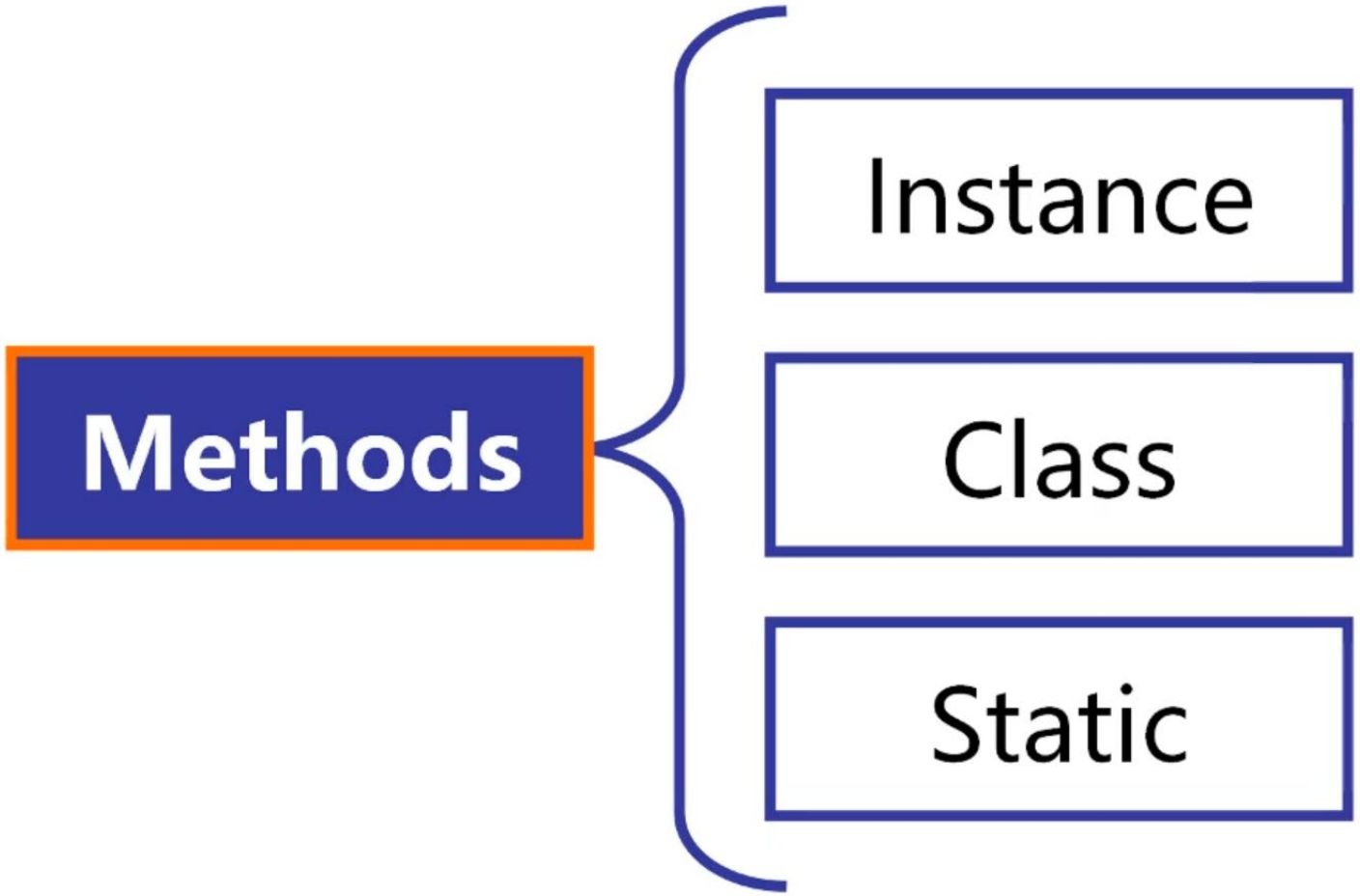
```
class BouncyBall:

    def __init__(self, price, size, brand):
        self.price = price
        self.size = size
        self.brand = brand
```

Methods

- หลังจากที่กล่าวถึง Attribute ก็มาถึงเรื่อง Methods เพราะ Object ประกอบด้วย state และ behavior ซึ่งส่วนที่ทำให้เกิด behavior คือ Methods



Methods



Methods

- Instance methods คือ methods ที่เป็นของ Object หนึ่ง โดยสามารถเข้าถึง attribute (state) ของ Object นั้น
- ดังนั้น methods ประเภทนี้จึงต้องมีคำว่า self เพื่อใช้ในการอ้างอิงถึง Object ที่เรียกใช้ method แม้จะไม่มี พารามิเตอร์ เลยก็ตาม

```
class MyClass:

    # Class Attributes

    # __init__()

    def method_name(self, param1, param2, ...):
        # Code
```

Methods

- ชื่อของ Method ควรเป็นคำกริยา เพื่อแสดงว่า Method นี้ “ทำ” อะไร
- ควรใช้ snake case (อักษรตัวเล็ก คั่นด้วย _) เพื่อให้อ่านง่าย
- ถ้าเป็น non public method ควรขึ้นต้นด้วย _

- **Build**
- **Show**
- **Shuffle**
- **Draw Card**
- **More…**

Methods

- ตัวอย่างของ Method

```python
class Circle:

    def __init__(self, radius):
        self.radius = radius

    # Printing the value
    def find_diameter(self):
        print(f"Diameter: {self.radius * 2}")
        # The value could be returned too with:
        # return self.radius * 2
```

Methods

- ตัวอย่างของ Method

```
class Backpack:

    def __init__(self):
        self._items = []

    @property
    def items(self):
        return self._items

    def add_item(self, item):
        if isinstance(item, str):
            self._items.append(item)
        else:
            print("Please provide a valid item.")

    def remove_item(self, item):
        if item in self._items:
            self._items.remove(item)
            return 1
        else:
            return 0

    def has_item(self, item):
        return item in self._items
```

Methods

- การเรียกใช้ Method จะคล้ายกับเรียก function แต่ระบุชื่อ Object ด้วย


```
my_list = [4, 5, 6, 7, 8]

my_list.sort()
print(my_list)
my_list.append(14)
print(my_list)
my_list.extend([1, 2, 3])
print(my_list)
```

Methods

- สามารถเรียก method ได้อีกแบบ (แต่ไม่นิยม)

```
<ClassName>.<method_name>(<instance>, <args>)
```

- ตัวอย่าง

```
1  class Bus:
2
3      def __init__(self, color):
4          self._color = color
5
6      def welcome_student(self, student_name):
7          print(f"Hello {student_name}, how are you today?")
```

```
1  bus = Bus("blue")
2  Bus.welcome_student(bus, "Johnathan")
```

Methods : Quiz

- เรียก bark method ข้อใดถูกต้อง กำหนด instance ชื่อ my_dog

```
1  class Dog:
2
3      def __init__(self, name, age):
4          self.name = name
5          self.age = age
6
7      def bark(self):
8          print("Bark... Bark!")
```

1. my_dog.bark()
2. bark(my_dog)
3. Dog.bark(my_dog)
4. 1 กับ 3

Methods : exercise

- จาก Class ต่อไปนี้ ให้สร้าง instance ชื่อ my_clinic และเพิ่มชื่อคนไข้

```python
class Clinic(object):

    max_patients = 3

    def __init__(self, name):
        self.name = name
        self.patients = []
        self.waiting_list = []

    def add_patient(self, patient):
        if len(self.patients) >= Clinic.max_patients:
            self.waiting_list.append(patient)
        else:
            self.patients.append(patient)
```

Methods

- ใน Class แต่ละ method สามารถเรียกใช้ระหว่างกันได้ ตามตัวอย่าง

```python
class Backpack:

    def __init__(self):
        self._items = []

    @property
    def items(self):
        return self._items

    def add_multiple_items(self, items):
        for item in items:
            self.add_item(item)

    def add_item(self, item):
        if isinstance(item, str):
            self._items.append(item)
        else:
            print("Please provide a valid item.")
```

Methods : chaining

- ลองดู Class ต่อไปนี้ (ดูที่ add_topping)

```
1  class Pizza:
2
3      def __init__(self):
4          self.toppings = []
5
6      def add_topping(self, topping):
7          self.toppings.append(topping.lower())
8          return self
9
10     def display_toppings(self):
11         print("This Pizza has:")
12         for topping in self.toppings:
13             print(topping.capitalize())
```

Methods : chaining

- จะเห็นคำสั่ง return self ซึ่งเป็นการ return instance ที่เรียกใช้ method
- ทำให้เราสามารถทำ method chaining ได้ ตามตัวอย่าง

```
1  pizza.add_topping("mushrooms") \
2      .add_topping("olives") \
3      .add_topping("chicken") \
4      .display_toppings()
```

Methods : Quiz

- จงเลือก ข้อที่ถูกในการกำหนด method

○
```
def <method_name>(self, <parameters>):
    # code
```

○
```
<method_name>(self, <parameters>):
    # code
```

○
```
def <method_name>(<parameters>):
    # code
```

Methods : assignment

- มีโปรแกรมเดิมอยู่ เป็นโปรแกรมสำหรับโรงเรียนสอนดนตรี โดยเป็นโปรแกรมบันทึกการเข้าเรียน
- งานที่ให้ทำ คือ ให้เพิ่ม 3 method
  - Method print_students_data แสดงชื่อของนักเรียนแต่ละคน อายุ และ class ที่เรียน ในรูปแบบของ dictionary ตามตัวอย่าง

```
"Student: Gino who is 15 years old and is taking ['Piano', 'Guitar']"
"Student: Talina who is 28 years old and is taking ['Cello']"
"Student: Eric who is 12 years old and is taking ['Singing']"
```

Method __str__

- เป็น method สำหรับจัดรูปแบบกรณีที่ print object

```
class MyClass:
    x = 0
    y = ""

    def __init__(self, anyNumber, anyString):
        self.x = anyNumber
        self.y = anyString

    def __str__(self):
        return 'MyClass(x=' + str(self.x) + ' ,y=' + self.y + ')'


myObject = MyClass(12345, "Hello")

print(myObject.__str__())
print(myObject)
print(str(myObject))
```

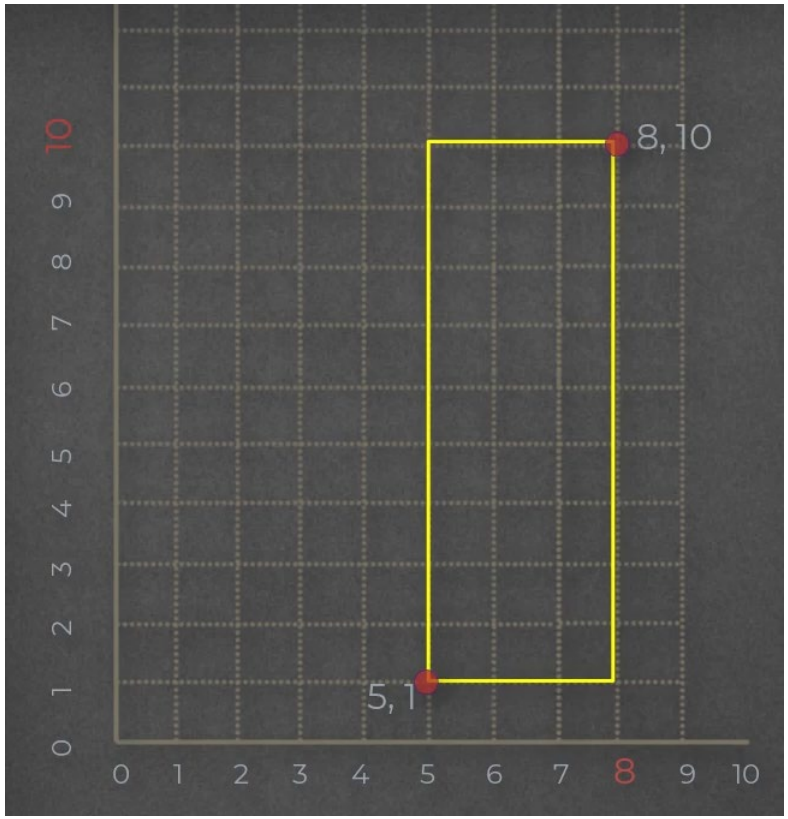
Methods : assignment

- งานที่ให้ทำ คือ ให้เพิ่ม 3 method
  - Method print_student แสดงชื่อของนักเรียน อายุ และ class ที่เรียน ในรูปแบบของ dictionary ตามเช่นกัน แต่รับ argument เป็นชื่อของนักเรียนและแสดงเฉพาะนักเรียนที่ระบุเท่านั้น
  - Method add_student ทำหน้าที่เพิ่มข้อมูลเข้าไปใน dictionary

```
class MusicSchool:

    students = {"Gino":   [15, "653-235-345", ["Piano", "Guitar"]],
                "Talina": [28, "555-765-452", ["Chello"]],
                "Eric":   [12, "583-356-223", ["Singing"]]}

    def __init__(self, working_hours, revenue):
        self.working_hours = working_hours
        self.revenue = revenue
```

Object Oriented Workshop #1

- เกม Geometry เกมจะกำหนดตำแหน่ง (coordinate) มา 2 ตำแหน่งตามรูป และให้ผู้ใช้ป้อนตำแหน่ง เกมจะบอกว่า อยู่ในกรอบสี่เหลี่ยมหรือไม่



Object Oriented Example #1

- งานแรกที่ต้องทำ คือ มี Class หรือ Object อะไรบ้าง ให้ นศ. ลองคิด
  - Point เป็น Class ของจุด
  - Rectangle เป็น Class ของสี่เหลี่ยม

Assignment

- #TODO : ให้แก้โดยเอา method falls_in_rectangle ไปไว้ใน Class Point แทน

- #TODO : ให้เขียน method คำนวณพื้นที่สี่เหลี่ยม และ เพิ่มให้ User บอกพื้นที่

For your attention