

CHƯƠNG 1

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Giảng viên: TS.Cao Diệp Thắng
Khoa: Khoa học Ứng dụng

Chương 1.

MỤC TIÊU

- + Nắm vững khái niệm lớp và đối tượng trong Python.
- + Hiểu về kế thừa và đa kế thừa, cách sử dụng và ứng dụng trong thiết kế lớp.
- + Cách sử dụng phương thức và thuộc tính tĩnh, hiểu được ý nghĩa và ứng dụng.
- + Phương pháp áp dụng đa hình và ghi đè phương thức, làm cho mã code linh hoạt và tái sử dụng được.
- + Cách tạo các lớp trừu tượng, hiểu ý nghĩa và cách sử dụng để định nghĩa hành vi chung cho các lớp con.
- + Xử lý ngoại lệ một cách hợp lý, thiết kế OOP hiệu quả để tăng tính ổn định và dễ bảo trì của chương trình.

NỘI DUNG BÀI HỌC

1.1

KHÁI NIỆM LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

1.2

LỚP VÀ ĐỐI TƯỢNG

1.3

CÁC TÍNH CHẤT CỦA HƯỚNG ĐỐI TƯỢNG

MỤC TIÊU BÀI HỌC

Sau khi học xong bài này, người học sẽ nắm được các vấn đề sau:

- Hiểu được khái niệm lớp, đối tượng.
- Cách khởi tạo lớp, khai báo đối tượng.
- Cách xây dựng hàm (phương thức) lớp,
- Làm việc với thuộc tính và phương thức của lớp.
- Phương thức và thuộc tính tĩnh: cách sử dụng phương thức và thuộc tính tĩnh để xử lý dữ liệu chung của lớp.
- Các tính chất và áp dụng của lập trình hướng đối tượng.

1.1. KHÁI NIỆM LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

- ❑ Lập trình hướng đối tượng (Object-Oriented Programming – OOP) là một phương pháp lập trình sử dụng các đối tượng (Object) để xây dựng hệ thống phần mềm.
- ❑ OOP sẽ chia nhỏ chương trình thành các đối tượng và các mối quan hệ.

Python là một ngôn ngữ lập trình đa mô hình.

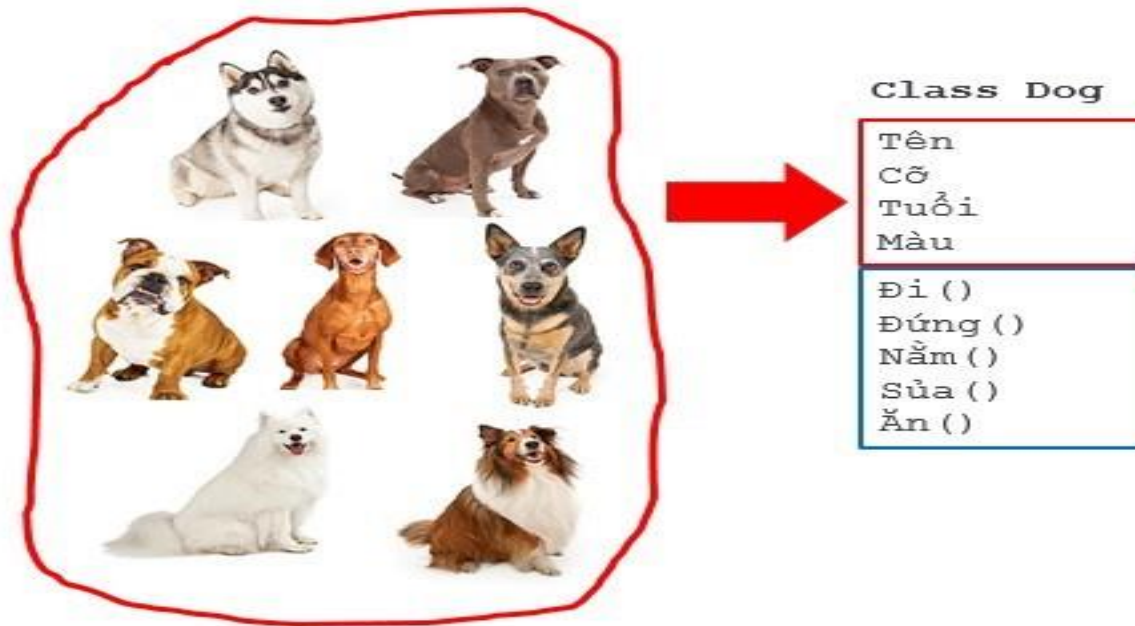
- Python hỗ trợ các cách tiếp cận lập trình khác nhau.
- Một trong những cách tiếp cận phổ biến để giải quyết vấn đề lập trình là tạo các **đối tượng**.

Một đối tượng sẽ có hai thành phần chính:

- **Thuộc tính** (dữ liệu)
- **Hành vi** (phương thức)

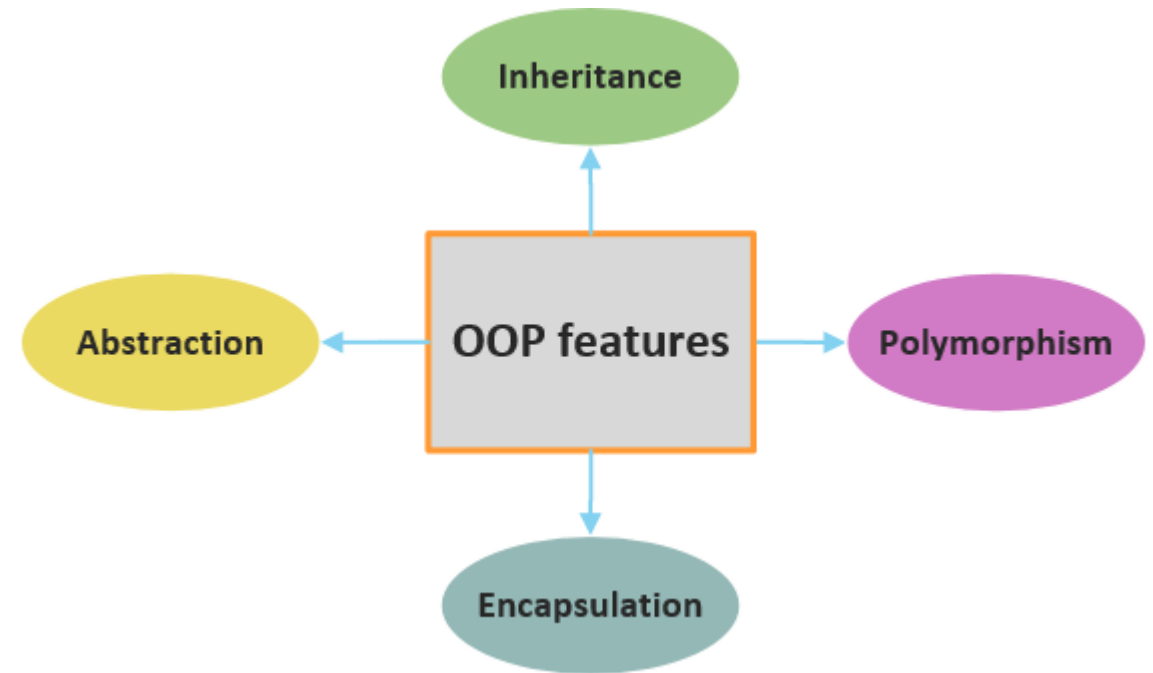
Ví dụ 1.1: Một con chó được xem như một đối tượng với các đặc tính sau:

- **Tên, cỡ, tuổi, màu,...** được xem là các *thuộc tính (Attribute)*.
- **Đi, đứng, nằm, sủa, ăn,..** sẽ được xem là các *hành vi /phương thức (Method)*.



❑ Các nguyên lý (đặc trưng) của OOPS

- **Tính kế thừa (Inheritance):** cho phép một lớp (class) có thể kế thừa các thuộc tính và phương thức từ các lớp khác đã được định nghĩa.
- **Tính đóng gói (Encapsulation):** là quy tắc yêu cầu trạng thái bên trong của một đối tượng được bảo vệ và tránh truy cập được từ code bên ngoài (tức là code bên ngoài không thể trực tiếp nhìn thấy và thay đổi trạng thái của đối tượng đó).
- **Tính đa hình (Polymorphism):** 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*.
- **Tính trừu tượng (Abstract):** trong ngôn ngữ Python, tính đa hình không được rõ nét. Python không cho phép khai báo lớp trừu tượng mà chỉ tạo ra một lớp *Abstract Base Class (ABC)* để người dùng thừa kế nếu muốn thực thi tính trừu tượng.



Hình 1.1. Các nguyên lý của lập trình hướng đối tượng [4,8,9]

1.2. LỚP VÀ ĐỐI TƯỢNG

1.2.1. Lớp (class). Trung tâm của lập trình hướng đối tượng là lớp (class).

- ☐ Một lớp là một bản thiết kế hay một bản mẫu/khuôn mẫu trong đó chúng ta khai báo các thuộc tính (**attribute**) và phương thức (**method**) để mô tả từ đó tạo ra được các object(đối tượng).
- ☐ Trong lập trình hướng đối tượng, một lớp đại diện cho tập hợp các đối tượng có cùng các *đặc điểm, hành vi, phương thức* hoạt động. Có thể xem lớp là sự kết hợp các thành phần **dữ liệu** và các **phương thức**.
- ☐ Khi tạo một lớp mới, chúng ta đã tạo ra một kiểu dữ liệu mới. Đây là kiểu dữ liệu do người dùng định nghĩa và khởi tạo.

Với ví dụ 1.1 về lớp con chó ở trên thì chúng ta có thể hình dung class sẽ như một bản mẫu phác thảo về con chó. Nó sẽ chứa tất cả các chi tiết về tên, màu sắc v.v... về con chó.

Các đặc điểm cơ bản về class trong Python:

- Các class đều được tạo ra bằng từ khóa **class**.
- Các thuộc tính chính là các biến thuộc về class.
- Các thuộc tính thường là **public**, và có thể được truy cập tới bằng cách sử dụng toán tử dấu chấm '.', ví dụ: Myclass.Myattribute.

Trong đó:

- *Đặc điểm của đối tượng* được thể hiện ra bên ngoài là các *thuộc tính* (property) của lớp;
- Các hành vi hay *phương thức hoạt động* của đối tượng gọi là phương thức của lớp.
- Các thuộc tính và phương thức được gọi chung là *thành viên (Member)* của lớp.

a. Khai báo class

Lớp được khai báo bởi từ khóa **class**, tên của lớp và dấu hai chấm ':'

```
class className:
```

```
    # khai báo thuộc tính của lớp
```

```
    # khai báo thuộc tính của đối tượng
```

```
    # khai báo phương thức
```

Lưu ý: mỗi lớp được liên kết với một chuỗi tài liệu có thể được truy cập bằng cách sử dụng **__doc__**

Một lớp chứa một bộ câu lệnh bao gồm các **trường(thuộc tính)**, Hàm khởi tạo lớp(constructor), phương thức, v.v.

Ví dụ 1.2.1.1

In[1]:	<pre>1 class Nhan_vien: 2 id = 10 3 name = "Nguyễn Văn A" 4 age = 40 5 6 def display (self): 7 print(self.id, self.name, self.age) 8 9 nv1 = Nhan_vien() 10 nv1.display()</pre>	<p>id, name, age là các thuộc tính của lớp, thuộc tính này được chia sẻ bởi tất cả các đối tượng thuộc lớp này.</p> <p>self.id, self.name, self.age là khai báo thuộc tính của đối tượng. Đây là các thuộc tính của từng đối tượng riêng biệt của lớp.</p>
Out[1]:	10 Nguyễn Văn A 28	

Tham số self

- ❑ Các phương thức của lớp đều phải có thêm một ***tham số đầu tiên*** trong phần định nghĩa phương thức.

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, việc sử dụng ký hiệu '**self**' là **tùy chọn** trong gọi hàm.

- ❑ Giả sử chúng ta có một phương thức mà không nhận vào bất kỳ đối số (argument) nào, tuy nhiên, thực tế là phương thức đó vẫn sẽ có một đối số được ẩn đi, chính là **self**, xem hàm `display()` trong ví dụ 1.2.1.1 ở trên.
- ❑ **self** trong Python thì tương đương với con trỏ `this` trong C++ và tham chiếu `this` trong Java [8].

Lưu ý: Trong Python, việc sử dụng **self** là một quy ước thông thường và đúng đắn trong việc định nghĩa và gọi phương thức của một lớp. Tuy nhiên, nó không bắt buộc phải được gọi là **self**. Thực tế, chúng ta có thể sử dụng bất kỳ tên nào bạn muốn cho đối số đầu tiên trong phương thức, nhưng **self** là tên phổ biến và được nhiều người lập trình Python ưa chuộng. Tên này có ý nghĩa là "**đối tượng hiện tại**" và giúp làm cho code dễ đọc hơn.

Tham số self, ...

Ví dụ

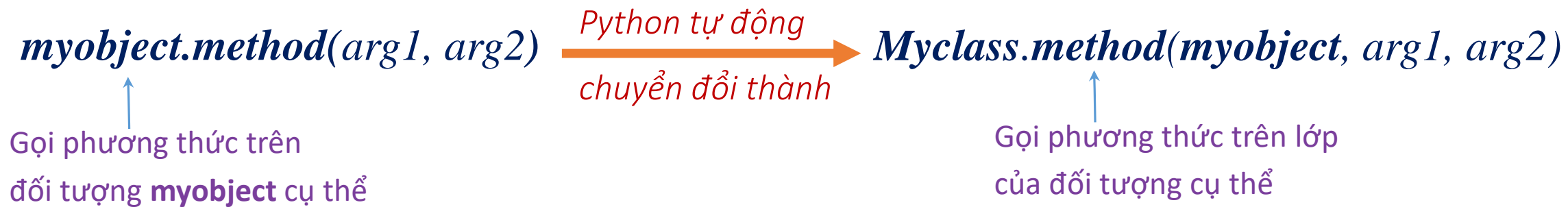
```
1  class MyClass:
2      def __init__(self,value):
3          self.value=value
4
5      def my_method(self):
6          print('Giá trị của đối tượng :',self.value)
7
8  #Tạo một đối tượng từ Lớp MyClass
9  obj = MyClass(50)    #Khởi tạo và khởi tạo đối tượng obj từ lớp MyClass
10
11 #Gọi phương thức my_method() của đối tượng obj
12 obj.my_method()
```

Out[]. Giá trị của đối tượng : 50

Tham số *self*, ...



Khi gọi đến một phương thức của một đối tượng cụ thể, ví dụ:



Cơ chế này giúp Python hiểu rằng chúng ta muốn gọi phương thức `method` trên đối tượng **myobject** và truyền **arg1** và **arg2** vào nó. Python sẽ tự động truyền đối tượng **myobject** làm đối số đầu tiên (**self**), sau đó ta có thể truy cập các thuộc tính và phương thức của đối tượng **myobject** bên trong phương thức `method`.

Đây chính là bản chất đặc biệt của **self** trong Python, giúp cho lập trình viên làm việc với đối tượng một cách dễ dàng và tự nhiên trong Python, giúp quản lý và tương tác với dữ liệu của đối tượng một cách hiệu quả.

Việc tự động truyền đối tượng **self** là một phần quan trọng của cách lập trình hướng đối tượng hoạt động trong Python.

b. Phương thức

- ❑ Phương thức là tập hợp các lệnh Python dùng để thực hiện một cách trọn vẹn một nhiệm vụ nào đó.
- ❑ Phương thức là một nhóm code có chức năng thực hiện một tác vụ cụ thể bên trong chương trình Python.
 - Hàm mà thuộc về một *class* cụ thể thì được gọi là **Phương thức**.
 - Tất cả các phương thức đều yêu cầu tham số '**self**'.
- ❑ Để tạo ra một phương thức mới, ta sử dụng từ khóa '**def**'.

Cú pháp `def MethodName(self, <các tham số khác nếu có>):`
`# Các Lệnh của phương thức`

- Tham số đầu tiên của phương thức luôn luôn là *self*, ngoài ra có thể bao gồm các tham số khác (xem phương thức `Eat`, `Lie`, `Stay`[]).
- *self* là tham số đặc biệt, nó ám chỉ đối tượng mà chúng ta đang làm việc. Khi gọi phương thức, không bao giờ truyền giá trị cho tham số này.

Ví dụ 1.2.1.2.

In[2]:

```
1 class Dog:
2
3     DogCount = 0 # Thuộc tính của lớp
4     #hàm khởi tạo đối tượng
5     def __init__(self, name, size, age, color):
6         self.name = name # Thuộc tính của đối tượng
7         self.size = size # Thuộc tính của đối tượng
8         self.age = age # Thuộc tính của đối tượng
9         self.color = color # Thuộc tính của đối tượng
10
11     def Go(self):
12         print("I'm going...")
13
14     def Stay(self, place):
15         print("I'm staying at {}".format(place))
16
17     def Lie(self, place):
18         print("I'm lying at {}".format(place))
19
20     def Bark(self): print("Whoop...")
21
22     def Eat(self, food):
23         print("I'm eating {}".format(food))
24
25 # khởi tạo đối tượng
26 bull = Dog("Bull", "large", 2, "Yellow");
27 bull.Stay("garden")
28 bull.Bark()
```

Hành vi/hoạt động "Đi"

Hành vi/hoạt động "Ở"

Hành vi/hoạt động "Nằm xuống"

Hành vi/hoạt động "sủa"

Hành vi/hoạt động "Ăn"

Ví dụ 1.2.1.2.

Out[2]:	I'm staying at garden Whoop...
----------------	-----------------------------------

Trong ví dụ : **Go()**, **Stay(self, palace)**, **Lie(self,place)**, **Bark()**, **Eat(self,food)** là phương thức mô phỏng các hành vi/hoạt động: **Đi** lại, **Ở** một vị trí (palace), **Nằm** xuống ở một nơi cụ thể (place), **Sủa**, **Ăn** một thức ăn cụ thể (food).

Cú pháp để gọi các phương thức của đối tượng:

Tên_đối_tượng.Tên_phương_phức(<tham số>)

Constructor trong python

Constructor trong Python là một loại phương thức (hàm) đặc biệt được sử dụng để khởi tạo các thể hiện của lớp.

Constructor có thể có hai loại.

- Constructor có tham số.
- Constructor không tham số.

Tạo constructor trong Python

❑ Phương thức `__init__()`

- Là một phương thức đặc biệt của lớp.
- Chạy ngay sau khi một đối tượng của một lớp được thể hiện.
- Được thực thi khi khởi tạo đối tượng.

Cú pháp

```
def __init__(self, <các tham số khác nếu có>):  
    # body of the constructor
```

Ví dụ 1.2.1.3. constrcutor không tham số:

In[2]:	<pre>1 class Dog: 2 <i>#Constructor không tham số</i> 3 def __init__(self): 4 print('Đây là constructor không tham số') 5 6 def display(self,name): 7 print('Hello',name) 8 9 bull = Dog() 10 bull.display("The Dom")</pre>
Out[2]:	<pre>Đây là constructor không tham số Hello The Dom</pre>

Ví dụ 1.2.1.4. constructor có tham số.

In[3]:	<pre>1 #constructor có tham số 2 class Dog: 3 #Constructor có tham số 4 def __init__(self, name): 5 print('Đây là constructor có tham số.') 6 self.name=name 7 8 def display(self): 9 print("Hello",self.name) 10 11 bull = Dog("The Dom") 12 bull.display()</pre>
Out[3]:	<pre>Đây là constructor không tham số Hello The Dom</pre>

Đoạn code dưới đây là phương thức khởi tạo cho lớp **Dog**:

```
def __init__(self, name, size, age, color):  
    self.name = name    # Thuộc tính của đối tượng  
    self.size = size    # Thuộc tính của đối tượng  
    self.age = age      # Thuộc tính của đối tượng  
    self.color = color  # Thuộc tính của đối tượng
```

Trong đoạn code trên:

- Ngoài tham số mặc định self, còn có tham số khác là name, size, age, color.
- name, size, age, color là các thuộc tính của đối tượng thuộc lớp Dog.

Sau đây là ví dụ hoàn chỉnh về lớp dog và một số phương thức của nó:

Ví dụ 1.2.1.5.



In[4]:	<pre> 1 class Dog(): 2 DogCount = 0 <i>#Thuộc tính của Lớp</i> 3 4 <i>#hàm khởi tạo đối tượng</i> 5 def init(self, name, size, age, color): 6 self.name = name <i># Thuộc tính của đối tượng</i> 7 self.size = size <i># Thuộc tính của đối tượng</i> 8 self.age = age <i># Thuộc tính của đối tượng</i> 9 self.color = color <i># Thuộc tính của đối tượng</i> 10 11 def Go(self): 12 print("I'm going...") 13 14 def Stay(self, place): 15 print("I'm staying at {}".format(place)) 16 17 def Lie(self, place): 18 print("I'm lying at {}".format(place)) 19 20 def Bark(self): print("Whoop...") 21 22 def Eat(self, food): 23 print("I'm eating {}".format(food)) 24 25 <i># khởi tạo đối tượng</i> 26 bull = Dog("Bull", "large", 2, "Yellow") 27 bull.Stay("garden") 28 bull.Bark()</pre>
Out[4]:	<pre> I'm staying at garden Whoop...</pre>

Phương thức hủy (destructor)

- Là một phương thức đặc biệt của lớp, dùng để hủy một đối tượng của lớp đó. Trong Python, phương thức hủy không thật sự cần thiết, bởi Python có cơ chế tự động quản lý bộ nhớ (tự động hủy đối tượng khi không sử dụng nữa).
- Người dùng vẫn có thể tự khai báo phương thức hủy cho lớp nếu muốn.

Cú pháp

```
def __del__(self):  
    # body of destructor
```


Ví dụ 1.2.1.6. phương thức hủy cho lớp **Dog**

In[5]:	<pre>1 class Dog: 2 # Class attributes 3 DogCount = 0 4 def __init__(self, name, size, age, color): 5 self.name = name # object attributes 6 self.size = size # object attributes 7 self.age = age # object attributes 8 self.color = color # object attributes 9 10 def __del__(self): 11 print("A dog object is being deleted.") 12 13 obj = Dog("bull", "large", 2, "yellow") 14 del obj</pre>
Out[5]:	A dog object is being deleted.

Lưu ý: việc sử dụng "del" để xóa đối tượng không phải lúc nào cũng cần thiết trong Python, vì Python có cơ chế tự động quản lý bộ nhớ thông qua việc thu gom rác (garbage collection). Việc sử dụng "del" thường chỉ cần khi người dùng muốn giải phóng tài nguyên một cách cụ thể, hoặc khi chúng ta cần thao tác với các tài nguyên bên ngoài Python (ví dụ: tạo một tập tin và muốn đóng tập tin sau khi sử dụng).

Các hàm lớp dựng sẵn trong Python

Bảng 1.2. Danh sách hàm dựng sẵn của lớp

Hàm	Mô tả
<code>getattr(obj, name, default)</code>	Được 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>	Nó được sử dụng để xóa một thuộc tính cụ thể.
<code>hasattr(obj, name)</code>	Trả về true nếu đối tượng chứa một số thuộc tính cụ thể.

Ví dụ 1.2.1.7. Sử dụng các hàm(phương thức) dựng sẵn của lớp.



In[6]:	<pre>1 class Dog: 2 # Class attributes 3 DogCount = 0 4 def __init__(self, name, size, age, color): 5 self.name = name # object attributes 6 self.size = size # object attributes 7 self.age = age # object attributes 8 self.color = color # object attributes 9 10 # tạo đối tượng của Lớp Dog 11 bull = Dog("Dom", "large", 2, "yellow") 12 13 # in thuộc tính name của đối tượng bull 14 print(getattr(bull, 'name')) 15 16 # gán giá trị của age cho 3 17 setattr(bull, "age", 3) 18 19 # in giá trị của age 20 print(getattr(bull, 'age')) 21 22 # true nếu bull chứa thuộc tính Large 23 print(hasattr(bull, 'large')) 24 25 # xóa thuộc tính age 26 delattr(bull, 'age') 27 28 # Kích hoạt lỗi nếu age đã bị xóa 29 print(bull.age)</pre>
Out[6]:	<pre>Dom 3 False ----- print(bull.age) AttributeError: 'Dog' object has no attribute 'age'</pre>

Các thuộc tính lớp thích hợp.

Một lớp Python cũng chứa một số thuộc tính lớp tích hợp cung cấp thông tin về lớp

Bảng 1.1. Một số thuộc tính lớp tích hợp

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

Ví dụ: 1.2.1.8.



In[7]:	<pre>1 class Dog: 2 """Ví dụ: Minh họa các hàm thuộc tính lớp tích hợp.""" 3 def __init__(self, name, size, age, color): 4 self.name = name # object attributes 5 self.size = size # object attributes 6 self.age = age # object attributes 7 self.color = color # object attributes 8 9 def display_details(self): 10 print("Name:%s, size:%s, age:%d, color:%s" % 11 (self.name, self.size, self.age, 12 self.color)) 13 14 bull = Dog("Dom", "large", 2, "yellow") 15 print(bull.__doc__) 16 print(bull.__dict__) print(bull.__module__)</pre>
Out[7]:	<pre>Ví dụ: Minh họa các hàm thuộc tính lớp tích hợp. {'name': 'Dom', 'size': 'large', 'age': 2, 'color': 'yellow'} __main__</pre>

Ví dụ 1.2.1.9. Xây dựng lớp Vector2D:

In[8]:

```
1  class Vector2D:
2      x = 0.0
3      y = 0.0
4
5      # Creating a method named Set
6      def Set(self, x, y):
7          self.x = x
8          self.y = y
9
10     def Main():
11         # vec is an object of class Vector2D
12         vec = Vector2D()
13
14         # Passing values to the function Set
15         # by using dot(.) operator.
16         vec.Set(5, 6)
17         print("X: " + str(vec.x) + ", Y: " + str(vec.y))
18
19     if __name__ == '__main__':
20         Main()
```

Kết quả thực hiện chương trình

Out[8]: X: 5, Y: 6

Phương thức của lớp và phương thức tĩnh

□ *Phương thức của lớp*

- Với các phương thức lớp (class method), đối số đầu tiên tự động đưa vào chính lớp gọi phương thức đó hoặc là lớp của đối tượng gọi phương thức đó. Python quy ước tham số nhận đối số này là **cls**.
- Trong Python, *phương thức của lớp* là *phương thức thuộc lớp* chứ không phải thuộc đối tượng.
- Phương thức lớp sử dụng **@classmethod** ngay trên dòng khai báo hàm.

Ví dụ 1.2.1.11. Phương thức của lớp.

In[9]:	<pre> 1 class Dog: 2 # Thuộc tính Lớp 3 DogCount = 0 4 def __init__(self, name, size, age, color): 5 self.name = name # Thuộc tính đối tượng 6 self.size = size # 7 self.age = age # 8 self.color = color # 9 10 @classmethod 11 def CreateDog(cls, name, size, age, color): 12 cls.DogCount = cls.DogCount + 1 13 return cls(name, size, age, color) 14 15 @classmethod 16 def get_dog_count(cls): 17 return cls.DogCount 18 19 obj = Dog.CreateDog("bull", "large", 2, "yellow") 20 print("Số lượng đối tượng Dog đã được tạo:", 21 Dog.get_dog_count()) 22 23 obj2 = Dog.CreateDog("poodle", "small", 1, "white") 24 print("Số lượng đối tượng Dog đã được tạo:", 25 Dog.get_dog_count()) </pre>
Out[9]:	<pre> Số lượng đối tượng Dog đã được tạo: 1 Số lượng đối tượng Dog đã được tạo: 2 </pre>

Phương thức tĩnh:

Phương thức tĩnh trong Python (được đánh dấu bằng *decorator* **@staticmethod**) có mục đích và tác dụng quan trọng thường được sử dụng khi chúng ta cần thực hiện một hành động không liên quan đến các thuộc tính hoặc phương thức của lớp hoặc đối tượng mà thuộc về lớp chung.

Đặc điểm:

- Phương thức tĩnh (static method), được đánh dấu bằng decorator **@staticmethod**.
- Static method không nhận bất kỳ tham số nào được gửi từ đối tượng hoặc lớp khi gọi phương thức.
- Static method hoạt động như một hàm bình thường, không thể truy cập hoặc thay đổi *các thuộc tính không tĩnh* của lớp.

Ví dụ 1.2.1.12. Phương thức tĩnh.

In[10]:	<pre>1 class Dog: 2 DogCount = 0 # Thuộc tính Lớp 3 def __init__(self, name, size, age, color): 4 self.name = name # Thuộc tính đối tượng 5 self.size = size # 6 self.color = color # 7 Dog.DogCount += 1 # 8 9 @staticmethod 10 def Report(): 11 print("Tổng số đối tượng Dog:{}".format(Dog.DogCount)) 12 13 # Tạo các đối tượng Dog 14 dog1 = Dog("Buddy", "Medium", 5, "Brown") 15 dog2 = Dog("Max", "Small", 3, "Black") 16 17 # Sử dụng phương thức tĩnh Report() để lấy số lượng đối tượng đã được tạo 18 dog1.Report()</pre>
Out[10]:	Tổng số đối tượng Dog: 2

- ❑ Trong Python, phương thức của lớp và phương thức tĩnh là phương thức thuộc lớp chứ không phải thuộc đối tượng.

Sự khác biệt giữa phương thức của lớp và phương thức tĩnh.

Phương thức của lớp (@classmethod)	Phương thức tĩnh (@staticmethod)
Có tham số đầu tiên là cls	Không có tham số cls
Có thể truy xuất và chỉnh sửa <i>trạng thái của lớp</i> .	Không thể chỉnh sửa hay truy xuất <i>trạng thái của lớp</i> .
Dùng để tạo ra các đối tượng của lớp (tương tự như phương thức khởi tạo)	Dùng để xử lý các công việc mang tính chất tổng quát.

c. Thuộc tính tĩnh của lớp

Trong Python, thuộc tính tĩnh của lớp là *các thuộc tính và phương thức* thuộc về lớp chứ không phải là các đối tượng cụ thể của lớp đó.

Các thuộc tính tĩnh được sử dụng để lưu trữ và chia sẻ dùng chung giữa tất cả các đối tượng của lớp và có thể truy cập mà không cần tạo một thể hiện của lớp.

Ý nghĩa



- **Chia sẻ thông tin chung:** Các thuộc tính tĩnh được chia sẻ giữa tất cả các đối tượng của lớp. Nó cho phép chia sẻ dữ liệu và phương thức chung mà không cần tạo nhiều bản sao cho mỗi đối tượng.
- **Biến toàn cục trong lớp:** Thuộc tính tĩnh có thể được sử dụng như biến toàn cục trong phạm vi của lớp. Điều này có nghĩa là chúng có thể được truy cập từ bất kỳ nơi nào trong lớp mà không cần tạo thể hiện (đối tượng) của lớp đó.
- **Dữ liệu không liên quan đến đối tượng cụ thể:** Thuộc tính tĩnh thường được sử dụng để lưu trữ các giá trị hoặc chức năng liên quan đến lớp chứ không liên quan đến các đối tượng cụ thể của lớp. Ví dụ, số lượng đối tượng của một lớp có thể được lưu trữ trong một thuộc tính tĩnh.
- **Tiết kiệm bộ nhớ:** Thuộc tính tĩnh không bị sao chép cho mỗi đối tượng được tạo. Nó được chia sẻ giữa tất cả các đối tượng của lớp, vì vậy giúp tiết kiệm bộ nhớ.
- **Tính đóng gói và tổ chức:** Sử dụng các thuộc tính tĩnh giúp tổ chức mã nguồn và tăng tính đóng gói (encapsulation). Các thuộc tính tĩnh thường được đặt tên một cách có ý nghĩa và phân loại chúng vào một nhóm nhất định.
- **Thực hiện các hàm tiện ích:** Thuộc tính tĩnh thường được sử dụng để triển khai các hàm tiện ích hoặc các phương thức hỗ trợ mà không cần sự hiện diện của đối tượng.

Ví dụ 1.2.1.13. Minh họa sử dụng thuộc tính tĩnh

In[11]:	<pre> 1 class Dog: 2 #Thuộc tính của lớp - thuộc tính tĩnh DogCount 3 DogCount = 0 4 def __init__ (self, name, size, age, color): 5 self.name = name # object attributes 6 self.size = size # object attributes 7 self.age = age # object attributes 8 self.color = color # object attributes 9 Dog.DogCount = Dog.DogCount + 1 10 11 def __del__ (self): 12 print("A dog object is being deleted.") 13 Dog.DogCount=Dog.DogCount - 1 14 15 obj1 = Dog("bull", "large", 2, "yellow") 16 obj2 = Dog ("Poodle", "small", 1, "white") 17 print ("Number of dogs: {}".format(Dog.DogCount)) </pre>
Out[11]:	Number of dogs: 2

DogCount là thuộc tính tĩnh

1.2.2. Đối tượng

- Đối tượng (Object) là những thực thể tồn tại có **thuộc tính** và **hành vi**.
- Trong Python, đối tượng là *một thể hiện* của lớp.
- Khi lớp được định nghĩa nó sẽ là mô tả cho một đối tượng được xác định.

Cú pháp

Tên_đối_tượng = **Tên_lớp**(<các tham số cần thiết>)

Ví dụ

```
obj1 = Dog ("bull", "large", 2, "yellow")  
obj2 = Dog ("Poodle", "small", 1, "white")
```

Ví dụ 1.2.2.1

Ví dụ 1.2.2.1

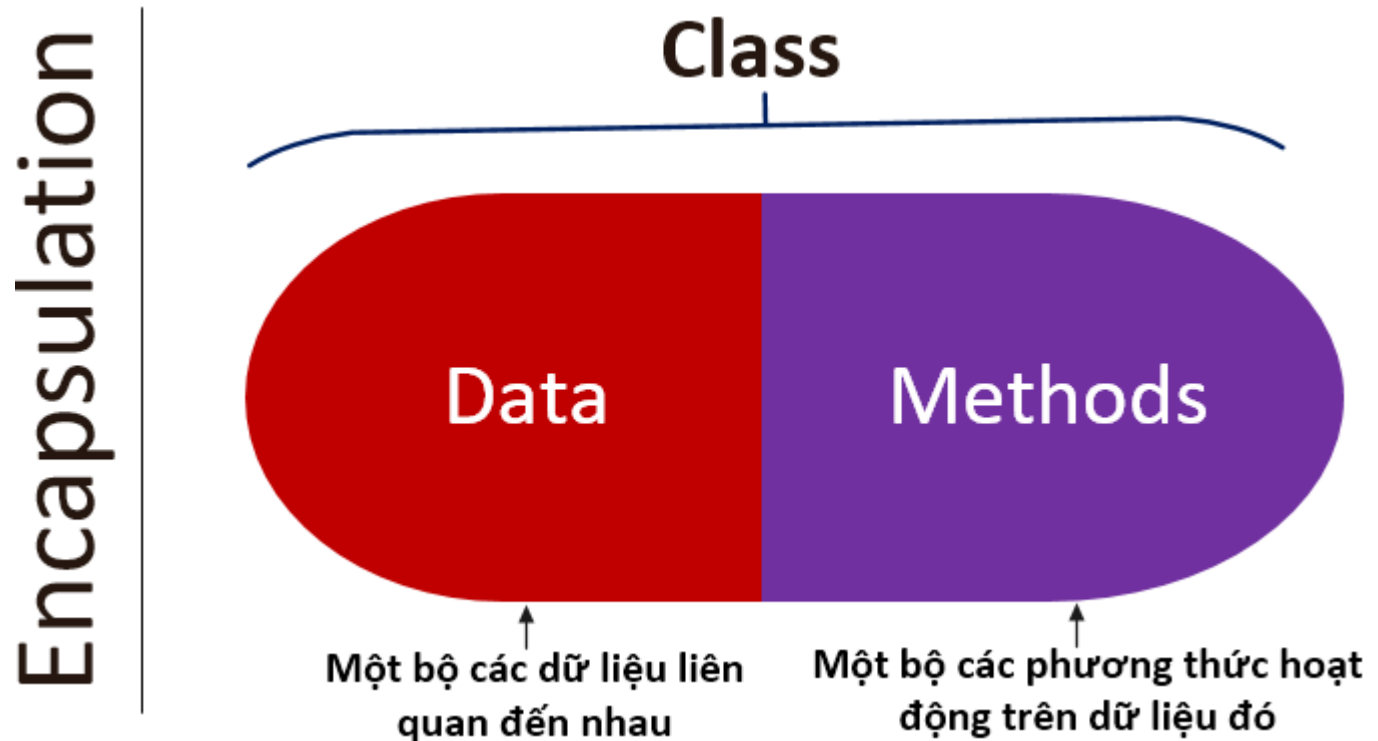
In[12]:	<pre>1 class Dog: 2 # Thuộc tính lớp 3 DogCount = 0 4 species = 'Dog' 5 6 # instance attribute 7 def __init__(self, name, size, age, color): 8 self.name = name # Thuộc tính đối tượng 9 self.size = size 10 self.age = age 11 self.color = color 12 13 #instance của lớp Dog 14 obj1 = Dog("Buddy", "Medium", 5, "Brown") 15 obj2 = Dog("Max", "Small", 3, "Black") 16 17 #Truy cập thuộc tính lớp 18 print("obj1 is a {}".format(obj1.__class__.species)) 19 print("obj2 is also a 20 {}".format(obj2.__class__.species)) 21 22 #Truy cập thuộc tính của instance 23 print("{} is {} years old".format(obj1.name, obj1.age)) 24 print("{} is {} years old".format(obj2.name, obj2.age))</pre>
Out[12]:	<pre>obj1 is a Dog obj2 is also a Dog Buddy is 5 years old Max is 3 years old</pre>

1.3. CÁC TÍNH CHẤT CỦA HƯỚNG ĐỐI TƯỢNG

1.3.1. Tính đóng gói (Encapsulation)

- Che dấu thông tin của đối tượng với môi trường bên ngoài.
- Đảm bảo tính toàn vẹn và bảo mật của đối tượng.
- Hạn chế quyền truy cập vào trạng thái bên trong của đối tượng.

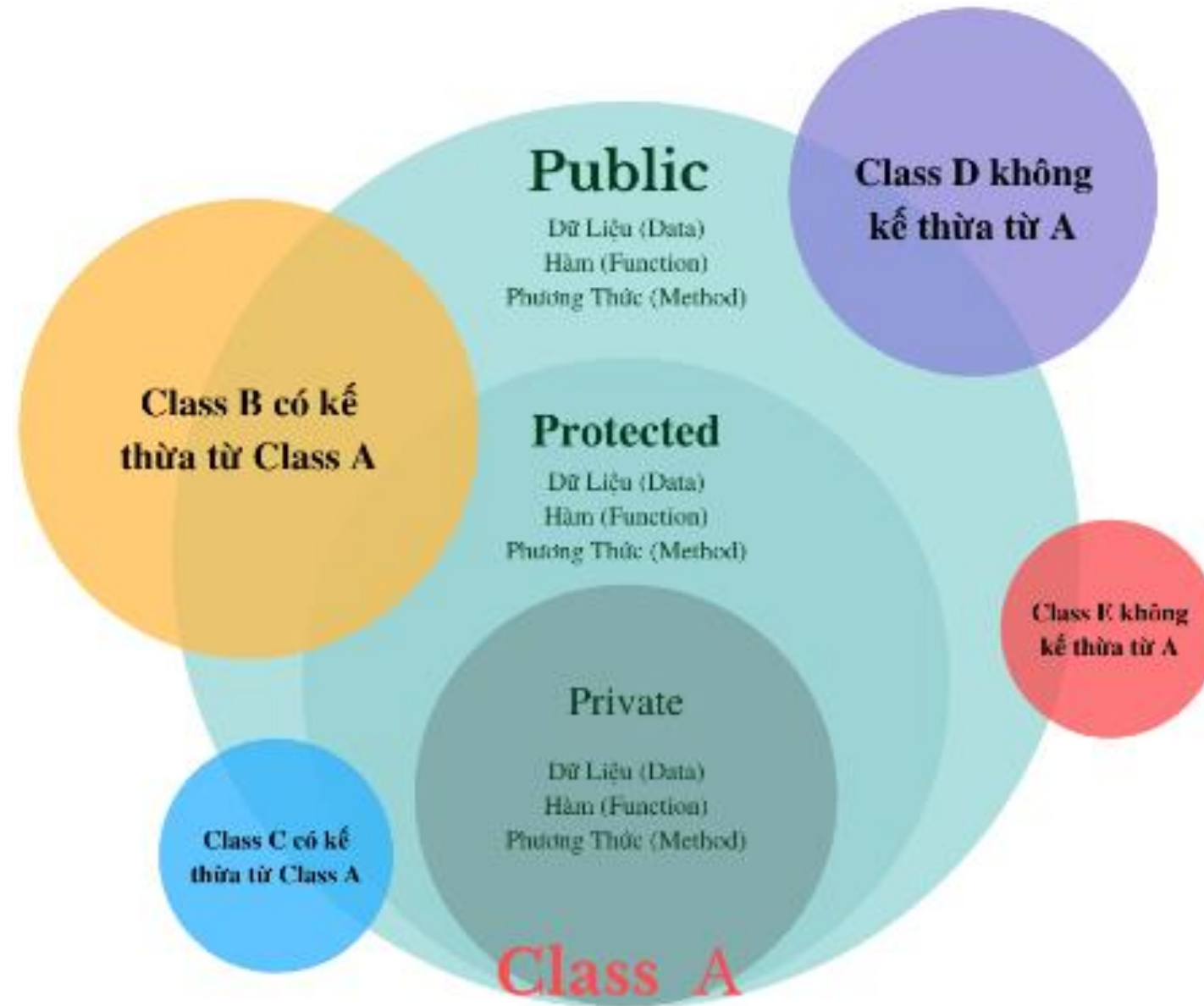
(Ngăn chặn dữ liệu bị sửa đổi trực tiếp).



Phạm vi sử dụng:

Thành viên của một lớp có thể là ở dạng : **private**, **protected** hoặc **public**.

- ❑ **Private**: không thể truy cập được khi đứng ngoài lớp.
 - ❖ Trước tên thành viên private có tiền tố là **2 dấu gạch** dưới '___'.
- ❑ **Protected**: chỉ truy cập được các thành viên này khi đứng ở trong lớp đó hoặc lớp con của nó.
 - ❖ Trước tên thành viên protected có tiền tố là **một dấu gạch dưới** '_'.
- ❑ **Public**: mặc định (khai báo như tên thông thường) thì thành viên ở chế độ public, tức là có thể truy xuất vào các thành viên khi đứng ở bất kì đâu (trong lớp, ngoài lớp...)



Tính đóng gói trong lập trình hướng đối tượng.[*]

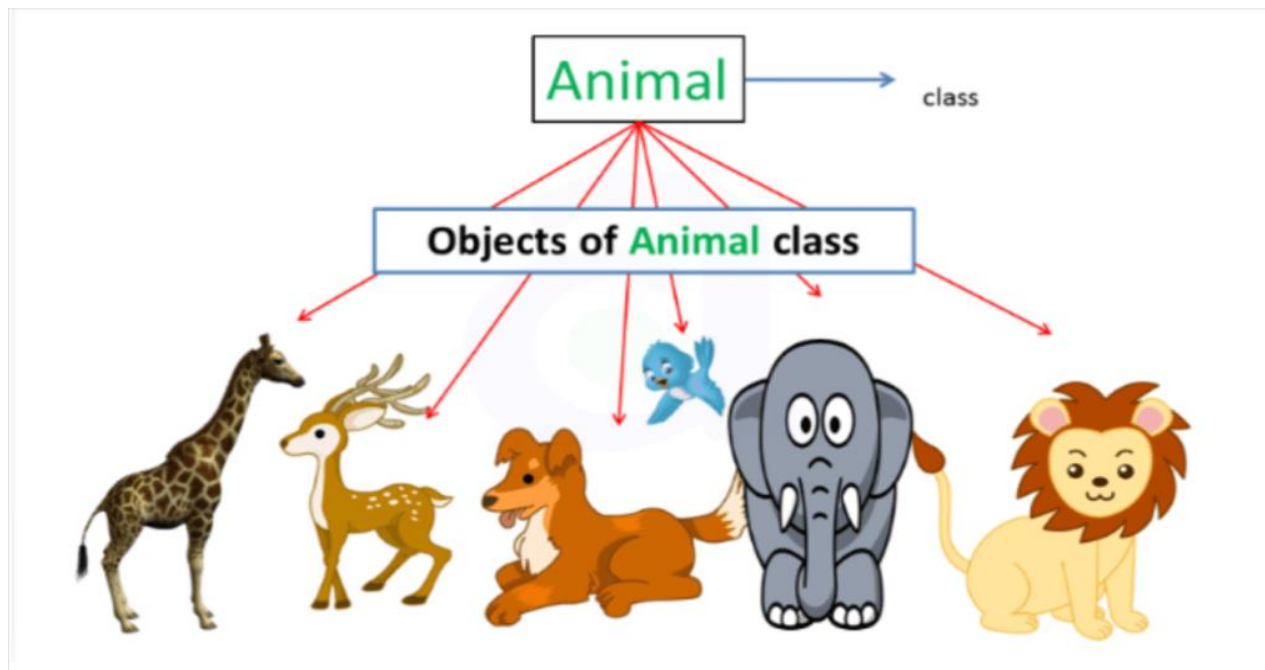
*:<https://funix.edu.vn/chia-se-kien-thuc/4-tinh-chat-trong-lap-trinh-huong-doi-tuong/>

Ví dụ 1.3.1.1. Đóng gói dữ liệu trong Python.

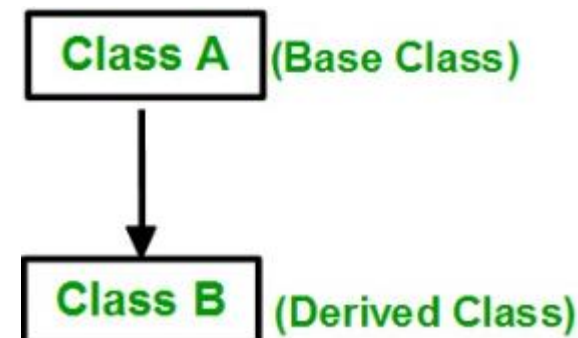
In[13]:	<pre>1 class Dog: 2 # Class attributes 3 __DogCount__ = 0 4 def __init__(self, name, size, age, color): 5 self._name = name # protected attributes 6 self._size = size # protected attributes 7 self.__age = age # private attributes 8 self.__color = color # private attributes 9 Dog.DogCount = Dog.DogCount + 1 10 11 obj = Dog("bull", "large", 2, "yellow") 12 print("Number of dogs: {}".format(Dog.DogCount))</pre>
Out[13]:	<pre>----- AttributeError Traceback (most recent call last) <ipython-input-3-92eff3f7b0aa> in <module> 9 Dog.DogCount = Dog.DogCount + 1 10 ----> 11 obj = Dog("bull", "large", 2, "yellow") 12 print("Number of dogs: {}".format(Dog.DogCount)) <ipython-input-3-92eff3f7b0aa> in __init__(self, name, size, age, color) 7 self.age = age # private attributes 8 self.color = color # private attributes ----> 9 Dog.DogCount = Dog.DogCount + 1 10 11 obj = Dog("bull", "large", 2, "yellow") AttributeError: type object 'Dog' has no attribute 'DogCount'</pre>

1.3.2. Tính kế thừa (Inheritance)

- ❑ Tính kế thừa là khả năng tái sử dụng lại một số thuộc tính, phương thức của lớp đã có sẵn.



Trong kế thừa, một lớp cơ sở (**base class**) thường được gọi là lớp cha (**super class**) sẽ được kế thừa lại từ một lớp khác (lớp dẫn xuất – **Derived Class**) hay thường gọi là lớp con (**subclass**). Lớp con này sẽ bổ sung thêm một số thuộc tính cho lớp cha.



Cú pháp

(lớp cơ sở/lớp cha): khai báo như cách khai báo một lớp thông thường.

```
class BaseClassName:
```

```
    # khai báo lớp
```

```
class ChildClassName(BaseClassName):
```

```
    def __init__(self, <các tham số khác nếu có>):
```

```
        BaseClassName.__init__(<tham số>) # gọi hàm khởi tạo của lớp cha
```

```
    # khai báo lớp
```

Tên lớp cơ sở (base)/lớp cha để trong cặp ngoặc ' () '.

Lưu ý:

- ❑ BaseClass (lớp cơ sở/lớp cha): khai báo như cách khai báo một lớp thông thường.
- ❑ ChildClass (lớp dẫn xuất/lớp con): để khai báo thừa kế từ lớp cơ sở. Thừa kế từ lớp nào thì tên lớp đó để trong dấu ngoặc tròn.
- ❑ Phương thức khởi tạo của lớp con sẽ gọi lại phương thức khởi tạo của lớp cha.

Mục đích của thừa kế:

Tái sử dụng code. Khi tạo một lớp mới, không cần thiết phải viết lại toàn bộ thuộc tính hoặc phương thức mà sử dụng thuộc tính hoặc phương thức của lớp đã có sẵn.

Ví dụ 1.3.2.1. Đơn kế thừa



In[]:	<pre>1 class Animal: 2 def __init__(self, name): 3 self._name = name 4 5 def Display(self): 6 print("I'm {}".format(self._name)) 7 8 class Dog(Animal): 9 def __init__(self, name, size, age, color): 10 super().__init__(name) 11 self.size = size 12 self.age = age 13 self.color = color 14 15 def Go(self, place): 16 print("I'm going to {}".format(place)) 17 18 obj1 = Dog("bull", "large", 2, "yellow") 19 obj1.Display() 20 obj1.Go("garden")</pre>
Out[]:	<pre>I'm bull I'm going to garden</pre>

b. Các loại thừa kế

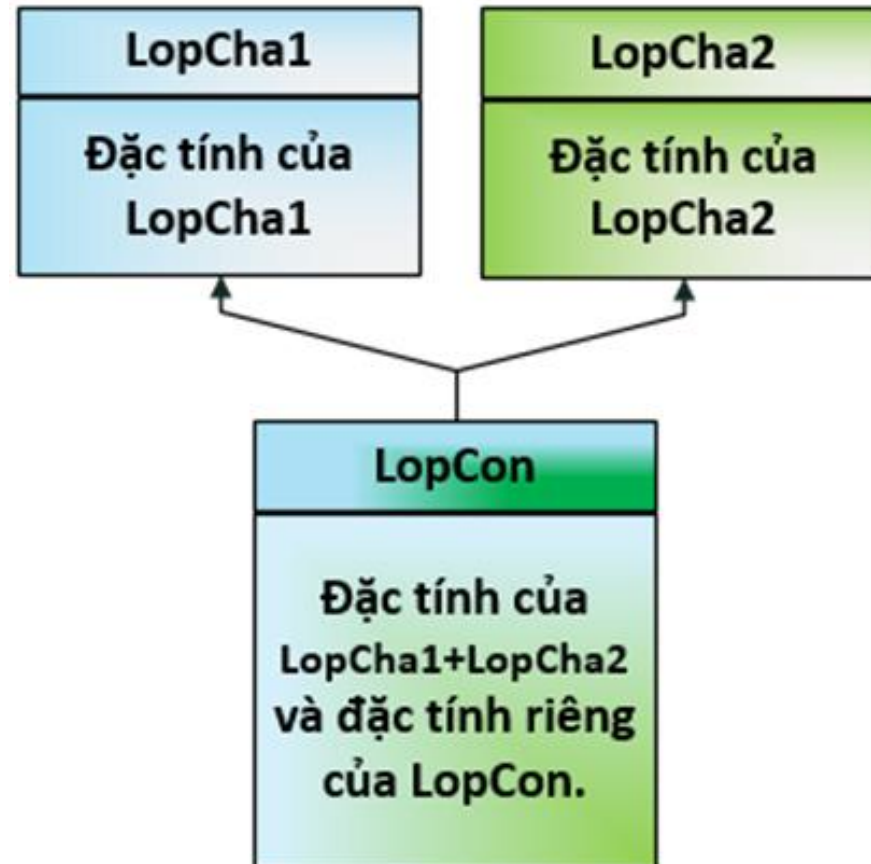
- *Đơn kế thừa (Single Inheritance)*: Đơn kế thừa là khi một lớp con chỉ kế thừa từ một lớp cha duy nhất.
- *Đa kế thừa (Multiple Inheritance)*
Đa kế thừa là khi một lớp con kế thừa từ nhiều lớp cha. Trong Python, người dùng có thể thực hiện đa kế thừa bằng cách liệt kê các lớp cha cần kế thừa trong cặp dấu ngoặc đơn `class SubClass(ParentClass1, ParentClass2, ...)`

Cú pháp

Lớp con thừa kế từ nhiều lớp cha, có thể chỉ định nhiều lớp cha, phân cách nhau bởi dấu phẩy, và tất cả được bao trong một cặp dấu ngoặc tròn () để thực hiện đa kế thừa.

```
class LopCha1:  
    # khai báo lớp cha 1  
    ...  
class LopChaN:  
    # khai báo lớp cha N  
class LopCon(LopCha1, LopCha2, ..., LopChaN):  
    # khai báo lớp con
```

Lớp con thừa kế từ nhiều lớp cha, có thể chỉ định nhiều lớp cha, phân cách nhau bởi dấu phẩy, và tất cả được bao trong một cặp dấu ngoặc tròn '()' để thực hiện đa kế thừa.



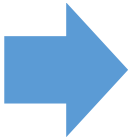
Hình 1.3. Minh họa đa kế thừa.

Ví dụ 1.3.2.2. Đa kế thừa.



```
In[ ]: 1 class Zebra:                                #Lớp cha1
        2     def __init__(self):
        3         print ("Zebra")
        4
        5     def Display(self):
        6         print("I'm a zebra")
        7
        8 class Donkey:                            #Lớp cha2
        9     def __init__(self):
       10         print("Donkey")
       11
       12     def Display(self):
       13         print("I'm a donkey")
       14
       15 class Zonkey(Zebra, Donkey):              Lớp con Zonkey kế thừa lớp cha1, cha2
       16     def __init__(self):
       17         Zebra.__init__(self)
       18         Donkey.__init__(self)
       19         print("Zonkey")
       20
       21 obj = Zonkey()
       22 obj.Display() # có vấn đề phát sinh khi chạy phương thức này!
```

Out[]:



```
Zebra
Donkey
Zonkey
I'm a zebra
```

Trong ví dụ 1.3.2.2 trên, lớp **Zonkey** thừa kế từ 2 lớp cha là **Zebra** và **Donkey**.

Problems?

Khi các lớp cha có phương thức cùng tên (trong ví dụ trên **Zebra** và **Donkey** đều có phương thức **Display**) thì trình thông dịch không rõ người dùng định gọi phương thức nào!????

Như trong ví dụ 1.3.2.2 trên, Python sẽ gọi phương thức **Display** của lớp **Zebra**, lớp xuất hiện trước trong danh sách thừa kế.

Nhận xét:

Trong Python, các lớp cha có thể có các *thuộc tính* hoặc các *phương thức* giống nhau. Lớp con sẽ ưu tiên thừa kế thuộc tính, phương thức của lớp *đứng đầu tiên* trong danh sách thừa kế.

c. Thừa kế đa cấp

Là hiện tượng, một lớp (cháu) thừa kế từ một lớp con. Trong trường hợp này, các thành viên của *lớp cha* và *lớp con* sẽ được *lớp cháu* thừa kế.

Cú pháp

```
class LopCha:  
    #khởi báo Lớp cha  
  
class LopCon(LopCha):  
    #khởi báo Lớp con  
  
class LopChau(LopCon1):  
    #khởi báo Lớp cháu
```

Thứ tự truy xuất:

[LopCon, LopCha1, LopCha2, object]

Ví dụ 1.3.2.3.



In[]

```
1 class LopCha:
2     def __init__(self, ten_cha):
3         self.ten_cha = ten_cha
4
5     def thong_tin(self):
6         print("Tôi là Lớp Cha, tên của tôi là", self.ten_cha)
7
8 class LopCon(LopCha):
9     def __init__(self, ten_cha, ten_con):
10         # Gọi hàm khởi tạo của lớp cha để khởi tạo các thuộc tính từ lớp cha
11         super().__init__(ten_cha)
12         self.ten_con = ten_con
13
14     def thong_tin(self):
15         print("Tôi là Lớp Con, tên của tôi là", self.ten_con)
16         # Gọi phương thức của lớp cha bằng cách sử dụng super()
17         super().thong_tin()
18
19 class LopChau(LopCon):
20     def __init__(self, ten_cha, ten_con, ten_chau):
21         # Gọi hàm khởi tạo của LopCha và LopCon để khởi tạo các thuộc tính từ
22         # LopCha và LopCon
23         super().__init__(ten_cha, ten_con)
24         self.ten_chau = ten_chau
25
26     def thong_tin(self):
27         print("Tôi là Lớp Cháu, tên của tôi là", self.ten_chau)
28         super().thong_tin()
29
30 # Sử dụng các lớp
31 lop_chau = LopChau("Ông A", "Bác B", "Cháu C")
32 lop_chau.thong_tin()
```

Out[]:

Tôi là Lớp Cháu, tên của tôi là Cháu C
Tôi là Lớp Con, tên của tôi là Bác B
Tôi là Lớp Cha, tên của tôi là Ông A

1.3.3. Tính đa hình (Polymorphism)

- ❑ Trong OOP, đa hình cho phép tạo ra phương thức trùng tên với phương thức ở lớp cha. Kỹ thuật này gọi là *Method overriding* (nạp chồng phương thức).
- ❑ Đa hình cho phép định nghĩa ra các phương thức trong lớp con mà có cùng tên với các phương thức trong lớp cha.
- ❑ Khi kế thừa, lớp con sẽ thừa kế các phương thức từ lớp cha.

Tuy nhiên, hoàn toàn có thể chỉnh sửa các phương thức thuộc lớp con, mà được kế thừa lại từ lớp cha.

Method Overriding.

Kỹ thuật cài đặt lại một phương thức được kế thừa từ lớp cha, tại lớp con, được gọi là ghi đè phương thức – **Method Overriding**.

Method Overriding đặc biệt hữu dụng trong những trường hợp mà phương thức được kế thừa lại từ lớp cha chưa thực sự phù hợp với lớp con.



Minh họa tính đa hình.[*]

Ví dụ 1.3.3

In[]:	<pre>1 #Khai báo lớp chim 2 class Bird: 3 def flight(self): #Định nghĩa phương thức flight() 4 print("Many birds can fly") 5 6 #Khai báo lớp chim sẽ kế thừa từ lớp Bird 7 class Sparrow(Bird): 8 def flight(self):#Ghi đè phương thức flight() 9 print("Sparrows can fly") 10 11 #Khai báo lớp Đà điểu kế thừa từ lớp Bird 12 class Ostrich(Bird): 13 def flight(self): 14 print ("Ostriches cannot fly") 15 16 obj1 = Bird() 17 obj2 = Sparrow() 18 obj3 = Ostrich() 19 obj1.flight() 20 obj2.flight() 21 obj3.flight()</pre>
Out[]:	<pre>Many birds can fly Sparrows can fly Ostriches cannot fly</pre>

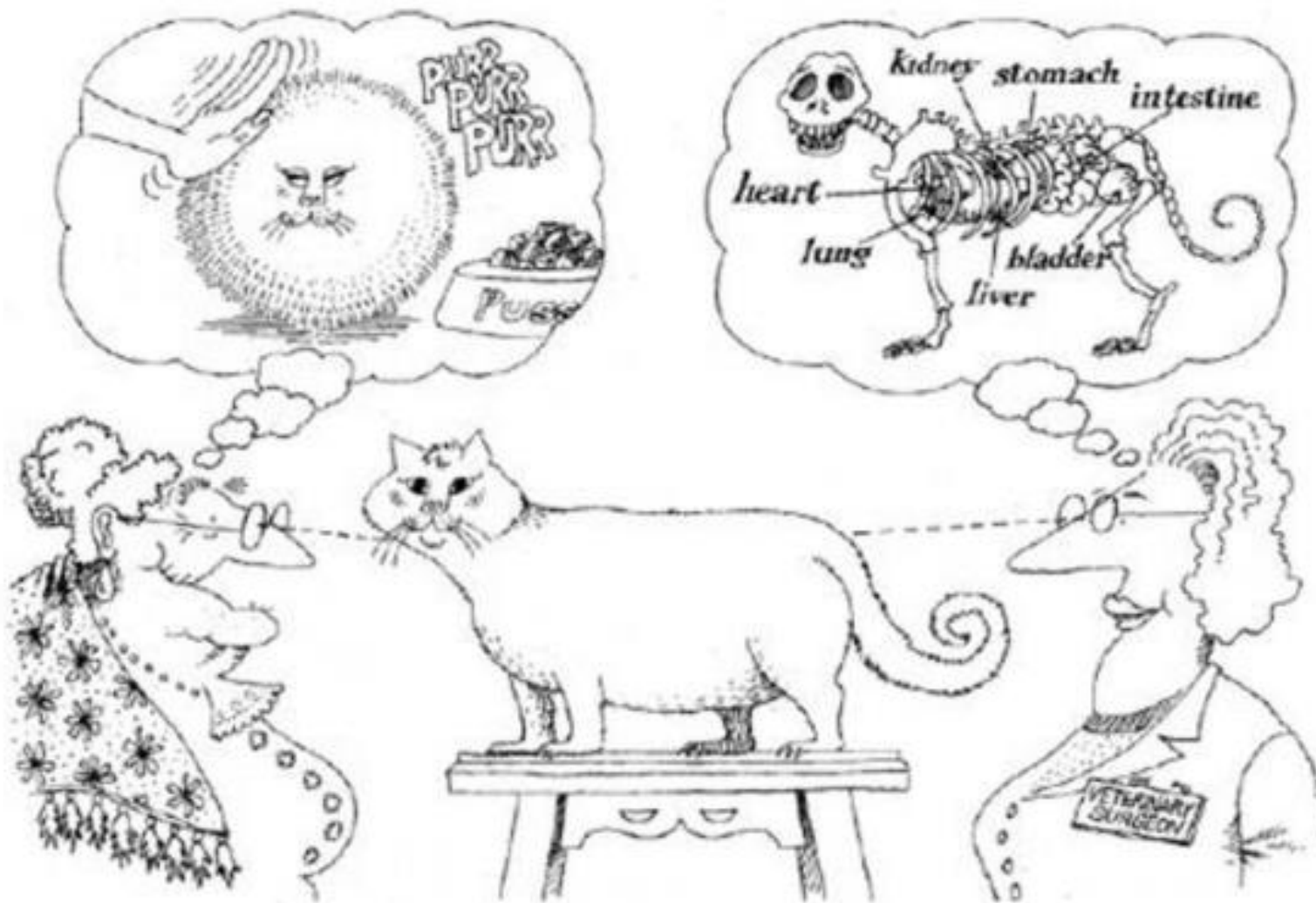
Phương thức **flight** được ghi đè (Overriding), thay đổi nội dung dòng lệnh print để in ra: "Ostriches cannot fly"

1.3.4. Tính trừu tượng (Abstract)

- ❑ Tính trừu tượng cho phép người dùng tạo các lớp trừu tượng, định nghĩa hành vi chung mà các lớp con có thể thừa hưởng và triển khai một cách cụ thể
- ❑ Lớp trừu tượng (abstract class) trong OOP được coi như là bản thiết kế hay bộ khung để các class khác tuân theo.
- ❑ Trừu tượng được sử dụng để ẩn chi tiết nội bộ và chỉ hiển thị các chức năng
- ❑ Một lớp có một hoặc nhiều phương thức trừu tượng gọi là lớp trừu tượng.
- ❑ Phương thức trừu tượng là phương thức *chỉ có khai báo* (chỉ đưa ra tên và tham số) mà *không có định nghĩa* (không có nội dung).

Lưu ý: Mặc định, Python không cho phép khai báo lớp trừu tượng, nó chỉ có một module chứa lớp cơ sở **Abstract Base Class** (ABC). Để tạo một lớp trừu tượng trong Python người lập trình cần kế thừa từ lớp ABC của module **ABC** và sử dụng từ khóa **@abstractmethod** trước các phương thức mà chúng ta muốn là trừu tượng

1.3.4. Tính trừu tượng (Abstract), ...



Ví dụ 1.3.4.1.



```
1  from abc import ABC, abstractmethod
2
3  # Tạo lớp trừu tượng 'Animal' với phương thức trừu tượng 'Say'
4  class Animal(ABC):
5      @abstractmethod
6      def Say(self):
7          pass
8
9  # Tạo lớp con 'Dog' kế thừa từ 'Animal'
10 class Dog(Animal):
11     def Say(self):
12         print("Whooooopp whooppp...")
13
14 # Tạo lớp con 'Cat' kế thừa từ 'Animal'
15 class Cat(Animal):
16     def Say(self):
17         print("Meooo meooo")
18
19     # Định nghĩa một phương thức 'Jump' cho Lớp 'Cat'
20     def Jump(self):
21         print("I'm jumping up high")
```

Ví dụ 1.3.4.1. (tiếp, ...)

```
23  # Khởi tạo đối tượng từ Lớp 'Dog'
24  obj_dog = Dog()
25  # Gọi phương thức 'Say' của đối tượng 'obj_dog'
26  obj_dog.Say()
27
28  # Khởi tạo đối tượng từ Lớp 'Cat'
29  obj_cat = Cat()
30  # Gọi phương thức 'Say' của đối tượng 'obj_cat'
32  obj_cat.Say()
```

Out[]:

```
Whooooopp whooppp...
Meooo meooo
```

CÂU HỎI TRẮC NGHIỆM



Câu 1. Điền vào chỗ trống.

Trong Python, một lớp làcho một đối tượng cụ thể.

a. Một bản thiết kế chi tiết

c. Một phương thức

b. Một ví dụ

d. Một thể hiện.

Câu 2: Trong lớp Dog như sau:

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

Cách chính xác để khởi tạo lớp Dog ở trên là:

- a. Dog.__init__("Rufus", 3)
- b. Dog.create("Rufus", 3)
- c. Dog()
- d. Dog("Rufus", 3)

Câu 3. Điền vào chỗ trống.

Trong Python, chúng ta có thể truy cập các thuộc tính đối tượng và lớp bằng ký hiệu _____.

a. Dấu chấm ‘.’

b. Lát cắt (Slice) ‘::’

c. Ký hiệu khoa học (scientific)

d. đối tượng (object).

Câu 4: Trong Python, một hàm trong định nghĩa lớp được gọi là:

- a. Một hàm nhà máy (a factory function)
- b. Một hàm lớp (a class function)
- c. Một phương thức (a method)
- d. Có thể gọi được (a callable)

Câu 5. Điền vào chỗ trống.

_____ đại diện cho một thực thể trong thế giới thực với danh tính và hành vi của nó

a) Một phương thức (a method)

b) Một đối tượng (an object)

c) Một lớp (a class)

d) Một toán tử (an operator)

Câu 6. Điền vào chỗ trống.

_____ được sử dụng để tạo ra một đối tượng.

a) Lớp (class)

c) Hàm do người dùng định nghĩa
(User-defined functions)

b) Hàm tạo (constructor)

d) Hàm có sẵn (in-built functions)

Câu 7: Cho biết kết quả của đoạn mã Python sau:

```
class test:
    def __init__(self, a="Hello World"):
        self.a=a
    def display(self):
        print(self.a)
obj=test()
obj.display()
```

- a. Chương trình bị lỗi do hàm khởi tạo không có đối số mặc định.
- b. Không có gì được hiển thị.
- c. Hiển thị "Hello World"
- d. Chương trình báo lỗi: "an error display function doesn't have parameters"

Câu 8: Cho biết kết quả của đoạn mã Python sau:

```
class test:
    def __init__(self,a):
        self.a=a

    def display(self):
        print(self.a)

obj=test()
obj.display()
```

- a. Chạy bình thường, không hiển thị gì.
- b. Hiển thị 0, là giá trị mặc định tự động.
- c. Lỗi vì cần có một đối số khi tạo đối tượng
- d. Lỗi vì hàm display cần bổ sung đối số.

Câu 9: Cho biết mã Python sau đúng hay sai:

```
>>> class A:
    def __init__(self,b):
        self.b=b
    def display(self):
        print(self.b)

>>> obj=A("Hello")
>>> del obj
```

a) True

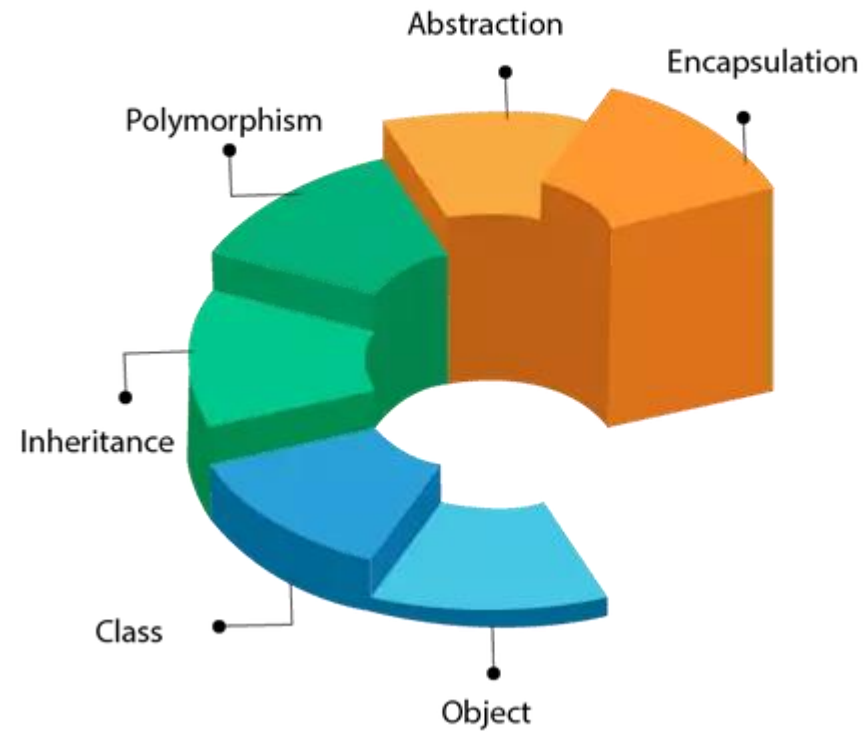
b) False

Câu 10: Cho biết kết quả của đoạn mã Python sau:

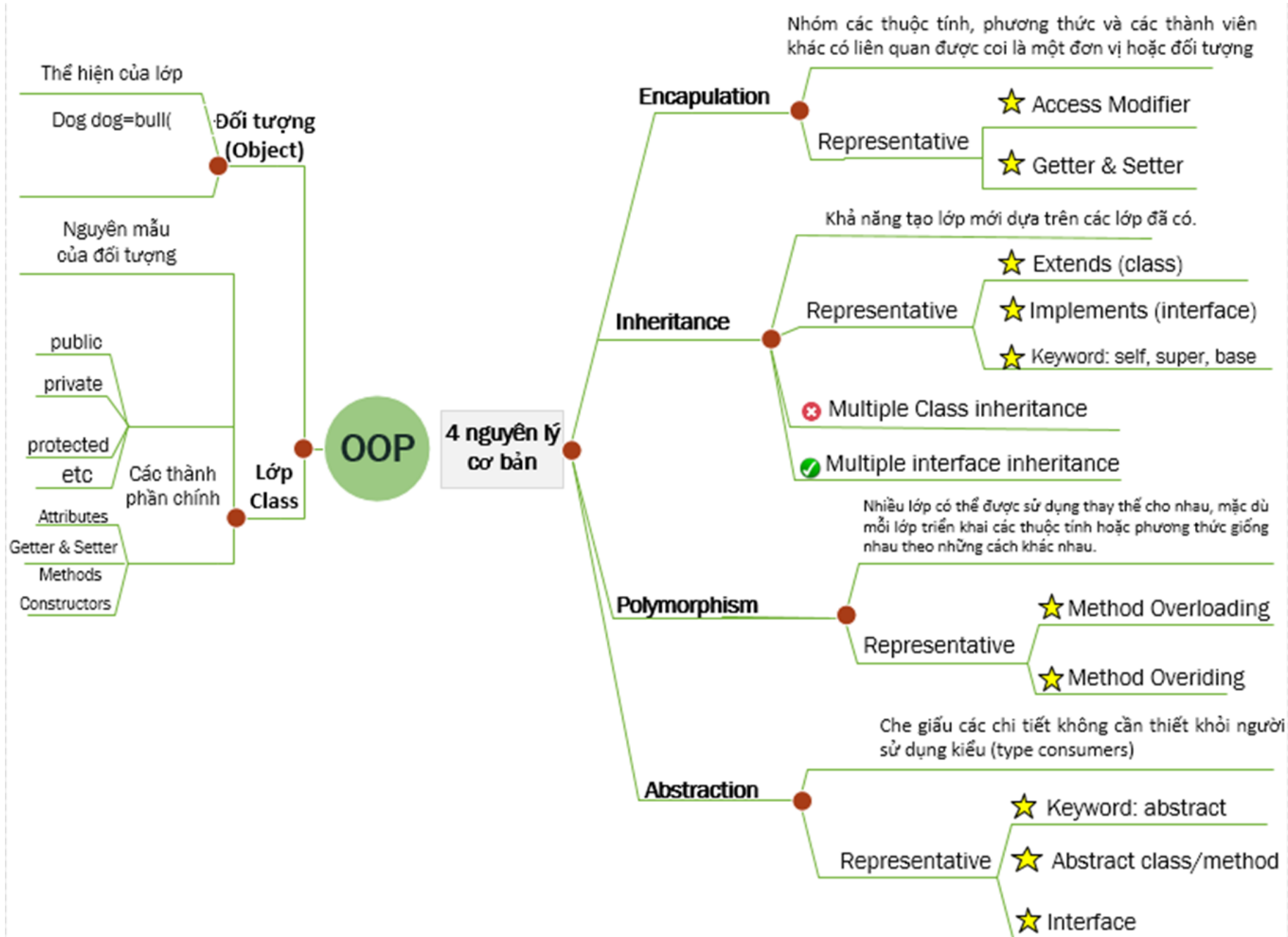
```
class test:
    def __init__(self):
        self.variable = 'Old'
        self.Change(self.variable)
    def display(self, var):
        var = 'New'
obj=test()
print(obj.variable)
```

- a. Chương trình bị lỗi do hàm khởi tạo không có đối số mặc định.
- b. Không có gì được hiển thị.
- c. Hiện thị "Hello World"
- d. Chương trình báo lỗi: "an error display function doesn't have parameters"

OOPs (Object-Oriented Programming System)



TỔNG KẾT



CÂU HỎI

Câu 1: Hãy so sánh lập trình hướng đối tượng (OOP) với lập trình cấu trúc (POP)?

Câu 2: Tính chất đặc thù của lập trình hướng đối tượng?

Câu 3: Tính đóng gói trong lập trình hướng đối tượng có đặc trưng gì?

Câu 4: Hãy nêu các loại kế thừa trong lập trình hướng đối tượng?

Câu 5: Đa hình là gì?

Câu 6: Khái niệm lớp trừu tượng?

BÀI TẬP



Làm các bài tập số 1 đến 11 trang 26, 27, trong TLHT hoặc trên LMS.