LẬP TRÌNH PYTHON

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI PYTHON

NỘI DUNG

1. Giới thiệu về Lớp - class
2. Phạm vi – scope và Không gian tên – namespace
3. Khai báo lớp, hàm dựng - constructor
4. Thuộc tính - attribute
5. Phương thức – method
6. Đóng gói - Encapsulation
7. Kế thừa – inheritance
8. Đa hình - polymorphism

KHÁI NIỆM VỀ LỚP

Lớp (class) được đưa ra như một phương tiện trên đó có thể kết hợp dữ liệu và các hàm chức năng cùng với nhau.

- Việc khởi tạo một lớp mới sẽ tạo ra một loại mới của đối tượng (object) cho phép tạo ra các thể hiện (instance) mới của loại đối tượng đó
- Mỗi thể hiện của một lớp có thể có các thuộc tính đính kèm với nó để duy trì các trạng thái của của đối tượng.
- Các thể hiện của lớp cũng có thể có các phương thức để thay đổi trạng thái của đối tượng.

PHẠM VI – SCOPE VÀ KHÔNG GIAN TÊN – NAMESPACE

Một không gian tên là một ánh xạ từ tên đến các đối tượng. Hầu hết các không gian tên hiện đang được triển khai dưới dạng từ điển Python

- Ví dụ về không gian tên là: tập hợp các tên dựng sẵn như các từ khóa (tên các hàm như abs () và các tên ngoại lệ dựng sẵn)

Điều quan trọng cần biết về không gian tên là hoàn toàn không có mối quan hệ giữa các tên trong các không gian tên khác nhau;

- Ví dụ hai mô-đun khác nhau có thể định nghĩa cùng một hàm maximize mà không dẫn đến một sự nhầm lẫn nào - người sử dụng mô-đun phải đặt tiền tố với tên mô-đun.

PHẠM VI – SCOPE VÀ KHÔNG GIAN TÊN – NAMESPACE

- Không gian tên được tạo ra tại những thời điểm khác nhau và có thời gian tồn tại khác nhau.
- Không gian tên chứa các tên dựng sẵn được tạo khi trình thông dịch Python khởi động và không bao giờ bị xóa.
  - Tên dựng sẵn thực sự cũng sống trong một mô-đun; mô-đun này gọi là **builtins**
- Không gian tên toàn cầu cho một mô-đun được tạo khi định nghĩa mô-đun được đọc vào; thông thường, không gian tên mô-đun cũng kéo dài cho đến khi trình thông dịch thoát.
- Không gian tên cục bộ cho hàm được tạo khi hàm được gọi và bị xóa khi hàm trả về giá trị, hoặc đưa ra một ngoại lệ không được xử lý trong hàm.

PHẠM VI – SCOPE VÀ KHÔNG GIAN TÊN – NAMESPACE

- Phạm vi là một vùng văn bản của chương trình Python nơi không gian tên có thể truy cập trực tiếp.
- Mặc dù phạm vi được xác định tĩnh, nhưng chúng được sử dụng linh hoạt. Bất cứ lúc nào trong khi thực hiện, có ít nhất ba phạm vi lồng nhau mà không gian tên có thể truy cập trực tiếp:
  - Phạm vi trong cùng, được tìm kiếm đầu tiên, chứa các tên địa phương
  - Phạm vi của bất kỳ hàm kèm theo nào, được tìm kiếm bắt đầu với phạm vi bao quanh gần nhất, chứa các tên không cục bộ, nhưng cũng không phải là toàn cục
  - Phạm vi kế tiếp đến cuối cùng chứa các tên toàn cầu của mô-đun hiện tại
  - Phạm vi ngoài cùng (tìm kiếm cuối cùng) là không gian tên chứa các tên dựng sẵn

```
def scope_test():
    def do_local():
        spam = "local spam"

    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"

    def do_global():
        global spam
        spam = "global spam"

    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)

scope_test()
print("In global scope:", spam)
```

```
def scope_test():
    def do_local():
        spam = "local spam"

    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"

    def do_global():
        global spam
        spam = "global spam"

    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)

scope_test()
print("In global scope:", spam)
```

```
After local assignment: test spam
After nonlocal assignment: nonlocal spam
After global assignment: nonlocal spam
In global scope: global spam
```

Lưu ý cách gán cục bộ (được mặc định) không làm thay đổi spam của hàm **scope_test**. Việc gán `nonlocal` đã thay đổi spam của **scope_testest** và việc gán toàn cục đã thay đổi liên kết cấp mô-đun

LỚP, HÀM DỰNG

- Hình thức đơn giản nhất của định nghĩa lớp (`class`) trông như sau:

```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```

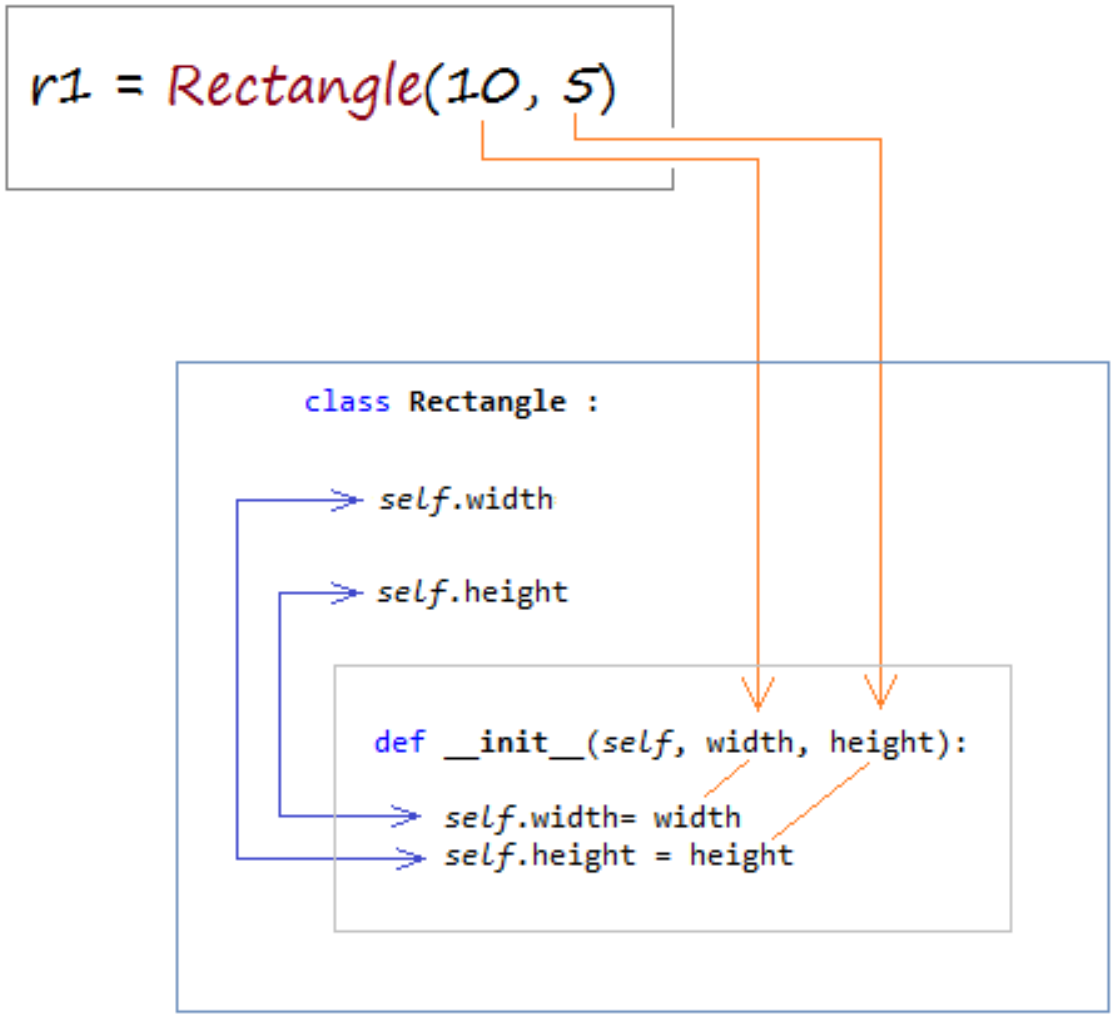
```
# Một lớp mô phỏng một hình chữ nhật.
class Rectangle :
    'This is Rectangle class'
    # Một phương (Contructor).
    def __init__(self, width, height):

        self.width = width
        self.height = height

    def getWidth(self):
        return self.width

    def getHeight(self):
        return self.height

    # Phương thức tính diện tích.
    def getArea(self):
        return self.width * self.height
```

LỚP, HÀM DỰNG

```
r1 = Rectangle(10,5)

r2 = Rectangle(20,11)

print ("r1.width = ", r1.width)
print ("r1.height = ", r1.height)
print ("r1.getWidth() = ",
r1.getWidth())
print ("r1.getArea() = ", r1.getArea())

print ("----------------")

print ("r2.width = ", r2.width)
print ("r2.height = ", r2.height)
print ("r2.getWidth() = ",
r2.getWidth())
print ("r2.getArea() = ", r2.getArea())
```

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

LỚP, HÀM DỰNG


r1 = Rectangle(10, 5)
class Rectangle :
self.width
self.height
def __init__(self, width, height):
self.width= width
self.height = height

LỚP, HÀM DỰNG

- 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.

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

```
class Person :

    # Tham số age và gender có giá trị mặc định.
    def __init__ (self, name, age = 1, gender = "Male" ):

        self.name = name
        self.age = age
        self.gender= gender


    def showInfo(self):

        print ("Name: ", self.name)
        print ("Age: ", self.age)
        print ("Gender: ", self.gender)
```

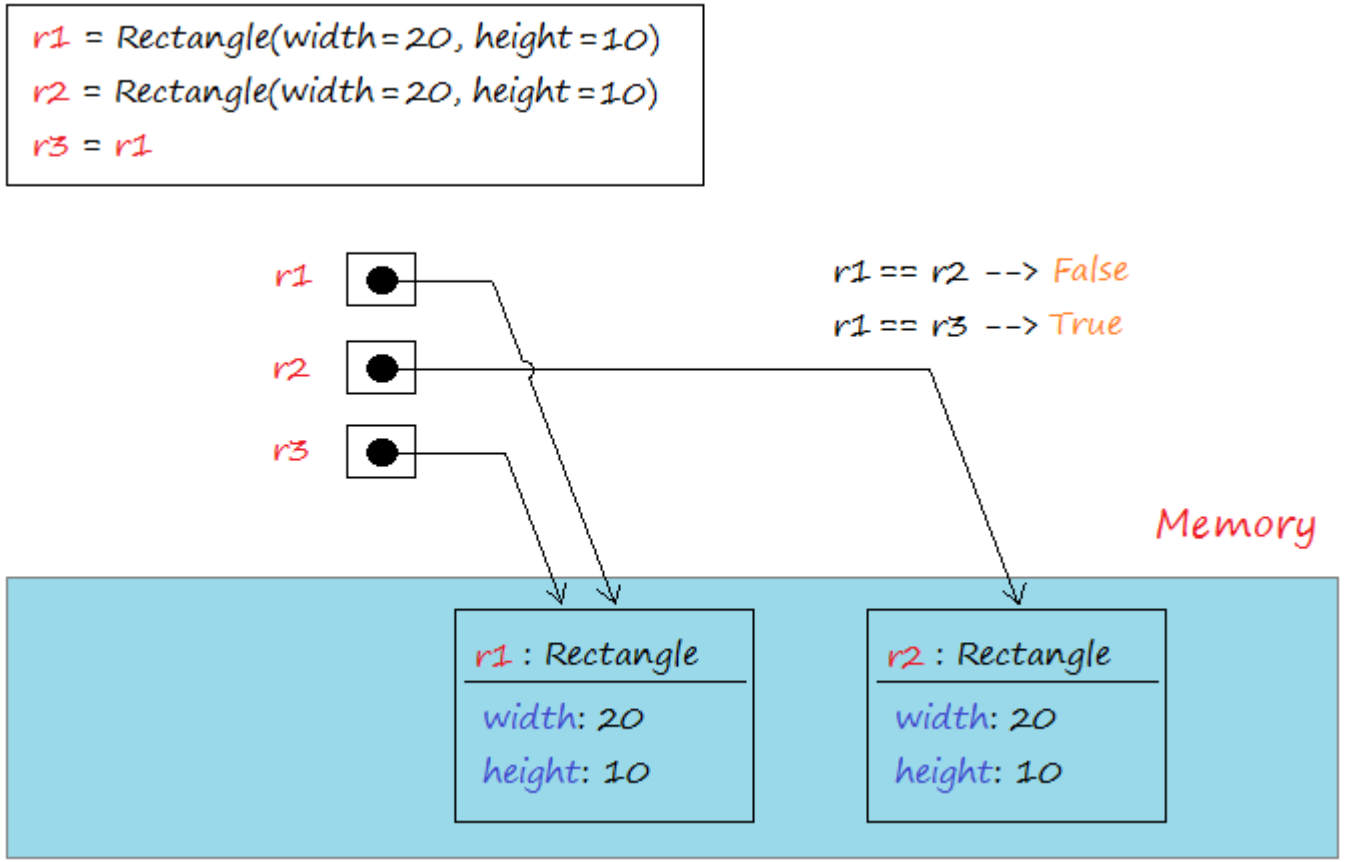
LỚP, HÀM DỰNG

```
class Person :

    # Tham số age và gender có giá trị mặc định.
    def __init__ (self, name, age = 1, gender = "Male" ):

        self.name = name
        self.age = age
        self.gender= gender


    def showInfo(self):

        print ("Name: ", self.name)
        print ("Age: ", self.age)
        print ("Gender: ", self.gender)
```

```
tom = Person("Tôm", 21, "Female")
tom.showInfo()
print (" --------------- ")

# age, gender mặc định.
cua = Person("Cua")
cua.showInfo()
print (" --------------- ")

# gender mặc định.
ca = Person("Cá", 37)
ca.showInfo()
```

LỚP, HÀM DỰNG

- So sánh đối tượng



THUỘC TÍNH - ATTRIBUTE

- Thuộc tính của đối tượng
  - Được tạo ra khi khởi tạo đối tượng
  - Mỗi đối tượng khác nhau có giá trị khác nhau, lưu trữ ở bộ nhớ khác nhau
- Biến của lớp
  - Dùng chung cho tất cả các đối tượng là thể hiện của lớp

```
class Player:

    # Biến của lớp.
    minAge = 18

    maxAge = 50

    def __init__(self, name, age):

        self.name = name  #thuộc tính
        self.age = age    #thuộc tính
```

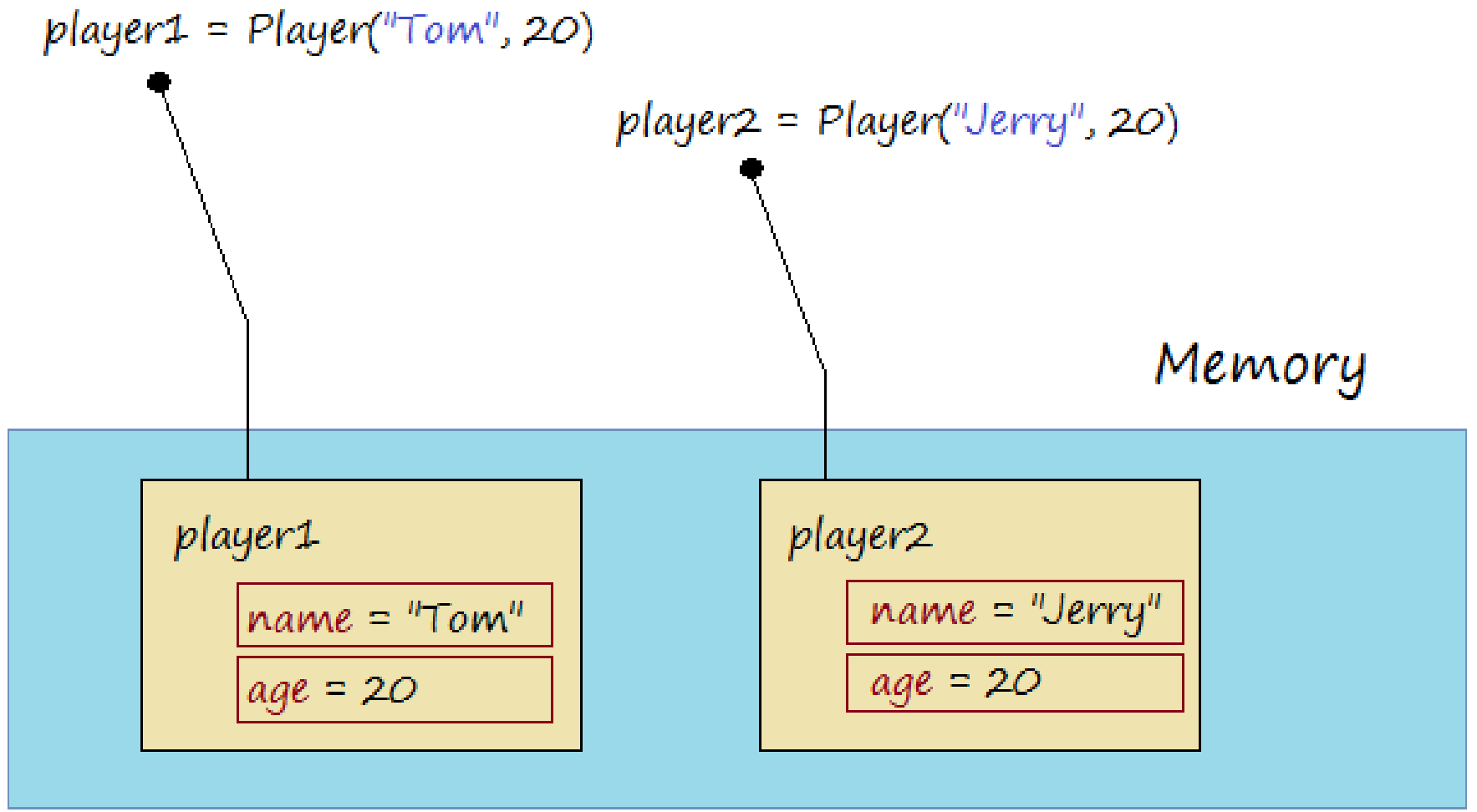
THUỘC TÍNH - ATTRIBUTE

- Thuộc tính của đối tượng
  - Được tạo ra khi khởi tạo đối tượng
  - Mỗi đối tượng khác nhau có giá trị khác nhau, lưu trữ ở bộ nhớ khác nhau
- Biến của lớp
  - Dùng chung cho tất cả các đối tượng là thể hiện của lớp

```
class Dog:

    kind = 'canine' # class variable

    def __init__(self, name):
        self.name = name    # instance variable
unique to each instance
>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.kind                  # shared by all dogs
'canine'
>>> e.kind                  # shared by all dogs
'canine'
>>> d.name                  # unique to d
'Fido'
>>> e.name                  # unique to e
'Buddy'
```

THUỘC TÍNH - ATTRIBUTE

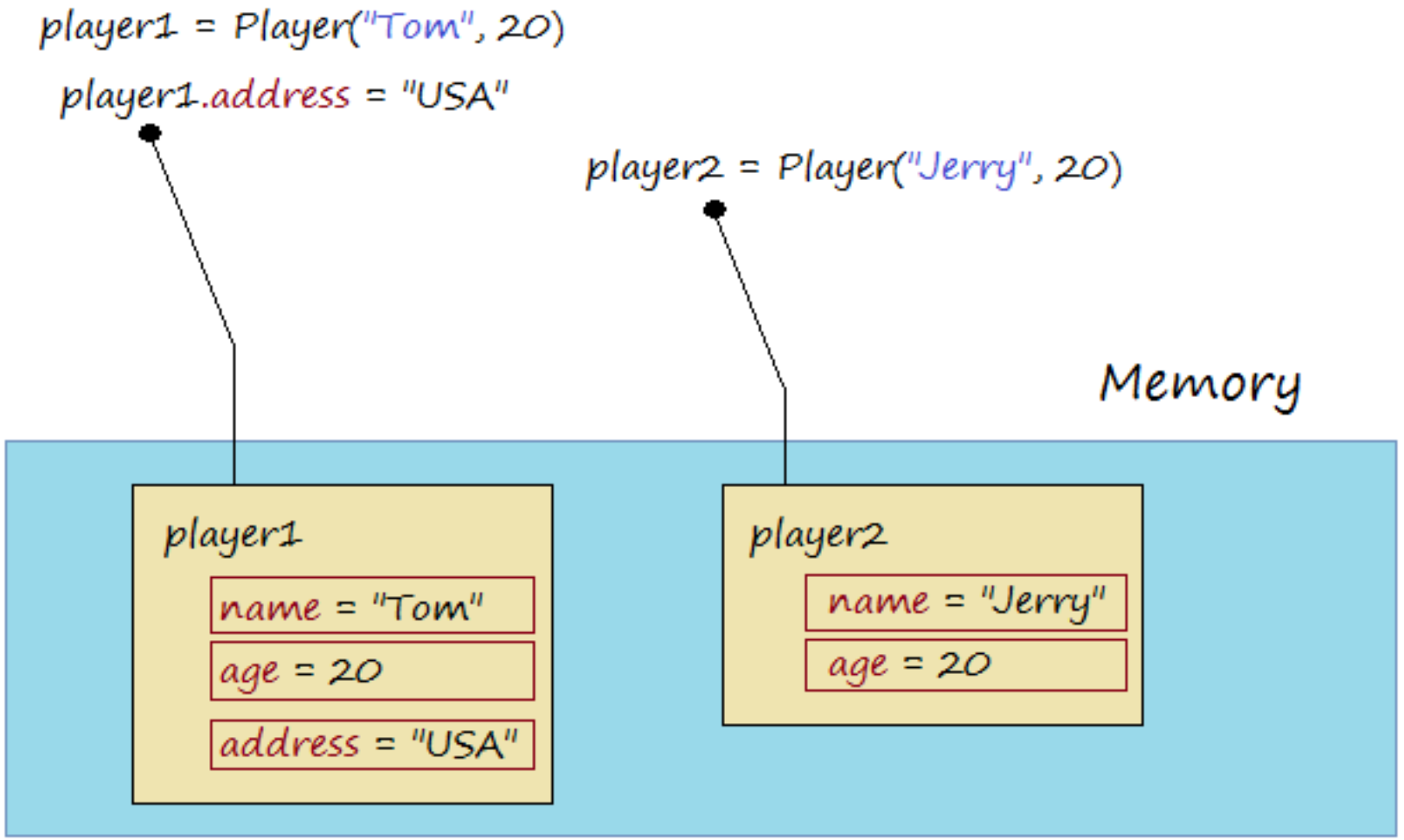
Thuộc tính của đối tượng



THUỘC TÍNH - ATTRIBUTE

Thuộc tính của đối tượng

- Python cho phép tạo ra một thuộc tính mới cho một đối tượng có trước



THUỘC TÍNH - ATTRIBUTE

Truy cập thuộc tính

- Thông thường bạn truy cập vào thuộc tính của một đối tượng thông qua toán tử "dấu chấm" (Ví dụ player1.name). Tuy nhiên Python cho phép bạn 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)	Đặt giá trị vào thuộc tính. Nếu thuộc tính không tồn tại, thì nó sẽ được tạo ra.
delattr(obj, name)	Xóa bỏ thuộc tính.

THUỘC TÍNH - ATTRIBUTE

Các lớp của Python đều kế thừa từ lớp object. Và vì vậy nó thừa kế các thuộc tính sau

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

THUỘC TÍNH - ATTRIBUTE

```
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 ("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__)
```

```
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__
```

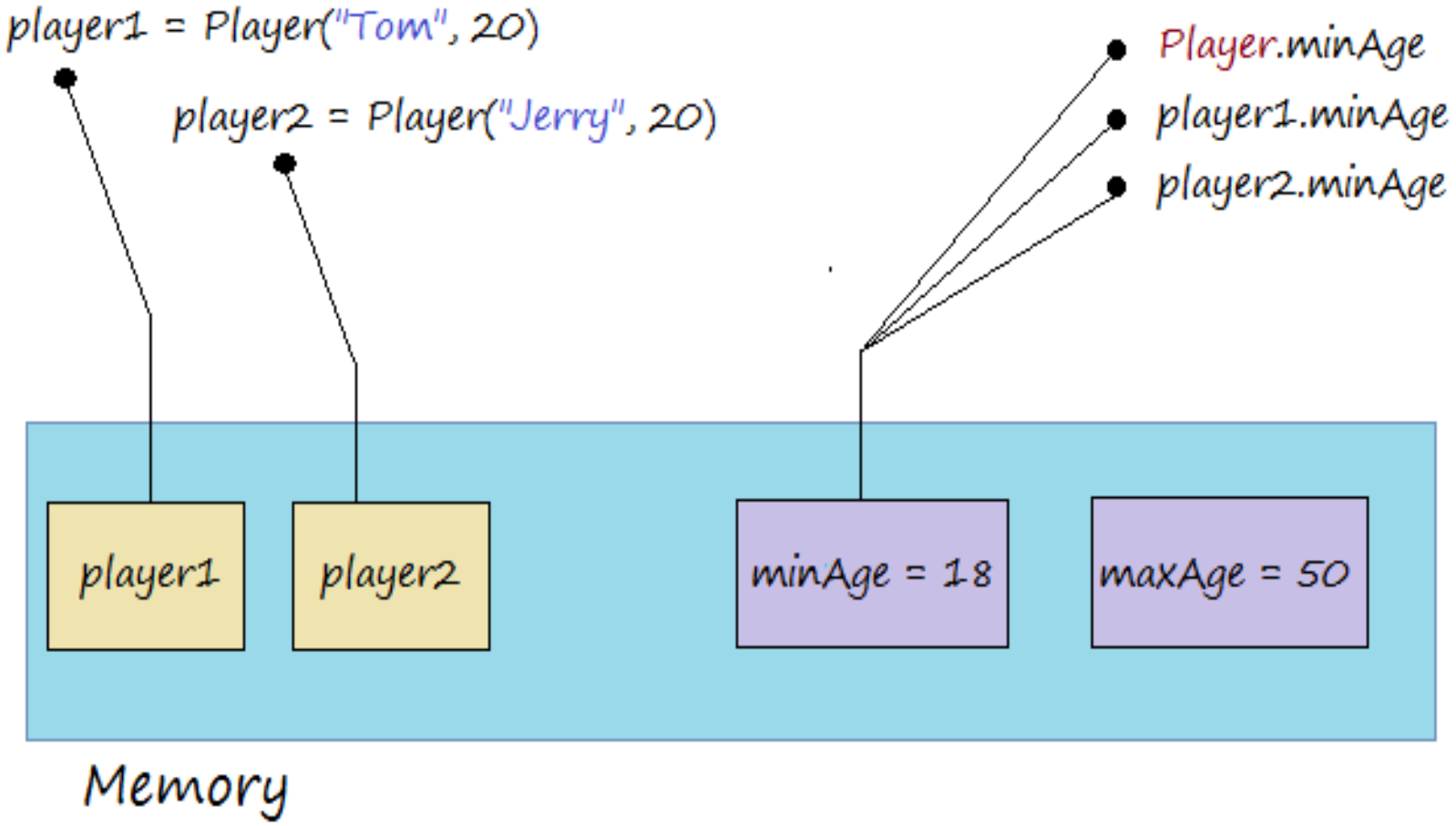
THUỘC TÍNH - ATTRIBUTE

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

```
class Player:

    # Biến của lớp.
    minAge = 18

    maxAge = 50

    def __init__(self, name, age):

        self.name = name  #thuộc tính
        self.age = age    #thuộc tính
```

THUỘC TÍNH - ATTRIBUTE

Mỗi biến của lớp, có một địa chỉ nằm trên bộ nhớ (memory). Và chia sẻ cho mọi đối tượng của lớp.



THUỘC TÍNH - ATTRIBUTE

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

```
print ( dir(Player) )
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) )
```

```
['__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']
```

PHƯƠNG THỨC - METHOD

- **Phương thức (Method)** là các hàm được định nghĩa bên trong phần thân của một lớp. Chúng được sử dụng để xác định các hành vi của một đối tượng

```
# Một lớp mô phỏng một hình chữ nhật.
class Rectangle :
    'This is Rectangle class'
    # Một phương (Contructor).
    def __init__(self, width, height):

        self.width = width
        self.height = height

    def getWidth(self):
        return self.width

    def getHeight(self):
        return self.height

    # Phương thức tính diện tích.
    def getArea(self):
        return self.width * self.height
```

PHƯƠNG THỨC - METHOD

```
r1 = Rectangle(10,5)

r2 = Rectangle(20,11)

print ("r1.width = ", r1.width)
print ("r1.height = ", r1.height)
print ("r1.getWidth() = ",
r1.getWidth())
print ("r1.getArea() = ", r1.getArea())

print ("-----------------")

print ("r2.width = ", r2.width)
print ("r2.height = ", r2.height)
print ("r2.getWidth() = ",
r2.getWidth())
print ("r2.getArea() = ", r2.getArea())
```

```
# Một lớp mô phỏng một hình chữ nhật.
class Rectangle :
    'This is Rectangle class'
    # Một phương (Contructor).
    def __init__(self, width, height):

        self.width = width
        self.height = height

    def getWidth(self):
        return self.width

    def getHeight(self):
        return self.height

    # Phương thức tính diện tích.
    def getArea(self):
        return self.width * self.height
```

TÍNH ĐÓNG GÓI - ENCAPSULATION

- Sử dụng OOP trong Python, chúng ta có thể hạn chế quyền truy cập vào trạng thái bên trong của đối tượng. Điều này ngăn chặn dữ liệu bị sửa đổi trực tiếp, được gọi là **đóng gói**.
- Trong Python, chúng ta biểu thị thuộc tính private này bằng cách sử dụng dấu gạch dưới làm tiền tố: "_" hoặc "__".

```
class Computer:
    def __init__(self):
        self.__maxprice = 900

    def sell(self):
        print("Giá bán sản phẩm:
{}".format(self.__maxprice))

    def setMaxPrice(self, price):
        self.__maxprice = price
c = Computer()
c.sell()
# thay đổi giá.
c.__maxprice = 1000
c.sell()
# sử dụng hàm setter để thay đổi giá.
c.setMaxPrice(1000)
c.sell()
```

TÍNH ĐÓNG GÓI - ENCAPSULATION

Giá bán sản phẩm: 900

Giá bán sản phẩm: 900

Giá bán sản phẩm: 1000

```
class Computer:
    def __init__(self):
        self.__maxprice = 900

    def sell(self):
        print("Giá bán sản phẩm:
{}".format(self.__maxprice))

    def setMaxPrice(self, price):
        self.__maxprice = price
c = Computer()
c.sell()
# thay đổi giá.
c.__maxprice = 1000
c.sell()
# sử dụng hàm setter để thay đổi giá.
c.setMaxPrice(1000)
c.sell()
```

KẾ THỪA - INHERITANCE

- **Tính kế thừa** cho phép một lớp (class) có thể kế thừa các thuộc tính và phương thức từ các lớp khác đã được định nghĩa.
- Lớp con kế thừa tất cả thành phần của lớp cha, có thể mở rộng các thành phần kế thừa và bổ sung thêm các thành phần mới

KẾ THỪA - INHERITANCE

```
# Lớp cha
class Car:
    # Constructor
    def __init__(self, brand, name, color):
        self.brand = brand
        self.name = name
        self.color = color

    # phương thức
    def run(self):
        print ("{} đang chạy trên đường".format(self.name))

    def stop(self, task):
        print ("{} đang dừng xe để {}".format(self.name, task))
```

KẾ THỪA - INHERITANCE

```
# Lớp Toyota mở rộng từ lớp Car.
class Toyota(Car):

    def __init__(self, brand, name, color, fuel):
 # Gọi tới constructor của lớp cha (Car)
        # để gán giá trị vào thuộc tính của lớp cha.
        super().__init__(brand, name, color)
        self.fuel = fuel

     # Ghi đè (override) phương thức cùng tên của lớp cha.
    def stop(self, task):
        print ("{} đang dừng xe để {}".format(self.name, task))
        print ("{} chạy bằng {}".format(self.name, self.fuel))

 # Bổ sung thêm thành phần mới
    def start(self):
        print ("{} đang nổ máy".format(self.name))
```

KẾ THỪA - INHERITANCE

```
toyota1 = Toyota("Toyota", "Toyota Hilux", "Đỏ", "Điện")
toyota2 = Toyota("Toyota", "Toyota Yaris", "Vàng", "Deisel")
toyota3 = Toyota("Toyota", "Toyota Vios", "Xanh", "Gas")

toyota1.stop("nạp điện")
toyota2.run()
toyota3.start()
```

```
Toyota Hilux đang dừng xe để nạp điện
Toyota Hilux chạy bằng Điện
Toyota Yaris đang chạy trên đường
Toyota Vios đang nổ máy
```

TÍNH ĐA HÌNH - POLYMORPHISM

Tính đa hình là khái niệm mà hai hoặc nhiều lớp có những phương thức giống nhau nhưng có thể thực thi theo những cách thức khác nhau.

- Về cơ bản, tính đa hình trong Python thể hiện rất rõ trong cơ chế kiểu dữ liệu động.

TÍNH ĐA HÌNH - POLYMORPHISM

```
class Toyota:

    def stop(self):
        print("Toyota dừng xe để nạp điện")

    def start(self):
        print("Toyota nổ máy bằng hộp số tự động")

class Porsche:

    def stop(self):
        print("Porsche dừng xe để bơm xăng")

    def start(self):
        print("Porsche nổ máy bằng hộp số cơ")
```

```
# common interface
def carStop(car):
    car.stop()

# instantiate objects
toyota = Toyota()
porsche = Porsche()

# passing the object
carStop(toyota)
carStop(porsche)
```

```
Toyota dừng xe để nạp điện
Porsche dừng xe để bơm xăng
```

Hết bài