Phần 1: OOP Concept ở các ngôn ngữ lập trình:
1.1 What is OOP?
1.2 Bốn tính chất của OOP.
1.3 Lợi ích của OOP trong lập trình hướng đối tượng

Phần 2: OOP trong Python
1.1 Pipeline cơ bản của OOP:

- Một code cơ bản của OOP
- Phân tích 2 level trong OOP: instance-level và class-level
- Phân tích đường đi, cách tạo ra object từ class
- Phân tích sâu các từ khóa có trong code cơ bản: self, __init__() (gọi là constructor), attribute, method. Đưa ra ví dụ về cách dùng của các từ khóa trên
- 4 tính chất của OOP : cho ra code về từng tính chất
    - Kế thừa: phân tích từ khóa super()
    - Đóng gói: phân tích __, _ và getter, setter
    - Abstraction: phân tích @abstract, interface
    - tính đa hình: ví dụ về tính đa hình khi kế thừa override function

1.2 OOP nâng cao: Nêu ra ví dụ và ứng dụng

- special function (magic function): built in function
- đa kế thừa: Diamond Problem
- static method

```
def Sum(sample_list):
    total = 0
   for x in sample_list:
       total += x
   return total

list1 = [10, 200, 50, 70]
list2 = [3, 26, 33, 13]
print(Sum(list1))
print(Sum(list2))
```

```
def function_a(x):
    x = .....
    return x

def function_b(x):
    x = .....
    return x

def function_c(x):
```

```
    x = .....
    return x

def flow_chay(x):
    x= function_a(x)
    x= function_b(x)
    x= function_c(x)
    return x
```

```
class Flow:
    def __init__(self,x):
        self.x = x
   def function_a(self):
        self.x = ....
   def function_b(self):
        self.x = ....
   def function_c(self):
        self.x = ....
   def run_flow(self):
        function_a()
        function_b()
        function_c()
```

```
class Flow_1(Flow):
    def __init__(self,x):
        super(self,x)
    def function_a(self):
       self.x =.... # thay đổi mà object 4 muốn
```

```
obj_1 = "..."
obj_1_result = Flow(obj_1)

obj_2 = "..."
obj_2_result = Flow(obj_2)

obj_3 = "..."
obj_3_result = Flow(obj_3)

obj_4 = "..."
obj_4_result = Flow_1(obj_4)
```

```
class Person:
```

```
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def print_info(self):
        print(f"Info: {self.name} is {self.age}"
```

```
class Student(Person):
    def __init__(self,name, age, classroom):
        super(self,name, age)
        self.classroom = classroom
    def print_info(self):
        print(f"Info: {self.name} is {self.age} who studying in class
{self.classroom}")
```

```
person_1 = Person("A", "20")
person_2 = Student("B", "21", "A1")

person_1.print_info()
person_2.print_info()
```

```
class MainView():
    resolution = "1920x1080"
    name = "view"

    def show_view():
        return "The view is shown"

class ChildView(MainView):
    name = "sub-view"

print(MainView.show_view())
print(ChildView.show_view())
```

```
class MainView():
    def __init__(self):
       self.resolution = "1920x1080"
       self.name = "view"

    def show_view(self):
        return f"{self.name} is shown"

class ChildView(MainView):
    def __init__(self):
        super().__init__()
    def show_view(self):
        return f"Child {self.name} is shown"

main_view = MainView()
print(main_view.show_view())
child_view = ChildView()
print(child_view.show_view())
```

```
class Person:
    def __init__(self):
        self.bien_public = "Public"
        self.__bien_private = "Private"

    def truy_cap_bien_private(self):
        print("Truy cap private: ",self.__bien_private)

class Student(Person):
    def __init__(self):
        super().__init__()

student_1 = Student()
print(student_1.bien_public)
student_1.truy_cap_bien_private()
print(student_1.__bien_private)
```

```
from abc import ABC, abstractmethod

class abstractClassName(ABC):
    @abstractmethod
    def print_info(self):
        pass
```

```
class Person(abstractClassName):
    def __init__(self):
        self.name = "A"
        self.age = 20
    def print_info(self):
        print(f"{self.name} is {self.age}")

person = Person()
person.print_info()
```

```
class Person:
    bien_class_level = "class-level"
    def __init__(self, bien_instance_level):
        self.bien_instance_level = bien_instance_level

person_1 = Person("instance_1")
person_2 = Person("instance_2")

print(person_1.bien_class_level,",", person_1.bien_instance_level)
print(person_2.bien_class_level,",", person_2.bien_instance_level)
```

```
class Animal:
    name = "animal"
    def __init__(self):
        pass
    def set_name(self,name):
        self.name = name

animal_1 = Animal()
print(animal_1.name)
animal_1.set_name("new_name")
print(animal_1.name)
```

```
class Animal:
    def __init__(self):
        self.name="animal"

    def set_name(self,name):
        self.name = name
```

```
class Animal:
    def __init__(self):
```

```
        self.name="animal"

    def get_self(self):
        print(self)

animal_1 = Animal()

print(animal_1)
animal_1.get_self()
Animal.get_self(animal_1)
```

```
class Animal:
    def __init__(self):
        self.name="animal"

    def get_self(self):
        print(self)

    def set_name(self,name):
        self.name = name

animal_1 = Animal()

animal_1.set_name("new_name_1")
print(animal_1.name)

Animal.set_name(animal_1,"new_name_2")
print(animal_1.name)
```

Bài tập:

1. Cho code class dưới đây:

```
class Number_Plate:
    def __init__(self, province_number, number):
        self.province_number =  province_number
        self.number = number
    def get_full_plate(self):
        return self.province_number + "-" + self.number

class Vehicle:
    def __init__(self, brand, number_plate):
        self.brand = brand
        self.number_plate = number_plate
    def get_car_info(self):
        return "Vehicle brand: "+self.brand+", number plate:
"+self.number_plate.get_full_plate()

class Car(Vehicle):
    def __init__(self,brand,number_plate,car_type):
        super().__init__(brand,number_plate)
        self.car_type = car_type
    def get_car_info(self):
        return "Car brand: "+self.brand+", number plate:
"+self.number_plate.get_full_plate()+", car type: "+self.car_type

class Police:
    def __init__(self, name, work_unit):
        self.name = name
        self.work_unit = work_unit
    def write_car_ticket(self,car):
        return "Police "+self.name+" at "+self.work_unit+" write a
ticket for "+car.get_car_info()
```

Sử dụng lại class ở trên và viết code khởi tạo object sao cho Command prompt in ra:

```
Car brand: BMW, number plate: 51B-3236, car type: Sedan
Police Chau at Hue write a ticket for Car brand: BMW, number plate: 51B
    -3236, car type: Sedan
```

Chau_car = Car(“BMW”,”51B-3236”,”Sedan”)
print(Chau_car.get_car_info())
Chau_police = Police(“Chau”,”Hue”)
print(Chau_police.write_car_ticket(Chau_car))

---

2. Đây là đoạn code sau khi đã tạo class thì sẽ gọi trong hàm main để chạy code:

```
p1 = Person("Khoa","7-7-2000")
user_horoscope = HoroscopeWeb.get_horoscope_info(p1)
print(user_horoscope)
p1.name # Will be error
```

Command prompt in ra:

```
Khoa's Horoscope is cancer
ERROR!
Traceback (most recent call last):
  File "<string>", line 20, in <module>
AttributeError: 'Person' object has no attribute '__name'
```

Nhiệm vụ là: Viết class Person và HoroscopeWeb để cho dựa vào đoạn code trên chạy ra kết quả như command prompt.
Lưu ý:

- HoroscopeWeb gọi thẳng từ class mà ko cần tạo ra object để gọi hàm get_horoscope_info()
- biến name không thể gọi từ bên ngoài object được

```
from datetime import datetime
from pytzodiac import get_zodiac_sign

class Person:
    def __init__(self, name, day_of_birth):
        self.__name = name
        self.day_of_birth = day_of_birth

class HoroscopeWeb(Person):
    def __init__(self,name, day_of_birth):
        super().__init__(name, day_of_birth)
    def get_horoscope_info(p):
        birthday = datetime.strptime(self.day_of_birth, "%d-%m-%Y")
        return self.name + "'s Horoscope is " +  get_zodiac_sign(birthday)
```

2. **Bài tập 2 (Thay thế) (Tập trung vào tính mở rộng)**

- Tạo một class Shape (lớp cha), gồm 2 hàm: calculate_area() (tính diện tích), calculate_perimeter() (tính chu vi)

- Tạo lớp con:
  - lớp Circle: có input là radius (bán kính), override lại 2 hàm calculate_area và calculate_perimeter
  - lớp Rectangle: có input là width (chiều rộng) và height (chiều dài), override lại 2 hàm calculate_area và calculate_perimeter
  - lớp Square: có input là side(cạnh), override lại 2 hàm calculate_area và calculate_perimeter

Đoạn code hàm main như sau:

```
circle = Circle(radius = 3)
print(circle.calculate_area())

rectangle = Rectangle(width = 3, height = 5)
print(rectangle.calculate_area())

square = Square(side = 2)
print(square.calculate_area())
```

3. Đây là đoạn code sau khi đã tạo class thì sẽ gọi trong hàm main để chạy code:

```
salary_employee_1 = SalaryEmployee(id="14321",name="Khoa",weekly_salary
= 750000)
hour_employee_2 = HourlyEmployee(id="15001",name="Long",hours_worked =
4.5, hourly_salary=20000)

system = PayrollSystem()
system.calculate_payroll([salary_employee_1, hour_employee_2])
```

Command prompt in ra:

```
Calculating Payroll
====================
Payroll for: 14321 - Khoa
- Check amount: 750000

Payroll for: 15001 - Long
- Check amount: 90000.0
```

```
from datetime import datetime
from pytzodiac import get_zodiac_sign

class SalaryEmployee:
    def __init__(self, id, name, weekly_salary):
```

```
        self.id = id
        self.name = name
        self.weekly_salary = weekly_salary
    def get_salary(self):
        return f"Payroll for: {self.id} - {self.name}\n- Check amount:
{self.weekly_salary}"

class HourlyEmployee:
    def __init__(self, id, name, hours_worked, hourly_salary):
        self.id = id
        self.name = name
        self.hours_worked = hours_worked
        self.hourly_salary = hourly_salary
    def get_hourly_salary(self):
        return f"Payroll for: {self.id} - {self.name}\n- Check amount:
{self.hours_worked * self.hourly_salary}"

class PayrollSystem():
    def calculate_payroll(employees):
        for employee in employees:
            for employee in employees:
                if isinstance(employee, SalaryEmployee):
                    print(employee.get_salary())
                elif isinstance(employee, HourlyEmployee):
                    print(employee.get_hourly_salary())
```

Bài giải bài 3:

```
class Employee:
    def __init__(self, id, name):
        self.id= id
        self.name = name

class SalaryEmployee(Employee):
    def __init__(self, id, name, weekly_salary):
        super().__init__(id, name)
        self.weekly_salary = weekly_salary
    def get_salary(self):
        return f"Payroll for: {self.id} - {self.name}\n- Check amount:
{self.weekly_salary}"

class HourlyEmployee(Employee):
    def __init__(self, id, name, hours_worked, hourly_salary):
        super().__init__(id, name)
        self.hours_worked = hours_worked
        self.hourly_salary = hourly_salary
    def get_salary(self):
        return f"Payroll for: {self.id} - {self.name}\n- Check amount:
{self.hours_worked * self.hourly_salary}"
```

```
class PayrollSystem():
    def __init__(self):
        pass
    def calculate_payroll(self, employees):
        for employee in employees:
               for employee in employees:
                   print(employee.get_salary())

salary_employee_1 = SalaryEmployee(id="14321",name="Khoa",weekly_salary = 750000)
hour_employee_2 = HourlyEmployee(id="15001",name="Long",hours_worked = 4.5,
hourly_salary=20000)

system = PayrollSystem()
system.calculate_payroll([salary_employee_1, hour_employee_2])
```

4. Thông tin người dùng: (tính mở rộng, và tính đóng gói)

- Tạo một lớp cha (UserInfo): gồm name, age, address **(các thông tin này đều phải là private)**, hàm print_info()
- Lớp con:
    - Teacher: gồm name, age, address, teach_class, override lại hàm print_info()
    - Student: gồm name, age, address, learn_class, override lại hàm print_info()

- từ thông tin của teacher và student, viết một hàm overall_info() để in ra tên giáo viên dạy những học sinh nào ở cùng lớp

Ví dụ:

teacher_A = Teacher(“A”,27,”ABC”,”class_A1”)
student_B = Student(“B”,17,”...”,”class_A1”)
student_C = Student(“C”,17,”...”,”class_A1”)
student_D = Student(“D”,17,”...”,”class_A1”)
overall_info([teacher_A, student_B, student_C, student_D]) # sẽ in ra: teacher A teaches student B,C,D.

5. Quản lý sách (Association, Aggregation relation)

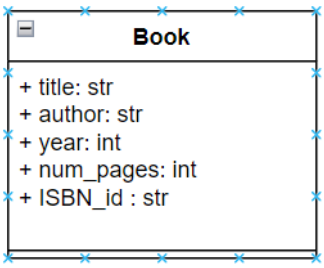
Tạo lớp `Book` đại diện cho một cuốn sách. Mỗi cuốn sách cần có các thông tin sau:

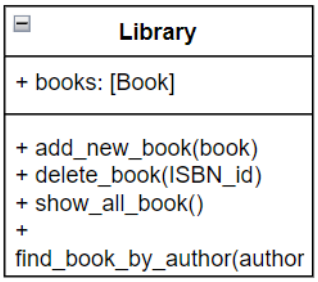
- Tiêu đề (title)
- Tác giả (author)
- Năm xuất bản (year)
- Số trang (pages)
- Mã số ISBN

Tạo một lớp `Library` để quản lý sách trong thư viện. Thư viện chứa một list các cuốn sách và cần có các chức năng sau:

- Thêm một cuốn sách mới vào thư viện.
- Xóa một cuốn sách khỏi thư viện dựa trên mã số ISBN.
- Hiển thị danh sách tất cả các cuốn sách trong thư viện.
- Tìm kiếm sách theo tên tác giả hoặc năm xuất bản.

Tạo một chương trình chính để kiểm tra các chức năng của thư viện. Cho phép người dùng thêm sách mới, xóa sách theo ISBN, hiển thị danh sách sách có trong thư viện, và tìm kiếm sách.





6. Một bài về việc return class (Kiến thức mới)

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

    def drive(self):
        return f"{self.name} is driving the car."

class Car:
    def __init__(self, driver):
        self.driver=driver
    def start(self):
        print("The car is started.")
        # Here, we create an instance of the Driver class and return it
        return self.driver

car_1 = Car(Driver("John"))
car_1.start().drive()
```

7. Đọc code trên docs

```
@classmethod
def tên_hàm(cls):
    cls.tên_thuộc_tính_class_level
```

```
def tên_hàm(self):
    self.tên_thuộc_tính_instance_level
```

```
class Car:
    __brand_car = "BMW"
    def __init__(self, color):
        self.__color = color
    @classmethod
    def get_brand_car(cls):
        return cls.__brand_car
    @classmethod
    def set_brand_car(cls, brand):
        cls.__brand_car = brand

    def get_car_color(self):
        return self.__color

car_1 = Car("red")
car_2 = Car("blue")

print(car_1.get_brand_car(), car_2.get_brand_car()) # BMW BMW
# Đổi giá trị của biến class-level
Car.set_brand_car("Toyota")
print(car_1.get_brand_car(), car_2.get_brand_car()) # Toyota Toyota
print(car_1.get_car_color(), car_2.get_car_color()) # red blue
```

```
class Library:
    def get_book(self, book):
        print("Get book: ",book.info())

class Book:
    def __init__(self,name,author):
        self.name= name
        self.author = author
    def info(self):
        return self.name + " by " + self.author

harry_potter = Book("Harry Potter","JK.Rowling")
nauy_forest = Book("Nauy Forest","Murakami Haruki")

fpt_library = Library()
fpt_library.get_book(harry_potter) # Get book:  Harry Potter by
JK.Rowling
```

```
fpt_library.get_book(nauy_forest) # Get book:  Nauy Forest by Murakami Haruki
```

```
class Student:
    def __init__(self, name, gender):
        self.name = name
        self.gender = gender
    def info(self):
        return self.name + " is "+self.gender

class Classroom:
    def __init__(self, classroom_name, students):
        self.students = students
        self.classroom_name = classroom_name
    def get_classroom_info(self):
        output = "Class "+self.classroom_name+" has: "
        for student in self.students:
            output+=f"\n {student.info()}"
        return output

chau_student = Student("Chau","F")
khoa_student = Student("Khoa","M")

oop_class = Classroom("OOP",[chau_student, khoa_student])
print(oop_class.get_classroom_info())
```

```
class Person:
    # Biến lớp (class variable)
    total_people = 0

    def __init__(self, name, age):
        # Biến của đối tượng (instance variable)
        self.name = name
        self.age = age
        Person.total_people += 1

    # Phương thức của đối tượng (instance method)
    def say_hello(self):
        print(f"Xin chào, tôi là {self.name} và tôi {self.age} tuổi.")

    # Phương thức của lớp (class method)
    @classmethod
    def get_total_people(cls):
        return cls.total_people
```

```python
# Tạo các đối tượng Person
person1 = Person("Alice", 25)
person2 = Person("Bob", 30)

# Gọi phương thức của đối tượng
person1.say_hello()
person2.say_hello()

# Gọi phương thức của lớp để lấy tổng số người đã tạo
total_people = Person.get_total_people()
print(f"Tổng số người: {total_people}")
```

```python
# Mối quan hệ Association
class School:
    def __init__(self, name):
        self.name = name
        self.teachers = []
        self.students = []

    def add_teacher(self, teacher):
        self.teachers.append(teacher)

    def add_student(self, student):
        self.students.append(student)

class Person:
    def __init__(self, name):
        self.name = name

    def introduce(self):
        pass

class Teacher(Person):
    def __init__(self, name, subject):
        super().__init__(name)
        self.subject = subject

    def introduce(self):
        return f"Hi, I'm {self.name}, and I teach {self.subject}."

class Student(Person):
    def __init__(self, name, grade):
        super().__init__(name)
```

```
        self.grade = grade

    def introduce(self):
        return f"Hi, I'm {self.name}, and I'm in grade {self.grade}."

# Mối quan hệ Inheritance và Aggregation
class Classroom:
    def __init__(self, number, teacher):
        self.number = number
        self.teacher = teacher
        self.students = []

    def add_student(self, student):
        self.students.append(student)

    def list_students(self):
        return [student.name for student in self.students]

school = School("Example School")
teacher1 = Teacher("Mr. Smith", "Math")
teacher2 = Teacher("Mrs. Johnson", "Science")
student1 = Student("Alice", 10)
student2 = Student("Bob", 9)

school.add_teacher(teacher1)
school.add_teacher(teacher2)
school.add_student(student1)
school.add_student(student2)

classroom1 = Classroom(101, teacher1)
classroom2 = Classroom(102, teacher2)

classroom1.add_student(student1)
classroom2.add_student(student2)

print(f"School: {school.name}")
print("Teachers:")
for teacher in school.teachers:
    print(teacher.introduce())

print("\nClassrooms:")
print(f"Classroom {classroom1.number} students: {',
'.join(classroom1.list_students())}")
print(f"Classroom {classroom2.number} students: {',
'.join(classroom2.list_students())}")
```

OOP Session 2 Bài tập:

1. Tạo ra một class BankAccount trong đó:
    a. biến exchange_rate là class variable, có giá trị 23000 (giá trị tỷ đổi của USD->VND)
    b. account_number và balance_usd là instance variable:
        i. account_number: là số tài khoản người dùng
        ii. balance_usd: số dư người dùng
    c. Các hàm:
        i. deposit(amount_usd): hàm gửi tiền, nhận đầu vào là số tiền muốn gửi, sau khi nạp thì cộng vào số dư người dùng
        ii. withdraw(amount_usd): hàm rút tiền, nhận đầu vào là số tiền muốn rút, sau khi rút thì trừ vào số dư người dùng, nếu rút không đủ tiền trong tài khoản thì báo lỗi
        iii. display_balance(): In ra
            1. Số tài khoản: …
            2. Số dư (VND): …
            3. Số dư (USD): …
2. Chỉnh sửa lại cho bài 1: những biến exchange_rate, account_number và balance_usd được set thành private. Cho nên cần phải có thêm hàm get và set nếu muốn lấy và set giá trị mới cho các biến trên