



TRƯỜNG ĐẠI HỌC THỦY LỢI
Khoa Công nghệ thông tin
Bộ môn Tin học và KTTT

LẬP TRÌNH PYTHON



Lập trình hướng đối tượng với Python

- Lập trình hướng đối tượng (OOP)
- Đối tượng (object)
- Lớp (class)
- Phương thức (method)
- Kế thừa
- Đa hình
- Đóng gói
- Trừu tượng
- Bài tập



Lập trình OOP với Python



- ❖ Python là một ngôn ngữ lập trình hướng đối tượng (OOP = Object Oriented Programming)
- ❖ Cho phép phát triển các ứng dụng bằng cách sử dụng phương pháp hướng đối tượng.
- ❖ Trong Python, dễ dàng tạo và sử dụng các lớp và các đối tượng.



1. Lớp và đối tượng trong Python



❖ Đối tượng (object)

- Đối tượng là một thực thể có trạng thái và hành vi.
- Nó có thể là bất kỳ đối tượng trong thế giới thực. Mọi thứ trong Python là một đối tượng và hầu hết mọi thứ đều có thuộc tính và phương thức.

❖ Lớp (class)

- Lớp có thể được định nghĩa là một tập hợp các đối tượng.
- Là một thực thể logic có một số thuộc tính và phương thức cụ thể.

❖ Phương thức

- Phương thức là một hàm được liên kết với một đối tượng.
- Trong Python, một phương thức không phải là duy nhất cho các thể hiện của lớp. Bất kỳ kiểu đối tượng nào cũng có thể có phương thức



1. Lớp và đối tượng trong Python



- ❖ Một lớp là một thực thể ảo và có thể được xem như một **bản thiết kế** của một đối tượng.
- ❖ Đối tượng là **thể hiện của một lớp**. Quá trình tạo một đối tượng có thể được gọi là khởi tạo.
- ❖ Mỗi lớp được thiết kế như một thực thể thống nhất, gồm 2 thành phần chính:
 - **Thuộc tính**: là các biến dữ liệu, dùng để lưu trữ các dữ liệu, thông tin để mô tả thực thể đó
 - **Phương thức**: là các chương trình con, dùng để tính toán những kết quả cần thiết, dựa trên dữ liệu lưu trữ trong các thuộc tính



1. Lớp và đối tượng trong Python



❖ Định nghĩa một lớp

```
class <Tên_lớp>:  
    [Khối lệnh]
```

❖ Tạo một đối tượng kiểu <Tên_lớp>

```
<Tên_đối_tượng> = <Tên_lớp> ([Các tham số])
```



1. Lớp và đối tượng trong Python



❖ Ví dụ 1:

#Định nghĩa lớp point

class point:

 x = 5

 y = 10

#Tạo đối tượng p1 kiểu point

p1 = point()

print('x = ', p1.x)

print('y = ', p1.y)

```
#Định nghĩa lớp point
class point:
    x = 5
    y = 10
#Tạo đối tượng p1 kiểu point
p1 = point()
print('x = ', p1.x)
print('y = ', p1.y)
```

➡ Kết quả:

```
x = 5
y = 10
```


❖ Xây dựng lớp point với 2 thuộc tính là x và y.

```
class point:
    x = 5
    y = 10
    def infor(self):
        print('x = ', self.x)
        print('y = ', self.y)

p = point()
p.infor()
```

Tham số
'self' là gì?

x = 5
y = 10
>>>

2. Hàm tạo của một lớp

❖ Cách viết hàm tạo:

```
class <Tên lớp>:
```

```
    def __init__(self, [các tham số]):
```

```
        [Khởi lệnh]
```

❖ Tham số **self** được sử dụng để tham chiếu đến từng đối tượng được tạo ra.

❖ Chương trình minh họa

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

p1 = point(10, 20)
p2 = point(15, 25)
print('p1.x = ', p1.x)
print('p1.y = ', p1.y)
print('p2.x = ', p2.x)
print('p2.y = ', p2.y)
```



```
p1.x = 10
p1.y = 20
p2.x = 15
p2.y = 25
>>>
```

❖ Chương trình minh họa

```
class point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def infor(self):
        print('x = ', self.x)
        print('y = ', self.y)

p = point(10, 20)
p.infor()
```



```
x = 10
y = 20
```

2. Hàm tạo của một lớp

- ❖ Cơ chế hoạt động của hàm tạo
 - Mỗi khi một đối tượng lớp được tạo ra, hàm tạo `__init__()` sẽ tự động được thực hiện.
 - Ví dụ: đối tượng `p = point(10, 20)`, hàm tạo sẽ tự động thực hiện với tham số `x = 10, y = 20`.
- ❖ Tham số "self" được sử dụng như một biến tham chiếu tham chiếu đến đối tượng lớp hiện tại. Nó luôn là đối số đầu tiên trong định nghĩa hàm. Tuy nhiên, việc sử dụng self là tùy chọn trong gọi hàm.

❖ Tham số "self" cũng có thể thay thế bởi một tên khác nhưng ý nghĩa của tham số này không thay đổi.

❖ Ví dụ 5

```
class Nguoi:
```

```
    def __init__(self1, ten, tuoi):
```

```
        self1.ten = ten
```

```
        self1.tuoi = tuoi
```

```
    def infor(self2):
```

```
        print("Ten:", self2.ten)
```

```
        print("Tuoi:", self2.tuoi)
```

```
p1 = Nguoi("Nguyen Thanh Thao", 20)
```

```
p1.infor()
```

```
class Nguoi:
    def __init__(self1, ten, tuoi):
        self1.ten = ten
        self1.tuoi = tuoi
    def infor(self2):
        print("Ten:", self2.ten)
        print("Tuoi:", self2.tuoi)

p1 = Nguoi("Nguyen Thanh Thao", 20)
p1.infor()
```



```
Ten: Nguyen Thanh Thao
Tuoi: 20
>>>
```

- ❖ Tên self đã được thay thế bằng self1, self2. Tuy nhiên chúng ta nên dùng tên self vì nó có tính gợi nhớ cao.

❖ Đếm số lượng đối tượng của một lớp

```
class Student:  
    count = 0  
  
    def __init__(self):  
        Student.count = Student.count + 1  
  
s1 = Student()  
s2 = Student()  
s3 = Student()  
print("Số lượng sinh viên là:", Student.count)
```



Số lượng sinh viên là: 3

❖ Ví dụ 7:

```
class Student:
```

```
    def __init__(self):
```

```
        print("Đây là constructor không tham số")
```

```
    def show(self, name):
```

```
        print("Xin chào!", name)
```

```
student = Student()
```

```
student.show("Thanh Thảo.")
```

❖ Ví dụ 7:

```
class Student:
    # Constructor không tham số
    def __init__(self):
        print("Đây là constructor không tham số")

    def show(self, name):
        print("Xin chào!", name)

student = Student()
student.show("Thanh Thảo")
```



Đây là constructor không tham số
Xin chào! Thanh Thảo



Hàm tạo có tham số



❖ Ví dụ 8:

class Student:

Constructor tham số

def __init__(self, name):

print("Đây là constructor tham số.")

self.name = name

def show(self):

print("Xin chào!", self.name)

student = Student("Thanh Thảo")

student.show()

❖ Ví dụ 8:

```
class Student:
    # Constructor tham số
    def __init__(self, name):
        print("Đây là constructor tham số.")
        self.name = name

    def show(self):
        print("Xin chào!", self.name)

student = Student("Thanh Thảo")
student.show()
```



Đây là constructor tham số.
Xin chào! Thanh Thảo



Ví dụ 9: hàm tạo có tham số



class Car:

thuộc tính lớp

loaixe = "Ô tô"

thuộc tính đối tượng

def __init__(self, tenxe, mausac, nguyenvlieu):

self.tenxe = tenxe

self.mausac = mausac

self.nguyenvlieu = nguyenvlieu

Porsche là Ô tô.

Toyota là Ô tô.

Lamborghini cũng là Ô tô.

Xe Toyota có màu Đỏ. Điện là nguyên liệu vận hành.

Xe Lamborghini có màu Vàng. Deisel là nguyên liệu vận hành.

Xe Porsche có màu Xanh. Gas là nguyên liệu vận hành.

toyota = Car("Toyota", "Đỏ", "Điện")

lamborghini = Car("Lamborghini", "Vàng", "Deisel")

porsche = Car("Porsche", "Xanh", "Gas")

print("Porsche là {}".format(porsche.loaixe))

print("Toyota là {}".format(toyota.loaixe))

print("Lamborghini cũng là {}".format(lamborghini.loaixe))

print("Xe {} có màu {}. {} là nguyên liệu vận hành.".format(toyota.tenxe, toyota.mausac, toyota.nguyenvlieu))

print("Xe {} có màu {}. {} là nguyên liệu vận hành.".format(lamborghini.tenxe, lamborghini.mausac, lamborghini.nguyenvlieu))

print("Xe {} có màu {}. {} là nguyên liệu vận hành.".format(porsche.tenxe, porsche.mausac, porsche.nguyenvlieu))



3. Thay đổi giá trị của các thuộc tính



❖ Các giá trị của thuộc tính của một đối tượng có thể thay đổi trong khi thực hiện chương trình

❖ Ví dụ 10:

```
class Người:
```

```
    def __init__(self, ten, tuoi):
```

```
        self.ten = ten
```

```
        self.tuoi = tuoi
```

```
    def infor(self):
```

```
        print("Ten:", self.ten)
```

```
        print("Tuoi:", self.tuoi)
```

```
p = Người("Nguyen Thanh Thao", 20)
```

```
p.infor()
```


```
p.tuoi = 22
```

```
print("Thông tin sau khi cập nhật:")
```

```
p.infor()
```

❖ Ví dụ 10:

```
class Nguoi:
    def __init__(self, ten, tuoi):
        self.ten = ten
        self.tuoi = tuoi
    def infor(self):
        print("Ten:", self.ten)
        print("Tuoi:", self.tuoi)
p = Nguoi("Nguyen Thanh Thao", 20)
p.infor()
p.tuoi = 22
print("Thông tin sau khi cập nhật:")
p.infor()
```



```
Ten: Nguyen Thanh Thao
Tuoi: 20
Thông tin sau khi cập nhật:
Ten: Nguyen Thanh Thao
Tuoi: 22
>>>
```



Các hàm lớp dựng sẵn của Python



Hàm	Mô tả
<code>getattr(obj, name, default)</code>	Sử dụng để truy cập thuộc tính của đối tượng.
<code>setattr(obj, name, value)</code>	Sử dụng để đặt một giá trị cụ thể cho thuộc tính cụ thể của một đối tượng.
<code>delattr(obj, name)</code>	Sử dụng để xóa một thuộc tính cụ thể.
<code>hasattr(obj, name)</code>	Trả về True nếu đối tượng chứa một thuộc tính cụ thể.

❖ Ví dụ 11:

```
class Student:
    def __init__(self, name, id, age):
        self.name = name;
        self.id = id;
        self.age = age

# tạo đối tượng của lớp Student
s = Student("Trung", 101, 22)
# in thuộc tính name của đối tượng s
print(getattr(s, 'name'))
# gán giá trị của age cho 23
setattr(s, "age", 23)
# in giá trị của age
print(getattr(s, 'age'))
# true nếu student chứa thuộc tính id
print(hasattr(s, 'id'))
# xóa thuộc tính age
delattr(s, 'age')
# in lỗi nếu age đã bị xóa
print(s.age)
```

Trung
23
True

AttributeError: 'Student' object has no attribute 'age'

Hàm	Mô tả
<code>__dict__</code>	Trả về dictionary chứa namespace của lớp.
<code>__doc__</code>	Chứa một chuỗi về tài liệu lớp.
<code>__name__</code>	Sử dụng để truy cập tên lớp.
<code>__module__</code>	Sử dụng để truy cập mô-đun trong đó, lớp này được định nghĩa.
<code>__bases__</code>	Chứa một tuple bao gồm tất cả các lớp cơ sở.

```
class Student:
```

```
    def __init__(self, name, id, age):
```

```
        self.name = name;
```

```
        self.id = id;
```

```
        self.age = age
```

```
Name: Trung, ID: 101, age: 22
None
{'name': 'Trung', 'id': 101, 'age': 22}
__main__
>>>
```

```
    def display_details(self):
```

```
        print("Name: %s, ID: %d, age: %d" %(self.name, self.id, self.age))
```

```
s = Student("Trung", 101, 22)
```

```
s.display_details()
```

```
print(s.__doc__)          #tra ve None
```

```
print(s.__dict__)        # tra ve Dict
```

```
print(s.__module__)      #tra ve __main__
```

4. Xóa một đối tượng

❖ Khi một đối tượng không cần cho việc lưu trữ thông tin, ta có thể xóa chúng để giải phóng bộ nhớ máy tính.

❖ Cú pháp:

```
del <Tên đối tượng>
```

❖ Ví dụ 13:

```
class Nguoi:
```

```
    def __init__(self, ten, tuoi):
```

```
        self.ten = ten
```

```
        self.tuoi = tuoi
```

```
    def infor(self):
```

```
        print("Ten:", self.ten)
```

```
        print("Tuoi:", self.tuoi)
```

```
p = Nguoi("Nguyen Thanh Thao", 20)
```

```
p.infor()
```

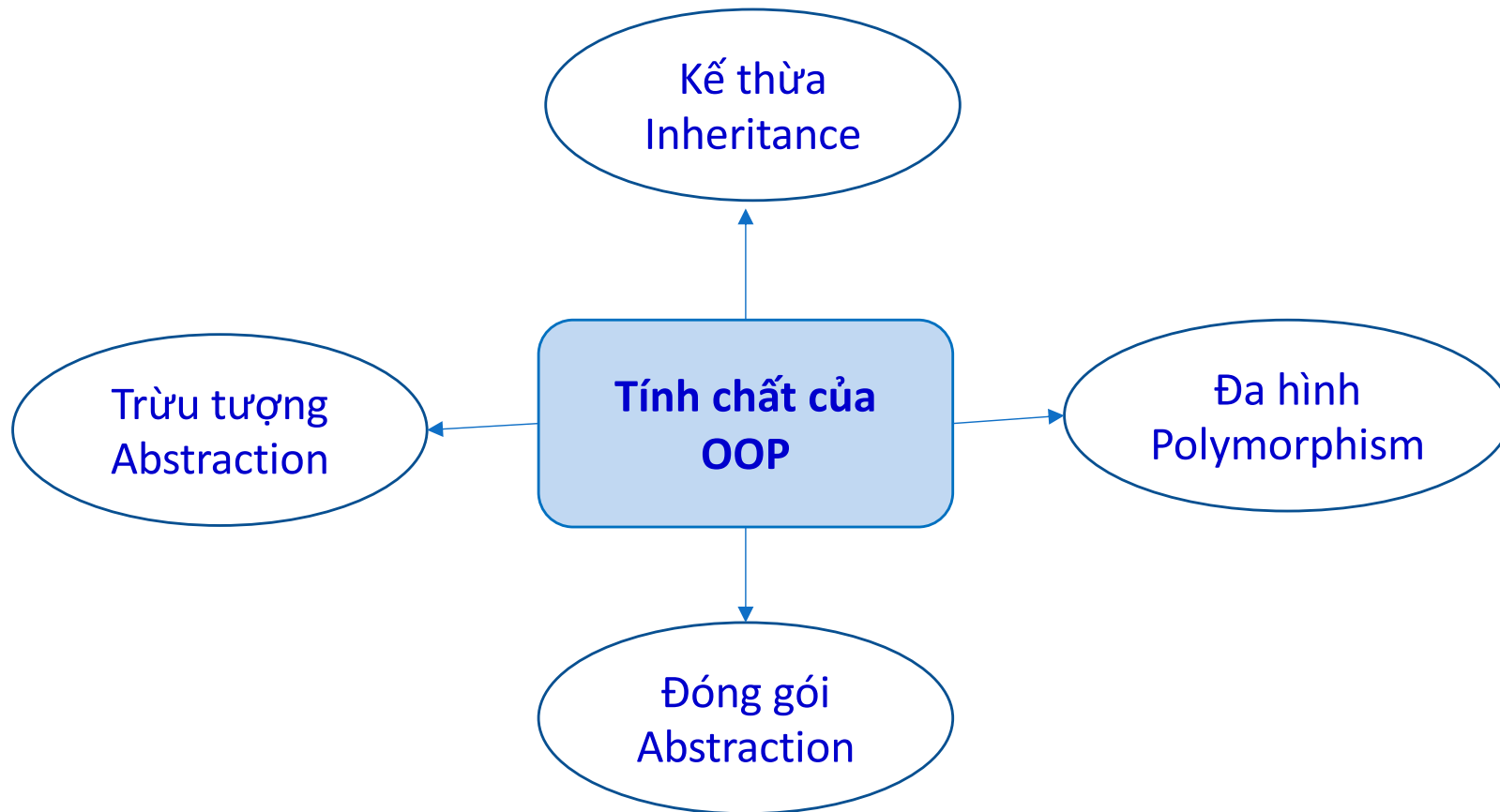
```
del p
```

```
p.infor()
```

Lỗi

```
NameError: name 'p' is not defined  
>>>
```

5. OOP với Python



❖ Đóng gói (Encapsulation)

- Sử dụng OOP trong Python, chúng ta có thể hạn chế quyền truy cập vào trạng thái bên trong của đối tượng. Điều này ngăn chặn dữ liệu bị sửa đổi trực tiếp, được gọi là **đóng gói**.
- Trong Python, biểu thị thuộc tính private này bằng cách sử dụng dấu gạch dưới làm tiền tố: "_" hoặc "__".

❖ Đa hình (Polymorphism)

- Tính đa hình là khái niệm mà hai hoặc nhiều lớp có những phương thức giống nhau nhưng có thể thực thi theo những cách thức khác nhau.

```
class Computer:

    def __init__(self):
        # Thuộc tính private ngăn chặn sửa đổi trực tiếp
        self.__maxprice = 900

    def sell(self):
        print("Giá bán sản phẩm: {}".format(self.__maxprice))

    def setMaxPrice(self, price):
        self.__maxprice = price

c = Computer()
c.sell()
# thay đổi giá.
c.__maxprice = 1000
c.sell()
# sử dụng hàm setMaxPrice để thay đổi giá.
c.setMaxPrice(1000)
c.sell()
```

Python đã coi `__maxprice` là thuộc tính private. Vậy nên để thay đổi giá trị, ta sử dụng hàm `setMaxPrice()`.

Giá bán sản phẩm: 900
Giá bán sản phẩm: 900
Giá bán sản phẩm: 1000


```
class Toyota:
    def dungxe(self):
        print("Toyota dừng xe để nạp điện")
    def nomay(self):
        print("Toyota nổ máy bằng hộp số tự động")

class Porsche:
    def dungxe(self):
        print("Porsche dừng xe để bơm xăng")
    def nomay(self):
        print("Porsche nổ máy bằng hộp số cơ")

# common interface
def kiểmtra_dungxe(car): car.dungxe()

# instantiate objects
toyota = Toyota()
porsche = Porsche()
# passing the object
kiểmtra_dungxe(toyota)
kiểmtra_dungxe(porsche)
```

- Hai lớp *Toyota* và *Porsche* đều có phương thức *dungxe()*. Tuy nhiên hàm của chúng khác nhau.
- Sử dụng tính đa hình để tạo hàm chung cho hai lớp, đó là *kiểmtra_dungxe()*.

```
Toyota dừng xe để nạp điện
Porsche dừng xe để bơm xăng
>>>
```




Trừu tượng dữ liệu trong Python



- ❖ Trừu tượng là một khía cạnh quan trọng của lập trình hướng đối tượng.
- ❖ Trong Python, có thể thực hiện ẩn dữ liệu bằng cách thêm dấu gạch dưới kép (__) làm tiền tố cho thuộc tính cần ẩn.
- ❖ Khi đó, thuộc tính sẽ không hiển thị bên ngoài lớp thông qua đối tượng.
- ❖ Ví dụ:

class Employee:

```
__count = 0;
```

```
def __init__(self):
```

```
    Employee.__count = Employee.__count + 1
```


```
def display(self):
```

```
    print("Số lượng nhân viên:", Employee.__count)
```

❖ Ví dụ:

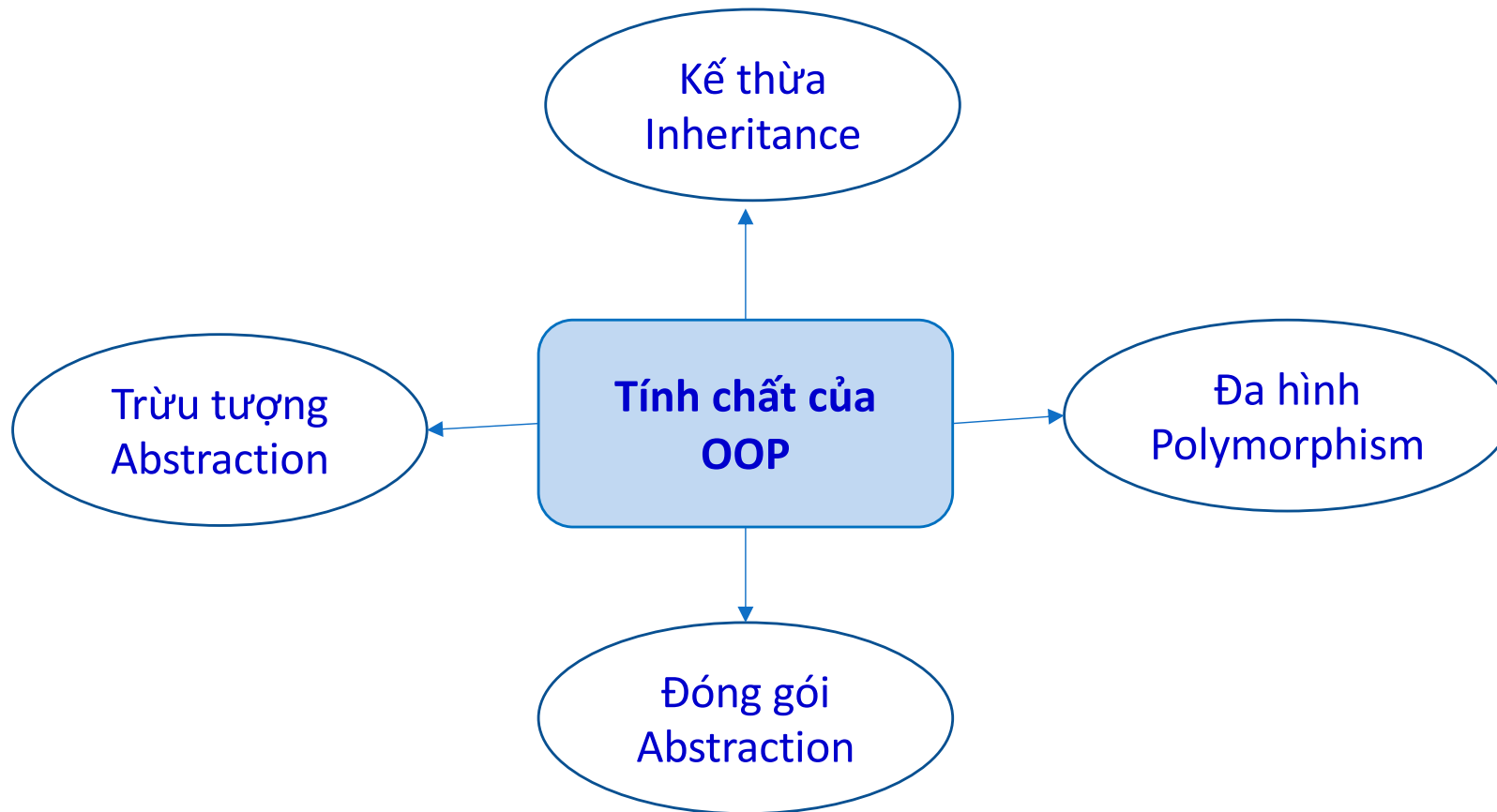
```
class Employee:
    __count = 0;
    def __init__(self):
        Employee.__count = Employee.__count + 1
    def display(self):
        print("Số lượng nhân viên:", Employee.__count)

emp1 = Employee()
try:
    print(emp1.__count)
except Exception as e:
    print(type(e), e)
finally:
    emp1.display()
```



```
<class 'AttributeError'> 'Employee' object has no attribute '__count'
Số lượng nhân viên: 1
>>>
```

5. OOP với Python





Sự kế thừa trong Python



Sự kế thừa trong Python



- ❖ Kế thừa là một tính chất quan trọng của lập trình hướng đối tượng.
- ❖ Kế thừa cung cấp khả năng sử dụng lại mã cho chương trình vì có thể sử dụng một lớp hiện có để tạo một lớp mới thay vì tạo nó từ đầu.
- ❖ Trong kế thừa, lớp con có được các thuộc tính và có thể truy cập tất cả các thành phần được định nghĩa trong lớp cha.
- ❖ Trong Python, một lớp dẫn xuất (hay còn gọi là lớp con - subclass) có thể kế thừa lớp cơ sở (lớp cha - superclass) bằng cách đề cập đến lớp cơ sở trong ngoặc sau tên lớp dẫn xuất.
 - Đơn kế thừa
 - Kế thừa đa cấp
 - Đa kế thừa

❖ Cú pháp:

```
class <Lớp con>(<Lớp cha>):  
    <Thân lớp>
```

❖ Ví dụ 14

```
class Nguoi:
```

```
...
```

```
class SinhVien(Nguoi):
```

```
    def __init__(self, ten, tuoi, nam):
```

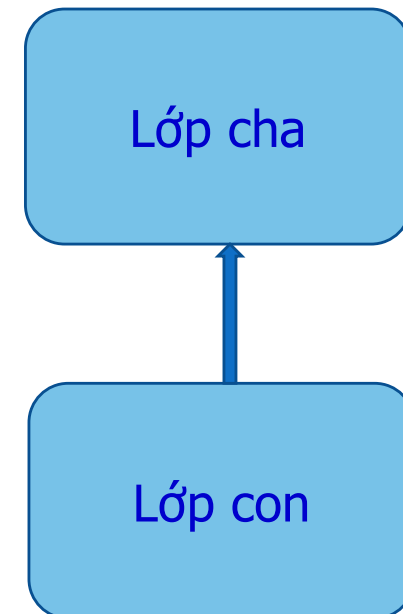
```
        Nguoi.__init__(self, ten, tuoi)
```

```
        self.totnghiep = nam
```

```
    def infor(self):
```

```
        Nguoi.infor(self)
```

```
        print("Nam tot nghiep:", self.totnghiep)
```



❖ Ví dụ 14:

```
class Nguoi:
    def __init__(self, ten, tuoi):
        self.ten = ten
        self.tuoi = tuoi
    def infor(self):
        print("Ten:", self.ten)
        print("Tuoi:", self.tuoi)
```

```
Ten: Nguyen Thanh Thao
Tuoi: 22
Nam tot nghiep: 2020
>>>
```

```
class SinhVien(Nguoi):
    def __init__(self, ten, tuoi, nam):
        Nguoi.__init__(self, ten, tuoi)
        self.totnghiep = nam
    def infor(self):
        Nguoi.infor(self)
        print("Nam tot nghiep:", self.totnghiep)
```

```
p = SinhVien("Nguyen Thanh Thao", 22, 2020)
p.infor()
```


❖ Ví dụ 15: Cập nhật thông tin các thuộc tính của đối tượng

```
class Nguoi:
    def __init__(self, ten, tuoi):
        self.ten = ten
        self.tuoi = tuoi
    def infor(self):
        print("Ten:", self.ten)
        print("Tuoi:", self.tuoi)

class SinhVien(Nguoi):
    def __init__(self, ten, tuoi, nam):
        Nguoi.__init__(self, ten, tuoi)
        self.totnghiep = nam
    def infor(self):
        Nguoi.infor(self)
        print("Nam tot nghiep:", self.totnghiep)

p = SinhVien("Nguyen Thanh Thao", 22, 2020)
p.infor()
p.ten = "Tran Thanh Thao"
p.totnghiep = 2021
print("Thong tin sau khi cap nhat:")
p.infor()
```

```
Ten: Nguyen Thanh Thao
Tuoi: 22
Nam tot nghiep: 2020
Thong tin sau khi cap nhat:
Ten: Tran Thanh Thao
Tuoi: 22
Nam tot nghiep: 2021
>>>
```


❖ Giả sử cho lớp **DaGiac** được tạo như sau:

```
class DaGiac:
    def __init__(self, socanh):
        self.n = socanh
        self.canh = [0 for i in range(socanh)]
    def nhapcanh(self):
        self.canh = [float(input("Nhập cạnh "+str(i+1)+": ")) for i in range(self.n)]
    def hienthicanh(self):
        for i in range(self.n):
            print("Giá trị cạnh", i+1, "là", self.canh[i])
```

❖ Hãy tạo lớp **TamGiac** kế thừa từ lớp **DaGiac**, và bổ sung thêm phương thức tính diện tích tam giác (giả sử các cạnh nhập thỏa mãn điều kiện)

❖ Tạo lớp **TamGiac** kế thừa từ lớp **DaGiac**

```
class DaGiac:
    def __init__(self, socanh):
        self.n = socanh
        self.canh = [0 for i in range(socanh)]
    def nhapcanh(self):
        self.canh = [float(input("Nhập cạnh "+str(i+1)+": ")) for i in range(self.n)]
    def hienthicanh(self):
        for i in range(self.n):
            print("Giá trị cạnh",i+1,"là",self.canh[i])

class TamGiac(DaGiac):
    def __init__(self):
        DaGiac.__init__(self,3)
    def dientich(self):
        a, b, c = self.canh
        # Tính nửa chu vi
        s = (a + b + c)/2
        area = (s*(s-a)*(s-b)*(s-c))**0.5
        print('Diện tích tam giác: %.2f' %area)

t = TamGiac()
t.nhapcanh()
t.hienthicanh()
t.dientich()
```

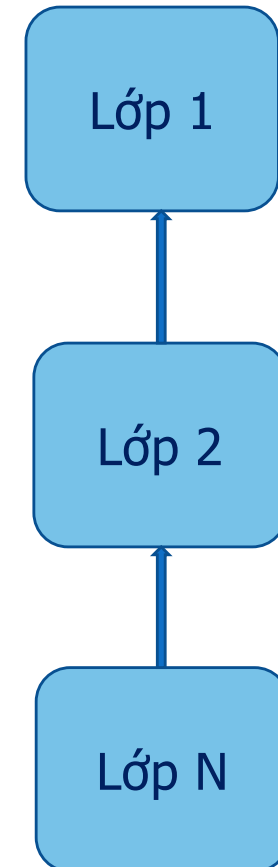
```
Nhập cạnh 1: 3
Nhập cạnh 2: 4
Nhập cạnh 3: 5
Giá trị cạnh 1 là 3.0
Giá trị cạnh 2 là 4.0
Giá trị cạnh 3 là 5.0
Diện tích tam giác: 6.00
>>>
```

❖ Kế thừa đa cấp:

- Kế thừa đa cấp khi một lớp dẫn xuất kế thừa một lớp dẫn xuất khác.
- Không có giới hạn về số lượng cấp độ, kế thừa đa cấp trong Python.

❖ Cú pháp

```
class <Lớp 1>:  
    <Thân lớp 1>  
class <Lớp 2>(<Lớp 1>):  
    <Thân lớp 2>  
...  
class <Lớp N>(<Lớp N-1>):  
    <Thân lớp N>
```



❖ Ví dụ 17:

```
class Animal:
    def speak(self):
        print("Animal Speaking")

# lớp con Dog kế thừa lớp Animal
class Dog(Animal):
    def bark(self):
        print("Gou gou!")

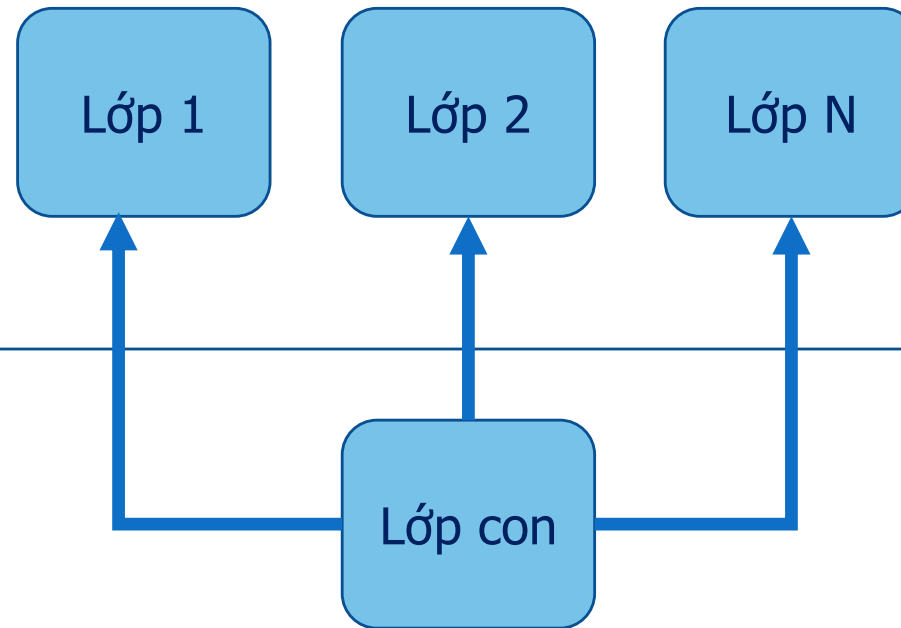
# lớp con Dogchild kế thừa lớp Dog
class DogChild(Dog):
    def eat(self):
        print("Eating milk...")

d = DogChild()
d.bark()
d.speak()
d.eat()
d.speak()
```

```
Gou gou!
Animal Speaking
Eating milk...
Animal Speaking
>>>
```

❖ Cú pháp:

```
class <Tên Lớp 1>:  
    <Thân lớp 1>  
class <Tên Lớp 2>:  
    <Thân lớp 2>  
...  
class <Tên Lớp N>:  
    <Thân lớp N>  
class <Tên Lớp Con>(<Tên Lớp 1>,<Tên Lớp 2>,...,<Tên Lớp N>):  
    <Thân lớp con>
```



❖ Ví dụ 18:

```
class Tinh1:
    def Cong(self, a, b):
        return a + b;

class Tinh2:
    def Nhan(self, a, b):
        return a * b;

class LopChia(Tinh1, Tinh2):
    def Chia(self, a, b):
        return a / b;

d = LopChia()
print(d.Cong(10, 20))
print(d.Nhan(10, 20))
print(d.Chia(10, 20))
```



```
30
200
0.5
>>>
```

- ❖ Phương thức **issubclass (sub, sup)**
 - Được sử dụng để kiểm tra mối quan hệ giữa các lớp được chỉ định.
 - Trả về **True** nếu lớp **sub** là lớp con của lớp **sup** và ngược lại là **False**.
- ❖ Phương thức **isinstance (obj, class)**
 - Được sử dụng để kiểm tra mối quan hệ giữa các đối tượng và các lớp.
 - Trả về **True** nếu tham số **obj** là thể hiện của tham số lớp **class**.

```
class Tinh1:
    def Cong(self, a, b):
        return a + b;

class Tinh2:
    def Nhan(self, a, b):
        return a * b;

class LopChia(Tinh1, Tinh2):
    def Chia(self, a, b):
        return a / b;

print("'LopChia' là con của 'Tinh2':", issubclass(LopChia, Tinh2))
print("'Tinh1' là con của 'Tinh2':", issubclass(Tinh1, Tinh2))
d = LopChia()
print("Đối tượng d là thể hiện của lớp 'LopChia':", isinstance(d, LopChia))
```



```
'LopChia' là con của 'Tinh2': True
'Tinh1' là con của 'Tinh2': False
Đối tượng d là thể hiện của lớp 'LopChia': True
```




Hàm super()



- ❖ Trong Python, hàm `super()` được sử dụng chính trong hai trường hợp:
 - Giúp tránh phải sử dụng tên lớp cơ sở một cách tường minh,
 - Xử lý Đa kế thừa

❖ Ví dụ 20: lớp SinhVien kế thừa từ lớp Nguoi:

```
class Nguoi:
    def __init__(self, ten, tuoi):
        self.ten = ten
        self.tuoi = tuoi
    def infor(self):
        print("Ten:", self.ten)
        print("Tuoi:", self.tuoi)

class SinhVien(Nguoi):
    def __init__(self, ten, tuoi, nam):
        Nguoi.__init__(self, ten, tuoi)
        self.totnghiep = nam
    def infor(self):
        Nguoi.infor(self)
        print("Nam tot nghiep:", self.totnghiep)

p = SinhVien("Nguyen Thanh Thao", 22, 2020)
p.infor()
```

Dùng
hàm
super()

❖ Ví dụ 20: Lớp `SinhVien` kế thừa từ lớp `Nguoi`:

```
class Nguoi:
    def __init__(self, ten, tuoi):
        self.ten = ten
        self.tuoi = tuoi
    def infor(self):
        print("Ten:", self.ten)
        print("Tuoi:", self.tuoi)

class SinhVien(Nguoi):
    def __init__(self, ten, tuoi, nam):
        super().__init__(ten, tuoi)
        self.totnghiep = nam
    def infor(self):
        super().infor()
        print("Nam tot nghiep:", self.totnghiep)

p = SinhVien("Nguyen Thanh Thao", 22, 2020)
p.infor()
```

Ten: Nguyen Thanh Thao
Tuoi: 22
Nam tot nghiep: 2020



6. Nạp chồng toán tử

6. Nạp chồng toán tử

- ❖ Một toán tử có thể được sử dụng để thực hiện nhiều hoạt động khác nhau.
 - Ví dụ với toán tử ' + ', có thể cộng số học hai số với nhau, có thể kết hợp hai danh sách, hoặc nối hai chuỗi khác nhau lại...
- ❖ Nạp chồng toán tử, cho phép cùng một toán tử được sử dụng khác nhau tùy từng ngữ cảnh.

```
class Point:
```

```
    def __init__(self, x = 0, y = 0):
```

```
        self.x = x
```

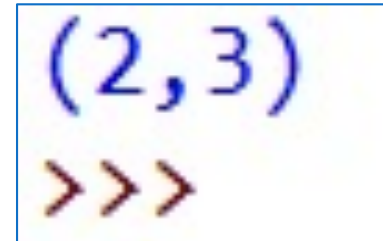
```
        self.y = y
```

```
    def __str__(self):
```

```
        return "({0},{1})".format(self.x, self.y)
```

```
p1 = Point(2,3)
```

```
print(p1)
```



```
(2,3)  
>>>
```

❖ Một số hàm đặc biệt dùng cho nạp chồng toán tử

Ký hiệu phép toán	Phương thức thực hiện
+	<code>__add__(self, other)</code>
-	<code>__sub__(self, other)</code>
*	<code>__mul__(self, other)</code>
**	<code>__pow__(self, other)</code>
/	<code>__truediv__(self, other)</code>
//	<code>__floordiv__(self, other)</code>
%	<code>__mod__(self, other)</code>

- ❖ Nạp chồng toán tử '+', sử dụng hàm `__add__()` trong class.
- ❖ Ví dụ cộng hai điểm tọa độ

```
class Point:
    def __init__(self, x = 0, y = 0):
        self.x = x
        self.y = y

    def __str__(self):
        return "({0},{1})".format(self.x,self.y)

    def __add__(self, other):
        x = self.x + other.x
        y = self.y + other.y
        return Point(x,y)

p1 = Point(2,3)
p2 = Point(-1,2)
print(p1 + p2)
```



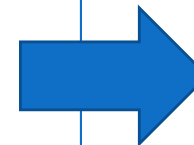
```
(1,5)
>>>
```


- ❖ Tạo lớp **Diem** với 2 thuộc tính hoành độ x , tung độ y .
- Nạp chồng toán tử '+' để cộng 2 điểm.
- Cho 3 điểm A, B, C . Tính tọa độ tổng $A+B, A+C, B+C$.

- ❖ Tạo lớp **Diem** với 2 thuộc tính hoành độ x, tung độ y. Nạp chồng toán tử '+' để cộng 2 điểm. Cho 3 điểm A, B, C. Tính tọa độ tổng A+B, A+C, B+C.

```
class Diem:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __str__(self):
        return "({},{})".format(self.x, self.y)
    def __add__(self, other):
        x = self.x + other.x
        y = self.y + other.y
        return Diem(x,y)
```

```
A = Diem(10,20)
B = Diem(15,25)
C = Diem(20,30)
print("A:", A.x, A.y)
print("B:", B.x, B.y)
print("C:", C.x, C.y)
print("{}+{} = {}".format(A,B,A+B))
print("{}+{} = {}".format(A,C,A+C))
print("{}+{} = {}".format(B,C,B+C))
```



```
A: 10 20
B: 15 25
C: 20 30
(10,20)+(15,25) = (25,45)
(10,20)+(20,30) = (30,50)
(15,25)+(20,30) = (35,55)
```

6. Nạp chồng toán tử

❖ Nạp chồng toán tử thao tác trên bit

Ký hiệu phép toán	Phương thức thực hiện
<< (Dịch trái)	<code>__lshift__(self, other)</code>
>> (Dịch phải)	<code>__rshift__(self, other)</code>
& (AND)	<code>__and__(self, other)</code>
(OR)	<code>__or__(self, other)</code>
^ (XOR)	<code>__xor__(self, other)</code>
~ (NOT)	<code>__invert__(self)</code>

❖ Nạp chồng toán tử so sánh trong Python

Ký hiệu phép toán	Phương thức thực hiện
<	<code>__lt__(self, other)</code>
>	<code>__gt__(self, other)</code>
<=	<code>__le__(self, other)</code>
>=	<code>__ge__(self, other)</code>
==	<code>__eq__(self, other)</code>
!=	<code>__ne__(self, other)</code>

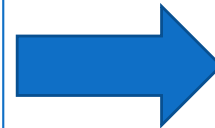
- ❖ Xây dựng lớp 'Nguoi' có thuộc tính họ tên và tuổi, có phép toán so sánh theo tuổi.
- ❖ Sử dụng phép toán so sánh để sắp xếp một danh sách các đối tượng theo tuổi.

```
Ten: Tran Hai Van  
Tuoi: 19  
Ten: Nguyen Thanh Thao  
Tuoi: 20  
Ten: Le Thanh Hien  
Tuoi: 21  
Ten: Nguyen Thanh Hung  
Tuoi: 22  
>>>
```

- ❖ Xây dựng lớp 'Nguoi' có thuộc tính họ tên và tuổi, có phép toán so sánh theo tuổi. Sử dụng phép toán so sánh để sắp xếp một danh sách các đối tượng theo tuổi.

```
class Nguoi:
    def __init__(self, ten, tuoi):
        self.ten = ten
        self.tuoi = tuoi
    def infor(self):
        print("Ten:", self.ten)
        print("Tuoi:", self.tuoi)
    def __lt__(self, other):
        return self.tuoi < other.tuoi

p1 = Nguoi("Nguyen Thanh Thao", 20)
p2 = Nguoi("Tran Hai Van", 19)
p3 = Nguoi("Nguyen Thanh Hung", 22)
p4 = Nguoi("Le Thanh Hien", 21)
ds = [p1,p2,p3,p4]
ds.sort()
for x in ds:
    x.infor()
```



```
Ten: Tran Hai Van
Tuoi: 19
Ten: Nguyen Thanh Thao
Tuoi: 20
Ten: Le Thanh Hien
Tuoi: 21
Ten: Nguyen Thanh Hung
Tuoi: 22
>>>
```

- ❖ Xây dựng lớp 'vecto' có thuộc tính x và y , có phép toán cộng, trừ và tích vô hướng.
- ❖ Cho 2 véc tơ u và v . Tính tổng, hiệu, tích vô hướng của 2 véc tơ.

```
u = (1,2)
v = (3,4)
Tong hai vector: (4, 6)
Hieu hai vector: (-2, -2)
Tich vo huong: 11
>>>
```


- ❖ Xây dựng lớp 'vecto' có thuộc tính x và y, có phép toán cộng, trừ và tích vô hướng. Cho 2 véc tơ u và v. Tính tổng, hiệu, tích vô hướng của 2 véc tơ.

```
class vecto:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __str__(self):
        return "({},{})".format(self.x,self.y)
    def __add__(self, other):
        return self.x + other.x, self.y + other.y
    def __sub__(self, other):
        return self.x - other.x, self.y - other.y
    def __mul__(self, other):
        return self.x * other.x + self.y * other.y
```

```
u = vecto(1,2); print('u =',u)
v = vecto(3,4); print('v =',v)
print('Tong hai vector:', u + v)
print('Hieu hai vector:', u - v)
print('Tich vo huong:', u * v)
```

```
u = (1,2)
v = (3,4)
Tong hai vector: (4, 6)
Hieu hai vector: (-2, -2)
Tich vo huong: 11
>>>
```




Tổng kết: OOP trong Python



- Lớp và đối tượng
- Phương thức
- Kế thừa/ Đa hình / Đóng gói / Trừu tượng
- Ưu điểm:
 - Lập trình trở nên dễ dàng và hiệu quả hơn.
 - Lớp (class) có thể chia sẻ được nên code dễ dàng được sử dụng lại.
 - Năng suất của chương trình tăng lên
 - Dữ liệu an toàn và bảo mật với trừu tượng hóa dữ liệu.



Bài tập



Bài tập



1. Xây dựng lớp Rectangle lưu thông tin về Hình chữ nhật có thuộc tính: chiều dài, chiều rộng và các phương thức tính chu vi và diện tích HCN.
2. Sắp xếp danh sách các đối tượng Hình chữ nhật.

Với lớp Rectangle ở bài 1, kế thừa và bổ sung thêm phương thức so sánh 2 đối tượng theo tiêu chí A nhỏ hơn B nếu thỏa mãn 1 trong 2 điều kiện:

- Diện tích của A nhỏ hơn B
- Diện tích của A bằng diện tích của B, nhưng chu vi của A nhỏ hơn chu vi của B.

Nhập 1 danh sách đối tượng và đưa ra DS đã được sắp xếp.

3. Cho dãy số a_0, a_1, \dots, a_{n-1} . Hãy thực hiện các công việc:

- Tính tổng các phần tử của dãy
- Tìm phần tử nhỏ nhất/ lớn nhất
- Liệt kê các bộ có 2 phần tử có tổng bằng 0.

- ❖ Xây dựng lớp Rectangle lưu thông tin về Hình chữ nhật có thuộc tính: chiều dài, chiều rộng và các phương thức tính chu vi và diện tích HCN.
- ❖ Hướng dẫn:
 - Sử dụng định nghĩa class xây dựng lớp Rectangle với hai thuộc tính chiều dài và chiều rộng.
 - Phương thức chu vi bằng tổng 4 cạnh
 - Phương thức diện tích bằng tích 2 cạnh dài và rộng

```
Nhap Hình chu nhât:  
Chieu dai: 10  
Chieu rong: 8  
Chu vi: 36.0  
Dien tich: 80.0  
>>>
```


```
class Rectangle:

    def __init__(self, dai, rong):
        self.dai = dai
        self.rong = rong

    def Chuvi(self):
        return 2*(self.dai+ self.rong)

    def Dientich(self):
        return self.dai * self.rong

print("Nhap Hinh chu nhat:")
dai = float(input("Chieu dai: "))
rong = float(input("Chieu rong: "))
hcn = Rectangle(dai,rong)
print("Chu vi:", hcn.Chuvi())
print("Dien tich:", hcn.Dientich())
```



```
Nhap Hinh chu nhat:
Chieu dai: 10
Chieu rong: 8
Chu vi: 36.0
Dien tich: 80.0
>>>
```



Bài tập 2



Với lớp Rectangle ở bài 1, kế thừa và bổ sung thêm phương thức so sánh hai đối tượng theo tiêu chí A nhỏ hơn B nếu thỏa mãn 1 trong 2 điều kiện:

- Diện tích của A nhỏ hơn B
- Diện tích của A bằng diện tích của B, nhưng chu vi của A nhỏ hơn chu vi của B.

Nhập 1 danh sách đối tượng và đưa ra DS đã được sắp xếp.

```
n = 4
Nhap thong tin cua n HCN:
Dai: 6
Rong: 4
Dai: 12
Rong: 4
Dai: 8
Rong: 6
Dai: 9
Rong: 7
Danh sach HCN da sap xep:
Dai      Rong      Dien tich      Chu vi
6.0      4.0      24.0      20.0
8.0      6.0      48.0      28.0
12.0     4.0      48.0      32.0
9.0      7.0      63.0      32.0
```

```
class Rectangle:
    def __init__(self, dai, rong):
        self.dai = dai
        self.rong = rong
    def Chuvi(self):
        return 2*(self.dai+ self.rong)
    def Dientich(self):
        return self.dai * self.rong
```

```
class RectangleSort(Rectangle):
    def __init__(self, dai, rong):
        super().__init__(dai,rong)
    def __lt__(self, other):
        dt_self = self.Dientich()
        dt_other = other.Dientich()
        cv_self = self.Chuvi()
        cv_other = other.Chuvi()
        if dt_self < dt_other:
            return True
        if dt_self == dt_other and cv_self < cv_other:
            return True
        return False
```

```
n = int(input("n = "))
print("Nhap thông tin của n HCN:")
L = []
for i in range(n):
    dai = float(input("Dai: "))
    rong = float(input("Rong: "))
    L.append(RectangleSort(dai,rong))
```

```
L.sort()
print("Danh sách HCN đã sắp xếp:")
print("Dai\tRong\tDien tích\tChu vi")
for x in L:
    print(x.dai,"\t", x.rong,"\t", x.Dientich(),"\t\t",x.Chuvi())
```




Bài tập 3



Cho dãy số a_0, a_1, \dots, a_{n-1} . Hãy thực hiện các công việc:

- Tính tổng các phần tử của dãy
- Tìm phần tử nhỏ nhất/ lớn nhất
- Liệt kê các bộ có 2 phần tử có tổng bằng 0.