Lecture 15 Objects

16 ตุลาคม 2557 9:24

เราได้ศึกษาเกี่ยวกับการเก็บสิ่งของรวมกันไว้ใน list และได้เรียน functions การรวมกลุ่มของโค้ดทำงานด้วยกันเพื่อง่ายแก่ใช้งาน แนวคิดของอ็อปเจ็ค (object) ได้พัฒนาหลักการต่อไปนี้มากไปกว่านั้น โดยได้นำหลักการการรวมกันหรือการอยู่ด้วยกันของข้อมูล(data) และฟังก์ชัน (functions) ซึ่งเป็นแนวคิดที่ดีของการเขียนโปรแกรมเป็นอย่างมาก ถ้าเราสังเกตตัวภาษา Python เองก็อาศัยหลักการนี้เป็นจำนวนมากเช่น list เราสามารถเก็บข้อมูลโดยใช้ฟังก์ชันที่อยู่กับตัวมันเอง เราเรียกหลักการนี้ว่า การจัดการเชิงวัตุ (object-oriented) เราจะได้ศึกษาการสร้างอ็อปเจ็คด้วยตัวเองในบทนี้

วัตถุคืออะไร Object in the real world

ถ้าเราจะอธิบายวัตถุคืออะไร วัตถุหรืออ็อปเจ็ค จะต้องมีองค์ประกอบหรือคุณสมบัติ สถานะ เราเรียกว่า แอตทริบิวต์ (attributes) เช่นถ้าลูกบอล จะประกอบด้วย ขนาด สี เป็นองค์ประกอบเป็นต้น นอกจากส่วนที่เป็นคุณสมบัติแล้ว วัตถุสามารถมีสิ่งที่เรียกว่าพฤติกรรมหรือการกระทำ (actions) เป็นสิ่งที่วัตถุสามารถที่จะกระทำหรือถูกกระทำ อาทิเช่น ถ้าเป็นลูกบอล เราสามารถที่จะขว้าง โยน หรือเก็บได้ ทำให้พองได้ ดังนั้นสรุปได้ว่า วัตถุ ประกอบด้วย

- สิ่งที่วัตถุสามารถกระทำได้หรือถูกกระทำ
- วัตถุมีองค์ประกอบและคุณสมบัติ

Object in Python

สิ่งที่เป็นคุณลักษณะหรือคุณสมบัติของอ็อปเจ็คเราเรียกว่าแอททริบิวต์ attributes และสิ่งที่เป็นการกระทำหรือพฤติกรรมเราเรียกว่า เมธอด (method) เป็นชื่อหนึ่งของ เป็นเพียงชื่อหนึ่งของฟังก์ชันที่อยู่ในอ็อปเจ็ค เราสามารถแสดงการเขียน Python ได้ดังลักษณะนี้ ลักษณะของแอททริบิวต์

```
ball.color
ball.size
ball.weight
```

ลักษณะของ เมธอด

```
ball.kick()
ball.throw()
ball.inflate()
```

แอททริบิวต์ What are attributes?

แอททริบิวต์คือคุณสมบัติของตัวอ็อปเจ็คเองประกอบด้วยอะไรบ้าง จะเป็นข้อมูลที่อธิบายลักษณะของอ็อปเจ็ค เราสามารถที่จะพิมพ์

```
print(ball.color)
```

เราสามารถที่จะให้ค่า

```
ball.color = 'green'
```

เราสามารถที่จะอ่านค่ามาเก็บไว้เอง

```
myColor = ball.color
```

หรือจะให้ค่าและอ่านค่าจากอ็อปเจ็คอื่นได้

```
myBall.color = yourBall.color
```

เมธอด What are method?

เมธอดคือฟังก์ชันที่อยู่ในวัตถุ เมื่อเป็นฟังก์ชันหน้าที่ของเมธอดคือ รวมกลุ่มของโค้ดเพื่อที่จะเรียกใช้งานได้ง่าย เราสามารถที่จะส่งค่าให้เมธอดและรับค่ากลับจากเมธอดได้

สัญลักษณ์ dot

เนื่องจากอ็อปเจ็คประกอบด้วย attributes และ methods การที่เราจะเข้าถึงองค์ประกอบทั้งสองอย่างนี้ เราจะต้องอ้างชื่อของอ็อปเจ็คก่อนแล้วตามด้วยจุด (dot) แล้วตามด้วย attributes หรือ methods ที่ต้องการ

```
Object.attribute
Object.method()
```

การสร้างอ็อปเจ็ค Creating Objects

การจะสร้างอ็อปเจ็คได้ในภาษา Python ประกอบด้วยสองขั้นตอนด้วยกัน

1. สร้างแบบของอ็อปเจ็คว่าประกอบด้วย attributes และ method อะไร เราเรียกว่า คลาส class หลังจากที่เรามีแบบแล้ว เราสามารถที่จะสร้างอ็อบเจ็คได้หลายอัน เหมือนกับถ้าเรามีแบบบ้านเราสามารถสร้างบ้านที่มีลักษณะนี้ได้หลายหลัง แต่อาจจะปรับแต่ง สีบ้านตามผู้อยู่ได้ แต่โครงบ้านเกิดจากแบบเดียวกัน แบบบ้านเราเรียกว่าคลาส ส่วนบ้านแต่ละหลังเราเรียกว่า อ็อปเจ็ค
2. เมื่อเรามีแบบ เราสามารถสร้าง อ็อปเจ็คได้ หรือสร้างบ้านแต่ละหลัง เราจะเรียกบ้านแต่ละหลังที่ออกจากแบบบ้านเดียวกันว่า อินสแตน (instance) ของคลาส

ตัวอย่างต่อไปเป็นการสร้าง class หรือ แบบของลูกบอล

```
class Ball:

    def bounce(self):
        if self.direction == "down":
            self.direction = "up"
```

เราสร้างคลาสชื่อ Ball ทีประกอบด้วย method คือ bounce() แปลว่าเด้ง ตัวอย่างนี้ยังไม่แสดงแอททริบิวต์ที่ชัดเจนนัก เมื่อเราสร้างคลาส เรามีแบบของบอลแล้ว เราสามารถสร้าง อ็อปเจ็คเองได้ เช่น

```
myBall = Ball()
```

แอททริบิวต์เป็นตัวแปรที่เกิดจากอ็อปเจ็คชั้นหนึ่ง เมื่อเรามีอ็อปเจ็ค myBall เราสามารถสร้างแอททริบิวต์ใดๆ ของอ็อปเจ็คนั้นได้ดังนี้

```
myBall.direction = "down"
myBall.color = "green"
myBall.size = "small"
```

และเราสามารถเรียกใช้ method ได้ดังนี้

```
myBall.bounce()
```

เมื่อแสดงโค้ดทั้งหมดสมบูรณ์ได้ดังนี้

```
class Ball:

    def bounce(self):
        if self.direction == "down":
            self.direction = "up"



myBall = Ball()
myBall.direction = "down"
myBall.color = "green"
myBall.size = "small"

print("I'm just create a ball.")
print("My ball is", myBall.size)
print("My ball is", myBall.color)
print("My ball's direction is", myBall.direction)
print("Now I'm going to bounce the ball")
print()
myBall.bounce()
print("Now, the ball's direction is", myBall.direction)
```

เมื่อแสดงผลลัพท์ได้ดังนี้

```
I'm just create a ball.
My ball is small
My ball is green
My ball's direction is down
Now I'm going to bounce the ball

Now, the ball's direction is up
```

การสร้างแบบให้มีค่าเริ่มต้น (Initializing and object)

จากตัวอย่างการสร้างอ็อปเจ็คที่แล้ว อ็อปเจ็คเราจำเป็นต้องให้ค่า attribute หลังจากสร้าง เรามีวิธีที่ดีกว่าคือ ตอนสร้างเราต้องการให้ค่าไปพร้อมกัน ทำให้อ็อปเจ็คพร้อมใช้งานได้ทันที หรือพร้อมพิมพ์ค่าได้เลย เราเรียกว่า การสร้างแบบเริ่มค่า (initializing) ซึ่งการสร้างเราจะสร้างเมธอดที่มีชื่อว่า __init__() คำว่า init ย่อมาจาก initialize จากตัวอย่างดังนี้

```
class Ball:
    def __init__(self, color, size, direction):
        self.color = color
        self.size = size
        self.direction = direction

    def bounce(self):
        if self.direction == "down":
            self.direction = "up"


myBall = Ball("green", "small", "down")

print("I'm just create a ball.")
print("My ball is", myBall.size)
print("My ball is", myBall.color)
print("My ball's direction is", myBall.direction)
print("Now I'm going to bounce the ball")
print()
myBall.bounce()
print("Now, the ball's direction is", myBall.direction)
```

```
def __init__(self, color, size, direction):
    self.color = color
    self.size = size
    self.direction = direction
```

ผลลัพท์ที่ได้ยังคงเหมือนเดิม แต่เราให้ค่าของ attributes เริ่มต้นของลูกบอลไปแล้ว

การสร้าง method __str__() คืนค่าข้อความ

ถ้าเราสั่งพิมพ์อ็อปเจ็คของเราตอนนี้ จะได้ข้อความที่ไม่สื่อความหมายใดๆ เกี่ยวกับอ็อปเจ็ค เช่น

```
>>> print(myBall)
<__main__.Ball object at 0x0000000002BE1358>
>>>
```

เราสามารถเปลี่ยนค่าการพิมพ์เพื่อให้ สื่อความหมายของอ็อปเจ็คได้ดียิ่งขึ้น โดยการสร้างชื่อของเมธอดที่มีชื่อว่า __str__() เป็นชื่อเมธอดพิเศษ เช่นเดียวกับที่เราเคยใช้ __ini__() ไปแล้ว แต่เมธอดนี้เราจำเป็นจะต้องคืนค่าข้อความที่เราต้องการแสดงออกไป ดังตัวอย่างนี้

```
class Ball:
    def __init__(self, color, size, direction):
        self.color = color
        self.size = size
        self.direction = direction

    def __str__(self):
        msg = "Hi, I'm a "+ self.size + " " + self.color + " " + self.direction
        return msg

myBall = Ball("green", "small", "down")
print(myBall)
```

ผลลัพธ์ที่ได้

```
>>>
Hi, I'm a small green down
```

เราสร้างเมธอดที่ใช้งานเฉพาะโดยใช้ อันเดอร์สกอร์สองอันแต่ละดานซ้ายขวา

Self คืออะไร

จากตัวอย่างหลายๆ ตัวอย่างด้านบน เราจะเห็นทั้งใน method และเป็นพารามิเตอร์ของ method เช่น

```
def __init__(self, color, size, direction):
    self.color = color
    self.size = size
    self.direction = direction
```

หรือ

```
def bounce(self):
```

อะไรคือความหมายของ self ถ้าเรายังจำได้ คลาสคือแบบบ้าน ที่เราสามารถสร้างบ้านได้หลายหลัง เช่นเดียวกันกับลูกบอลที่สามารถสร้างได้มากกว่าหนึ่ง

```
myBall = Ball("red", "small", "down")
yourBall = Ball("green", "medium", "up")
```

และตอนลูกบอลเด้งเราจะเรียกใช้

```
yourBall.bounce()
```

จากโครงสร้างของคลาสที่เป็นเพื่อให้แต่ละอ็อปเจ็ค เช่น myBall และ yourBall ใช้งาน ถ้าเราสั่ง bounce() ลูกใดลูกหนึ่ง เช่น สั่ง myBall อย่างเดียว แบบของลูกบอลที่เป็นแนวทางทั่วไป จำเป็นจะต้องรู้ว่าตัวไหนเป็นคนเรียกใช้ ซึ่งตัวคำว่า self นี้ เป็นตัวช่วยอ้างอิงอ็อปเจ็คที่ใช้งาน เราเรียกว่า intance reference
ดังนั้นเมื่อเราสั่งเกตุ เวลาเราเรียกใช้ bounce() เราไม่ส่งค่าใดๆ ไป แต่ตอนเราประกาศจะมี self อยู่ด้านในดังนี้ bounce(self) เพื่อรับค่า instance ที่เราส่งเข้ามา ถ้าพิจารณาจากโครงสร้างของคลาสแล้ว เราคิดว่าโค้ดควรเขียนในลักษณะนี้

```
Ball.bounce(yourBall)
```

แต่ทางที่ง่ายกว่าคือการที่เราจะเขียนแบบนี้

```
yourBall.bounce()
```

ตัวอย่างการสร้าง HotDog

เราจะเขียนโปรแกรมเกี่ยวกับ Hotdog อีกครั้ง โดย HotDog คราวนี้จะประกอบด้วยขนมปังทุกอัน (Bun) เรามาพิจารณา Attribute เกี่ยวกับ Hotdog ตั้งแต่การอบ

- ระดับเวลาการอบ cookedLevel ถ้ามีค่าดังนี้
  - 0-3 ยังดิบอยู่ (Raw)
  - 4-5 สุกปานกลาง (Medium)
  - 6-8 สุกพอดี (Well-done)
  - มากกว่า 8 ไหม้ (Charcoal)
- ข้อความแสดงระดับ cookedString
- ส่วนประกอบ condiments เช่น ketchup , mustard ซึ่งเป็น list ที่เก็บได้หลายค่า

Method มีดังนี้

- cook() ปรุงอาหารโดยระบุเวลา
- addCondiment() การใส่ส่วนประกอบ
- __init__() การสร้างอ็อปเจ็คและ instance
- __str__() ให้พิมพ์อ็อปเจ็คออกมาสื่อความหมายดี

เริ่มโดยการสร้าง class และเมธอด __init__()

```
class HotDog:
    def __init__(self):
        self.cookedLevel = 0
        self.cookedString = "Raw"
        self.condiments = []
```

ทดสอบการสร้างอ็อปเจ็คดังนี้

```
myDog = HotDog()
print("Cooked Level:",myDog.cookedLevel)
print("Cooked String:", myDog.cookedString)
print("Condiments:",myDog.condiments)
```

ทดสอบการแสดงผล

```
>>>
Cooked Level: 0
Cooked String: Raw
Condiments: []
```

จากนั้นเพิ่มโค้ดการปรุง HotDog ด้วย cook()

```
def cook(self, time):
    self.cookedLevel = self.cookedLevel + time
    if self.cookedLevel > 8:
        self.cookedString = "Charcoal"
    elif self.cookedLevel > 5:
        self.cookedString = "Well-done"
    elif self.cookedLevel > 3:
        self.cookedString = "Medium"
    else:
        self.cookedString = "Raw"
```

เมธอดนี้จะรับเวลา time เข้ามา และ พิจารณาว่าการปรุงอยู่ระดับไหน แล้วให้ข้อความอธิบายการระดับการปรุง
เราจะทดสอบการปรุงตัวอย่างเช่น ส่งเวลาเท่ากับ 4 เข้าไป

```
myDog.cook(4)
```

เมื่อรวมโค้ดทดสอบจะมีลักษณะนี้

```
class HotDog:
    def __init__(self):
        self.cookedLevel = 0
        self.cookedString = "Raw"
        self.condiments = []

    def cook(self, time):
        self.cookedLevel = self.cookedLevel + time
        if self.cookedLevel > 8:
            self.cookedString = "Charcoal"
        elif self.cookedLevel > 5:
            self.cookedString = "Well-done"
        elif self.cookedLevel > 3:
            self.cookedString = "Medium"
        else:
            self.cookedString = "Raw"



myDog = HotDog()
print("Cooked Level:",myDog.cookedLevel)
print("Cooked String:", myDog.cookedString)
print("Condiments:",myDog.condiments)

myDog.cook(4)
print("Now I'm going to cook the hot dog.")
print("Cooked Level:",myDog.cookedLevel)
print("Cooked String:", myDog.cookedString)
```

จากการรันได้ดังนี้

```
>>>
Cooked Level: 0
Cooked String: Raw
Condiments: []
Now I'm going to cook the hot dog.
Cooked Level: 4
Cooked String: Medium
```

เราเพิ่มส่วนประกอบด้วยการเพิ่มคำสั่ง addCondiment() ลงไปคลาส HotDog ดังนี้

```
def addCondiment(self, condiment):
    self.condiments.append(condiment)
```

ซึ่งในเมธอดนี้จะเรียก attribute ของส่วนประกอบทั้งหมด condiments ซึ่งเป็น list แล้วเรียกคำสั่งเพิ่ม append ส่วนประกอบที่ส่งเข้ามา condiment
และแก้ไขโค้ดโดยการเพิ่มส่วนทดสอบการเพิ่มส่วนประกอบดังนี้

```
myDog = HotDog()
myDog.addCondiment("Ketchup")
myDog.addCondiment("Mastard")
myDog.addCondiment("Onion")

print("Cooked Level:",myDog.cookedLevel)
print("Cooked String:", myDog.cookedString)
print("Condiments:",myDog.condiments)
```

ถ้าเรารันดูผลส่วนนี้จะได้

```
>>>
Cooked Level: 0
Cooked String: Raw
Condiments: ['Ketchup', 'Mastard', 'Onion']
```

เปลี่ยนให้คลาส HotDog สามารถพิมพ์ได้ โดยการเพิ่มคำสั่ง__str__()

```
def __str__(self):
    msg = "hot gog "
    if len(self.condiments) > 0:
        msg = msg + " with "
    for condiment in self.condiments:
        msg = msg + condiment + ", "
    msg = msg.strip(", ")
    msg = self.cookedString + " "+msg + "."
    return msg
```

และลองทดสอบด้วยการพิมพ์อ็อบเจ็ค

```
myDog = HotDog()
myDog.addCondiment("ketchup")
myDog.addCondiment("mastard")
myDog.addCondiment("onion")
print(myDog)
```

จะได้ผลลัพท์ดังนี้

```
>>>
Raw hot gog  with ketchup, mastard, onion.
```

เมื่อนำส่วนย่อยก่อนหน้านี้มารวมเป็นโปรแกรมใหม่ได้ดังนี้

```
class HotDog:
    def __init__(self):
        self.cookedLevel = 0
        self.cookedString = "Raw"
        self.condiments = []
    def cook(self, time):
        self.cookedLevel = self.cookedLevel + time
        if self.cookedLevel > 8:
            self.cookedString = "Charcoal"
        elif self.cookedLevel > 5:
            self.cookedString = "Well-done"
        elif self.cookedLevel > 3:
            self.cookedString = "Medium"
        else:
            self.cookedString = "Raw"
    def addCondiment(self, condiment):
        self.condiments.append(condiment)
    def __str__(self):
        msg = "hot gog "
        if len(self.condiments) > 0:
            msg = msg + " with "
        for condiment in self.condiments:
            msg = msg + condiment + ", "
        msg = msg.strip(", ")
        msg = self.cookedString + " "+msg + "."
        return msg


myDog = HotDog()
print(myDog)
print()
print("Cooking hot dog for 4 minutes...")
myDog.cook(4)
print(myDog)
print()
print("Cooking hot dog for 3 more minutes...")
myDog.cook(3)
print(myDog)
print()
print("What happens if I cook it for 10 more minutes?")
myDog.cook(10)
print(myDog)
print()
print("Now, I'm going to add some stuff on my hot dog")
myDog.addCondiment("ketchup")
myDog.addCondiment("mustard")
print(myDog)
```

เมื่อรันโปรแกรมจะได้ผลดังนี้

```
>>>
Raw hot gog.

Cooking hot dog for 4 minutes...
Medium hot gog.

Cooking hot dog for 3 more minutes...
Well-done hot gog.

What happens if I cook it for 10 more minutes?
Charcoal hot gog.

Now, I'm going to add some stuff on my hot dog
Charcoal hot gog  with ketchup, mustard.
```

การห่อหุ้ม Encapsulation , การซ้อนข้อมูล Hiding the data

จากโค้ด

```
myDog.cook(4)
```

ด้านในนี้จะมีการเปลี่ยนค่า cookedLevel เราทราบว่า เราสามารถที่จะให้ค่าโดยตรงเช่นกัน ดังนี้

```
myDog.cookedLevel = 4
```

แต่ทำไมเราไม่ทำอย่างนี้โดยตรง นี้คือหลักการห่อหุ้มencapsulation เราจะให้ค่าผ่าน method ไม่ใช่โดยตรงกับ attribute ทำไมเราไม่เปลี่ยนโดยตรงทั้งที่ยังใช้ได้ เนื่องจากถ้าเราเปลี่ยน cookedLevel โดยตรงสิ่งที่ตามมาคือ cookedString ไม่เปลี่ยนตาม แต่ถ้าเราใช้ cook(4) cookString จะเปลี่ยนตาม เราจะเห็นปัญหาของการให้ค่าโดยตรง ดังนั้นเราจำเป็นจะต้องจำกัดขอบเขตการใช้ของผู้ใช้ ให้ใช้ในรูปแบบที่เราต้องการ เราจะไม่ให้ค่าโดยตรง เราจะซ่อน attribute นี้คือแนวคิดของ การซ่อนข้อมูล (data hiding)

การสืบทอด Inheritance

การสืบทอดคือการนำของเก่าจากคลาสเดิม เพื่อนำมาใช้ใหม่ หรือขยายความสามารถของคลาสเดิม จงดูตัวอย่าง โค้ดคลาส Dog กับ Cat ดังนี้

```
class Dog:
    def __init__(self, name, color):
        self.name = name
        self.color = color
    def eat(self):
        print("I'm eating a chicken.")
    def sleep(self):
        print("I'm sleeping.")
    def play(self):
        print("I'm playing")
    def __str__(self):
        msg = "My name is "+self.name
        return msg
class Cat:
    def __init__(self, name, color):
        self.name = name
        self.color = color
    def eat(self):
        print("I'm eating a fish.")
    def sleep(self):
        print("I'm sleeping.")
    def play(self):
        print("I'm playing")
    def __str__(self):
        msg = "My name is "+self.name
        return msg
```

```
d = Dog("Fido", "red")
print(d)
d.eat()
d.sleep()
d.play()

c = Cat("Mini", "blak")
print(c)
c.eat()
c.sleep()
c.play()
```

ผลจากการรันดังนี้

```
My name is Fido
I'm eating a chicken.
I'm sleeping.
I'm playing
My name is Mini
I'm eating a fish.
I'm sleeping.
I'm playing
```

เราจะเห็นได้ว่า จะมีส่วนโค้ดที่เหมือนกัน ของ Dog กับ Cat ร่วมกันอยู่ คือสามารถที่จะกิน เล่น นอน ได้เหมือนกัน มีชื่อ มีสีตัวยกัน และเป็นสัตว์เลี้ยง (pet)

```
class Pet:
    def __init__(self, name, color):
        self.name = name
        self.color = color
    def eat(self):
        print("I'm eating a someting.")
    def sleep(self):
        print("I'm sleeping.")
    def play(self):
        print("I'm playing")
    def __str__(self):
        msg = "My name is "+self.name
        return msg
```

เมื่อเรามี Pet ซึ่งเป็นคลาสต้นแบบ หรือที่นิยมเรียก คลาสแม่ (superclass) เราสามารถสืบทอดและนำกลับมาใช้ใหม่ได้

```
class Dog(Pet):
    def __init__(self, name, color):
        Pet.__init__(self, name, color)

class Cat(Pet):
    def __init__(self, name, color):
        Pet.__init__(self, name, color)
```

เมื่อเราใช้ Dog หรือ Cat ที่สืบทอด เราสามารถที่จะใช้คุณสมบัติหรือเมธอดที่มีใน Pet ได้ ทำให้เราเรียกใช้ได้เหมือนเดิมดังนี้

```
d = Dog("Fido", "red")
print(d)
d.eat()
d.sleep()
d.play()

c = Cat("Mini", "blak")
print(c)
c.eat()
c.sleep()
c.play()
```

แต่ผลลัพท์ ตอนนี้การกิน ยังเหมือนกัน

```
>>>
My name is Fido
I'm eating a someting.
I'm sleeping.
I'm playing
My name is Mini
I'm eating a someting.
I'm sleeping.
I'm playing
```

Polymorphism เมธอดเดียวกันแต่ให้ผลลัพท์ที่ต่างกัน

จากผลลัพท์เดิมการกินของแมวกับสุนัขต่างกัน แต่เมื่อเรามาสืบทอด เราจะใช้การกินที่มาจาก Pet ดังนั้นเราสามารถที่จะเขียนทับ (override) เพื่อให้มีพฤติกรรมที่ต่างกัน ตัวอย่างต่อไปนี้เป็นการสืบทอดมาและเขียนทับเมธอดใหม่ดังนี้

```
class Dog(Pet):
    def __init__(self, name, color):
        Pet.__init__(self, name, color)
    def eat(self):
        print("I'm eating a chicken.")
    def __str__(self):
        msg = "I'm a dog. "+ Pet.__str__(self)
        return msg

class Cat(Pet):
    def __init__(self, name, color):
        Pet.__init__(self, name, color)
    def eat(self):
        print("I'm eating a fish.")
    def __str__(self):
        msg = "I'm a cat. "+ Pet.__str__(self)
        return msg
```

แล้วทำการรันโค้ดเดิมจะได้ผลดังนี้

```
>>>
I'm a dog. My name is Fido
I'm eating a chicken.
I'm sleeping.
I'm playing
I'm a cat. My name is Mini
I'm eating a fish.
I'm sleeping.
I'm playing
```

ทบทวน

- Object คือ
- Attribute และ methods
- Class คือ แบบการใช้สร้าง object
- การสร้างอ็อปเจ็คหรือ ที่เรียกว่า instance of class
- Method __init__() และ __str__()
- Encapsulation
- Polymorphism
- Inheritance

ทดสอบความรู้

1. คำอะไรที่ใช้ในการสร้างชนิดใหม่
2. Attribute คืออะไร
3. Method คืออะไร
4. ข้อแตกต่างระหว่าง class และ instance
5. คำที่นิยมใช้อ้างอิง instance (instance reference) ที่อยู่ใน class คืออะไร
6. Encapsulation คืออะไร ยกตัวอย่างโค้ดประกอบ
7. Polymorphism คืออะไร ยกตัวอย่างโค้ดประกอบ
8. Inheritance คืออะไร ยกตัวอย่างโค้ดประกอบ

จงเขียนโปรแกรมต่อไปนี้

1. สร้างคลาสบัญชีธนาคาร BankAccount ที่ attributes ดังนี้
    a. ชื่อ (name ชนิด string)

b. เลขที่บัญชี (account number เป็น string หรือ integer)
c. ยอดเงินคงเหลือ (balance เป็นชนิด float)

ให้มีเมธอด ดังนี้

- printBalance() พิมพ์มียอดเงินคงเหลือ
- deposite(money) ฝากเงิน
- Withdrawl(money) ถอนเงิน

ตัวอย่างโค้ดการเรียกใช้

```
a1 = BankAccount("Sarayut", "0001", 1000)
a2 = BankAccount("Bank", "0002", 500)
a1.printBalance()
a2.printBalance()
print()
print(a1.name, "deposits 100")
print(a2.name, "withdrawals 200")
a1.deposit(100)
a2.withdrawal(200)
a1.printBalance()
a2.printBalance()
```

ผลลัพท์ที่ได้

```
>>>
Name: Sarayut Account Number: 0001 Balance: 1000
Name: Bank Account Number: 0002 Balance: 500

Sarayut deposits 100
Bank withdrawals 200
Name: Sarayut Account Number: 0001 Balance: 1100
Name: Bank Account Number: 0002 Balance: 300
```

2. สร้างคลาส InterestAccount ที่สามารถได้ดอกเบี้ย ซึ่งเป็นคลาสที่สืบทอดมาจาก BankAccount และเพิ่ม Attribute: interest rate อัตราดอกเบี้ย และเมธอดในการเพิ่มดอกเบี้ยไปยังเงินต้น addInterest()

โค้ดทดสอบ

```
myAccount = InterestAccount("Sarayut", "0001", 1000, 5)
myAccount.printBalance()
for i in range(1, 6):
    print("=========== Year", i)
    myAccount.addInterest()
    myAccount.printBalance()
```

ผลรัน

```
>>>
Name: Sarayut    Account Number: 0001    Balance: 1000
=========== Year 1
Name: Sarayut    Account Number: 0001    Balance: 1050.0
=========== Year 2
Name: Sarayut    Account Number: 0001    Balance: 1102.5
=========== Year 3
Name: Sarayut    Account Number: 0001    Balance: 1157.62
=========== Year 4
Name: Sarayut    Account Number: 0001    Balance: 1215.51
=========== Year 5
Name: Sarayut    Account Number: 0001    Balance: 1276.28
```