ตอนที่ 1
การเขียนโปรแกรมเชิงวัตถุ

การเขียนโปรแกรมคอมพิวเตอร์แบบมีโครงสร้าง มีลักษณะแยกส่วนการทำงานของโปรแกรมในการจัดการข้อมูล โดยใช้รูปแบบของฟังก์ชันหรือโปรแกรมย่อย การทำงานของฟังก์ชันยังมีข้อเสียสำหรับการนำไปใช้ในการพัฒนาโปรแกรมที่มีความซับซ้อนสูงขึ้น เมื่อนำไปใช้ทำให้โปรแกรมนั้นมีความซับซ้อนตามไปด้วย จึงเกิดปัญหาเรื่องของการนำมาพัฒนา ปรับปรุง เปลี่ยนแปลงและแก้ไข ซึ่งจะมีความยุ่งยากมาก ดังนั้นจึงได้มีการพัฒนาแนวคิด ที่เรียกว่า “การเขียนโปรแกรมเชิงวัตถุ” แก้ปัญหาและสนับสนุนการพัฒนาโปรแกรมที่มีความซับซ้อน จึงทำให้มีความนิยมนำมาใช้พัฒนาจำนวนมากในปัจจุบัน การเขียนโปรแกรมเชิงวัตถุมีรายละเอียดดังต่อไปนี้

ความรู้พื้นฐานของการเขียนโปรแกรมเชิงวัตถุ

การสร้างโปรแกรมย่อยหรือฟังก์ชันจึงขึ้นตรงต่อโครงสร้างข้อมูล เมื่อมีการเปลี่ยนแปลงโครงสร้างของข้อมูล จึงต้องเปลี่ยนการทำงานในโปรแกรมไปด้วย ในบางครั้งอาจต้องกลับมาเขียนโปรแกรมขึ้นมาใหม่ จะเห็นได้จากระบบใหญ่ที่มีความซับซ้อนมาก วิธีการเขียนโปรแกรมแบบมีโครงสร้าง จึงไม่เหมาะสมสำหรับการสร้างแอพพลิเคชั่นที่ไม่ใหญ่มาก (ธีรวัฒน์ ประกอบผล, 2558)

การเขียนโปรแกรมเชิงวัตถุ (Object Oriented Programming) หมายถึง การมองการเขียนโปรแกรมเป็นลักษณะของวัตถุมากกว่าการกระทำ (Weisfeld, 2013) วัตถุบนโลกของการเขียนโปรแกรมมีหน้าที่ความรับผิดชอบเป็นของตนเอง (ศุภชัย สมพานิช, 2559) และการเขียนโปรแกรมเชิงวัตถุจะแยกปัญหาหรือระบบงานออกเป็นส่วน เพื่อลดความซับซ้อนให้น้อยลง (สุชาติ คุ้มมณี, 2558) เชิงวัตถุเป็นวิธีการใหม่ในการเขียนโปรแกรม โดยมุ่งที่จะเพิ่มประสิทธิภาพในการสร้างโปรแกรม ทำให้การพัฒนาโปรแกรมด้วยวิธีนี้ใช้เวลาสั้นลงและต้นทุนต่ำลง โดยการนำชุดคำสั่งที่สร้างไปแล้วกลับมาใช้งานใหม่ได้ และยังหาจุดที่ต้องการแก้ไขในโปรแกรมได้ง่ายขึ้น เป็นวิธีการเขียนโปรแกรมที่นำแนวคิดในความเป็นจริงที่อยู่รอบตัว มาใช้กับการเขียนโปรแกรม เช่น เปรียบเทียบวิธีการที่ผู้เขียนโปรแกรมกับช่างยนต์ ประกอบรถยนต์ขึ้นมาหนึ่งคัน สิ่งที่ต้องการคือ เครื่องยนต์ ถังน้ำมัน ล้อรถ โครงสร้างรถ เป็นต้น เมื่อได้ชิ้นส่วนที่ต้องการแล้ว ก็นำมาประกอบเข้าด้วยกัน จะเห็นว่าช่างยนต์ไม่จำเป็นต้องสร้างเครื่องยนต์ และอุปกรณ์ต่าง ๆ ด้วยตัวช่างยนต์เอง แต่จะเป็นผู้สร้างที่มีความเข้าใจส่วนประกอบ ทำงานอย่างไรและจะนำเอาไปใช้ร่วมกับชิ้นส่วนอื่นได้อย่างไร ช่างยนต์สามารถประกอบรถยนต์ขึ้นมาด้วยความชำนาญของตนเอง เช่นเดียวกันกับการสร้าง

โปรแกรมคอมพิวเตอร์แบบเชิงวัตถุ คือ การมองโปรแกรมว่าเป็นส่วนประกอบด้วยชิ้นส่วนต่าง ๆ ที่นักเขียนโปรแกรมนำมาประกอบกัน เช่น เมื่อต้องการเขียนโปรแกรมบนวินโดวน์ อุปกรณ์พื้นฐานที่ต้องการ เช่น Label Button Textbox Form เป็นต้น สามารถประกอบกันเป็นโปรแกรมได้ โดยไม่จำเป็นต้องทราบว่าอุปกรณ์พื้นฐานนั้นสร้างขึ้นมาได้อย่างไร เพียงแต่ต้องศึกษาให้เข้าใจว่าจะใช้ชิ้นส่วนใด ใช้อย่างไร ก็สามารถสร้างโปรแกรมขึ้นมาได้ โดยการนำวัตถุมาประกอบกัน เหมือนกับช่างยนต์นำเอาชิ้นส่วนมาประกอบเข้าด้วยกัน (ธีรวัฒน์ ประกอบผล, 2558)

ตารางที่ 1 ข้อดีและข้อเสียของการเขียนโปรแกรมเชิงวัตถุ

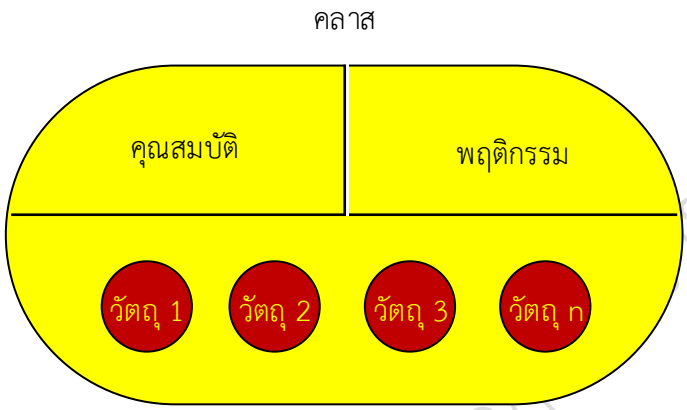
ข้อดี	ข้อเสีย
1. เข้าง่าย โดยการมองความจริงในรอบตัวที่เป็นวัตถุและมีความหมายภายในตัวเอง 2. สามารถปรับปรุงแก้ไขโปรแกรมได้ง่าย และส่งผลกระทบต่อโปรแกรมน้อย เนื่องจากแยกส่วนการทำงานกัน 3. มีรูปแบบการตรวจสอบข้อมูลผิดของโปรแกรม จึงมีความถูกต้อง และปลอดภัยสูง 4. มีความปลอดภัยในขบวนการเข้าถึงข้อมูล (Information Hidden) ที่ซ้อนกันของข้อมูลสูง 5. มีกระบวนการของการสืบทอด ทำให้สะดวกในการเขียนโปรแกรม 6. ลดขั้นตอนการเขียนโปรแกรม เนื่องจากมีคุณสมบัติของการนำกลับมาใช้ใหม่ (Reusability) 7. เนื่องด้วยโปรแกรมมีคุณภาพสูง จึงสามารถนำไปใช้ได้หลายแพลตฟอร์ม (Platform)	1. การเรียนรู้ขั้นตอนอาจยากต่อการทำงานความเข้าใจ อันเนื่องยังยึดติดกับแนวคิดการเขียนโปรแกรมแบบโครงสร้าง 2. การทำงานเนื่องจากมีความซับซ้อนจึงได้ให้การทำงานนั้นช้ากว่าการเขียนโปรแกรมแบบโครงสร้าง 3. คำจำกัดความในเชิงวัตถุมีค่อนข้างมากและมีความสัมพันธ์ที่ซ้อนกัน จึงอาจทำให้เกิดความกำกวมในการเข้าใจ

ที่มา : สุชาติ คุ้มมณี (2558)

การเขียนโปรแกรมเชิงวัตถุ จะพบว่า มีความแตกต่างจากมุมมองการทำงานของฟังก์ชัน คุณสมบัติที่มีในรูปแบบการเขียนโปรแกรมเชิงวัตถุ จึงเหมาะสมกับการนำไปเขียนโปรแกรมที่มีความซับซ้อน มีความยืดหยุ่นต่อการใช้งาน เพิ่มประสิทธิภาพในการเขียนโปรแกรมมากยิ่งขึ้น ซึ่งมีข้อดีดังตารางที่ 1

องค์ประกอบของเชิงวัตถุ

การเขียนโปรแกรมเชิงวัตถุ มีแนวคิดและองค์ประกอบมาจากความหมายของเชิงวัตถุ ประกอบไปด้วย 4 ส่วนสำคัญ ได้แก่ คลาส วัตถุ คุณสมบัติ และพฤติกรรม (Weisfeld, 2013) แต่ละส่วนมีรายละเอียดดังต่อไปนี้

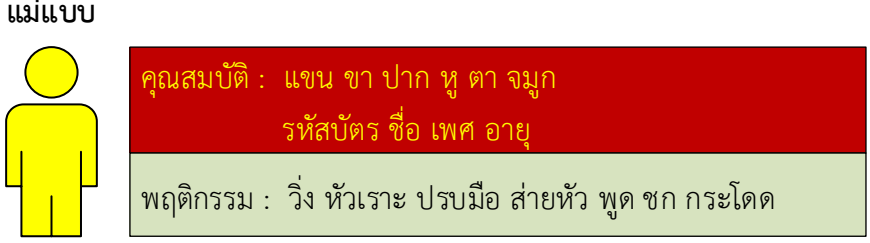


ภาพที่ 1 ส่วนประกอบของเชิงวัตถุ

จากภาพที่ 1 อธิบายส่วนประกอบของเชิงวัตถุ โดยแนวคิดประกอบไปด้วยคลาส ภายในคลาสนั้นจะประกอบด้วยคุณสมบัติและพฤติกรรม กำหนดคุณลักษณะของวัตถุ วัตถุใดที่มีคุณสมบัติหรือพฤติกรรมเดียวกับคลาส ก็ถือว่าเป็นวัตถุนั่นอยู่ในคลาสดังกล่าว ส่วนประกอบของเชิงวัตถุมีรายละเอียดดังต่อไปนี้

1. คลาส

คลาส (Class) หมายถึง ศัพท์ที่อธิบายโครงร่างหรือพิมพ์เขียวที่กำหนดขอบเขตกลุ่มของวัตถุ (Weisfeld, 2013) มีลักษณะการให้นิยามความหมายและคำจำกัดความกลุ่มของวัตถุกลุ่มใดกลุ่มหนึ่ง มีรายละเอียดดังต่อไปนี้



ภาพที่ 2 คลาส

ภาพที่ 2 ตัวอย่างคลาสมนุษย์ เมื่อมองในความหมายขอบเขตของมนุษย์ ก็จะให้ความหมายว่าเป็นลักษณะของสิ่งมีชีวิตที่เป็นรูปธรรม เช่น แขน ขา หู ตา จมูก เป็นต้น หรือคุณสมบัติด้านนามธรรมภายในตัวของมนุษย์ เช่น ชื่อ รหัสประจำตัวประชาชน ที่อยู่ อายุ เพศ น้ำหนัก ส่วนสูง เป็นต้น ในส่วนนี้จะมองในลักษณะของคลาสนั้นบ่งบอกลักษณะภายในตัวของคลาสและส่วนที่สอง เป็นส่วนของการมองในส่วนของคลาสนั้นสามารถเกิดการกระทำ จากตัวอย่างคลาสของมนุษย์ สามารถที่จะกระทำ หรือแสดงพฤติกรรมออกมาได้ เช่น วิ่ง หัวเราะ ปรบมือ ส่ายหัว พูด แกว่งแขน กระโดด เป็นต้น ซึ่งเมื่อมองในความหมายภาพรวมเห็นได้ว่าเป็นลักษณะของมุมมองที่เป็นนามธรรม ที่กำหนดขอบเขตของกลุ่มใดกลุ่มหนึ่ง

2. คุณสมบัติ

คุณสมบัติ (Properties) หมายถึง องค์ประกอบภายในคลาส ใช้ในการกำหนดข้อมูลที่เก็บภายในคลาสหรือส่วนที่กำหนดคุณลักษณะ ข้อกำหนดของคลาสนั้นมี (บัญชา ปะสีละเตสัง, 2558) แต่ละคลาสจะกำหนดคุณสมบัติที่ซึ่งกำหนดสถานะของวัตถุที่เกิดจากคลาสนั้น (Weisfeld, 2013) มีรายละเอียดดังต่อไปนี้

ภาพที่ 3 คุณสมบัติ

ภาพที่ 3 ตัวอย่างคุณสมบัติที่เกิดจากคลาสมนุษย์ มนุษย์จะมีคุณสมบัติภายในคลาสของมนุษย์นั้นมี เช่น แขน ขา ชื่อ รหัสบัตร ที่อยู่ อายุ เพศ เป็นต้น คุณสมบัติเป็นองค์ประกอบที่สำคัญของคลาส โดยจะกำหนดคุณลักษณะที่มีของคลาสนั้น คุณสมบัติบางอย่างจะส่งผลต่อเนื่องไปยังพฤติกรรมที่เกิดขึ้นของคลาส

3. พฤติกรรม

พฤติกรรม (Behavior) หรือการกระทำ (Action) หมายถึง องค์ประกอบที่อยู่ภายในคลาส ใช้ในการกำหนดการกระทำบางอย่างของคลาส (ธีรวัฒน์ ประกอบผล, 2558) พฤติกรรมเป็นการ

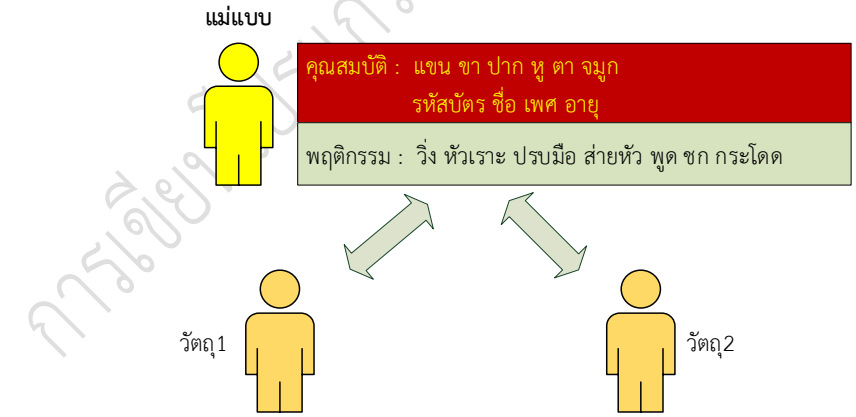
กำหนดขอบเขตที่สามารถกระทำได้ของคลาส ส่วนมากจะเกิดจากคุณสมบัติที่มีอยู่ในคลาส พฤติกรรม มีรายละเอียดดังต่อไปนี้

ภาพที่ 4 พฤติกรรม

ภาพที่ 4 ตัวอย่างพฤติกรรมที่เกิดจากคลาสมนุษย์ พฤติกรรมมักเกิดจากคุณสมบัติที่มีภายในคลาส เช่น คุณสมบัติมีขาก็จะสามารถวิ่งได้ มีแขนก็สามารถชกได้ มีปากก็สามารถหัวเราะหรือพูดได้ เป็นต้น พฤติกรรมเป็นกระบวนการที่เกิดขึ้นตั้งแต่นำเข้าข้อมูล ประมวลผลและเกิดผลลัพธ์

4. วัตถุ

วัตถุ (Object) หมายถึง การมองสิ่งที่อยู่รอบข้างเป็นวัตถุเป้าหมาย ภายในวัตถุจะประกอบไปด้วยคุณสมบัติของวัตถุและพฤติกรรมหรือการกระทำ ที่ตอบสนองต่อเหตุการณ์ที่จะทำให้เกิดขึ้นของวัตถุ (สุชาติ คุ้มมณี, 2558) มีรายละเอียดดังต่อไปนี้



คุณสมบัติ	พฤติกรรม
รหัสบัตร : 2131261548974 ชื่อ : สมหมาย ใจดี เพศ : ชาย อายุ : 27	วิ่งเร็ว หัวเราะ ปรบมือ ส่ายหัว

คุณสมบัติ	พฤติกรรม
รหัสบัตร : 1234569878954 ชื่อ : สมใจ โชคดี เพศ : หญิง อายุ : 40	วิ่งช้า หัวเราะ ปรบมือ ส่ายหัว

ภาพที่ 5 วัตถุ

วัตถุ คือ สิ่งที่มีตัวตนที่เกิดขึ้นจากคลาส ดังนั้น จึงมีคุณสมบัติและพฤติกรรมที่ได้รับมาจากคลาส วัตถุมีข้อมูลของตนเองที่จะบ่งบอกความแตกต่างเฉพาะวัตถุ ดังตัวอย่างภาพที่ 5 วัตถุ1 และวัตถุ2 จะมีคุณสมบัติเหมือนกัน คือ รหัสบัตร ชื่อ เพศและอายุ แต่ข้อมูลที่อยู่ภายในคุณสมบัตินี้จะมีความแตกกต่างกัน ที่บ่งบอกคุณลักษณะเฉพาะของวัตถุ

จากหลักการแนวความคิดของเชิงวัตถุดังกล่าว การเขียนโปรแกรมด้วยภาษาโปรแกรมที่สนับสนุนการเขียนโปรแกรมแบบเชิงวัตถุ เมื่อนำหลักการนี้ไปเขียนในโปรแกรม จะมีรูปแบบและคำสั่งที่แตกต่างกัน แต่แนวคิดของการทำงานของโปรแกรมจะใช้หลักเดียวกันดังที่ได้กล่าวไว้ข้างต้น

ขั้นตอนการเขียนโปรแกรมเชิงวัตถุ

การเขียนโปรแกรมเชิงวัตถุหรือการเขียนโปรแกรมแบบวัตถุ จะมองทุกอย่างของปัญหาเป็นวัตถุ หากวัตถุใดมีลักษณะคล้ายกันก็จะรวมทั้งหมดให้เป็นคลาส ถ้าหากมีการประกาศตัวแปรหรือสร้างข้อมูลขึ้นมา ข้อมูลนั้นก็จะถูกเก็บไว้ใช้ในวัตถุนั้น การกระทำกับวัตถุจะกระทำผ่านพฤติกรรมของคลาสนั้น ภายในวัตถุจะประกอบไปด้วยส่วนคำญสองส่วน คือ คุณสมบัติ เป็นข้อมูลประจำตัวของวัตถุ และพฤติกรรม บ่งบอกวัตถุกำลังทำสิ่งใดอยู่ ถ้าหากมีการสร้างวัตถุขึ้นมาและโปรแกรมต้องการจัดการกับข้อมูลที่เป็นคุณลักษณะของวัตถุ จะกระทำผ่านพฤติกรรม (ธีรวัฒน์ ประกอบผล, 2558)

ภาษาไพธอน สนับสนุกการเขียนโปรแกรมเชิงวัตถุ จากความหมายและหลักการเชิงวัตถุสามารถนำมาเขียนโปรแกรม มีส่วนประกอบรายละเอียดดังต่อไปนี้

1. คลาส แอททริบิวต์และเมธอด

การสร้างคลาสนั้น จะใช้คำสั่ง class เป็นการกำหนดการสร้างคลาส ซึ่งในคลาสจะประกอบไปด้วยการทำงาน 3 ส่วน คือ ส่วนของแอทริบิวต์ เมธอด และคำสั่ง self

แอททริบิวต์ (Attribute) หมายถึง ตัวแปรที่อยู่ภายในคลาส สำหรับเก็บข้อมูลเพื่อใช้งานภายในคลาสและอ็อบเจกต์ ในเบื้องต้น คือ คุณสมบัติหรือพร็อพเพอร์ตี้ (สุชาติ คุ้มมณี, 2558)

เมธอด (Method) หมายถึง ฟังก์ชันสร้างขึ้นภายในคลาส เพื่อดำเนินการหรือกระทำต่อเหตุการณ์ต่าง ๆ (สุชาติ คุ้มมณี, 2558) ในเบื้องต้นก็คือ พฤติกรรมหรือการกระทำ การดำเนินการสร้างคลาสมีรูปแบบการเขียนเหมือนกับฟังก์ชัน

คำสั่ง self หมายถึง คำสั่งที่มีความความสำคัญภายในคลาส คำสั่งนี้จะถูกอ้างอิงใช้งานภายในคลาสผ่านทั้งแอทริบิวต์และเมธอด (Sheppard, 2014)

จากลักษณะข้างต้น รูปแบบของการสร้างคำสั่ง คลาส แอททริบิวต์และเมธอด มีรายละเอียดดังต่อไปนี้

```
class ชื่อคลาส():
        ชื่อแอททริบิวต์
        หรือ ชื่อแอททริบิวต์=กำหนดค่า
        def ชื่อเมธอด(self):
            ชุดคำสั่ง
```

องค์ประกอบของคลาส เมื่อนำมาเขียนโปรแกรม มีรูปแบบโครงสร้างการทำงาน รายละเอียดดังต่อไปนี้

ตัวอย่างที่ 1 การสร้างคลาส

บรรทัดที่	คำสั่ง
1	class circle_area():
2	radius=10
3	area=0
4	def cal_area(self):
5	self.area =22/7*(self.radius**2)
6	def show_area(self):
7	print ('พื้นที่วงกลม =',self.area)

โปรแกรมตัวอย่างที่ 1 โครงสร้างของคลาสประกอบไปด้วยแอททริบิวต์ radius และ area กำหนดเป็นค่าคงที่ เมธอด cal_area() ทำหน้าที่ในการคำนวณหาพื้นที่วงกลมและเมธอด show_area(self) ทำหน้าที่ในการแสดงผลที่ได้ เมื่อทำการประมวลผล จะไม่มีผลลัพธ์อะไรเกิดขึ้น เนื่องจากมีเฉพาะโครงสร้างของคลาสเท่านั้น ที่กำหนดขอบเขตการทำงานของคลาส

2. วัตถุและอินสแตนซ์

วัตถุ หรือ อ๊อบเจกต์ (ราชบัณฑิตยสถาน, 2554) คือ การให้นิยามความหมายด้านแนวคิด แต่เมื่อนำไปใช้ในการเขียนโปรแกรมเชิงวัตถุ เมื่อมีการสร้างวัตถุที่เกิดจากคลาส วัตถุที่ถูกสร้างขึ้นใหม่นั้น จะเรียกว่า “อินสแตนซ์ (Instance)” (Gerrard, 2016) หรือสิ่งที่เป็นจริง ที่สร้างหรือเกิดจากคลาส (สุชาติ คุ้มมณี, 2558) การทำงานของโปรแกรม จะใช้งานผ่านอินสแตนซ์ โดยการสร้างอินสแตนซ์มีรูปแบบคำสั่งดังต่อไปนี้

```
ชื่ออินสแตนซ์=ชื่อคลาส()
```

รูปแบบการใช้คำสั่งสร้างอินสแตนซ์ สามารถนำไปใช้ในการเขียนโปรแกรมได้ดังโปรแกรมตัวอย่างที่ 2

ตัวอย่างที่ 2 การสร้างอ็อบเจกต์หรืออินสแตนซ์

บรรทัดที่	คำสั่ง
1	class circle_area():
2	radius=10
3	area=0
4	pi=22/7
5	def cal_area(self):
6	self.area =self.pi*(self.radius**2)
7	def show_area(self):
8	print ('พื้นที่วงกลม =',self.area)
9	circle1=circle_area()
10	print(circle1.radius)
11	print(circle1.area)
12	circle1.cal_area()
13	circle1.show_area()
ผลลัพธ์	
10 0 พื้นที่วงกลม = 314.2857142857143	

การสร้างอินสแตนซ์ circle1 ที่เกิดจากคลาส circle_area() ในบรรทัดที่ 8 อินสแตนซ์ที่ได้จะได้รับถ่ายทอดแอทริบิวต์และเมธอดมาจากคลาส circle_area() ดังนั้นอินสแตนซ์จึงสามารถเรียกใช้แอททริบิวต์และเมธอดที่อยู่ภายในคลาสได้ โดยการเรียกจะใช้รูปแบบ “อินสแตนซ์.แอททริบิวต์” และ “อินสแตนซ์.เมธอด()” ซึ่งได้ผลลัพธ์ดังโปรแกรมตัวอย่างที่ 2

3. การเข้าถึงคลาส

อ็อบเจกต์ที่เกิดจากคลาส สามารถที่จะเข้าถึงรายละเอียดภายในคลาสได้ โดยทั่วไปการประกาศแอททริบิวต์และเมธอด จะมีลักษณะของการเข้าถึงแบบ public จึงจะสามารถเปลี่ยนแปลงรายละเอียดข้อมูลที่อยู่ในแอททริบิวต์ได้ ดังโปรแกรมตัวอย่างที่ 3

ตัวอย่างที่ 3 การเข้าถึงคลาส

บรรทัดที่	คำสั่ง
1	class circle_area():
2	radius=10
3	pi=22/7
4	area=0
5	def cal_area(self):
6	self.area =self.pi*(self.radius**2)
7	def show_area(self):
8	print ('ค่า pi = ',self.pi)
9	print ('พื้นที่วงกลม =',self.area)
10	circle1=circle_area()
11	circle1.radius = 20
12	print("รัศมี = ",circle1.radius)
13	circle1.cal_area()
14	circle1.show_area()
ผลลัพธ์	
รัศมี = 20 ค่า pi = 3.142857142857143 พื้นที่วงกลม = 1257.142857142857	

การเข้าถึงแบบ public โปรแกรมตัวอย่างที่ 3 การทำงานมีการส่งค่าเข้าไปในคลาสผ่านอินสแตนซ์ circile1 ผ่านเข้าไปยังแอททริบิวต์ radius ที่อยู่ในคลาส ดังบรรทัดที่ 11 แอททริบิวต์ได้มีการเปลี่ยนแปลงค่าเดิมเป็นค่าใหม่ที่ถูกส่งเข้าไปภายในคลาส การเข้าถึงคลาสในลักษณะนี้เป็นรูปแบบการเข้าถึงคลาสแบบ public

การทำงานบางกรณีแอททริบิวต์ ไม่ต้องการให้สามารถเปลี่ยนแปลงข้อมูลได้ เพื่อรักษาความปลอดภัยให้กับข้อมูลนั้น จะสามารถกำหนดได้โดยใช้สัญลักษณ์ __ หน้าชื่อแอททริบิวต์หรือเมธอด เป็นการกำหนดให้การเข้าถึงนั้นเป็นแบบ private ดังโปรแกรมตัวอย่างที่ 4

ตัวอย่างที่ 4 การเข้าถึงคลาสแบบ private

บรรทัดที่	คำสั่ง
1	class circle_area():
2	__radius=10
3	__pi=22/7
4	area=0
5	def __cal_area(self):
6	self.area =self.__pi*(self.__radius**2)
7	def show_area(self):
8	self.__cal_area()
9	print ('ค่ารัศมี จากแอททริบิวต์ = ',self.__radius)
10	print ('ค่า pi จากแอททริบวิต์ = ',self.__pi)
11	print ('พื้นที่วงกลม =',self.area)
12	circle1=circle_area()
13	circle1.__radius =5
14	circle1.__pi = 20/7
15	print("ค่ารัศมี จากอินสแตนซ์ = ",circle1.__radius)
16	print("ค่า pi จากอินสแตนซ์ = ",circle1.__pi)
17	circle1.show_area()

ผลลัพธ์
ค่ารัศมี จากอินสแตนซ์ = 5 ค่า pi จากอินสแตนซ์ = 2.857142857142857 ค่ารัศมี จากแอททริบิวต์ = 10 ค่า pi จากแอททริบวิต์ = 3.142857142857143 พื้นที่วงกลม = 314.2857142857143

โปรแกรมตัวอย่างที่ 4 มีการสร้างอินสแตนซ์ circle1 ที่เรียกใช้แอททริบิวต์ภายในคลาส ที่เป็นแบบ private คือ __radius และ __pi อินสแตนซ์มีการกำหนดค่าใหม่ของแอททริบิวต์ ในบรรทัดที่ 13 และ 14 ซึ่งผลลัพธ์ที่ได้ คือ ไม่สามารถเปลี่ยนแปลงข้อมูลแอทริบิวต์ที่อยู่ในคลาสได้ ซึ่งจะสามารถใช้แอททริบิวต์ภายในอินสแตนซ์ตัวเองได้เท่านั้น ดังผลลัพธ์ที่แสดงในตัวอย่างที่ 4 และการเรียกใช้เมธอดก็เช่นเดียวกัน ไม่สามารถเรียกใช้เมธอดที่เป็น private โดยตรงได้ เมื่อมีการเรียกเมธอด __cal_area() จะเกิดการผิดพลาดเของโปรแกรม คือ AttributeError: 'circle_area' object has no attribute '__cal_area'

ลักษณะของคุณบัติแบบ private มีวัตถุประสงค์เพื่อต้องการใช้งานข้อมูลแอททริบิวต์และเมธอดภายในคลาสเท่านั้น จะไม่ยอมอนุญาตให้สิ่งที่อยู่ภายนอกคลาสสามารถเปลี่ยนแปลงข้อมูลหรือใช้งานได้ การอนุญาตใช้งานจะต้องผ่านเมธอดที่เป็นแบบ public ภายในคลาส ที่เมธอดนั้นจะไปดำเนินการแอทริบิวต์และเมธอดที่อยู่ภายในคลาสได้ จากโปรแกรมตัวอย่างที่ 4 จะเห็นได้ว่าอินสแตนซ์ circle1 ได้เรียกใช้เมธอด show_area() ที่เป็นเมธอดแบบ public อินสแตนซ์สามารถเรียกใช้งานได้และเมธอดแบบ show_area() จะไปเรียกใช้เมธอด ____cal_area() อีกครั้ง

4. คอนสตรัคเตอร์และดีสตรัคเตอร์

คอนสตรัคเตอร์ (Constructor) หมายถึง เมธอดหรือฟังก์ชันแรกที่จะถูกเรียกใช้ เมื่อมีการสร้างวัตถุหรืออินสแตนซ์ เพื่อกำหนดสภาพแวดล้อมเบื้องต้นทุกครั้ง (สุชาติ คุ้มมณี, 2558) ก่อนนำข้อมูลนั้นไปสู่ขบวนการ การทำงานต่อตามเงื่อนไขต่อไป การสร้างคอนสตรัคเตอร์จะใช้คำสั่ง __int__ เป็นการระบุเมธอดนั้นเป็นคอนสตรัคเตอร์

ดีสตรัคเตอร์ (Destructor) หมายถึง เมธอดหรือฟังก์ชัน ที่จะถูกเรียกใช้เมื่อมีการลบวัตถุหรืออินสแตนซ์ มีจุดมุ่งหมายเพื่อทำการล้างตัวแปรและคืนหน่วยความจำสู่ระบบ การสร้างดีสตรัคเตอร์ จะใช้คำสั่ง __del__ เป็นการระบุเมธอดนั้นเป็นดีสตรัคเตอร์

ตัวอย่างที่ 5 การสร้างคอนสตรัคเตอร์

บรรทัดที่	คำสั่ง
1	class circle_area():
2	radius=10
3	area=0
4	def cal_area(self):
5	self.area =22/7*(self.radius**2)
6	def show_area(self):

```
7           print ('พื้นที่วงกลม =',self.area)
8       def __init__(self):
9           print("คอนสตรัคเตอร์ เริ่มต้นการทำงาน")
10      def __del__(self):
11          print("ดีสตรัคเตอร์ สิ้นสุดการทำงาน")
12  circle1=circle_area()
13  print(circle1.radius)
14  print(circle1.area)
15  circle1.cal_area()
16  circle1.show_area()
17  del circle1
```

ผลลัพธ์

```
คอนสตรัคเตอร์ เริ่มต้นการทำงาน
10
0
พื้นที่วงกลม = 314.2857142857143
ดีสตรัคเตอร์ สิ้นสุดการทำงาน
```

โปรแกรมตัวอย่างที่ 5 จะเห็นได้ว่า เมื่อมีการสร้างอินสแตนซ์ขึ้นมาในบรรทัดที่ 12 ดีสตรัคเตอร์จะถูกเรียกเป็นอันดับแรกอัตโนมัติ โดยไม่ต้องผ่านการเรียกเมธอดโดยอินสแตนซ์ และเมื่อมีการลบอินสแตนซ์ ในบรรทัดที่ 17 โปรแกรมจะทำการเรียกดีสตรัคเตอร์โดยอัตโนมัติ ซึ่งจะเป็นการดำเนินการขั้นสุดท้ายของการทำงานอินสแตนซ์ เพื่อคืนหน่วยความจำสู่ระบบ จะทำให้ระบบมีประสิทธิภาพการทำงานสูงขึ้น

5. การส่งผ่านค่าและคืนค่าพารามิเตอร์

การเข้าถึงคลาสที่เป็นแบบ private จะไม่สามารถเปลี่ยนแปลงข้อมูลที่อยู่ภายในในคลาสโดยตรงผ่านอินสแตนซ์ได้ ดังนั้นวิธีการที่จะให้สามารถเปลี่ยนแปลงข้อมูลได้ คือ การใช้วิธีส่งค่าพารามิเตอร์เข้าในเมธอดภายในคลาส เมธอดที่รับค่าเข้าไปจะต้องเป็นเมธอดแบบ public ที่อนุญาตให้ใช้เมธอด เพื่อดำเนินการเปลี่ยนแปลงข้อมูลที่อยู่ภายในคลาสและคืนค่าข้อมูลออกมาจากเมธอด ซึ่งค่าที่คืนอาจนำไปใช้ในการทำงานในเมธอดอื่นภายในคลาสหรือคืนค่ากลับมายังอินสแตนซ์ที่ส่งค่าพารามิเตอร์ ดังโปรแกรมตัวอย่างที่ 6

ตัวอย่างที่ 6 การส่งผ่านค่าและคืนค่าพารามิเตอร์

บรรทัดที่	คำสั่ง
1	class cal_cost:
2	def __init__(self, width, height, u_cost):
3	self.width = width
4	self.height = height
5	self.u_cost = u_cost
6	def cal_area(self):
7	return self.width * self.height
8	def cal_cost(self):
9	area = self.cal_area()
10	return area * self.u_cost
11	value_width=float(input("ความกว้าง (เมตร) = "))
12	value_height=float(input("ความยาว (เมตร) = "))
13	unit_cost=float(input("ราคา/ตารางเมตร = "))
14	area_cost = cal_cost(value_width, value_height, unit_cost)
15	print("พื้นที่ (ตารางเมตร) = ", (area_cost.cal_area()))
16	print("ราคารวมของที่ดิน = ",(area_cost.cal_cost()))

ผลลัพธ์

```
ความกว้าง (เมตร) = 30                    (รับข้อมูลจากแผงแป้นอักขระ)
ความยาว (เมตร) = 40                      (รับข้อมูลจากแผงแป้นอักขระ)
ราคา/ตารางเมตร = 5000                    (รับข้อมูลจากแผงแป้นอักขระ)
พื้นที่ (ตารางเมตร) =  1200.0
ราคารวมของที่ดิน =  6000000.0
```

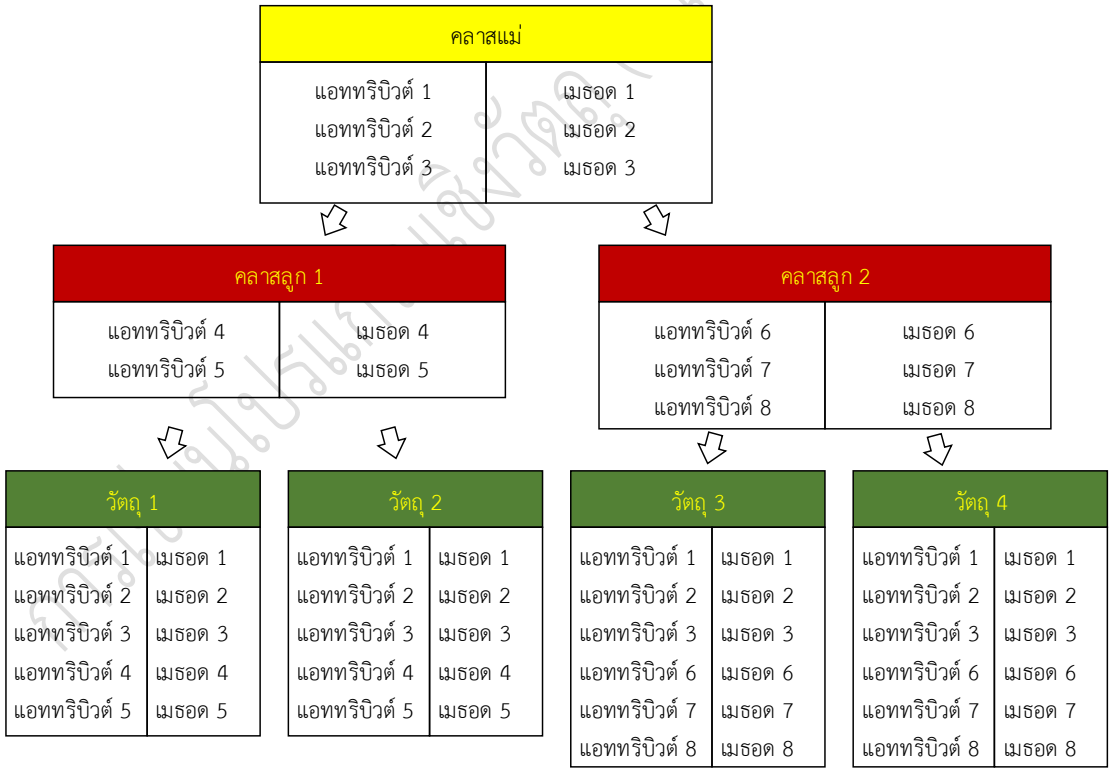
โปรแกรมตัวอย่างที่ 6 การคำนวณหาราคาของที่ดิน โดยคิดเป็นราคาต่อตารางเมตร มีการรับข้อมูลจากแป้นพิมพ์ประกอบด้วย w h และ c คือ ความกว้าง ความยาวและราคาต่อตารางเมตร เมื่อสร้างอินสแตนซ์พร้อมกับส่งค่าพารามิเตอร์ เมธอดคอนสตรัคเตอร์ จะทำหน้าที่รับค่าข้อมูลเบื้องเพื่อใช้งานภายในคลาสและเมื่ออินสแตนซ์มีการเรียกเมธอด cal_cost() ในบรรทัดที่ 16 เมธอด cal_cost() ทำการเรียกเมธอด cal_area() ทำหน้าที่ในการคำนวณหาพื้นที่ พร้อมทั้งคืนค่าที่ได้จาก

การคำนวณกลับมายังเมธอด cal_cost() นำค่าที่ได้ไปคำนวณราคารวมของที่ดินและคืนค่ามากลับแสดงผลลัพธ์

การทำงานของเมธอดภายในคลาส นอกเหนือจากความสัมพันธ์ระหว่างอินสแตนซ์กับเมธอด ยังมีความสัมพันธ์ระหว่างเมธอดกับเมธอดภายในคลาส เมธอดมีหน้าที่การทำงานแตกต่างกัน เมื่อประมวลผลเสร็จภายในเมธอด จะดำเนินการคือค่ากลับมายังส่วนที่เรียกใช้เมธอดนั้น เพื่อนำค่าที่ได้ไปใช้ดำเนินการต่อไปนี้

6. การสืบทอดคลาส

การสืบทอดคลาส (Inheritance) หมายถึง ความสามารถของคลาสที่มีอยู่แล้วกลับมาใช้ใหม่ (ธีรวัฒน์ ประกอบผล, 2558) เป็นการสืบทอดคุณสมบัติจากคลาสหลัก (Super Class) ไปยังคลาสย่อย (Sub Class) นั่นก็คือ แอททริบิวต์และเมธอด โดยมีแนวคิด คุณสมบัติบางประการที่เหมือนกันควรสืบทอดระหว่างคลาสแม่และคลาสลูกได้ โดยไม่จำเป็นต้องสร้างคุณสมบัติใหม่ (สุชาติ คุ้มมณี, 2558) ลักษณะของการสืบทอดคลาสมีโครงสร้างดังภาพที่ 6



ภาพที่ 6 โครงสร้างการสืบทอดคลาส

การสร้างคลาสลูกที่เกิดจากคลาสแม่ ในลักษณะของการสืบทอดคลาส มีรูปแบบของการสร้างคลาสคล้ายกัน แต่แตกต่างกันตรงที่การสร้างคลาสลูกจะต้องระบุคลาสลูกเกิดจากคลาสแม่

ใด โดยมีรูปแบบการใช้คำสั่ง “class ชื่อคลาสลูก(ชื่อคลาสแม่)” การใช้งานแบบสืบทอดคลาส มีรายละเอียดดังโปรแกรมตัวอย่างที่ 7

ตัวอย่างที่ 7 การสืบทอดคลาส

บรรทัดที่	คำสั่ง
1	class area_cal():
2	area=0
3	def show_area(self):
4	print("---การคำนวณพื้นที่---")
5	class circle(area_cal):
6	radius=10
7	def circle_area():
8	area=22/7*(radius**2)
9	class ractangle(area_cal):
10	width=5
11	height=4
12	def ractangle_area():
13	area=width*height
14	circle1=circle()
15	print('เมธอดที่อยู่ในคลาส circle ของอินสแตนซ์ circle1 =',dir(circle1))
16	print('-'*25)
17	ractangle1=ractangle()
18	print('เมธอดที่อยู่ในคลาส ractangle ของอินสแตนซ์ ractangle1 =',dir(ractangle1))

ผลลัพธ์

เมธอดที่อยู่ในคลาส circle ของอินสแตนซ์ circle1 = ['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'area', 'circle_area', 'radius', 'show_area']

```
เมธอดที่อยู่ในคลาส ractangle ของอินสแตนซ์ ractangle1 = ['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'area', 'height', 'ractangle_area', 'show_area', 'width']
```

ผลลัพธ์ของโปรแกรมตัวอย่างที่ 7 นอกเหนือจากเมธอดแบบ Built-In ที่ภาษาไพธอนได้มีให้ใช้งานภายในคลาสเป็นจำนวนมาก สามารถสร้างแอททริบิวต์และเมธอดขึ้นมาใช้งานใหม่ได้ โดยบรรทัดที่ 15 ให้แสดงคุณสมบัติที่มีอยู่ในอินสแตนซ์ circle1 ที่เกิดจากคลาสลูก circle และคลาส circle เกิดจากคลาสแม่ area ดังนั้นอินสแตนซ์ circle1 จึงได้รับการถ่ายทอดแอททริบิวต์และเมธอดมาจากคลาสทั้งสอง ซึ่งผลลัพธ์ของคุณสมบัติที่มีนอกเหนือจาก Built-In นั่นก็คือแอททริบิวต์ area และ radius เมธอด circle_area() และ show_area() ส่วนอินสแตนซ์ ractangle1 ก็ได้รับการถ่ายทอดแอททริบิวต์และเมธอดมาจากคลาส area_cal และ ractangle_cal ทำให้มีอินสแตนซ์มี แอททริบิวต์ area width และ height และมีเมธอด ractangle_area() และ show_area()

จากกระบวนการทำงานดังกล่าว จะเห็นได้ว่า คลาสหลักหนึ่งคลาส สามารถแยกย่อยไปได้อีกหลายคลาสและแต่ละคลาสย่อยจะสามารถแยกคลาสย่อยลงไปอีกเรื่อย ๆ จนสิ่งสุดท้ายที่จะสามารถนำไปใช้งานได้ คือ อ็อบเจกต์ต์หรืออินสแตนซ์ การถ่ายทอดแอททริบิวต์และเมธอดจากระดับคลาสที่อยู่ลำดับขั้นบนลงมาให้ จะมีประโยชน์มากในการเขียนโปรแกรม เพราะไม่ต้องกำหนดคุณสมบัติที่มีอยู่แล้วเพิ่ม ทำให้ประสิทธิภาพการทำงานของโปรแกรมสูงขึ้น

7. การโอเวอร์ไรดิ้ง

การโอเวอร์ไรดิ้ง (Overriding) หมายถึง การดำเนินการของเมธอดหรือฟังก์ชันที่มีชื่อเหมือนกันในคลาสแม่ แต่มีการดำเนินการที่แตกต่างกันในคลาสลูก (สุชาติ คุ้มมณี, 2558) ซึ่งคลาสที่ถูกแยกออกไปจากคลาสแม่ สามารถเปลี่ยนแปลงรายละเอียดภายในเมธอดลูกง่ายกว่าการเปลี่ยนแปลงในเมธอดแม่ (Rossum, 2013) ในเมธอดที่มีชื่อเดียวกัน ดังโปรแกรมตัวอย่างที่ 8

ตัวอย่างที่ 8 การโอเวอร์ไรดิ้ง

บรรทัดที่	คำสั่ง
1	class cal_area:
2	def show_area(self):

```
3           print ("พื้นที่ = ไม่มี")
4   class circle(cal_area):
5       def show_area(self):
6           print ("การคำนวณพื้นที่วงกลม = 22/7*(รัศมี*รัศมี)")
7   class square(cal_area):
8       def show_area(self):
9           print ("การคำนวนพื้นที่สี่เหลี่ยมจัสตุรัส = ความกว้าง*ความยาว")
10  class drummy_area(cal_area):
11      pass
12  obj1=cal_area()
13  obj1.show_area()
14  obj2=circle()
15  obj2.show_area()
16  obj3=square()
17  obj3.show_area()
18  obj4=drummy_area()
19  obj4.show_area()
```

ผลลัพธ์

```
พื้นที่ = ไม่มี
การคำนวณพื้นที่วงกลม = 22/7*(รัศมี*รัศมี)
การคำนวนพื้นที่สี่เหลี่ยมจัสตุรัส = ความกว้าง*ความยาว
พื้นที่ = ไม่มี
```

โปรแกรมตัวอย่างที่ 8 ประกอบไปด้วยคลาสแม่ cal_area และคลาสลูก circle square โดยมีเมธอด ชื่อเดียวกัน show_area() โดยทั้งสามคลาสจะมีการทำงานภายในเมธอดแตกต่างกัน เมื่อเรียกใช้เมธอดของแต่ละคลาสผ่านอินสแตนซ์ จะพบว่า ผลลัพธ์มีความแตกต่างกัน และคลาส drummy_area ไม่เมธอด แต่มีคำสั่ง pass นั่นคือ ไม่มีการทำงานหรือเปลี่ยนแปลงรายละเอียดในคลาสจะได้ผลลัพธ์เหมือนกับเมธอดในคลาสแม่ ซึ่งลักษณะดังกล่าวคือ การโอเวอร์ไรดิ้งเมธอด

8. การโอเวอร์โหลดดิ้ง

การโอเวอร์โหลดดิ้ง (Overloading) ที่ใช้งานภายในคลาส ประกอบไปด้วย การโอเวอร์โหลดดิ้งเมธอดและการโอเวอร์โหลดดิ้งตัวดำเนินการ ซึ่งมีลักษณะการทำงานดังต่อไปนี้

8.1 การโอเวอร์โหลดดิ้ง

การโอเวอร์โหลดดิ้ง (Overloading Method) หมายถึง เมธอดหรือฟังก์ชันที่มีชื่อเหมือนกัน แต่สามารถแยกการทำงานด้วยอากิวเมนต์ มีลักษณะใกล้เคียงกันกับโอเวอร์ไรดิ้ง แต่ต่างกันกันตรงที่โอเวอร์โหลดดิ้งจะอยู่ในคลาสเดียวกันหรือสืบทอดคลาสก็ได้ แต่โอโวอร์ไรดิ้งจะเป็นลักษณะของการสืบทอดคลาส (สุชาติ คุ้มมณี, 2558) มีลักษณะของการทำงานดังโปรแกรมตัวอย่างที่ 9

ตัวอย่างที่ 9 การโอเวอร์โหลดดิ้งเมธอด

บรรทัดที่	คำสั่ง
1	class chk_val:
2	def print_val(self,*args):
3	if(len(args)==0):
4	print("ไม่มีการโอเวอร์โหลดดิ้ง")
5	else:
6	print("โอเวอร์โหลดดิ้ง อาร์กิวเมนต์ =%d "%(len(args)))
7	lists = []
8	for i in args:
9	lists.append(i)
10	print("โอเวอร์โหลดดิ้ง อาร์กิวเมนต์ = ",lists)
11	val1 = chk_val()
12	val2 = chk_val()
13	val3 = chk_val()
14	val1.print_val()
15	val2.print_val("python")
16	val3.print_val("python", 5, 8,)
ผลลัพธ์	
ไม่มีการโอเวอร์โหลดดิ้ง โอเวอร์โหลดดิ้ง อาร์กิวเมนต์ =1 โอเวอร์โหลดดิ้ง อาร์กิวเมนต์ = ['python'] โอเวอร์โหลดดิ้ง อาร์กิวเมนต์ =3 โอเวอร์โหลดดิ้ง อาร์กิวเมนต์ = ['python', 5, 8]	

การทำงานของเมธอด print_val() เป็นการทำงานแบบโอเวอร์โหลดดิ้ง โดยสามารถรับพารามิเตอร์ได้ไม่จำกัด ภายในเมธอดจะมีการตรวจสอบพารามิเตอร์ โดยถ้าไม่มีพารามิเตอร์เข้ามาจะไม่มีการทำงานของเมธอดหรือเมธอดโอเวอร์โหลดดิ้ง จะทำการโอเวอร์โหลดดิ้งก็ต่อเมื่อมีการส่งค่าพารามิเตอร์เข้ามาในเมธอด 1 ค่าขึ้นไป ซึ่งจะแสดงผลจำนวนอาร์กิวเมนต์ที่โอเวอร์โหลดภายในเมธอด ดังผลลัพธ์ที่ได้โปรแกรมตัวอย่างที่ 9

การโอเวอร์โหลดดิ้งเมธอดนั้น จะใช้อาร์กิวเมนต์ *args เป็นตัวแปรประเภททูเพิลในการทำงาน เป็นเมธอดที่สามารถรับค่าพารามิเตอร์ไม่จำกัด โดยชื่อเหมือนเดิมแตกต่างกจากโอเวอร์ไรดิ้งที่ต้องมีชื่อเมธอดและอาร์กิวเมนต์เหมือนกันและทำงานตามหลักการของการสืบทอดคลาส แต่โอเวอร์โหลดดิ้งอาจอยู่ในคลาสเดียวกันหรือแบบสืบทอดคลาสได้ (สุชาติ คุ้มมณี, 2558)

8.2 การโอเวอร์โหลดตัวดำเนินการ

การโอเวอร์โหลตัวดำเนินการ หมายถึง การอนุญาตให้อ๊อบเจ็กที่สร้างจากคลาสสามารถใช้ตัวดำเนินการพื้นฐานได้ (สุชาติ คุ้มมณี, 2558) ซึ่งสามารถแยกประเภทการการโอเวอร์โหลดตามกลุ่มการใช้งานมีรายละเอียดดังต่อไปนี้

8.2.1 การโอเวอร์โหลดตัวดำเนินการด้านการคำนวณ ประกอบด้วยสัญลักษณ์และฟังก์ชันมีในภาษาโปรแกรม มีรายละเอียดดังตารางที่ 2

ตารางที่ 2 สัญลักษณ์ของฟังก์ชันการโอเวอร์โหลดตัวดำเนินการด้านการคำนวณ

สัญลักษณ์	รูปแบบฟังก์ชัน	ความหมาย
+	__add__(self, other)	การบวก
*	__mul__(self, other)	การคูณ
-	__sub__(self, other)	การลบ
%	__mod__(self, other)	การหารเอาเศษ
/	__truediv__(self, other)	การหาร
//	__floordiv__(self, other)	การหารปัดเศษ

จากสัญลักษณ์และฟังก์ชันการโอเวอร์โหลดตัวดำเนินการด้านการคำนวณ เมื่อนำไปใช้งานในโปรแกรม จะมีรายละเอียดดังโปรแกรมตัวอย่างที่ 10

ตัวอย่างที่ 10 การโอเวอร์โหลดดิ้งตำดำเนินการทางด้านการคำนวณ

บรรทัดที่	คำสั่ง
1	class oper_over:

```
2     def __init__(self,p1,p2,p3):
3        self.v1 = p1
4        self.v2 = p2
5        self.v3 = p3
6     def __str__(self):
7        return '('+str(self.v1)+','+str(self.v2)+','+str(self.v3)+')'
8     def __add__(self,obj):
9        return oper_over(self.v1+obj.v1, self.v2+obj.v2, self.v3+obj.v3)
10    def __sub__(self,obj):
11       return oper_over(self.v1-obj.v1, self.v2-obj.v2, self.v3-obj.v3)
12    def __mul__(self,obj):
13       return oper_over(self.v1*obj.v1, self.v2*obj.v2, self.v3*obj.v3)
14    def __truediv__(self,obj):
15       return oper_over(self.v1/obj.v1, self.v2/obj.v2, self.v3/obj.v3)
16    def __floordiv__(self,obj):
17       return oper_over(self.v1//obj.v1, self.v2//obj.v2, self.v3//obj.v3)
18    def __mod__(self,obj):
19       return oper_over(self.v1%obj.v1, self.v2%obj.v2, self.v3%obj.v3)
20 p1=oper_over(2,3,4)
21 p2=oper_over(5,6,7)
22 print("p1 = ",p1)
23 print("p2 = ",p2)
24 print("p2+p1 =",p1+p2)
25 print("p2-p1 =",p2-p1)
26 print("p2*p1 =",p2*p1)
27 print("p2/p1 =",p2/p1)
28 print("p2//p1 =",p2//p1)
29 print("p2%p1 =",p2%p1)
```

ผลลัพธ์

```
p1 =  (2,3,4)
p2 =  (5,6,7)
p2+p1 = (7,9,11)
```

```
p2-p1 = (3,3,3)
p2*p1 = (10,18,28)
p2/p1 = (2.5,2.0,1.75)
p2//p1 = (2,2,1)
p2%p1 = (1,0,3)
```

การทำงานของโปรแกรมตัวอย่างที่ 10 มีการสร้างคอนสตรัคเตอร์ที่ทำหน้าที่ในการรับค่าพารามิเตอร์ 3 อาร์กิวเมนต์ พร้อมทั้งเรียกฟังก์ชัน __str__ ที่จะแสดงรูปแบบแทนอ็อบเจกต์ต์ ซึ่งถ้าไม่มีฟังก์ชันนี้จะได้ผลลัพธ์ เช่น ในบรรทัดที่ 22 ให้แสงค่าของอ๊อบเจ๊ก ผลลัพธ์ที่ได้คือ p1 = <__main__.oper_over object at 0x00000280BDBC05F8> ดังนั้นฟังก์ชัน __str__ จะแสดงในรูปแบบที่สามารถเข้าใจได้ นอกเหนือจากนี้ยังมีฟังก์ชันโอเวอร์โหลดตัวดำเนินการ ซึ่งจะถูกใช้เมื่ออินสแตนซ์นั้นได้มีการใช้สัญลักษณ์การดำเนินการ เช่น บรรทัดที่ 24 ให้ทำการบวกกันระหว่าง 2 อินสแตนซ์ โปรแกรมจะดำเนินการเรียกใช้ฟังก์ชันที่มีอยู่คลาสที่ทำหน้าที่การบวก พร้อมทั้งเก็บค่าพารามิเตอร์ของของแต่ละอินสแตนซ์ อินสแตนซ์ p1 จะเก็บไว้ที่ตัวแปร self.v1 และเก็บพารามิเตอร์ของ p2 ไว้ที่อ็อบเจกต์ต์ที่สร้างขึ้นมาใหม่ obj.v1 ส่วน self.v2 obj.v2 self.v3 obj.v3 จะมีลักษณะการเก็บค่าแบบเดียวกันและทำการดำเนินการตามการทำงานของสัญลักษณ์ตัวดำเนินการแต่ละตำแหน่งของค่าพารามิเตอร์ เมื่อดำเนินการเสร็จสิ้น จะทำการคืนค่ากลับคืนซึ่งจะได้ผลลัพธ์เป็นตัวเลขที่ได้จากการคำนวณตามตัวดำเนินการที่ใช้ดังกล่าว

8.2.2 การโอเวอร์โหลดตัวดำเนินการในการเปรียบเทียบ ประกอบด้วยสัญลักษณ์และฟังก์ชันมีในภาษาโปรแกรม มีรายละเอียดดังตารางที่ 3

ตารางที่ 3 สัญลักษณ์ของฟังก์ชันการโอเวอร์โหลดตัวดำเนินการ

สัญลักษณ์	รูปแบบฟังก์ชัน	ความหมาย
<	__lt__(self, other)	น้อยกว่า
<=	__le__(self, other)	น้อยกว่าหรือเท่ากับ
==	__eq__(self, other)	เท่ากับ
!=	__ne__(self, other)	ไม่เท่ากับ
>	__gt__(self, other)	มากกว่า
>=	__ge__(self, other)	มากกว่าหรือเท่ากับ

จากสัญลักษณ์และฟังก์ชันการโอเวอร์โหลดตัวดำเนินการด้านการเปรียบเทียบ เมื่อนำไปใช้งานในโปรแกรม จะมีรายละเอียดดังโปรแกรมตัวอย่างที่ 11

ตัวอย่างที่ 11 การโอเวอร์โหลดตัวดำเนินการด้านการเปรียบเทียบ

บรรทัดที่	คำสั่ง
1	class oper_over:
2	def __init__(self,p1,p2,p3):
3	self.v1 = p1
4	self.v2 = p2
5	self.v3 = p3
6	def __str__(self):
7	return '('+str(self.v1)+','+str(self.v2)+','+str(self.v3)+')'
8	def __lt__(self,obj):
9	return oper_over(self.v1<obj.v1, self.v2<obj.v2, self.v3<obj.v3)
10	def __le__(self,obj):
11	return oper_over(self.v1<=obj.v1, self.v2<=obj.v2, self.v3<=obj.v3)
12	def __eq__(self,obj):
13	return oper_over(self.v1==obj.v1, self.v2==obj.v2, self.v3==obj.v3)
14	def __ne__(self,obj):
15	return oper_over(self.v1!=obj.v1, self.v2!=obj.v2, self.v3!=obj.v3)
16	def __gt__(self,obj):
17	return oper_over(self.v1>obj.v1, self.v2>obj.v2, self.v3>obj.v3)
18	def __ge__(self,obj):
19	return oper_over(self.v1>=obj.v1, self.v2>=obj.v2, self.v3>=obj.v3)
20	p1=oper_over(2,6,4)
21	p2=oper_over(5,3,4)
22	print("p1 = ",p1)
23	print("p2 = ",p2)
24	print("p1<p2 =",p1<p2)
25	print("p1<=p2 =",p1<=p2)
26	print("p1==p2 =",p1==p2)

```
27  print("p1!=p2 =",p1!=p2)
28  print("p1>p2 =",p1>p2)
29  print("p1>=p2 =",p1>=p2)
```

ผลลัพธ์

```
p1 =  (2,6,4)
p2 =  (5,3,4)
p1<p2 = (True,False,False)
p1<=p2 = (True,False,True)
p1==p2 = (False,False,True)
p1!=p2 = (True,True,False)
p1>p2 = (False,True,False)
p1>=p2 = (False,True,True)
```

การโอเวอร์โหลดตัวดำเนินการด้านการเปรียบเทียบ ดังโปรแกรมตัวอย่างที่ 11 จะมีลักษณะของการทำงานเช่นเดียวกับการโอเวอร์โหลดตัวดำเนินการด้านการคำนวณ แตกต่างกันตรงที่ ผลลัพธ์ที่ได้จากการเปรียบเทียบ จะเป็นจริงหรือเท็จเท่านั้น

นอกเหนือจากการโอเวอร์โหลดตัวดำเนินการ ยังมีการโอเวอร์โหลดด้านอื่น เช่น __getitem__ ใช้จัดเรียงตัวดำเนินการ __contains__ ใช้ในการตรวจสอบความเป็นสมาชิก __len__ ใช้ในการตรวจสอบจำนวนสมาชิกและยังมีอีกมากมายที่ภาษาไพธอนได้เตรียมไว้ใช้งานในโปรแกรม

9. การสร้างโมดูลคลาสและเรียกใช้โมดูลคลาส

การทำงานของโปรแกรม เพื่อให้ได้ผลลัพธ์ตามที่ต้องการ อาจต้องมีหลายส่วนที่จะทำหน้าที่แตกต่างกันออกไปในการช่วยกันทำงาน ซึ่งลักษณะของการแยกส่วนการทำงานที่มีหน้าที่แตกต่างกันแต่ละส่วนจะถูกสร้างไว้ในลักษณะของคลาส แต่ละคลาสต้องมีการพึงพาอาศัยในการทำงานเพื่อให้ได้ผลลัพธ์ที่ต้องการ เช่น การประมวลผลการเรียน จะต้องสร้างคลาสออกเป็น 3 คลาส ประกอบไปด้วย คลาสที่หนึ่ง ทำหน้าที่ในการรวมคะแนนเก็บ ที่ได้ตามรายจุดประสงค์ คลาสที่สองทำหน้าที่ในการรวมคะแนนทั้งหมดที่ได้ ซึ่งนำคะแนนเก็บรวมกับคะแนนกลางภาคและคะแนนปลายภาค และคลาสที่สาม ทำหน้าที่ในการเอาคะแนนทั้งหมดมาดำเนินการหาเกรดที่ได้ต่อไป ซึ่งทั้งสามคลาสมีการทำงานสัมพันธ์กัน ดังนั้น ควรเก็บคลาสเหล่านี้ไว้ด้วยกันในลักษณะของ โมดูลคลาส

คลาสที่ถูกสร้างเป็นจำนวนมาก จะเก็บกลุ่มคลาสการทำงานที่เหมือนกันไว้ด้วยกันเรียกว่า โมดูล ซึ่งในภาษาไพธอนมีโมดูลที่สนับสนุนการทำงานมากมาย โมดูลเหล่านี้สามารถพัฒนาขึ้นมาใช้งานเองได้ โดยมีขั้นดังต่อไปนี้

1. การสร้างกลุ่มคลาสในโมดูล

การสร้างกลุ่มคลาสในโมดูล มีลักษณะเป็นแฟ้มข้อมูลภาษาไพธอน โดยมีรูปแบบเช่นเดียวกันกับการสร้างชุดคำสั่งโปรแกรในแฟ้มข้อมูลทั่วไป แต่ลักษณะของแฟ้มข้อมูลโปรแกรมนี้จะไม่ถูกประมวลผลโดยตรง แต่จะถูกเรียกใช้งานในแฟ้มข้อมูลโปรแกรมอื่น ซึ่งการถูกเรียกใช้นี้คือโมดูลคลาส ประกอบไปด้วยการทำงานของคลาสหลายคลาสร่วมกัน ดังโปรแกรมตัวอย่างที่ 12

ตัวอย่างที่ 12 การสร้างกลุ่มคลาสในโมดูล

บรรทัดที่	คำสั่ง
1	class cal_point:
2	def __init__ (self,t1,t2,t3,t4,t5,t6):
3	self.point= t1+t2+t3+t4+t5+t6
4	def _cal_point_total(self):
5	return self.point
6	class point_total:
7	def __init__ (self,p_total,midterm,final):
8	self.total = p_total+midterm+final
9	def _cal_total(self):
10	return self.total
11	class cal_grade:
12	def __init__(self,p_total):
13	self.total = p_total
14	if self.total >= 80:
15	self.grade = 'A'
16	elif self.total >= 70:
17	self.grade = 'B'
18	elif self.total >= 60:
19	self.grade = 'C'
20	elif self.total >= 50:

```
21              self.grade = 'D'
22          else:
23              self.grade = 'E'
24      def _check_grade(self):
25          return self.grade
```

โปรแกรมตัวอย่างที่ 12 การทำงานร่วมกันระหว่างคลาส ประกอบไปด้วยคลาส cal_point ทำหน้าทีในการรวมคะแนนเก็บแต่ละหน่วย ประกอบไปด้วย 6 หน่วย คลาส point_total ทำหน้าที่ในการรวมคะแนนเก็บ กลางภาคและปลายภาค และคลาส cal_grade ทำหน้าที่ในการตรวจสอบเกรดที่ได้ ซึ่งทั้งสามคลาสนั้นจะมีความสัมพันธ์กันในการทำงาน ตั้งแต่ขบวนการเริ่มต้น จนสิ้นสุดกระบวนการ ถูกเก็บไว้ในชื่อแฟ้มข้อมูลโปรแกรม grade_process.py เก็บกลุ่มคลาสในโมดูลเพื่อใช้งานต่อไป

2. การเรียกใช้กลุ่มคลาสในโมดูล

คลาสในโมดูลจะถูกเก็บกระบวนการทำงานที่ทำหน้าที่อย่างใดอย่างหนึ่งไว้ การเรียกใช้กลุ่มคลาสที่ถูกสร้างเป็นโมดูลดังกล่าว จะต้องมีการนำเข้ากลุ่มคลาสของแฟ้มข้อมูลโปรแกรมหลัก ซึ่งมีโครงสร้างการทำงานที่จะส่งข้อมูลเข้าไปในกลุ่มคลาสในโมดูล ทำการประมวลผลและให้ได้ผลลัพธ์ออกมาเพื่อนำไปใช้งานในกระบวนการต่อไป ซึ่งจากกลุ่มคลาสในโมดูลโปรแกรมตัวอย่าง 12 ได้มีการสร้างคลาสเก็บไว้ใช้งาน วิธีการเรียกใช้กลุ่มคลาส มีขั้นตอนและลักษณะการทำงาน ดังโปรแกรมตัวอย่างที่ 13

ตัวอย่างที่ 13 การเรียกใช้กลุ่มคลาสในโมดูล

บรรทัดที่	คำสั่ง
1	import grade_process
2	p1=int(input('คะแนนหัวข้อที่ 1 = '))
3	p2=int(input('คะแนนหัวข้อที่ 2 = '))
4	p3=int(input('คะแนนหัวข้อที่ 3 = '))
5	p4=int(input('คะแนนหัวข้อที่ 4 = '))
6	p5=int(input('คะแนนหัวข้อที่ 5 = '))
7	p6=int(input('คะแนนหัวข้อที่ 6 = '))
8	mid=int(input('คะแนนกลางภาค = '))
9	fin=int(input('คะแนนปลายภาค = '))

```
10  obj1=grade_process.cal_point(p1,p2,p3,p4,p5,p6)
11  print('คะแนนเก็บระหว่างภาคเรียน = ',obj1._cal_point_total())
12  obj2=grade_process.point_total(obj1._cal_point_total(),mid,fin)
13  print('คะแนนเก็บ + คะแนนกลางภาค + คะแนนปลายภาค = ',obj2._cal_total())
14  obj3=grade_process.cal_grade(obj2._cal_total())
15  print('เกรดที่ได้ = ',obj3._check_grade())
```

ผลลัพธ์

```
คะแนนหัวข้อที่ 1 = 5                                    (รับข้อมูลจากแผงแป้นอักขระ)
คะแนนหัวข้อที่ 2 = 10                                   (รับข้อมูลจากแผงแป้นอักขระ)
คะแนนหัวข้อที่ 3 = 10                                   (รับข้อมูลจากแผงแป้นอักขระ)
คะแนนหัวข้อที่ 4 = 5                                    (รับข้อมูลจากแผงแป้นอักขระ)
คะแนนหัวข้อที่ 5 = 2                                    (รับข้อมูลจากแผงแป้นอักขระ)
คะแนนหัวข้อที่ 6 = 2                                    (รับข้อมูลจากแผงแป้นอักขระ)
คะแนนกลางภาค = 12                                       (รับข้อมูลจากแผงแป้นอักขระ)
คะแนนปลายภาค = 15                                       (รับข้อมูลจากแผงแป้นอักขระ)
คะแนนเก็บระหว่างภาคเรียน =  34
คะแนนเก็บ + คะแนนกลางภาค + คะแนนปลายภาค =  61
เกรดที่ได้ =  C
```

แฟ้มข้อมูลหลักที่ใช้ประมวลผลการทำงานของโปรแกรม กระบวนการทำงานของโปรแกรมประกอบไปด้วยการนำเข้าโมดูล grade_process.py และสร้างอ๊อบเจกต์ obj1 obj2 obj3 ที่เกิดจากคลาสในโมดูล grade_process โดย obj1 จะทำการนำค่าที่รับเข้ามาและส่งค่าพารามิเตอร์ไปยังคลาส cal_point ในโมดูล เพื่อให้คอนสตรัคเตอร์ของคลาส cal_point ทำการคำนวณหาคะแนนเก็บรวมและส่งค่าที่ได้กลับคืนมายังอ็อบเจกต์ obj1 ผ่านเมธอด _cal_point_total() อ๊อบเจกต์ obj2 ทำหน้าที่ในการส่งค่าพารามิเตอร์ที่ได้จาก obj1 และค่าที่รับเข้ามาส่งเข้าไปยังคอนสตรัคเตอร์คลาส point_total นำค่าที่ได้ไปทำการคำนวณและคืนค่าผ่านเมธอด _cal_total() และอ๊อบเจกต์ obj3 ทำหน้าที่ในการส่งค่าพารามิเตอร์ที่ได้จาก obj2 ไปยังคอนสตรัคเตอร์ของคลาส cal_grade ซึ่งจะทำการประมวลผลเกรดที่ได้และคืนค่ากลับมายัง obj3 ผ่านเมธอด _check_grade() ดังผลลัพธ์ที่ได้ในโปรแกรมโปรแกรมตัวอย่างที่ 13

สรุป

เชิงวัตถุ คือ แนวคิดที่มองภาพความเป็นจริงที่อยู่บนโลกนี้ ในรูปแบบของวัตถุ นำแนวคิดที่ได้มาออกแบบการเขียนโปรแกรม ที่เรียกว่า การเขียนโปรแกรมเชิงวัตถุ ซึ่งแนวคิดนี้จะประกอบไปด้วยองค์ประกอบหลัก 4 ส่วนประกอบไปด้วย คลาส คุณสมบัติ พฤติกรรม และวัตถุ ซึ่งคลาสเป็นการนิยามขอบเขตของวัตถุ ซึ่งนิยามจากคุณสมบัติและพฤติกรรมที่มี ส่วนวัตถุก็คือสิ่งที่มีตัวตน มีคุณสมบัติและพฤติกรรมเหมือนกับคลาสที่วัตถุเกิดจากคลาสนั้น การเขียนไปรแกรมเชิงวัตถุ จะมีนิยามศัพท์บางส่วนที่แตกต่างกันกับแนวคิด แต่มีความหมายเดียวกัน เช่น วัตถุ เมื่อนำเขียนโปรแกรมจะเรียกว่าอินสแตนซ์หรืออ๊อบเจ๊ก คุณสมบัติจะเรียกว่า แอททริบิวต์ และการกระทำเรียกว่า เมธอด

หลักการทำงานของโปรแกรมเชิงวัตถุมีลักษณะของการใช้งานหลากหลายรูปแบบ แนวคิดเรื่องเกี่ยวกับการใช้งานภายในคลาส ประกอบด้วย การอนุญาตการเข้าถึงข้อมูลและเมธอดภายในคลาสรูปแบบการห่อหุ้มข้อมูล การซ่อนข้อมูลหรือการเข้าถึงคลาส การเรียกใช้เมธอดอัตโนมัติเมื่อมีการสร้างและลบอินสแตนซ์ รูปแบบของคอนสตรัคเตอร์และดีสตรัคเตอร์ การส่งผ่านค่าระหว่างอ๊อบเจกต์กับเมธอด รูปแบบของการส่งค่าและคืนค่าพารามิเตอร์ การที่อ๊อบเจกต์ได้รับแอททริบิวต์และเมธอดจากคลาสแม่และคลาสลูกรูปแบบของการสืบทอดคลาส การใช้งานเมธอดหรือฟังก์ชันที่เหมือนกันแต่การทำงานต่างกันระหว่างคลาสแม่และคลาสลูกรูปแบบของโอเวอร์ไรดิ้ง การทำงานของเมธอดที่ชื่อเหมือนกันแต่การทำงานต่างกันตามจำนวนอาร์กิวเมนต์รูปแบบของโอเวอร์โหลดดิ้งเมธอด การอนุญาตให้อ๊อบเจกต์ที่สร้างภายในคลาสสามารถใช้งานตัวดำเนินการพื้นฐานรูปแบบของโอเวอร์โหลดดิ้งตัวดำเนินการ และการสร้างกลุ่มคลาสแยกไว้ในแฟ้มข้อมูลโปรแกรมและสามารถถูกเรียกใช้คลาสที่แฟ้มข้อมูลโปรแกรมอื่นรูปแบบของการสร้างโมดูลคลาสและการเรียกใช้โมดูลคลาส

นอกเหนือจากนี้ภาษาไพธอน ยังได้เตรียมเครื่องมือจำนวนมาก ที่สนับสนุนการเขียนโปรแกรมเชิงวัตถุ แนวคิดและหลักการของเชิงวัตถุ ทุกภาษาที่สนับสนุนการเขียนโปรแกรมเชิงวัตถุจะใช้แนวคิดและหลักการเหมือนกัน แต่รูปแบบการเขียนอาจแตกต่างกัน เมื่อมีความรู้ความเข้าใจในเรื่องเชิงวัตถุ ก็จะสามารถใช้งานของภาษาโปรแกรม เพื่อเขียนโปรแกรมรองรับกับการทำงานของโปรแกรมที่มีความซับซ้อนให้มีประสิทธิภาพการทำงานขั้นสูงได้

แบบฝึกหัด

1. ให้ผู้เรียนอธิบายความแตกต่างระหว่างการเขียนโปรแกรมแบบเชิงวัตถุและแบบโครงสร้าง พร้อมทั้งอธิบายข้อดีและข้อเสียของแต่รูปแบบการเขียนโปรแกรม

2. ให้ผู้เรียนบอกองค์ประกอบของเชิงวัตถุประกอบด้วยอะไรบ้างและแต่ละส่วนมีลักษณะอย่างไร

3. ให้ผู้เรียนอธิบายลักษณะของการเข้าถึงคลาสมีลักษณะอย่างไร พร้อมทั้งอธิบายแต่ละรูปแบบของการเข้าถึงคลาส

4. ให้ผู้เรียนบอกส่วนประกอบภายในคลาสมีอะไรบ้าง พร้อมทั้งอธิบายแต่ละส่วนประกอบ

5. ให้ผู้เรียนอธิบายเรียงลำดับขั้นตอนการเขียนโปรแกรมเชิงวัตถุ พร้อมทั้งเขียนโปรแกรมประกอบการอธิบาย

6. ให้ผู้เรียนอธิบายการทำงานและความแตกต่างระหว่างคอนสตรัคเตอร์และดีสตรัคเตอร์ พร้อมทั้งเขียนโปรแกรมประกอบการอธิบาย

7. ให้ผู้เรียนอธิบายวิธีความหมายของพารามิเตอร์และวิธีการส่งค่าพารามิเตอร์ พร้อมทั้งเขียนโปรแกรมประกอบการอธิบาย

8. ให้ผู้เรียนอธิบายความหมายของการสืบทอดคลาส พร้อมทั้งเขียนโปรแกรมประกอบการอธิบาย

9. ให้ผู้เรียนอธิบายลักษณะของโอเวอร์ไรดิ้งและโอเวอร์โหลดดิ้ง พร้อมทั้งเขียนโปรแกรมประกอบการอธิบาย

10. ให้ผู้เรียนอธิบายการสร้างโมดูลคลาสและการเรียกใช้โมดูลคลาส พร้อมทั้งเขียนโปรแกรมประกอบการอธิบาย

11. ให้ผู้เรียนเขียนโปรแกรมให้มีส่วนการทำงาน คือ ส่วนที่ 1 รวมราคาสินค้าทั้งหมด, ส่วนที่ 2 คำนวณส่วนลด และส่วนที่ 3 ให้หาราคาเงินที่จ่ายค่าสินค้า โดยให้เขียนโปรแกรมอยู่ในรูปของการเขียนโปรแกรมเชิงวัตถุ

12. จากข้อที่ 11 ให้ผู้เรียนประยุกต์การเขียนโปรแกรมในรูปแบบของโมดูลคลาส

13. ให้ผู้เรียนเขียนโปรแกรมแปลง องศาเซลเซียสเป็นองศาฟาเรนไฮต์ องศาเซลเซียสเป็นเคลวิน องศาฟาเรนไฮต์เป็นองศาเซลเซียส องศาฟาเรนไฮต์เป็นเคลวิน เคลวินเป็นองศาเซลเซียส และเคลวินเป็นองศาฟาเรนไฮต์ โดยให้เขียนโปรแกรมอยู่ในรูปของการเขียนโปรแกรมเชิงวัตถุ

14. จากข้อที่ 13 ให้ผู้เรียนประยุกต์การเขียนโปรแกรมในรูปแบบของโมดูลคลาส

15. ให้ผู้เรียนยกตัวอย่างเขียนโปรแกรมเชิงวัตถุที่มีองค์ประกอบ คลาส คุณสมบัติ พฤติกรรม และวัตถุ ประยุกต์การใช้งานในปัจจุบัน