

OOP



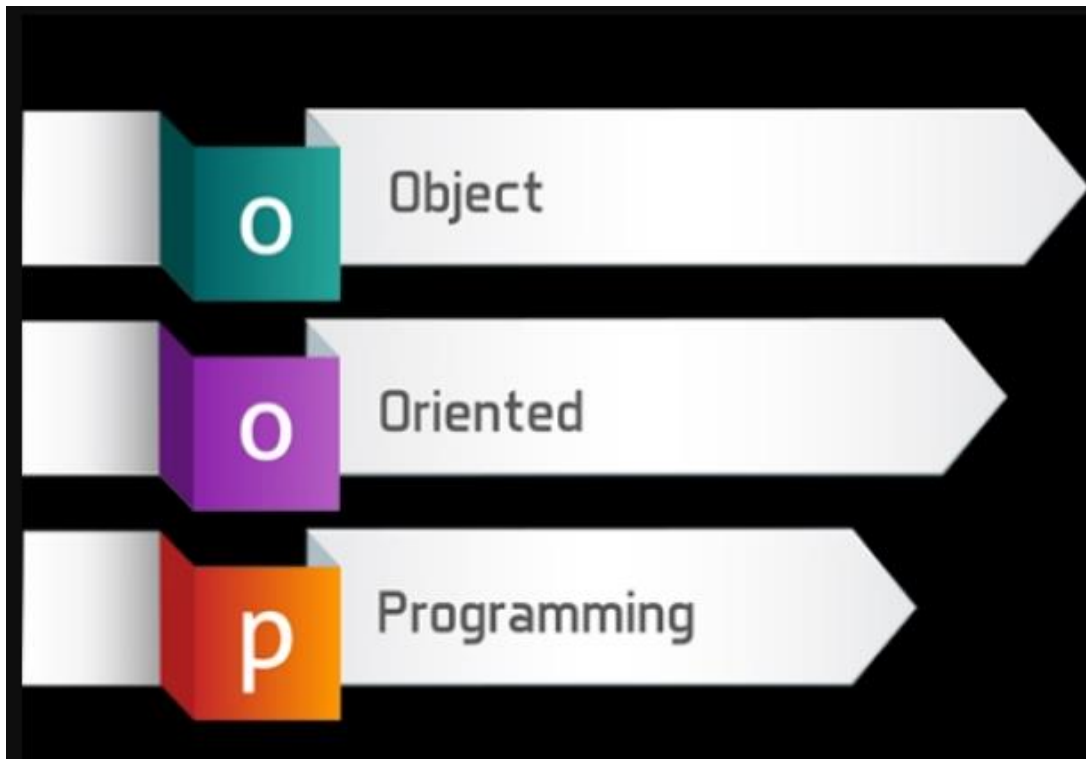
fit@hcmus

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO THUYẾT TRÌNH	
Giảng viên:	Hồ Tuấn Thanh
Lớp:	CQ2019/4
Người thực hiện:	Đặng Thái Duy
MSSV:	19120491

Đề tài: So sánh OOP trong C++ và Python

OOP



OOP

- ❑ Lập trình hướng đối tượng (Object Oriented Programming - OOP) là một kỹ thuật lập trình cho phép tạo ra các đối tượng để trừu tượng hóa 1 đối tượng thực tế (đưa các đối tượng trong thực tế vào trong code). Cho phép lập trình viên tương tác với các đối tượng.
- ❑ Hiện nay có khá nhiều ngôn ngữ lập trình theo hướng đối tượng như C++, Java, PHP, Python, ...



OOP

4 đặc tính cơ bản

E

ENCAPSULATION

Tính đóng gói

ABSTRACTION

Tính trừu tượng

A

INHERITANCE

Tính kế thừa

I

POLYMORPHISM

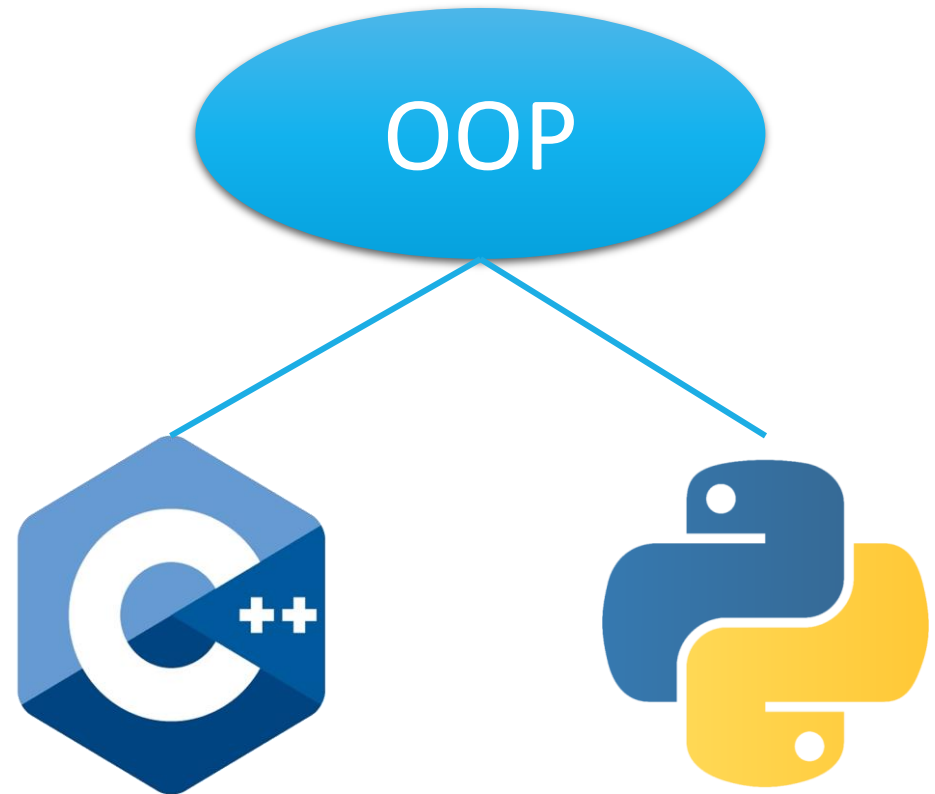
Tính đa hình

P

OOP



Bài thuyết trình này sẽ tập trung so sánh OOP của hai ngôn ngữ khá phổ biến trên thế giới, là C++ và Python.





Nội dung báo cáo

- 01 Class, Object, Attribute, Method, Encapsulation, Access Modifier
- 02 Constructor, Destructor
- 03 Static Attribute, Static Method
- 04 Inheritance
- 05 Polymorphism

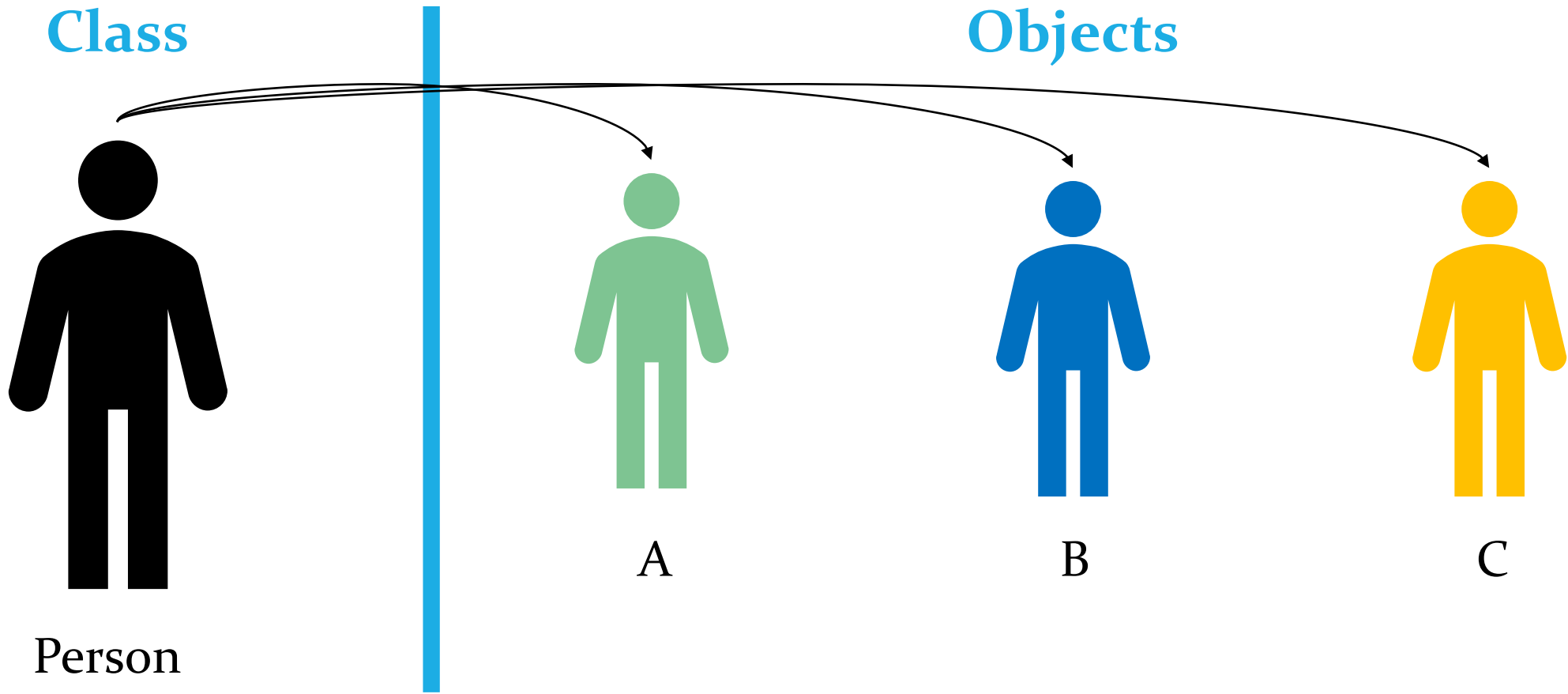
01.

**Class, Object, Attribute,
Method, Encapsulation,
Access Modifier**

Class, Object

- ❑ Class (Lớp): Mỗi lớp, hay lớp đối tượng được dùng để mô hình hóa một nhóm các thực thể cùng loại trong thế giới thực. Có thể coi lớp là một khuôn dùng làm hình mẫu cho các thực thể đó.
- ❑ Object (Đối tượng): Mỗi đối tượng được dùng để chỉ một thực thể cụ thể thuộc về một lớp nào đó. Mỗi đối tượng thuộc về một lớp còn được gọi là một thể hiện (instance) hay một “phần tử điển hình” của lớp đó.

Class, Object



Class, Object

- ❑ Trong C++ và Python thì cú pháp khởi tạo class và object sẽ có sự khác nhau như sau:

C++

```
1 class Animal {};  
2  
3 int main() {  
4     Animal animal;  
5  
6     cout << typeid(Animal).name() << endl; // class Animal  
7     cout << typeid(animal).name() << endl; // class Animal  
8  
9     return 0;  
10 }
```

Python

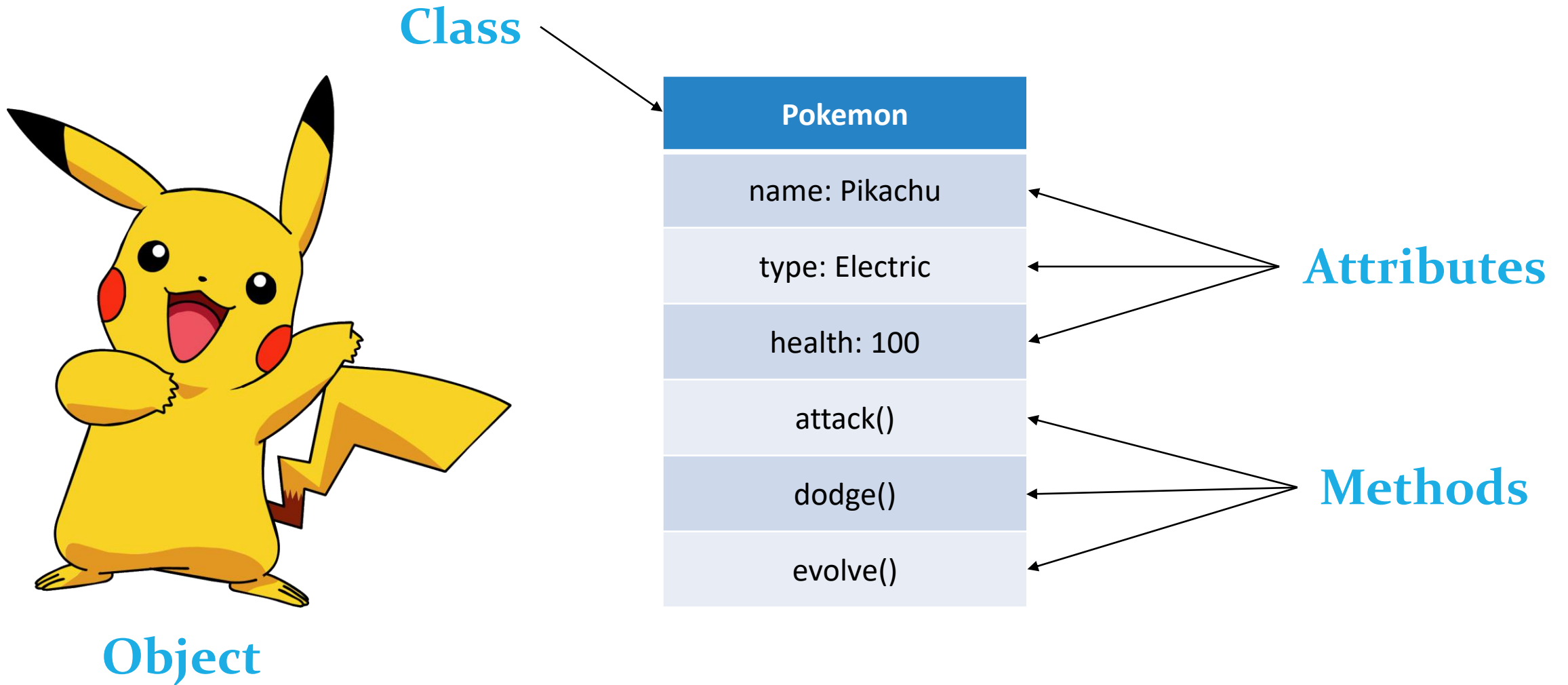
```
1 class Animal:  
2     pass  
3  
4 animal = Animal()  
5  
6 print(type(Animal)) # class type  
7 print(type(animal)) # instance type
```

Attribute, Method

Mỗi lớp bao gồm các phần tử bộ phận của lớp (class members). Việc mô tả cấu trúc một lớp bao gồm mô tả các phần chính như sau:

- ❑ Class name (tên lớp): mỗi lớp có một tên duy nhất để phân biệt với các lớp khác trong cùng một phạm vi.
- ❑ Attribute (thuộc tính): mô tả đặc tính của đối tượng. Chúng mô tả các trường để lưu dữ liệu cho mỗi đối tượng của lớp đang mô tả hay là lưu các tham chiếu đến các đối tượng của lớp khác.
- ❑ Method (phương thức): mô tả hành động của đối tượng. Mỗi phương thức của lớp thực chất là một hàm được viết riêng cho các đối tượng của lớp, chỉ được phép gọi để tác động lên chính các đối tượng đó.

Attribute, Method



Attribute

C++

- ❑ Khai báo thuộc tính bên trong thân class.
- ❑ Không cần định nghĩa trước giá trị cho các thuộc tính đó.

```
1 class Animal {  
2     // declare attributes  
3     string name;  
4     int age;  
5 };
```

Python

- ❑ Khai báo thuộc tính có thể bên trong hay bên ngoài lớp đều được.
- ❑ Khai báo phải đồng thời định nghĩa giá trị mặc định cho thuộc tính đó.

```
1 class Animal:  
2     # declare and set value for attributes inside class  
3     name = ""  
4  
5     # declare and set value for attributes outside class  
6     Animal.age = 0
```

Attribute

C++

- ☐ Khai báo thuộc tính bên trong thân class.
- ☐ Không cần định nghĩa trước giá trị cho các thuộc tính đó.
- ☐ Các đối tượng chỉ có đúng số thuộc tính mà lớp đó có.
- ☐ Dùng từ khóa “this→” để trỏ đến thuộc tính.

Python

- ☐ Khai báo thuộc tính có thể bên trong hay bên ngoài lớp đều được.
- ☐ Khai báo phải đồng thời định nghĩa giá trị mặc định cho thuộc tính đó.
- ☐ Các đối tượng có thể thêm các thuộc tính riêng cho chúng mà không thuộc về lớp của đối tượng đó.
- ☐ Dùng từ khóa “self.” để trỏ đến thuộc tính.

Attribute

C++

```
1 class Animal {
2 public:
3     // declare attributes
4     string name;
5     int age;
6
7     void description() {
8         // can use this-> to access attributes
9         cout<< this->name << " is " << this->age << " years old\n";
10    }
11 };
12
13 int main() {
14     Animal animal;
15
16     animal.name = "rocky";
17     animal.age = 10;
18
19     return 0;
20 }
```

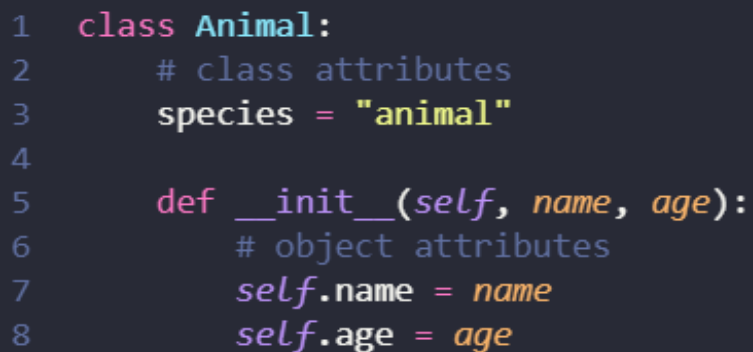
Python

```
1 class Animal:
2     name = ""
3
4     def description(self):
5         # use self. to access attributes
6         return f"{self.name} is {self.age} years old"
7
8
9     Animal.age = 0
10
11     animal = Animal()
12     animal.species = "dog"
13
14     print(animal.species) # dog
15     # print(Animal.species) # error
```

Attribute

Python

- ❑ Trong Python, thuộc tính của đối tượng có thể được khai báo bên trong constructor `__init__()` của lớp.



```
1 class Animal:
2     # class attributes
3     species = "animal"
4
5     def __init__(self, name, age):
6         # object attributes
7         self.name = name
8         self.age = age
```


Method

C++

- ☐ Khai báo phương thức bên trong lớp.
- ☐ Có thể khai báo phương thức trước, sau đó định nghĩa phương thức sau.
- ☐ Không cần truyền vào đối số là class.

Python

- ☐ Khai báo phương thức bên trong hoặc bên ngoài lớp đều được.
- ☐ Khai báo và định nghĩa phương thức phải đồng thời.
- ☐ Khai báo phương thức của đối tượng cần truyền vào đối số đầu tiên là self (chính đối tượng đó).

- ☐ Ở cả 2 ngôn ngữ thì định nghĩa phần thân phương thức bên trong hoặc bên ngoài lớp đều được.

Method

C++

```
1 class Animal {
2 public:
3     // declare attributes
4     string name;
5     int age;
6
7     // declare methods
8     // define methods inside class
9     void description() {
10         cout<< name << " is " << age << " years old\n";
11     }
12     void speak(string sound);
13 };
14
15 // define methods outside class
16 void Animal::speak(string sound) {
17     cout<< name <<"says " << sound << endl;
18 }
```

Python

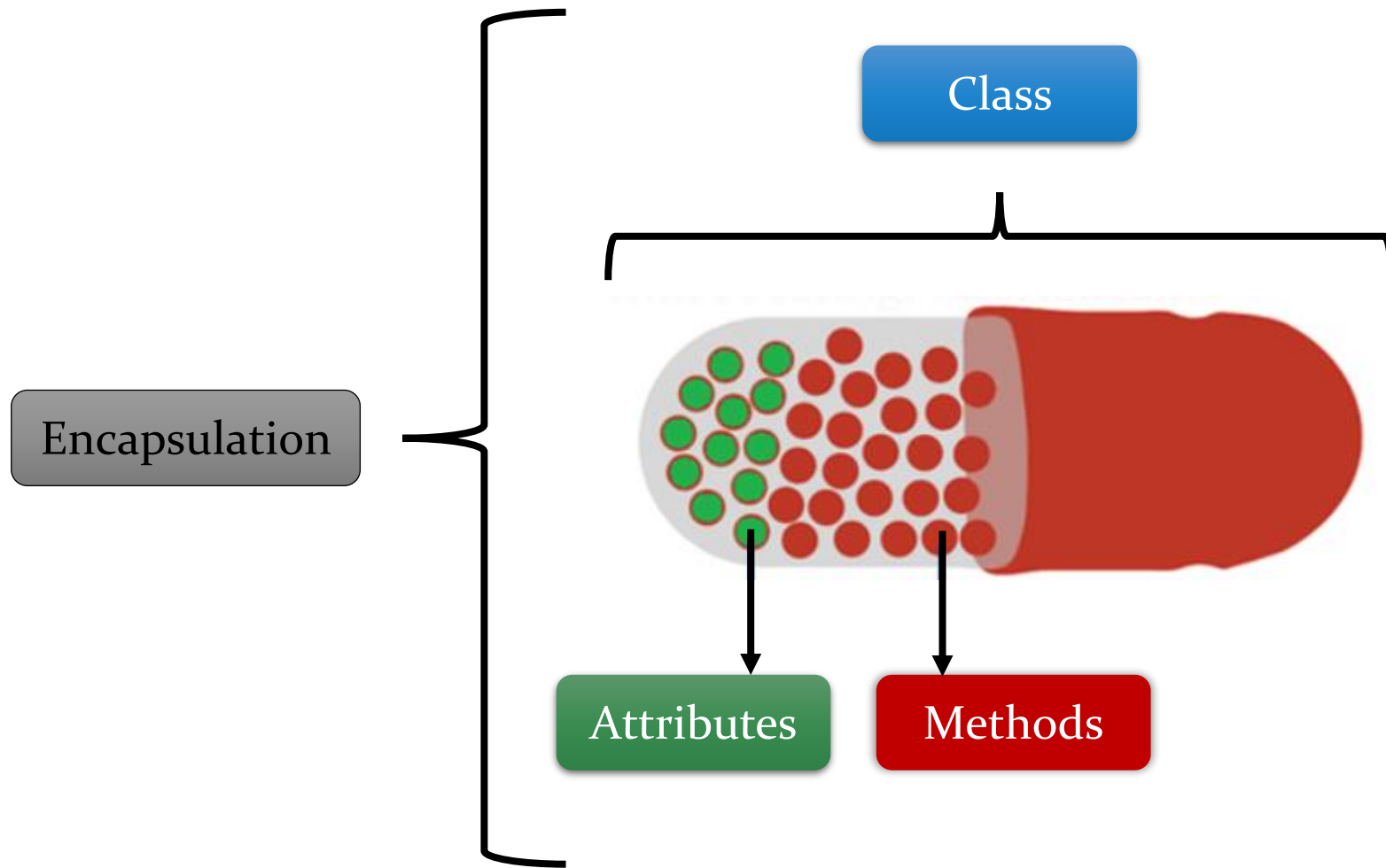
```
1 class Animal:
2     def __init__(self, name, age):
3         self.name = name
4         self.age = age
5
6     # declare and define methods inside class
7     def description(self):
8         return f"{self.name} is {self.age} years old"
9
10
11 # declare and define methods outside class
12 def func(self, sound):
13     return f"{self.name} says {sound}"
14
15 Animal.speak = func
16
17 animal = Animal("rocky", 10)
18
19 print(animal.description()) # rocky is 10 years old
20 print(animal.speak("Grr")) # rocky says Grr
```

Encapsulation

Tính đóng gói (Encapsulation) và che giấu thông tin (Hiding information):

- ☐ Là trạng thái của đối tượng được bảo vệ không cho phép các truy cập từ bên ngoài như thay đổi trạng thái hay nhìn thấy trực tiếp.
- ☐ Đây là tính chất đảm bảo sự toàn vẹn, bảo mật của đối tượng trong lập trình.
- ☐ Các thuộc tính và phương thức liên quan với nhau được đóng gói thành các lớp.
- ☐ Tính đóng gói được thể hiện thông qua phạm vi truy cập (Access Modifier).

Encapsulation




Access Modifier

Access Modifier (phạm vi truy cập, hay tầm vực): C++ và Python có 3 loại phạm vi truy cập nhất định cho các thuộc tính và phương thức của lớp:

- ☐ Public: có thể truy cập ở mọi nơi, bên trong hay bên ngoài lớp đều được.
- ☐ Protected: chỉ có chính lớp đó và các lớp kế thừa (lớp con) của nó mới truy cập được.
- ☐ Private: chỉ có lớp đó có quyền truy cập.

Access Modifier

Access Modifier (Phạm vi truy cập)	Own Class (Lớp gốc)	Derived Class (Lớp kế thừa)	Main (Chương trình chính)
Private			
Protected			
Public			

Access Modifier

C++

Public:

- ☐ Dùng từ khóa public: để thể hiện.

Protected:

- ☐ Dùng từ khóa protected: để thể hiện.

Private:

- ☐ Dùng từ khóa private: để thể hiện (mặc định).

Python

- ☐ Khai báo như bình thường (mặc định).

- ☐ Khai báo bằng cách bắt đầu bằng một dấu gạch dưới “_”.

- ☐ Khai báo bằng cách bắt đầu bằng 2 dấu gạch dưới “__”.

Access Modifier

C++

```
1 class Animal {
2     private:
3         string name;
4         string description() {
5             return name + " is " + to_string(age) + " years old\n";
6         }
7     protected:
8         int age;
9         string run(int distance) {
10             return name + " run " + to_string(distance) + " miles\n";
11         }
12     public:
13         string species;
14         string speak(string sound) {
15             return name + " says " + sound + "\n";
16         }
17
18         Animal(string name, int age, string species) {
19             this->name = name;
20             this->age = age;
21             this->species = species;
22         }
23 };
```

Python

```
1 class Animal:
2     def __init__(self, name, age, species):
3         # private attributes
4         self.__name = name
5
6         # protected attributes
7         self._age = age
8
9         # public attributes
10        self.species = species
11
12        # private methods
13        def __description(self):
14            return f"{self.__name} is {self._age} years old"
15
16        # protected methods
17        def _run(self, distance):
18            return f"{self._name} run {distance} miles"
19
20        # public methods
21        def speak(self, sound):
22            return f"{self.__name} says {sound}"
```


Access Modifier

C++

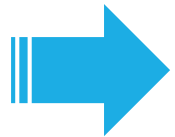
```
1  int main() {
2  Animal animal("rocky", 10, "dog");
3
4      // cout << "name: " << animal.name << endl;           // error
5      // cout << "age: " << animal.age << endl;               // error
6      cout << "species: " << animal.species << endl;         // success
7
8      // cout << animal.description() << endl;               // error
9      // cout << animal.run(10) << endl;                       // error
10     cout << animal.speak("Grr..") << endl;                 // success
11
12     return 0;
13 }
14
```

Python

```
1  animal = Animal("rocky", 10, "dog")
2
3  # check attributes access modifier
4  # print("name:", animal.__name)           # error
5  # print("age: ", animal._age)             # error
6  print("species:", animal.species)         # success
7
8  # check methods access modifier
9  # print(animal.__description())            # error
10 # print(animal._run(10))                   # error
11 print(animal.speak("Grr.."))              # success
12
```

Getter, Setter

Encapsulation (Tính đóng gói): mục đích chính là để ẩn giấu thông tin của đối tượng đi, vậy nếu ta cần lấy ra thông tin đó để sử dụng hay để sửa đổi thì phải làm sao ?



Khi đó ta dùng tới Getter và Setter.

- ☐ Getter và Setter có mối quan hệ mật thiết với tính đóng gói. Chúng không mâu thuẫn với tính chất bảo mật thông tin của tính đóng gói mà hỗ trợ cho việc quản lý thông tin đó một cách chặt chẽ và tổ chức hơn.
- ☐ Getter: lấy ra thông tin đối tượng.
- ☐ Setter: cập nhật lại thông tin đối tượng.

Getter, Setter

Getter:

C++

```
1 class Animal {  
2     private:  
3         string name;  
4     public:  
5         // getter methods  
6         string get_name() {  
7             return this->name;  
8         }  
9 };
```

Setter:

```
1 class Animal {  
2     private:  
3         string name;  
4     public:  
5         // setter methods  
6         void set_name(string name) {  
7             this->name = name;  
8         }  
9 };
```


Python

```
1 class Animal:  
2     def __init__(self, name):  
3         self.__name = name  
4  
5     # getter methods  
6     def get_name(self):  
7         return self.__name
```

```
1 class Animal:  
2     def __init__(self, name):  
3         self.__name = name  
4  
5     # setter methods  
6     def set_name(self, name):  
7         self.__name = name
```


Getter, Setter

C++



```
1 int main() {  
2     Animal animal("rocky", 10, "dog");  
3  
4     cout << animal.get_name() << endl; // rocky  
5     animal.set_name("kuro");  
6     cout << animal.get_name() << endl; // kuro  
7  
8     return 0;  
9 }
```

Python

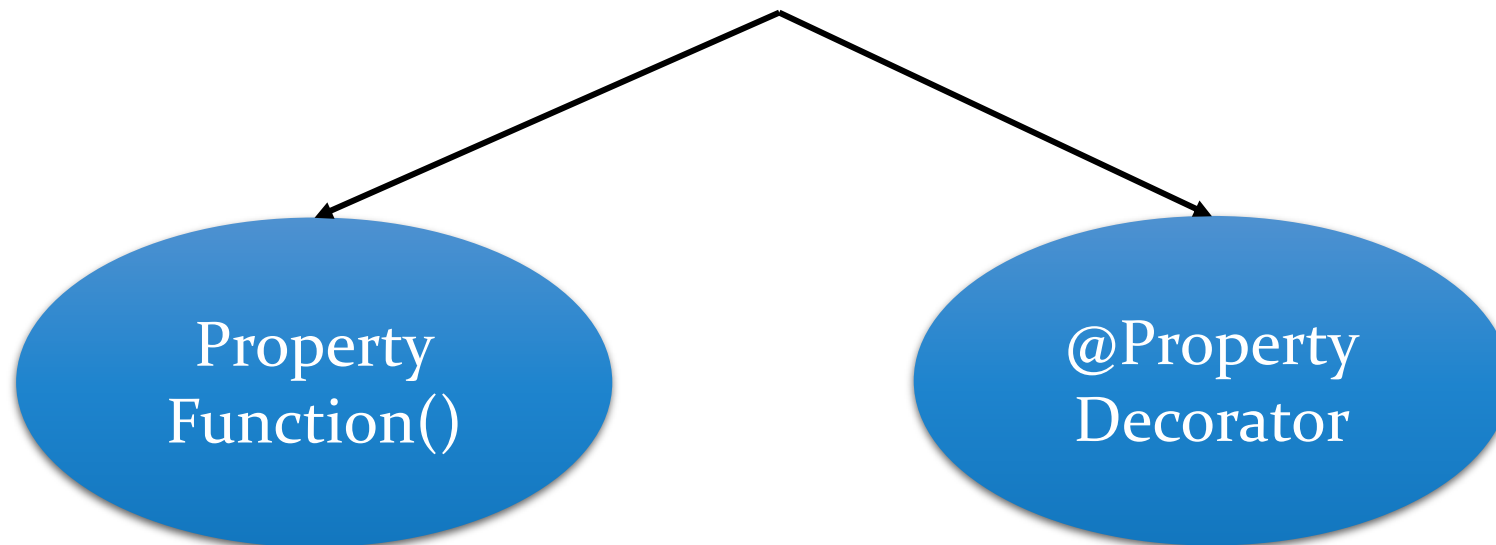


```
1 animal = Animal("rocky", 10, "dog")  
2  
3 print(animal.get_name()) # rocky  
4 animal.set_name("kuro")  
5 print(animal.get_name()) # kuro
```

Getter, Setter

Python

- ❑ Đặc biệt, trong Python nếu không muốn khi gọi đến Getter và Setter theo tên hàm như `get_name()`, `set_name()` như trên mà muốn sử dụng chúng một cách quen thuộc và thân thiện hơn như gọi thẳng “`animal.name`” cho Getter và Setter luôn thì có thể dùng `Property()` Function hoặc `@Property Decorator`.



Getter, Setter

Python

- ❑ **Property() Function:** là 1 hàm có sẵn của Python tạo ra và trả về 1 đối tượng property. Nó nhận vào tham số đầu vào lần lượt là `getter()`, `setter()` và `deleter()` (không bàn đến `deleter()` ở đây).



```
1 class Animal:
2     def __init__(self, name):
3         self.__name = name
4
5     def get_name(self):
6         return self.__name
7
8     def set_name(self, name):
9         self.__name = name
10
11     # property functions
12     name = property(get_name, set_name)
13
14
15 animal = Animal("rocky")
16
17 print(animal.name) # rocky
18 animal.name = "kuro"
19 print(animal.name) # kuro
```

Getter, Setter

Python

- ❑ @Property Decorator: Mục đích dùng cũng để làm cho việc sử dụng Setter và Getter quen thuộc như Property() Function. Nhưng cách khai báo bên trong lớp có sự khác biệt.

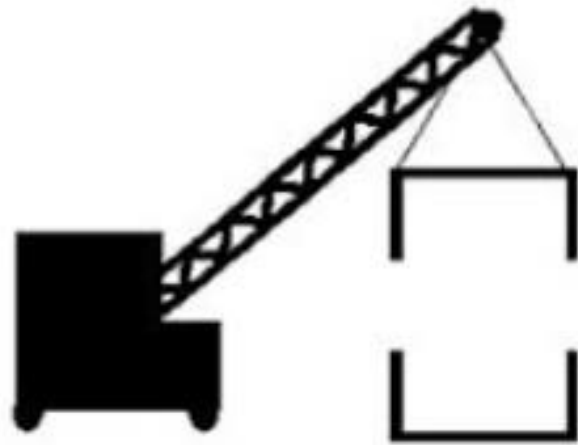


```
1 class Animal:
2     def __init__(self, name):
3         self.__name = name
4
5     # using property decorator
6     # getter methods
7     @property
8     def name(self):
9         return self.__name
10
11    # setter methods
12    @name.setter
13    def name(self, name):
14        self.__name = name
15
16
17    animal = Animal("rocky")
18
19    print(animal.name) # rocky
20    animal.name = "kuro"
21    print(animal.name) # kuro
```

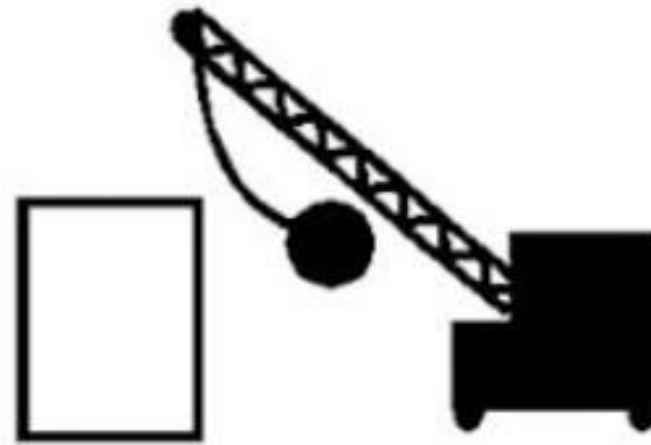
The background is a solid dark blue color. It is decorated with various light blue geometric elements: several thin diagonal lines, some circles of different sizes, and larger, semi-transparent rounded rectangular shapes that overlap each other. In the center of the image is a dark blue rectangular box with a thin white border. Inside this box, the text '02. Constructor, Destructor' is written in a white, serif font. The text is centered both horizontally and vertically within the box.

02. Constructor, Destructor

Constructor, Destructor



Constructor



Destructor

Constructor

- ❑ Constructor (phương thức khởi tạo) là một phương thức đặc biệt ở trong class, phương thức này mặc định sẽ được gọi khi chúng ta khởi tạo class đó.
- ❑ Thường được dùng để khởi tạo các thuộc, xử lý phương thức hoặc là dùng để nhận các tham số truyền vào class khi khởi tạo.
- ❑ Constructor sẽ tự động chạy ngay khi đối tượng được tạo.

Constructor

Các loại constructor

- ❑ Default Constructor: phương thức tạo lập mặc định.
- ❑ Parameterized Constructor: phương thức tạo lập có tham số đầu vào.
- ❑ Copy Constructor: phương thức tạo lập sao chép.

Constructor

Cú pháp

C++

- ❑ Tên phương thức trùng với tên lớp, có thể nhận các tham số truyền vào tương ứng cho từng thuộc tính.

```
1 class className {  
2     className(data_type params1, data_type params2, ...) {  
3  
4     }  
5 };
```

Python


- ❑ Tên phương thức khởi tạo là `__init__()`, luôn nhận tham số đầu vào đầu tiên là `self`.

```
1 class className:  
2     def __init__(self, params1, params2, ...):  
3         # code
```

Constructor


Default Constructor

C++



```
1  class Animal {
2  private:
3      string name;
4      int age;
5  public:
6      // default constructor
7      Animal() {
8          name = "Animal";
9          age = 0;
10     }
11 };
```

Python



```
1  class Animal:
2      # default constructor
3      def __init__(self):
4          self.name = "Animal"
5          self.age = 0
```

Constructor

Parameterized Constructor

C++

```
1 class Animal {
2 private:
3     string name;
4     int age;
5 public:
6     // parameterized constructor
7     Animal(string name, int age) {
8         this->name = name;
9         this->age = age;
10    }
11 };
```

Python

```
1 class Animal:
2     # parameterized constructor
3     def __init__(self, name, age):
4         self.name = name
5         self.age = age
```

Constructor

Copy Constructor

C++

```
1 class Animal {  
2     private:  
3         string name;  
4         int age;  
5     public:  
6         // copy constructor  
7         Animal(const Animal& other) {  
8             this->name = other.name;  
9             this->age = other.age;  
10        }  
11    };
```

Python

- ❑ Python không có `__init__()` copy constructor. Nhưng thay vào đó có thể import thư viện `copy` để dùng.

```
1 import copy  
2  
3 class Animal:  
4     pass  
5  
6 animal = Animal()  
7  
8 clone = copy.copy(animal) # shallow copy of animal  
9 clone = copy.deepcopy(clone) # deep copy of animal
```

Constructor

Overloading Constructor

C++

- ❑ Cho phép khởi tạo đối tượng bằng nhiều constructor khác nhau.
- ❑ Mặc định compiler sẽ tạo ra default constructor cho class. Nếu cài đặt các constructor khác thì bắt buộc phải cài đặt default constructor nếu không sẽ gây lỗi.

Python

- ❑ Chỉ được có duy nhất 1 constructor cho 1 lớp, nếu có cả default constructor và parameterized constructor thì lớp đó sẽ áp dụng parameterized constructor cho mọi đối tượng được khởi tạo.

Constructor

Áp dụng

C++

```
1  int main() {
2      // default constructor
3      Animal animal1;
4      animal1.description(); // Animal is 0 years old
5
6      // parameterized constructor
7      Animal animal2("Rocky", 10);
8      animal2.description(); // Rocky is 10 years old
9
10     // copy constructor
11     Animal animal3(animal2);
12     animal3.description(); // Rocky is 10 years old
13
14     // destructor will call before the program finish
15     return 0;
16 }
```

Python

```
1  # default constructor
2  # animal1 = Animal() # error
3
4  # parameterized constructor
5  animal2 = Animal("Rocky", 10)
6  animal2.description() # Rocky is 10 years old
7
8  # copy class by using copy library
9  clone1 = copy.copy(animal2) # shallow copy of animal
10 clone2 = copy.deepcopy(animal2) # deep copy of animal
11
12 clone1.description() # Rocky is 10 years old
13 clone2.description() # Rocky is 10 years old
```

Destructor

- ❑ Destructor (phương thức hủy): Trái ngược với phương thức khởi tạo, thì phương thức hủy sẽ được gọi khi chúng ta hủy một lớp.
- ❑ Luôn được thực thi cuối cùng khi chúng ta khởi tạo một class, thường dùng để giải phóng tài nguyên của lớp.
- ❑ Chỉ có duy nhất 1 destructor cho 1 lớp.
- ❑ Tự động chạy mỗi khi đối tượng của lớp bị hủy (hết phạm vi sử dụng).
- ❑ Trong một quá trình sống của đối tượng chỉ có 1 lần duy nhất destructor được gọi thực hiện.

Destructor

Cú pháp

C++

- ❑ Tên phương thức trùng với tên lớp, có dấu ~ ở đầu, không có tham số đầu vào và không có kiểu trả về

```
1 class className {  
2     ~className() {  
3         // code  
4     }  
5 };
```

Python

- ❑ Tên phương thức hủy là __del__(), chỉ có duy nhất 1 tham số đầu vào là self.

```
1 class className:  
2     # destructor  
3     def __del__(self):  
4         # code
```

Destructor

Áp dụng

C++

- ❑ Mặc định compiler sẽ tự động tạo ra Destructor cho lớp nếu không được định nghĩa. Và chỉ cần cài đặt Destructor khi lớp đối tượng sử dụng vùng nhớ cấp phát động hay tài nguyên của hệ thống.

Python

- ❑ Destructor không thật sự cần thiết như bên C++ vì Python có garbage collector sẽ quản lý việc sử dụng vùng nhớ một cách tự động và hủy nếu cần thiết.

Destructor

Áp dụng

C++

```
1 // destructor
2 ~Animal() {
3     cout << "Animal destructor" << endl;
4 }
```

Python

```
1 # destructor
2 def __del__(self):
3     print("Animal destructor")
```

➡ Cài đặt destructor vào trong chương trình đã viết ở constructor ở trên, ta được kết quả chạy chương trình ở cả 2 ngôn ngữ:

```
Rocky is 10 years old
Rocky is 10 years old
Rocky is 10 years old
Animal destructor
Animal destructor
Animal destructor
```

The background is a solid dark blue color. It is decorated with various light blue geometric elements: several thin diagonal lines, some circles of different sizes, and larger, semi-transparent rounded rectangular shapes that overlap each other. In the center of the image is a dark blue rectangular box with a thin white border. Inside this box, the text '03. Static Attribute, Static Method' is written in a white, serif font. The number '03.' is on the top line, and the words 'Static Attribute, Static Method' are on the two lines below it.

03. Static Attribute, Static Method

Static Attribute



Tại sao cần dùng đến static attribute (thuộc tính tĩnh) ?

Đặt vấn đề

Static Attribute

- ❑ Trong 1 số trường hợp, ta có nhu cầu sử dụng một biến là thành phần dữ liệu chung cho tất cả các đối tượng. Lúc này, ta có thể sử dụng biến toàn cục tuy nhiên cách này sẽ có thể gây ra nguy cơ nhất định về mặt ngữ nghĩa và khó bảo quản code, nên sẽ không hợp lý.



Để giải quyết vấn đề này, ta có thể dùng static attribute (thuộc tính tĩnh). Thuộc tính này không phụ thuộc vào 1 đối tượng cụ thể mà nó là thuộc tính chung của lớp bao gồm tất cả các đối tượng của nó.

Static Attribute

Cú pháp

C++

- ❑ Khai báo giống thuộc tính bình thường nhưng có thêm từ khóa “static” phía trước.

```
1 class className {  
2     private:  
3         static data_type variableName;  
4 };
```

Python

- ❑ Khai báo và khởi tạo giá trị ngay bên trong lớp (không cần qua constructor như các thuộc tính khác)

```
1 class className:  
2     variableName = "(set value)"
```

Static Attribute

Truy cập

C++

- ❑ Truy cập thông qua tên lớp bằng toán tử ::

```
1 class Animal {  
2     private:  
3         static int count;  
4 };  
5  
6 int Animal::count = 0;
```

Python

- ❑ Có thể truy cập bằng cả tên lớp và đối tượng.

```
1 class Animal:  
2     count = 0  
3  
4     animal = Animal()  
5  
6     print(Animal.count)      # access through class  
7     print(animal.count)     # access through object
```

Static Attribute

Python

- ❑ Có thể thay đổi giá trị thuộc tính tĩnh thông qua lớp và đối tượng.

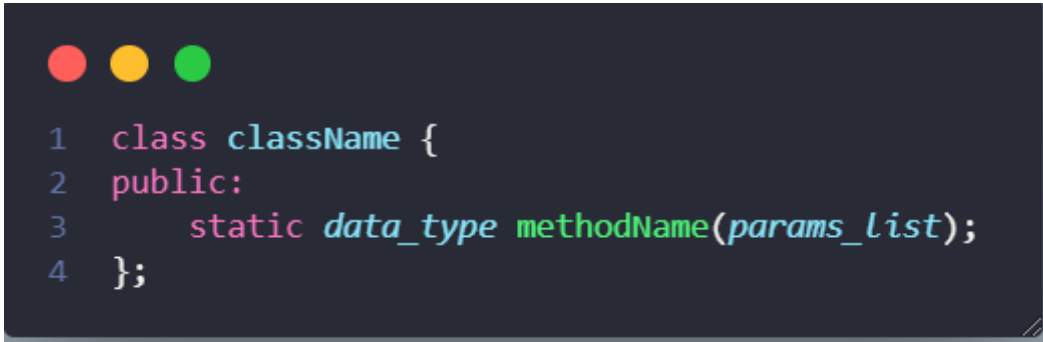
```
1 class Animal:
2     count = 0
3
4
5 animal1 = Animal()
6 animal2 = Animal()
7
8 animal1.count = 1
9
10 print(Animal.count) # 0
11 print(animal1.count) # 1 -> change
12 print(animal2.count) # 0
13
14 Animal.count = 2
15 print(Animal.count) # 2 -> change
16 print(animal1.count) # 1 -> still have value 1
17 print(animal2.count) # 2 -> change with Animal
```

Static Method

C++

- ❑ Tương tự như static attribute, trong trường hợp một số phương thức của lớp khi sử dụng không cần một đối tượng cụ thể nào đó, ta cũng có thể khai báo chúng là các phương thức tĩnh của lớp (static method).

- ❑ Cú pháp:



```
1 class className {  
2     public:  
3         static data_type methodName(params_list);  
4 };
```

- ❑ Truy cập thông qua tên lớp bằng toán tử ::

Static Method

C++



```
1 class Animal {
2     private:
3         static int count;
4     public:
5         Animal() {
6             count++;
7         }
8         ~Animal() {
9             count--;
10        }
11        static int show() {
12            return count;
13        }
14    };
15
16    int Animal::count = 0;
17
18    int main() {
19        Animal animal1;
20        Animal animal2;
21
22        cout << "Number of Animal: " << Animal::show() << endl; // 2 instance
23
24        return 0;
25    }
```

Static Method

Python

- ❑ Phương thức trước giờ ta dùng là instance method (phương thức thể hiện) thông thường. Nó sẽ nhận lấy 1 tham số đầu vào là self trỏ đến lớp đó khi phương thức được gọi. Khi phương thức được gọi, Python sẽ thay thế đối số self bằng instance object tạo bởi lớp đó.

```
1 class Animal:
2     def __init__(self, name, age):
3         self.name = name
4         self.age = age
5
6     # instance methods
7     def description(self):
8         print(f"{self.name} is {self.age} years old")
9
10
11 animal = Animal("Rocky", 10)
12 animal.description()    # Rocky is 10 years old
```

Static Method

Python

- ❑ Ngoài ra còn có 2 phương thức đặc biệt của lớp là class method (phương thức lớp) và static method (phương thức tĩnh), cả 2 phương thức này đều được gắn với chính lớp đó chứ không đến đối tượng của lớp.



Static Method

Python

Class Method

Cú pháp:

- ❑ Đánh dấu bằng Decorator là `@classmethod`.

```
1 class className:
2     @classmethod
3     def fun(cls, arg1, arg2, ...):
4         # code
```

Tham số đầu vào:

- ❑ Thay vì `self`, class method lấy tham số đầu tiên là `cls` trỏ đến class.

Static Method

- ❑ Đánh dấu bằng Decorator là `@staticmethod`.

```
1 class className:
2     @staticmethod
3     def fun(arg1, arg2, ...):
4         # code
```

- ❑ Không nhận `self` hay `cls` làm tham số.

Static Method

Python

Class Method

Truy cập:

- ❑ Có thể truy cập các thuộc tính và sửa đổi trạng thái của lớp. Khi sửa đổi thì sẽ cập nhật cho tất cả các đối tượng của lớp đó.

Áp dụng:

- ❑ Thường được dùng để tạo factory methods. Nó sẽ trả về các đối tượng của lớp (giống như constructor) cho nhiều mục đích khác nhau.

Static Method

- ❑ Không thể truy cập hay sửa đổi trạng thái của lớp.

- ❑ Gần như không can hệ gì đến lớp, nhưng nó vẫn tồn tại để tạo tính hợp lý khi sử dụng. Mục đích chính là tạo ra các utility methods (các nhóm hàm tiện ích) cho lớp.

Static Method

Python

```
1 from datetime import date
2
3
4 class Animal:
5     def __init__(self, name, age):
6         self.name = name
7         self.age = age
8
9     # a class method to create a Animal object by birth year.
10    @classmethod
11    def fromBirthYear(cls, name, year):
12        return cls(name, date.today().year - year)
13
14    # a static method to check if a Animal is old or not.
15    @staticmethod
16    def isOldEnough(age):
17        return age > 10
18
19
20 animal1 = Animal("Haruto", 7) # parameterized constructor
21 animal2 = Animal.fromBirthYear("Kuro", 2010) # constructor by using class method
22
23 print(animal1.age) # 7
24 print(animal2.age) # 11
25
26 print(Animal.isOldEnough(15)) # True
```

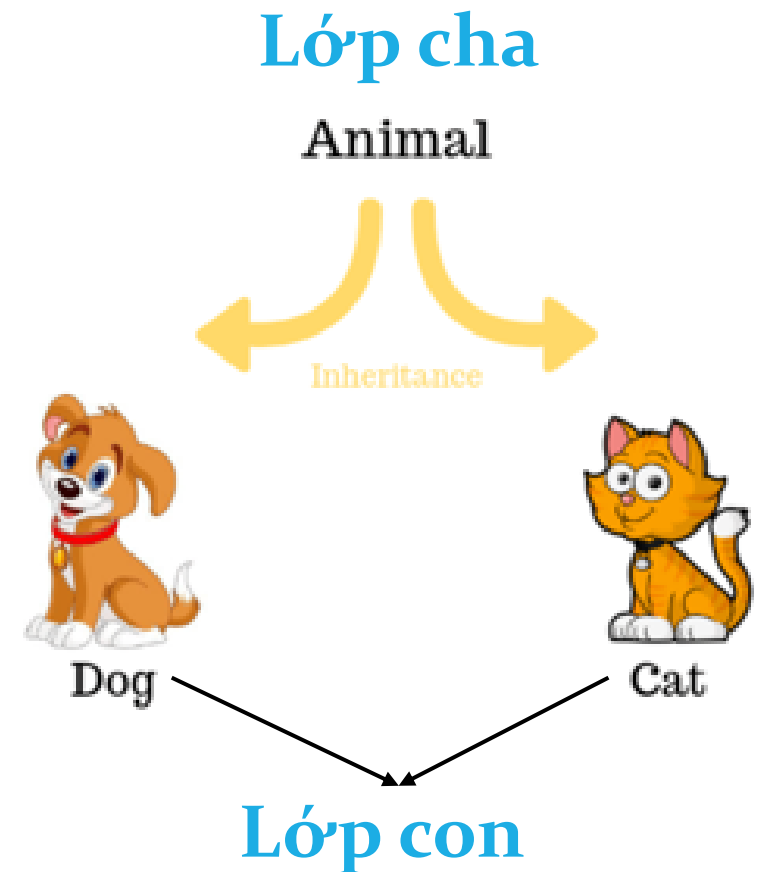


The background is a solid dark blue color. It is decorated with various light blue geometric elements: several thin diagonal lines, some circles of different sizes, and larger, semi-transparent rounded rectangular shapes that overlap each other. In the center of the image, there is a dark blue rectangular box with a thin white border. Inside this box, the text '04. Inheritance' is written in a white, serif font. The number '04.' is on the top line, and the word 'Inheritance' is on the line below it.

04. Inheritance

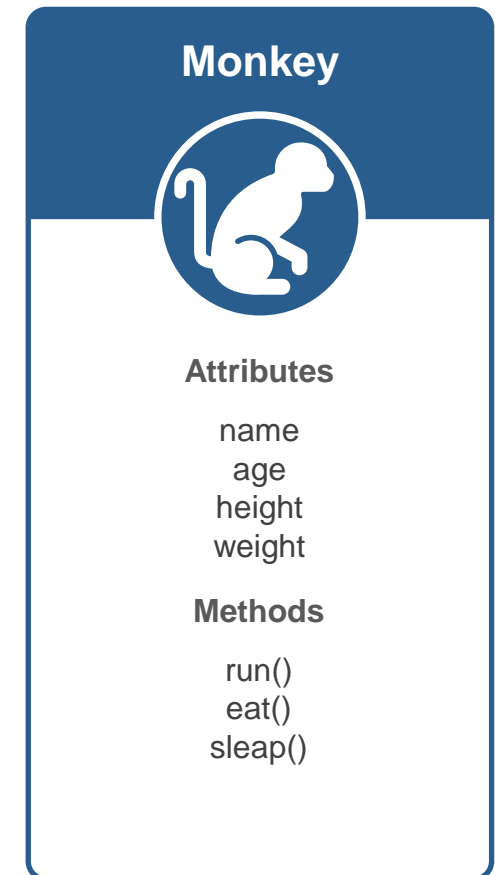
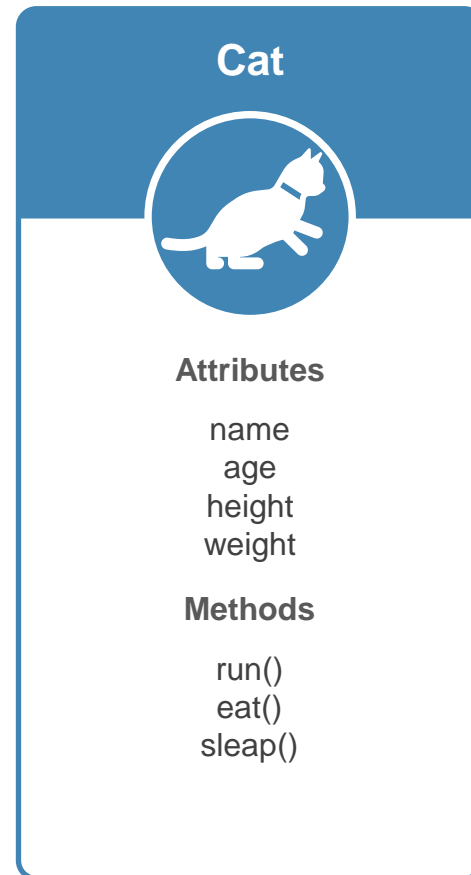
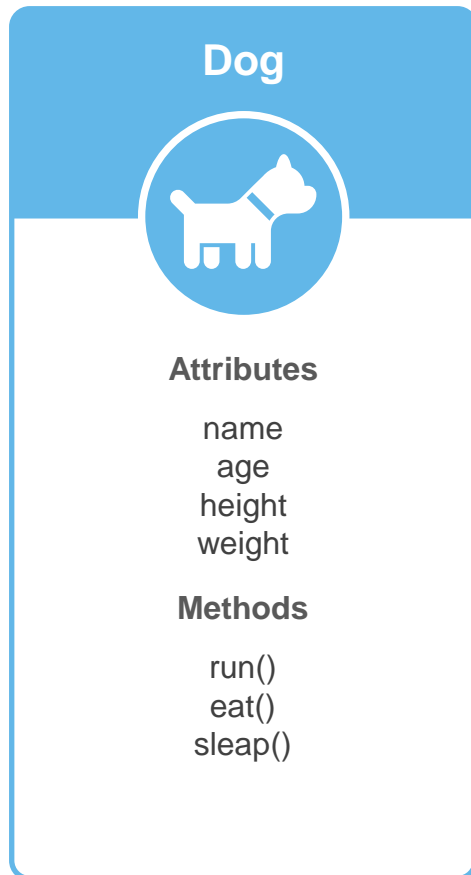
Inheritance

- ❑ Inheritance (Tính kế thừa) là một trong những đặc tính quan trọng nhất của lập trình hướng đối tượng.
- ❑ Nó là khả năng lấy một thuộc tính, đặc tính của một lớp cha để áp dụng lên lớp con.
- ❑ Lớp kế thừa các thuộc tính từ một lớp khác được gọi là Lớp con hoặc Lớp dẫn xuất.
- ❑ Lớp có các thuộc tính được kế thừa bởi lớp con được gọi là Lớp cha hoặc Lớp cơ sở.



Inheritance

Bài toán: Tạo ra 3 lớp Dog, Cat và Monkey với các thuộc tính và phương thức như sau:



Inheritance

Đặt vấn đề

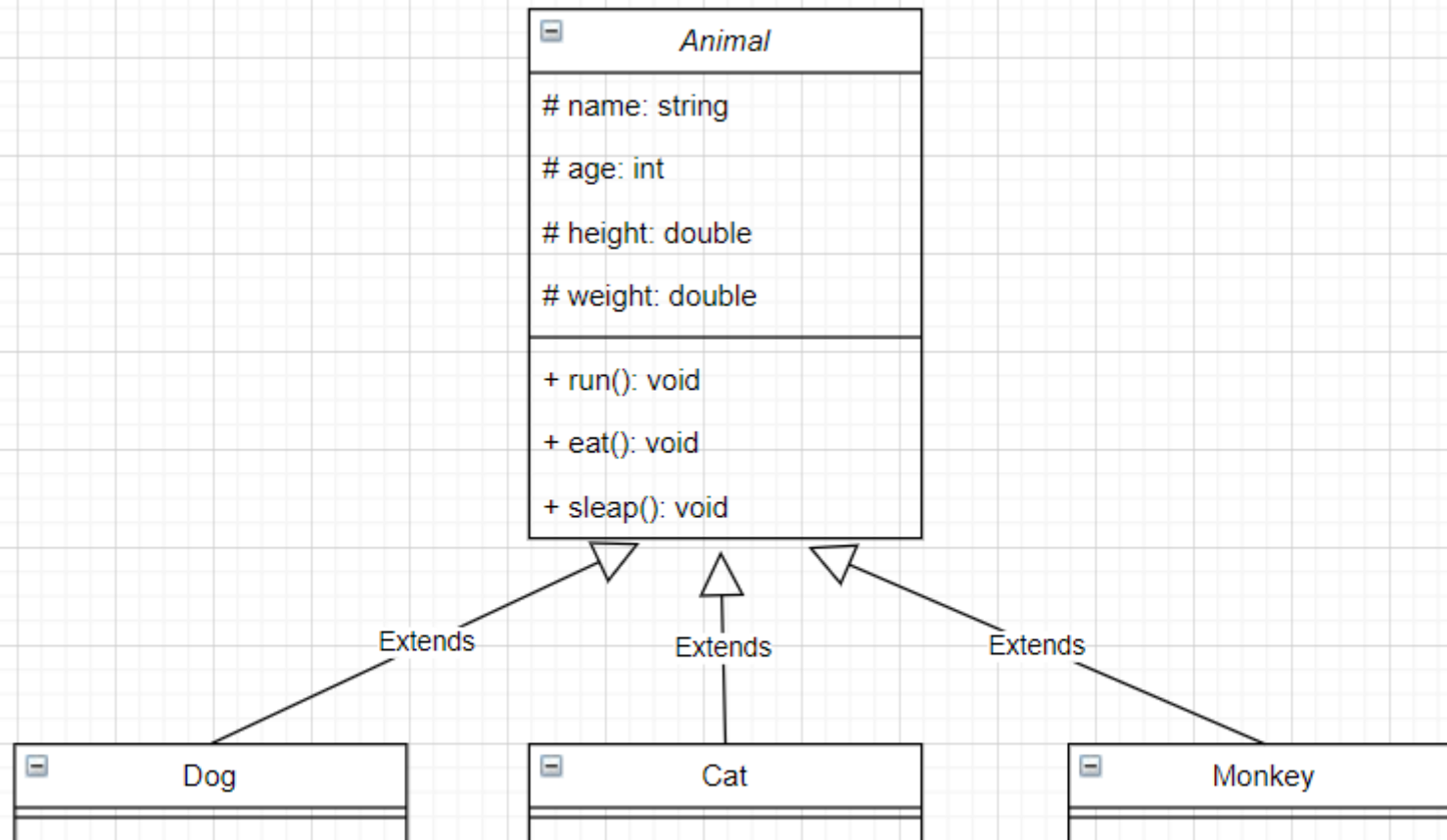
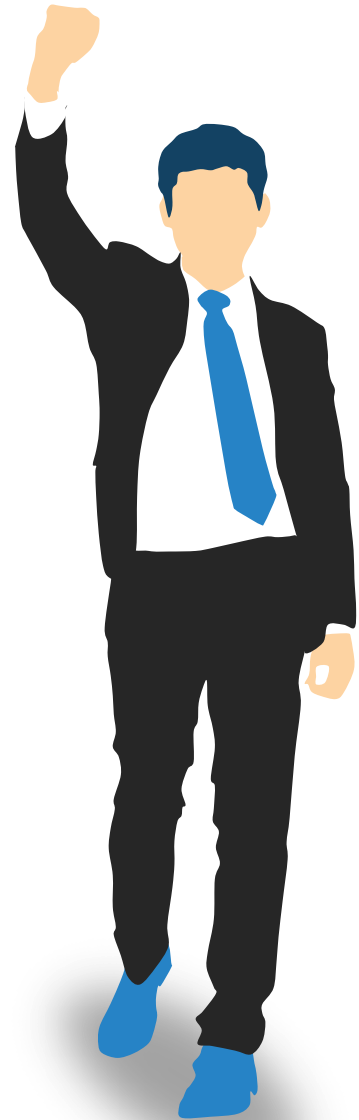
- ❑ Cả 3 lớp trên đều có chung thuộc tính và phương thức. Nếu ta tạo ra 3 lớp đó thì phải viết lặp lại 3 lần liên tục, rất bất tiện.
- ❑ Hơn thế nữa, nếu muốn sửa lại code trong một phương thức nào đó thì phải sửa chúng cả ở 3 lớp sẽ rất tốn thời gian, và có thể dễ sai sót.



Áp dụng tính kế thừa để giải quyết vấn đề trên: Ta tạo ra một lớp Animal có các thuộc tính và phương thức của cả 3 lớp trên và cho 3 lớp Dog, Cat và Monkey kế thừa từ Animal.

Inheritance

Mô hình UML



Inheritance

Ở đây ta chỉ
bàn về Single
Inheritance

Các loại kế thừa

Single Inheritance

Đơn kế thừa

Một lớp chỉ
được kế thừa
từ duy nhất một
lớp cha

Đa kế thừa

Một lớp có thể
kế thừa từ
nhiều hơn một
lớp cơ sở

Multiple Inheritance

Multilevel Inheritance

Kế thừa đa cấp

Một lớp con
được tạo từ
một lớp con
khác

Kế thừa phân cấp

Có nhiều hơn
một lớp con
được kế thừa
từ một lớp cha
duy nhất

Hierarchical Inheritance

Hybrid Inheritance

Kế thừa lai

Được thực hiện
bằng cách kết
hợp nhiều hơn
một loại thừa
kế

Inheritance

Cú pháp

C++

```
1 class subclass_name : access_mode base_class_name
2 {
3     // body of subclass
4 };
```

❑ Trong đó:

Subclass_name: tên lớp con

Base_class_name: tên lớp cha

access_mode: phạm vi kế thừa

Python

```
1 class subclass(base_class):
2     pass
```

```
1 class A:
2     pass
3
4 class A(object):
5     pass
```

❑ Class Object được xem là gốc của mọi class. Trong Python 3.x, “class A(object)” và “class A” là giống nhau.

Inheritance

Phạm vi kế thừa

C++

Phạm vi truy cập lớp cha	Phạm vi kế thừa		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Không truy cập được	Không truy cập được	Không truy cập được

Python

- ❑ Chỉ có 1 phạm vi kế thừa duy nhất (có thể xem giống như Public bên C++).

Inheritance

Thêm mới thuộc tính và phương thức

C++

```
1 class Animal {
2 protected:
3     string name;
4     int age;
5 };
6
7 class Dog : public Animal {
8 private:
9     // new attributes
10    string breed;
11 public:
12    // new methods
13    void bark() {
14        cout << name << " is barking\n";
15    }
16 };
```

Python

```
1 # Base or Super Class
2 class Animal:
3     def __init__(self, name, age):
4         self.name = name
5         self.age = age
6
7
8 # Inherited or Subclass
9 class Dog(Animal):
10     def __init__(self, name, age, breed):
11         super().__init__(name, age)
12
13         # new attributes
14         self.breed = breed
15
16         # new methods
17         def bark(self):
18             print(f"{self.name} is barking")
```

Inheritance

Constructor, Destructor

C++

- ❑ Khi constructor hoặc destructor được override (ghi đè) lại thì trình biên dịch sẽ mặc định gọi đến default constructor hoặc default destructor của lớp cha để thực hiện.

Python

- ❑ Khi constructor hoặc destructor được override lại thì toàn bộ mã code bên trong hàm đó sẽ bị thay đổi hết. Để gọi đến constructor hay destructor của lớp cha phải thông qua hàm `super()` hoặc dùng tên lớp cha với đối số `self` để sử dụng.

Inheritance

Constructor, Destructor

C++

- ❑ Thứ tự thực hiện:
 - Constructor: gọi đến constructor lớp cha trước rồi đến constructor lớp con.
 - Destructor: gọi đến destructor lớp con trước rồi đến destructor lớp cha.
- ❑ Có thể tùy chọn constructor của lớp cha để áp dụng cho lớp con.

Python

- ❑ Thứ tự thực hiện: Tùy thuộc vào code bên trong các hàm đó được override lại như thế nào.

Inheritance

Constructor, Destructor



C++

```
1 class Animal {
2     protected:
3         string name;
4         int age;
5     public:
6         Animal() {
7             name = "";
8             age = 0;
9             cout << "Animal default constructor\n";
10        }
11        Animal(string name, int age) {
12            this->name = name;
13            this->age = age;
14            cout << "Animal parameterized constructor\n";
15        }
16        ~Animal() {
17            cout << "Animal destructor\n";
18        }
19    };
```

```
1 class Dog : public Animal {
2     private:
3         string breed;
4     public:
5         Dog() {
6             breed = "";
7             cout << "Dog default constructor\n";
8         }
9         // inherit the parameterized constructor from the parent
10        Dog(string name, int age, string breed) : Animal(name, age){
11            this->breed = breed;
12            cout << "Dog parameterized constructor\n";
13        }
14        ~Dog() {
15            cout << "Dog destructor\n";
16        }
17    };
```

```
1 int main() {
2     Dog dog("Rocky", 10, "Husky");
3     return 0;
4 }
```



```
Animal parameterized constructor
Dog parameterized constructor
Dog destructor
Animal destructor
```

Inheritance

Constructor, Destructor



Python

```
1 class Animal:
2     def __init__(self, name, age):
3         self.name = name
4         self.age = age
5         print("Animal constructor")
6
7     def __del__(self):
8         print("Animal destructor")
9
10
11 class Dog(Animal):
12     def __init__(self, name, age, breed):
13         super().__init__(name, age)
14         self.breed = breed
15         # Animal.__init__(self, name, age) # the same to super()
16         print("Dog constructor")
17
18     def __del__(self):
19         super().__del__()
20         # Animal.__del__(self) # the same to super()
21         print("Dog destructor")
22
23
24 dog = Dog("Rocky", 10, "Husky")
```

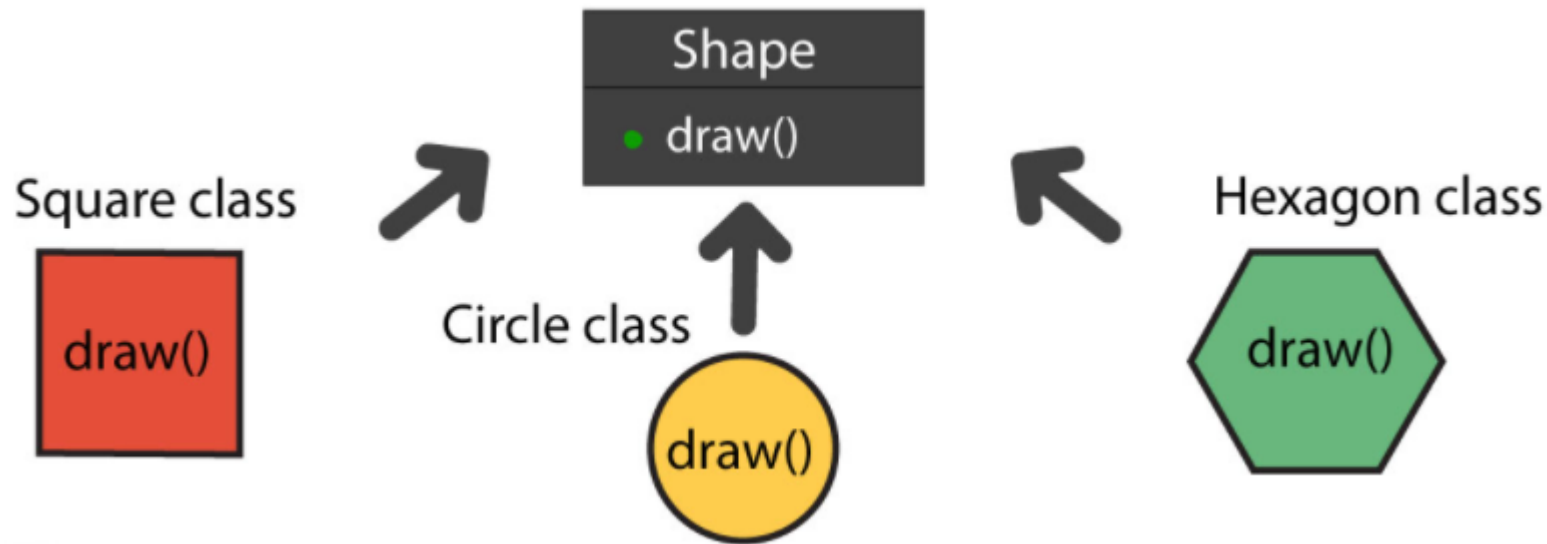


Animal constructor
Dog constructor
Animal destructor
Dog destructor

Inheritance

Overriding

- ❑ Overriding (ghi đè phương thức): là một tính năng cho phép một lớp con cung cấp một triển khai cụ thể của phương thức đã được cung cấp bởi một trong các lớp cha của nó. Nói dễ hiểu hơn, nếu lớp con có một hoặc nhiều phương thức giống với một trong các lớp cha của nó, thì đó là ghi đè phương thức.



Inheritance

Overriding

C++

```
1  class Animal {
2  protected:
3      string name;
4      int age;
5  public:
6      Animal();
7      Animal(string name, int age);
8      void description() {
9          cout << "Animal " << name << " is " << age << " years old\n";
10     }
11 };
12
13 class Dog : public Animal {
14 private:
15     string breed;
16 public:
17     Dog();
18     Dog(string name, int age, string breed);
19     ~Dog();
20     // override methods
21     void description() {
22         cout << "Dog " << name << " is " << age << " years old and is a " << breed << endl;
23     }
24 };
25
26 int main() {
27     Animal animal("Kuro", 5);
28     animal.description(); // Animal Kuro is 5 years old
29
30     Dog dog("Rocky", 10, "Husky");
31     dog.description(); // Dog Rocky is 10 years old and is a Husky
32
33     return 0;
34 }
```

Python

```
1  class Animal:
2      def __init__(self, name, age):
3          self.name = name
4          self.age = age
5
6      def description(self):
7          print(f"Animal {self.name} is {self.age} years old")
8
9
10 class Dog(Animal):
11     def __init__(self, name, age, breed):
12         super().__init__(name, age)
13         self.breed = breed
14         print("Dog constructor")
15
16     # override methods
17     def description(self):
18         print(f"Dog {self.name} is {self.age} years old and is a {self.breed}")
19
20
21 animal = Animal("Kuro", 5)
22 animal.description() # Animal Kuro is 5 years old
23
24 dog = Dog("Rocky", 10, "Husky")
25 dog.description() # Dog Rocky is 10 years old and is a Husky
```

The background is a solid dark blue color. It is decorated with various light blue geometric elements: several thin diagonal lines, some circles, and larger, semi-transparent rounded rectangular shapes that overlap each other. In the center of the image is a dark blue rectangular box with a thin white border. Inside this box, the text '05. Polymorphism' is written in a white, serif font. The number '05.' is on the top line, and the word 'Polymorphism' is on the line below it.

05. Polymorphism

Polymorphism

- ❑ Polymorphism (Tính đa hình): là tính năng cho phép các đối tượng khác nhau thực thi chức năng giống nhau theo những cách khác nhau.

- ❑ Ví dụ:
 - Chó và mèo cùng nghe mệnh lệnh “kêu đi” từ người chủ. Chó sẽ “gâu gâu” còn mèo lại kêu “meo meo”.
 - Ở lớp smartphone, mỗi một dòng máy đều kế thừa các thành phần của lớp cha nhưng iPhone chạy trên hệ điều hành iOS, còn Samsung lại chạy trên hệ điều hành Android.

Polymorphism

Smartphone

OS ?

iOS



Iphone

Android



Samsung

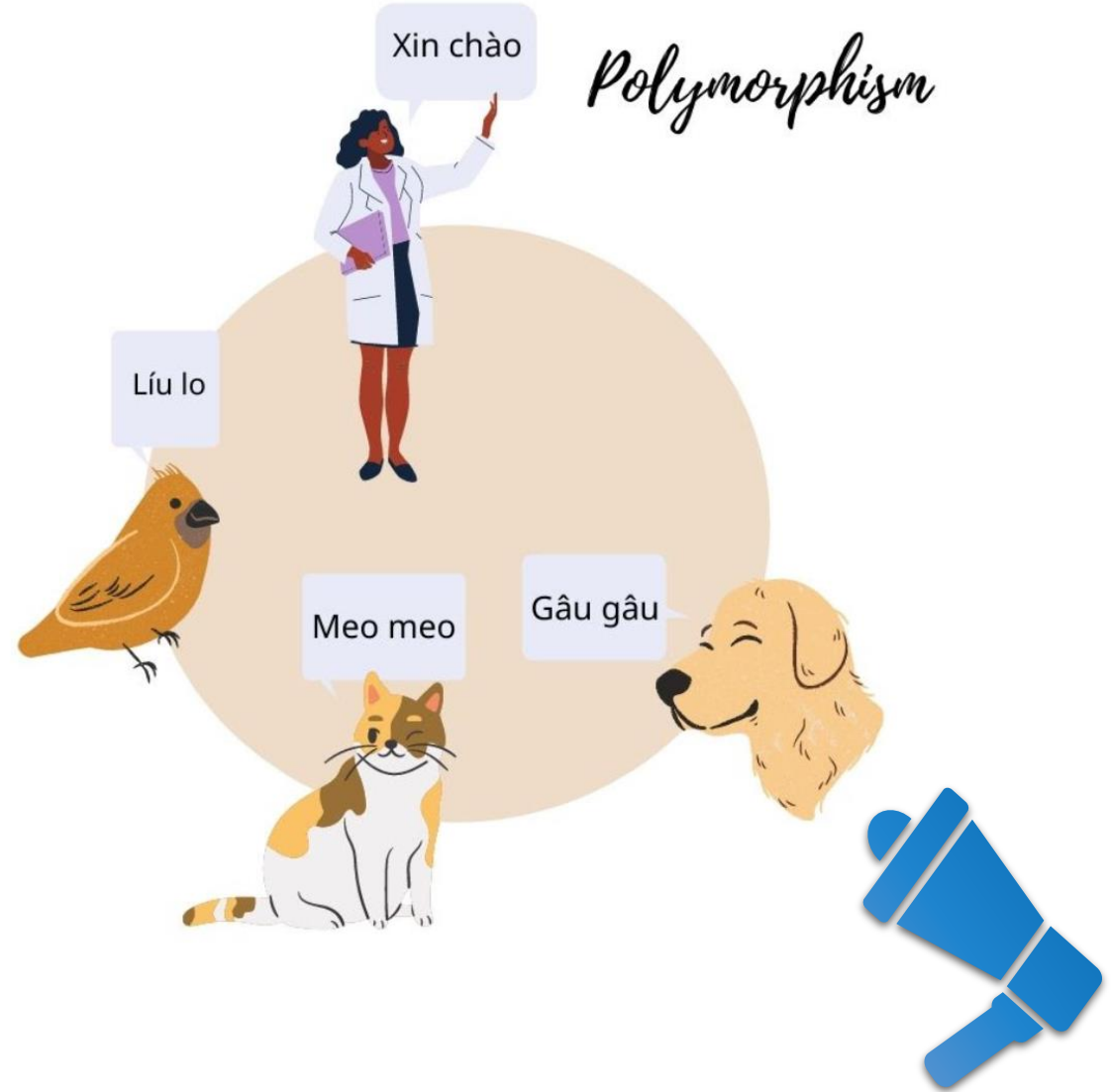
Xin chào

Polymorphism

Líu lo

Meo meo

Gâu gâu



Polymorphism

C++

Trong C++, Polymorphism (tính đa hình) tập trung xoay quanh con trỏ và kế thừa của lớp. Sự kế thừa trong C++ cho phép có sự tương ứng giữa lớp cơ sở và các lớp dẫn xuất trong sơ đồ thừa kế:

- ❑ Một con trỏ có kiểu lớp cơ sở luôn có thể trỏ đến địa chỉ của một đối tượng của lớp dẫn xuất.
- ❑ Tuy nhiên, khi thực hiện lời gọi một phương thức của lớp, trình biên dịch sẽ quan tâm đến kiểu của con trỏ chứ không phải đối tượng mà con trỏ đang trỏ tới: phương thức của lớp mà con trỏ có kiểu được gọi chứ không phải phương thức của đối tượng mà con trỏ đang trỏ tới được gọi.

Polymorphism

C++

```
1 class Animal {
2 public:
3     void show() {
4         cout << "This is an animal\n";
5     }
6 };
7
8 class Dog: public Animal {
9 public:
10    void show() {
11        cout << "This is a dog\n";
12    }
13 };
14
15 int main() {
16     Dog dog;    // initialize Dog object
17     Animal* animal = &dog; // set pointer Animal to dog
18     animal->show();    // This is an animal -> not the answer we want !!!
19
20     return 0;
21 }
```

❑ Chương trình gọi đến phương thức show() của lớp Animal mà không gọi đến phương thức show() của lớp Dog !!!

➡ Cách giải quyết: Để chương trình gọi đến phương thức show() của lớp Dog ta sẽ sử dụng virtual function (hàm ảo).

Polymorphism

C++

A graphic consisting of two overlapping circles. The left circle is blue and the right circle is light gray. The text "Virtual Function" is written in blue, with "Virtual" on the top line and "Function" on the bottom line, centered over the intersection of the two circles.

Virtual Function

- ❑ Khi khai báo virtual function (hàm ảo) với từ khóa virtual nghĩa là hàm này sẽ được gọi theo loại đối tượng được trỏ (hoặc tham chiếu), chứ không phải theo loại của con trỏ (hoặc tham chiếu).
- ❑ Áp dụng cơ chế liên kết động để gọi đúng hàm mà con trỏ đang trỏ đến: Khi nhận thấy có khai báo virtual trong lớp cơ sở, compiler sẽ thêm vào mỗi đối tượng của lớp cơ sở và các lớp dẫn xuất của nó một con trỏ chỉ đến bảng phương thức ảo (virtual function table).

Polymorphism

C++

Virtual
Function

```
1 class Animal {  
2 public:  
3     // virtual functions  
4     virtual void show() {  
5         cout << "This is an animal\n";  
6     }  
7 };
```

Cho từ khóa virtual
trước hàm show()
của lớp Animal

```
1 int main() {  
2     Dog dog;    // initialize Dog object  
3     Animal* animal = &dog; // set pointer Animal to dog  
4     animal->show(); // This is a dog -> successfully  
5  
6     return 0;  
7 }
```

Đã in ra đúng hàm
show() của lớp Dog.

Polymorphism

C++



Pure Virtual Function

- ❑ Pure virtual function (hàm thuần ảo): chỉ dùng hàm ảo tại lớp cơ sở để khai báo, chứ không có cài đặt bất kì câu lệnh nào bên trong hàm đó.
- ❑ Cú pháp: `virtual functionName() = 0;`

```
1  class Animal {  
2  public:  
3      // pure virtual functions  
4      virtual void speak() = 0;  
5  };
```

Polymorphism

C++

A graphic consisting of a light gray circle with a blue crescent shape on its left side. The text "Abstract Class" is written in blue inside the circle.

Abstract
Class

- ❑ Abstract Class (Lớp trừu tượng): 1 lớp được xem là lớp trừu tượng nếu có ít nhất 1 hàm thuần ảo bên trong lớp đó.
- ❑ Không thể tạo đối tượng từ lớp trừu tượng, nhưng có thể áp dụng con trỏ hoặc tham chiếu của lớp đó.
- ❑ Khi 1 lớp con kế thừa lớp trừu tượng thì bắt buộc phải cài đặt lại tất cả các hàm thuần ảo, nếu không sẽ thành lớp trừu tượng.

Polymorphism

C++



Abstract
Class

```
1 // Abstract class
2 class Animal {
3 public:
4     // pure virtual functions
5     virtual void speak() = 0;
6 };
7
8 class Dog: public Animal {
9 public:
10     void speak() {
11         cout << "Woof Woof\n";
12     }
13 };
14
15 class Cat : public Animal {};
16
17 int main() {
18     // Animal animal;           // error: Abstract Class
19     Dog dog;                   // success: Not an Abstract Class
20     // Cat cat;                 // error: Abstract Class
21     Animal* animal = &dog;     // success: use pointer class Animal
22
23     return 0;
24 }
```

Polymorphism

C++

Virtual Destructor

```
1 class Animal {
2 public:
3     ~Animal() {
4         cout << "Animal destructor\n";
5     }
6 };
7
8 class Dog : public Animal {
9 public:
10     virtual ~Dog() {
11         cout << "Dog destructor\n";
12     }
13 };
14
15 int main() {
16     Animal* animal = new Dog;
17     delete animal; // Animal destructor !!!
18
19     return 0;
20 }
```

ĐẶT VẤN ĐỀ:
animal là biến đối tượng kiểu Animal nhưng đang giữ đối tượng kiểu Dog. Tuy nhiên khi delete animal thì chỉ gọi đến destructor lớp Animal mà không gọi đến destructor của lớp Dog !!!

Polymorphism

C++

Virtual Destructor

- ❑ Cách giải quyết: dùng virtual destructor (hàm hủy ảo) cho phương thức Animal destructor. Hàm hủy ảo sẽ chuyển lời gọi hàm xuống destructor của lớp kế thừa.
- ❑ Cú pháp: Khai báo từ khóa virtual trước hàm hủy của lớp cha.

Thành công



```
1 virtual ~Animal() {  
2     cout << "Animal destructor\n";  
3 }
```

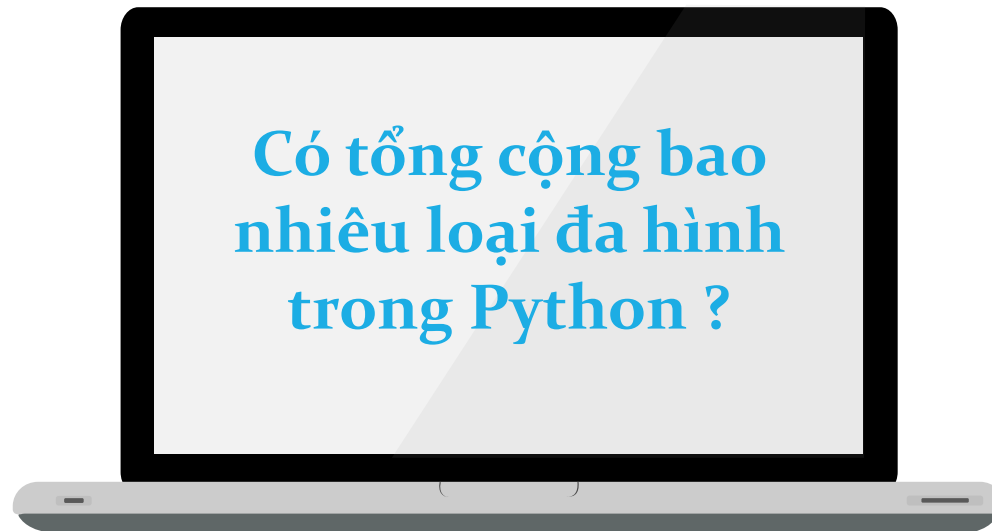


```
1 int main() {  
2     Animal* animal = new Dog;  
3     delete animal; // Dog destructor  
4                     // Animal destructor  
5     return 0;  
6 }
```

Polymorphism

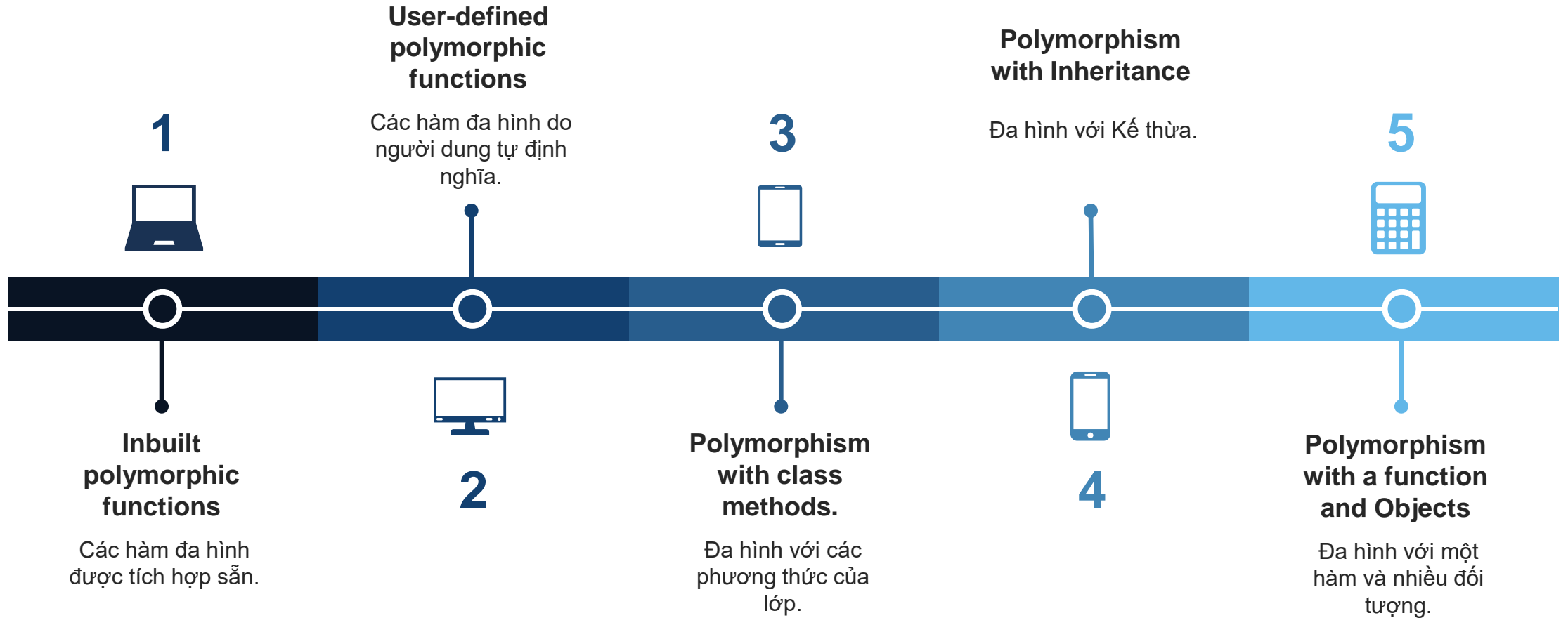
Python

- ❑ Không như các ngôn ngữ khác, trong Python không có khái niệm con trỏ. Thay vào đó, các đối tượng đều được sử dụng thông qua tham chiếu.
- ❑ Nên không như C++, Polymorphism (đa hình) trong Python không tập trung vào vấn đề con trỏ mà thay vào đó sẽ được chia ra làm các loại khác nhau.



Polymorphism

Python



Polymorphism

Python

- ❑ Inbuilt polymorphic functions (Hàm đa hình được tích hợp sẵn): những hàm có sẵn trong bộ thư viện của Python. Ví dụ: hàm len(), ...



```
1 # len for string
2 print(len("Duy Onix")) # 8
3
4 # len for tuple
5 print(len((1, 2, 3))) # 3
```


Polymorphism

Python

- ❑ User-defined polymorphic functions (Hàm đa hình do người dùng tự định nghĩa): thường là những hàm có giá trị tham số đầu vào mặc định.



```
1 def plus(a, b, c=0):      # c = 0: default value
2     return a + b + c
3
4 print(plus(1, 2))         # 3
5 print(plus(1, 2, 3))     # 6
```

Polymorphism

Python

- ❑ Polymorphism with class methods (Đa hình với các phương thức của lớp): Python có thể sử dụng phương thức của 2 kiểu class khác nhau theo cùng 1 cách.



```
1 class Dog:
2     def speak(self):
3         print("Woof Woof")
4
5 class Cat:
6     def speak(self):
7         print("Meow Meow")
8
9 dog = Dog()
10 cat = Cat()
11
12 # loop through each instance of Dog and Cat
13 for animal in (dog, cat):
14     animal.speak()
15     # Woof Woof
16     # Meow Meow
```

Polymorphism

Python

- ❑ Polymorphism with Inheritance (Đa hình với kế thừa): Trong Python, đa hình cho phép ta có thể định nghĩa lại phương thức trong lớp con mà có cùng tên với các phương thức trong lớp cha.

➡ Method Overriding (ghi đè phương thức)



```
1 class Animal:
2     def description(self):
3         print("This is an animal")
4
5 class Dog(Animal):
6     def description(self):
7         print("This is a dog")
8
9 class Cat(Animal):
10    def description(self):
11        print("This is a cat")
12
13 animal = Animal()
14 dog = Dog()
15 cat = Cat()
16
17 animal.description()    # This is an animal
18 dog.description()      # This is a dog
19 cat.description()      # This is a cat
```

Polymorphism

Python

- ❑ Polymorphism with a Function and Objects (Đa hình với 1 hàm và nhiều đối tượng): Chúng ta có thể tạo ra một hàm mà có khả năng nhận vào bất kỳ đối tượng thuộc kiểu class nào, nhằm thực hiện đa hình.



```
1 class Dog:
2     def speak(self):
3         print("Woof Woof")
4
5 class Cat:
6     def speak(self):
7         print("Meow Meow")
8
9 # function that can pass any type of params
10 def polymorphism(obj):
11     obj.speak()
12
13 dog = Dog()
14 cat = Cat()
15
16 polymorphism(dog) # Woof Woof
17 polymorphism(cat) # Meow Meow
```

The background is a deep blue gradient with various geometric elements. There are several light blue circles of different sizes, some solid and some outlined. Diagonal lines in a lighter blue shade are scattered across the frame. Large, soft-edged, light blue shapes, resembling stylized leaves or abstract forms, are layered in the background. In the center, a dark blue rectangle with a thin white border contains the word 'SUMMARY' in white, bold, serif capital letters.

SUMMARY

SUMMARY

Tổng kết lại những gì đã nêu trên, ta đã tìm hiểu và so sánh được 5 nội dung cơ bản về OOP của C++ và Python. Và ta rút ra được những kết luận chung sau đây:

- ❑ Nhìn chung, cả 2 ngôn ngữ C++ và Python đều có thể lập trình hướng đối tượng, đảm bảo đầy đủ yêu cầu và tính chất của OOP. Tuy nhiên, cách tổ chức và triển khai OOP cho 2 ngôn ngữ có sự khác biệt về cú pháp, ngữ nghĩa và cách sử dụng khác nhau cho mỗi trường hợp.
- ❑ C++ và Python là 2 ngôn ngữ nổi tiếng trên toàn thế giới, người lập trình có thể cân nhắc những yếu tố trên để chọn ra ngôn ngữ phù hợp cho bản thân để lập trình OOP.

References

Các nguồn tài liệu tham khảo

- ❑ <https://realpython.com/python3-object-oriented-programming/>
- ❑ <https://viblo.asia/p/oop-voi-python-E375zQGblGW>
- ❑ <https://quantrimang.com/lap-trinh-huong-doi-tuong-trong-python-160230/>
- ❑ <https://www.geeksforgeeks.org/python-oops-concepts/>
- ❑ <https://www.geeksforgeeks.org/class-method-vs-static-method-python/>
- ❑ <https://www.geeksforgeeks.org/getter-and-setter-in-python/>
- ❑ <https://www.geeksforgeeks.org/inheritance-in-python/>
- ❑ <https://nguyenvanhieu.vn/tinh-ke-thua-trong-c/>
- ❑ <https://www.geeksforgeeks.org/polymorphism-in-python/>
- ❑ <https://cafedev.vn/tu-hoc-python-da-hinh-trong-python/>
- ❑ <https://codelearn.io/sharing/tinh-da-hinh-trong-oop/>
- ❑ <https://www.geeksforgeeks.org/getter-and-setter-in-python/>
- ❑ Sách “Lập trình hướng đối tượng” và tài liệu bài giảng của thầy Hồ Tuấn Thanh.

Link Video

Link drive video thuyết trình

<https://drive.google.com/file/d/1Zksf8htriTo2H73sqGALV7iSp4IKqDlJ/view?usp=sharing>

The background is a gradient of blue, with darker shades on the left and lighter shades on the right. There are several abstract geometric shapes and lines in various shades of blue, including circles, lines, and rounded rectangles, scattered across the left side of the image.

THANK YOU

Chân thành cảm ơn thầy cô và các bạn đã lắng
nghe bài thuyết trình của mình.