**Class và đối tượng trong Python**

1- [Hướng đối tượng trong Python](http://o7planning.org/vi/11415/class-va-doi-tuong-trong-python#a7505463)

2- [Tạo class trong Python](http://o7planning.org/vi/11415/class-va-doi-tuong-trong-python#a7505729)

3- [Tham số có mặc định trong Constructor](http://o7planning.org/vi/11415/class-va-doi-tuong-trong-python#a7509145)

4- [So sánh các đối tượng](http://o7planning.org/vi/11415/class-va-doi-tuong-trong-python#a7512326)

5- [Thuộc tính (Attribute)](http://o7planning.org/vi/11415/class-va-doi-tuong-trong-python#a7520251)

6- [Các hàm truy cập vào thuộc tính](http://o7planning.org/vi/11415/class-va-doi-tuong-trong-python#a7524326)

7- [Các thuộc tính có sẵn của class](http://o7planning.org/vi/11415/class-va-doi-tuong-trong-python#a7513063)

8- [Biến của lớp](http://o7planning.org/vi/11415/class-va-doi-tuong-trong-python#a7525418)

9- [Liệt kê danh sách các thành viên của lớp hoặc đối tượng](http://o7planning.org/vi/11415/class-va-doi-tuong-trong-python#a7530899)

10- [Hủy đối tượng](http://o7planning.org/vi/11415/class-va-doi-tuong-trong-python#a7512817)

# 1- Hướng đối tượng trong Python

**Python** là một ngôn ngữ lập trình hướng thủ tục (Procedural-oriented), đồng thời nó cũng là ngôn ngữ lập trình hướng đối tượng (Object Oriented).

### Hướng thủ tục (Procedural-oriented)

Hướng thủ tục biểu hiện ở việc sử dụng các hàm trong **Python**. Bạn có thể định nghĩa các hàm, và các hàm này có thể sử dụng tại các module khác trong chương trình **Python**.

### Hướng đối tượng (Object Oriented)

Hướng đối tượng trong **Python** biểu hiện ở việc sử dụng lớp (class), bạn có thể định nghĩa một class, class là một nguyên mẫu (prototype) để tạo ra các đối tượng (object/instance).

# 2- Tạo class trong Python

Cú pháp tạo một class:

\*\* class syntax \*\*

class ClassName:

'Mô tả ngắn về class (Không bắt buộc)'

# Code ...

* Để định nghĩa một lớp bạn sử dụng từ khóa **class**, tiếp sau đó là tên của lớp và dấu hai chấm ( : ). Dòng đầu tiên trong thân của lớp là chuỗi (string) mô tả ngắn gọn về lớp này (Không bắt buộc), bạn có thể truy cập vào chuỗi này thông qua ***ClassName.\_\_doc\_\_*** .
* Trong thân của lớp bạn có thể khai báo các thuộc tính, phương thức (method) và các phương thức khởi tạo (Constructor).

### Thuộc tính (Attribute):

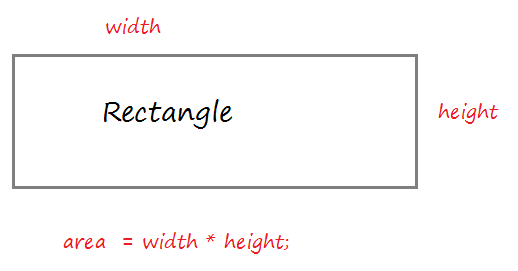
Thuộc tính là một thành viên thành viên của lớp. Chẳng hạn hình chữ nhật có hai thuộc tính **width** và **height** (Chiều rộng và chiều cao).

### 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).
* 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.
* Bạn chỉ 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 kết từ constructor của lớp cha.



rectangle.py

# Class mô phỏng một hình chữ nhật.

class Rectangle :

'This is Rectangle class'

# Phương thức tạo đối tượng (Contructor)

def \_\_init\_\_(self, width, height):

self.width= width

self.height = height

def getWidth(self):

return self.width

def getHeight(self):

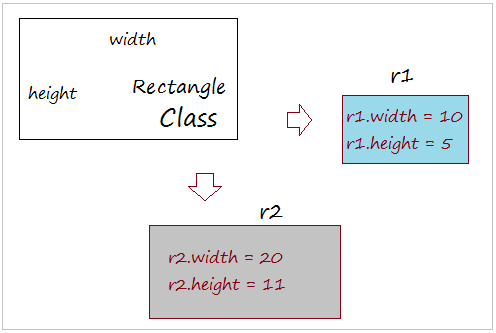
return self.height

# Phương thức tính diện tích

def getArea(self):

return self.width \* self.height

Tạo đối tượng từ lớp **Rectangle**:



from rectangle import Rectangle

# Tạo 2 đối tượng: r1 & r2

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())

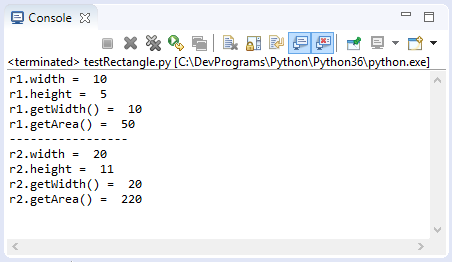
print ("-----------------")

print ("r2.width = ", r2.width)

print ("r2.height = ", r2.height)

print ("r2.getWidth() = ", r2.getWidth())

print ("r2.getArea() = ", r2.getArea())



### Điều gì xẩy ra khi bạn tạo đối tượng từ một class?

Khi bạn 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:

# 3- 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.

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

person.py

class Person :

# Tham số age, 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)

Ví dụ sử dụng:

testPerson.py

from person import Person

# Tạo một đối tượng Person

aimee = Person("Aimee", 21, "Female")

aimee.showInfo()

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

# age, gender mặc định

alice = Person( "Alice" )

alice.showInfo()

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

# gender mặc định

tran = Person("Tran", 37)

tran.showInfo()

# 4- So sánh các đối tượng

Trong **Python**, khi bạn 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.

compareObject.py

from rectangle import Rectangle

r1 = Rectangle(20, 10)

r2 = Rectangle(20 , 10)

r3 = r1

# So sánh địa chỉ của r1 và r2

test1 = r1 == r2 # --> False

# So sánh địa chỉ của r1 và r3

test2 = r1 == r3 # --> True

print ("r1 == r2 ? ", test1)

print ("r1 == r3 ? ", test2)

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

print ("r1 != r2 ? ", r1 != r2)

print ("r1 != r3 ? ", r1 != r3)

# 5- Thuộc tính (Attribute)

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

1. Thuộc tính (Attribute)
2. Biến của lớp

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

player.py

class Player:

# Biến của lớp

minAge = 18

maxAge = 50

def \_\_init\_\_(self, name, age):

self.name = name

self.age = age

### Thuộc tính (Attribute)

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. Như hình minh họa dưới đây:

testAttributePlayer.py

from player import Player

player1 = Player("Tom", 20)

player2 = Player("Jerry", 20)

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 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)

**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***.

testNewAttributePlayer.py

from player import Player

player1 = Player("Tom", 20)

player2 = Player("Jerry", 20)

# Tạo một thuộc tính có tên address cho player1

player1.address = "USA"

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 xẩy ra tại đây)

print ("player2.address = ", player2.address)

# 6- Các hàm truy cập vào 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) | 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. |

testAttFunctions.py

from player import Player

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 address?

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 bỏ thuộc tính 'address'

delattr(player1, "address")

# 7- 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** |
| \_\_dict\_\_ | Thông tin về class này được biểu diễn 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. |

testBuildInAttributes.py

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\_\_)

# 8- 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.

*Lời khuyên là bạn 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.*

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

testVariablePlayer.py

from player import Player

player1 = Player("Tom", 20)

player2 = Player("Jerry", 20)

# 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..")

# Thay đổi giá trị biến minAge thông qua tên lớp

Player.minAge = 19

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

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

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

# 9- 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.

testDirFunction.py

from player import Player

# In ra danh sách các thuộc tính, phương thức, biến của class Player.

print ( dir(Player) )

print ("\n\n")

player1 = Player("Tom", 20)

player1.address ="USA"

# In ra danh sách các thuộc tính, phương thức, biến của đối tượng 'player1'

print ( dir(player1) )

Chạy ví dụ:

['\_\_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']

# 10- Hủy đối tượng