Lập trình hướng đối tượng

Object Oriented Programming

OOP

- Lớp (class)
- **Dữ liệu, thuộc tính**
- **Phương thức (method), hành vi**
- Đối tượng (object)/ thể hiện (instance)
- Truyền thông điệp (message passing)

- Trừu tượng hoá (abstraction)
- Đóng gói (encapsulation)
- Kế thừa (inheritance)
- Đa hình ((polymorphism)

Định nghĩa lớp

```
class MyNewClass:
    '''This is a docstring. I have created a new class’’’
    #Khai báo và cài đặt các phương thức khởi tạo đối tượng nếu có (constructor)

    #Khai báo và cài đặt các phương thức cho đối tượng nếu có:    (object method)
    #Khai báo và cài đặt các phương thức cho lớp nếu có     (static method)
```

Định nghĩa lớp

```
class Person:
    "This is a person class"
    age = 10

    def greet(self):
        print('Hello')

# Output: 10
print(Person.age)

# Output: <function Person.greet>
print(Person.greet)

# Output: "This is a person class"
print(Person.__doc__)
```

```
10
<function Person.greet at 0x000001A2532C0D38>
This is a person class
```

Định nghĩa lớp

```
class Person:
    "This is a person class"
    age = 10

    def greet(self):
        print('Hello')

# create a new object of Person class
harry = Person()
print(Person.greet)
print(harry.greet)
harry.greet()
```

```
<function Person.greet at 0x00000218E8D40D38>
<bound method Person.greet of <__main__.Person object at
0x00000218E8D4DAC8>>
Hello
```

Bao gói - Encapsulation

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

    def work(self):
        print(self.name, 'is working on', self.project)
```

Data Members

Method

Wrapping data and the methods that work on data within one unit

Class (Encapsulation)

Bao gói - Encapsulation

- **Public Member**: Có thể truy xuất mọi nơi
- **Private Member**: giới hạn bên trong class, sử dụng **một dấu _**
- **Protected Member**: bên trong class và các class con, sử dụng **hai dấu _**

```
class Employee:

    def __init__(self, name, salary):
        self.name = name
        self._project = project
        self.__salary = salary
```

Public Member (accessible within or outside of a class

Protected Member (accessible within the class and it's sub-classes)

Private Member (accessible only within a class)

```
    class Employee:
    def __init__(self, name, salary):
        # public data members
        self.name = name
        self.salary = salary
    # public instance methods
    def show(self):
        # accessing public data member
        print("Name: ", self.name, 'Salary:', self.salary)
# creating object of a class
emp = Employee('Jessa', 10000)
# accessing public data members
print("Name: ", emp.name, 'Salary:', emp.salary)

# calling public method of the class
emp.show()
```

➔Name: Jessa Salary: 10000
Name: Jessa Salary: 10000

Bao gói - Encapsulation

```
    # constructor
    def __init__(self, name, salary):
        # public data member
        self.name = name
        # private member
        self.__salary = salary

# creating object of a class
emp = Employee('Jessa', 10000)

# accessing private data members
print('Salary:', emp.__salary)
```

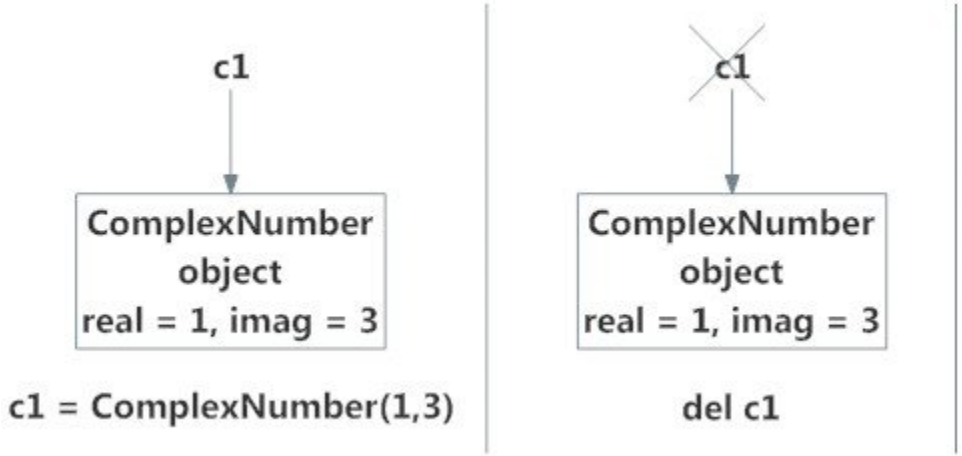
AttributeError: 'Employee' object has no attribute '__salary'

Constructors, destructor

```
class ComplexNumber:
    def __init__(self, r=0, i=0):
        self.real = r
        self.imag = i
    def get_data(self):
        print(f'{self.real}+{self.imag}j')
num1 = ComplexNumber(2, 3)
num1.get_data()
num2 = ComplexNumber(5)
# and create a new attribute 'attr’ for num2
num2.attr = 10
print((num2.real, num2.imag, num2.attr))
# but c1 object doesn't have attribute 'attr'
# AttributeError: 'ComplexNumber' object has no attribute
'attr'
print(num1.attr)
```

Deleting Attributes and Objects

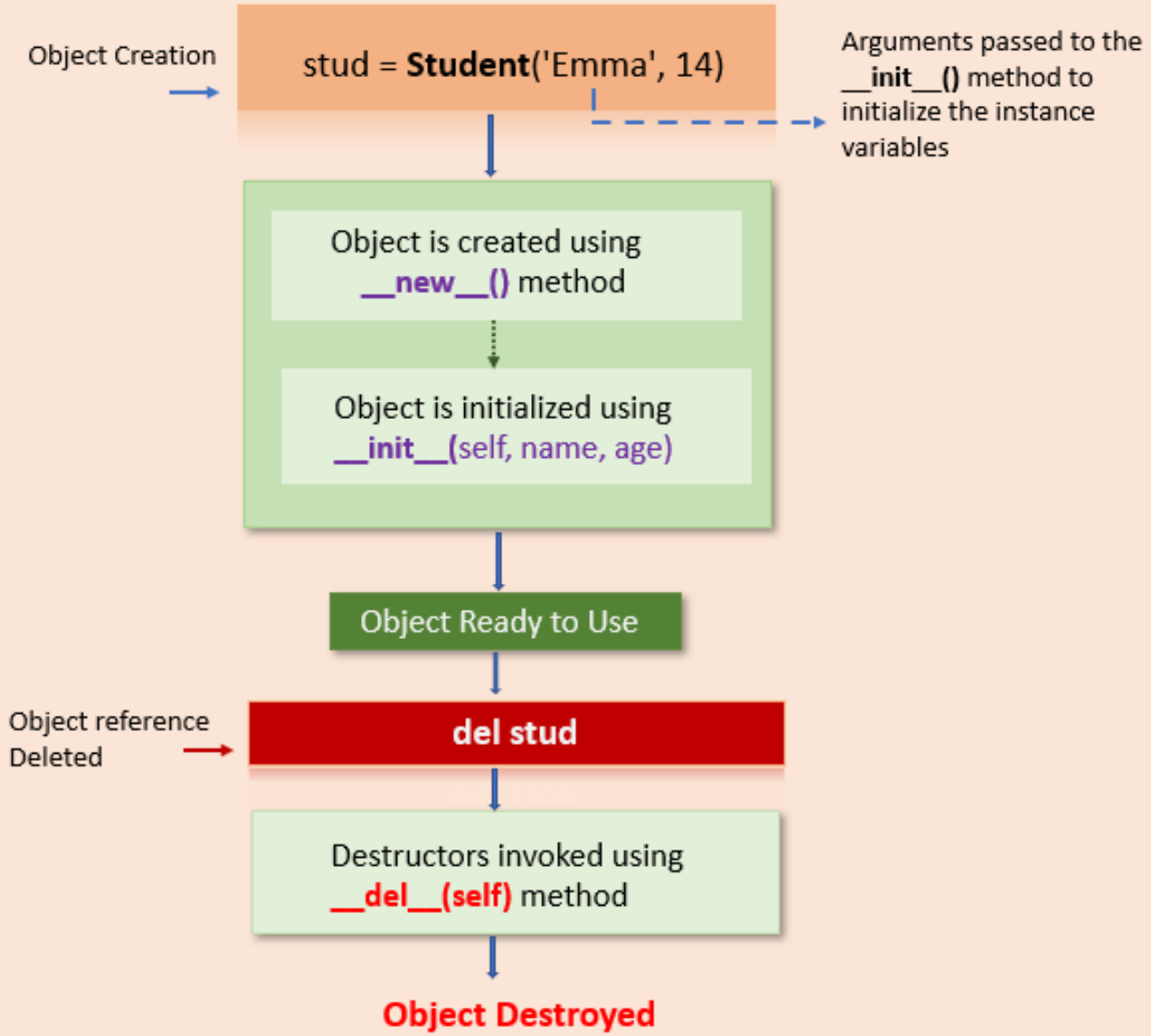
- Đối tượng có thể: thêm thuộc tính, xóa thuộc tính public, bất cứ nơi nào và khi nào (bên trong lớp và ngoài lớp)
- Thuộc tính public của một đối tượng có thể bị xóa bằng câu lệnh del
- del đối tượng là xóa đối tượng



Constructors, destructor

```
class Student:
    # constructor
    def __init__(self, name, mark):
        print('Inside Constructor')
        self.name = name
        self.mark = mark
        print('Object initialized')
    def show(self):
        print('Hello, my name is', self.name)
    # destructor
    def __del__(self):
        print('Inside destructor')
        print('Object destroyed')
s1 = Student('Emma', 14)
s1.show()
del s1
```

Object Creation and Deletion in Python



Phương thức lớp
Static method

Không tham chiếu đến thuộc tính(biến) lớp/đối tượng

```
class Calculator:

    @staticmethod
    def add_numbers(num1, num2):
        return num1 + num2

# convert add_numbers() to static method
Calculator.add_numbers = staticmethod(Calculator.add_numbers)


sum = Calculator.add_numbers(5, 7)
print('Sum:', sum)
# Output: Sum: 12
```

Phương thức lớp

Tham chiếu đến thuộc tính(biến) lớp/đối tượng, sử dụng cls

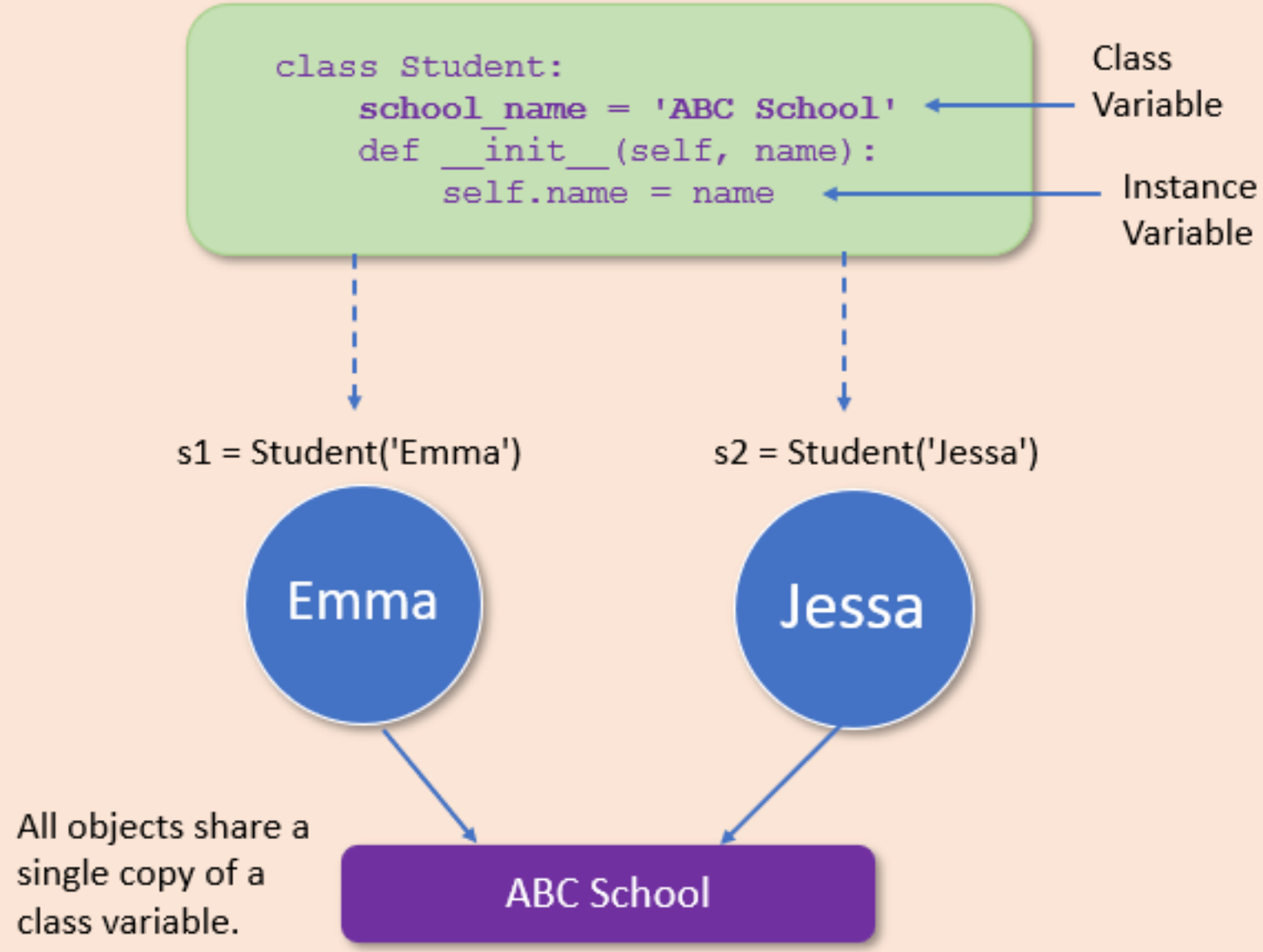
```
from datetime import date
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    @classmethod
    def calculate_age(cls, name, birth_year):
        # calculate age an set it as a age
        # return new object
        return cls(name, date.today().year - birth_year)
    def show(self):
        print(self.name + "'s age is: " + str(self.age))
jessa = Student('Jessa', 20)
jessa.show()
# create new object using the factory method
joy = Student.calculate_age("Joy", 1995)
joy.show()
```

Phương thức lớp

```
class School:
    # class variable
    name = 'ABC School'

    def school_name(cls):
        print('School Name is :', cls.name)

# create class method
School.school_name = classmethod(School.school_name)

# call class method
School.school_name()
```

School Name is : ABC School

Thuộc tính/biến lớp



Thuộc tính/biến lớp

```
class Student:
    # Class variable
    school_name = 'ABC School '
    def __init__(self, name, roll_no):
        self.name = name
        self.roll_no = roll_no
# create first object
s1 = Student('Emma', 10)
print(s1.name, s1.roll_no, Student.school_name)
# access class variable
# create second object
s2 = Student('Jessa', 20)
# access class variable
print(s2.name, s2.roll_no, Student.school_name)
```

➔Emma 10 ABC School
Jessa 20 ABC School

Thuộc tính/biến lớp

```
class Student:
    school_name = 'ABC School'

    def __init__(self, name, age):
        self.name = name
        self.age = age
    @classmethod
    def change_school(cls, school_name):
        # class_name.class_variable
        cls.school_name = school_name
    # instance method
    def show(self):
        print(self.name, self.age, 'School:',
Student.school_name)
jessa = Student('Jessa', 20)
jessa.show()
```

Jessa 20 School: ABC School
Jessa 20 School: XYZ School

```
class Vehicle:
    brand_name = 'BMW'
    def __init__(self, name, price):
        self.name = name
        self.price = price
    @classmethod
    def from_price(cls, name, price):
        # ind_price = dollar * 76
        # create new Vehicle object
        return cls(name, (price * 75))
    def show(self):
        print(self.name, self.price)
class Car(Vehicle):
    def average(self, distance, fuel_used):
        mileage = distance / fuel_used
        print(self.name, 'Mileage', mileage)
bmw_us = Car('BMW X5', 65000)
bmw_us.show()
bmw_ind = Car.from_price('BMW X5', 65000)
bmw_ind.show()
print(type(bmw_ind))
```

Thuộc tính/biến lớp

```
class Student:
    school_name = 'ABC School'
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def show(self):
        print(self.name, self.age)
# class ended
# method outside class
def exercises(cls):
    # access class variables
    print("Below exercises for", cls.school_name)
# Adding class method at runtime to class
Student.exercises = classmethod(exercises)
jessa = Student("Jessa", 14)
jessa.show()
# call the new method
Student.exercises()
```

Thuộc tính/biến lớp

```
class Student:
    school_name = 'ABC School'
    def __init__(self, name, age):
        self.name = name
        self.age = age
    @classmethod
    def change_school(cls, school_name):
        cls.school_name = school_name
jessa = Student('Jessa', 20)
print(Student.change_school('XYZ School'))
print(Student.school_name)

# delete class method
del Student.change_school
# call class method
# it will give error
print(Student.change_school('PQR School'))
```

Thuộc tính/biến lớp

```
class Student:
    school_name = 'ABC School'
    def __init__(self, name, age):
        self.name = name
        self.age = age
    @classmethod
    def change_school(cls, school_name):
        cls.school_name = school_name
jessa = Student('Jessa', 20)
print(Student.change_school('XYZ School'))
print(Student.school_name)

# delete class method
delattr(Student, 'change_school')

# call class method
# it will give error
print(Student.change_school('PQR School'))
```

Đối tượng **Mutable or Immutable?**

```
class ComplexNumber:
    def __init__(self, r=0, i=0):
        self.real = r
        self.imag = i

    def get_data(self):
        print(f'{self.real}+{self.imag}j')
num1 = ComplexNumber(2, 3)
num2 = num1
print(id(num1))
print(id(num2))
```

```
➔
1562031217544
1562031217544
```

Đối tượng **Mutable or Immutable?**

```
class ComplexNumber:
    def __init__(self, r=0, i=0):
        self.real = r
        self.imag = i

    def get_data(self):
        print(f'{self.real}+{self.imag}j')
num1 = ComplexNumber(2, 3)
num2 = num1
num1.get_data()
num2.real=4
num1.get_data()
```

```
➔
2+3j
4+3j
```

Thừa kế

```
class BaseClass:
  Body of base class

class DerivedClass(BaseClass):
  Body of derived class
```

- Lớp con kế thừa các tính năng từ lớp cơ sở
- Các tính năng mới có thể được thêm vào lớp con
- Có thể cải tiến các tính năng cũ của lớp cha

```
class Polygon:
    def __init__(self, no_of_sides):
        self.n = no_of_sides
        self.sides = [0 for i in range(no_of_sides)]
    def inputSides(self):
        self.sides = [float(input("Enter side "+str(i+1)+" : "))
for i in range(self.n)]
    def dispSides(self):
        for i in range(self.n):
            print("Side",i+1,"is",self.sides[i])
class Triangle(Polygon):
    def __init__(self):
       super().__init__(3) #or Polygon.__init__(self,3)
    def findArea(self):
        a, b, c = self.sides
        # calculate the semi-perimeter
        s = (a + b + c) / 2
        area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
        print('The area of the triangle is %0.2f' %area)
t = Triangle()
t.inputSides()
t.dispSides()
t.findArea()
```

```
class Polygon:
    def __init__(self, no_of_sides):
        self.n = no_of_sides
        self.sides = [0 for i in range(no_of_sides)]
    def inputSides(self):
        self.sides = [float(input("Enter side "+str(i+1)+" : ")) for i in
range(self.n)]
    def dispSides(self):
        for i in range(self.n):
            print("Side",i+1,"is",self.sides[i])
class Triangle(Polygon):
    def __init__(self):
       super().__init__(3)
    def findArea(self):
        a, b, c = self.sides
        # calculate the semi-perimeter
        s = (a + b + c) / 2
        area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
        print('The area of the triangle is %0.2f' %area)
t = Triangle()
t.inputSides()
t.dispSides()
t.findArea()
```

➔

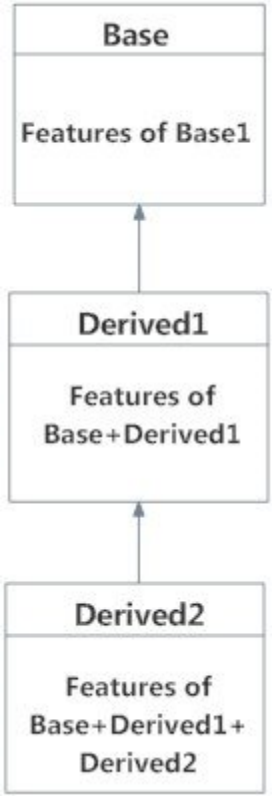
```
Enter side 1 : 3
Enter side 2 : 4
Enter side 3 : 5
Side 1 is 3.0
Side 2 is 4.0
Side 3 is 5.0
The area of the triangle is 6.00
```

Thừa kế

```
>>> isinstance(t,Triangle)
True

>>> isinstance(t,Polygon)
True

>>> isinstance(t,int)
False

>>> isinstance(t,object)
True
```

Multilevel Inheritance

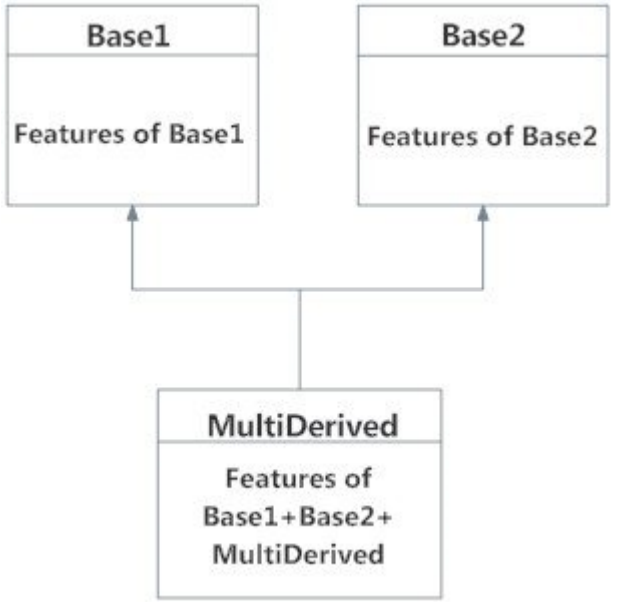
```
class Base:
    pass

class Derived1(Base):
    pass

class Derived2(Derived1):
    pass
```




```
# Output: True
print(issubclass(list,object))

# Output: True
print(isinstance(5.5,object))

# Output: True
print(isinstance("Hello",object))
```

Đa thừa kế
Multiple Inheritance

```
class Base1:
    pass

class Base2:
    pass

class MultiDerived(Base1, Base2):
    pass

print( MultiDerived.__mro__)
```



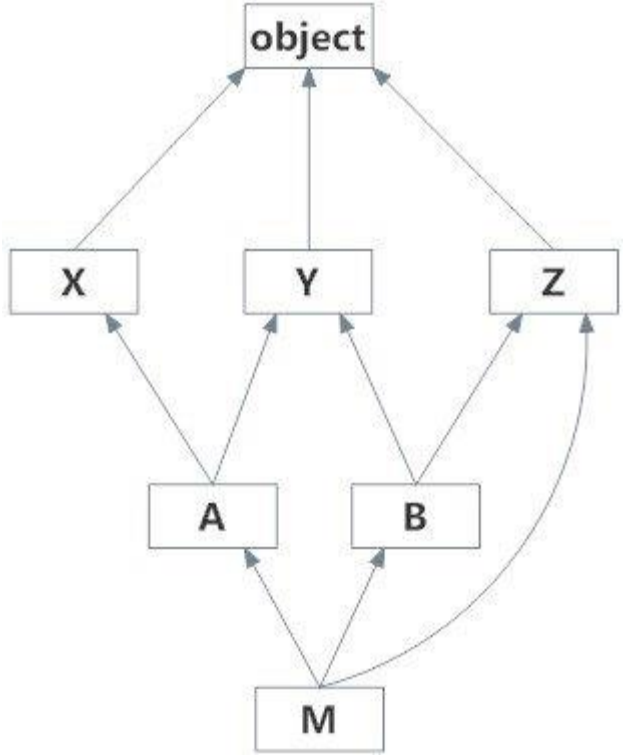

```
(<class '__main__.MultiDerived'>, <class '__main__.Base1'>,
<class '__main__.Base2'>, <class 'object'>)
```

Đa thừa kế
Multiple Inheritance

- Ví dụ C thừa kế A,B A có phương thức/thuộc tính cùng tên f, vậy C thừa kế f của A hay B? MRO là tên viết tắt của Method Resolution Order để giải quyết sự nhập nhằng của đa thừa kế.
- Là một chuỗi inheritance mà Python tính toán và lưu nó ở MRO attribute trong class.
- Khi tìm attributes, Python sẽ đi lần lượt qua các phần tử trong MRO.
- Trong kịch bản đa kế thừa, bất kỳ thuộc tính nào được chỉ định sẽ được tìm kiếm đầu tiên trong lớp hiện tại. Nếu không tìm thấy, tìm kiếm sẽ tiếp tục vào các lớp cha theo chiều sâu đầu tiên, từ trái sang phải (duyệt cây theo mức) và không tìm kiếm cùng một lớp hai lần.

Đa thừa kế
Multiple Inheritance

```
class X:
    pass
class Y:
    pass
class Z:
    pass
class A(X, Y):
    pass
class B(Y, Z):
    pass
class M(B, A, Z):
    pass

print(M.mro())
```

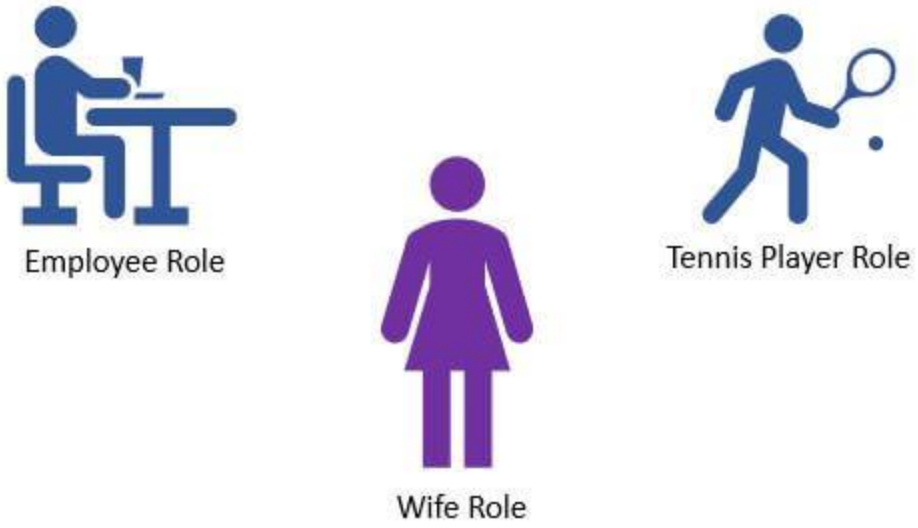



```
[<class '__main__.M'>, <class '__main__.B'>, <class '__main__.A'>,
<class '__main__.X'>, <class '__main__.Y'>, <class '__main__.Z'>,
<class 'object'>]
```

Các hàm hỗ trợ làm việc với object

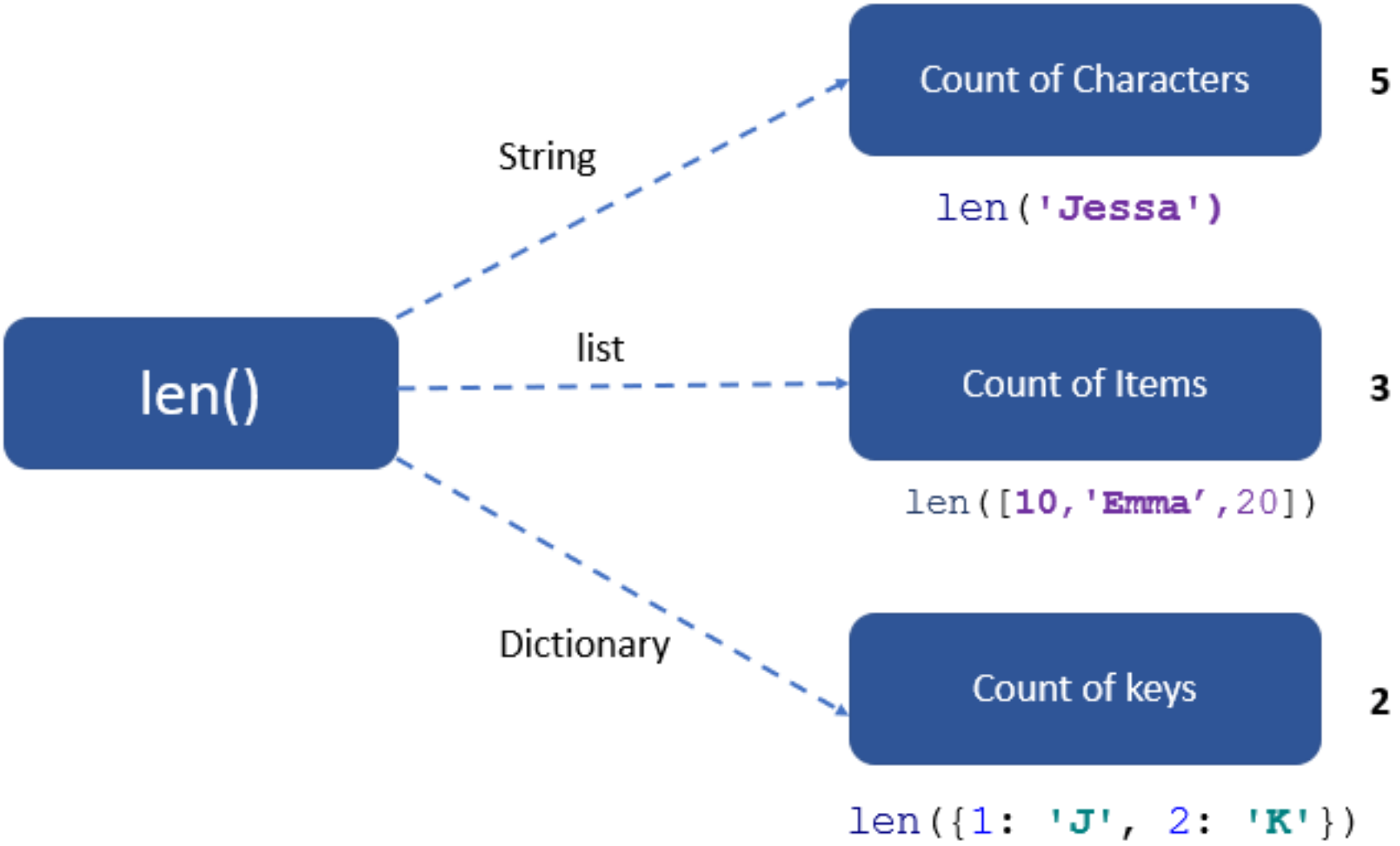
hasattr(obj, name): Trả về True/False, kiểm tra đối tượng obj có thuộc tính name hay không
setattr(obj,name,value): gán giá trị cho thuộc tính
delattr(obj, name): Xóa thuộc tính

Đa hình
Polymorphism



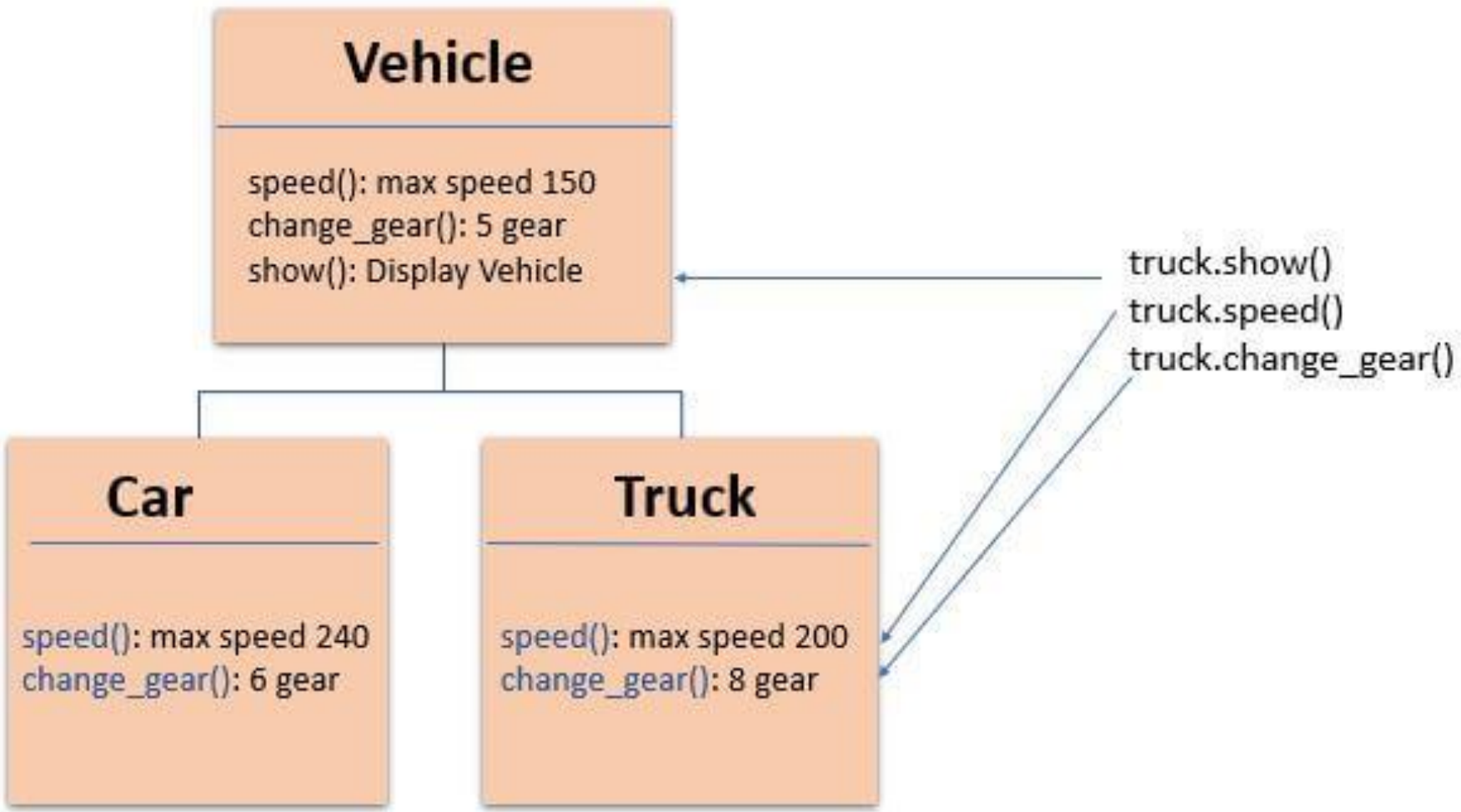
Inheritance + overriding method

Đa hình
Polymorphism



Polymorphic len() function

Đa hình
Polymorphism



Method overridden in Car and Truck class

Đa hình

```
class Vehicle:
    def __init__(self, name, color, price):
        self.name = name
        self.color = color
        self.price = price
    def show(self):
        print('Details:', self.name, self.color,
self.price)
    def max_speed(self):
        print('Vehicle max speed is 150')
    def change_gear(self):
        print('Vehicle change 6 gear')
# inherit from vehicle class
class Car(Vehicle):
    def max_speed(self):
        print('Car max speed is 240')
    def change_gear(self):
        print('Car change 7 gear')
```

Đa hình

```
# Car Object
car = Car('Car x1', 'Red', 20000)
car.show()
# calls methods from Car class
car.max_speed()
car.change_gear()
# Vehicle Object
vehicle = Vehicle('Truck x1', 'white', 75000)
vehicle.show()
# calls method from a Vehicle class
vehicle.max_speed()
vehicle.change_gear()
```

```
Details: Car x1 Red 20000
Car max speed is 240
Car change 7 gear
Details: Truck x1 white 75000
Vehicle max speed is 150
Vehicle change 6 gear
```

Ví dụ thừa kế, Đa hình

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

    def speak(self):
        print("I am", self.name, "and I am", self.age, "years
old")
class Dog(Animal):
    def __init__(self, name, age):
        # This will call the Animal classes constructor
method
        super().__init__(name, age)

    def speak(self):
        super().speak()
        print("I am a Dog,  bow wow ")
```

Ví dụ thừa kế, Đa hình

```
class Cat(Animal):
    def __init__(self, name, age):
        # This will call the Animal classes constructor
method
        super().__init__(name, age)
    def speak(self):
        super().speak()
        print("I am a Cat, meow meoư")
########################################
tim = Dog("Tim", 5)
tom = Cat("Tom", 6)
ani = Animal("Ani", 1000)
animal_list = [tim, tom, ani]
for i in range(0, len(animal_list)):
    animal_list[i].speak()
```

```
I am Tim and I am 5 years old
I am a Dog,  bow wow
I am Tom and I am 6 years old
I am a Cat, meow meoư
I am Ani and I am 1000 years
old
```

Nạp chồng toán tử
Operator Overloading

```
class Book:
    def __init__(self, pages):
        self.pages = pages

    # Overloading + operator with magic method
    def __add__(self, other):
        return self.pages + other.pages

b1 = Book(400)
b2 = Book(300)
print("Total number of pages: ", b1 + b2)
```

Nạp chồng toán tử

```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
    def __mul__(self, timesheet):
        print('Worked for', timesheet.days, 'days')
        # calculate salary
        return self.salary * timesheet.days
class TimeSheet:
    def __init__(self, name, days):
        self.name = name
        self.days = days
emp = Employee("Jessa", 800)
timesheet = TimeSheet("Jessa", 50)
print("salary is: ", emp * timesheet)
```

➔Worked for 50 days
salary is: 40000

Magic methods available to perform overloading operations.

Operator Name	Symbol	Magic method
Addition	+	__add__(self, other)
Subtraction	-	__sub__(self, other)
Multiplication	*	__mul__(self, other)
Division	/	__div__(self, other)
Floor Division	//	__floordiv__(self,other)
Modulus	%	__mod__(self, other)
Power	**	__pow__(self, other)
Increment	+=	__iadd__(self, other)
Decrement	-=	__isub__(self, other)
Product	*=	__imul__(self, other)
Division	/+	__idiv__(self, other)
Modulus	%=	__imod__(self, other)
Power	**=	__ipow__(self, other)
Less than	<	__lt__(self, other)
Greater than	>	__gt__(self, other)
Less than or equal to	<=	__le__(self, other)
Greater than or equal to	>=	__ge__(self, other)
Equal to	==	__eq__(self, other)
Not equal	!=	__ne__(self, other)

Q & A

Thank you!