


TRƯỜNG: ĐH Công thương TP HCM	BUỔI 3. LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI PYTHON	
KHOA: CÔNG NGHỆ THÔNG TIN		
BỘ MÔN: KHDL&TTNT		
MH: LẬP TRÌNH PYTHON		

A. MỤC TIÊU

- Tạo lớp và đối tượng trong Python

B. DỤNG CỤ - THIẾT BỊ THÍ NGHIỆM CHO MỘT SV

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	Computer	1	1	

C. NỘI DUNG THỰC HÀNH

I. Tóm tắt lý thuyết

1. Giới thiệu

Python là một ngôn ngữ lập trình hướng đối tượng. Nó cho phép chúng ta phát triển các ứng dụng bằng cách sử dụng phương pháp hướng đối tượng. Trong Python, chúng ta có thể dễ dàng tạo và sử dụng các lớp và các đối tượng.

Các khái niệm OOPs trong Python:

- Đối tượng (object)
- Lớp (class)
- Phương thức (method)
- Kế thừa
- Đa hình
- Trừu tượng
- Đóng gói

2. Lớp (Class) và Đối tượng (Object)

Class và Object là hai khái niệm cơ bản trong lập trình hướng đối tượng.

Đối tượng (Object) là những thực thể tồn tại có hành vi.

Ví dụ đối tượng là một xe ô tô có tên hãng, màu sắc, loại nguyên liệu, hành vi đi, dừng, đỗ, nổ máy...

Lớp (Class) là một kiểu dữ liệu đặc biệt do người dùng định nghĩa, tập hợp nhiều thuộc tính đặc trưng cho mọi đối tượng được tạo ra từ lớp đó.

Thuộc tính là các giá trị của lớp. Sau này khi các đối tượng được tạo ra từ lớp, thì thuộc tính của lớp lúc này sẽ trở thành các đặc điểm của đối tượng đó.

a. Phân biệt giữa Đối tượng (Object) và Lớp (Class):

- **Đối tượng (Object):** có trạng thái và hành vi.
- **Lớp (Class):** có thể được định nghĩa như là một template mô tả trạng thái và hành vi mà loại đối tượng của lớp hỗ trợ. Một đối tượng là một thực thể (instance) của một lớp



3. Lớp (class)

a. Định nghĩa một lớp: sử dụng từ khóa lớp để tạo một lớp trong Python

```
class Tenlop:
    'Mô tả ngắn về class (Không bắt buộc)'
    # code
```

Code: khai báo các thuộc tính (Attribute), phương thức (method) và các phương thức khởi tạo (Constructor).

Thuộc tính (Attribute): Thuộc tính là một thành viên thành viên của lớp. Chẳng hạn hình chữ nhật có hai thuộc tính width và height (Chiều rộng và chiều cao).

Phương thức (Method): Phương thức của class nó tương tự như một hàm thông thường, nhưng nó là một hàm của class, để sử dụng nó bạn cần phải gọi thông qua đối tượng.

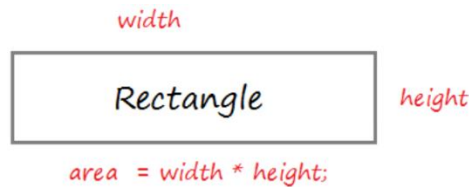
- Tham số đầu tiên của phương thức luôn là self (Một từ khóa ám chỉ chính class đó).
- Phương thức khởi tạo (Constructor):
- Phương thức khởi tạo (Constructor) là một phương thức đặc biệt của lớp (class), nó luôn có tên là `__init__`
- Tham số đầu tiên của constructor luôn là self (Một từ khóa ám chỉ chính class đó).

Phương thức khởi tạo: Constructor được sử dụng để tạo ra một đối tượng.

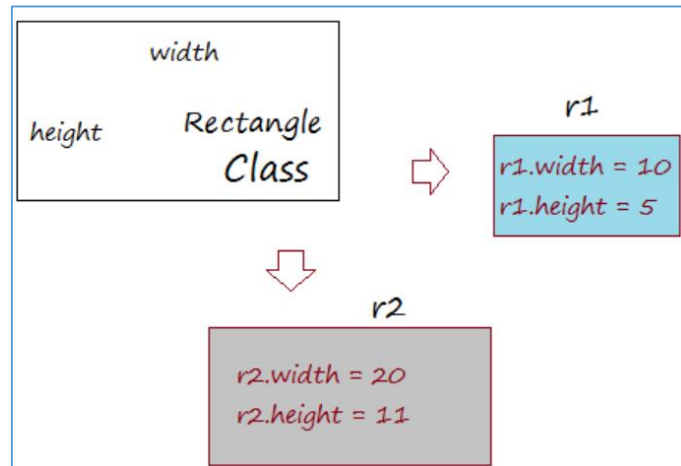
- Constructor gán các giá trị từ tham số vào các thuộc tính của đối tượng sẽ được tạo ra.

Lưu ý: Có thể định nghĩa nhiều nhất một phương thức khởi tạo (constructor) trong class. Nếu class không được định nghĩa constructor, Python mặc định coi rằng nó thừa kế từ constructor của lớp cha.

Ví dụ 1:



Tạo đối tượng từ lớp **Rectangle**:

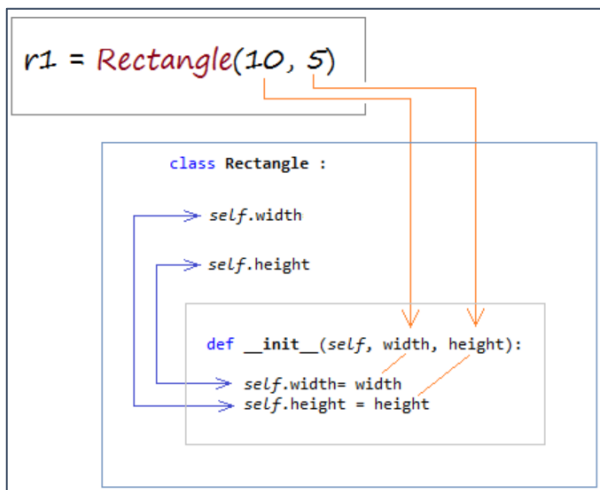


```
9 #Ví dụ 1: Hình chữ nhật
10 # Một Lớp mô phỏng một hình chữ nhật.
11 class Rectangle :
12     'This is Rectangle class'
13     # Một phương thức được sử dụng để tạo đối tượng (Constructor).
14     def __init__(self, width, height):
15         self.width= width
16         self.height = height
17
18     def getWidth(self):
19         return self.width
20
21     def getHeight(self):
22         return self.height
23
24     # Phương thức tính diện tích.
25     def getArea(self):
26         return self.width * self.height
27 # Tạo 2 đối tượng: r1 & r2
28 r1 = Rectangle(10,5)
29 r2 = Rectangle(20,11)
30
31 print ("r1.width = ", r1.width)
32 print ("r1.height = ", r1.height)
33 print ("r1.getWidth() = ", r1.getWidth())
34 print ("r1.getArea() = ", r1.getArea())
35
36 print ("-----")
37
38 print ("r2.width = ", r2.width)
39 print ("r2.height = ", r2.height)
40 print ("r2.getWidth() = ", r2.getWidth())
41 print ("r2.getArea() = ", r2.getArea())
42
```

Kết quả:

```
r1.width = 10
r1.height = 5
r1.getWidth() = 10
r1.getArea() = 50
-----
r2.width = 20
r2.height = 11
r2.getWidth() = 20
r2.getArea() = 220
```

Ta thấy việc tạo một đối tượng của lớp *Rectangle*, phương thức khởi tạo (constructor) của class đó sẽ được gọi để tạo một đối tượng, và các thuộc tính của đối tượng sẽ được gán giá trị từ tham số. Nó giống với hình minh họa dưới đây:



Tham số có mặc định trong Constructor:

Khác với các ngôn ngữ khác, lớp trong **Python** chỉ có nhiều nhất một phương thức khởi tạo (Constructor). Tuy nhiên **Python** cho phép tham số có giá trị mặc định.

Lưu ý: Tất cả các tham số bắt buộc (required parameters) phải đặt trước tất cả các tham số có giá trị mặc định.

Ví dụ 2: Khai báo lớp *Person* với các thuộc tính tên, tuổi và giới tính

```

45 #Ví dụ 2: Khai báo lớp Person với các thuộc tính:
46 class Person :
47     # Tham số age và gender có giá trị mặc định.
48     def __init__(self, name, age = 1, gender = "Male" ):
49         self.name = name
50         self.age = age
51         self.gender = gender
52
53     def showInfo(self):
54         print ("Name: ", self.name)
55         print ("Age: ", self.age)
56         print ("Gender: ", self.gender)
57
58 # Tạo một đối tượng Person.
59 aimee = Person("Aimee", 21, "Female")
60 aimee.showInfo()
61 print (" ----- ")
62 # age, gender mặc định.
63 alice = Person("Alice" )
64 alice.showInfo()
65 print (" ----- ")
66 # gender mặc định.
67 tran = Person("Tran", 37)
68 tran.showInfo()

```

Kết quả:

```

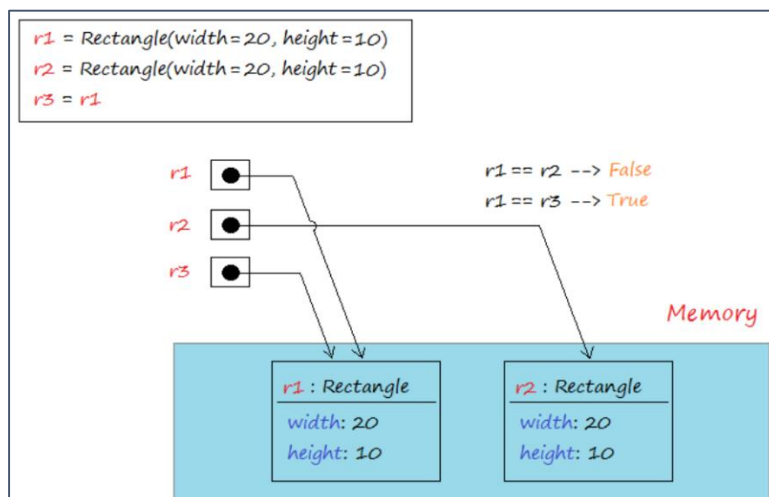
Name: Aimee
Age: 21
Gender: Female
-----
Name: Alice
Age: 1
Gender: Male
-----
Name: Tran
Age: 37
Gender: Male

```

b. So sánh các đối tượng:

Trong **Python**, khi tạo một đối tượng thông qua phương thức khởi tạo (Constructor), sẽ có một thực thể thực sự được tạo ra nằm trên bộ nhớ, nó có một địa chỉ xác định.

Một phép toán gán đối tượng **AA** bởi một đối tượng **BB** không tạo ra thêm thực thể trên bộ nhớ, nó chỉ là trỏ địa chỉ của **AA** tới địa chỉ của **BB**.



- Toán tử `==` dùng để so sánh địa chỉ 2 đối tượng trỏ đến, nó trả về True nếu cả 2 đối tượng cùng trỏ tới cùng một địa chỉ trên bộ nhớ.
- Toán tử `!=` cũng sử dụng để so sánh 2 địa chỉ của 2 đối tượng trỏ đến, nó trả về True nếu 2 đối tượng trỏ tới 2 địa chỉ khác nhau.

```

45 #SO SÁNH HAI ĐỐI TƯỢNG
46 r1 = Rectangle(20, 10)
47 r2 = Rectangle(20, 10)
48 r3 = r1
49
50 # So sánh địa chỉ của r1 và r2
51 test1 = r1 == r2 # --> False
52
53 # So sánh địa chỉ của r1 và r3
54 test2 = r1 == r3 # --> True
55
56 print ("-----")
57 print ("r1 == r2 ? ", test1)
58 print ("r1 == r3 ? ", test2)
59
60 print (" ----- ")
61 print ("r1 != r2 ? ", r1 != r2)
62 print ("r1 != r3 ? ", r1 != r3)

```

c. Thuộc tính (Attribute):

Trong Python có 2 khái niệm khá giống nhau, chúng ta cần phải phân biệt nó:

- Thuộc tính (Attribute)
- Biến của lớp

Để đơn giản, chúng ta hãy phân tích ví dụ dưới đây:

```

class Player:
    minAge = 18
    maxAge = 50

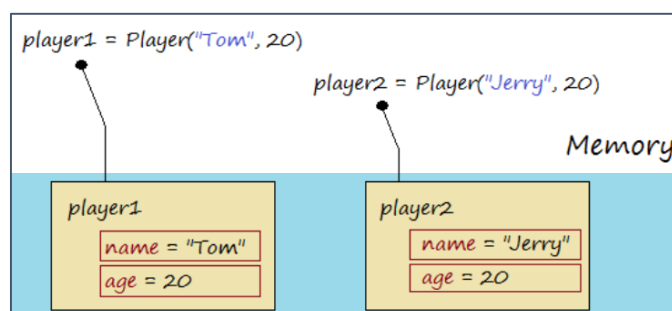
    def __init__(self, name, age):
        self.name = name
        self.age = age

```

Variables of the Player class (pointing to minAge and maxAge)

The Attributes of the Player class (pointing to self.name and self.age)

Các đối tượng được tạo ra từ một lớp, chúng sẽ nằm tại các địa chỉ khác nhau trên bộ nhớ (memory), và các thuộc tính "cùng tên" của chúng cũng có các địa chỉ khác nhau trên bộ nhớ. **Ví dụ:**



```

player1 = Player("Tom", 20)
player2 = Player("Jerry", 20)

print (" ----- ")
print ("player1.name = ", player1.name)
print ("player1.age = ", player1.age)

print ("player2.name = ", player2.name)
print ("player2.age = ", player2.age)

print (" ----- ")
print ("Assign new value to player1.age = 21 ")

# Gán giá trị mới cho thuộc tính (attribute) age của player1.
player1.age = 21
print ("player1.name = ", player1.name)
print ("player1.age = ", player1.age)

print ("player2.name = ", player2.name)
print ("player2.age = ", player2.age)

```

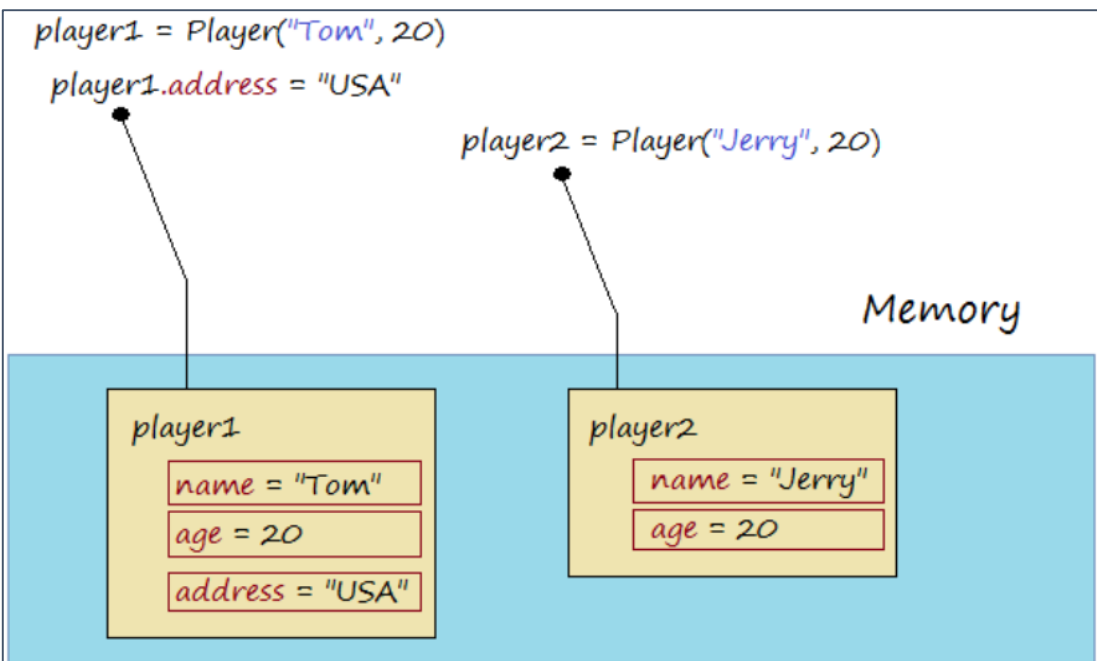
Kết quả:

```

player1.name = Tom
player1.age = 20
player2.name = Jerry
player2.age = 20
-----
Assign new value to player1.age = 21
player1.name = Tom
player1.age = 21
player2.name = Jerry
player2.age = 20

```

Python cho phép tạo ra một thuộc tính mới cho một đối tượng có trước. Ví dụ đối tượng **player1** và thuộc tính mới có tên **address**.



```

player1 = Player("Tom", 20)

player2 = Player("Jerry", 20)

# Tạo một thuộc tính có tên 'address' cho player1.
player1.address = "USA"
print (" ----- ")
print ("player1.name = ", player1.name)
print ("player1.age = ", player1.age)
print ("player1.address = ", player1.address)

print (" ----- ")

print ("player2.name = ", player2.name)
print ("player2.age = ", player2.age)

# player2 không có thuộc tính 'address' (Lỗi xảy ra tại đây).
#print ("player2.address = ", player2.address)

```

Kết quả:

```

player1.name = Tom
player1.age = 20
player1.address = USA
-----
player2.name = Jerry
player2.age = 20

```

Các hàm truy cập vào thuộc tính:

Thông thường để truy cập vào thuộc tính của một đối tượng thông qua toán tử "dấu chấm" (Ví dụ *player1.name*). Tuy nhiên **Python** cho phép truy cập chúng thông qua hàm (function).

Hàm	Mô tả
getattr(obj, name[, default])	Trả về giá trị của thuộc tính, hoặc trả về giá trị mặc định nếu đối tượng không có thuộc tính này.
hasattr(obj,name)	Kiểm tra xem đối tượng này có thuộc tính cho bởi tham số 'name' hay không.
setattr(obj,name,value)	Sét giá trị vào thuộc tính. Nếu thuộc tính không tồn tại, thì nó sẽ được tạo ra.
delattr(obj, name)	Xóa bỏ thuộc tính.

Ví dụ:

Kết quả:


```

player1 = Player("Tom", 20)

# getattr(obj, name[, default])
print ("getattr(player1,'name') = " , getattr(player1,"name") )

print ("setattr(player1,'age', 21): ")

# setattr(obj,name,value)
setattr(player1,"age", 21)

print ("player1.age = ", player1.age)

# Kiểm tra player1 có thuộc tính (attribute) address hay không?
hasAddress = hasattr(player1, "address")

print ("hasattr(player1, 'address') ? ", hasAddress)

# Tạo thuộc tính 'address' cho đối tượng 'player1'.
print ("Create attribute 'address' for object 'player1'")
setattr(player1, 'address', "USA")

print ("player1.address = ", player1.address)

# Xóa thuộc tính 'address'.
delattr(player1, "address")

```

```

-----Sử dụng hàm truy cập thuộc tính-----
getattr(player1,'name') = Tom
setattr(player1,'age', 21):
player1.age = 21
hasattr(player1, 'address') ? False
Create attribute 'address' for object 'player1'
player1.address = USA

```

Các thuộc tính có sẵn của class:

Các lớp của **Python** đều là hậu duệ của lớp **object**. Và vì vậy nó thừa kế các thuộc tính sau:

Thuộc tính	Mô tả
<code>__dict__</code>	Đưa ra thông tin về lớp này một cách ngắn gọn, dễ hiểu, như một bộ từ điển (Dictionary)
<code>__doc__</code>	Trả về chuỗi mô tả về class, hoặc trả về None nếu nó không được định nghĩa
<code>__class__</code>	Trả về một đối tượng, chứa thông tin về lớp, đối tượng này có nhiều thuộc tính có ích, trong đó có thuộc tính <code>__name__</code> .
<code>__module__</code>	Trả về tên module của lớp, hoặc trả về " <code>__main__</code> " nếu lớp đó được định nghĩa trong module đang được chạy.

Ví dụ:

```
#Các thuộc tính có sẵn của class:
class Customer :
    'This is Customer class'

    def __init__(self, name, phone, address):

        self.name = name
        self.phone = phone
        self.address = address

john = Customer("John",1234567, "USA")

print ("")
print ("Các thuộc tính có sẵn của class")
print ("john.__dict__ = ", john.__dict__)

print ("john.__doc__ = ", john.__doc__)

print ("john.__class__ = ", john.__class__)
print ("john.__class__.__name__ = ", john.__class__.__name__)
print ("john.__module__ = ", john.__module__)
```

Kết quả:

```
Các thuộc tính có sẵn của class
john.__dict__ = {'name': 'John', 'phone': 1234567, 'address': 'USA'}
john.__doc__ = This is Customer class
john.__class__ = <class '__main__.Customer'>
john.__class__.__name__ = Customer
john.__module__ = __main__
```

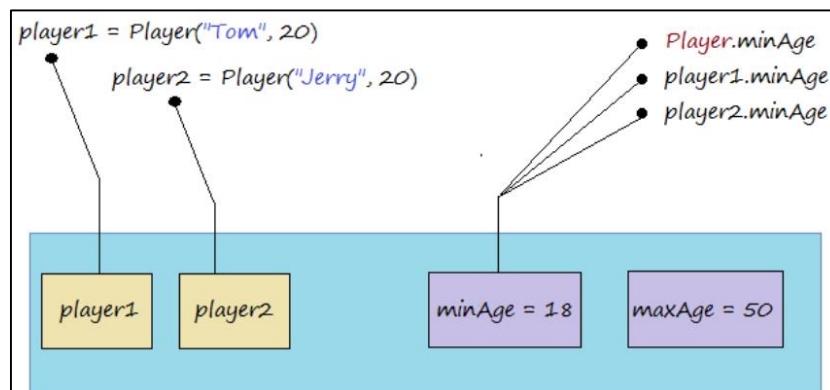
Biến của lớp:

Trong **Python** khái niệm "Biến của lớp (Class's Variable)" tương đương với khái niệm trường tĩnh (Static Field) của các ngôn ngữ khác như **Java**, **CSharp**. Biến của lớp có thể được truy cập thông qua tên lớp hoặc thông qua đối tượng.

```
class Player:
    # Biến của Lớp.
    minAge = 18
    maxAge = 50

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

Chú ý: chúng ta nên truy cập "biến của lớp" thông qua tên lớp thay vì truy cập thông qua đối tượng. Điều này giúp tránh nhầm lẫn giữa "biến của lớp" và thuộc tính.



```

#Biến của Lớp
player1 = Player("Tom", 20)
player2 = Player("Jerry", 20)
print ("")
print ("Biến của lớp")
# Truy cập thông qua tên Lớp.
print ("Player.minAge = ", Player.minAge)

# Truy cập thông qua đối tượng.
print ("player1.minAge = ", player1.minAge)
print ("player2.minAge = ", player2.minAge)
print (" ----- ")
print ("Assign new value to minAge via class name, and print..")

# Gán một giá trị mới cho minAge thông qua tên lớp.
Player.minAge = 19

print ("Player.minAge = ", Player.minAge)
print ("player1.minAge = ", player1.minAge)
print ("player2.minAge = ", player2.minAge)

```

Kết quả:

```

Biến của lớp
Player.minAge = 18
player1.minAge = 18
player2.minAge = 18
-----
Assign new value to minAge via class name, and print..
Player.minAge = 19
player1.minAge = 19
player2.minAge = 19

```

Liệt kê danh sách các thành viên của lớp hoặc đối tượng:

Python cung cấp cho bạn hàm **dir**, hàm này liệt kê ra danh sách các phương thức, thuộc tính, biến của lớp hoặc của đối tượng.

```

#Liệt kê danh sách các thành viên của Lớp hoặc đối tượng

print ("")
print ("Liệt kê danh sách các thành viên của lớp hoặc đối tượng")
print ( dir(Player) )
print ("\n\n")
player1 = Player("Tom", 20)
player1.address ="USA"

# In ra danh sách các thuộc tính, phương thức và biến của đối tượng 'player1'.
print ( dir(player1) )

```

Kết quả:

```

Liệt kê danh sách các thành viên của lớp hoặc đối tượng
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__',
 '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'maxAge', 'minAge']

['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__',
 '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'address', 'age',
 'maxAge', 'minAge', 'name']

```

```

class Car:

    # thuộc tính lớp
    loaixe = "Ô tô"

    # thuộc tính đối tượng
    def __init__(self, tenxe, mausac, nguyenvlieu):
        self.tenxe = tenxe
        self.mausac = mausac
        self.nguyenvlieu = nguyenvlieu

# instantiate the Car class
toyota = Car("Toyota", "Đỏ", "Điện")
lamborghini = Car("Lamborghini", "Vàng", "Deisel")
porsche = Car("Porsche", "Xanh", "Gas")

# access the class attributes
print("Porsche là {}".format(porsche.__class__.loaixe))
print("Toyota là {}".format(toyota.__class__.loaixe))
print("Lamborghini cũng là {}".format(lamborghini.__class__.loaixe))

# access the instance attributes
print("Xe {} có màu {}. {} là nguyên liệu vận hành.".format( toyota.tenxe, toyota.mausac,
toyota.nguyenvlieu))
print("Xe {} có màu {}. {} là nguyên liệu vận hành.".format( lamborghini.tenxe,
lamborghini.mausac,lamborghini.nguyenvlieu))
print("Xe {} có màu {}. {} là nguyên liệu vận hành.".format( porsche.tenxe, porsche.mausac,
porsche.nguyenvlieu))

```

Kết quả:

```

Porsche là Ô tô.
Toyota là Ô tô.
Lamborghini cũng là Ô tô.
Xe Toyota có màu Đỏ. Điện là nguyên liệu vận hành.
Xe Lamborghini có màu Vàng. Deisel là nguyên liệu vận hành.
Xe Porsche có màu Xanh. Gas là nguyên liệu vận hành.

```

Chương trình trên tạo một lớp *Car*, sau đó xác định các thuộc tính, đặc điểm của đối tượng

Chúng ta truy cập thuộc tính class bằng cách sử dụng `__class__.loaixe`. Các thuộc tính lớp được chia sẻ cho tất cả các cá thể của lớp.

Tương tự, chúng ta truy cập các thuộc tính instance bằng cách sử dụng `toyota.tenxe`, `toyota.mausac` và `toyota.nguyenvlieu`.

```

class Car:

    # thuộc tính đối tượng
    def __init__(self, tenxe, mausac, nguyenvlieu):
        self.tenxe = tenxe
        self.mausac = mausac
        self.nguyenvlieu = nguyenvlieu

    # phương thức
    def dungxe(self, mucdich):
        return "{} đang dùng xe để {}".format(self.tenxe,mucdich)

    def chayxe(self):

```

```

        return "{} đang chạy trên đường".format(self.tenxe)

    def nomay(self):
        return "{} đang nổ máy".format(self.tenxe)

# instantiate the Car class
toyota = Car("Toyota", "Đỏ", "Điện")
lamborghini = Car("Lamborghini", "Vàng", "Deisel")
porsche = Car("Porsche", "Xanh", "Gas")

# call our instance methods
print(toyota.dungxe("nạp điện"))
print(lamborghini.chayxe())
print(porsche.nomay())

```

Kết quả:

```

Toyota đang dừng xe để nạp điện
Lamborghini đang chạy trên đường
Porsche đang nổ máy

```

II. Bài tập hướng dẫn mẫu

Bài 1. Tạo lớp xe (Car) và xây dựng các thuộc tính cho lớp Car: loại xe và các thuộc tính đối tượng: tên xe, màu sắc, nguyên liệu.

```

class Car:

    # thuộc tính lớp
    loaixe = "Ô tô"

    # thuộc tính đối tượng
    def __init__(self, tenxe, mausac, nguyenvl):
        self.tenxe = tenxe
        self.mausac = mausac
        self.nguyenvl = nguyenvl

# instantiate the Car class
toyota = Car("Toyota", "Đỏ", "Điện")
lamborghini = Car("Lamborghini", "Vàng", "Deisel")
porsche = Car("Porsche", "Xanh", "Gas")

# access the class attributes
print("Porsche là {}".format(porsche.__class__.loaixe))
print("Toyota là {}".format(toyota.__class__.loaixe))
print("Lamborghini cũng là {}".format(lamborghini.__class__.loaixe))

# access the instance attributes
print("Xe {} có màu {}. {} là nguyên liệu vận hành.".format( toyota.tenxe, toyota.mausac,
toyota.nguyenvl))
print("Xe {} có màu {}. {} là nguyên liệu vận hành.".format( lamborghini.tenxe,
lamborghini.mausac,lamborghini.nguyenvl))
print("Xe {} có màu {}. {} là nguyên liệu vận hành.".format( porsche.tenxe, porsche.mausac,
porsche.nguyenvl))

```

Kết quả:

```

Porsche là Ô tô.
Toyota là Ô tô.
Lamborghini cũng là Ô tô.
Xe Toyota có màu Đỏ. Điện là nguyên liệu vận hành.
Xe Lamborghini có màu Vàng. Deisel là nguyên liệu vận hành.

```

Xe Porsche có màu Xanh. Gas là nguyên liệu vận hành.

Chương trình trên tạo một lớp *Car*, sau đó xác định các thuộc tính, đặc điểm của đối tượng

Chúng ta truy cập thuộc tính class bằng cách sử dụng `__class__`. Các thuộc tính lớp được chia sẻ cho tất cả các cá thể của lớp.

Tương tự, chúng ta truy cập các thuộc tính instance bằng cách sử dụng `toyota.tenxe`, `toyota.mausac` và `toyota.nguyenlieu`.

```
class Car:

    # thuộc tính đối tượng
    def __init__(self, tenxe, mausac, nguyenlieu):
        self.tenxe = tenxe
        self.mausac = mausac
        self.nguyenlieu = nguyenlieu

    # phương thức
    def dungxe(self, mucdich):
        return "{} đang dừng xe để {}".format(self.tenxe, mucdich)

    def chayxe(self):
        return "{} đang chạy trên đường".format(self.tenxe)

    def nomay(self):
        return "{} đang nổ máy".format(self.tenxe)

# instantiate the Car class
toyota = Car("Toyota", "Đỏ", "Điện")
lamborghini = Car("Lamborghini", "Vàng", "Deisel")
porsche = Car("Porsche", "Xanh", "Gas")

# call our instance methods
print(toyota.dungxe("nạp điện"))
print(lamborghini.chayxe())
print(porsche.nomay())
```

Kết quả:

Toyota đang dừng xe để nạp điện
Lamborghini đang chạy trên đường
Porsche đang nổ máy

Bài 2. Tạo một lớp thực hiện thao tác Máy tính tay bao gồm các phép toán +, -, *, /

Yêu cầu:

1. Tạo một lớp và sử dụng hàm để khởi tạo các giá trị của lớp đó.
2. Tạo các phương thức cộng, trừ, nhân, chia hai số và trả về kết quả tương ứng.
3. Lấy hai số làm đầu vào và tạo một đối tượng cho lớp truyền hai số đó làm tham số cho lớp.
4. Sử dụng đối tượng, gọi hàm tương ứng tùy theo lựa chọn của người dùng.
5. In kết quả cuối cùng.
6. Thoát

```

class calculator_implementation():
    def __init__(self,in_1,in_2):
        self.a=in_1
        self.b=in_2
    def add_vals(self):
        return self.a+self.b
    def multiply_vals(self):
        return self.a*self.b
    def divide_vals(self):
        return self.a/self.b
    def subtract_vals(self):
        return self.a-self.b
input_1 = int(input("Enter the first number: "))
input_2 = int(input("Enter the second number: "))
print("The entered first and second numbers are : ")
print(input_1, input_2)
my_instance = calculator_implementation(input_1,input_2)
choice=1
while choice!=0:
    print("0. Exit")
    print("1. Addition")
    print("2. Subtraction")
    print("3. Multiplication")
    print("4. Division")
    choice=int(input("Enter your choice... "))
    if choice==1:
        print("The computed addition result is : ",my_instance.add_vals())
    elif choice==2:
        print("The computed subtraction result is : ",my_instance.subtract_vals())
    elif choice==3:
        print("The computed product result is : ",my_instance.multiply_vals())
    elif choice==4:
        print("The computed division result is : ",round(my_instance.divide_vals(),2))
    elif choice==0:
        print("Exit")
    else:
        print("Sorry, invalid choice!")
print()

```

Bài 3. Xây dựng một chương trình lấy chiều dài và chiều rộng từ người dùng và tìm diện tích hình chữ nhật bằng cách sử dụng các lớp.

Yêu cầu:

1. Lấy giá trị chiều dài và chiều rộng từ người dùng.
2. Tạo một lớp và sử dụng hàm tạo để khởi tạo các giá trị của lớp đó.
3. Tạo một phương thức gọi là diện tích và trả về diện tích của lớp.
4. Tạo đối tượng cho lớp.

5. Sử dụng đối tượng, gọi phương thức vùng() với các tham số là chiều dài và chiều rộng được lấy từ người dùng.

6. In kết quả

7. Thoát

```
class rectangle():
    def __init__(self,breadth,length):
        self.breadth=breadth
        self.length=length
    def area(self):
        return self.breadth*self.length
a=int(input("Enter length of rectangle: "))
b=int(input("Enter breadth of rectangle: "))
obj=rectangle(a,b)
print("Area of rectangle:",obj.area())

print()
```

III. Bài tập ở lớp

Bài 1. Xây dựng một chương trình để tính diện tích và chu vi hình tròn.

Yêu cầu:

1. Lấy giá trị bán kính hình tròn từ người dùng.
2. Tạo một lớp và sử dụng hàm tạo để khởi tạo các giá trị của lớp đó.
3. Tạo một phương thức gọi là diện tích/chu vi và trả về diện tích/chu vi của lớp.
4. In kết quả
5. Thoát

Bài 2. Viết một chương trình để tạo một lớp trong đó một phương thức chấp nhận một chuỗi từ người dùng và một phương thức để in chuỗi.

Yêu cầu:

1. Tạo một lớp và sử dụng hàm tạo để khởi tạo các giá trị của lớp đó.
2. Tạo phương thức getName() in ra màn hình Tên và MSSV của Sinh viên.
3. Tạo hai phương thức gọi là **get** để nhận giá trị của một chuỗi và một phương thức khác gọi là **put** để in chuỗi đó.
4. Tạo đối tượng cho lớp.
5. Sử dụng đối tượng, gọi cả hai phương thức.
6. In ra chuỗi.
7. Thoát

Bài 3. Viết một chương trình để tạo một lớp và lấy tất cả các tập hợp con riêng biệt có thể có từ một tập hợp các số nguyên riêng biệt.

Yêu cầu:

1. Tạo một lớp và định nghĩa hai phương thức trong lớp:

- Phương thức f1 được sử dụng để chuyển một danh sách trống và danh sách đã sắp xếp được lấy từ người dùng sang phương thức f2.
- Phương pháp f2 được sử dụng để tính toán tất cả các tập hợp con có thể có của danh sách.

2. In ra kết quả các tập hợp con có thể có của danh sách

3. Thoát

Bài 4. Viết một chương trình tạo lớp nhân viên gồm các thuộc tính: tên, tuổi, địa chỉ, tiền lương, tổng số giờ làm và các phương thức

- inputInfo(): Nhập thông tin nhân viên từ bàn phím
- printInfo(): In ra tất cả các thông tin nhân viên
- tinhThuong(): Tính và trả về tiền thưởng của nhân viên theo công thức sau:
 - + Nếu tổng số giờ làm của nhân viên ≥ 200 thì thưởng = lương * 20%
 - + Nếu tổng số giờ làm của nhân viên < 200 và ≥ 100 thì thưởng = lương * 10%
 - + Nếu tổng số giờ làm của nhân viên < 100 thì thưởng = 0

Bài 5. Viết một chương trình tạo lớp sinh viên và lưu trữ các thông tin cho một sinh viên: Mã số sinh viên (10 ký tự), điểm trung bình từ 0 đến 10, tuổi phải lớn hơn hoặc bằng 18, Lớp phải bắt đầu bằng Khóa học (14) - kể đến là 2 ký tự DH - kể đến chuyên ngành học (TH/BM/DS) – cuối cùng là tên lớp có 2 chữ số từ 01 đến 99 (Ví dụ: 14DHTH02/14DHBM04) và xây dựng các phương thức:

- Phương thức createNew(): tạo ngẫu nhiên các thông tin cho 1 sinh viên
- Phương thức inputInfo(): nhập thông tin sinh viên từ bàn phím
- Phương thức showInfo(): hiển thị tất cả các thông tin của sinh viên
- Phương thức xét xem sinh viên có được học bổng không? Biết rằng, nếu điểm trung bình ≥ 8 là được học bổng.

Bài 6. Xây dựng lớp tam giác (Triangle) có các thành phần: 3 cạnh a, b, c của một tam giác và các phương thức:

- Nhập độ dài 3 cạnh của tam giác
- Xác định tam giác thuộc loại tam giác gì
- Tính chu vi của tam giác
- Tính diện tích của tam giác

IV. Bài tập về nhà

Câu 1. Xây dựng một lớp người (Person). Bao gồm các thuộc tính như tên, quốc gia và ngày sinh và các phương thức:

- Xác định tuổi của người đó.
- In ra thông tin của người đó:

Person 1:

Name: Nguyễn Văn Nam

Country: Việt nam

Date of Birth: 1962-07-12

Age: 60

Câu 2. Xây dựng một lớp phân số (Fraction) có các thuộc tính: Tử số, mẫu số và các phương thức:

- Phương thức tạo để khởi tạo giá trị cho tử số và mẫu số
- Các phương nhập, xuất, rút gọn, nghịch đảo phân số
- Các phương thức cộng (add), trừ (sub), nhân (mul), chia (div), hai phân số.

Câu 3. Xây dựng lớp số phức có các thuộc tính: phần thực, phần ảo và các phương thức:

- Cộng, trừ, nhân, chia hai số phức
- Nhập 2 số phức và thực hiện các phép toán trên hai số phức, xuất ra kết quả.

Câu 4. Xây dựng một chương trình thêm, xóa và hiển thị các phần tử của danh sách bằng cách sử dụng các lớp.

Yêu cầu:

1. Tạo một lớp và sử dụng hàm tạo để khởi tạo các giá trị của lớp đó.
2. Tạo các phương thức thêm, xóa, hiển thị các phần tử trong danh sách và trả về các giá trị tương ứng.
3. Tạo đối tượng cho lớp và sử dụng đối tượng để gọi hàm tương ứng tùy theo lựa chọn của người dùng.
5. In danh sách kết quả cuối cùng.
6. Thoát chương trình