



Class & OOP for Python programming by Uncle Engineer



Uncle Engineer
ลุงวิศวกร สอนคำนวณ

Contents

- Class, Object & OOP
- Constructor
- Static
- 4 principles of OOP

Class, Object & OOP

Class, Object OOP

คลาส (Class) คือ แม่แบบ หรือพิมพ์เขียวที่สร้างขึ้น เพื่อนำไปใช้สร้างวัตถุ (Object)

วัตถุ (Object) คือสิ่งที่มีอยู่จริงบนโลก เป็นได้ทั้งรูปธรรม และนามธรรม วัตถุแต่ละชิ้นสามารถกำหนดคุณสมบัติเฉพาะของตัวเองได้ ทำให้แต่ละวัตถุมีความแตกต่างกัน แต่คุณสมบัติพื้นฐาน ยังได้รับมาจากคลาส หรือแม่แบบเหมือนเดิม

“วัตถุ” ไม่สามารถเกิดขึ้นเองได้ ถ้าไม่มี “แม่แบบ”

Class, Object OOP

5



Class, Object OOP

6



Class, Object OOP

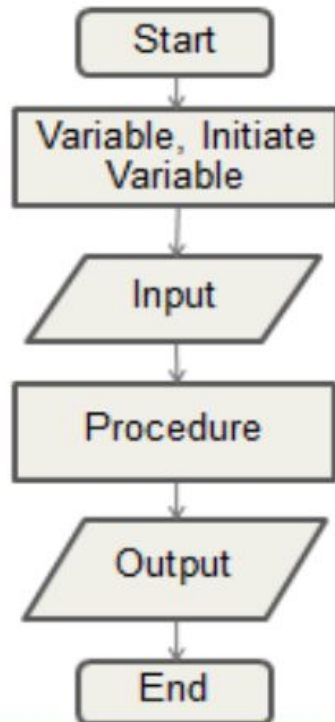
คลาส ในโลกของการเขียนโปรแกรม คือการรวบรวมกลุ่มของคำสิ่งที่มีความสัมพันธ์กัน มีส่วนประกอบ 2 อย่างคือ

- attribute คือ ข้อมูลที่บอกคุณลักษณะทั่วไป หรือคุณสมบัติเฉพาะตัวของวัตถุว่ามีข้อมูลอะไรบ้าง
- method คือ ข้อมูลที่บอกหน้าที่ พฤติกรรม หรือการกระทำของวัตถุว่าสามารถทำอะไรได้บ้าง

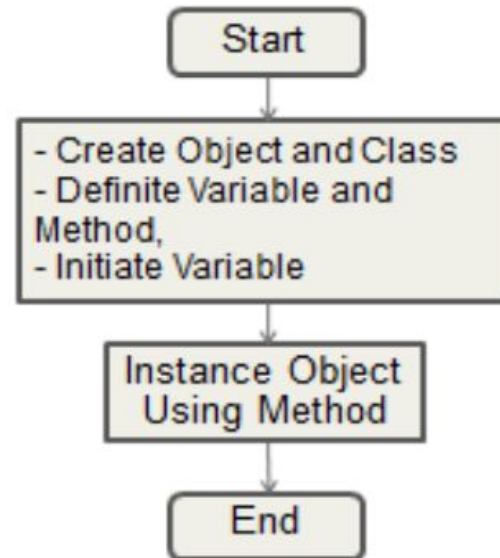
การเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programing : OOP) คือแนวคิดการเขียนโปรแกรม ที่มองทุกสิ่งทุกอย่างให้เป็นวัตถุ (object)

Class, Object OOP

► *structured programming*



► *object-oriented programming.*



Class, Object OOP

แนวทางการตั้งชื่อคลาส, แอตทริบิวต์ และเมธอด

- ชื่อคลาส ควรเป็นคำนาม และขึ้นต้นด้วยอักษรตัวพิมพ์ใหญ่ ถ้าประกอบด้วยคำมากกว่า 1 คำ ให้ตัวขึ้นต้นแต่ละคำด้วยอักษรตัวพิมพ์ใหญ่ เขียนติดกัน ห้ามเว้นวรรค หลีกเลี่ยงการใช้เครื่องหมาย _ (underscore) คั่น
- ชื่อแอตทริบิวต์ ควรเป็นคำนาม หรือคำวิเศษณ์ และขึ้นต้นด้วยอักษรตัวพิมพ์เล็ก ถ้าประกอบด้วยคำมากกว่า 1 คำ ให้คำหลังขึ้นต้นด้วยอักษรตัวพิมพ์ใหญ่ เขียนติดกัน ห้ามเว้นวรรค หลีกเลี่ยงการใช้เครื่องหมาย _ (underscore) คั่น
- ชื่อเมธอด ควรเป็นคำกริยา และขึ้นต้นด้วยอักษรตัวพิมพ์เล็ก ถ้าประกอบด้วยคำมากกว่า 1 คำ ให้คำหลังขึ้นต้นด้วยอักษรตัวพิมพ์ใหญ่ เขียนติดกัน ห้ามเว้นวรรค หลีกเลี่ยงการใช้เครื่องหมาย _ (underscore) คั่น

Class example

```
01_class.py x
1 class Uncle:
2     fullname = 'ลุง วิศวกร'
3     occupation = 'วิศวกร'
4     age = 50
5     money = 100000.25
6
7     def learn(self):
8         print('ฉันกำลังเรียนเขียนโปรแกรม Python')
```

} Attribute

← Method

การสร้าง object

syntax

instance_name = class_name

เมื่อเราทดสอบแสดงประเภทของตัวแปร
โดยใช้คำสั่ง type จะพบว่าทั้ง uncle1
และ uncle2 ล้วนเป็น object ที่อยู่ใน
คลาส Uncle นั้นเอง

```

FOLDERS
▼ Python_Class-OOP
  /* 01_class.py
1  class Uncle:
2      fullname = 'ลุง วิศวกร'
3      occupation = 'วิศวกร'
4      age = 50
5      money = 100000.25
6
7      def learn(self):
8          print('ฉันกำลังเรียนเขียนโปรแกรม Python')
9
10 if __name__ == '__main__':
11     uncle1 = Uncle()
12     print(type(uncle1))
13     uncle2 = Uncle()
14     print(type(uncle2))

```

```

<class '__main__.Uncle'>
<class '__main__.Uncle'>
[Finished in 125ms]

```

การเรียกใช้งานตัวแปร และเมธอด

เรียกใช้งานตัวแปร -> object_name.attribute_name

เรียกใช้งานฟังก์ชัน -> object_name.method_name()

```
01_class.py x
10 if __name__ == '__main__':
11     uncle1 = Uncle()
12
13     print(f'ชื่อ : {uncle1.fullname}')
14     print(f'อาชีพ : {uncle1.occupation}')
15     print(f'อายุ : {uncle1.age}')
16     print(f'เงินในบัญชี : {uncle1.money}')
17
18     uncle1.learn()
```



```
ชื่อ : ลุง วิศวกร
อาชีพ : วิศวกร
อายุ : 50
เงินในบัญชี : 100000.25
ฉันกำลังเรียนเขียนโปรแกรม Python
[Finished in 141ms]
```

Constructor

Constructor

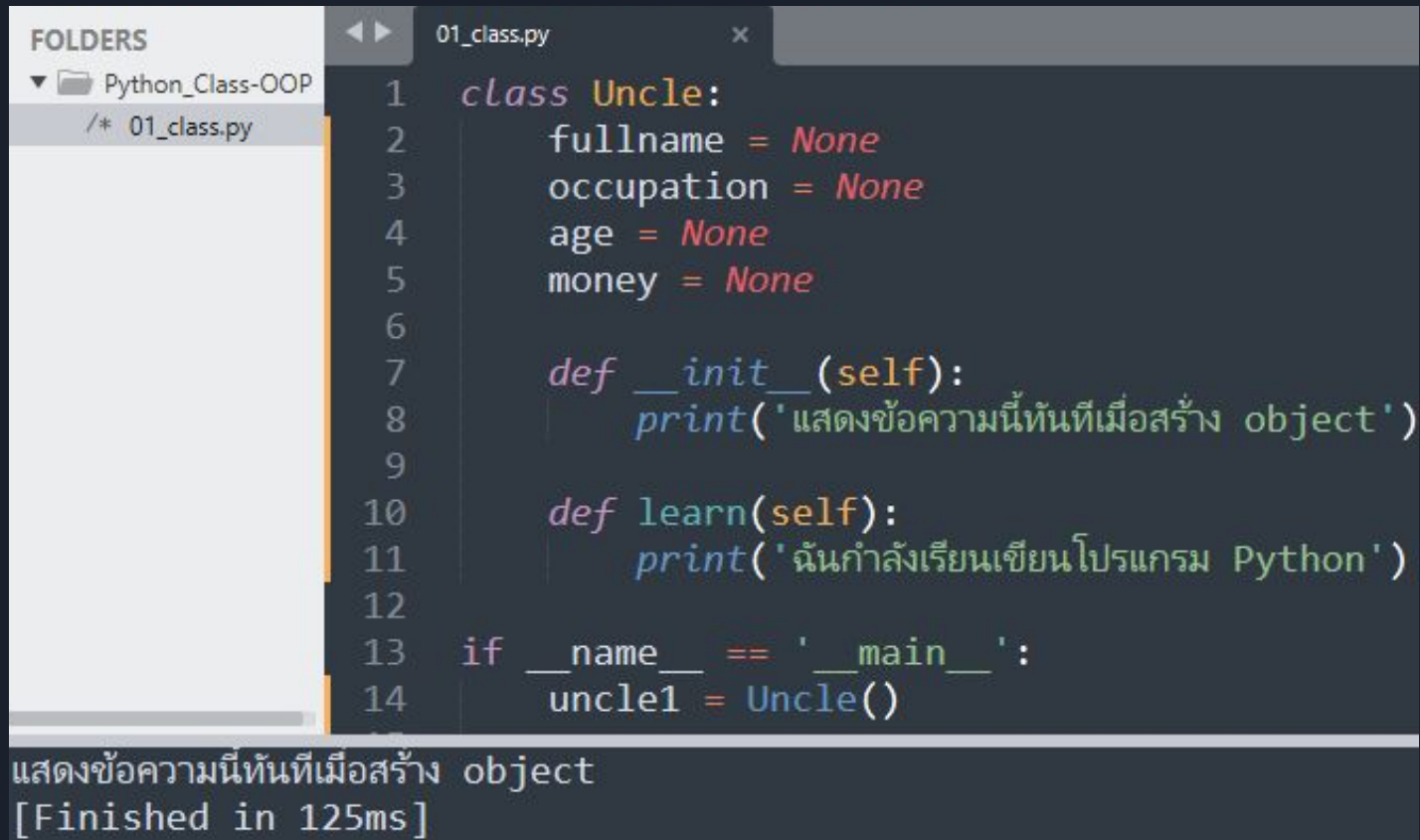
Constructor เป็นเมธอดตัวหนึ่งภายในคลาส จะทำงานโดยอัตโนมัติทันทีหลังจากสร้าง object

ในภาษาโปรแกรมอย่าง C หรือ Java จะเป็นเมธอดที่มีชื่อเดียวกับชื่อคลาส แต่ในภาษา Python จะเป็นเมธอดพิเศษที่มีชื่อว่า `__init__()`

Constructor ใน Python มี 2 ชนิด คือ

- Constructor แบบไม่มีพารามิเตอร์ (Non-Parameterized Constructor)
- Constructor แบบมีพารามิเตอร์ (Parameterized Constructor)

Non-Parameterized Constructor



```
FOLDERS
▼ Python_Class-OOP
  /* 01_class.py

1  class Uncle:
2      fullname = None
3      occupation = None
4      age = None
5      money = None
6
7      def __init__(self):
8          print('แสดงข้อความนี้ทันทีเมื่อสร้าง object')
9
10     def learn(self):
11         print('ฉันกำลังเรียนเขียนโปรแกรม Python')
12
13 if __name__ == '__main__':
14     uncle1 = Uncle()
```

แสดงข้อความนี้ทันทีเมื่อสร้าง object
[Finished in 125ms]

Non-Parameterized Constructor

```
01_class.py x
1 class Uncle:
2     fullname = None
3     occupation = None
4     age = None
5     money = None
6
7     def __init__(self):
8         self.fullname = 'ลุง วิศวกร'
9         self.occupation = 'วิศวกร'
10        self.age = 50
11        self.money = 50000.50
12
13    def learn(self):
14        print('ฉันกำลังเรียนเขียนโปรแกรม Python')
```

```
FOLDERS
▼ Python_Class-OOP
  /* 01_class.py

01_class.py x
16 if __name__ == '__main__':
17     uncle1 = Uncle()
18     print(f'ชื่อ : {uncle1.fullname}')
19     print(f'อาชีพ : {uncle1.occupation}')
20     print(f'อายุ : {uncle1.age}')
21     print(f'เงินในบัญชี : {uncle1.money}')
22
23     uncle1.learn()

ชื่อ : ลุง วิศวกร
อาชีพ : วิศวกร
อายุ : 50
เงินในบัญชี : 50000.5
ฉันกำลังเรียนเขียนโปรแกรม Python
[Finished in 141ms]
```


Parameterized Constructor

```
01_class.py x
1  class Uncle:
2      fullname = None
3      occupation = None
4      age = None
5      money = None
6
7      def __init__(self, fullname, occupation, age, money):
8          self.fullname = fullname
9          self.occupation = occupation
10         self.age = age
11         self.money = money
12
13     def learn(self):
14         print('ฉันกำลังเรียนเขียนโปรแกรม Python')
```

self เป็นคีย์เวิร์ดที่ใช้สำหรับอ้างถึงตัวแปรที่อยู่ในคลาส เพื่อให้แตกต่างจากชื่อที่ถูกมองเห็นในขอบเขตนั้น จะใช้ในกรณีที่ตัวแปรในคลาส มีชื่อเหมือนกับพารามิเตอร์ หรือเมธอดที่อยู่ในคลาส

Parameterized Constructor

```
FOLDERS
▼ Python_Class-OOP
  /* 01_class.py

01_class.py x
16  if __name__ == '__main__':
17      uncle1 = Uncle('ลุง วิศวกร', 'วิศวกร', 50, 50000.50)
18
19      print(f'ชื่อ : {uncle1.fullname}')
20      print(f'อาชีพ : {uncle1.occupation}')
21      print(f'อายุ : {uncle1.age}')
22      print(f'เงินในบัญชี : {uncle1.money}')
23
24      uncle1.learn()
```

ชื่อ : ลุง วิศวกร

อาชีพ : วิศวกร

อายุ : 50

เงินในบัญชี : 50000.5

ฉันกำลังเรียนเขียนโปรแกรม Python

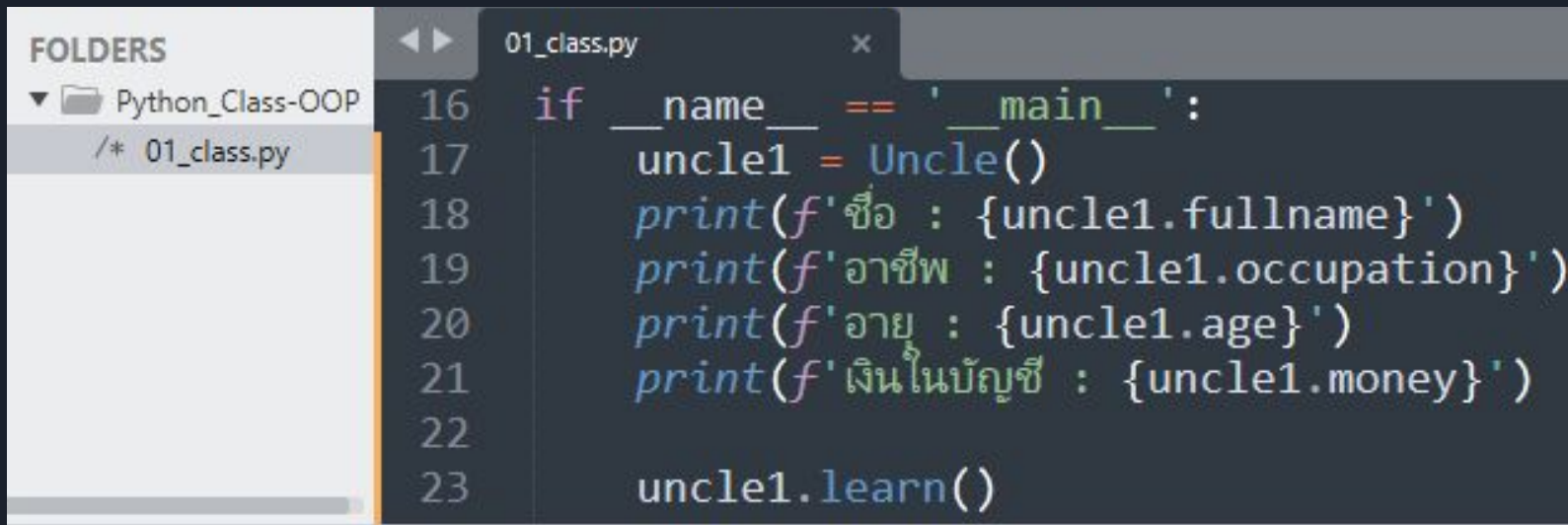
[Finished in 141ms]

Parameterized Constructor

parameter ที่อยู่ใน constructor สามารถกำหนดเป็นค่า default ได้ และเมื่อมีการสร้าง object จะระบุค่าที่ส่งไปใน parameter หรือไม่ก็ได้

```
01_class.py x
1  class Uncle:
2      fullname = None
3      occupation = None
4      age = None
5      money = None
6
7      def __init__(self, fullname='ลุง วิสวกร', occupation='วิศวกร', age=50, money=50000.50):
8          self.fullname = fullname
9          self.occupation = occupation
10         self.age = age
11         self.money = money
12
13     def learn(self):
14         print('ฉันกำลังเรียนเขียนโปรแกรม Python')
```

Parameterized Constructor



The image shows a code editor window with a file named '01_class.py'. The left sidebar shows a folder structure with 'Python_Class-OOP' and a file '01_class.py'. The main editor area contains the following Python code:

```
16 if __name__ == '__main__':  
17     uncle1 = Uncle()  
18     print(f'ชื่อ : {uncle1.fullname}')
```

```
19     print(f'อาชีพ : {uncle1.occupation}')
```

```
20     print(f'อายุ : {uncle1.age}')
```

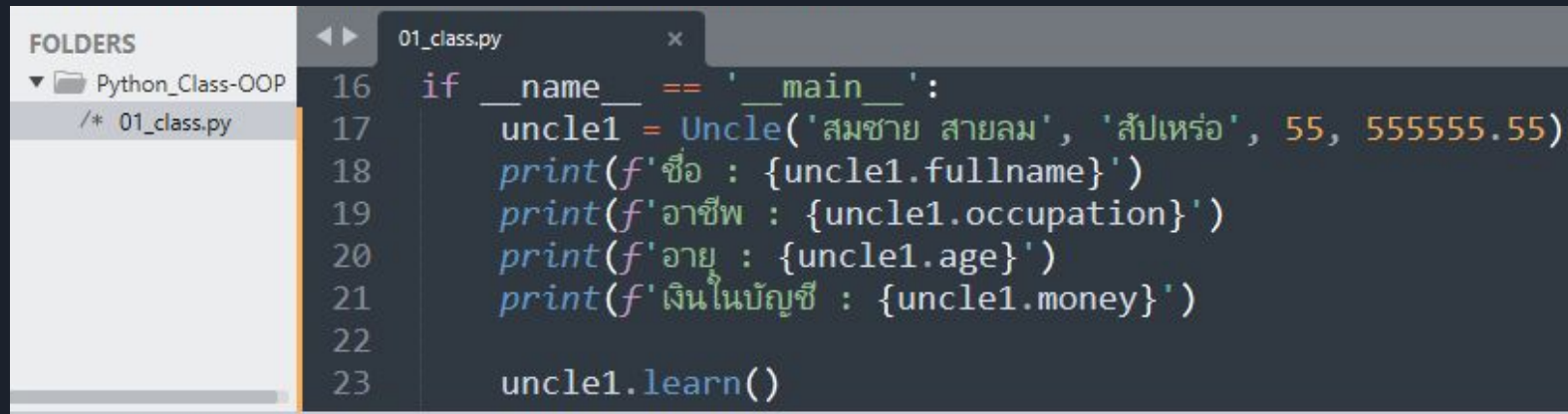
```
21     print(f'เงินในบัญชี : {uncle1.money}')
```

```
22  
23     uncle1.learn()
```

Below the code editor, the output of the program is displayed:

```
ชื่อ : ลุง วิศวกร  
อาชีพ : วิศวกร  
อายุ : 50  
เงินในบัญชี : 50000.5  
ฉันกำลังเรียนเขียนโปรแกรม Python  
[Finished in 141ms]
```

Parameterized Constructor



```
FOLDERS
▼ Python_Class-OOP
  /* 01_class.py

01_class.py x
16 if __name__ == '__main__':
17     uncle1 = Uncle('สมชาย สายลม', 'สัปเหร่อ', 55, 555555.55)
18     print(f'ชื่อ : {uncle1.fullname}')
19     print(f'อาชีพ : {uncle1.occupation}')
20     print(f'อายุ : {uncle1.age}')
21     print(f'เงินในบัญชี : {uncle1.money}')
22
23     uncle1.learn()
```

ชื่อ : สมชาย สายลม

อาชีพ : สัปเหร่อ

อายุ : 55

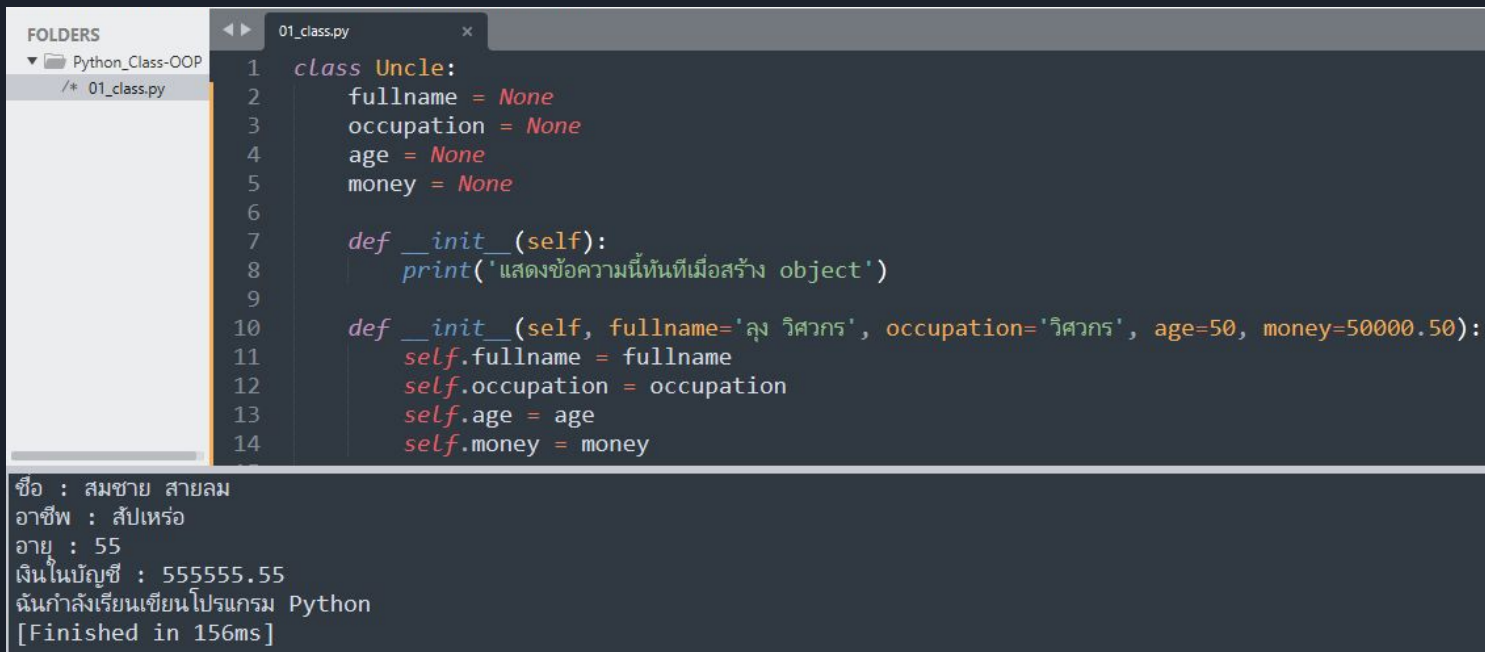
เงินในบัญชี : 555555.55

ฉันกำลังเรียนเขียนโปรแกรม Python

[Finished in 156ms]

ข้อควรระวังในการกำหนด Constructor

ในภาษา Python ไม่แนะนำให้มี Constructor อยู่ในคลาสเดียวกัน มากกว่า 1 ตัว เนื่องจาก Interpreter ของ Python จะเลือกทำงานใน Constructor ตัวหลังสุด และมีจำนวน parameter ที่มากกว่า



```
FOLDERS
▼ Python_Class-OOP
  /* 01_class.py

1  class Uncle:
2      fullname = None
3      occupation = None
4      age = None
5      money = None
6
7      def __init__(self):
8          print('แสดงข้อความนี้ทันทีเมื่อสร้าง object')
9
10     def __init__(self, fullname='ลุง วิศวกร', occupation='วิศวกร', age=50, money=50000.50):
11         self.fullname = fullname
12         self.occupation = occupation
13         self.age = age
14         self.money = money
```

ชื่อ : สมชาย สายลม
อาชีพ : สัปเหร่อ
อายุ : 55
เงินในบัญชี : 555555.55
ฉันกำลังเรียนเขียนโปรแกรม Python
[Finished in 156ms]

Static

Static

Static เป็นวิธีการเรียกใช้งานตัวแปร หรือเมธอด โดยไม่ต้องสร้าง object ซึ่งจะใช้งาน

หน่วยความจำเฉพาะการทำงานในครั้งแรกเท่านั้น เมื่อจบการทำงานแล้ว จะคืนหน่วยความจำทันที และการใช้งานครั้งต่อไป จะใช้ค่าจากหน่วยความจำเดิม

ควรใช้ static ในกรณีดังต่อไปนี้

- เป็นค่าคงที่ หรือไม่ต้องการเปลี่ยนแปลงค่า
- นำค่าตัวแปร หรือเมธอดไปใช้กับหลายๆ คลาส

Static

ในภาษา Python แอตทริบิวต์ (ตัวแปร) ในคลาส จะเป็น static อยู่แล้ว สามารถเรียกใช้ผ่าน “ชื่อคลาส.ชื่อตัวแปร” ได้เลย แต่สำหรับเมธอดนั้น ถ้าต้องการกำหนดให้เมธอดใด เป็น static ไม่ต้องใส่ self ไว้ในพารามิเตอร์ แต่สามารถรับค่าพารามิเตอร์รูปแบบอื่นๆ ได้ตามปกติ

การเรียกใช้งานตัวแปร หรือเมธอด non-static และ static ภายในคลาส

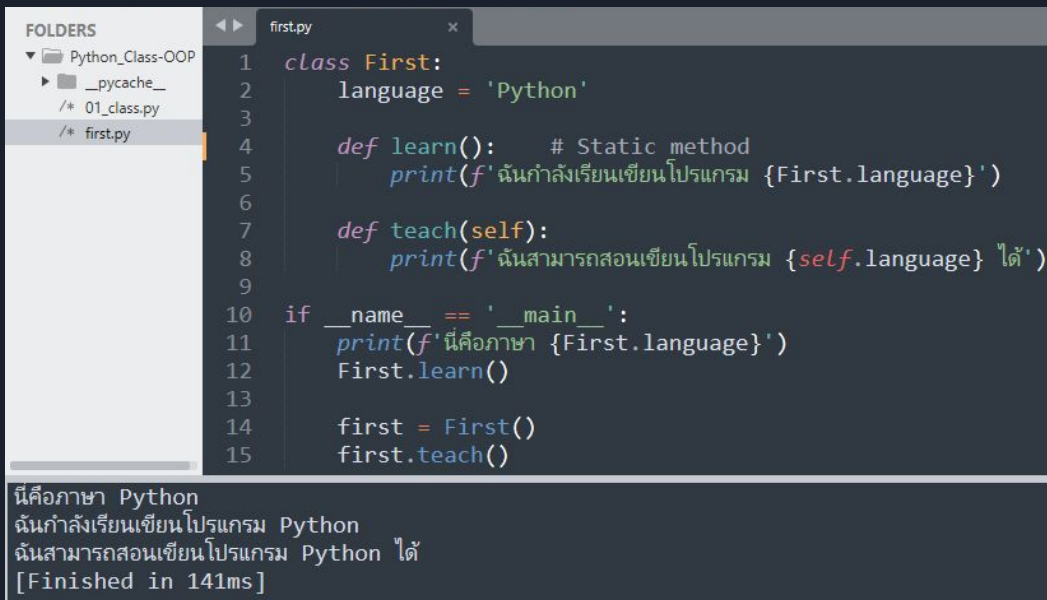
เรียกใช้งานตัวแปร -> class_name.attribute_name

เรียกใช้งานฟังก์ชัน -> class_name.method_name();

```
first.py x
1 class First:
2     language = 'Python'
3
4     def learn():      # Static method
5         print(f'ฉันกำลังเรียนเขียนโปรแกรม {First.language}')
6
7     def teach(self):
8         First.learn()
9         print(f'ฉันสามารถสอนเขียนโปรแกรม {self.language} ได้')
```

การเรียกใช้งานตัวแปร หรือเมธอด non-static และ static ภายนอกคลาส

ฟังก์ชัน `if __name__ == '__main__':` ถือว่าอยู่ภายนอกคลาส ถ้าเป็น static จะต้องเรียกชื่อคลาส ตามด้วยชื่อตัวแปร หรือเมธอด แต่ถ้าเป็น non-static ต้องเรียกใช้ผ่าน object



```
FOLDERS
▼ Python_Class-OOP
  ► __pycache__
    /* 01_class.py
    /* first.py

first.py
1  class First:
2      language = 'Python'
3
4      def learn(): # Static method
5          print(f'ฉันกำลังเรียนเขียนโปรแกรม {First.language}')
6
7      def teach(self):
8          print(f'ฉันสามารถสอนเขียนโปรแกรม {self.language} ได้')
9
10 if __name__ == '__main__':
11     print(f'นี่คือภาษา {First.language}')
12     First.learn()
13
14     first = First()
15     first.teach()
```

นี่คือภาษา Python
ฉันกำลังเรียนเขียนโปรแกรม Python
ฉันสามารถสอนเขียนโปรแกรม Python ได้
[Finished in 141ms]

การเรียกใช้งานตัวแปร หรือเมธอด non-static และ static ภายนอกคลาส

ไฟล์นามสกุล .py ถือว่าเป็นโมดูลในภาษา Python ที่สามารถใช้คำสั่ง from หรือ import เพื่อเข้าถึงข้อมูลจากไฟล์ หรือโมดูลอื่นได้ และสามารถเรียกใช้ตัวแปร หรือเมธอด ที่เป็น static ไปใช้กับคลาสอื่นๆ ได้โดยไม่ต้องสร้าง object แต่ถ้าไม่เป็น static ต้องสร้าง และเรียกใช้งานผ่าน object

if __name__ == '__main__': คือการตรวจสอบว่า คำสั่งที่รันอยู่ภายใน __main__ มาจากคลาสที่อยู่ในไฟล์ หรือโมดูลเดียวกันหรือไม่ ช่วยให้สามารถที่จะเลือกรันหรือไม่รันโค้ดในไฟล์ที่ต้องการได้ ซึ่งมีประโยชน์เป็นอย่างมากเกี่ยวกับการรันและการอิมพอร์ตโมดูลต่าง ๆ เข้ามาใช้งาน

การเรียกใช้งานตัวแปร หรือเมธอด non-static และ static ภายนอกคลาส



```
FOLDERS
▼ Python_Class-OOP
  ► __pycache__
    /* 01_class.py
    /* first.py
    /* second.py

1  from first import First
2
3  class Second:
4      pass
5
6  if __name__ == '__main__':
7      print(f'นี่คือภาษา {First.language}')
8      First.learn()
9
10     first = First()
11     first.teach()
```

นี่คือภาษา Python
ฉันกำลังเรียนเขียนโปรแกรม Python
ฉันสามารถสอนเขียนโปรแกรม Python ได้
[Finished in 125ms]

สรุปการเรียกใช้งานเมธอด non-static และ static กับคลาส

การเรียกใช้งาน เมธอด	ภายในคลาส	ภายนอกคลาส
ไม่เป็น static	ต้องสร้าง object และเรียกใช้งานผ่าน object	ต้องสร้าง object และเรียกใช้งานผ่าน object
เป็น static	เรียกชื่อคลาส.ชื่อเมธอด	เรียกชื่อคลาส.ชื่อเมธอด

4 principles of OOP

4 principles of OOP

ในภาษา Python รองรับการเขียนโปรแกรมแบบ OOP เช่นเดียวกับภาษาโปรแกรม

หลายๆ ภาษา ซึ่งภาษาโปรแกรมที่เป็น OOP นั้น จะต้องมี “เสาหลัก 4 ประการ” คือ

- Inheritance (การสืบทอด)
- Encapsulation (การห่อหุ้ม)
- Abstract (การซ่อน)
- Polymorphism (การพ้องรูป)

Inheritance

การสืบทอด (Inheritance) คือการที่คลาสลูก (sub class) รับตัวแปร และเมธอดมาจากคลาสแม่ (super class)

คลาสลูกที่สืบทอดมาจากคลาสแม่ จะมีความสามารถ 2 อย่าง

- ใช้งานตัวแปร และเมธอดจากคลาสแม่ได้
- เพิ่มตัวแปร และเมธอดในคลาสตัวเองได้

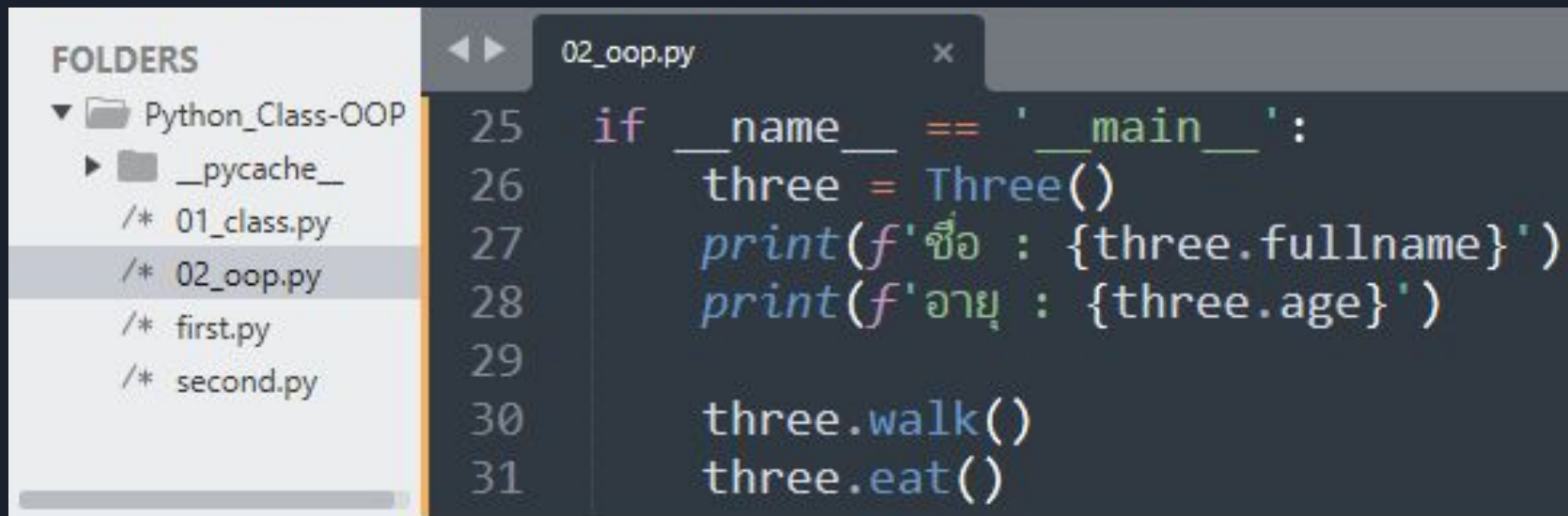
วิธีการสืบทอดคลาสในภาษา Python จะใช้ชื่อคลาสแม่ ไปอยู่ในวงเล็บต่อท้ายชื่อคลาสลูก และสามารถสืบทอดได้มากกว่า 1 คลาสในเวลาเดียวกัน

Inheritance

```
02_oop.py x
1 class One:
2     fullname = 'สมชาย สายลม'
3
4     def walk(self):
5         print('ฉันเดินได้')
6
7 class Two:
8     nickname = 'สมชาย'
9
10    def run(self):
11        print('ฉันวิ่งได้')
12
```

```
13 class Three(One):
14     age = 30
15
16     def eat(self):
17         print('ฉันกินได้')
18
19 class Four(One, Two):
20     money = 1000000
21
22     def fly(self):
23         print('ฉันบินได้')
```

Inheritance

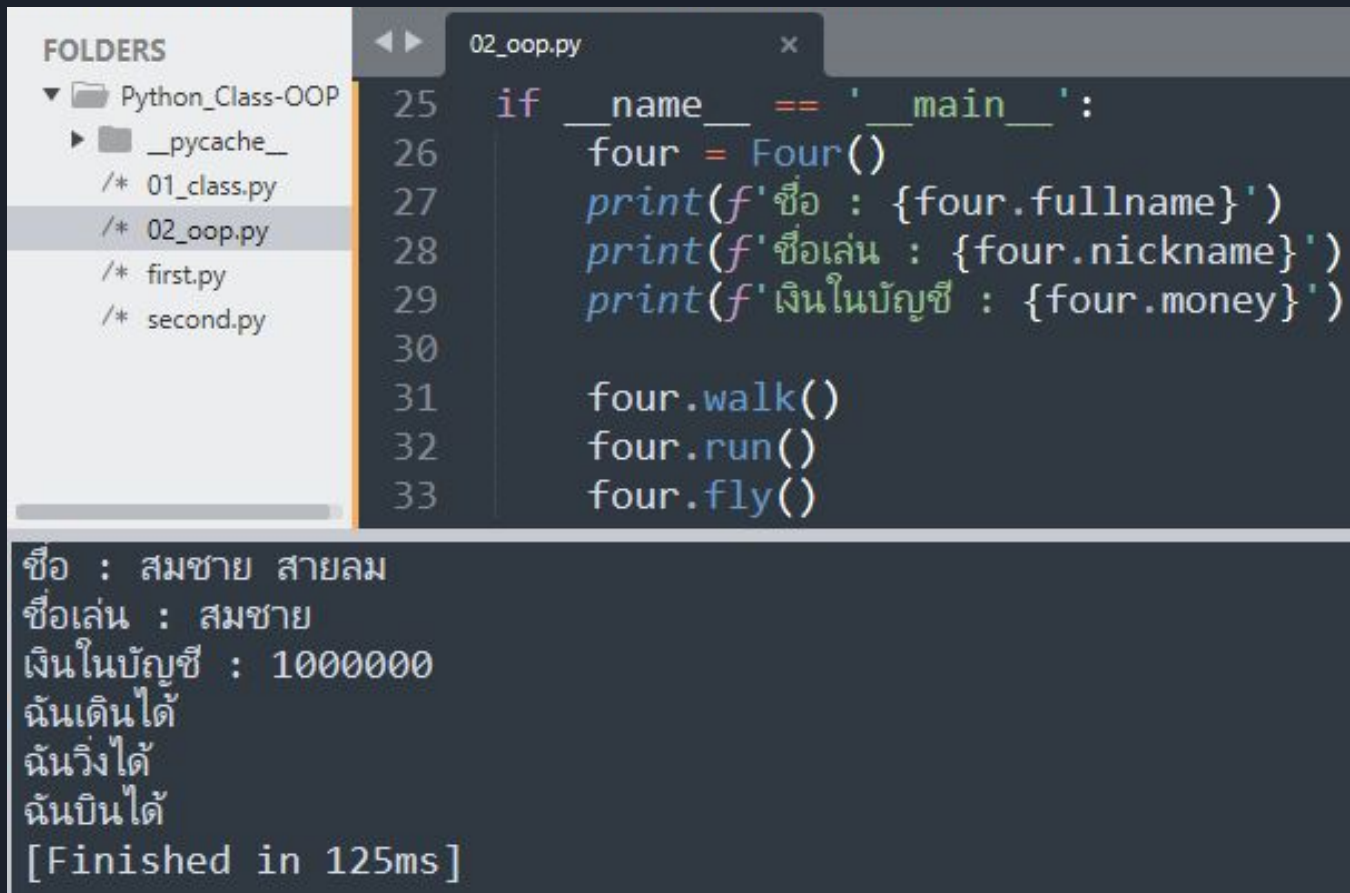


The screenshot shows a Python IDE interface. On the left is a 'FOLDERS' panel with a tree view containing 'Python_Class-OOP', which is expanded to show subfolders like '__pycache__' and files '01_class.py', '02_oop.py', 'first.py', and 'second.py'. The file '02_oop.py' is selected. The main editor window shows the code for '02_oop.py' with line numbers 25 to 31. The code defines a main block that creates an instance of a class named 'Three' and prints its attributes 'fullname' and 'age', then calls 'walk()' and 'eat()' methods.

```
25 if __name__ == '__main__':  
26     three = Three()  
27     print(f'ชื่อ : {three.fullname}')  
28     print(f'อายุ : {three.age}')  
29  
30     three.walk()  
31     three.eat()
```

```
ชื่อ : สมชาย สายลม  
อายุ : 30  
ฉันเดินได้  
ฉันกินได้  
[Finished in 125ms]
```

Inheritance



```
FOLDERS
▼ Python_Class-OOP
  ► __pycache__
  /* 01_class.py
  /* 02_oop.py
  /* first.py
  /* second.py

25 if __name__ == '__main__':
26     four = Four()
27     print(f'ชื่อ : {four.fullname}')
28     print(f'ชื่อเล่น : {four.nickname}')
29     print(f'เงินในบัญชี : {four.money}')
30
31     four.walk()
32     four.run()
33     four.fly()

ชื่อ : สมชาย สายลม
ชื่อเล่น : สมชาย
เงินในบัญชี : 1000000
ฉันเดินได้
ฉันวิ่งได้
ฉันบินได้
[Finished in 125ms]
```

Inheritance : super()

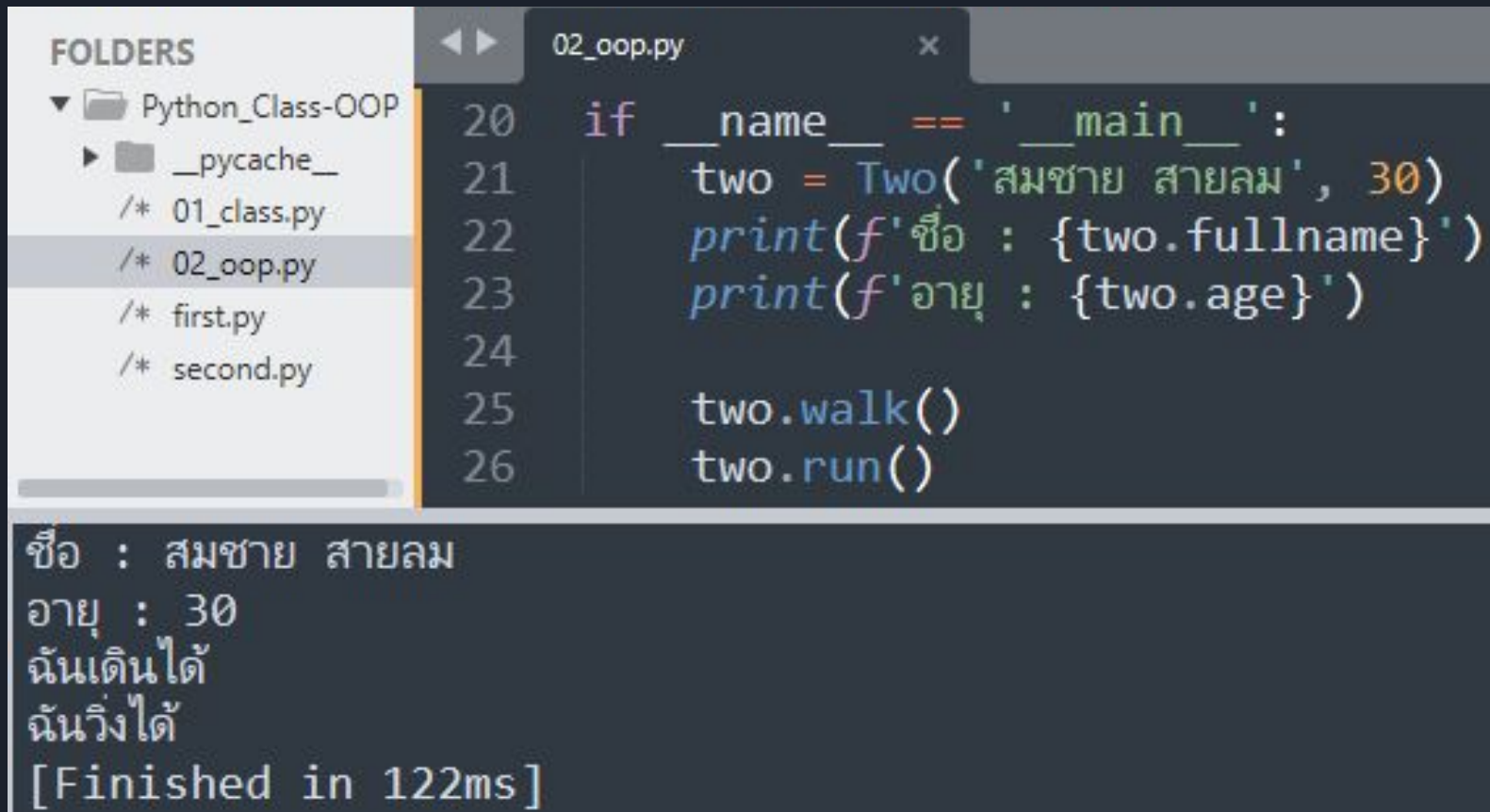
super เป็นคีย์เวิร์ดที่ใช้สำหรับเรียกใช้งานตัวแปร เมธอด จากคลาสแม่ (super class)

หรือกำหนด constructor จากคลาสลูก (sub class) ให้กับคลาสแม่ (super class)

```
02_oop.py x
1 class One:
2     fullname = None
3
4     def __init__(self, fullname):
5         self.fullname = fullname
6
7     def walk(self):
8         print('ฉันเดินได้')
9
```

```
10 class Two(One):
11     age = None
12
13     def __init__(self, fullname, age):
14         super().__init__(fullname)
15         self.age = age
16
17     def run(self):
18         print('ฉันวิ่งได้')
```

Inheritance : super()



The screenshot shows a Python IDE interface. On the left is a 'FOLDERS' panel with a tree view containing 'Python_Class-OOP', '__pycache__', and several Python files. The main editor window, titled '02_oop.py', displays Python code. The code defines a class 'Two' (partially visible) and uses 'super()' in its methods. The 'if __name__ == '__main__':' block creates an instance 'two' with Thai name and age, prints its details, and calls 'walk()' and 'run()' methods. Below the editor is a terminal window showing the program's output in Thai and the execution time.

```
FOLDERS
▼ Python_Class-OOP
  ► __pycache__
    /* 01_class.py
    /* 02_oop.py
    /* first.py
    /* second.py

02_oop.py
20 if __name__ == '__main__':
21     two = Two('สมชาย สายลม', 30)
22     print(f'ชื่อ : {two.fullname}')
23     print(f'อายุ : {two.age}')
24
25     two.walk()
26     two.run()

ชื่อ : สมชาย สายลม
อายุ : 30
ฉันเดินได้
ฉันวิ่งได้
[Finished in 122ms]
```

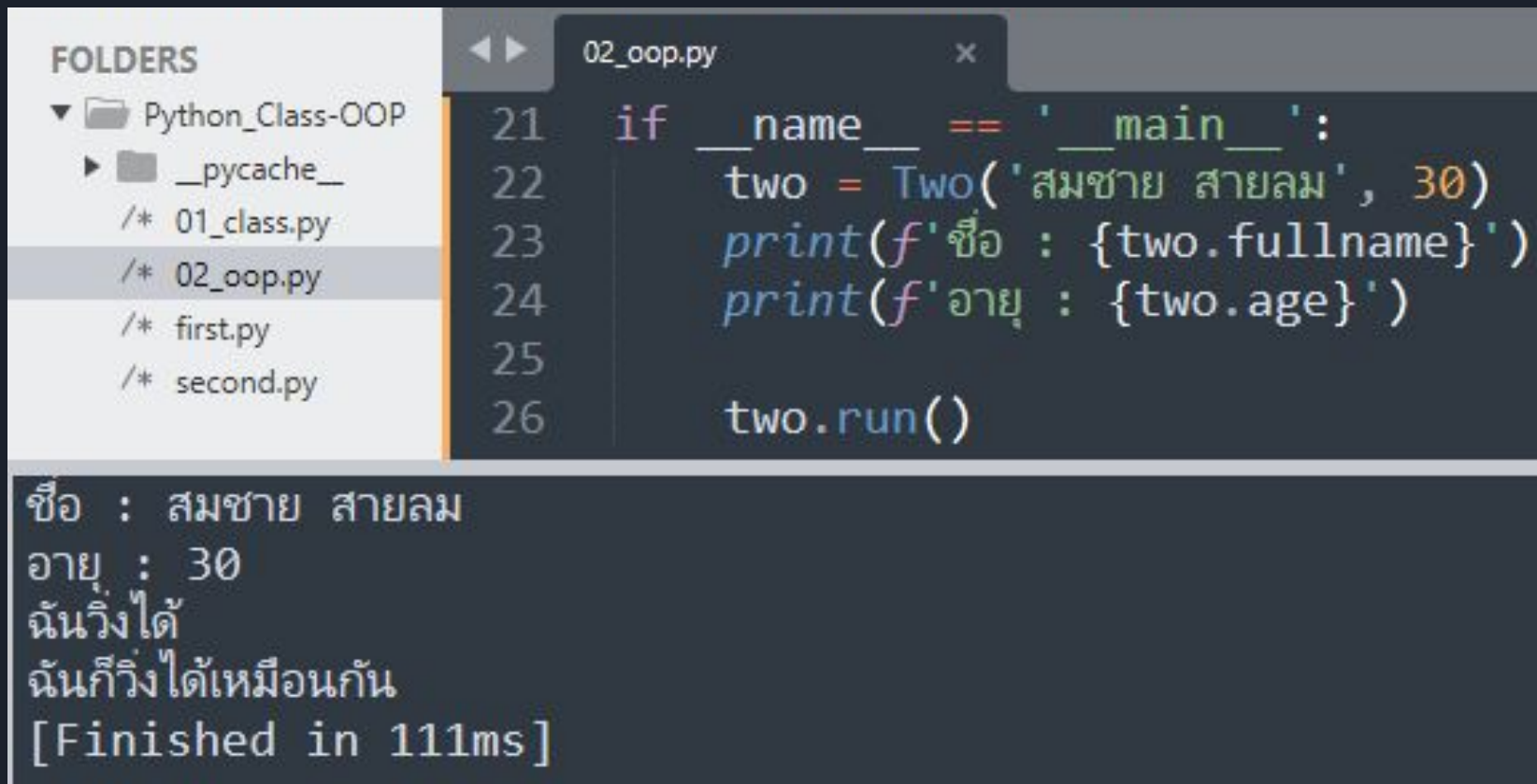
Inheritance : super() (method overriding)

override คือการทำให้เมธอดของคลาสลูก (sub class) ทำงานแตกต่างจากเมธอด ของคลาสแม่ (super class) โดยจะมีรูปแบบของเมธอดที่เหมือนกัน ซึ่งเรียกแบบง่ายๆ ว่า “การเขียนทับ” นั่นเอง ให้ใช้คำสั่ง `super().method_name()`

```
02_oop.py x
1 class One:
2     fullname = None
3
4     def __init__(self, fullname):
5         self.fullname = fullname
6
7     def run(self):
8         print('ฉันวิ่งได้')
9
```

```
10 class Two(One):
11     age = None
12
13     def __init__(self, fullname, age):
14         super().__init__(fullname)
15         self.age = age
16
17     def run(self):
18         super().run()
19         print('ฉันก็วิ่งได้เหมือนกัน')
```


Inheritance : super() (method overriding)



The screenshot shows a Python IDE interface. On the left is a 'FOLDERS' panel showing a project named 'Python_Class-OOP' with subfolders like '__pycache__' and files '01_class.py', '02_oop.py', 'first.py', and 'second.py'. The '02_oop.py' file is selected and its code is displayed in the main editor. The code defines a 'Two' class and a 'main' block that creates an instance of 'Two' with the name 'สมชาย สายลม' and age 30, then prints its details and calls the 'run()' method. Below the editor is a terminal window showing the output of the program: 'ชื่อ : สมชาย สายลม', 'อายุ : 30', 'ฉันวิ่งได้', 'ฉันก็วิ่งได้เหมือนกัน', and a completion message '[Finished in 111ms]'.

```
FOLDERS
▼ Python_Class-OOP
  ► __pycache__
    /* 01_class.py
    /* 02_oop.py
    /* first.py
    /* second.py

02_oop.py
21  if __name__ == '__main__':
22      two = Two('สมชาย สายลม', 30)
23      print(f'ชื่อ : {two.fullname}')
24      print(f'อายุ : {two.age}')
25
26      two.run()

ชื่อ : สมชาย สายลม
อายุ : 30
ฉันวิ่งได้
ฉันก็วิ่งได้เหมือนกัน
[Finished in 111ms]
```


Encapsulation

การห่อหุ้ม (Encapsulation) คือการกำหนดสิทธิ์ในการเข้าถึงสมาชิกภายในคลาส ไม่ว่าจะมาจากภายในหรือภายนอกคลาสก็ตาม ทำให้ข้อมูลมีความปลอดภัย และเป็นความลับ

ระดับความสามารถในการเข้าถึงตัวแปร และเมธอดในคลาส มี 3 ระดับ คือ

- เข้าถึงตัวแปรและเมธอดได้ทั้งภายในคลาส และภายนอกคลาส
- เข้าถึงตัวแปรและเมธอดได้เฉพาะภายในคลาสตัวเอง และคลาสที่สืบทอดมา
- เข้าถึงตัวแปรและเมธอดได้เฉพาะภายในคลาสตัวเองเท่านั้น

Encapsulation

คีย์เวิร์ดที่ใช้ระบุระดับความสามารถในการเข้าถึงตัวแปร และเมธอดของคลาส
ในภาษา Java และ Python

ระดับสิทธิ์การเข้าถึง	ภาษา Java	ภาษา Python
เข้าถึงได้ทุกคลาส	public	ไม่มีคีย์เวิร์ด
เข้าถึงได้ในคลาสตัวเอง และคลาสที่สืบทอด	protected	_ (single underscore)
เข้าถึงได้เฉพาะในคลาสตัวเอง	private	__ (double underscore)

Encapsulation

กำหนดให้คลาส BankAccount เป็น super class

```
02_oop.py x
1 class BankAccount:
2     def __init__(self, accountname, age, amount):
3         self.accountname = accountname
4         self._age = age
5         self.__amount = amount
6
7     def showMessage():
8         print('กำลังทำรายการฝาก-ถอนเงินในบัญชี')
9
10    def _deposit(self, deposit):
11        self.__amount += deposit
12        print(f'ฝากเพิ่ม {deposit} บาท รวมเงิน {self.__amount} บาท')
13
14    def __withdraw(self, withdraw):
15        self.__amount -= withdraw
16        print(f'ถอนออก {withdraw} บาท เหลือเงิน {self.__amount} บาท')
```

Encapsulation

คลาส Employee เป็น sub class และสืบทอดจากคลาส BankAccount

```
02_oop.py x
1 class BankAccount:
2     def __init__(self, accountname, age, amount):
3         self.accountname = accountname
4         self._age = age
5         self.__amount = amount
6
7     def showMessage(self):
8         print('กำลังทำรายการฝาก-ถอนเงินในบัญชี')
9
10    def _deposit(self, deposit):
11        self.__amount += deposit
12        print(f'ฝากเพิ่ม {deposit} บาท รวมเงิน {self.__amount} บาท')
13
14    def __withdraw(self, withdraw):
15        self.__amount -= withdraw
16        print(f'ถอนออก {withdraw} บาท เหลือเงิน {self.__amount} บาท')
17
18 class Employee(BankAccount):
19     def __init__(self, accountname, age, amount):
20         super().__init__(accountname, age, amount)
```

Encapsulation

object ของคลาสที่สืบทอดมาจากคลาสแม่ สามารถเรียกใช้งานตัวแปร หรือเมธอด ที่เป็น protected ของคลาสแม่ได้

```
02_oop.py x
22  if __name__ == '__main__':
23      employee = Employee('ลุง วิสวกร', 45, 100000)
24      print(f'ชื่อบัญชี : {employee.accountname}')
25      print(f'อายุ : {employee._age}')
26
27      employee.showMessage()
28      employee._deposit(5000)
```

Encapsulation

แต่ตัวแปร หรือเมธอดที่เป็น private จะไม่สามารถเรียกใช้งานภายนอกคลาสได้ (ฟังก์ชัน `if __name__ == '__main__':` ใน Python ถือเป็นการเรียกใช้งานภายนอกคลาส)

```
02_oop.py x
22  if __name__ == '__main__':
23      account = BankAccount('ลุงวิศกร สอนคำนวณ', 55, 50000)
24      print(f'ชื่อบัญชี : {account.accountname}')
25      print(f'อายุ : {account._age}')
26      print(f'จำนวนเงิน : {account.__amount}') # Error
27
28      account.showMessage()
29      account._deposit(500)
30      account.__withdraw(49000) # Error
```

Encapsulation : private accessing

หลักการสำคัญอย่างหนึ่งของ Encapsulation คือ การกำหนดตัวแปร หรือเมธอดให้เป็น private เพื่อควบคุมไม่ให้มีการเข้าถึงข้อมูลแบบ public

แต่ถ้าจะต้องการเข้าถึงตัวแปร หรือเมธอดที่เป็น private ในภาษา Python ก็สามารถทำได้หลายวิธี เช่น

- Underscore accessing
- Getter & Setter
- Property

Encapsulation : Underscore accessing

เรียกชื่อ object เชื่อมด้วยจุด ตามด้วยเครื่องหมาย _ (underscore) หน้าชื่อคลาส

ต่อด้วย __ (double underscore) หน้าชื่อตัวแปร หรือเมธอด

ตัวแปร -> object_name._Class_name__attribute_name

เมธอด -> object_name._Class_name__method_name()

Encapsulation : Underscore accessing

FOLDERS

▼ Python_Class-OOP

► `__pycache__`

`/* 01_class.py`

`/* 02_oop.py`

`/* first.py`

`/* second.py`

02_oop.py

21

`if __name__ == '__main__':`

22

`employee = Employee('ลุง วิศวกร', 45, 100000)`

23

`print(f'ชื่อบัญชี : {employee.accountname}')`

24

`print(f'อายุ : {employee.age}')`

25

`# เข้าถึงตัวแปรที่เป็น private ให้ใช้ ชื่อออบเจกต์._ชื่อคลาส__ชื่อตัวแปร`

26

`print(f'จำนวนเงิน : {employee._BankAccount__amount}')`

27

28

`# เข้าถึงเมธอดที่เป็น private ให้ใช้ ชื่อออบเจกต์._ชื่อคลาส__ชื่อเมธอด()`

29

`employee._BankAccount__withdraw(49000)`

ชื่อบัญชี : ลุง วิศวกร

อายุ : 45

จำนวนเงิน : 100000

ถอนออก 49000 บาท เหลือเงิน 51000 บาท

[Finished in 151ms]

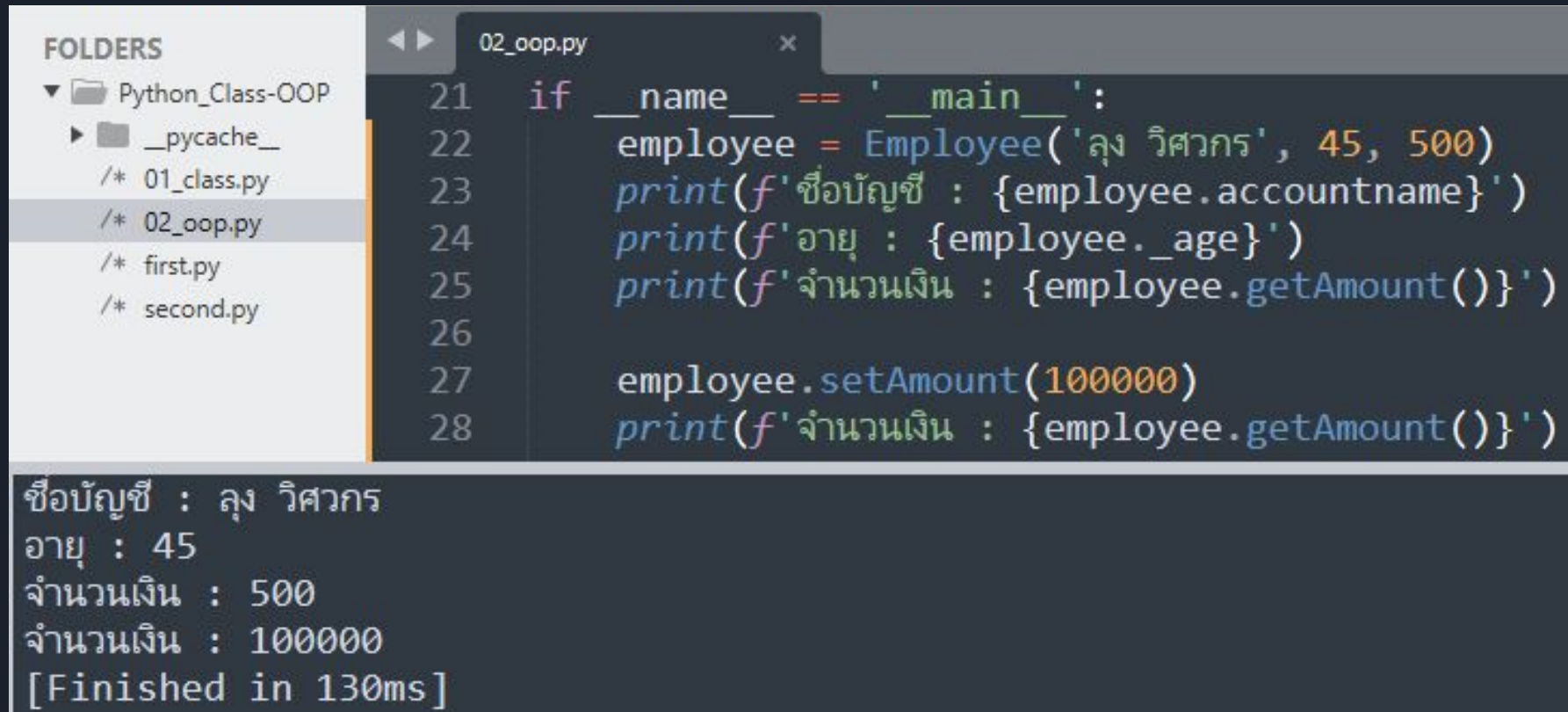
Encapsulation : Getter & Setter

การเข้าถึงตัวแปร `__amount` โดยใช้วิธีสร้างฟังก์ชัน Getter และ Setter

```
02_oop.py x
1  class BankAccount:
2      def __init__(self, accountname, age, amount):
3          self.accountname = accountname
4          self._age = age
5          self.__amount = amount
6
7      def getAmount(self):          # Getter
8          return self.__amount
9
10     def setAmount(self, amount):  # Setter
11         self.__amount = amount
12
13     def __withdraw(self, withdraw):
14         self.__amount -= withdraw
15         print(f'ถอนออก {withdraw} บาท เหลือเงิน {self.__amount} บาท')
16
17 class Employee(BankAccount):
18     def __init__(self, accountname, age, amount):
19         super().__init__(accountname, age, amount)
```

Encapsulation : Getter & Setter

การเข้าถึงตัวแปร `__amount` โดยใช้วิธีสร้างฟังก์ชัน Getter และ Setter



```
FOLDERS
▼ Python_Class-OOP
  ▸ __pycache__
    /* 01_class.py
    /* 02_oop.py
    /* first.py
    /* second.py

21  if __name__ == '__main__':
22      employee = Employee('ลุง วิสวกร', 45, 500)
23      print(f'ชื่อบัญชี : {employee.accountname}')
24      print(f'อายุ : {employee._age}')
25      print(f'จำนวนเงิน : {employee.getAmount()}')
26
27      employee.setAmount(100000)
28      print(f'จำนวนเงิน : {employee.getAmount()}')
```

ชื่อบัญชี : ลุง วิสวกร
อายุ : 45
จำนวนเงิน : 500
จำนวนเงิน : 100000
[Finished in 130ms]

Encapsulation : Property

property คือการสร้าง getter และ setter ในสไลต์ของภาษา Python

ทำได้ 2 วิธี

- ใช้ฟังก์ชัน `property()` แล้วรับค่าพารามิเตอร์ที่เป็น getter และ setter
- แอด Decorator ชื่อว่า `@property` ครอบไว้ด้านบนเมธอดที่เป็น getter ส่วนเมธอดที่เป็น setter ให้แอด Decorator เป็นชื่อ `@method_name.setter` ครอบไว้บนเมธอดที่เป็น setter โดยชื่อเมธอดจะเหมือนกันทั้ง getter และ setter

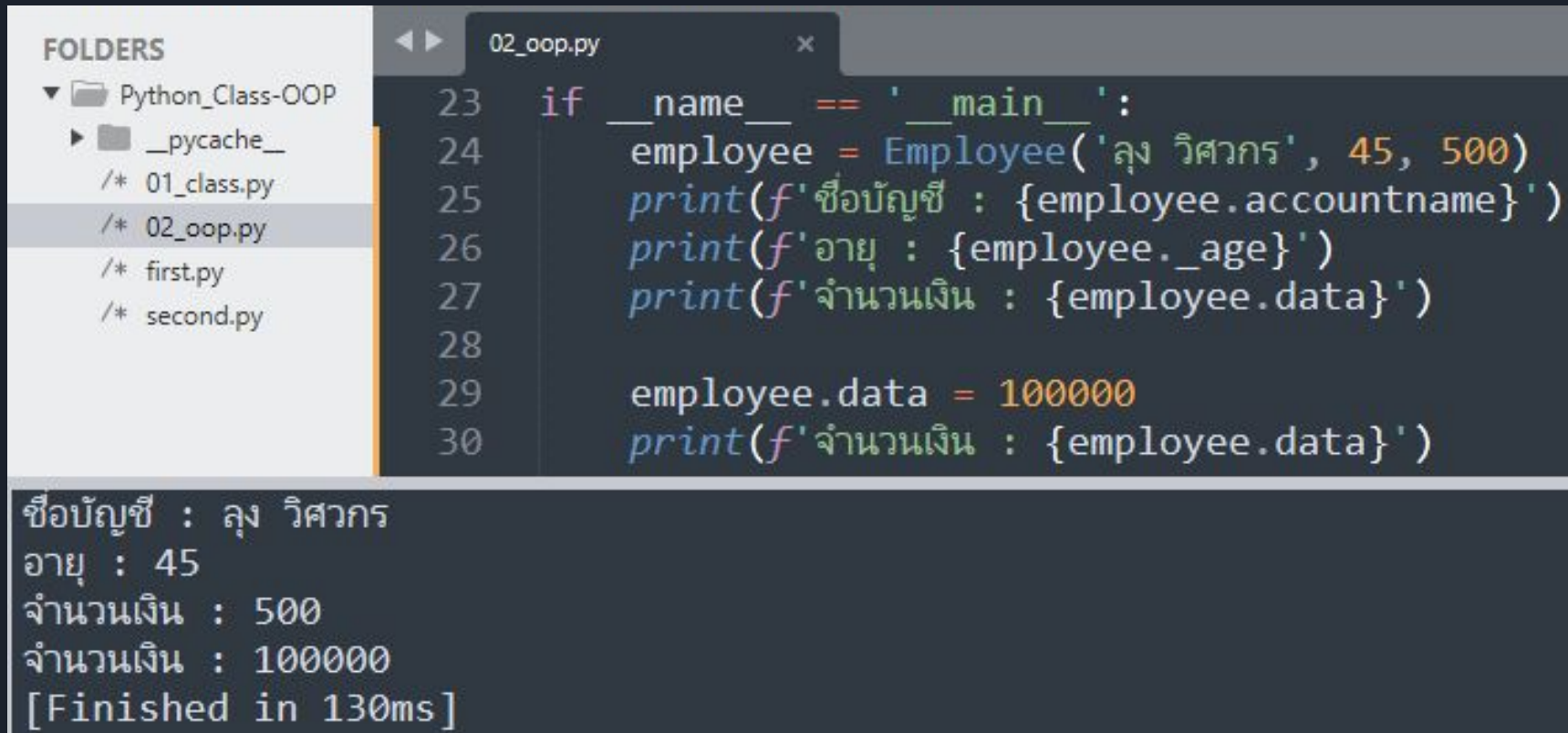
Encapsulation : Property

การเข้าถึงตัวแปร `__amount` โดยใช้ฟังก์ชัน `property()`

```
02_oop.py x
1 class BankAccount:
2     def __init__(self, accountname, age, amount):
3         self.accountname = accountname
4         self._age = age
5         self.__amount = amount
6
7     def getAmount(self):          # Getter
8         return self.__amount
9
10    def setAmount(self, amount):  # Setter
11        self.__amount = amount
12
13    def __withdraw(self, withdraw):
14        self.__amount -= withdraw
15        print(f'ถอนออก {withdraw} บาท เหลือเงิน {self.__amount} บาท')
16
17    data = property(getAmount, setAmount)
18
19 class Employee(BankAccount):
20     def __init__(self, accountname, age, amount):
21         super().__init__(accountname, age, amount)
```

Encapsulation : Property

การเข้าถึงตัวแปร __amount โดยใช้ฟังก์ชัน property()



```
FOLDERS
▼ Python_Class-OOP
  ► __pycache__
  /* 01_class.py
  /* 02_oop.py
  /* first.py
  /* second.py

02_oop.py
23 if __name__ == '__main__':
24     employee = Employee('ลุง วิศวกร', 45, 500)
25     print(f'ชื่อบัญชี : {employee.accountname}')
26     print(f'อายุ : {employee._age}')
27     print(f'จำนวนเงิน : {employee.data}')
28
29     employee.data = 100000
30     print(f'จำนวนเงิน : {employee.data}')
```

```
ชื่อบัญชี : ลุง วิศวกร
อายุ : 45
จำนวนเงิน : 500
จำนวนเงิน : 100000
[Finished in 130ms]
```

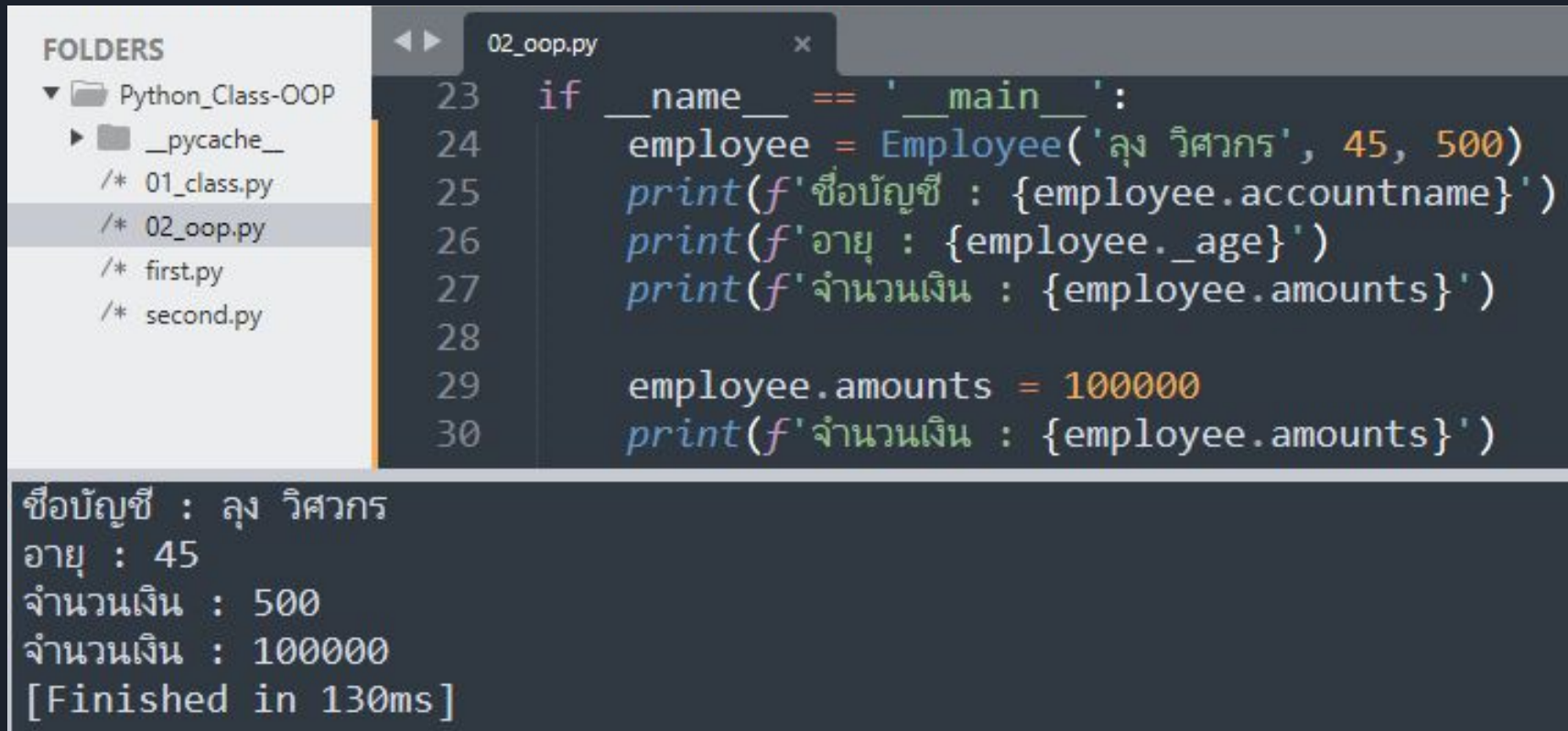

Encapsulation : Property

การเข้าถึงตัวแปร __amount โดยใช้ Decorator (@property)

```
02_oop.py x
1 class BankAccount:
2     def __init__(self, accountname, age, amount):
3         self.accountname = accountname
4         self._age = age
5         self.__amount = amount
6
7     @property                                # Getter
8     def amounts(self):
9         return self.__amount
10
11    @amounts.setter                           # Setter
12    def amounts(self, amount):
13        self.__amount = amount
14
15    def __withdraw(self, withdraw):
16        self.__amount -= withdraw
17        print(f'ถอนออก {withdraw} บาท เหลือเงิน {self.__amount} บาท')
18
19 class Employee(BankAccount):
20     def __init__(self, accountname, age, amount):
21         super().__init__(accountname, age, amount)
```

Encapsulation : Property

การเข้าถึงตัวแปร `__amount` โดยใช้ Decorator (`@property`)



The screenshot shows a Python IDE interface. On the left is a 'FOLDERS' panel with a tree view containing 'Python_Class-OOP', '__pycache__', and several Python files including '02_oop.py'. The main area is a code editor for '02_oop.py' showing Python code that creates an 'Employee' object and prints its attributes. At the bottom is a terminal window showing the execution output.

```
FOLDERS
▼ Python_Class-OOP
  ▸ __pycache__
    /* 01_class.py
    /* 02_oop.py
    /* first.py
    /* second.py

23  if __name__ == '__main__':
24      employee = Employee('ลุง วิศวกร', 45, 500)
25      print(f'ชื่อบัญชี : {employee.accountname}')
26      print(f'อายุ : {employee._age}')
27      print(f'จำนวนเงิน : {employee.amounts}')
28
29      employee.amounts = 100000
30      print(f'จำนวนเงิน : {employee.amounts}')
```

```
ชื่อบัญชี : ลุง วิศวกร
อายุ : 45
จำนวนเงิน : 500
จำนวนเงิน : 100000
[Finished in 130ms]
```


Abstract

การซ่อน (Abstract) คือคลาสแม่ (super class) ที่กำหนดการทำงานแบบคร่าวๆ ยังไม่มีการใส่รายละเอียด และให้คลาสลูก (sub class) ไปขยายรายละเอียดเพิ่มเติมให้สมบูรณ์ โดยใช้วิธีการกำหนด interface และ abstract class

Abstract : interface

ในภาษา Python ไม่มีคีย์เวิร์ด interface โดยมองว่าคลาสทุกคลาส เป็น interface อยู่แล้ว คลาสลูกสามารถใช้การสืบทอดทุกอย่างจากคลาสแม่ก็ได้ แต่โดยส่วนใหญ่แล้ว Abstract จะนิยมใช้การ implement คือให้คลาสลูก มีโครงสร้างตัวแปร และเมธอดที่เหมือนกับคลาสแม่ (interface) ทุกประการ และเมธอดในคลาสลูก ควรมีการ override เพื่อให้แตกต่างจาก interface

Abstract : interface

`super().__init__()` คือการกำหนดค่าตัวแปรที่อ้างอิงจากคลาสแม่

`super().calculate()` คือการ override เมธอด `calculate()` จากคลาสแม่

```
02_oop.py x
1  class Square:
2      def __init__(self, length):
3          self.length = length
4
5      def calculate(self):
6          return self.length * self.length
7
8  class Cube(Square):
9      def __init__(self, length):
10         super().__init__(length)
11
12     def calculate(self):
13         return super().calculate() * self.length
```

Abstract : interface

FOLDERS

▼ Python_Class-OOP

▶ `_pycache_`

`/* 01_class.py`

`/* 02_oop.py`

`/* first.py`

02_oop.py

```
15 if __name__ == '__main__':  
16     square = Square(5)  
17     print(f'พื้นที่รูปสี่เหลี่ยมจัตุรัส = {square.calculate()} ตร.ม.')  
18     cube = Cube(10)  
19     print(f'ปริมาตรลูกบาศก์ = {cube.calculate()} ลบ.ม.')
```

พื้นที่รูปสี่เหลี่ยมจัตุรัส = 25 ตร.ม.

ปริมาตรลูกบาศก์ = 1000 ลบ.ม.

[Finished in 125ms]

Abstract : abstract class

การใช้งาน Abstract ในภาษา Python จะต้อง import โมดูล ABC (Abstract Base Class) จากไลบรารี abc ซึ่งเป็น standard library ที่มีมาให้แล้วตอนติดตั้ง Python

คลาสที่เป็น Abstract จะต้อง implements คลาส ABC ในวงเล็บต่อท้ายชื่อคลาส และเมธอดที่จะกำหนดเป็น abstract จะต้อง import โมดูล abstractmethod พร้อมทั้งแอด Decorator ว่า @abstractmethod ครอบไว้ด้านบนของเมธอดด้วย

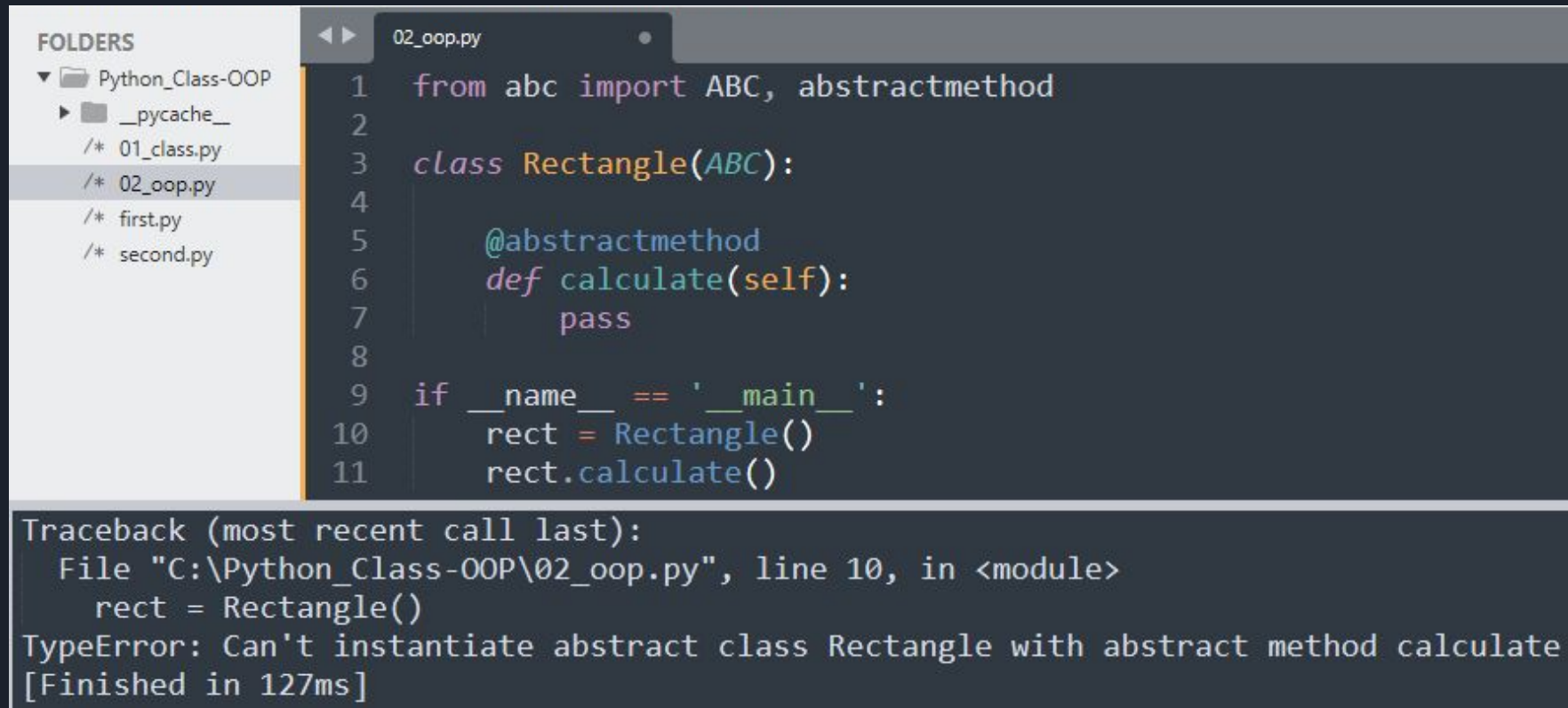
Abstract : abstract class

คลาสที่ถูกกำหนดให้เป็น abstract นั้น

- จะมีเฉพาะเมธอดที่ประกาศชื่อเอาไว้เฉยๆ ไม่มีคำสั่งการทำงานใดๆ แต่ก็สามารถมีเมธอดที่มีคำสั่งการทำงาน อยู่ใน abstract class ด้วยก็ได้
- ไม่สามารถสร้าง object เพื่อเรียกใช้งานตัวแปรหรือเมธอดได้
- คลาสลูกที่สืบทอด หรือ implement จากคลาส abstract ควรจะมีการ override เมธอดด้วย

Abstract : abstract class

error ที่เกิดขึ้น ถ้ามีการสร้าง object จาก abstract class



The screenshot shows a Python IDE with a file explorer on the left and a code editor on the right. The file explorer shows a folder named 'Python_Class-OOP' containing several files, with '02_oop.py' selected. The code editor shows the following Python code:

```
1  from abc import ABC, abstractmethod
2
3  class Rectangle(ABC):
4
5      @abstractmethod
6      def calculate(self):
7          pass
8
9  if __name__ == '__main__':
10     rect = Rectangle()
11     rect.calculate()
```

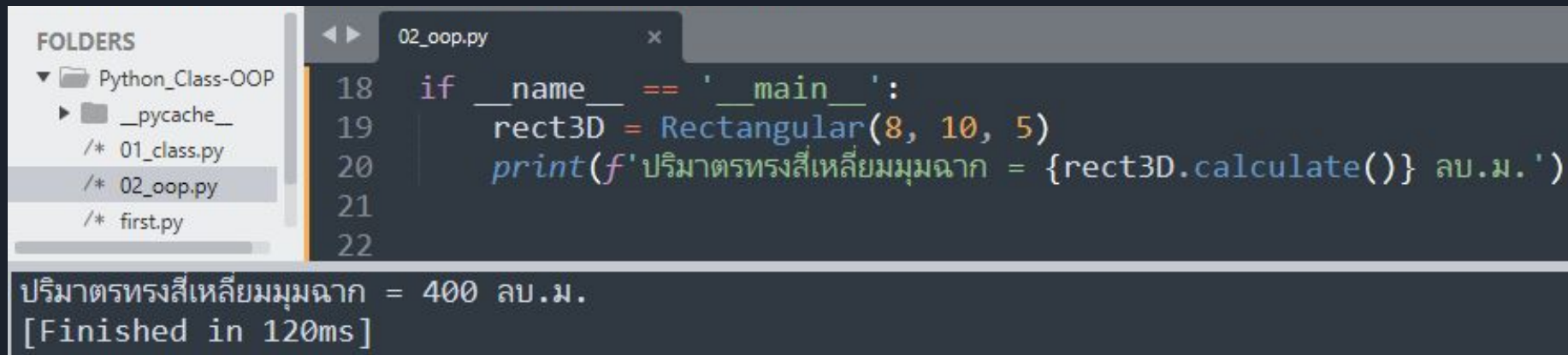
Below the code editor, a traceback is displayed, indicating an error when attempting to instantiate the abstract class:

```
Traceback (most recent call last):
  File "C:\Python_Class-OOP\02_oop.py", line 10, in <module>
    rect = Rectangle()
TypeError: Can't instantiate abstract class Rectangle with abstract method calculate
[Finished in 127ms]
```

Abstract : abstract class

```
02_oop.py x
1  from abc import ABC, abstractmethod
2
3  class Rectangle(ABC):
4
5      @abstractmethod
6      def calculate(self):
7          pass
8
9  class Rectangular(Rectangle):
10     def __init__(self, width, length, depth):
11         self.width = width
12         self.length = length
13         self.depth = depth
14
15     def calculate(self):
16         return self.width * self.length * self.depth
```


Abstract : abstract class



```
FOLDERS
▼ Python_Class-OOP
  ▸ __pycache__
    /* 01_class.py
    /* 02_oop.py
    /* first.py

18  if __name__ == '__main__':
19      rect3D = Rectangular(8, 10, 5)
20      print(f'ปริมาตรทรงสี่เหลี่ยมมุมฉาก = {rect3D.calculate()} ลบ.ม.')
21
22

ปริมาตรทรงสี่เหลี่ยมมุมฉาก = 400 ลบ.ม.
[Finished in 120ms]
```

Polymorphism

การพ้องรูป (Polymorphism) คือการที่คลาสลูก (sub class) หลายๆ คลาส มีโครงสร้างตัวแปร หรือเมธอด ที่เหมือนกับคลาสแม่ (super class) ทุกประการ แต่การทำงานของเมธอดในคลาสลูก จะแตกต่างกันออกไป

แนวคิดการพ้องรูป ให้กำหนดคลาสแม่เป็น abstract class ส่วนคลาสลูก ก็ให้ใช้การ override ตัวแปร หรือเมธอดทับลงไป

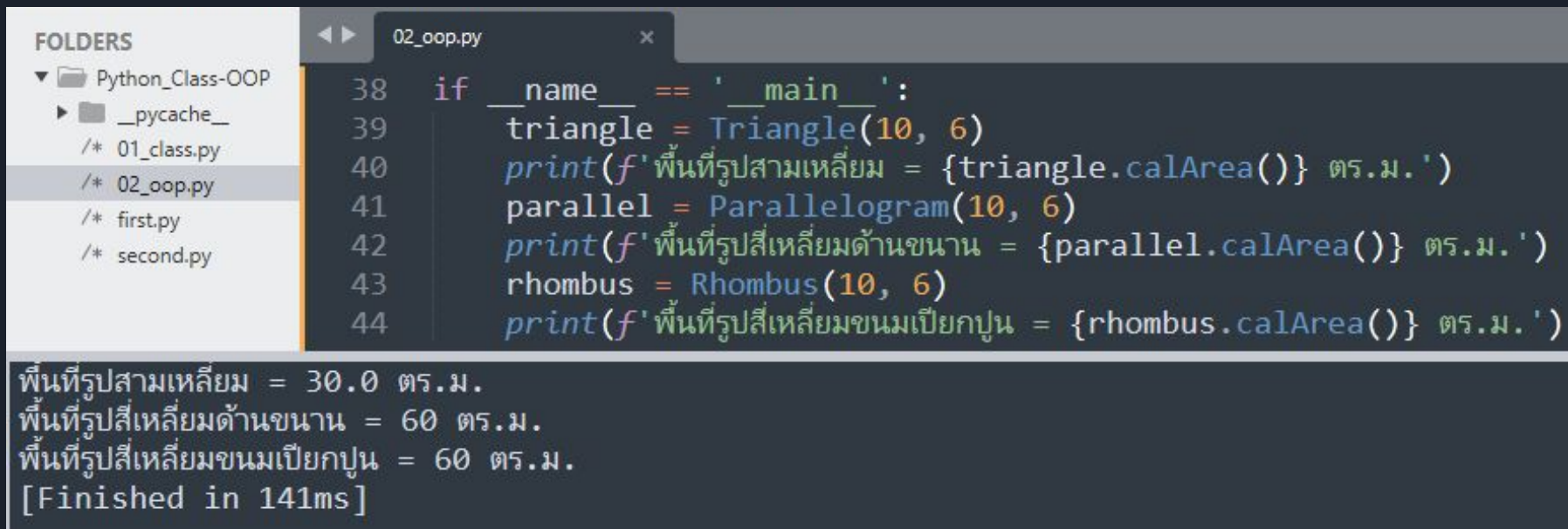
Polymorphism

กำหนดให้ Geometry เป็น abstract class และมีคลาส 3 คลาส ที่ implements จากคลาส Geometry

```
02_oop.py x
1  from abc import ABC, abstractmethod
2
3  class Geometry(ABC):
4      def __init__(self, base, height):
5          self.base = base
6          self.height = height
7
8      @abstractmethod
9      def calArea(self):
10         # คำนวณหาพื้นที่
11         pass
12
13 # สามเหลี่ยม
14 class Triangle(Geometry):
15     def __init__(self, base, height):
16         super().__init__(base, height)
17
18     def calArea(self):
19         return 0.5 * self.base * self.height
20
```

```
21 # สี่เหลี่ยมด้านขนาน
22 class Parallelogram(Geometry):
23     def __init__(self, base, height):
24         super().__init__(base, height)
25
26     def calArea(self):
27         return self.base * self.height
28
29 # สี่เหลี่ยมขนมเปียกปูน
30 class Rhombus(Geometry):
31     def __init__(self, base, height):
32         super().__init__(base, height)
33
34     def calArea(self):
35         return self.base * self.height
36
```

Polymorphism



The screenshot shows a Python IDE interface. On the left is a 'FOLDERS' panel with a tree view containing 'Python_Class-OOP', '__pycache__', and three Python files: '01_class.py', '02_oop.py' (selected), 'first.py', and 'second.py'. The main editor window is titled '02_oop.py' and contains the following Python code:

```
38 if __name__ == '__main__':
39     triangle = Triangle(10, 6)
40     print(f'พื้นที่รูปสามเหลี่ยม = {triangle.calArea()} ตร.ม.')
41     parallel = Parallelogram(10, 6)
42     print(f'พื้นที่รูปสี่เหลี่ยมด้านขนาน = {parallel.calArea()} ตร.ม.')
43     rhombus = Rhombus(10, 6)
44     print(f'พื้นที่รูปสี่เหลี่ยมขนมเปียกปูน = {rhombus.calArea()} ตร.ม.')
```

Below the code editor is a terminal window showing the output of the program:

```
พื้นที่รูปสามเหลี่ยม = 30.0 ตร.ม.
พื้นที่รูปสี่เหลี่ยมด้านขนาน = 60 ตร.ม.
พื้นที่รูปสี่เหลี่ยมขนมเปียกปูน = 60 ตร.ม.
[Finished in 141ms]
```



Uncle Engineer

ลุงวิศวกร สอนคำนวณ

THE END