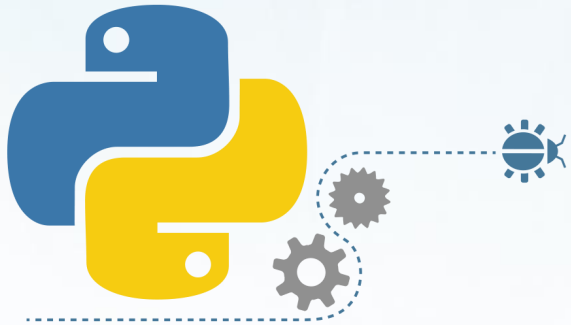
TRƯỜNG ĐẠI HỌC
SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH
HCMC University of Technology and Education

KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN HỆ THỐNG THÔNG TIN

NHẬP MÔN LẬP TRÌNH PYTHON (IPPA233277)

CLASS



GV. Trần Quang Khải

1. Hiểu được các khái niệm về lớp đối tượng
2. Nắm và vận dụng triển khai các thuộc tính của lớp và đối tượng
3. Nắm và thực hiện được các phép toán trên đối tượng
4. Hiểu được phạm vi truy cập thuộc tính và phương thức
5. Hiểu OOP trong Python

1. Dẫn nhập
2. Khai báo class, instance
3. Thuộc tính của lớp và thuộc tính đối tượng
4. Các phép toán trên đối tượng
5. Phạm vi truy cập thuộc tính và phương thức
6. OOP trong python
7. Hủy đối tượng

- Python là một ngôn ngữ đa mô hình (**multiparadigm**) hỗ trợ lập trình hướng đối tượng (OOP) thông qua các lớp (**class**) để xác định dữ liệu và hành vi của một đối tượng cụ thể được mô hình hóa.
- Thuật ngữ **attributes** (thuộc tính) để chỉ những thuộc tính hoặc dữ liệu được liên kết với đối tượng cụ thể của một lớp. Thuộc tính được định nghĩa trong một lớp với mục đích lưu trữ tất cả dữ liệu cần thiết cho lớp hoạt động.
- Thuật ngữ **method** (phương thức) để chỉ những hành vi khác nhau của đối tượng. Phương thức được định nghĩa trong một lớp.
- Thuật ngữ **instance** (thể hiện) là một đối tượng thể hiện cụ thể của lớp.
- Ngoài ra, cách thức tổ chức các lớp cũng được xây dựng dựa trên hệ thống phân cấp hay kế thừa nhằm sử dụng lại các đoạn mã và loại bỏ sự lặp lại mã nguồn trong chương trình

- Mỗi lớp python đều có các thuộc tính được xây dựng sẵn:
  - __dict__ : là dictionary chứa namespace của lớp
  - __doc__ : được sử dụng để truy cập document string
  - __name__ : là tên lớp
  - __module__ : là tên module trong đó lớp được định nghĩa
  - __bases__ : là một tuple chứa các lớp cơ sở

- Sử dụng từ khóa class để khai báo một lớp đối tượng

```
class ClassName:
    # Class body
```

Trong đó:

- ✓ ClassName tuân theo quy tắc đặc tên định danh
- ✓ Class body chứa các thuộc tính và phương thức cần xây dựng cho lớp

Ví dụ:

```
import math # circle.py
class Circle:
    """Lớp đối tượng đường tròn"""
    def __init__(self, radius):
        self.radius = radius
    def calculate_area(self):
        return round(math.pi * self.radius ** 2, 2)
```

Từ khóa

Tên lớp

Thuộc tính

Phương thức

- Khởi tạo đối tượng thể hiện (instance)

```
c = Circle(5)
```

- Hàm __init__() dùng để khởi tạo một thể hiện (instance) của đối tượng, nhận tham số self (tham chiếu đến instance đang được tạo) cùng với các thuộc tính của thể hiện đó.
- Để truy cập thuộc tính và phương thức sử dụng dấu chấm (.)

```
obj.attribute_name
obj.method_name()
```

Ví dụ:

```
print(c.__doc__)            # Lớp đối tượng đường tròn
c.radius = 5
print(c.calculate_area()) # 78.54
```

- Bỏ qua các giá trị không cần thiết

```
a, *_, b = (1, 2, 3, 4, 5, 6, 7)
print(a, b) # 1 7
```

- Sử dụng như một biến trong vòng lặp

```
for _ in range(5):
    print(_, end = " ") # 0 1 2 3 4
```

- Phân tách các chữ số của các con số

```
million = 1_000_000; print(million) # 1000000
binary = 0b_0010; print(binary)     # 2
octa = 0o_64; print(octa)           # 52
hexa = 0x_23_ab; print(hexa)        # 9131
```


- Đặt tên cho các biến, hàm, lớp,…
  - ✓ (1) Dấu gạch dưới trước biến _variable chỉ các biến sử dụng nội bộ bên trong một lớp
  - ✓ (2) Dấu gạch dưới sau biến variable_ tránh xung đột khi sử dụng từ khóa python đặt tên
  - ✓ (3) Hai dấu gạch dưới trước biến __variable theo cơ chế name mangling để yêu cầu trình thông dịch đặt lại tên để tránh xung đột với các lớp con khi kế thừa sử dụng lại biến
  - ✓ (4) Hai dấu gạch dưới trước và sau biến __variable__ chỉ các phương thức magic hoặc dunder là những phương thức đặc biệt được python tự động gọi

```python
class Sample():
    # (4) phương thức khởi tạo tự động thực thi
    def __init__(self):
        self.a = 1
        self._b = 2  # biến nội bộ (1)
        self.__c = 3 # biến mangling (3)
class SecondClass(Sample): # kế thừa
    def __init__(self):
        super().__init__()
        self.a = "overridden"
        self._b = "overridden"
        self.__c = "overridden"
```

```python
obj2 = SecondClass()
print(obj2.a)      # … ?
print(obj2._b)     # … ?
print(obj2.__c)    # … ?
```


- Sử dụng dấu chấm (.) và phép gán để thêm các thuộc tính và phương thức mới vào một lớp hoặc dùng các phương thức để tạo thêm thuộc tính:
  - ✓ Hàm getattr(obj, name[, default]) : để truy cập thuộc tính của đối tượng.
  - ✓ Hàm hasattr(obj,name) : để kiểm tra xem một thuộc tính có tồn tại hay không.
  - ✓ Hàm setattr(obj,name,value) : để thiết lập một thuộc tính. Nếu thuộc tính không tồn tại, thì nó sẽ được tạo.
  - ✓ Hàm delattr(obj, name) : để xóa một thuộc tính.

- **Thuộc tính lớp (class attribute)** là các biến mà được khai báo trực tiếp trong lớp nhưng nằm ngoài phương thức. Các thuộc tính này gắn với lớp, không gắn với đối tượng cụ thể của lớp
- Tất cả các đối tượng được tạo từ lớp đều có chung thuộc tính lớp với cùng giá trị ban đầu

```
class ObjectCounter:
    num_instances = 0 # thuộc tính lớp
    def __init__(self):
        type(self).num_instances += 1
```

- **Thuộc tính đối tượng (instance attribute)** là các biến được gắn với một đối tượng cụ thể của một lớp nhất định. Giá trị của một thuộc tính instance được gắn vào chính đối tượng đó, dành riêng cho thể hiện chứa nó.
- Các thuộc tính này thường được khai báo trong instance method thường là phương thức nhận self làm tham số đầu tiên, thường dùng là __init__()

Ví dụ:

```
# car.py
class Car:
    def __init__(self, model, color):
        self.model = model
        self.color = color
        self.started = False
        self.speed = 0
        self.max_speed = 200
```

Thuộc tính đối tượng

- Trong python cả lớp và đối tượng đều có một thuộc tính đặc biệt là __dict__ chứa các thuộc tính và phương thức của lớp đó.
- Khi truy xuất thành phần của lớp hoặc đối tượng, python tìm tên thành phần này trong __dict__, nếu không có sẽ trả về lỗi **AttributeError** hoặc **NameError**

```
# sample_dict.py
class SampleClass:
    class_attr = 100
    def __init__(self, instance_attr):
        self.instance_attr = instance_attr
    def method(self):
        print(f"Class attribute: {self.class_attr}")
        print(f"Instance attribute: {self.instance_attr}")
```

```
print(SampleClass.class_attr)
print(SampleClass.__dict__)
print(SampleClass.__dict__["class_attr"])
```


- Getter là một phương thức cho phép truy cập một thuộc tính trong một lớp nhất định
- Setter là một phương thức cho phép thay đổi giá trị của mộ thuộc tính trong một lớp
- Nên được thiết lập cho những thuộc tính cần ràng buộc điều kiện khi muốn gán hoặc truy xuất giá trị, các thuộc tính này thường được sử dụng nội bộ trong lớp.

```python
# xây dựng phương thức getter/setter
class Label:
    def __init__(self, text, font):
        self._text = text
        # hoặc self.set_text(text)
        self._font = font
    def get_text(self):
        return self._text
    def set_text(self, value):
        self._text = value.upper() # Attached behavior
    def get_font(self):
        return self._font
    def set_font(self, value):
        self._font = value
```

```
>>> from label import Label

>>> label = Label("Fruits", "Drinks")
>>> label.get_text()
'FRUITS'

>>> label.set_text("Vegetables")
>>> label.get_text()
'VEGETABLES'
```


```python
from datetime import date
class Employee:
    def __init__(self, name, birth_date):
        self.name = name
        self.birth_date = birth_date
    @property
    def name(self):
        return self._name
    @name.setter
    def name(self, value):
        self._name = value.upper()
    @property
    def birth_date(self):
        return self._birth_date
    @birth_date.setter
    def birth_date(self, value):
        self._birth_date = date.fromisoformat(value)
```

```
>>> from employee import Employee

>>> john = Employee("John", "2001-02-07")

>>> john.name
'JOHN'

>>> john.birth_date
datetime.date(2001, 2, 7)

>>> john.name = "John Doe"
>>> john.name
'JOHN DOE'
```


```python
from datetime import date
class Employee:
    def __init__(self, name, birth_date, start_date):
        self.name = name
        self.birth_date = birth_date
        self.start_date = start_date # xác định ngày làm việc nhân viên
    @property
    def name(self):
        return self._name
    @name.setter
    def name(self, value):
        self._name = value.upper()
    @property
    def birth_date(self):
        return self._birth_date
    @birth_date.setter
    def birth_date(self, value):
        self._birth_date = date.fromisoformat(value)
    @property
    def start_date(self):
        return self._start_date
    @start_date.setter
    def start_date(self, value):
        self._start_date = date.fromisoformat(value)
```

Trùng lặp hành vi

Chuyển đổi

```python
from datetime import date
class Date:
    def __set_name__(self, owner, name):
        self._name = name
    def __get__(self, instance, owner):
        return instance.__dict__[self._name]
    def __set__(self, instance, value):
        instance.__dict__[self._name] = date.fromisoformat(value)

class Employee:
    birth_date = Date()
    start_date = Date()
    def __init__(self, name, birth_date, start_date):
        self.name = name
        self.birth_date = birth_date
        self.start_date = start_date
    @property
    def name(self):
        return self._name
    @name.setter
    def name(self, value):
        self._name = value.upper()
```

- __getattr__(), __setattr__() là cách triển khai chung của getter/setter

```
# point.py
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __getattr__(self, name: str):
        return self.__dict__[f"_{name}"]
    def __setattr__(self, name, value):
        self.__dict__[f"_{name}"] = float(value)
```

```
>>> from point import Point

>>> point = Point(21, 42)

>>> point.x
21.0
>>> point.y
42.0

>>> point.x = 84
>>> point.x
84.0

>>> dir(point)
['__class__', '__delattr__', ..., '_x', '_y']
```


- Các phép toán số học +, -, *, / tương tứng với các magic method __add__(), __sub__(), __mul__(), __div__(). Phép toán âm (-) dương (+) tương tứng với __pos__(), __neg__().
- Do đó, để định nghĩa cách thức xử lý cho các phép toán này, cần định nghĩa (ghi đè) magic method tương ứng.

```
class Vector: # định nghĩa lại toán tử cách thức xử lý của vector
    """A class for vector"""
    def __init__(self, x:float, y:float):
        self.x = x; self.y = y
    def __str__(self):
        return f'({self.x}, {self.y})'
    def __repr__(self):
        return f'({self.x}, {self.y})’
    def __add__(self, v):
        x = self.x + v.x; y = self.y + v.y
        return Vector(x, y)
    def __sub__(self, v):
        x = self.x - v.x; y = self.y - v.y
        return Vector(x, y)
    def __mul__(self, n):
        x = self.x * n; y = self.y * n
        return Vector(x, y)
    def __neg__(self):
        return Vector(self.x * -1, self.y * -1)
```


- Các hàm toán học cơ bản và magic method tương ứng

Operator	Method	Operator	Method
+	__add__(self, other)	+=	__iadd__(self, other)
-	__sub__(self, other)	-=	__isub__(self, other)
*	__mul__(self, other)	*=	__imul__(self, other)
//	__floordiv__(self, other)	/=	__idiv__(self, other)
/	__truediv__(self, other)	//=	__ifloordiv__(self, other)
%	__mod__(self, other)	%=	__imod__(self, other)
**	__pow__(self, other[, modulo])	**=	__ipow__(self, other[, modulo])
<<	__lshift__(self, other)	<<=	__ilshift__(self, other)
>>	__rshift__(self, other)	>>=	__irshift__(self, other)
&	__and__(self, other)	&=	__iand__(self, other)
^	__xor__(self, other)	^=	__ixor__(self, other)
|	__or__(self, other)	|=	__ior__(self, other)

- Các hàm toán học cơ bản và magic method tương ứng

Operator	Method	Operator	Method
-	__neg__(self)	<	__lt__(self, other)
+	__pos__(self)	<=	__le__(self, other)
abs()	__abs__(self)	==	__eq__(self, other)
~	__invert__(self)	!=	__ne__(self, other)
complex()	__complex__(self)	>=	__ge__(self, other)
int()	__int__(self)	>	__gt__(self, other)
long()	__long__(self)		
float()	__float__(self)		
oct()	__oct__(self)		
hex()	__hex__(self		

- Kế thừa là cách thức tạo ra một lớp mới mà sử dụng lại các thành phần của một lớp hiện có.
- Lớp mới hình thành được gọi là lớp dẫn xuất – derived class (hoặc lớp con, subclass), lớp được kế thừa là lớp cơ sở - base class (hoặc lớp cha, superclass)

Cú pháp:

```
class DerivedClassName(BaseClassName):
    <statement-1>
    ...
    <statement-N>
```

Nếu lớp cơ sở định nghĩa ở module khác

```
class DerivedClassName(modname.BaseClassName):
```

Đa kế thừa

```
class DerivedClassName(Base1, Base2, Base3):
    <statement-1>
    ...
    <statement-N>
```


- Subclass có một số tính chất sau:
  - Kế thừa toàn bộ dữ liệu và phương thức của superclass
  - Có thể tạo thêm dữ liệu, phương thức nếu cần
  - Có thể ghi đè (override) lên phương thức superclass để đặc trưng cho lớp con
  - Sử dụng hàm super() ddeeer tham chiếu tới superclass
- Python có hai hàm dựng sẵn cho tính kế thừa
  - Sử dụng **isinstance(obj, class)** để kiểm tra loại đối tượng, trả về True nếu obj là một thể hiện của class hoặc là thể hiện của subclass của class
  - Sử dụng **issubclass(sub, sup)** để kiểm tra tính kế thừa của lớp, trả về True nếu sub là một lớp con của sup

- Hạn chế quyền truy cập phương thức và thuộc tính trong lớp từ đó giúp cho việc che giấu dữ liệu (data hiding)
- Nên thiết lập các thành phần của lớp ở chế độ private (riêng tư) thông qua dấu gạch dưới (_) hoặc (__)
- Các thuộc tính và phương thức private sẽ không thể được sử dụng bên ngoài lớp chứa chúng

Ví dụ:

```
class Person:
    def __init__(self):
        self.__age = 20
    def showAge(self):
        print(self.__age)
    def setAge(self, age):
        self.__age = age
```

```
david = Person()
print("Age of david: ", end='')
david.showAge()
david.__age = 99
print("Age of david after david.__age = 99: ", end='')
david.showAge()
david.setAge(99)
print("Age of david after david.setAge(99): ", end='')
david.showAge()
```


- Các đối tượng và phương thức giống nhau có thể có các hành vi khác nhau tùy vào từng tình huống khác nhau
- Cho phép định nghĩa các phương thức ở lớp con cùng tên với phương thức ở lớp cha
- Nếu phương thức lớp con giống hoàn toàn với lớp cha thì gọi là ghi đè (override) phương thức lớp cha

```
class Bird:
    def intro(self):
        print("This is bird")
    def flight(self):
        print("Flying method")
class Eagle(Bird):
    def flight(self):
        print("Eagle Flying")
class Hawks(Bird):
    def flight(self):
        print("Hawks Flying")
```

```
obj_bird = Bird()
obj_eag = Eagle()
obj_haw = Hawks()
obj_bird.intro()    # …?
obj_bird.flight()   # …?
obj_eag.intro()     # …?
obj_eag.flight()    # …?
obj_haw.intro()     # …?
obj_haw.flight()    # …?
```

- Lớp trừu tượng là một lớp không được khởi tạo trực tiếp và được sử dụng như một lớp cơ sở cho các lớp khác.
- Phương thức trừu tượng được khai báo trong lớp trừu tượng nhưng không được cài đặt cụ thể, công việc này sẽ dành cho các lớp con kế thừa triển khai.

Ví dụ:

```
from abc import ABC, abstractmethod
class MyAbstractClass(ABC):
    @abstractmethod
    def my_abstract_method(self):
        pass
```

```python
from abc import ABC, abstractmethod
class Animal(ABC):
    @abstractmethod
    def sound(self):
        pass
    def sleep(self):
        print("This animal is sleeping.")
class Dog(Animal):
    def sound(self):
        return "Woof"
class Cat(Animal):
    def sound(self):
        return "Meow"
```

```python
# Tạo các đối tượng động vật
dog = Dog()
cat = Cat()

# Gọi các phương thức
print(dog.sound())  # …?
print(cat.sound())  # …?
dog.sleep()         # …?
cat.sleep()         # …?
```


- Python sẽ hủy các đối tượng mà không cần dùng đến (các kiểu hoặc instance) một cách tự động để giải phóng không gian bộ nhớ, tiến trình này được gọi là Garbage Collection được thực hiện bởi Garbage Collector
- Trình dọn rác của Python chạy trong khi thực thi chương trình và được kích hoạt khi số tham chiếu của đối tượng tiến về 0.
- Số tham chiếu thay đổi khi số alias mà trỏ đến nó thay đổi. Số tham chiếu tăng khi được gán một tên mới hoặc đặt trong một container mới (list, tuple, dictionary,…) và giảm khi nó bị xóa bởi lệnh del, tham chiếu của nó được tái gán, hoặc thoát ra khỏi phạm vi
- Lớp có một phương thức __del__(), còn gọi là destructor được gọi khi instance chuẩn bị được hủy

Ví dụ:

```
a = 40      # Create object <40>
b = a       # Increase ref. count  of <40>
c = [b]     # Increase ref. count  of <40>
del a       # Decrease ref. count  of <40>
b = 100     # Decrease ref. count  of <40>
c[0] = -1   # Decrease ref. count  of <40>
```


__del__() destructor này in tên lớp của một instance mà chuẩn bị được hủy.

```
class Point:
   def __init__( self, x=0, y=0):
      self.x = x
      self.y = y
   def __del__(self):
      class_name = self.__class__.__name__
      print class_name, "destroyed"
```

```
pt1 = Point()
pt2 = pt1
pt3 = pt1
# prints the ids of the objects
print id(pt1), id(pt2), id(pt3)
del pt1
del pt2
del pt3
```

Kết quả sau:

```
3083401324 3083401324 3083401324
Point destroyed
```


- ✓ Họ tên : Trần Quang Khải
- ✓ Email : khaitq@hcmute.edu.vn
- ✓ Zalo (mã Qr)