01076105, 01076106

Object Oriented Programming

Object Oriented Programming Project

Object and Class

Software Development Life Cycle

- วงจรการพัฒนาซอฟต์แวร์

Software Development Life Cycle

- Requirements คือ การหาความต้องการของซอฟต์แวร์ เป้าหมายเพื่อจะตอบคำถามว่า ซอฟต์แวร์นี้ใช้สำหรับทำอะไร และทำงานอะไรได้บ้าง (What?)
  Output ของขั้นตอน Requirement คือ Software Specification ซึ่งโดยทั่วไปจะอยู่ในรูปแบบของข้อความ
- Analysis คือ การนำเอาความต้องการมาวิเคราะห์ ซึ่งการวิเคราะห์จะมีหลายระดับ ตั้งแต่วิเคราะห์ว่าซอฟต์แวร์ควรมีโครงสร้างการทำงานอย่างไร มีขั้นตอนการทำงานอย่างไร อาจรวมไปถึงมีส่วนติดต่อผู้ใช้แบ่งเป็นกี่ส่วน โดยส่วนขั้นตอนการทำงานหรือมักเขียนในรูปแบบ Use Case Diagram และ Use Case Description (How?)
  Output ของขั้นตอนนี้ คือ Use Case Diagram

Software Development Life Cycle

- Design คือ การนำเอาการวิเคราะห์มาออกแบบ โดยแบ่งออกเป็นการออกแบบส่วนติดต่อผู้ใช้ (User Interface Design) และการออกแบบโครงสร้างการทำงานของโปรแกรม (Software Design)
  โดยทั่วไปขั้นตอน Analysis กับ Design มักทำควบคู่กัน เรียกว่า Analysis and Design
  Output ของการออกแบบมักอยู่ในรูปของ Diagram โดยส่วนของโครงสร้างการทำงานในแบบ Object Oriented จะได้เป็น Class Diagram, Sequence Diagram และอื่นๆ ในส่วนของ UI Design ก็จะได้เป็น Wireframe หรือ UI Screen
- Coding คือ การนำเอาการออกแบบที่ได้ทำไว้ มาเขียนเป็นซอฟต์แวร์ โดยให้มีผลการทำงานตามที่ออกแบบไว้ โดยอาจมีรูปแบบการพัฒนาหลายแบบ เช่น Agile, Scrum, Extreme Programming, Lean หรืออื่นๆ

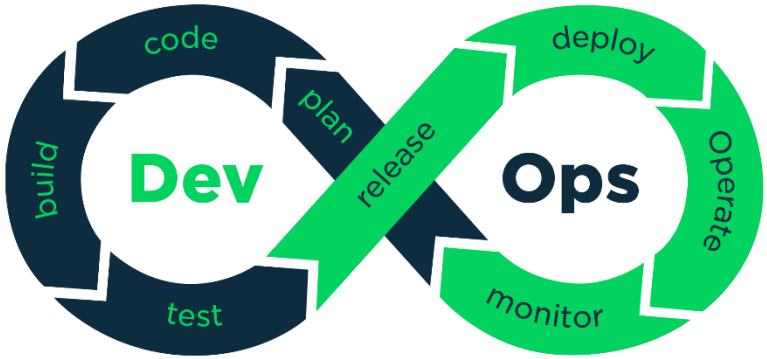
Software Development Life Cycle

- Testing คือ ขั้นตอนการทดสอบโปรแกรม ซึ่งในปัจจุบัน ถือได้ว่าเป็นขั้นตอนที่มีความสำคัญมาก ขนาดที่มีตำแหน่งงานที่ทำเรื่องการทดสอบโปรแกรมโดยเฉพาะ การทดสอบโปรแกรมแบ่งออกเป็น
  - Unit Testing คือการทดสอบระดับฟังก์ชันหรือคลาส
  - Integrate Testing คือการทดสอบเมื่อนำโปรแกรมมารวมกัน
  - User Acceptance Test (UAT) คือ การทดสอบในรูปแบบที่มีการจำลองการใช้งานจริง

  ปัจจุบัน Testing มีการพัฒนาเป็น Automate Testing คือ มีซอฟต์แวร์สำหรับช่วยการทดสอบโดยเฉพาะ โดยมีการจัดทำเป็น Test Script

Software Development Life Cycle

- Deployment คือ ขั้นตอนการนำซอฟต์แวร์ไปสู่การใช้งานจริง ตั้งแต่การติดตั้งเครื่องคอมพิวเตอร์ การติดตั้งซอฟต์แวร์พื้นฐาน จนถึงการติดตั้งซอฟต์แวร์ที่พัฒนา ในปัจจุบันงานด้านการ Deployment ได้พัฒนาไปมาก เนื่องจากการแข่งขันทางธุรกิจทำให้ความต้องการ**ความถี่**ของการ Deployment เพิ่มขึ้นอย่างมาก ซอฟต์แวร์บางตัวอาจมีการ Deploy ทุกวัน เนื่องจากมีการเพิ่ม feature ต่างๆ เข้าไปในซอฟต์แวร์ และ**เวลา**ที่ใช้ในการ Deploy ต้องสั้นด้วย เช่น Netflix สามารถ Deploy ได้ภายใน 1 นาที การจะบรรลุทั้งความถี่และเวลาที่ใช้ในการ Deploy ต้องอาศัยการทำงานที่เรียกว่า DevOps ซึ่งจะหาวิธีการร่วมกันระหว่าง Coder, Tester และ DevOps เพื่อให้การปรับปรุงซอฟต์แวร์ทำได้เร็วและบ่อยตามต้องการได้



ขอบเขตของวิชา

- วิชานี้จะถือว่ามีความต้องการอยู่แล้ว โดยจะใช้ User Interface เป็นเครื่องมือที่บอกว่าซอฟต์แวร์จะต้องทำอะไรได้บ้าง
- เนื้อหาจะครอบคลุมการทำงาน Analysis, Design และ Coding
- การเรียนจะเป็น Project Based โดยนักศึกษาจะต้องใช้ Application ที่นักศึกษาเลือกมาวิเคราะห์ ออกแบบ และ เขียนโปรแกรม
- โดยค่อยๆ ทำไปทีละขั้นตอนจนถึงปลายเทอมจะได้ซอฟต์แวร์ 1 ชิ้นที่ออกแบบตามหลักการ Object Oriented

หลักการสำคัญของ Object Oriented

- Object Oriented Programming มีหลักการสำคัญอยู่ 4 ข้อ
  1. **Encapsulation** เป็นหลักการ modular คือแบ่งโปรแกรมเป็นส่วนย่อย เพื่อให้ความซับซ้อนโดยรวมลดลง โดย OOP จะนำ data และ code ที่เกี่ยวกับเรื่องใดเรื่องหนึ่งมารวมไว้ด้วยกันโดยเรียกว่า object และป้องกันไม่ให้ผู้อื่นมายุ่งกับข้อมูล
  2. **Abstraction** คือ การแยกระหว่าง ส่วนที่ให้ผู้อื่นมองเห็น (Interface) กับการทำงาน (Implementation) ของส่วนนั้น ยกตัวอย่างเช่น บริการดึงข้อมูลอุณหภูมิ กับการทำงานของบริการนั้น กล่าวคือ ผู้ที่เรียกใช้บริการไม่จำเป็นต้องรู้ว่าบริการนั้นทำงานอย่างไร เพียงแต่รู้วิธีการเรียกใช้
  3. **Inheritance** คือ การถ่ายทอดคุณสมบัติของ object ที่มีความคล้ายคลึงกัน
  4. **Polymorphism** คือ การใช้ Interface ที่เหมือนกันกับ object ที่ต่างกัน

ประโยชน์ของ Object Oriented

- การ reuse code จะทำได้ดีกว่า
- การขยายหรือแก้ไขโปรแกรมจะทำได้ง่ายกว่า
- การพัฒนา การทดสอบ และ การดูแลรักษาทำได้ง่ายกว่า
- เหมาะสมกับการพัฒนา software ขนาดใหญ่มากกว่า
- การอธิบายการทำงานให้กับ non technical จะทำได้ง่ายกว่า

Case Study

- ตัวอย่าง application ที่ใช้ คือ โรงภาพยนตร์ เริ่มจากการค้นหา object
- จากหน้ารายละเอียดภาพยนตร์ตามรูป

Case Study

- ภาพยนตร์ มีความเป็น object ชัดเจน จึงกำหนดให้เป็น object ของภาพยนตร์ และตั้งชื่อว่า Movie
- คุณลักษณะของ object ที่เห็นจาก user interface คือ
  - ภาพใบปิดภาพยนตร์ (poster)
  - ชื่อภาพยนตร์
  - รายละเอียด
  - หมวดหมู่
  - Rating
  - ความยาวภาพยนตร์

Class

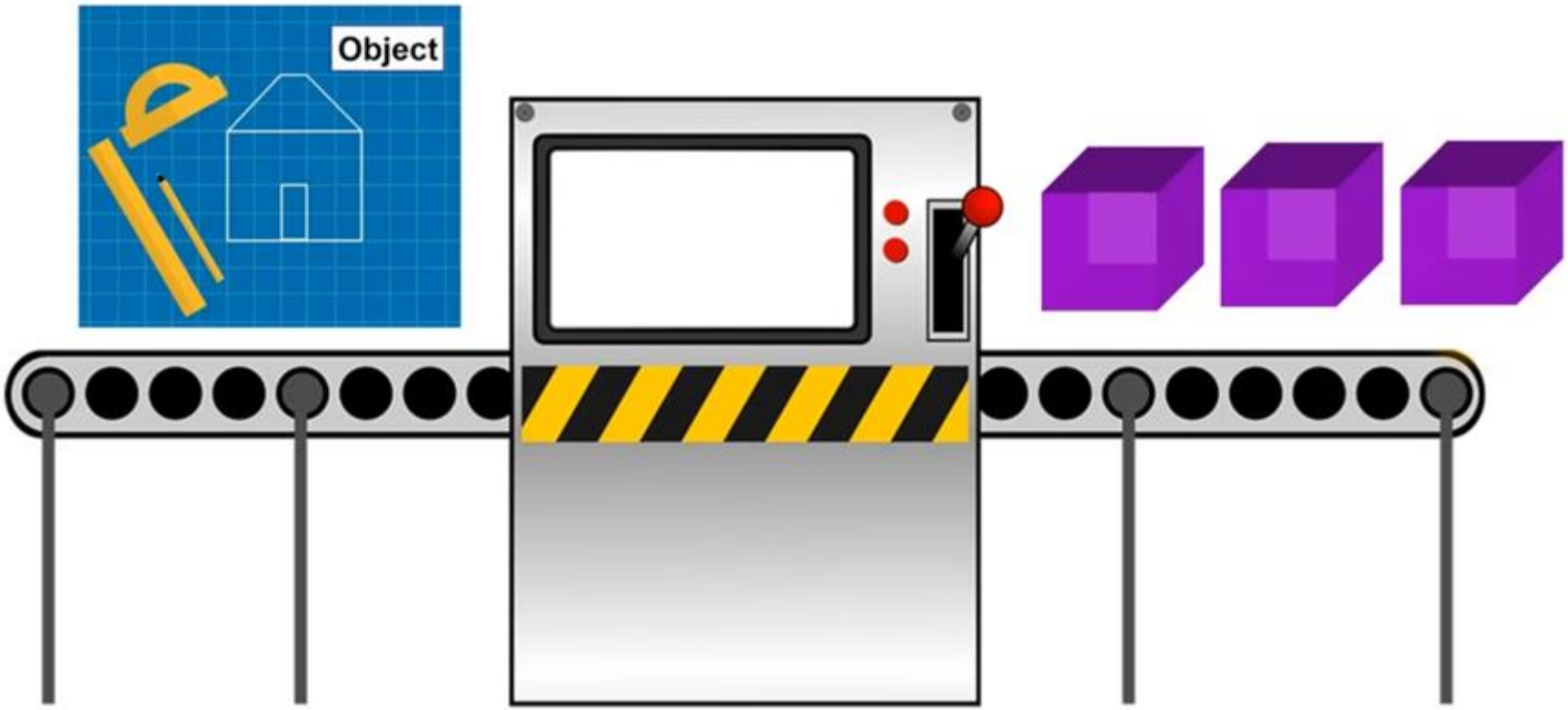
- การจะสร้าง Object จะต้องมี Class ก่อน

> A **blueprint** for creating objects.

- คลาสทำหน้าที่คล้าย “พิมพ์เขียว” ของ object โดยจะต้องกำหนดรายละเอียดของ Class ก่อน จึงจะสามารถสร้าง object ได้
- ภายในคลาสจะประกอบด้วย attribute (คุณลักษณะ) ของ Object นั้น และ behavior ที่ Object นั้นสามารถทำได้

Class

- Class เป็นต้นแบบในการผลิต object



Class

- ในภาษา Python เราจะใช้ Pascal Case (หรือ Upper camel case) ในการกำหนดชื่อ Class
- การเขียนในแบบ Pascal Case คือ ให้ขึ้นต้นตัวแรกด้วยอักษรตัวใหญ่ ของแต่ละคำ เช่น
  - House
  - BackAccount
  - Movie
  - Seat

Class

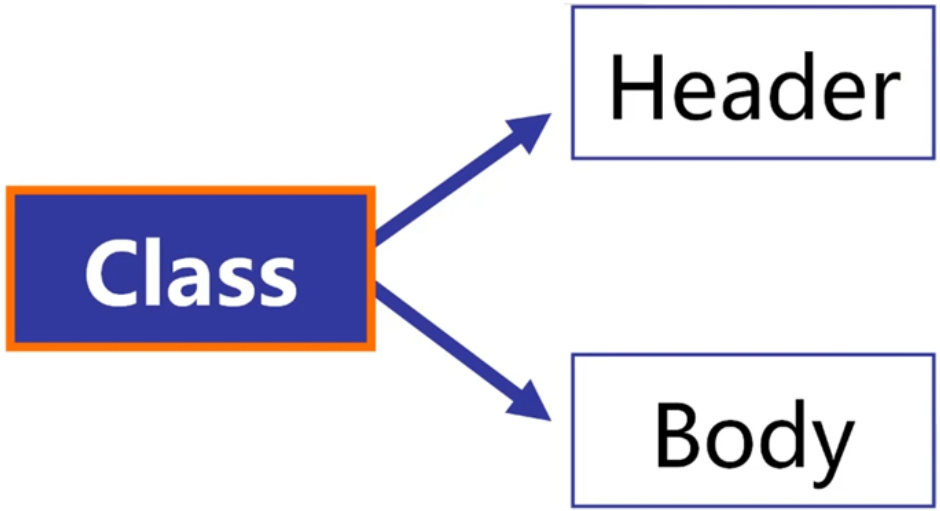
- การตั้งชื่อ Class มักจะใช้เป็นคำนาม เพราะคลาสเป็น object

```
class <ClassName>(object):
```

- การกำหนด Class จะเขียนคำว่า class เป็นตัวเล็ก และใช้ชื่อ class เป็น Pascal Case และปิดท้ายด้วยเครื่องหมาย :
- สำหรับ (object) ภาษา python ตั้งแต่ 3.0 ขึ้นไป ไม่ต้องเขียนก็ได้
- เช่น
  - class Movie:
  - class Seat:

Class

- โครงสร้าง Class ใน python ประกอบด้วย 2 ส่วน คือ Header และ Body
- Header ทำหน้าที่บอกข้อกำหนดของ Class
- Body บอกรายละเอียดภายใน Class



Class

- Body ของคลาสบรรจุรายละเอียดของ “blueprint” ซึ่งประกอบด้วย attribute และ behavior ของ object
- ส่วนประกอบของ Class body มักจะประกอบด้วย
  - Class Attribute เป็นส่วนที่เก็บ Data ของ Class
  - __init__() เป็น Constructor คือ ส่วนที่จะถูกเรียกขึ้นมาทำงาน เมื่อมีการสร้าง object
  - Methods ทำหน้าที่อธิบาย behavior หรือ action ของคลาสที่จะมีในกรณีต่างๆ

```
class ClassName:

    # Class Attributes

    # __init__()

    # Methods
```

Class

- ตัวอย่าง การกำหนด class

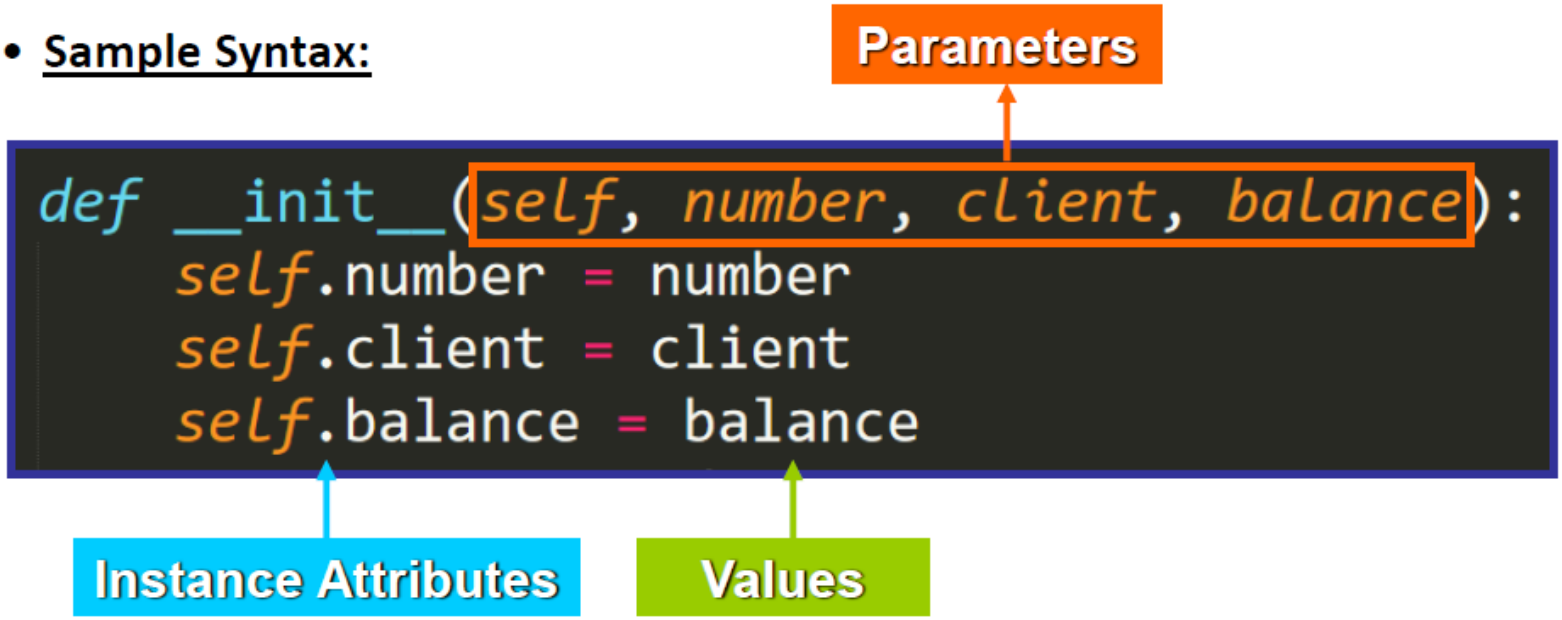
```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
```

- จากภาพเป็นคลาสของจุด โดย self.x และ self.y จะเรียกว่า attribute ซึ่งจะต่างจากตัวแปรทั่วไป คือ attribute เป็นตัวแปรที่ผูกกับคลาสนั้นๆ เท่านั้น โดย attribute จะมีคำว่า self. นำหน้า โดยมีความหมายว่าเป็นตัวแปรของ class
- โดย def __init__(self, x, y) เป็นฟังก์ชัน หรือ method พิเศษ เรียกว่า constructor มีหน้าที่สร้างค่าเริ่มต้นให้กับคลาส โดยกรณีนี้จะรับพารามิเตอร์จำนวน 2 ตัว คือ x และ y จากนั้นจึงมากำหนดค่าให้กับ attribute self.x และ self.y

Class

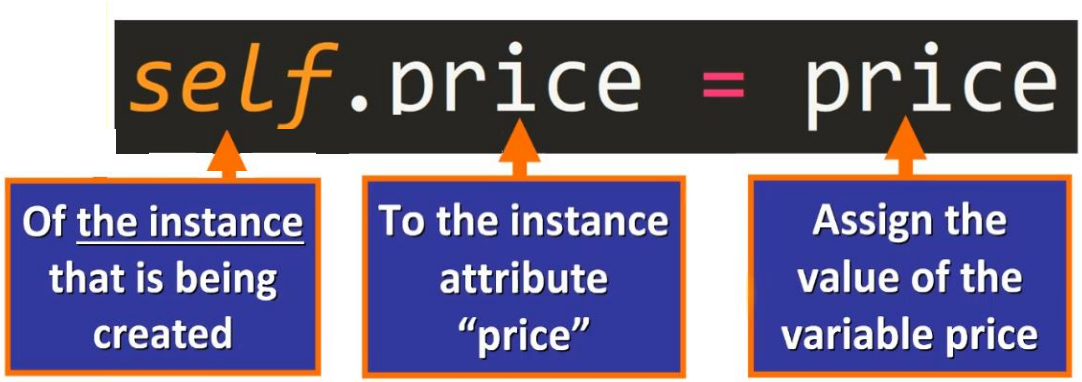
- constructor มีส่วนประกอบดังนี้
  - Parameters เป็นข้อมูลที่ส่งมาเป็นค่าเริ่มต้นของคลาส
  - Instance attribute เป็นข้อมูลของคลาส อาจจะไม่เท่ากับ parameters ก็ได้
  - Values เป็นค่าของ parameter ที่จะมาเป็นค่าเริ่มต้นให้กับ instance attribute

- **Sample Syntax:**



Class

- สิ่งที่สร้างขึ้นมาจาก blueprint ของคลาส อาจเรียกว่า object ก็ได้ แต่ต่อไปจะเรียกว่า Instance เพราะคำว่า object ไม่ชัดเจนเท่ากับ instance
- รูปแบบทั่วไปของการกำหนดค่าเริ่มต้นให้กับ attribute มีดังนี้



- price ทั้ง 2 ตัวต่างกัน ฝั่งขวาคือ พารามิเตอร์ที่ส่งเข้ามา ฝั่งซ้ายคือ instance attribute

Case Study

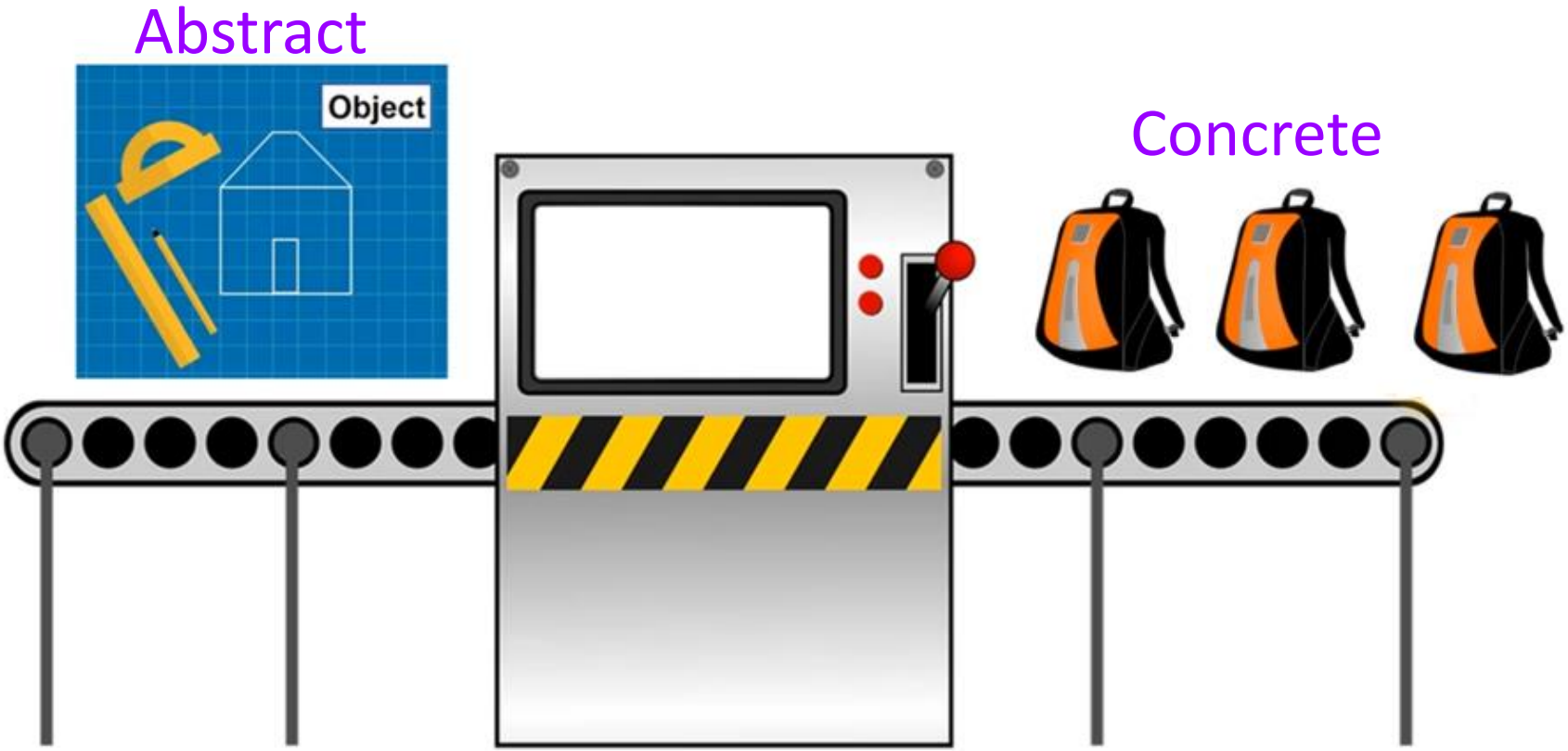
- จาก Slide 11 เราสามารถเขียน Class Movie ในภาษา Python ได้ดังนี้ (ไม่รวมโปสเตอร์ เพื่อให้ข้อมูลเป็น text ทั้งหมด)

```
class Movie:
    def __init__(self, name, detail, genre, \
                 rating, length, release_date):
        self.name = name
        self.detail = detail
        self.genre = genre          # หมวดหมู่
        self.rating = rating
        self.length = length
        self.release_date = release_date
```

- จากที่ได้กล่าวไปแล้วว่าคลาสเป็นต้นแบบในการผลิต Instance ดังนั้นจะต้องนำคลาสไปสร้างเป็น Instance อีกที

Instance

- สิ่งที่สร้างมาจาก Class จะเรียกว่า Instance ซึ่งจะมีที่อยู่แน่นอนในหน่วยความจำ ของแต่ละ Instance แยกกันออกไป ดังนั้นแต่ละ Instance จึงเป็นตัวของตัวเอง เพียงแต่มาจากต้นแบบเดียวกัน



Instance

- จาก Class Movie สามารถสร้าง Instance ในภาษา Python ได้ดังนี้

```
avatar = Movie('Avatar',
               'pandora',
               ['Action', 'Adventure', 'Fantasy'],
               'PG-13',
               190,
               '14/12/2022')
wakanda = Movie('Black Panther : Wakanda Forever',
               'wakanda',
               ['Action', 'Adventure', 'Drama'],
               'PG-13',
               160,
               '09/11/2022')
```

Instance

- รูปแบบทั่วไปของการสร้าง Instance คือ

```
<variable> = <ClassName>(<arguments>)
```

```
my_account = BankAccount("5621", "Gino Navone", 33424.4)
```

```
class BankAccount:

    accounts_created = 0

    def __init__(self, number, client, balance):
        self.number = number
        self.client = client
        self.balance = balance
        BankAccount.accounts_created += 1
```

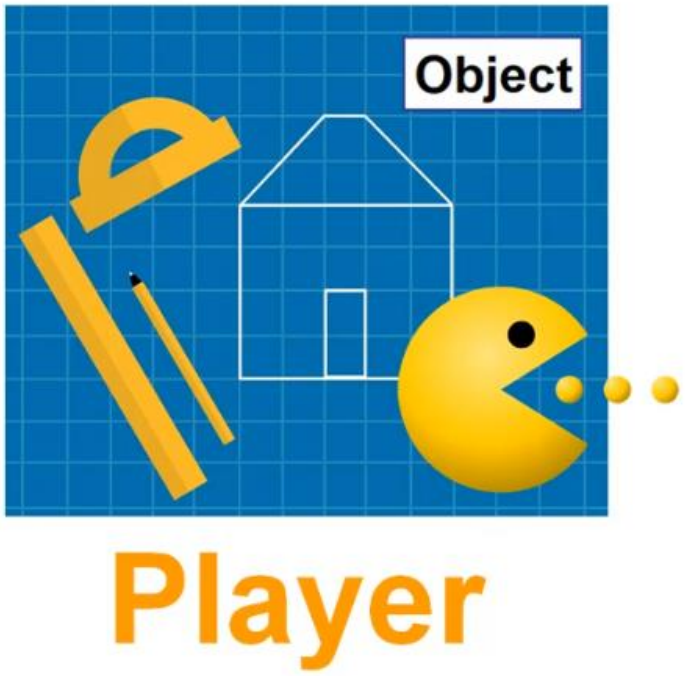
Instance

- แต่ละ Instance จะมีที่อยู่แยกกันในหน่วยความจำ แม้จะสร้างจากคลาสเดียวกัน แต่เมื่อสร้างขึ้นมาแล้ว จะเป็นข้อมูลที่แยกกันโดยเด็ดขาด

avatar	wakanda
'Avatar'	'Black Panther : Wakanda Forever'
'pandora'	'wakanda'
['Action', 'Adventure', 'Fantasy']	['Action', 'Adventure', 'Drama']
'PG-13'	'PG-13'
190	160
'14/12/2022'	'09/11/2022'

Instance

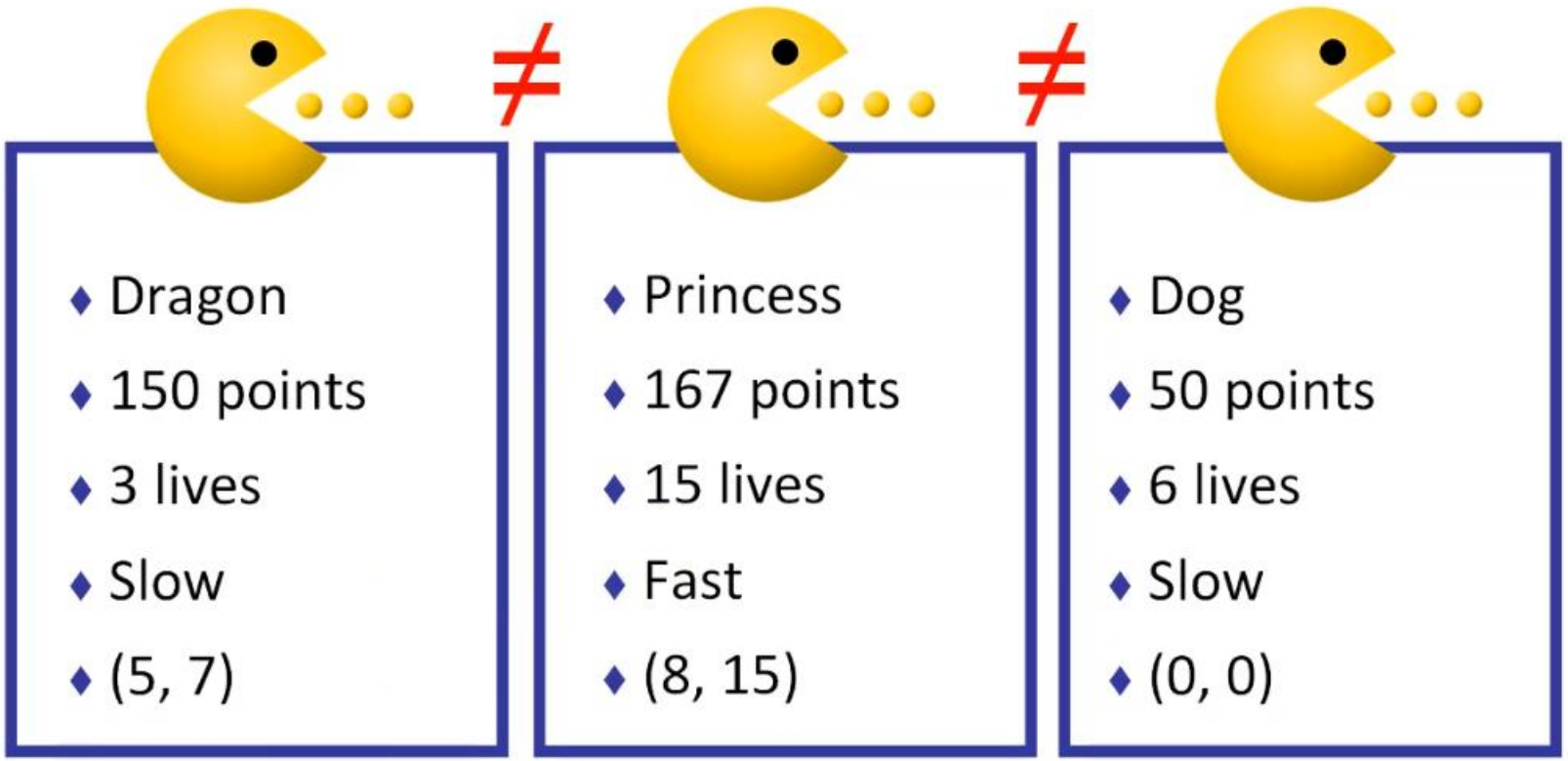
- อีกตัวอย่างของ Instance



- Sprite
- Score
- Number of lives
- Speed
- X-coordinate
- Y-coordinate

Instance

- ในตัวละครของเกม เมื่อมีการเปลี่ยนตำแหน่ง หรือ เปลี่ยนความเร็ว ก็จะมีผลเฉพาะ Instance นั้น

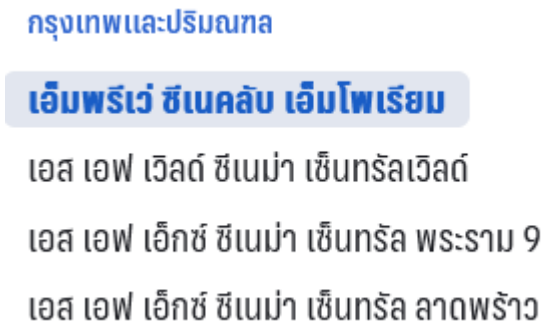


Instance

- ข้อผิดพลาดที่มักเกิดขึ้นกับ __init__()
  - ลืมเขียน def
  - ใช้เครื่องหมาย _ แค่ อันเดียว เช่น _init_()
  - ลืมเขียน self ใน parameter แม้ใน object ที่ไม่มี parameter เลยก็ต้องมี self เช่น
    - def __init__(width, height):
  - ลืมเขียน self.<attribute> ในการกำหนด instance attributes
  - PEP8 Style ต้องวรรค 1 เคาะระหว่าง parameter เช่น
    - def __init__(self, name, age):

Case Study

- ต่อไปจะสร้าง Class อื่นๆ ของระบบจองตั๋วภาพยนตร์ต่อ
- เนื่องจากภาพยนตร์ต้องฉายในโรงภาพยนตร์ ดังนั้นจะต้องมีคลาส Cinema



- โดยมี attribute คือ ชื่อโรง, จำนวนโรงย่อย และ สถานที่ตั้ง
- โรงย่อยของแต่ละ Cinema จะกำหนดเป็นคลาสชื่อ CinemaHall
- โดยมี attribute คือ หมายเลข และ จำนวนที่นั่งในโรง

Case Study

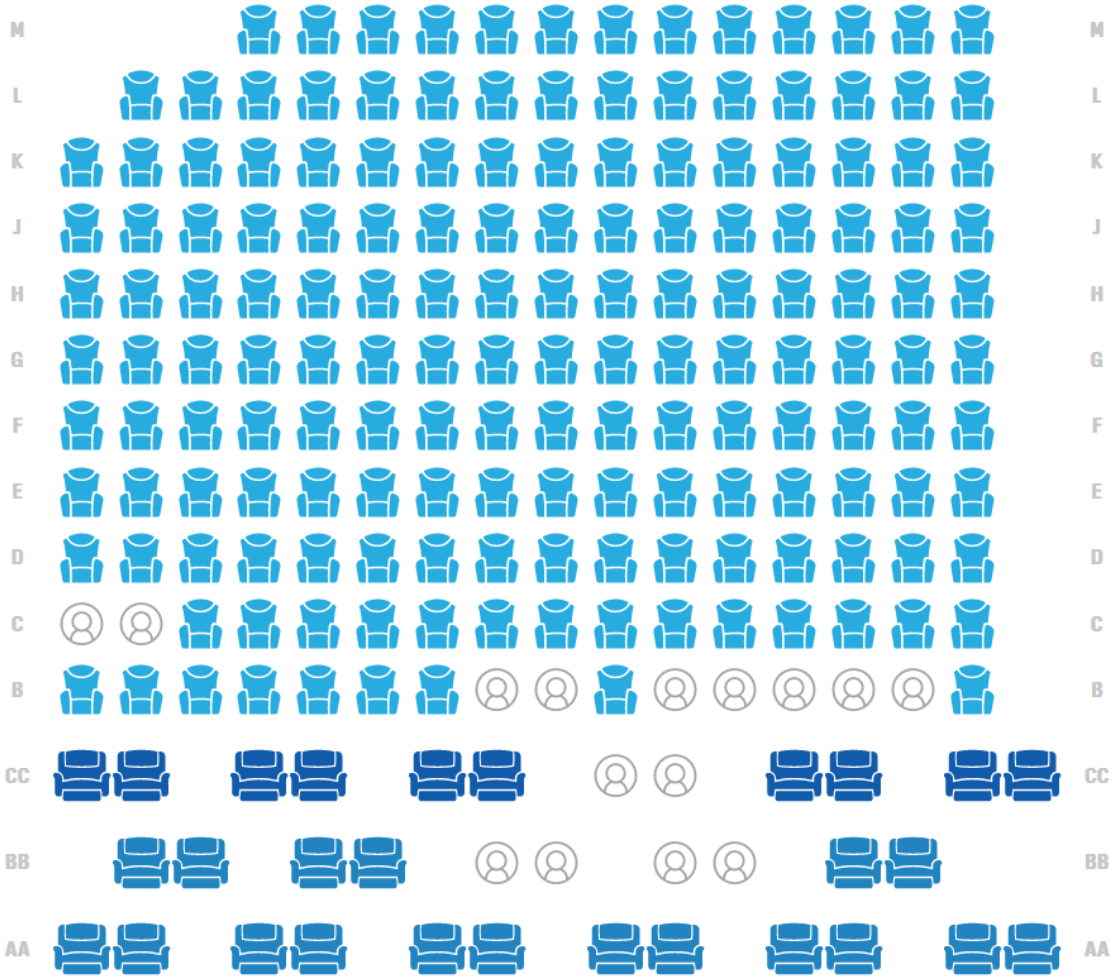
- สามารถเขียน Class เบื้องต้นในภาษา Python ได้ดังนี้

```
class Cinema:
    def __init__(self, name, total_cenema_hall, location):
        self.name = name
        self.total_cenema_hall = total_cenema_hall
        self. location = location

class CinemaHall:
    def __init__(self,name, total_seat):
        self.name = name
        self.total_seat = total_seat
```

Case Study

- ในโรงภาพยนตร์ จะต้องมีที่นั่ง และ มีที่นั่งหลายประเภท



Case Study

- จะกำหนดชื่อคลาสเป็น CinemaHallSeat โดยมี attribute ประกอบด้วย seat_row, seat_column, seat_type
- มีประเด็นเรื่องราคาตั๋ว ที่เกี่ยวข้องกับที่นั่ง โดยจะเปลี่ยนไปตามประเภทที่นั่ง ดังนั้นจึงควรพิจารณาว่าราคาตั๋วควรเป็น attribute ของ CinemaHallSeat หรือไม่
- คำตอบ คือ ได้ แต่เมื่อพิจารณาต่อว่าเมื่อนำราคาตั๋วมาเป็น attribute จะทำให้เกิดข้อมูลซ้ำเป็นจำนวนมาก เพราะทุกที่นั่งจะต้องมีราคาประกบติดตลอด ซึ่งความซ้ำซ้อนเป็นสิ่งที่พึงหลีกเลี่ยงในการออกแบบ
- และเมื่อพิจารณาให้ละเอียด ราคาตั๋วไม่ได้ผูกอยู่กับที่นั่ง แต่ผูกอยู่กับประเภทที่นั่ง จึงควรพิจารณาต่อไปว่าควรมีคลาสประเภทที่นั่งหรือไม่

Case Study

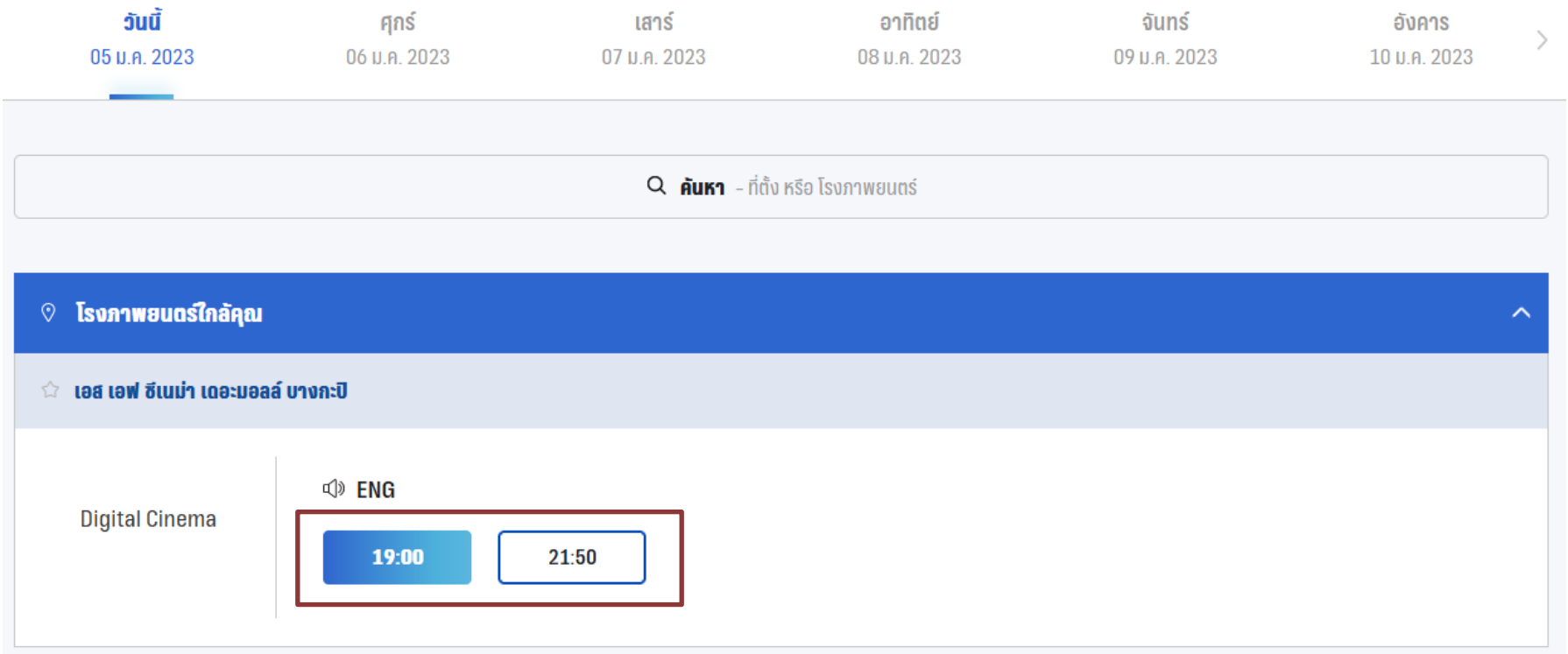
- คลาสที่นั่ง และ ประเภทที่นั่งสามารถเขียนเป็นคลาสได้ดังนี้

```
class CinemaHallSeat:
    def __init__(self, seat_row, seat_col, seat_type):
        self.seat_row = seat_row
        self.seat_col = seat_col
        self.seat_type = seat_type

class SeatPrice:
    def __init__(self, seat_type, price):
        self.seat_type = seat_type
        self.price = price
```

Case Study

- มาพิจารณาข้อมูลส่วนต่อไป คือ ข้อมูลตามภาพนี้



Case Study

- คำถาม คือ รอบฉาย ควรเป็นข้อมูลของใคร
- เป็นข้อมูลของภาพยนตร์? หรือ เป็นข้อมูลของโรงภาพยนตร์?
- มีหลักคิดประการหนึ่งในการออกแบบคลาส คือ เรื่องของความใกล้ชิด (cohesion) โดยหลักนี้กล่าวว่า “ข้อมูลในคลาส ควรเกี่ยวข้องกันมากที่สุด”
- จะเห็นว่ารอบฉาย ไม่เกี่ยวข้องกับภาพยนตร์ หรือ โรงภาพยนตร์มากพอ แต่จะอยู่ตรงกลางระหว่าง ภาพยนตร์ กับ โรงภาพยนตร์ กรณีนี้ควรจะสร้างเป็นคลาสใหม่
- คลาสรอบฉาย อาจจะให้ความรู้สึกจับต้องได้น้อยกว่า ภาพยนตร์ หรือ โรงภาพยนตร์ แต่กรณีนี้ก็สามารถจัดเป็นคลาสได้
- คลาสรอบฉาย จะใช้ชื่อว่า Show มี attribute start_time และ end_time

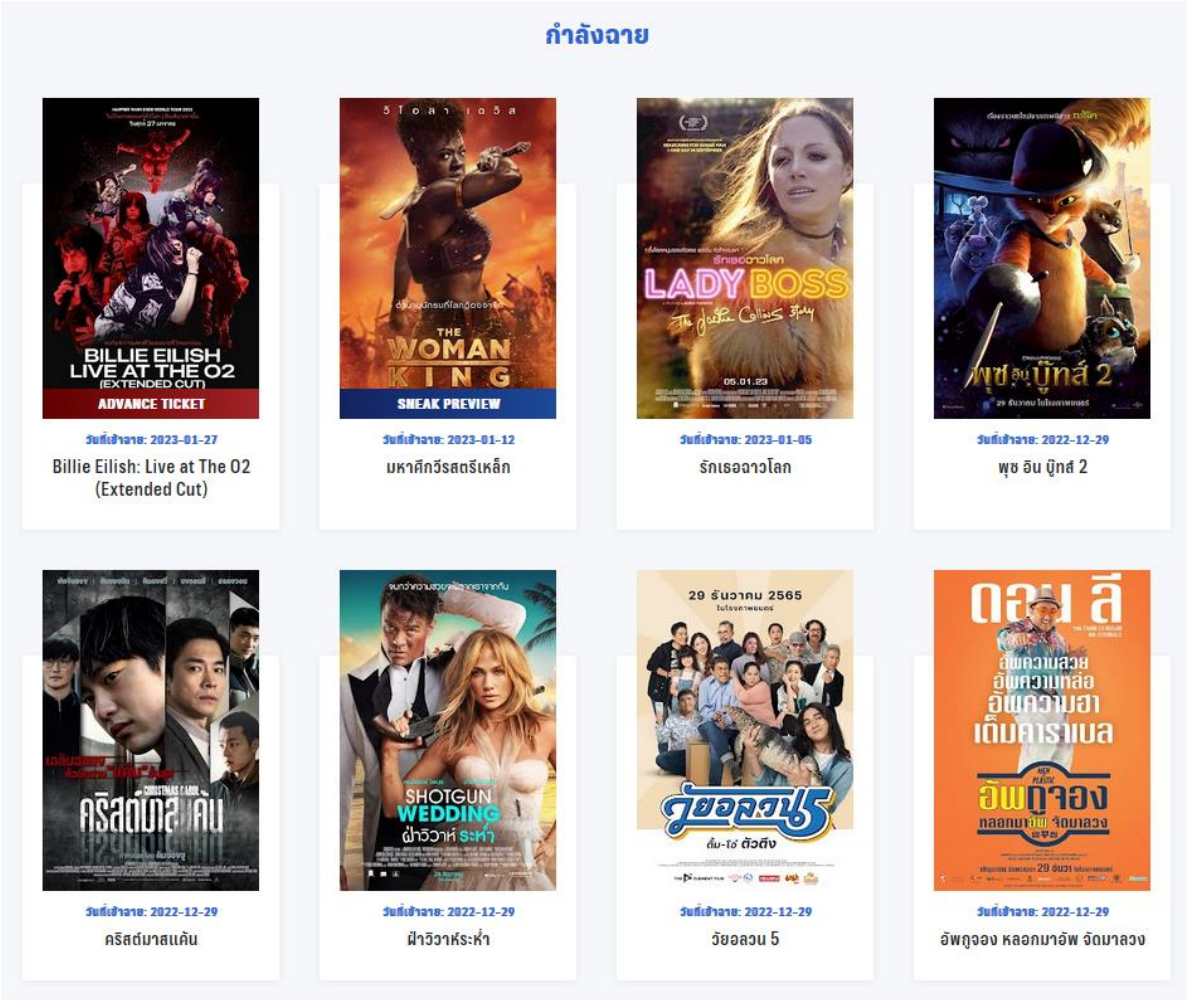
Case Study

- สามารถเขียนเป็นภาษา Python ได้ดังนี้

```
class Show:
    def __init__(self, start_time, end_time):
        self.start_time = start_time
        self.end_time = end_time
```

Case Study

- ในเว็บไซต์โรงภาพยนตร์จะขาดหน้าแบบนี้ไม่ได้



Case Study

- เราจะเรียกหน้านี้ว่า catalog และสร้างเป็นคลาส Catalog
- คลาส Catalog จะมีประกอบด้วย attribute ดังนี้
  - last_update
  - movies[ ] จะเป็น list ของ object Movie ที่เข้าฉายหรือมีโปรแกรมเข้าฉาย
- สามารถเขียนเป็นโปรแกรมได้ดังนี้

```
class MovieCatalog:
    def __init__(self, last_update, movies):
        self.last_update = last_update
        self.movie = []    # list of Movie Object
```

Case Study

- ต่อไปจะเป็นการจองรอบชมภาพยนตร์ หรือ ซื้อบัตรชมภาพยนตร์
- ด้วยเหตุผลเดียวกับรอบฉาย เราจะจัดให้การจองตั๋วเป็น คลาส เช่นกัน

มหาศึกวีรสตรีเหล็ก

05 มกราคม 2023 | 19:00

เอส เอฟ ซีเนม่า เดอะมอลล์ บางกะปิ

โรงภาพยนตร์ 8 | ENG

ที่นั่งที่เลือก	ราคารวม
D8	200 บาท

Case Study

- การจองบัตร หรือ ซื้อบัตร จะออกแบบให้เป็นคลาสเดียวกัน เพราะใช้ข้อมูลชุดเดียวกัน ต่างตรงที่ชำระเงินหรือยัง
- จะตั้งชื่อคลาสว่า Booking และข้อมูลจะประกอบด้วย หมายเลขการจอง (เพื่อใช้อ้างอิง) จำนวนที่นั่ง และ สถานะ
- จะเห็นได้ว่าการจองขาดข้อมูลไป 1 อย่าง คือ การจองข้างต้นเป็นการจองไว้ที่นั่งใด ก็มีคำถามว่า จะเก็บที่นั่งที่จองไปแล้วไว้ในคลาส Booking ได้หรือไม่
- คำตอบ คือ ได้ แต่มีประเด็นว่าในการแสดงที่นั่งให้เลือกในการจอง จะค้นหาข้อมูลจากที่ใดว่าที่นั่งนั้นว่างหรือไม่ เราจะค้างประเด็นนี้ไปก่อน แล้วค่อยพิจารณาภายหลัง
- เราเลือกการเพิ่มคลาสอีก 1 คลาส ตั้งชื่อว่า ShowSeat โดยทำหน้าที่เก็บข้อมูล ที่นั่ง ราคา และ การจองที่นั่ง

Case Study

- สามารถเขียนเป็นโปรแกรมได้ดังนี้

```
class Booking:
    def __init__(self, no_of_seat, status):
        self.no_of_seat = no_of_seat
        self.status = status

class ShowSeat:
    def __init__(self, seat_no, price):
        self.seat_no = seat_no
        self.price = price
        self.is_reserve = None
```

Case Study

- จะมีคลาสประเภทหนึ่ง ที่ใช้ในการเก็บค่าคงที่ ซึ่งจะใช้ในการรวบรวมค่าคงที่ในเรื่องใดเรื่องหนึ่งเอาไว้ในคลาสเดียวกัน
- คลาสลักษณะนี้ จะเรียกว่า enum class มีข้อมูลหลายประเภทที่ควรใช้ enum class

```
class SeatType(Enum):
    DELUXE, PREMIUM, EXECUTIVE, FIRST_CLASS, SOFA_SWEET \
    = 1, 2, 3, 4, 5

class BookingStatus(Enum):
    PENDING, CONFIRMED, CANCELED = 1, 2, 3

class AccountStatus(Enum):
  ACTIVE, BLOCKED, BANNED, ARCHIVED, UNKNOWN = 1, 2, 3, 4, 5

class PaymentStatus(Enum):
  UNPAID, PENDING, COMPLETED, DECLINED, CANCELLED, REFUNDED \
    = 1, 2, 3, 4, 5, 6
```

Case Study

- การใช้งาน enum จะคล้ายกับ dictionary

```
print(SeatType.PREMIUM)
print(SeatType.PREMIUM.name)
print(SeatType.PREMIUM.value)
print(type(SeatType.PREMIUM))
print(SeatType(2).name)
print(SeatType['PREMIUM'].value)
```

```
SeatType.PREMIUM
PREMIUM
2
<enum 'SeatType'>
PREMIUM
2
```

- และใช้กับ loop ได้

```
for seat in SeatType:
    print(('{:15} = {}'.format(seat.name, seat.value)))
```

```
DELUXE          = 1
PREMIUM         = 2
EXECUTIVE       = 3
FIRST_CLASS     = 4
SOFA_SWEET      = 5
```

Case Study

- ข้อมูลส่วนต่อไป คือ ข้อมูลการชำระเงิน จะมีคำถามว่า เราควรนำข้อมูลการชำระเงินไปรวมกับการจองตั๋วหรือไม่
- คำตอบ คือ ขึ้นกับการออกแบบ โดยหากมีรายละเอียดปลีกย่อยเกี่ยวกับการชำระเงิน เช่น ประเภทของการชำระเงิน หรือ Transaction ID ก็ควรจะแยกออกมาเพื่อให้เป็นไปตามหลัก “ข้อมูลในคลาส ควรเกี่ยวข้องกันมากที่สุด”
- จะตั้งชื่อคลาสข้อมูลการชำระว่า Payment โดยมีข้อมูล คือ วันที่ของรายการ จำนวนเงิน สถานะชำระเงิน และหมายเลขรายการ

Case Study

- สามารถเขียนเป็นโปรแกรม Python ได้ดังนี้

```
import datetime
class Payment:
  def __init__(self, amount, transaction_id, payment_status):
    self.__amount = amount
    self.__created_on = datetime.date.today()
    self.__transaction_id = transaction_id
    self.__status = payment_status
```

Case Study

- โรงภาพยนตร์จะมีระบบสมาชิก

Case Study

- ดังนั้นจะต้องมีคลาสสมาชิก และต้องมีคลาสของเจ้าหน้าที่ขายตั๋ว กับผู้ดูแลระบบสามารถเขียนเป็นโปรแกรมได้ดังนี้

```
class Admin:
    def __init__(self, name, email, phone):
        self.name = name
        self.email = email
        self.phone = phone

class Customer:
    def __init__(self, name, email, phone):
        self.name = name
        self.email = email
        self.phone = phone

class FrontDeskOfficer:
    def __init__(self, name, email, phone):
        self.name = name
        self.email = email
        self.phone = phone
```

Case Study

- นอกจากนั้นจะต้องมีข้อมูลของ User Account ซึ่งทำหน้าที่เก็บ ชื่อ Username, Password และ สถานะของผู้ใช้
- สามารถเขียนเป็นโปรแกรมภาษา Python ได้ดังนี้

```
class Account:
  def __init__(self, id, password, status=AccountStatus.Active):
    self.__id = id
    self.__password = password
    self.__status = status
```

- จะเห็นว่าการใช้ enum จะช่วยให้ code มีความหมายและเข้าใจง่ายขึ้น
- ในส่วน status จะเรียกการใช้งานแบบนี้ว่า default argument โดยสามารถใช้เป็น account_1 = Account(“john”, “ripper) ได้โดยไม่ต้องกำหนด status โดยจะมีค่าเป็น Active โดยปริยาย (จะใช้งาน default argument ได้เฉพาะ last argument)

การเข้าถึง Instance Attribute

- เมื่อสร้าง Instance แล้ว หากต้องการเข้าถึงข้อมูล (attribute) ในแต่ละ Instance สามารถทำได้โดยใช้รูปแบบที่เรียกว่า dot notation
- โดยการอ้างถึงชื่อของ Instance แล้วตามด้วย . จากนั้นจึงเป็นชื่อของ attribute ตามตัวอย่าง

Modify/Update Instance Attribute

- การแก้ไขหรือเปลี่ยนแปลง Instance attribute ถือเป็นการเปลี่ยนข้อมูล (state) ของ Instance
- วิธีการ คือ กำหนดค่าทับเข้าไปใน attribute โดยใช้ dot notation เช่นกัน

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

- สำหรับกรณีที่ใช้ภายใน Class เดียวกันสามารถใช้ self แทนชื่อของ instance ได้

```
self.<attribute> = <new_value>
```

For your attention