


TRƯỜNG: ĐH Công thương TP.HCM KHOA: CÔNG NGHỆ THÔNG TIN BỘ MÔN: KHDL&TTNT MH: Lập trình Python	BUỔI 4. Kế thừa và đa hình trong Python	
---	--	---

A. MỤC TIÊU

Sau khi học xong buổi học sinh viên có thể:

- Trình bày được tính kế thừa trong Python
- Trình bày được tính ghi đè phương thức
- Trình bày được tính phương thức trừu tượng trong Python
- Trình bày được tính đa thừa kế trong Python
- Sử dụng được hàm issubclass và isinstance
- Sử dụng được tính đa hình với hàm

B. DỤNG CỤ - THIẾT BỊ THÍ NGHIỆM CHO MỘT SV

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	Computer	1	1	

C. NỘI DUNG THỰC HÀNH

I. Tóm tắt lý thuyết

1. Kế thừa trong Python

Trong lập trình hướng đối tượng có một khái niệm gọi là kế thừa, trong đó bạn có thể tạo một lớp được xây dựng từ một lớp khác. Khi bạn làm điều này, lớp mới sẽ nhận được tất cả các biến và phương thức của lớp mà nó kế thừa (được gọi là lớp cơ sở). Sau đó, nó có thể định nghĩa các biến và phương thức bổ sung không có trong lớp cơ sở và nó cũng có thể ghi đè một số phương thức của lớp cơ sở. Tức là nó có thể viết lại chúng cho phù hợp với mục đích riêng của mình.

Cú pháp:

```
class className(inherit1, inherit2,...):
    #code
```

Trong đó: inherit1, inherit2,... là tên của các class kế thừa.

Ví dụ 1. viết 2 class *Male* và *Femal* chứa 2 phương thức *getName*, *getAge*. Nếu như không sử dụng tính kế thừa trong hướng đối tượng thì phải viết 2 class có các phương thức và thuộc tính tương đối giống nhau như trên. Nhưng ta thấy 2 class này có các điểm chung giống nhau thì cho nên có thể tạo ra một class cha chứa những điểm chung này, từ đó ở mỗi class con sẽ kế thừa những điểm chung đó và phát triển thêm các điểm riêng.

Lúc này class **Male** sẽ sử dụng được toàn bộ các phương thức và thuộc tính của class **Person**.

Khởi tạo class **Male** và gọi một vài phương thức được khai báo trong lớp cha của nó.

```
male = Male("Nguyễn Văn A", 22)
male.getName()
male.getAge()
male.getSex()
```

Kết quả:

```
Name: Nguyễn Văn A
Age: 22
Sex: Male
```

Ta thấy, trong class **Male** hoàn toàn không chứa một phương thức nào nhưng do nó được kế thừa từ class **Person** nên nó sẽ sử dụng lại được những thành phần được cho phép trong class **Person**.

Ví dụ 2: Tạo một lớp có tên **Person**, với các thuộc tính **firstname** và **Lastname** và một phương thức **printname**:

```
class Person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname

    def printname(self):
        print(self.firstname, self.lastname)

x = Person("John", "Doe")
x.printname()
```

2. Ghi đè

Trong trường hợp cả hai class cha và con tồn tại các thuộc tính và phương thức có cùng tên, thì trong Python sẽ ưu tiên thực thi và gọi các phương thức và thuộc tính khai báo trong class được khởi tạo. Ví dụ

```
#Ghi đè
class Foo:
    name = 'Foo'
    def getName(self):
        print("Class: Foo")
class Bar(Foo):
    name = 'Bar'
    def getName(self):
        print("Class: Bar")

print(Foo().name)
Foo().getName()
print(Bar().name)
Bar().getName()
```

Kết quả:

```
Foo
Class: Foo
Bar
Class: Bar
```

Ví dụ trên ta thấy, khi chúng ta khởi tạo class nào thì nó sẽ ưu tiên đến các thành phần trong class đó hơn.

3. Phương thức trừu tượng

Abstract class là một class mà bên trong nó chứa một hoặc nhiều phương thức trừu tượng. **Phương thức trừu tượng** ở đây là một phương thức mà chúng ta chỉ được phép khai báo nó và không được phép viết code thực thi nó. Khi một class được khai báo ở dạng abstract thì nó sẽ không thể nào khởi tạo được, mà chỉ có thể khởi tạo được thông qua các class con của nó. Một class kế thừa lại abstract class thì phải khai báo lại toàn bộ các phương thức trừu tượng bên trong abstract class mà nó kế thừa.

3.1. Khai báo abstract class trong Python

Để có thể khai báo được một abstract class trong Python, thì class này bắt buộc phải được kế thừa từ một ABC (Abstract Base Classes) của Python.

Và để gọi được class này trong chương trình thì bạn phải import nó. Cú pháp import như sau:

```
from abc import ABC
```

Cú pháp khai báo abstract class:

```
class ClassName(ABC):
    # code
```

Trong đó: ClassName là tên của abstract class mà bạn muốn khai báo.

Ví dụ 1: khai báo một lớp trừu tượng person.

```

from abc import ABC, abstractmethod

class PersonAbstract(ABC):
    name = None
    age = 0
    def getName(self):
        print(self.name)
    def getAge(self):
        print(self.age)

```

Vì abstract class là một class, nên chúng ta có thể hoàn toàn khai báo thuộc tính và phương thức như một class bình thường.

3.2. Khai báo phương thức abstract trong Python

Để có thể khai báo một abstract method - phương thức trừu tượng trong Python thì chúng ta cần phải import thêm module `abstractmethod` ở trong package `abc`.

```
from abc import ABC, abstractmethod
```

Và một phương thức trừu tượng thì bắt buộc phải được khai báo ở trong lớp trừu tượng.

Cú pháp:

```

from abc import ABC, abstractmethod

class ClassName(ABC):
    # khai báo phương thức trừu tượng
    @abstractmethod
    def methodName(self):
        pass

```

Trong đó:

- `@abstractmethod` là bắt buộc, đây là cú pháp khai báo cho Python biết phía dưới là phương thức trừu tượng.
- `methodName` là tên của phương thức trừu tượng.

Ví dụ : khai báo thêm phương thức trừu tượng `getFull` vào trong **PersonAbstract** của VD trên.

```

from abc import ABC, abstractmethod

class PersonAbstract(ABC):
    name = None
    age = 0
    def getName(self):
        print(self.name)
    def getAge(self):
        print(self.age)
    @abstractmethod
    def getFull(self):
        pass

```

Các ví dụ:

Đầu tiên, để chứng minh là abstract class không thể khởi tạo được một cách trực tiếp thì mình sẽ thử khởi tạo class **PersonAbstract** ở ví dụ trên xem sao:

```
from abc import ABC, abstractmethod

class PersonAbstract(ABC):
    name = None
    age = 0
    def getName(self):
        print(self.name)
    def getAge(self):
        print(self.age)
    @abstractmethod
    def getFull(self):
        pass

PersonAbstract()
# Lỗi
# Can't instantiate abstract class PersonAbstract with abstract methods getFull
```

Có thông báo lỗi. Tạo một class và kế thừa lại **PersonAbstract** này và đồng thời khởi tạo nó xem sao.

```
from abc import ABC, abstractmethod

class PersonAbstract(ABC):
    name = None
    age = 0
    def getName(self):
        print(self.name)
    def getAge(self):
        print(self.age)
    @abstractmethod
    def getFull(self):
        pass

class Person(PersonAbstract):
    pass

Person();
# Lỗi
# Can't instantiate abstract class Person with abstract methods getFull
```

Nó vẫn báo lỗi, bây giờ ở class con này chúng ta sẽ khai báo lại method getFull và khởi tạo lại nó.

```

from abc import ABC, abstractmethod

class PersonAbstact(ABC):
    name = None
    age = 0
    def getName(self):
        print(self.name)
    def getAge(self):
        print(self.age)
    @abstractmethod
    def getFull(self):
        pass

class Person(PersonAbstact):
    name = 'Vu Thanh Tai'
    age = 22
    def getFull(self):
        self.getName()
        self.getAge()

Person();|

```

Chương trình đã không báo lỗi. Bây giờ chúng ta sẽ gọi thử phương thức getFull xem sao.

```

from abc import ABC, abstractmethod

class PersonAbstact(ABC):
    name = None
    age = 0
    def getName(self):
        print(self.name)
    def getAge(self):
        print(self.age)
    @abstractmethod
    def getFull(self):
        pass

class Person(PersonAbstact):
    name = 'Nguyễn Văn A'
    age = 22
    def getFull(self):
        self.getName()
        self.getAge()

Person().getFull();

```

Ví dụ 2: Để định nghĩa một lớp cơ sở trừu tượng có tên là **Polygon** (Đa giác) bằng cách sử dụng mô-đun ABC (Lớp cơ sở trừu tượng) trong Python. Lớp Polygon có một phương thức trừu tượng được gọi là “noofsides” cần được các lớp con của nó triển khai.

Có bốn lớp con của đa giác được định nghĩa: **Triangle** (Tam giác), **Pentagon** (Ngũ giác), **Hexagon** (Lục giác), và **Quadrilateral** (Tứ giác). Mỗi lớp con này ghi đề phương thức “noofsides” và cung cấp cách triển khai riêng bằng cách in số cạnh mà nó có.

```
# abstract base class work
from abc import ABC, abstractmethod

class Polygon(ABC):
    @abstractmethod
    def noofsides(self):
        pass

class Triangle(Polygon):
    # overriding abstract method
    def noofsides(self):
        print("I have 3 sides")

class Pentagon(Polygon):
    # overriding abstract method
    def noofsides(self):
        print("I have 5 sides")

class Hexagon(Polygon):
    # overriding abstract method
    def noofsides(self):
        print("I have 6 sides")

class Quadrilateral(Polygon):
    # overriding abstract method
    def noofsides(self):
        print("I have 4 sides")
```

Test:

```
# Driver code
R = Triangle()
R.noofsides()

K = Quadrilateral()
K.noofsides()

R = Pentagon()
R.noofsides()
```

Kết quả:

```
I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides
```

Như vậy phần này mọi người cần phải chú ý những điều sau:

- Abstract là một class, và class này chứa một hoặc nhiều phương thức trừu tượng.
- Abstract class không thể khởi tạo trực tiếp được.
- Một class kế thừa từ abstract class thì bắt buộc phải khai báo lại các các phương thức trừu tượng có trong abstract mà nó kế thừa.

4. Đa thừa kế

Giống như C++ thì Python cũng hỗ trợ chúng ta đa kế thừa.

```
class First:
    def getFirst(self):
        print("Class First")

class Second:
    def getSecond(self):
        print("Class Second")

class Third(First, Second):
    def getThird(self):
        print("Class Third")

third = Third()
third.getFirst()
third.getSecond()
third.getThird()

# Kết Quả:
# Class First
# Class Second
# Class Third
```

Về Phần ghi đè thì vẫn như cũ là: Nó sẽ ưu tiên thành phần được khai báo trong lớp được khởi tạo.

Trong đa kế thừa nếu như bạn sử dụng `super()` thì nó sẽ gọi đến thành phần trong class kế thừa được truyền vào đầu tiên trong argument của class con.

```
class First:
    def getClass(self):
        print("Class First")

class Second:
    def getClass(self):
        print("Class Second")

class Third(First, Second):
    def getClass(self):
        super().getClass()

third = Third()
third.getClass()

# Kết Quả:
# Class First
```

Do class `First` được truyền vào đầu tiên nên khi sử dụng `super` thì nó sẽ tìm đến thành phần được gọi ở trong class `First`.

Trong trường hợp muốn gọi cả phương thức `getName()` trong class `Second` nữa thì bạn cần thay đổi code một chút như sau:

```
class First:
    def getClass(self):
        print("Class First")
        super().getClass()

class Second:
    def getClass(self):
        print("Class Second")

class Third(First, Second):
    def getClass(self):
        super().getClass()

third = Third()
third.getClass()

# Kết Quả:
# Class First
# Class Second
```

Tính chất đa kế thừa của Python thực chất là dựa vào Tính chất bắc cầu, A Kế thừa B và B kế thừa C \Rightarrow A cũng kế thừa C. Hay nói đơn giản trong Ví dụ trên chúng ta có thể chuyển về dạng bắc cầu như sau:


```

class First:
    def getClass(self):
        print("Class First")

class Second(First):
    def getClass(self):
        super().getClass()
        print("Class Second")

class Third(Second):
    def getClass(self):
        super().getClass()

third = Third()
third.getClass()

# Kết Quả:
# Class First
# Class Second

```

Từ đó ta có thể hiểu rằng vị trí các tham số argument truyền vào trong class con càng đứng phía sau nó càng có cấp cao hơn class phía trước nó.

5. Hàm isinstance và issubclass

Python có 2 hàm hữu ích: isinstance & issubclass

- Hàm **isinstance** giúp bạn kiểm tra xem một "cái gì đó" có phải là một đối tượng của một lớp nào đó hay không.
- Hàm **issubclass** kiểm tra xem lớp này có phải là con của một lớp khác hay không.

Ví dụ:

```

class A: pass
class B(A): pass
# True
print ("isinstance('abc', object): ", isinstance('abc', object))
# True
print ("isinstance(123, object): ", isinstance(123, object))

b = B()
a = A()

# True
print ("isinstance(b, A): ", isinstance(b, A) )
print ("isinstance(b, B): ", isinstance(b, B) )

# False
print ("isinstance(a, B): ", isinstance(a, B) )

# B is subclass of A? ==> True
print ("issubclass(B, A): ", issubclass(B, A) )

# A is subclass of B? ==> False
print ("issubclass(A, B): ", issubclass(A, B) )

```

Kết quả:

```

isinstance('abc', object): True
isinstance(123, object): True
isinstance(b, A): True
isinstance(b, B): True
isinstance(a, B): False
issubclass(B, A): True
issubclass(A, B): False

```

6. Đa hình với hàm

Tính đa hình trong OOP sử dụng một giao diện chung cho nhiều kiểu dữ liệu.

Giả sử, chúng ta cần tô màu cho một vật, có nhiều tùy chọn hình dạng (hình chữ nhật, hình vuông, hình tròn). Tuy nhiên, chúng ta có thể sử dụng cùng một phương pháp để tô màu bất kỳ hình dạng nào. Khái niệm này được gọi là Đa hình.

Sử dụng tính đa hình trong Python:

```
class Parrot:
    def fly(self):
        print("Parrot can fly")

    def swim(self):
        print("Parrot can't swim")

class Penguin:
    def fly(self):
        print("Penguin can't fly")

    def swim(self):
        print("Penguin can swim")

# common interface
def flying_test(bird):
    bird.fly()

# instantiate objects
blu = Parrot()
peggy = Penguin()

# passing the object
flying_test(blu)
flying_test(peggy)
```

Kết quả:

```
Parrot can fly
Penguin can't fly
```

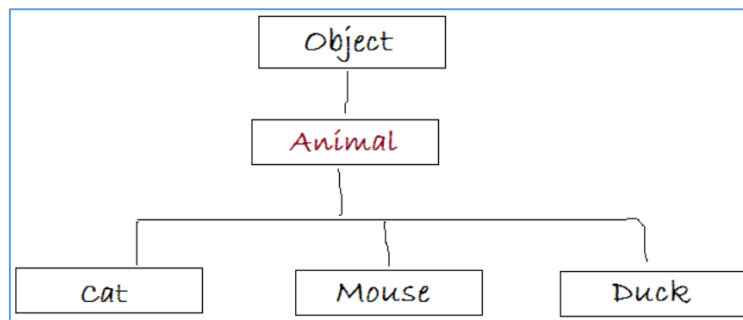
Ở ví dụ ta đã định nghĩa 2 lớp Parrot và Penguin. Mỗi lớp đều có phương thức chung là fly(). Tuy nhiên chức năng thì sẽ khác nhau.

Để sử dụng tính đa hình. Ta tạo một hàm fly_test() nhận bất kỳ đối tượng là và gọi vào phương thức fly() của đối tượng đó. Vì vậy khi ta chuyển các đối tượng blu và peggy thì hàm sử dụng được và trả ra kết quả.

II. Bài tập hướng dẫn mẫu

Bài 1. Xét một vài class tham gia vào minh họa.

- **Animal:** Class mô phỏng một lớp Động vật.
- **Duck:** Class mô phỏng lớp vịt, là một class con của **Animal**.
- **Cat:** Class mô phỏng lớp mèo, là một class con của **Animal**
- **Mouse:** Class mô phỏng lớp chuột, là một class con của **Animal**.



Mô tả cho cho lớp Animal:

```
#Mô tả Lớp Animal
class Animal:
    # Constructor
    def __init__(self, name):
        # Lớp Animal có 1 thuộc tính (attribute): 'name'.
        self.name= name
    # Phương thức (method):
    def showInfo(self):
        print ("I'm " + self.name)
    # Phương thức (method):
    def move(self):
        print ("moving ...")
```

Mô tả cho lớp Cat mở rộng (extends) từ lớp Animal:

```
# Lớp Cat mở rộng (extends) từ Lớp Animal.
class Cat (Animal):

    def __init__(self, name, age, height):
        # Gọi tới constructor của lớp cha (Animal)
        # để gán giá trị vào thuộc tính 'name' của Lớp cha.
        super().__init__(name)

        self.age = age
        self.height = height
```

Mô tả cho tính ghi đè (override) phương thức cùng tên của lớp cha:

```
# Ghi đè (override) phương thức cùng tên của Lớp cha.
    def showInfo(self):

        print ("I'm " + self.name)
        print (" age " + str(self.age))
        print (" height " + str(self.height))
```

Kiểm tra kết quả:

```
#Test kết quả
tom = Cat("Tom", 3, 20)

print ("Call move() method")

tom.move()

print ("\n")
print ("Call showInfo() method")

tom.showInfo()
```

Kết quả:

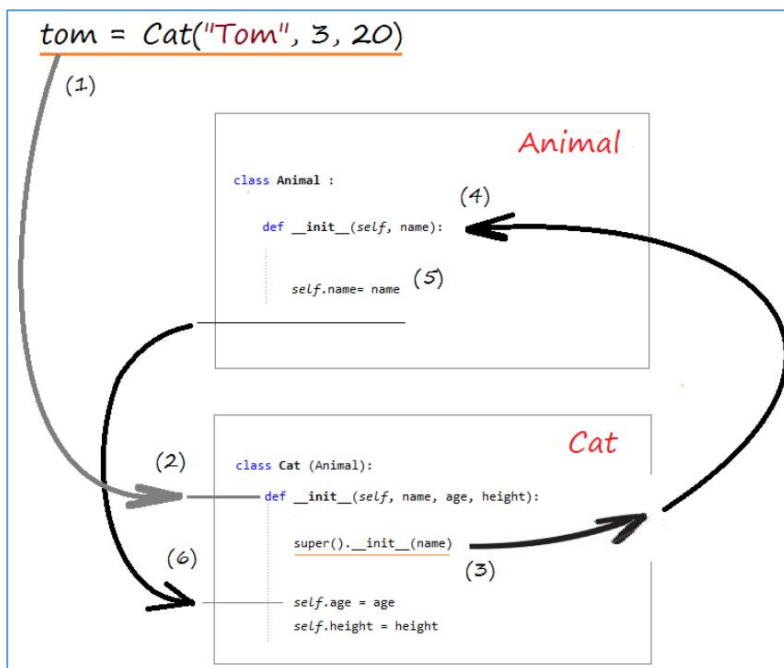
Call move() method
moving ...

Call showInfo() method
I'm Tom
age 3
height 20

Điều gì xảy ra khi khởi tạo một đối tượng từ phương thức khởi tạo (constructor)?

Nó sẽ gọi lên phương thức khởi tạo của lớp cha như thế nào?

Chúng ta hãy xem hình minh họa dưới đây:



Mặc định lớp con được thừa kế các phương thức từ lớp cha, tuy nhiên lớp con có thể ghi đè (override) phương thức của lớp cha.

```
# Lớp Mouse mở rộng (extends) từ Lớp Animal.
class Mouse (Animal):

    def __init__(self, name, age, height):
        # Gọi tới Constructor của Lớp cha (Animal)
        # để gán giá trị vào thuộc tính 'name' của Lớp cha.
        super().__init__(name)

        self.age = age
        self.height = height

    # Ghi đè (override) phương thức cùng tên của Lớp cha.
    def showInfo(self):
        # Gọi phương thức của Lớp cha.
        super().showInfo()
        print (" age " + str(self.age))
        print (" height " + str(self.height))

    # Ghi đè (override) phương thức cùng tên của Lớp cha.
    def move(self):
        print ("Mouse moving ...")
```

Kiểm tra kết quả:

```
#Test |
jerry = Mouse("Jerry", 3, 5)

print ("Call move() method")

jerry.move()

print ("\n")
print ("Call showInfo() method")

jerry.showInfo()
```

Kết quả:

```
Call showInfo() method
I'm Jerry
age 3
height 5
```

Ví dụ 2: Person (lớp cha) và Employee (Lớp con). Lớp Employee kế thừa từ lớp Person. Chúng ta có thể sử dụng các phương thức của lớp Person thông qua lớp Employee như đã thấy trong hàm hiển thị ở đoạn mã trên. Lớp con cũng có thể sửa đổi hành vi của lớp cha như được thấy qua phương thức Details().

```
# Lớp cha
class Person(object):

    # __init__ is known as the constructor
    def __init__(self, name, idnumber):
        self.name = name
        self.idnumber = idnumber

    def display(self):
        print(self.name)
        print(self.idnumber)

    def details(self):
        print("My name is {}".format(self.name))
        print("IdNumber: {}".format(self.idnumber))

# Lớp con
class Employee(Person):
    def __init__(self, name, idnumber, salary, post):
        self.salary = salary
        self.post = post

        # Gọi __init__ của lớp cha
        Person.__init__(self, name, idnumber)

    def details(self):
        print("My name is {}".format(self.name))
        print("IdNumber: {}".format(self.idnumber))
        print("Post: {}".format(self.post))

# Tạo 1 biến
a = Employee('Nam', 886012, 200000, "Intern")

# calling a function of the class Person using
# its instance
a.display()
a.details()
```

Bài 3: Sử dụng tính kế thừa hãy tạo một lớp cơ sở có tên là “Động vật” và hai lớp con, “Chó” và “Mèo”. Thêm các phương thức và thuộc tính cụ thể cho từng lớp con.

```
# oop python exercises: Inheritance, Sub-Class
class Animal:
    def __init__(self, name):
        self.name = name

class Dog(Animal):
    def sound(self):
        return "Woof!"

class Cat(Animal):
    def sound(self):
        return "Meow!"

# Create instances of Dog and Cat
dog_instance = Dog("Buddy")
cat_instance = Cat("Whiskers")

# Call the sound method on the instances
dog_sound = dog_instance.sound()
cat_sound = cat_instance.sound()

# Print the output
print(f"{dog_instance.name} says: {dog_sound}")
print(f"{cat_instance.name} says: {cat_sound}")
```

Bài 4: Tạo một lớp **Person**. Bao gồm các thuộc tính: tên, quốc gia và ngày sinh. Thực hiện một phương pháp để xác định tuổi của người đó.

```
# Import the date class from the datetime module to work with dates
from datetime import date

# Define a class called Person to represent a person with a name, country, and date of birth
class Person:
    # Initialize the Person object with a name, country, and date of birth
    def __init__(self, name, country, date_of_birth):
        self.name = name
        self.country = country
        self.date_of_birth = date_of_birth

    # Calculate the age of the person based on their date of birth
    def calculate_age(self):
        today = date.today()
        age = today.year - self.date_of_birth.year
        if today < date(today.year, self.date_of_birth.month, self.date_of_birth.day):
            age -= 1
        return age

# Example usage
# Create three Person objects with different attributes
person1 = Person("Nam", "VietNam", date(2000, 7, 12))
person2 = Person("Shweta Maddox", "Canada", date(1990, 10, 20))
person3 = Person("Elizaveta Tilman", "USA", date(2001, 1, 1))

# Access and print the attributes and calculated age for each person
print("Person 1:")
print("Name:", person1.name)
print("Country:", person1.country)
print("Date of Birth:", person1.date_of_birth)
print("Age:", person1.calculate_age())
```

Kết quả:

```
Person 1:
Name: Nam
Country: VietNam
Date of Birth: 2000-07-12|
Age: 23

Person 2:
Name: Shweta Maddox
Country: Canada
Date of Birth: 1990-10-20
Age: 33

Person 3:
Name: Elizaveta Tilman
Country: USA
Date of Birth: 2001-01-01
Age: 23
```

III. Bài tập ở lớp

Bài 1. Tạo ba lớp, Person có 2 thuộc tính: name, age, Employee (Nhân viên) có 2 thuộc tính: emp_id, salary và Student (Sinh viên) có 2 thuộc tính: student_id , grade . Sử dụng nhiều kế thừa để tạo một lớp “PersonInfo” kế thừa từ cả “Nhân viên” và “Sinh viên”. Thêm các thuộc tính và phương thức cụ thể cho từng lớp.

Ví dụ: PersonInfo("John", 30, "E123", 50000, "S456", "A")

Kết quả:

Name: John

Age: 30

Employee ID: E123

Salary: 50000

Student ID: S456

Grade: A

Bài 2. Tính chu vi & diện tích các hình (abstract)

Viết chương trình tính chu vi và diện tích của một số hình như sau:

- Hình tròn
- Hình chữ nhật
- Hình tam giác

Bài 3. Viết chương trình Python để tạo một lớp đại diện cho giỏ hàng (ShoppingCart). Bao gồm các phương pháp thêm (add_item) và xóa (remove_item) các mặt hàng cũng như tính tổng giá (calculate_total). **Ví dụ:**

```
Current Items in Cart:
Papaya - 100
Guava - 200
Orange - 150
Total Quantity: 450

Updated Items in Cart after removing Orange:
Papaya - 100
Guava - 200
Total Quantity: 300
```

Bài 4. Viết một lớp có tên là **Sản phẩm**. Lớp này phải có các thuộc tính: số lượng và giá, tên sản phẩm, số lượng mặt hàng của sản phẩm đó trong kho và giá thông thường của sản phẩm. Cần có một phương thức **get_price** để nhận số lượng mặt hàng cần mua và trả về chi phí mua nhiều mặt hàng đó, trong đó giá thông thường được tính cho các đơn hàng dưới 10 mặt hàng, áp dụng giảm giá 10% cho các đơn hàng từ giữa 10 và 99 mặt hàng, giảm giá 20% cho đơn hàng từ 100 mặt hàng trở lên. Cũng nên có một phương thức gọi là **make_purchase** để nhận số lượng mặt hàng cần mua và giảm số lượng đó đi.

Bài 5. Tạo một lớp có tên là Giỏ hàng và thực hiện các thao tác sau:

- (1) Tạo một hàm tạo không có đối số và đặt thuộc tính tổng bằng 0, đồng thời khởi tạo một thuộc tính dict trống có tên là items.
- (2) Tạo một phương thức **add_item** yêu cầu các đối số **item_name**, số lượng và giá. Phương pháp này sẽ cộng chi phí của các hạng mục được thêm vào và giá trị tổng hiện tại. Nó cũng nên thêm một mục nhập vào lệnh của các mục sao cho khóa là **item_name** và giá trị là số lượng của mục đó.
- (3) Tạo một phương thức **Remove_item** yêu cầu các đối số tương tự như **add_item**. Nó sẽ loại bỏ các mặt hàng đã được thêm vào giỏ hàng và không cần thiết. Phương pháp này sẽ khấu trừ chi phí của các mặt hàng đã xóa khỏi tổng số hiện tại và cũng cập nhật chính sách các mặt hàng tương ứng.
- (4) Nếu số lượng của một mặt hàng cần loại bỏ vượt quá số lượng hiện tại của mặt hàng đó trong giỏ hàng, giả định rằng tất cả các mục của mặt hàng đó sẽ bị xóa.
- (5) Tạo phương thức thanh toán nhận tiền mặt_đã thanh toán và trả về giá trị số dư từ khoản thanh toán. Nếu tiền mặt_đã thanh toán không đủ để thanh toán tổng số tiền, hãy trả về "Tiền mặt thanh toán không đủ".
- (6) Tạo một lớp có tên Shop có hàm tạo không có đối số và khởi tạo thuộc tính có tên số lượng ở mức 100. Đảm bảo Shop kế thừa từ ShopCart.
- (7) Trong lớp Shop, ghi đè phương thức **Remove_item**, chẳng hạn như gọi **Remove_item** của Shop mà không có đối số sẽ giảm số lượng đi một.

Bài 6. Viết một lớp tên là Wordplay. Nó phải có một trường chứa danh sách các từ. Người dùng của lớp phải chuyển danh sách các từ họ muốn sử dụng cho lớp. Cần có các phương thức sau:

- (1) `words_with_length(length)`: trả về danh sách độ dài của tất cả các từ
- (2) `started_with(s)`: trả về danh sách tất cả các từ bắt đầu bằng s
- (3) `end_with(s)`: trả về danh sách tất cả các từ kết thúc bằng s
- (4) `only(L)`: trả về danh sách các từ chỉ chứa các chữ cái đó trong L
- (5) `avoids (L)`: trả về danh sách các từ không chứa chữ cái nào trong L

Bài 7. Tạo một lớp và định nghĩa hai phương thức trong lớp.

- (1) Phương thức `f1` được sử dụng để chuyển một danh sách trống và danh sách đã sắp xếp được lấy từ người dùng sang phương thức `f2`.
- (2) Phương thức `f2` được sử dụng để tính toán tất cả các tập con có thể có của danh sách.
- (3) Sau đó kết quả được trả về từ hàm và được in.

```
Enter number of elements of list: 2
Enter element: 4
Enter element: 5
Subsets:
[[], [5], [4], [4, 5]]
```

Bài 8. Xây dựng một chương trình quản lý thông tin sinh viên bao gồm mã sinh viên và họ và tên. Thực hiện các thao tác thêm, xóa, sửa và xem danh sách sinh viên.

Bài 9. Xây dựng ứng dụng quản lý danh sách các giao dịch:

Mô tả: Hệ thống quản lý 2 loại giao dịch:

- Giao dịch vàng: Mã giao dịch, ngày giao dịch (ngày/tháng/năm), đơn giá, số lượng, loại vàng có 3 loại 18k, 24k, 9999. Thành tiền được tính như sau: thành tiền = số lượng * đơn giá.
- Giao dịch tiền tệ: Mã giao dịch, ngày giao dịch (ngày/tháng/năm), tỷ giá (cũng là đơn giá), số lượng, loại tiền tệ có 3 loại: USD, EUR, AUD, loại giao dịch mua/bán. Thành tiền được tính như sau:
 - + Nếu loại giao dịch là “mua” thì: thành tiền = số lượng * tỷ giá
 - + Nếu loại giao dịch là “bán” thì: thành tiền = (số lượng * tỷ giá) * 1.05

```

Quản lý giao dịch:
Nhập mã GD:      gd001
Nhập ngày GD:    13/03/2017
Nhập số lượng:   10
Chọn loại giao dịch: 1: Vàng, 2: Tiền Tệ:      1
Chọn loại: 18k / 24k / 9999:      18k
Nhập đơn giá:    2350000
gd001 - 13/03/2017 - 18k - 10 - 2350000 - Thành tiền = 23500000
Tổng số lượng: 10
Tổng số tiền: 23500000
Bạn muốn tiếp tục giao dịch? 1: Có, 0: Không    1
Nhập mã GD:      gd002
Nhập ngày GD:    14/03/2017
Nhập số lượng:   100
Chọn loại giao dịch: 1: Vàng, 2: Tiền Tệ:      2
Chọn loại: USD / EUR / AUD:      USD
Nhập tỷ giá:     23000
Bạn mua hay bán? 1: mua, 0: bán:      1
GD mua: gd002 - 14/03/2017 - USD - 100 - 23000 - Thành tiền = 2300000
Tổng số lượng: 100
Tổng số tiền: 2300000
Bạn muốn tiếp tục giao dịch? 1: Có, 0: Không    1

```

Dựa vào mô tả trên, hãy:

- (1) Tạo lớp GiaoDich với các thuộc tính và phương thức chung (giao dịch vàng cũng là giao dịch).
- (2) Tạo lớp GiaoDichTienTeké thừa từ lớp GiaoDich với các thuộc tính riêng và phương thức cần thiết.
- (3) Nhập xuất danh sách các giao dịch.
- (4) Tính tổng số lượng cho từng loại.
- (5) Tính tổng thành tiền cho từng loại.

Bài 10. Một sinh viên gồm các thông tin sau:

- Mã sinh viên là chuỗi có 10 ký tự
- Tên là chuỗi tối đa 20 ký tự
- Năm sinh là một số nguyên
- Điểm trung bình là một số thực a.

- (1) Xây dựng cấu trúc SINHVIEN mô tả một sinh viên
- (2) Cho một mảng có n sinh viên.
- (3) Viết hàm cho biết có bao sinh viên đủ điều kiện lên lớp, biết rằng sinh viên đủ điều kiện lên lớp khi điểm trung bình lớn hơn hoặc bằng 5.
- (4) Xuất các sinh viên đủ 20 tuổi.
- (5) Đếm số sinh viên học hệ đại học, biết rằng sinh viên hệ DH có mã sinh viên chứa 2 ký tự DH ở vị trí 2,3 trong chuỗi. VD: 02DH0001.
- (6) Cho biết trong mảng có bao nhiêu sinh viên có tên «Lan»
- (7) Cho biết trong mảng có bao nhiêu sinh viên có họ «Phan»