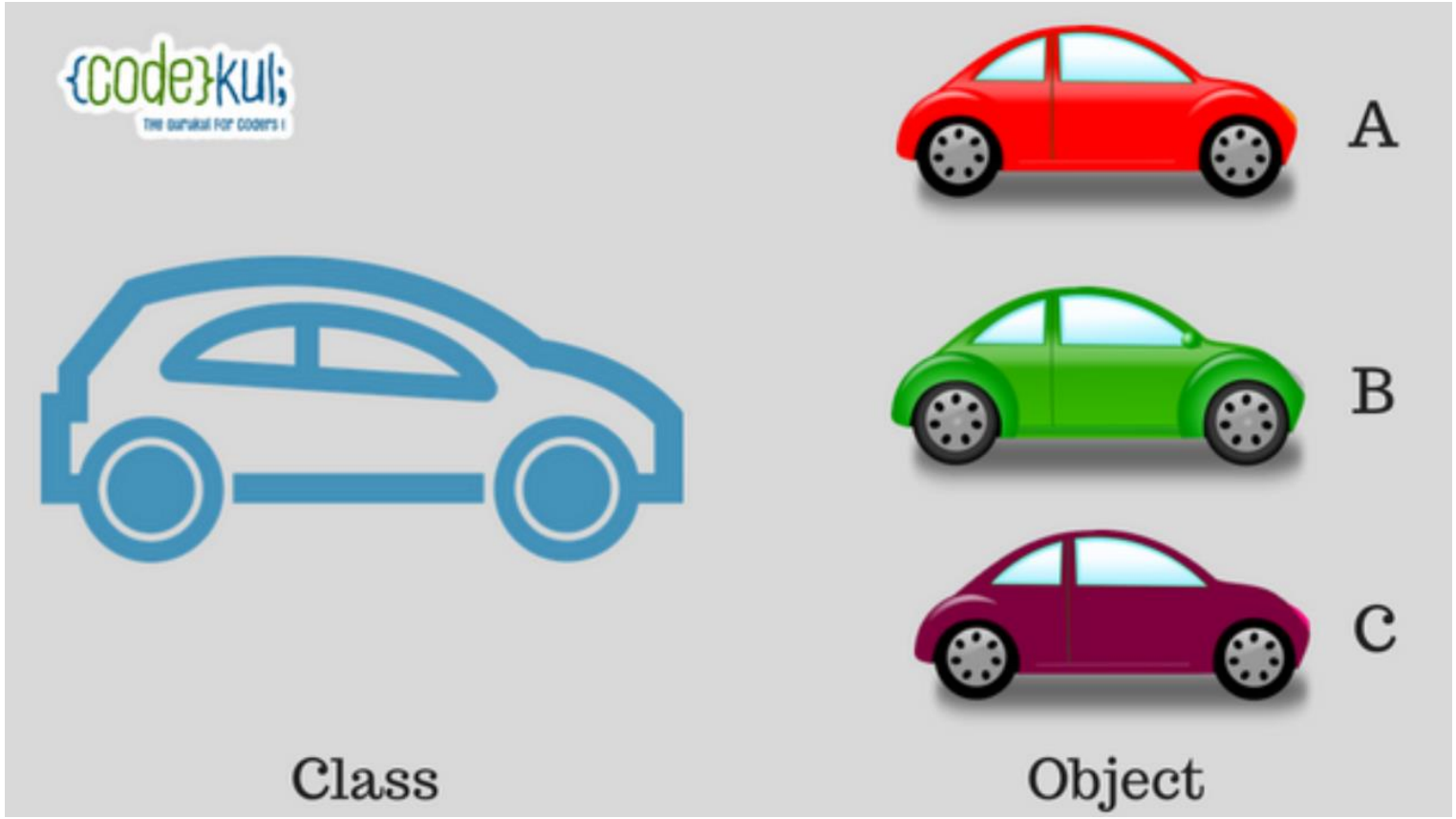
Level type in OOP

Recap: Class và Object

Trong Python:

- Class là bản mẫu của xe hơi
- Object A,B,C là thành phẩm từ bản mẫu
- Object A,B,C có tất cả các thuộc tính đã define từ class



Level type in OOP

- **Class level attribute** : Là attribute của class đó.
- **Instance level attribute**: Là attribute của từng object tạo ra từ class đó.

```python
class Person:
    bien_class_level = "class-level"
    def __init__(self, bien_instance_level):
        self.bien_instance_level = bien_instance_level

person_1 = Person("instance_1")
person_2 = Person("instance_2")

print(person_1.bien_class_level,",", person_1.bien_instance_level)
print(person_2.bien_class_level,",", person_2.bien_instance_level)
```

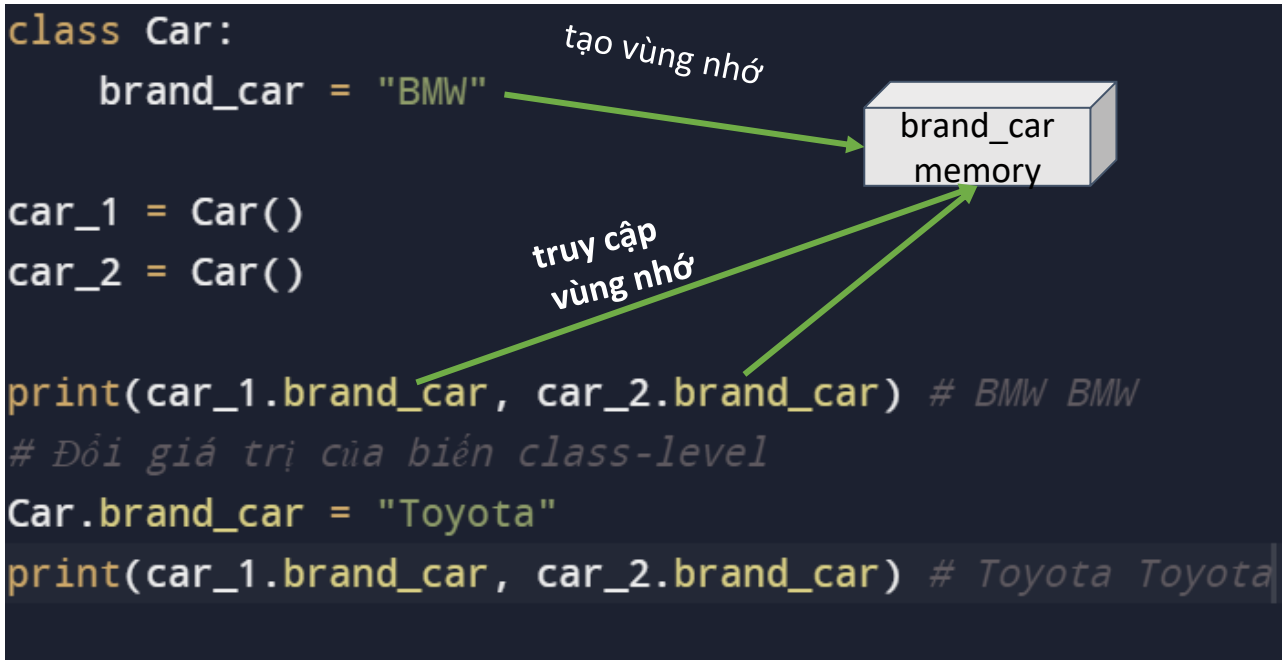
```
class-level , instance_1
class-level , instance_2
>
```

Thuộc tính: Class level

- Khi khởi tạo **class-level attribute** thì tất cả các object của class đó đều share có cùng một biến.
- Khi biến class-level **bị thay đổi**, thì biến đó ở **tất cả các object** đều bị thay đổi.

Cú pháp:

<tên class>.<tên biến>

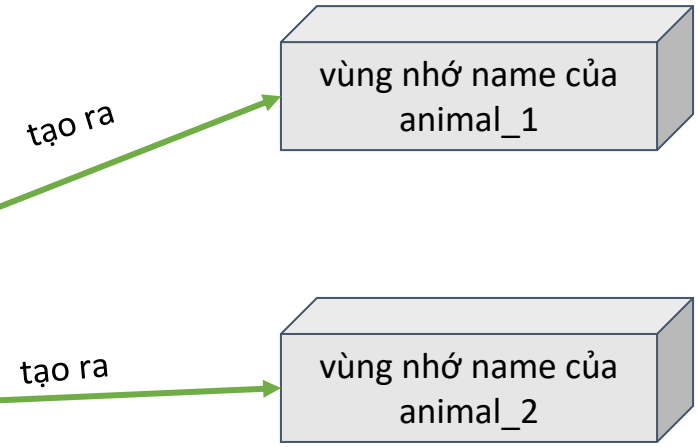


Thuộc tính: Instance level

- Là **attribute riêng biệt** của từng object chứ nó **không sharing** attribute đó cho nhau.
- Khi khởi tạo instance attribute, class sẽ tạo ra **một vùng memory** của biến đó cho từng object.

```
class Animal:
    def __init__(self):
        self.name="animal"

    def set_name(self,name):
        self.name = name
```

```
animal_1 = Animal()
animal_1.set_name("animal_1")

animal_2 = Animal()
animal_2.set_name("animal_2")
```



Cú pháp

<tên object>.<tên biến>

Hàm : Class level (class method)

- Hàm class level dung để giao tiếp với thuộc tính class level
- Cách khai báo hàm:

```
@classmethod
def tên_hàm(cls):
cls.tên_thuộc_tính_class_level
```

Trong ví dụ này:

- biến “brand_car” là class-level được set là private
- Bên ngoài class không thể truy cập trực tiếp
- Phải thông qua hàm class level để truy cập và thay đổi

```
class Car:
    __brand_car = "BMW"
    @classmethod
    def get_brand_car(cls):
        return cls.__brand_car
    @classmethod
    def set_brand_car(cls, brand):
        cls.__brand_car = brand

car_1 = Car()
car_2 = Car()

print(car_1.get_brand_car(), car_2.get_brand_car()) # BMW BMW
# Đổi giá trị của biến class-level
Car.set_brand_car("Toyota")
print(car_1.get_brand_car(), car_2.get_brand_car()) # Toyota
    Toyota
```

Hàm : Instance level (instance method)

- Hàm instance level dung để giao tiếp với thuộc tính instance level và class level
- Cách khai báo hàm:

```
def tên_hàm(self):
    self.tên_thuộc_tính_instance_level
```

get_car_color: là hàm instance level, dùng để giao tiếp với:

- instance level: self.__color
- class level: Car.__brand_car

```
class Car:
    __brand_car = "BMW"
    def __init__(self, color):
        self.__color = color
    @classmethod
    def get_brand_car(cls):
        return cls.__brand_car
    @classmethod
    def set_brand_car(cls, brand):
        cls.__brand_car = brand

    def get_car_color(self):
        return self.__color, Car.__brand_car

car_1 = Car("red")
car_2 = Car("blue")

print(car_1.get_brand_car(), car_2.get_brand_car()) # BMW BMW
# Đổi giá trị của biến class-level
Car.set_brand_car("Toyota")
print(car_1.get_brand_car(), car_2.get_brand_car()) # Toyota Toyota
print(car_1.get_car_color(), car_2.get_car_color()) # red BMW blue BMW
```

Ví dụ về level type

- total_people: biến class level
- name, age : biến instance level
- say_hello(): hàm instance level
- get_total_people: hàm class level

people1.say_hello(): là gọi hàm instance level. Thì trong hàm say_hello có thể giao tiếp với age, name và total_people

People.get_total_people(): là gọi hàm class level, chỉ giao tiếp được với biến total_people.

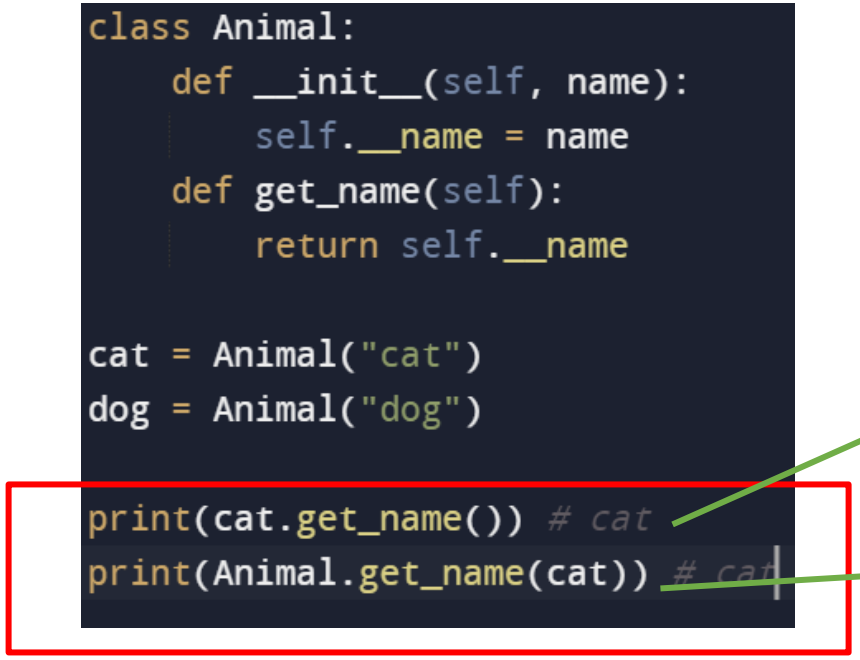
```python
class Person:
    # Biến lớp (class variable)
    total_people = 0

    def __init__(self, name, age):
        # Biến của đối tượng (instance variable)
        self.name = name
        self.age = age
        Person.total_people += 1

    # Phương thức của đối tượng (instance method)
    def say_hello(self):
        print(f"Xin chào, tôi là {self.name} và tôi {self.age} tuổi.")

    # Phương thức của lớp (class method)
    @classmethod
    def get_total_people(cls):
        return cls.total_people

# Tạo các đối tượng Person
person1 = Person("Alice", 25)
person2 = Person("Bob", 30)

# Gọi phương thức của đối tượng
person1.say_hello()
person2.say_hello()

# Gọi phương thức của lớp để lấy tổng số người đã tạo
total_people = Person.get_total_people()
print(f"Tổng số người: {total_people}")
```

Level type in OOP: Kết luận

Class level	Instance level
Hàm và thuộc tính thuộc sở hữu của **class** đó	Hàm và thuộc tính thuộc sở hữu của **object**
Chỉ có **hàm class-level mới giao tiếp** được với **thuộc tính class-level (biến/hằng số)**	Chỉ có **hàm instance-level** giao tiếp được với **thuộc tính** instance-level và class-level (biến/hằng số)
Define những **cái chung của toàn thể các object**	Các object mang giá trị riêng
Chỉ có một biến duy nhất **(một vùng nhớ)** cho tất cả object thuộc class đó	**Mỗi object sẽ có vùng nhớ riêng** cho biến đó để lưu giá trị

Tham số “self” (nâng cao)

- Tham số self có ở các hàm instance-level để làm gì?



```
class Animal:
    def __init__(self, name):
        self.__name = name
    def get_name(self):
        return self.__name

cat = Animal("cat")
dog = Animal("dog")

print(cat.get_name()) # cat
print(Animal.get_name(cat)) # cat
```

- Cả hai dòng lệnh dưới **đều chạy được và cho ra cùng kết quả**
- **Cách thứ nhất** thì ta **không cần phải** truyền object vào, mà thay vào đó sẽ dùng: **<tên object>.<tên hàm>** thay cho việc truyền object cat vào tham số self
- **Cách thứ hai** thì ta truyền **object cat vào hàm get_name** dưới tên là **self**

-> Trả lời: tham số self **dùng để truyền vào object cat** vào **hàm get_name**

Tương tự như vậy, giải thích tham số “cls” ở class-level thử xem

Class Relation: Association, Aggregation, Inheritance

Overview Concept

- Giả sử: chúng ta có **rất nhiều class**, và các class này có những **quan hệ với nhau** như sau:
  - Hàm của class này có **dùng object** của class kia làm **input parameter**.
  - Class này **có** một/nhiều **object** của class kia.
  - Class này **kế thừa (có tất cả)** các tính chất của class kia và **mở rộng thêm**.
- Ví dụ:
  - Class **Library** có hàm **get_info_book(book)** sẽ dung object của class **Book** để làm **input parameter**
  - Class **Classroom** có nhiều object (**Student)**
  - Class **Student** thừa hưởng hết các thuộc tính và hàm của class **Person** và dùng để **mở rộng them cho class Person**

Association relation (uses a)

- Class **Library** dùng object **book** của class **Book** trong hàm **get_book** để gọi hàm **info()**

```
class Library:
    def get_book(self, book):
        print("Get book: ",book.info())

class Book:
    def __init__(self,name,author):
        self.name= name
        self.author = author
    def info(self):
        return self.name + " by " + self.author

harry_potter = Book("Harry Potter","JK.Rowling")
nauy_forest = Book("Nauy Forest","Murakami Haruki")

fpt_library = Library()
fpt_library.get_book(harry_potter) # Get book:  Harry Potter by JK.Rowling
fpt_library.get_book(nauy_forest) # Get book:  Nauy Forest by Murakami Haruki
```

Object **book** chỉ dùng trong phạm vi của hàm **get_book**

object **book** không thể được gọi ở ngoài hàm get_book nữa

Parameter **data type** của hàm class-level: class

Aggregation relation (has a)

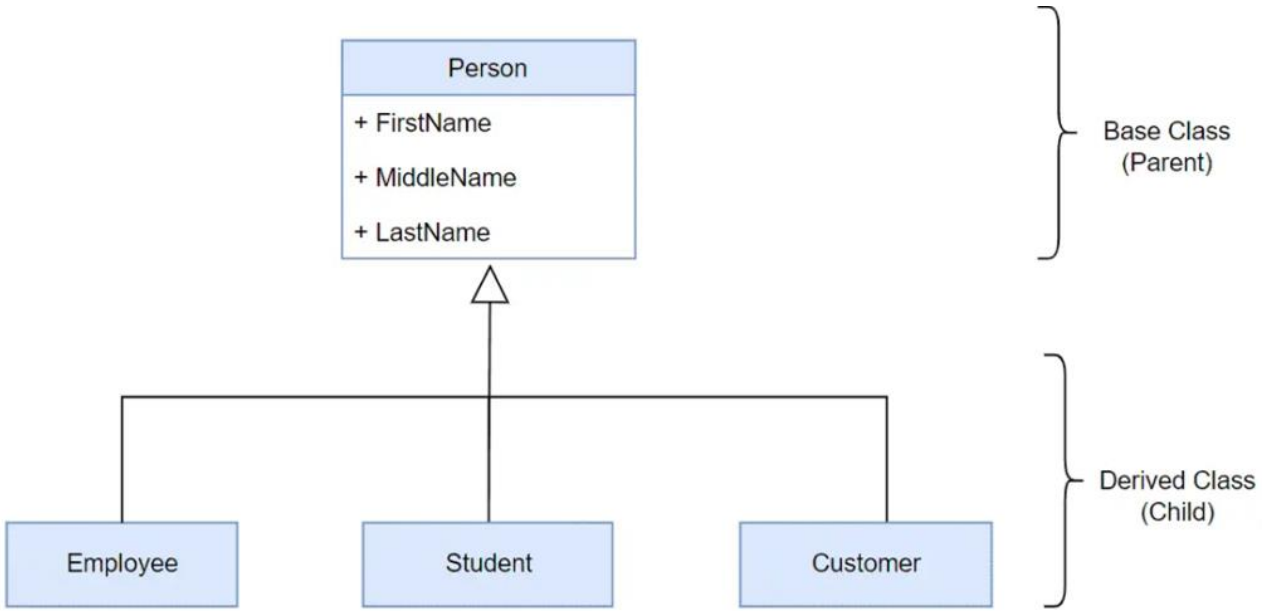
- Lớp **Classroom** có nhiều object của lớp **Student**

```
class Student:
    def __init__(self, name, gender):
        self.name = name
        self.gender = gender
    def info(self):
        return self.name + " is "+self.gender

class Classroom:
    def __init__(self, classroom_name, students):
        self.students = students
        self.classroom_name = classroom_name
    def get_classroom_info(self):
        output = "Class "+self.classroom_name+" has: "
        for student in self.students:
            output += f"\n {student.info()}"
        return output

chau_student = Student("Chau","F")
khoa_student = Student("Khoa","M")

oop_class = Classroom("OOP",[chau_student, khoa_student])
print(oop_class.get_classroom_info())
```

Object **students** có thể dùng ở bất cứ đâu trong class **"Classroom"**
Attitude **data type** của hàm instance-level là list of classes

Inheritance relation (is a)

- Các lớp Employee, Student, Customer sẽ kế thừa lại các thuộc tính: FirstName, MiddleName, LastName của lớp Person.



Ví dụ về các Class relation

Aggregation (has a)

Inheritance(is a)

Association (use a)

```python
class Student(Person):
    def __init__(self, name, grade):
        super().__init__(name)
        self.grade = grade

    def introduce(self):
        return f"Hi, I'm {self.name}, and I'm in grade {self.grade}."

# Mối quan hệ Inheritance và Aggregation
class Classroom:
    def __init__(self, number, teacher):
        self.number = number
        self.teacher = teacher
        self.students = []

    def add_student(self, student):
        self.students.append(student)

    def list_students(self):
        return [student.name for student in self.students]
```

```python
# Mối quan hệ Association
class School:
    def __init__(self, name):
        self.name = name
        self.teachers = []
        self.students = []

    def add_teacher(self, teacher):
        self.teachers.append(teacher)

    def add_student(self, student):
        self.students.append(student)

class Person:
    def __init__(self, name):
        self.name = name

    def introduce(self):
        pass

class Teacher(Person):
    def __init__(self, name, subject):
        super().__init__(name)
        self.subject = subject

    def introduce(self):
        return f"Hi, I'm {self.name}, and I teach {self.subject}."
```

Class Relation Level

Association	Aggregation	Inheritance
Define khi cần dung object trong phạm vi một hàm	Define khi cần dung object trong phạm vi toàn class	Dùng khi muốn mở rộng tính năng của object
Tính bảo mật cao nhất, khi object sẽ không thể bị gọi bên ngoài hàm đó	Tính bảo mật trung bình, chỉ được dung trong class nhưng không thể chỉnh sửa object đó	Toàn quyền chỉnh sửa và sử dụng object

Cần xác định rõ rang mối quan hệ giữa hai class với nhau để đưa ra cách code phù hợp

__call__