LECTURER: TAI LE QUY

OBJECT ORIENTED AND FUNCTIONAL PROGRAMMING WITH PYTHON

Thanks Prof. Dr. Max Pumperla for his contribution

Who am I?

- Name: Tai Le Quy
- PhD candidate at L3S Research Center – Leibniz University Hannover
  - Topic: Fairness-aware machine learning in educational data mining
  - Project: LernMINT (lernmint.org)
- MSc in Information Technology at National University of Vietnam
- Profile: tailequy.github.io
- Email: tai.le-quy@iu.org
- Additional materials: https://github.com/tailequy/IU-OOP-PythonProgramming

Who are you?

- Name
- Employer
- Position/responsibilities
- Fun Fact
- Previous knowledge? Expectations?

TOPIC OUTLINE

Object oriented programming	1 +2
Functional programming	3
Projects and testing in Python	4
Working with Database in Python	5
Documenting a project	6

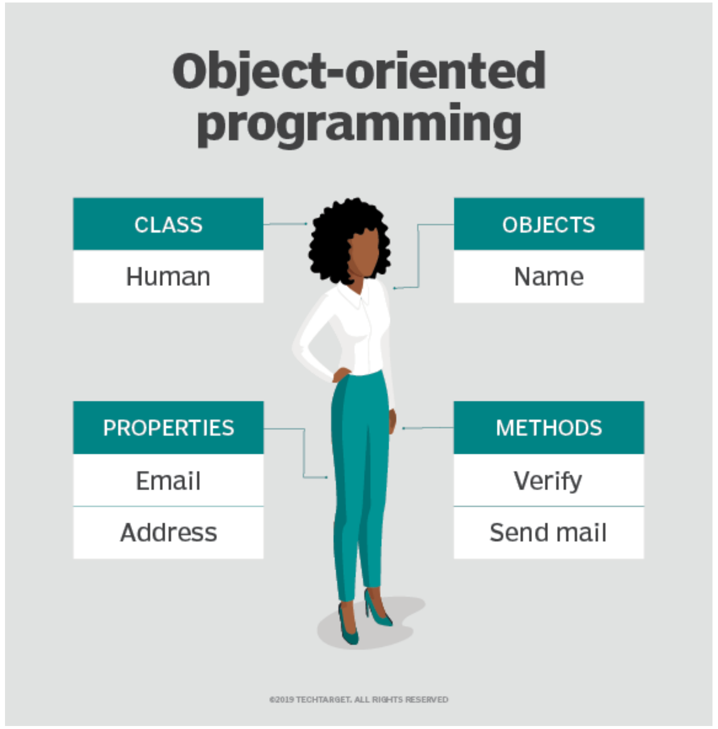
UNIT 1

OBJECT ORIENTED PROGRAMMING

STUDY GOALS

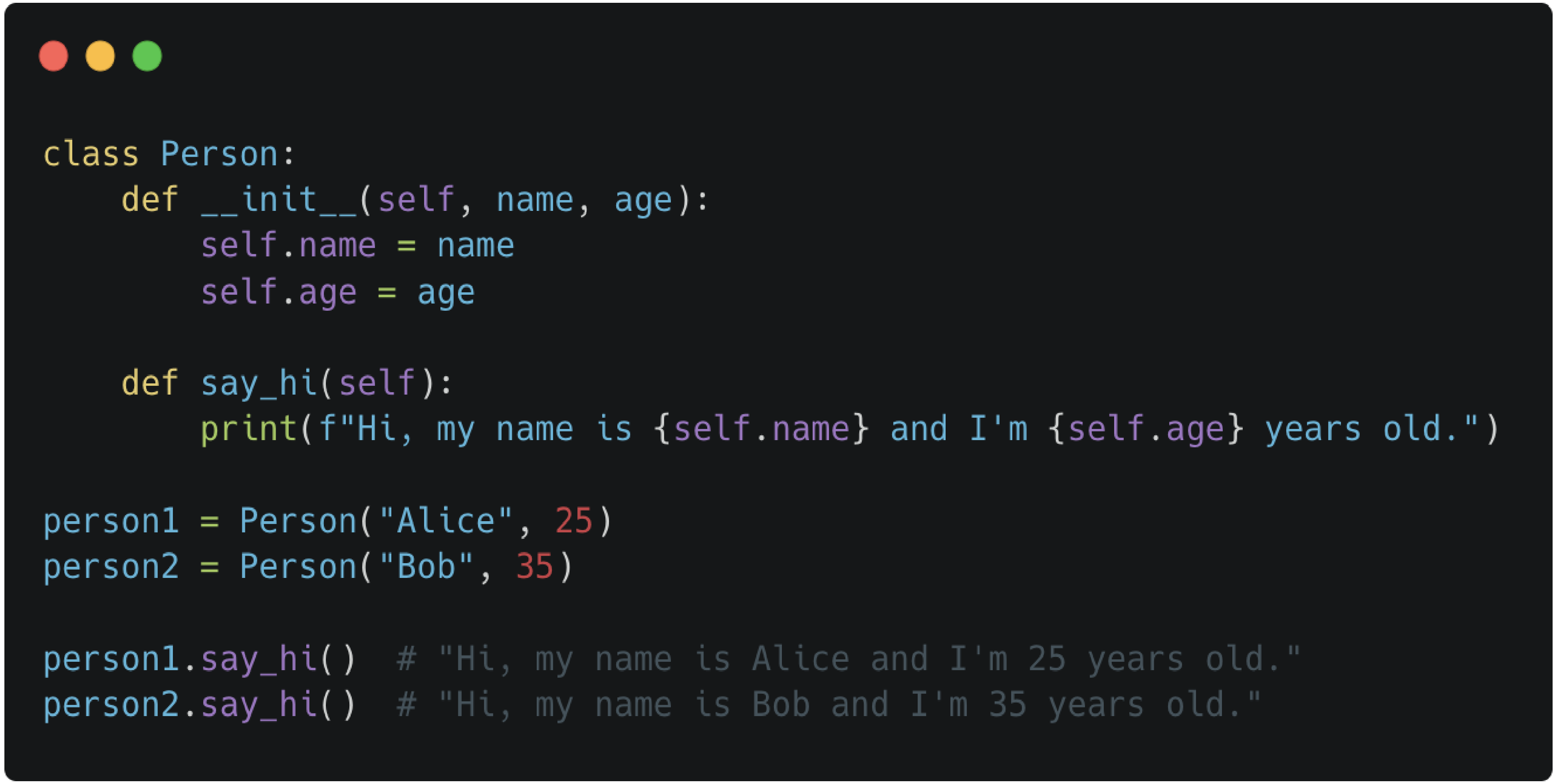
- Introduction to Object-Oriented Programming (OOP) in Python
- Classes and Objects in Python
- Attributes and Methods in Python Classes
- Inheritance in Python

- OOP differs from procedural programming
  - **Classes and objects central**
- Pros: modularity, abstraction, maintenance, reusability, etc.
- Cons: Complexity, overuse, performance
- Classes are blueprints for objects
- Objects are instances of classes
- Classes have attributes and methods
- Attributes define “state”
  - **The data your objects contain**
  - **Access or set state using “self”**
- Methods define what your objects do



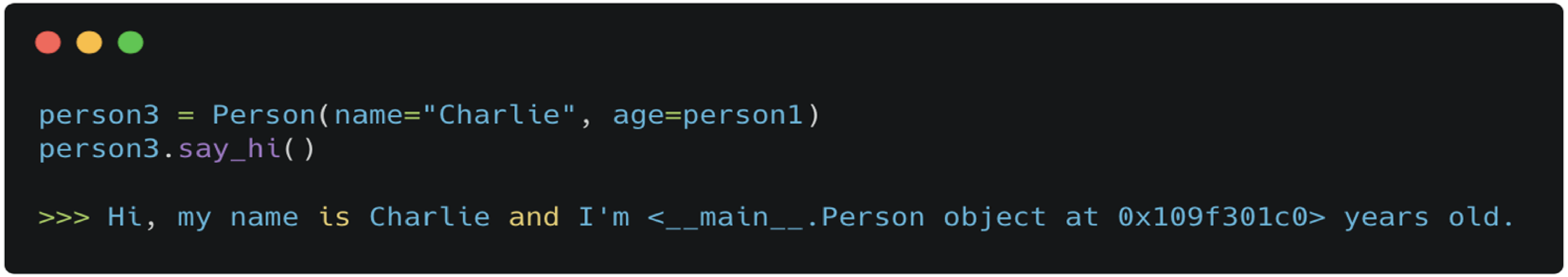
Source of the image: https://www.techtarget.com/searchapparchitecture/definition/object-oriented-programming-OOP

PYTHON OOP BASIC EXAMPLE



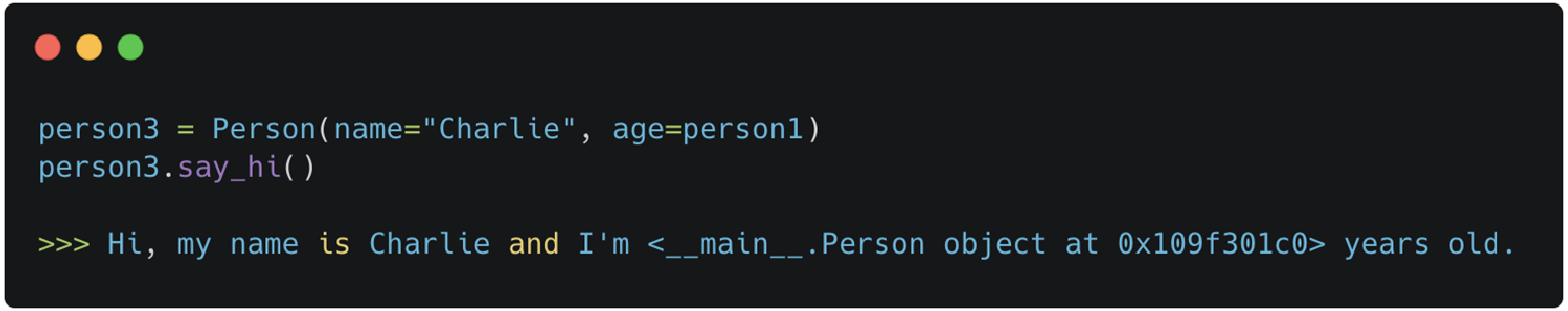
```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def say_hi(self):
        print(f"Hi, my name is {self.name} and I'm {self.age} years old.")

person1 = Person("Alice", 25)
person2 = Person("Bob", 35)

person1.say_hi()  # "Hi, my name is Alice and I'm 25 years old."
person2.say_hi()  # "Hi, my name is Bob and I'm 35 years old."
```

WHAT CAN GO WRONG?



```
person3 = Person(name="Charlie", age=person1)
person3.say_hi()

>>> Hi, my name is Charlie and I'm <__main__.Person object at 0x109f301c0> years old.
```

IT WORKS, BUT BE CAREFUL



```
person3 = Person(name="Charlie", age=person1)
person3.say_hi()

>>> Hi, my name is Charlie and I'm <__main__.Person object at 0x109f301c0> years old.
```

- “class” keyword to define the class header
- Special method “__init__”
  - **Constructor**
  - **Takes “self” as first argument**
  - **Define as many attributes as you want**
  - **Can define default values**
- Objects are created using this constructor
- Classes can have methods
  - **usually start with “self”**
- Different types of attributes in Python

- Class Attributes
- Class attributes are the variables defined directly in the class that are shared by all objects of the class. Class attributes can be accessed using the class name as well as using the objects.

Example:

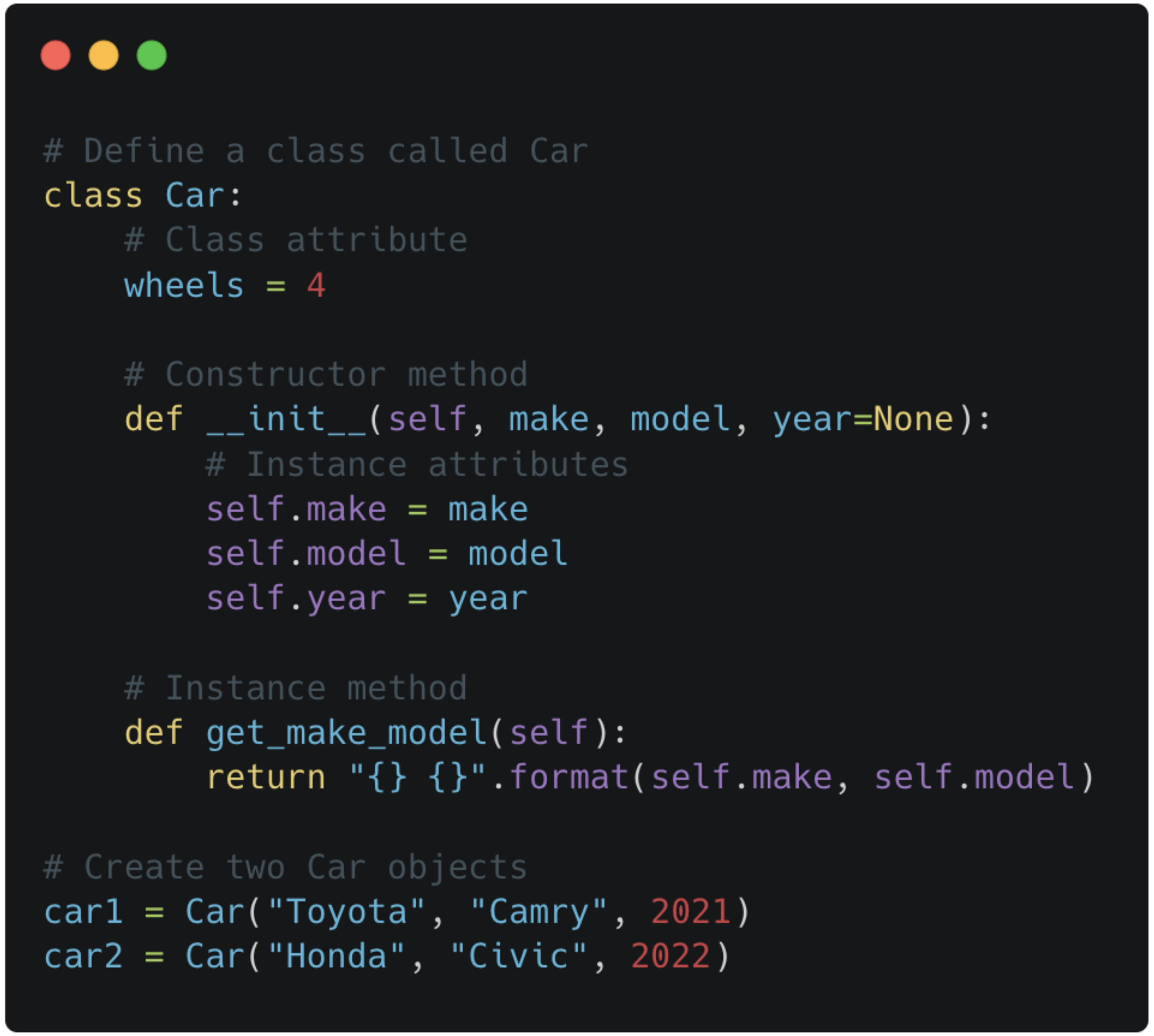
```
class Student:
       schoolName = 'XYZ School’

print(Student.schoolName) #'XYZ School'
Student.schoolName = 'ABC School'
print(Student.schoolName) #'ABC School’
std = Student()
print(std.schoolName) #'ABC School'
std.schoolName = 'Super School'
print(std.schoolName) #'Super School'
print(Student.schoolName) #'ABC School
```

- Instance Attributes
- Instance attributes are attributes or properties attached to an instance of a class. Instance attributes are defined in the constructor.

Example:

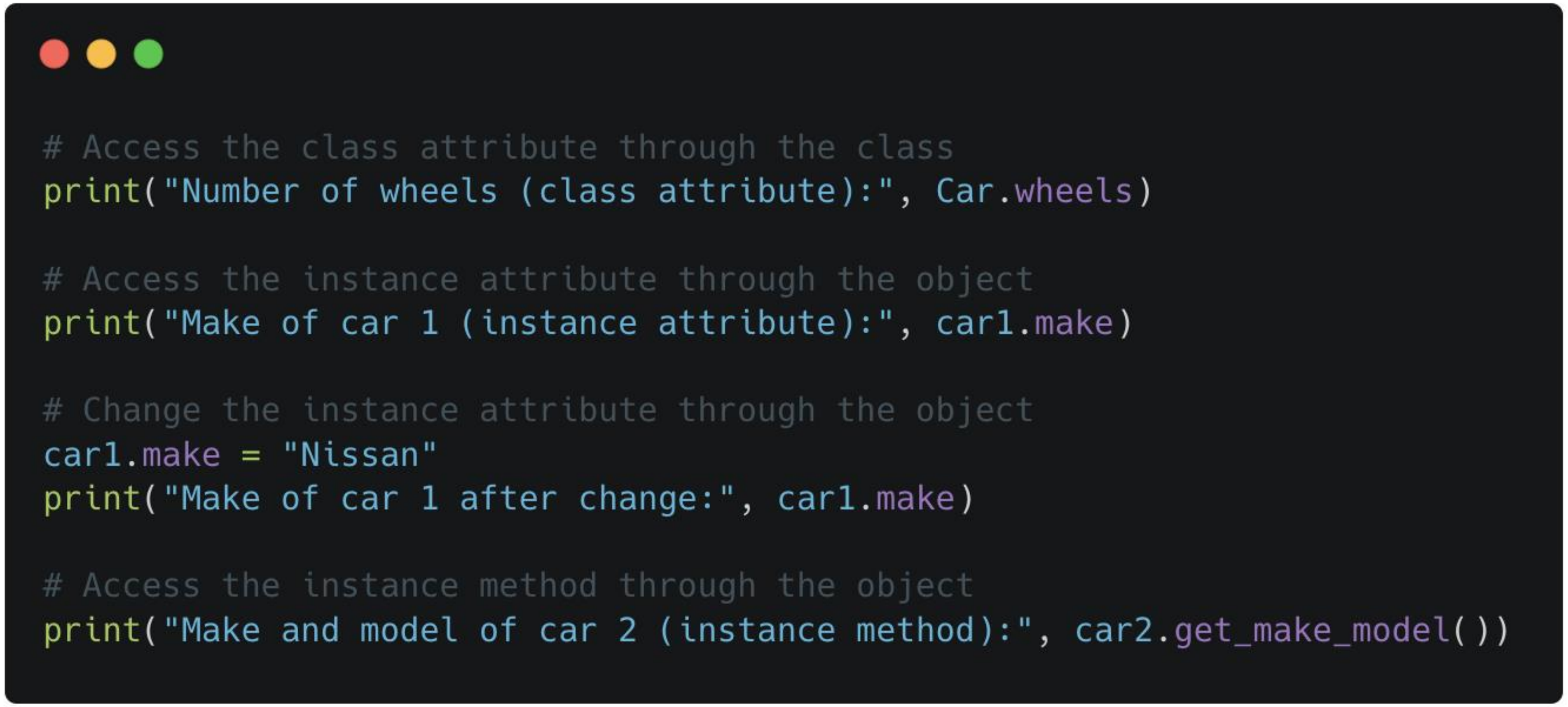
```
class Student:
   schoolName = 'XYZ School' # class attribute
   def __init__(self): # constructor
       self.name = '' # instance attribute
       self.age = 0 # instance attribute

std = Student('Bill',25)
#passing values to constructor
print(std.name)
print(std.age)
```

CLASS & INSTANCE ATTRIBUTES



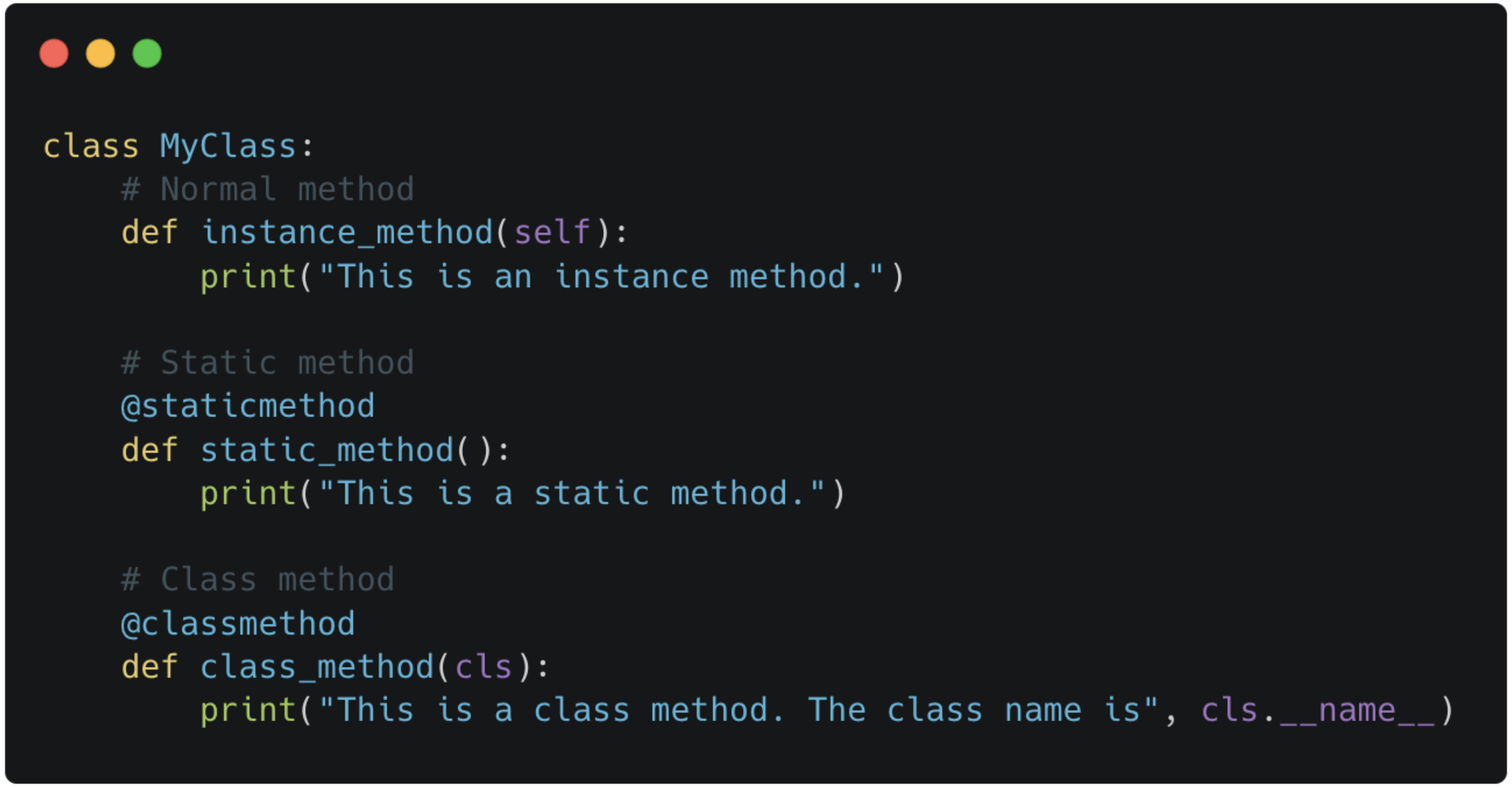
```python
# Define a class called Car
class Car:
    # Class attribute
    wheels = 4

    # Constructor method
    def __init__(self, make, model, year=None):
        # Instance attributes
        self.make = make
        self.model = model
        self.year = year

    # Instance method
    def get_make_model(self):
        return "{} {}".format(self.make, self.model)

# Create two Car objects
car1 = Car("Toyota", "Camry", 2021)
car2 = Car("Honda", "Civic", 2022)
```

CLASS & INSTANCE ATTRIBUTES



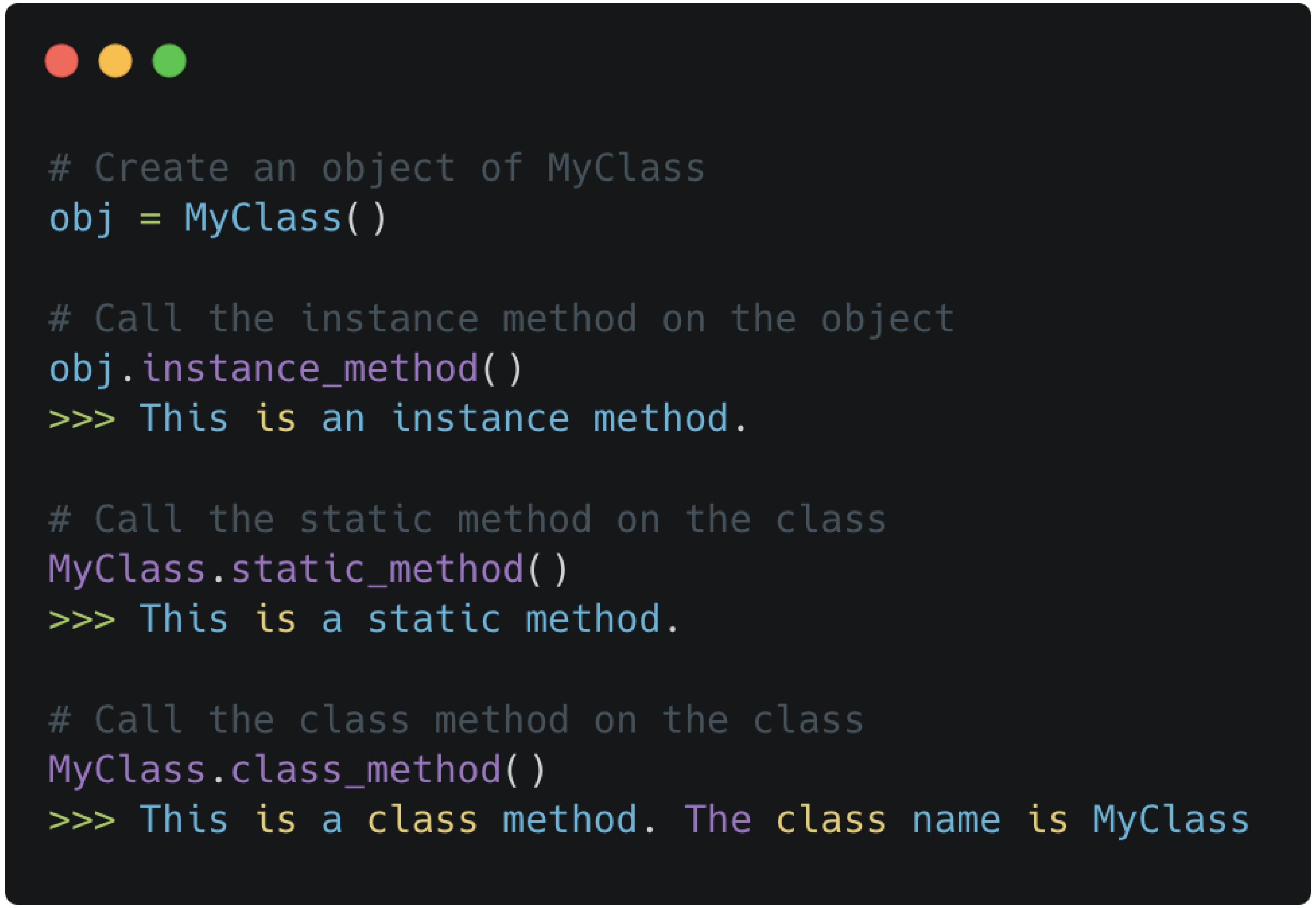
```
# Access the class attribute through the class
print("Number of wheels (class attribute):", Car.wheels)

# Access the instance attribute through the object
print("Make of car 1 (instance attribute):", car1.make)

# Change the instance attribute through the object
car1.make = "Nissan"
print("Make of car 1 after change:", car1.make)

# Access the instance method through the object
print("Make and model of car 2 (instance method):", car2.get_make_model())
```

INSTANCE, CLASS & STATIC METHODS



```python
class MyClass:
    # Normal method
    def instance_method(self):
        print("This is an instance method.")

    # Static method
    @staticmethod
    def static_method():
        print("This is a static method.")

    # Class method
    @classmethod
    def class_method(cls):
        print("This is a class method. The class name is", cls.__name__)
```

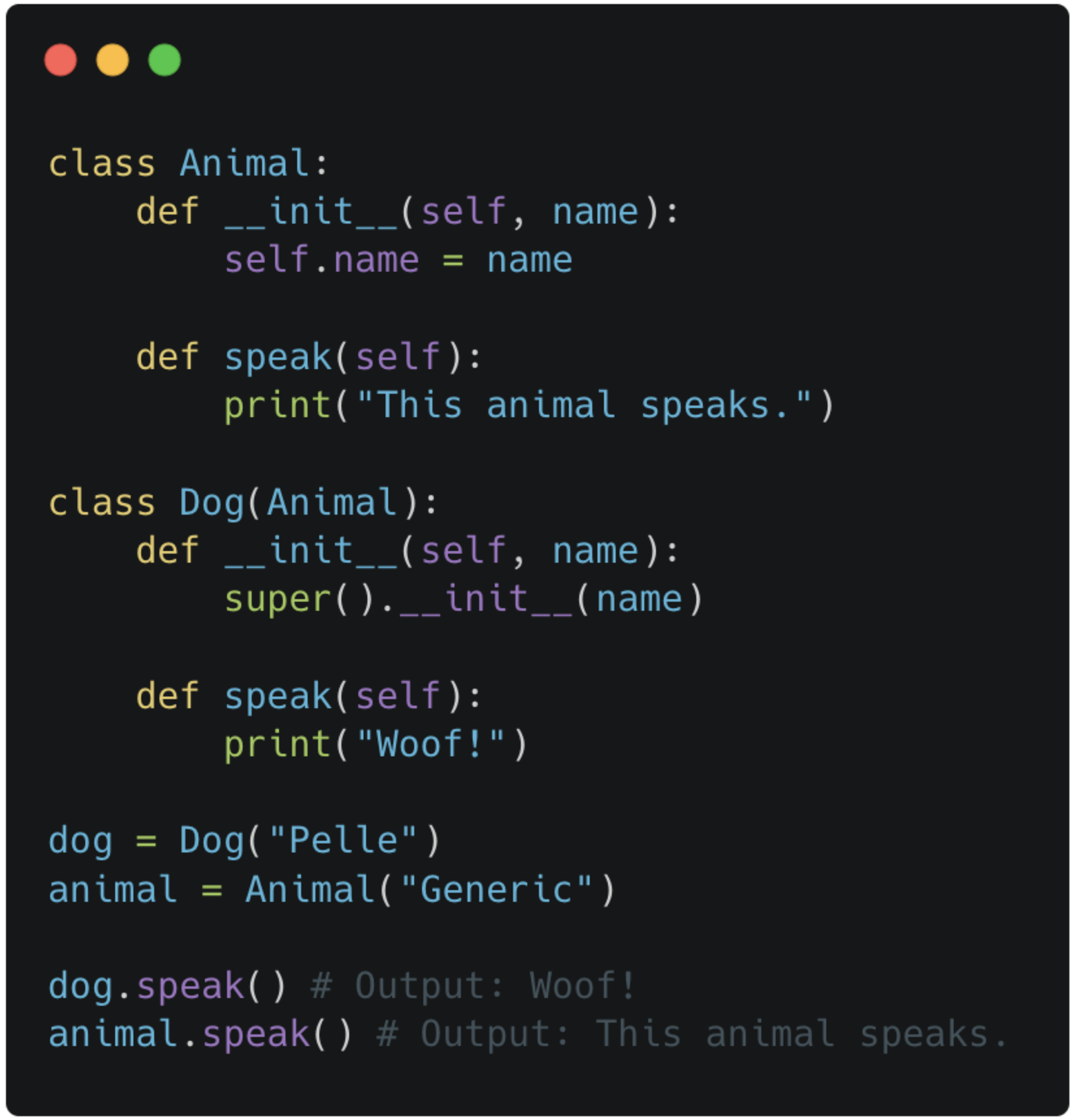
INSTANCE, CLASS & STATIC METHODS



```
# Create an object of MyClass
obj = MyClass()

# Call the instance method on the object
obj.instance_method()
>>> This is an instance method.

# Call the static method on the class
MyClass.static_method()
>>> This is a static method.

# Call the class method on the class
MyClass.class_method()
>>> This is a class method. The class name is MyClass
```

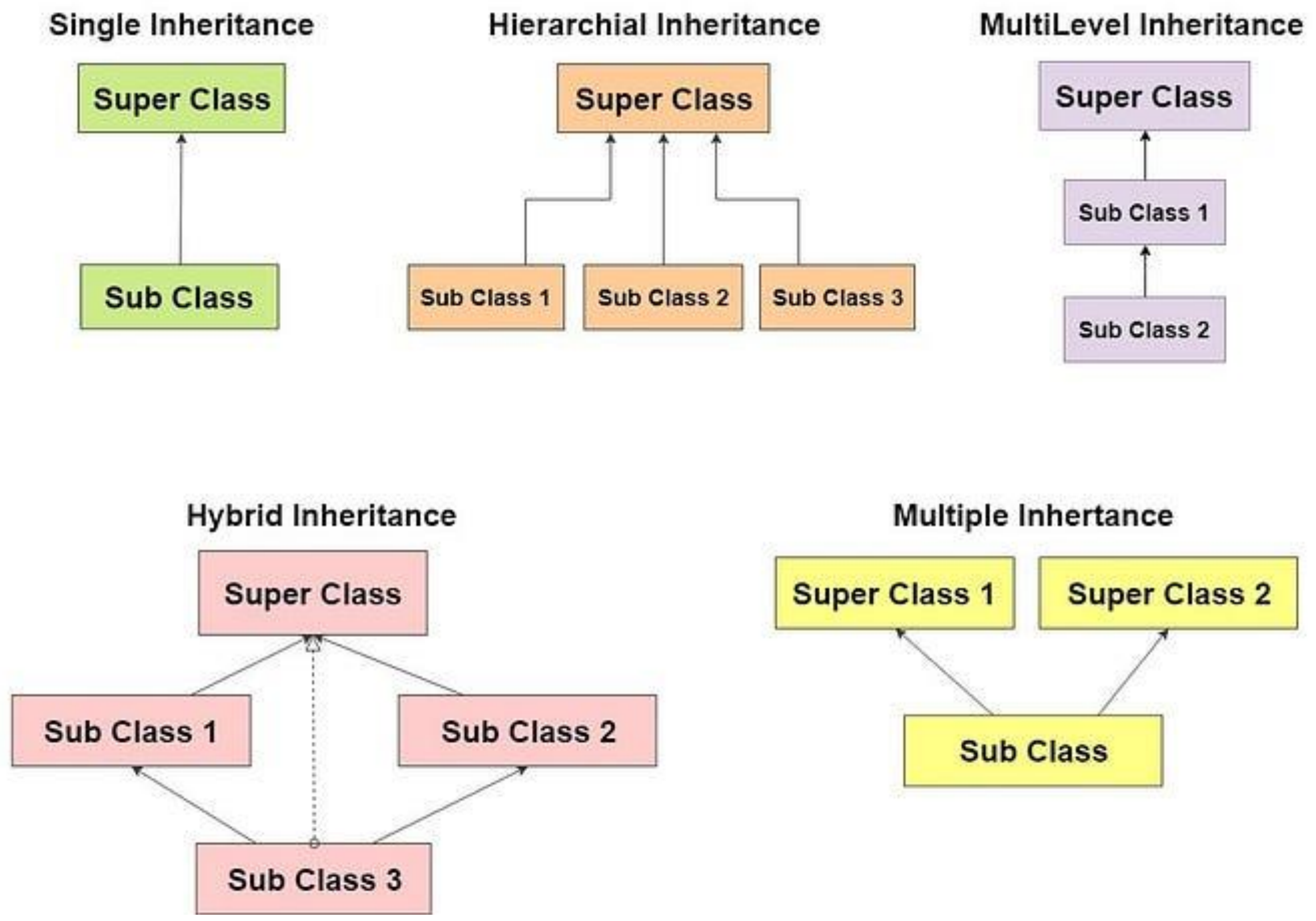
- Parent-child relationship
- Inherit attributes and methods from a parent class
- Uses the subclass syntax
  - **Class header specifies parent**
  - **Constructor calls “super”**
- Child can access or override all parent attributes and methods
- Useful e.g. to build hierarchies and DRY

INHERITANCE CLASSIC EXAMPLE



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

    def speak(self):
        print("This animal speaks.")

class Dog(Animal):
    def __init__(self, name):
        super().__init__(name)

    def speak(self):
        print("Woof!")

dog = Dog("Pelle")
animal = Animal("Generic")

dog.speak() # Output: Woof!
animal.speak() # Output: This animal speaks.
```

TYPES OF INHERITANCE



Source of the image: https://python.plainenglish.io/python-inheritance-45df7ab2d524

Unit 1

TRANSFER TASK

TRANSFER TASK

- Write a Python program to create a **People** class with:
  - Instance attributes: name, date of birth, nationality
  - Method: show the information
- Create a Student class that inherits from the **People** class with
  - New attributes: major, university
  - Method: show the information

TRANSFER TASK
PRESENTATION OF THE RESULTS

Please present your results.

The results will be discussed in plenary.