ใบงาน

การเขียนโปรแกรมเชิงวัตถุ

1. จงอธิบายความแตกต่างระหว่างการเขียนโปรแกรมแบบเชิงวัตถุและแบบโครงสร้าง

พร้อมทั้งอธิบายข้อดีและข้อเสียของแต่ละรูปแบบ

ตอบ การเขียนโปรแกรมแบบเชิงวัตถุ (OOP) คือการแบ่งโปรแกรมเป็นวัตถุที่มีข้อมูลและฟังก์ชัน ทำให้โปรแกรมขยายและบำรุงรักษาง่าย แต่ซับซ้อนและใช้เวลาศึกษานาน

การเขียนโปรแกรมแบบโครงสร้าง คือการเขียนโปรแกรมเป็นลำดับขั้นตอน ใช้ฟังก์ชันและกระบวนการ เหมาะกับโปรแกรมเล็ก ๆ แต่ยากในการขยายและบำรุงรักษาเมื่อโปรแกรมโตขึ้น

- **ข้อดี OOP**: ขยายและบำรุงรักษาง่าย, นำกลับมาใช้ใหม่ได้
  ข้อเสีย OOP: ซับซ้อน, ประสิทธิภาพต่ำในบางกรณี

- **ข้อดี Structured**: เข้าใจง่าย, ประสิทธิภาพสูง
  ข้อเสีย Structured: ขยายยาก, บำรุงรักษายาก

2. จงบอกองค์ประกอบของเชิงวัตถุว่ามีอะไรบ้าง และอธิบายลักษณะของแต่ละส่วน

ตอบ 1. คลาส (Class)

ลักษณะ: คลาสเป็นแม่แบบหรือแบบแผนที่ใช้ในการสร้างอ็อบเจ็กต์ โดยกำหนดคุณสมบัติ (Attributes) และพฤติกรรม (Methods) ของอ็อบเจ็กต์

ตัวอย่าง: ในการสร้างโปรแกรมจำลองรถยนต์ Car อาจจะมีคลาส Car ที่มีคุณสมบัติ เช่น สี, ยี่ห้อ และฟังก์ชัน เช่น ขับ, เบรก

2. อ็อบเจ็กต์ (Object)

ลักษณะ: อ็อบเจ็กต์เป็นการสร้างจากคลาส ซึ่งถือเป็นตัวตนที่แท้จริงของคลาส มีคุณสมบัติและฟังก์ชันตามที่คลาสกำหนด

ตัวอย่าง: หากมีคลาส Car ก็สามารถสร้างอ็อบเจ็กต์ myCar ซึ่งมีค่าต่าง ๆ เช่น สี = แดง, ยี่ห้อ = Toyota

3. การห่อหุ้ม (Encapsulation)

ลักษณะ: การซ่อนรายละเอียดภายในคลาสและเปิดเผยเฉพาะส่วนที่จำเป็นให้กับผู้ใช้ เพื่อป้องกันการเข้าถึงข้อมูลหรือฟังก์ชันที่ไม่ควร

ตัวอย่าง: ในคลาส Car อาจจะมีตัวแปร engineStatus ที่ไม่ให้ผู้ใช้งานแก้ไขโดยตรง แต่เปิดให้ฟังก์ชัน startEngine() หรือ stopEngine() ทำงานแทน

4. การสืบทอด (Inheritance)

ลักษณะ: การสร้างคลาสใหม่จากคลาสเดิม โดยที่คลาสใหม่สามารถนำคุณสมบัติและฟังก์ชันจากคลาสเดิมมาใช้และสามารถเพิ่มคุณสมบัติหรือฟังก์ชันใหม่ได้

ตัวอย่าง: หากมีคลาส Car แล้วเราสร้างคลาส ElectricCar ที่สืบทอดจาก Car จะทำให้ ElectricCar สามารถใช้ฟังก์ชันต่าง ๆ ของ Car ได้ เช่น ขับ, เบรก และเพิ่มฟังก์ชันใหม่เช่น ชาร์จแบตเตอรี่

5. การหลากหลายรูปแบบ (Polymorphism)

ลักษณะ: การใช้ฟังก์ชันเดียวกัน แต่ทำงานได้หลากหลายรูปแบบ ขึ้นอยู่กับชนิดของอ็อบเจ็กต์ที่เรียกใช้

ตัวอย่าง: ฟังก์ชัน makeSound() ในคลาส Animal ซึ่งสามารถทำงานแตกต่างกันตามชนิดของสัตว์ เช่น dog.makeSound() อาจจะส่งเสียงเห่า, cat.makeSound() อาจจะส่งเสียงเหมียว

---

3. จงอธิบายลักษณะของการเข้าถึงคลาส พร้อมทั้งอธิบายรูปแบบต่าง ๆ ของการเข้าถึงคลาส

ตอบ การเข้าถึงคลาสใน **การเขียนโปรแกรมเชิงวัตถุ (OOP)** หมายถึง วิธีที่เราจะสามารถเข้าถึงและใช้งาน **คุณสมบัติ** และ **ฟังก์ชัน** ของคลาสจากภายนอกคลาสนั้นได้ ซึ่งสามารถแบ่งออกเป็น 3 รูปแบบหลัก ได้แก่:

1. การเข้าถึงแบบสาธารณะ (Public)

ลักษณะ: คุณสมบัติหรือฟังก์ชันที่ถูกกำหนดให้เป็น **public** สามารถเข้าถึงได้จากภายนอกคลาสโดยตรง เช่น การเข้าถึงอ็อบเจ็กต์หรือการเรียกใช้งานฟังก์ชัน

ตัวอย่าง:

```
class Car:

    def __init__(self, brand):

        self.brand = brand  # public attribute



    def start_engine(self):  # public method

        print(f"{self.brand} engine started!")
```

```
my_car = Car("Toyota")print(my_car.brand)  # เข้าถึงคุณสมบัติ public

my_car.start_engine()  # เรียกใช้ฟังก์ชัน public
```

ข้อดี: เข้าถึงได้ง่ายและสะดวก

ข้อเสีย: อาจทำให้ข้อมูลภายในคลาสถูกเปลี่ยนแปลงโดยไม่ตั้งใจ

2. การเข้าถึงแบบส่วนตัว (Private)

ลักษณะ: คุณสมบัติหรือฟังก์ชันที่ถูกกำหนดให้เป็น **private** จะไม่สามารถเข้าถึงจากภายนอกคลาสได้โดยตรง โดยจะต้องใช้ฟังก์ชันภายในคลาสเพื่อเข้าถึงข้อมูลนั้น

```
class Car:

    def __init__(self, brand):

        self.__brand = brand  # private attribute



    def __start_engine(self):  # private method

        print(f"{self.__brand} engine started!")



    def get_brand(self):

        return self.__brand



my_car = Car("Toyota")# print(my_car.__brand)  # จะเกิดข้อผิดพลาด

print(my_car.get_brand())  # ใช้ฟังก์ชันภายในเพื่อเข้าถึง
```

ข้อดี: ข้อมูลภายในคลาสจะไม่ถูกแก้ไขหรือเข้าถึงโดยตรงจากภายนอก ซึ่งช่วยให้โปรแกรมปลอดภัยมากขึ้น

ข้อเสีย: ต้องใช้ฟังก์ชันหรือเมธอดในการเข้าถึง ทำให้ต้องมีการจัดการซับซ้อนมากขึ้น

3. การเข้าถึงแบบจำกัด (Protected)

ลักษณะ: คุณสมบัติหรือฟังก์ชันที่ถูกกำหนดให้เป็น **protected** มักจะใช้ในกรณีที่ต้องการให้สามารถเข้าถึงได้จากภายในคลาสและคลาสที่สืบทอด (Subclass) เท่านั้น แต่จะไม่สามารถเข้าถึงจากภายนอกคลาสหรือโปรแกรมหลักได้

```
class Car:
    def __init__(self, brand):
        self._brand = brand  # protected attribute

    def _start_engine(self):  # protected method
        print(f"{self._brand} engine started!")
class ElectricCar(Car):
    def __init__(self, brand, battery):
        super().__init__(brand)
        self.battery = battery
```

```
    def start_electric(self):
        print(f"Charging {self._brand}...")
        self._start_engine()  # ใช้ฟังก์ชัน protected จากคลาสพ่อแม่

my_electric_car = ElectricCar("Tesla", "100kWh")
my_electric_car.start_electric()
```

ข้อดี: ช่วยให้การใช้งานภายในคลาสหรือคลาสลูกเป็นระเบียบและสามารถควบคุมการเข้าถึงได้

ข้อเสีย: ไม่สามารถป้องกันการเข้าถึงจากภายนอกคลาสได้อย่างสมบูรณ์ (ในบางภาษาอาจเข้าถึงได้อยู่ดี

---

4. จงบอกส่วนประกอบภายในคลาสว่ามีอะไรบ้าง พร้อมทั้งอธิบายลักษณะของแต่ละส่วนประกอบ

ตอบ คลาสใน **การเขียนโปรแกรมเชิงวัตถุ (OOP)** ประกอบด้วยส่วนต่าง ๆ ที่ช่วยให้การออกแบบโปรแกรมเป็นระเบียบและสามารถทำงานได้ตามต้องการ ส่วนประกอบหลักภายในคลาสมีดังนี้:

1. ตัวแปร (Attributes or Fields)

ลักษณะ: ตัวแปรที่เก็บข้อมูลหรือคุณสมบัติของคลาส ตัวแปรเหล่านี้จะใช้ในการบ่งบอกสถานะหรือคุณลักษณะของอ็อบเจ็กต์ที่สร้างจากคลาสนั้น

```
class Car:
    def __init__(self, brand, color):
        self.brand = brand  # ตัวแปร attributes
        self.color = color  # ตัวแปร attributes
```

ประเภท:

Public: สามารถเข้าถึงจากภายนอกคลาส

Private: เข้าถึงได้เฉพาะภายในคลาส

Protected: สามารถเข้าถึงได้จากภายในคลาสและคลาสที่สืบทอด

2. เมธอด (Methods or Functions)

ลักษณะ: เมธอดคือฟังก์ชันที่ถูกกำหนดภายในคลาส ซึ่งใช้เพื่อดำเนินการหรือเปลี่ยนแปลงสถานะของอ็อบเจ็กต์ที่ถูกสร้างจากคลาส

```
class Car:

    def __init__(self, brand, color):

        self.brand = brand

        self.color = color



    def start_engine(self):  # เมธอด

        print(f"{self.brand} engine started.")
```

ประเภท:

Public: สามารถเรียกใช้จากภายนอกคลาส

Private: ใช้ภายในคลาสเท่านั้น

Protected: ใช้ภายในคลาสและคลาสที่สืบทอด

3. คอนสตรักเตอร์ (Constructor)

ลักษณะ: คอนสตรักเตอร์คือเมธอดพิเศษที่ถูกเรียกใช้เมื่อมีการสร้างอ็อบเจ็กต์จากคลาส โดยมักใช้เพื่อกำหนดค่าเริ่มต้นให้กับตัวแปรภายในคลาส

```
class Car:

    def __init__(self, brand, color):  # คอนสตรักเตอร์

        self.brand = brand

        self.color = color
```

ความสำคัญ: คอนสตรักเตอร์ช่วยให้โปรแกรมเมอร์สามารถกำหนดค่าเริ่มต้นให้กับอ็อบเจ็กต์ในตอนที่อ็อบเจ็กต์ถูกสร้างขึ้น

\

4. เดสตรักเตอร์ (Destructor)

ลักษณะ: เดสตรักเตอร์คือเมธอดพิเศษที่ถูกเรียกเมื่ออ็อบเจ็กต์ของคลาสถูกทำลาย หรือเมื่อไม่ใช้งานแล้ว

```
class Car:
    def __init__(self, brand):
        self.brand = brand
    def __del__(self):  # เดสตรักเตอร์
        print(f"{self.brand} object is being deleted.")
```

การใช้งาน: แม้ว่าจะไม่ใช่ฟีเจอร์ที่พบได้บ่อยในการเขียนโปรแกรม OOP แต่เดสตรักเตอร์มีประโยชน์ในการจัดการกับการปิดการเชื่อมต่อหรือการทำความสะอาดข้อมูลที่เกี่ยวข้องกับอ็อบเจ็กต์

5. โปรเพอร์ตี้ (Property)

ลักษณะ: โปรเพอร์ตี้เป็นการใช้ตัวแปรภายในคลาสให้สามารถเข้าถึงได้เหมือนฟังก์ชัน แต่สามารถควบคุมการเข้าถึงหรือการตั้งค่าได้

```
class Car:
    def __init__(self, brand, color):
        self._brand = brand
        self._color = color
    @property
    def brand(self):  # ใช้โปรเพอร์ตี้เพื่อเข้าถึงตัวแปร
        return self._brand
```

```
    @brand.setter
    def brand(self, value):  # ใช้โปรเพอร์ตี้ setter เพื่อตั้งค่าตัวแปร
        self._brand = value
```

ประโยชน์: ทำให้สามารถควบคุมการเข้าถึงข้อมูลและให้ความยืดหยุ่นในการตั้งค่าหรือเปลี่ยนแปลงข้อมูลได้อย่างปลอดภัย

6. คำสั่ง Static

ลักษณะ: เมธอดและตัวแปรแบบ static จะไม่เกี่ยวข้องกับอ็อบเจ็กต์ของคลาส แต่จะเชื่อมโยงกับคลาสเอง ซึ่งทำให้สามารถเรียกใช้เมธอดหรือเข้าถึงตัวแปรนั้น ๆ ได้โดยไม่ต้องสร้างอ็อบเจ็กต์

```
class Car:
    count = 0  # ตัวแปร static

    def __init__(self, brand):
        self.brand = brand
        Car.count += 1

    @staticmethod
    def car_count():
        print(f"Total cars: {Car.count}")
```

ประโยชน์: ใช้ในกรณีที่ต้องการฟังก์ชันหรือข้อมูลที่ไม่ขึ้นอยู่กับอ็อบเจ็กต์ แต่ยังคงเป็นส่วนหนึ่งของคลาส

5. จงอธิบายและเรียงลำดับขั้นตอนการเขียนโปรแกรมเชิงวัตถุ

พร้อมทั้งเขียนโปรแกรมประกอบการอธิบาย

ตอบ การเขียนโปรแกรมเชิงวัตถุ (OOP) มีขั้นตอนหลักดังนี้:

1.วิเคราะห์ปัญหา: ทำความเข้าใจปัญหาที่จะพัฒนา

2.ออกแบบคลาสและอ็อบเจ็กต์: ระบุวัตถุและคุณสมบัติของมัน เช่น Customer, Product

3.กำหนดคุณสมบัติและฟังก์ชัน: ระบุข้อมูลและฟังก์ชันที่จะใช้ในคลาส

4.สร้างอ็อบเจ็กต์: สร้างอ็อบเจ็กต์จากคลาสที่ออกแบบ

5.ทดสอบและปรับปรุง: ทดสอบโปรแกรมและปรับปรุงให้ทำงานได้ดี

ตัวอย่างโปรแกรม: สร้างคลาส Customer และเพิ่มเมธอดเพื่อบันทึกและแสดงข้อมูลลูกค้า.

```
class Customer:
    def __init__(self, name, address):
        self.name = name
        self.address = address
        self.orders = []

    def add_order(self, product, quantity):
        self.orders.append({"product": product, "quantity": quantity})
```

```
    def show_info(self):
        print(f"Name: {self.name}, Address: {self.address}")
        for order in self.orders:
            print(f"{order['product']} x{order['quantity']}")
# สร้างอ็อบเจ็กต์และใช้งาน
customer1 = Customer("John", "123 Street")
customer1.add_order("Laptop", 1)
customer1.show_info()
```

---

6. จงอธิบายการทำงานและความแตกต่างระหว่างคอนสตรัคเตอร์และดีสตรัคเตอร์

พร้อมทั้งเขียนโปรแกรมประกอบการอธิบาย

ตอบ คอนสตรัคเตอร์ (Constructor) และ ดีสตรัคเตอร์ (Destructor) เป็นฟังก์ชันพิเศษที่ใช้ในคลาสเพื่อจัดการกับการสร้างและทำลายอ็อบเจ็กต์ในโปรแกรมเชิงวัตถุ (OOP) โดยมีลักษณะการทำงานและความแตกต่างดังนี้:

1. คอนสตรัคเตอร์ (Constructor):

การทำงาน: คอนสตรัคเตอร์คือฟังก์ชันพิเศษที่ถูกเรียกใช้งานอัตโนมัติเมื่อมีการสร้างอ็อบเจ็กต์จากคลาส โดยมักจะใช้ในการกำหนดค่าตัวแปรเริ่มต้นให้กับอ็อบเจ็กต์

รูปแบบ: คอนสตรัคเตอร์จะมีชื่อเดียวกับคลาสในหลายภาษา (เช่น Python ใช้ __init__), และจะไม่มีค่า return

ตัวอย่างการใช้งาน: เมื่อสร้างอ็อบเจ็กต์ใหม่ คอนสตรัคเตอร์จะถูกเรียกใช้อัตโนมัติ

2. ดีสตรัคเตอร์ (Destructor):

การทำงาน: ดีสตรัคเตอร์คือฟังก์ชันพิเศษที่ถูกเรียกใช้เมื่ออ็อบเจ็กต์ถูกทำลาย หรือเมื่อไม่จำเป็นต้องใช้งานอีกต่อไป ดีสตรัคเตอร์มักใช้ในการทำความสะอาด เช่น ปิดการเชื่อมต่อฐานข้อมูลหรือจัดการกับทรัพยากรที่อาจจะหลงเหลือ

รูปแบบ: ดีสตรัคเตอร์จะมีชื่อเดียวกับคลาสในหลายภาษา (ใน Python ใช้ __del__) และจะไม่มีค่า return

ความแตกต่างระหว่างคอนสตรัคเตอร์และดีสตรัคเตอร์:

คุณสมบัติ	คอนสตรัคเตอร์ (Constructor)	ดีสตรัคเตอร์ (Destructor)
การทำงาน	ถูกเรียกเมื่อสร้างอ็อบเจ็กต์	ถูกเรียกเมื่ออ็อบเจ็กต์ถูกทำลาย
ชื่อ	ชื่อเดียวกับคลาส (ใน Python __init__)	ชื่อเดียวกับคลาส (ใน Python __del__)
การใช้งาน	ใช้สำหรับกำหนดค่าเริ่มต้นให้กับอ็อบเจ็กต์	ใช้สำหรับทำความสะอาดหรือการจัดการเมื่ออ็อบเจ็กต์ไม่ใช้งาน
ค่า Return	ไม่มีการคืนค่า (ไม่มี return)	ไม่มีการคืนค่า (ไม่มี return)

ตัวอย่างโปรแกรมใน Python:

```
class Car:

    def __init__(self, brand, model):
```

```
        # คอนสตรัคเตอร์
        self.brand = brand  # กำหนดค่าเริ่มต้นให้กับ brand
        self.model = model  # กำหนดค่าเริ่มต้นให้กับ model
        print(f"{self.brand} {self.model} car created.")
    def __del__(self):
        # ดีสตรัคเตอร์
        print(f"{self.brand} {self.model} car destroyed.")
# สร้างอ็อบเจ็กต์ใหม่
car1 = Car("Toyota", "Corolla")
# ลบอ็อบเจ็กต์เพื่อให้ดีสตรัคเตอร์ทำงานdel car1
```

คำอธิบายโปรแกรม:

คอนสตรัคเตอร์ (__init__): เมื่อสร้างอ็อบเจ็กต์ car1 จากคลาส Car, คอนสตรัคเตอร์จะถูกเรียกเพื่อกำหนดค่าเริ่มต้นให้กับคุณสมบัติ brand และ model และแสดงข้อความว่า "Toyota Corolla car created."

ดีสตรัคเตอร์ (__del__): เมื่อเราใช้ del car1 เพื่อทำลายอ็อบเจ็กต์ car1, ดีสตรัคเตอร์จะถูกเรียกเพื่อทำความสะอาดและแสดงข้อความว่า "Toyota Corolla car destroyed."

ผลลัพธ์ที่ได้:

```
Toyota Corolla car created.
Toyota Corolla car destroyed.
```

7. จงอธิบายความหมายของพารามิเตอร์และวิธีการส่งค่าพารามิเตอร์

พร้อมทั้งเขียนโปรแกรมประกอบการอธิบาย

ตอบ พารามิเตอร์ (Parameter)

พารามิเตอร์ในโปรแกรมมิ่งคือ **ตัวแปร** ที่ใช้ในฟังก์ชันหรือเมธอดเพื่อรับข้อมูล (ค่า) ที่ถูกส่งเข้ามาจากภายนอกฟังก์ชัน เมื่อฟังก์ชันถูกเรียกใช้งาน พารามิเตอร์จะรับค่าจาก **อาร์กิวเมนต์ (Argument)** ซึ่งเป็นข้อมูลที่ถูกส่งเข้าไปยังฟังก์ชันนั้น ๆ

การส่งค่าพารามิเตอร์

การส่งค่าพารามิเตอร์ในฟังก์ชันมี 2 วิธีหลัก:

การส่งค่าโดยตรง (Pass by Value):

ค่าที่ถูกส่งเข้าไปจะถูกคัดลอกไปยังพารามิเตอร์ในฟังก์ชัน ทำให้ตัวแปรภายในฟังก์ชันไม่สามารถแก้ไขตัวแปรภายนอกได้

ใช้ในภาษาเช่น C, Java (สำหรับประเภทข้อมูลพื้นฐาน)

การส่งค่าโดยการอ้างอิง (Pass by Reference):

ตัวแปรในฟังก์ชันจะอ้างอิงถึงตัวแปรภายนอก ทำให้สามารถแก้ไขค่าของตัวแปรภายนอกได้

ใช้ในภาษาเช่น Python (สำหรับชนิดข้อมูลที่สามารถเปลี่ยนแปลงได้ เช่น ลิสต์, ดิกชันนารี)

ตัวอย่างโปรแกรม:

1. การส่งค่าพารามิเตอร์โดยการส่งค่า (Pass by Value)

```
def add_numbers(a, b):
    sum_result = a + b
    print(f"The sum of {a} and {b} is {sum_result}")
# การเรียกฟังก์ชันและส่งค่าเข้าไป
x = 10
y = 5
add_numbers(x, y)  # ส่งค่า x และ y ไปที่ฟังก์ชัน
# ผลลัพธ์:# The sum of 10 and 5 is 15
```

ฟังก์ชัน add_numbers รับพารามิเตอร์ a และ b ซึ่งเป็นค่าที่ส่งเข้าไปจากตัวแปร x และ y

ค่าที่ถูกส่งเข้าไปจะถูกคัดลอกไปยัง a และ b และไม่มีการเปลี่ยนแปลงค่าของ x และ y ภายนอกฟังก์ชัน

2. การส่งค่าพารามิเตอร์โดยการอ้างอิง (Pass by Reference)

ใน Python การส่งค่าโดยการอ้างอิงจะเกิดขึ้นกับประเภทข้อมูลที่สามารถเปลี่ยนแปลงได้ (Mutable types) เช่น **ลิสต์**, **ดิกชันนารี**, หรือ **เซต**:

```
def modify_list(lst):
    lst.append(100)  # เพิ่มค่าเข้าไปในลิสต์
```

my_list = [1, 2, 3]print(f"Before modification: {my_list}")

modify_list(my_list) # ส่งอ้างอิงลิสต์ไปยังฟังก์ชัน

print(f"After modification: {my_list}")# ผลลัพธ์:# Before modification: [1, 2, 3]# After modification: [1, 2, 3, 100]

ฟังก์ชัน modify_list รับพารามิเตอร์ lst ซึ่งเป็นลิสต์

เมื่อส่งลิสต์ my_list เข้าไปในฟังก์ชัน Python จะส่งอ้างอิงไปยังลิสต์นี้ ดังนั้นเมื่อเราทำการเพิ่มค่า 100 เข้าไปใน lst ลิสต์ my_list ภายนอกฟังก์ชันจะถูกเปลี่ยนแปลงด้วย

3. การส่งค่าพารามิเตอร์ด้วยการใช้ค่าเริ่มต้น (Default Parameters)

เราสามารถตั้งค่าพารามิเตอร์ให้มีค่าเริ่มต้นได้ในกรณีที่ไม่ได้ส่งค่ามาให้:

```
def greet(name, greeting="Hello"):
    print(f"{greeting}, {name}!")


greet("Alice")              # ใช้ค่า greeting เริ่มต้น
greet("Bob", "Goodbye")     # ส่งค่า greeting ใหม่# ผลลัพธ์:# Hello, Alice!# Goodbye, Bob!
```

ฟังก์ชัน greet มีพารามิเตอร์ greeting ที่มีค่าเริ่มต้นเป็น "Hello"

หากไม่ส่งค่า greeting มา จะใช้ค่าเริ่มต้น "Hello"

หากส่งค่าใหม่เข้าไปเช่น "Goodbye", ค่าพารามิเตอร์ greeting จะถูกแทนที่ด้วยค่าที่ส่งเข้าไป

8. จงอธิบายความหมายของการสืบทอดคลาส พร้อมทั้งเขียนโปรแกรมประกอบการอธิบาย

ตอบ การสืบทอดคลาส (Inheritance)

การสืบทอดคลาส (Inheritance) คือแนวคิดในโปรแกรมเชิงวัตถุ (OOP) ที่อนุญาตให้ **คลาสใหม่** (เรียกว่า **คลาสลูก** หรือ **subclass**) สามารถสืบทอดคุณสมบัติ (Attributes) และพฤติกรรม (Methods) จาก **คลาสเก่า** (เรียกว่า **คลาสแม่** หรือ **superclass**) ได้ ทำให้สามารถนำคุณสมบัติหรือฟังก์ชันของคลาสแม่มาใช้ในคลาสลูกได้โดยไม่ต้องเขียนโค้ดซ้ำ

การสืบทอดช่วยให้เราสามารถขยายฟังก์ชันการทำงานของโปรแกรมได้ง่ายขึ้น โดยการเพิ่มฟังก์ชันหรือคุณสมบัติใหม่ในคลาสลูก โดยไม่กระทบกับคลาสแม่

ข้อดีของการสืบทอด

การนำกลับมาใช้ใหม่ (Reusability): ใช้ฟังก์ชันและคุณสมบัติจากคลาสแม่ได้โดยไม่ต้องเขียนโค้ดซ้ำ

การขยายโปรแกรม (Extensibility): สามารถเพิ่มฟังก์ชันใหม่ ๆ ในคลาสลูกได้ง่าย

การปรับปรุงโค้ด (Maintainability): ลดความซับซ้อนและทำให้โปรแกรมดูแลรักษาง่ายขึ้น

ลักษณะของการสืบทอด:

คลาสแม่ (Superclass): คลาสที่มีคุณสมบัติและฟังก์ชันที่สามารถสืบทอด

คลาสลูก (Subclass): คลาสที่สืบทอดคุณสมบัติและฟังก์ชันจากคลาสแม

ตัวอย่างโปรแกรมการสืบทอดคลาส:

```
# คลาสแม่ (Superclass)class Animal:

    def __init__(self, name):

        self.name = name  # คุณสมบัติของคลาสแม่
```

```
    def speak(self):
        print(f"{self.name} makes a sound")
# คลาสลูก (Subclass) ที่สืบทอดจาก Animalclass Dog(Animal):
    def __init__(self, name, breed):
        # เรียกคอนสตรัคเตอร์ของคลาสแม่
        super().__init__(name)
        self.breed = breed  # เพิ่มคุณสมบัติใหม่ในคลาสลูก

    def speak(self):
        print(f"{self.name} barks")  # กำหนดพฤติกรรมใหม่ในคลาสลูก
# คลาสลูก (Subclass) ที่สืบทอดจาก Animalclass Cat(Animal):
    def __init__(self, name, color):
        # เรียกคอนสตรัคเตอร์ของคลาสแม่
        super().__init__(name)
        self.color = color  # เพิ่มคุณสมบัติใหม่ในคลาสลูก

    def speak(self):
        print(f"{self.name} meows")  # กำหนดพฤติกรรมใหม่ในคลาสลูก
```

สร้างอ็อบเจ็กต์จากคลาสลูก

dog = Dog("Buddy", "Golden Retriever")

cat = Cat("Whiskers", "Grey")

เรียกใช้งานเมธอดจากคลาสแม่และคลาสลูก

dog.speak() # Buddy barks

cat.speak() # Whiskers meows

คลาส Animal:

เป็นคลาสแม่ที่มีฟังก์ชัน speak() ซึ่งทำหน้าที่แสดงข้อความว่า "ทำเสียง" โดยใช้ค่า name ที่ถูกกำหนดในคอนสตรัคเตอร์

คลาส Dog **และ** Cat:

คลาส Dog สืบทอดจาก Animal และกำหนดฟังก์ชัน speak() ใหม่ให้สุนัขเห่า

คลาส Cat สืบทอดจาก Animal และกำหนดฟังก์ชัน speak() ใหม่ให้แมวเหมียว

ในการสร้างอ็อบเจ็กต์จาก Dog และ Cat, ฟังก์ชัน speak() จะแตกต่างกันไปตามชนิดของสัตว์

การใช้ super():

ฟังก์ชัน super().__init__(name) ถูกใช้ในคลาสลูกเพื่อเรียกคอนสตรัคเตอร์ของคลาสแม่ (Animal) เพื่อให้ได้ค่าของ name ที่ถูกกำหนดใน Animal

ผลลัพธ์ที่ได้:

Buddy barks

Whiskers meows

9. จงอธิบายลักษณะของโอเวอร์ไรด์ (Overriding) และโอเวอร์โหลด (Overloading)

พร้อมทั้งเขียนโปรแกรมประกอบการอธิบาย

ตอบ โอเวอร์ไรด์ (Overriding) และ โอเวอร์โหลด (Overloading)

โอเวอร์ไรด์ (Overriding) และ **โอเวอร์โหลด (Overloading)** เป็นแนวคิดที่เกี่ยวข้องกับการใช้งานฟังก์ชันหรือเมธอดในโปรแกรมเชิงวัตถุ (OOP) แต่มีความแตกต่างในลักษณะการใช้งานและวิธีการทำงาน ดังนี้:

---

1. โอเวอร์ไรด์ (Overriding):

การทำงาน: โอเวอร์ไรด์เกิดขึ้นเมื่อคลาสลูก (subclass) ปรับปรุงหรือแทนที่ฟังก์ชันหรือเมธอดที่มีในคลาสแม่ (superclass) ซึ่งในกรณีนี้จะใช้ฟังก์ชันหรือเมธอดที่มีชื่อเดียวกันในคลาสแม่และคลาสลูก แต่ทำงานได้แตกต่างกันไป

จุดประสงค์: ทำให้คลาสลูกสามารถเปลี่ยนแปลงพฤติกรรมของฟังก์ชันในคลาสแม่ให้เหมาะสมกับคลาสลูก

ข้อควรระวัง: ฟังก์ชันที่โอเวอร์ไรด์จะต้องมีชื่อและพารามิเตอร์เหมือนกับฟังก์ชันในคลาสแม่

ตัวอย่างโปรแกรมของโอเวอร์ไรด์ (Overriding):

```
class Animal:
    def speak(self):
        print("Animal makes a sound")
class Dog(Animal):
    def speak(self):  # การโอเวอร์ไรด์
        print("Dog barks")
```

```
class Cat(Animal):
    def speak(self):  # การโอเวอร์ไรด์
        print("Cat meows")
# สร้างอ็อบเจ็กต์จากคลาสลูก
dog = Dog()
cat = Cat()
# เรียกใช้เมธอดที่โอเวอร์ไรด์
dog.speak()  # Dog barks
cat.speak()  # Cat meows
```

คลาส Dog และ Cat มีการโอเวอร์ไรด์เมธอด speak() จากคลาส Animal ทำให้เมธอด speak() ในคลาสลูกทำงานต่างจากในคลาสแม่

ผลลัพธ์:

Dog barks

Cat meows

---

2. โอเวอร์โหลด (Overloading):

การทำงาน: โอเวอร์โหลดเกิดขึ้นเมื่อคลาสมีฟังก์ชันหรือเมธอดหลายตัวที่มีชื่อเดียวกัน แต่พารามิเตอร์ต่างกัน (จำนวนหรือชนิดของพารามิเตอร์)

จุดประสงค์: การโอเวอร์โหลดช่วยให้สามารถใช้ชื่อเมธอดเดียวกันในการทำงานหลายๆ แบบตามพารามิเตอร์ที่ถูกส่งมา

ข้อควรระวัง: ในภาษา Python การโอเวอร์โหลดฟังก์ชันจะไม่ทำงานเหมือนภาษาอื่น ๆ (เช่น C++, Java) เพราะ Python ไม่รองรับการโอเวอร์โหลดในลักษณะนี้โดยตรง เราสามารถจำลองการโอเวอร์โหลดได้โดยการใช้ค่าเริ่มต้นของพารามิเตอร์หรือการตรวจสอบชนิดข้อมูลในฟังก์ชัน

ตัวอย่างโปรแกรมของโอเวอร์โหลด (Overloading):

```
class MathOperations:
    def add(self, a, b=None):
        if b is None:
            return a + a  # ถ้าไม่ส่ง b จะบวก a กับ a
        else:
            return a + b  # ถ้าส่ง b จะบวก a กับ b
# สร้างอ็อบเจ็กต์จากคลาส MathOperations
math_op = MathOperations()
# เรียกใช้เมธอด add ที่โอเวอร์โหลดprint(math_op.add(5))  # 10 (บวก 5 + 5)print(math_op.add(5, 3))
# 8 (บวก 5 + 3)
```

เมธอด add() ในคลาส MathOperations ใช้การโอเวอร์โหลดโดยการใช้พารามิเตอร์ b ที่มีค่าเริ่มต้นเป็น None

หากไม่ส่งค่า b เข้ามา, ฟังก์ชันจะทำการบวก a + a

หากส่งค่า b เข้ามา, ฟังก์ชันจะทำการบวก a + b

ผลลัพธ์

10

---

12. จงยกตัวอย่างโปรแกรมเชิงวัตถุที่มีองค์ประกอบ ได้แก่ คลาส คุณสมบัติ พฤติกรรม และวัตถุ โดยประยุกต์ให้สอดคล้องกับการใช้งานจริงในปัจจุบัน

ตอบ โปรแกรมนี้จำลองระบบ **บัญชีธนาคาร** โดยใช้การเขียนโปรแกรมเชิงวัตถุ (OOP) ซึ่งมีองค์ประกอบหลัก ๆ

คลาส (BankAccount**)**:

เป็นแบบแผนสำหรับสร้างบัญชีธนาคาร

คุณสมบัติ:

account_holder: ชื่อเจ้าของบัญชี

balance: ยอดเงินในบัญชี

พฤติกรรม:

deposit(): ฝากเงิน

withdraw(): ถอนเงิน

get_balance(): ตรวจสอบยอดเงิน

วัตถุ:

customer1_account และ customer2_account: คือบัญชีของลูกค้าแต่ละคนที่ใช้บริการ

ตัวอย่างการทำงาน:

สมชายมีบัญชีเริ่มต้น 1000 บาท, เขาฝากเงิน 500 บาทและถอน 200 บาท

สุนารีมีบัญชีเริ่มต้นที่ 0 บาท, เขาฝากเงิน 2000 บาท

ผลลัพธ์:

สมชาย: 1300 บาทสุนารี: 2000 บาท