# OBJECT & CLASS

## Module trong Python: Cách tạo và sử dụng Module có sẵn

Trong bài này chúng ta sẽ học cách tạo **module trong Python**, qua bài này bạn sẽ hiểu được khái niệm module là gì? Làm thế nào để gọi đến một module tạo mới hoặc module có sẵn trong Python.

Nếu Python là ngôn ngữ đầu tiên mà bạn học thì khái niệm này hơi mới lạ, nhưng nếu bạn đã học ngôn ngữ khác rồi thì không còn xa lạ gì khái niệm lập trình hướng module nữa. Vậy Module là gì thì chúng ta cùng tìm hiểu ngay nhé.

**Mục lục**

* [1. Module trong Python là gì?](https://freetuts.net/module-trong-python-2058.html#goto-h2-0)
* [2. Cách tạo module trong Python](https://freetuts.net/module-trong-python-2058.html#goto-h2-1)
* [3. Lệnh import trong Python](https://freetuts.net/module-trong-python-2058.html#goto-h2-2)
  + [Sử dụng import](https://freetuts.net/module-trong-python-2058.html#goto-h3-0)
  + [Sử dụng from-import](https://freetuts.net/module-trong-python-2058.html#goto-h3-1)
* [4. Đổi tên module với AS trong Python](https://freetuts.net/module-trong-python-2058.html#goto-h2-3)
* [5. Xem thuộc tính và phương thức của module](https://freetuts.net/module-trong-python-2058.html#goto-h2-4)
* [6. Tải lại module với hàm reload](https://freetuts.net/module-trong-python-2058.html#goto-h2-5)
* [7. Phạm vi của biến của module trong Python](https://freetuts.net/module-trong-python-2058.html#goto-h2-6)
* [8. Xem các module có sẵn trong Python](https://freetuts.net/module-trong-python-2058.html#goto-h2-7)

### Module trong Python là gì?

Trong Python, modules là những file có phần mở rộng .py, các file này chứa mã Python và đó có thể là các biến, hàm, hoặc một lớp nào đó.

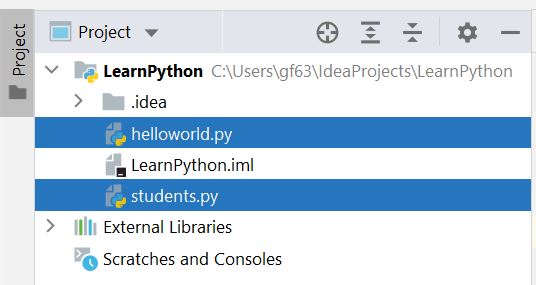
Các module giúp chúng ta code một cách linh hoạt hơn, mỗi file sẽ chứa những đoạn code phục vụ cho một chức năng cụ thể, được sắp xếp tùy vào cách thiết kế của mỗi lập trình viên.

Để sử dụng các hàm của module A trong module B thì ta phải sử dụng từ khóa import, sau đó là bạn có thể sử dụng toàn bộ tài nguyên của module A. Cách import như thế nào thì chúng ta cùng tìm hiểu qua phần thứ hai nhé.

### Cách tạo module trong Python

Trước khi tạo một module thì bạn phải xác định rằng nhu cầu cần tạo một module tên gì? nằm phục vụ một công việc gì? Để từ đó đặt tên module cho có ngữ nghĩa.

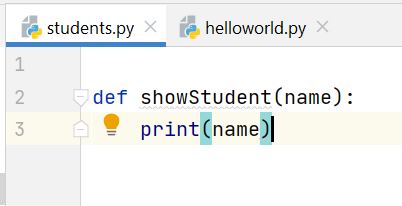
Giả sử mình cần tạo một thư viện chứa những hàm dùng để quản lý sinh viên, lúc này mình sẽ tạo một file tên là students.py cùng cấp với file chạy chính như sau:



Như bạn thấy, mình có file helloworld.py là file chương trình chính của project, còn file students.py là module chứa những hàm quản lý sinh viên mà minh sắp viết vào.

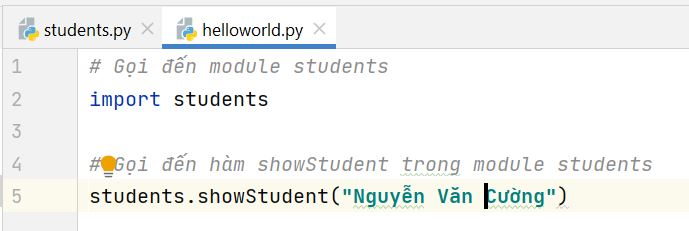
Trong file students.py bạn hãy dùng lệnh def để tạo một hàm như sau:

|  |  |
| --- | --- |
| 1  2 | def showStudent(name):      print(name) |

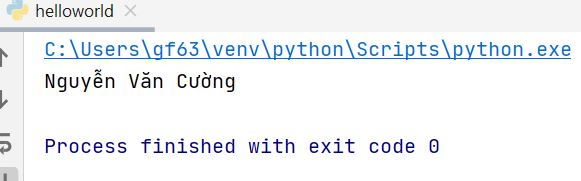


Trong file helloworld.py bạn hãy viết đoạn code sau:

|  |  |
| --- | --- |
| 1  2  3  4  5 | # Gọi đến module students  import students    # Gọi đến hàm showStudent trong module students  students.showStudent("Nguyễn Văn Cường") |



Chạy file chính helloworld.py lên thì bạn sẽ thấy kết quả như sau:



**Tóm lại**: Giả sử trong module stu**d**ents có biến **name**và hàm **showMessage**, lúc này ta sẽ gọi đến biến và hàm này như sau:

|  |  |
| --- | --- |
| 1  2 | students.name  students.showMessage() |

Như vậy mình đã sử dụng lệnh **import**để gọi đến module students. Ta sẽ tìm hiểu sâu hơn về lệnh này ở phần tiếp theo.

### Lệnh import trong Python

Để sử dụng các đoạn code ở module A vào trong module B thì ta sẽ phải sử dụng lệnh **import**. Có hai cách sử dụng, thứ nhất là dùng lệnh **import**đơn lẻ, thứ hai là lệnh **from-import**.

#### Sử dụng import

Lệnh import sẽ gọi module A vào trong module B, lúc này bạn có thể sử dụng mọi tài nguyên của module A trong module B.

|  |  |
| --- | --- |
| 1 | import module |

Nếu bạn muốn import một lúc nhiều module thì hãy ngăn chúng bằng dấu phẩy.

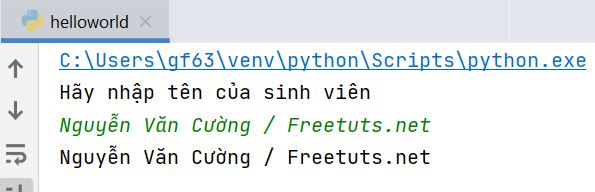
|  |  |
| --- | --- |
| 1 | import module1,module2,........ module n |

**Ví dụ**: Sử dụng hàm **showStudent**trong module **students**.

**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | # Gọi đến module students  import students    # Lấy dữ liệu  print("Hãy nhập tên của sinh viên")  name = input()    # Gọi đến hàm showStudent  students.showStudent(name) |

Chạy chương trình này bạn sẽ thu được kết quả như sau:



#### Sử dụng from-import

Giả sử trong module A bạn định nghĩa 10 function, nhưng trong module B bạn chỉ muốn sử dụng 1 trong 10 funciton đó thôi thì sử dụng from .. import.

Vậy sự khác nhau giữa import và from ... import là một bên sẽ gọi tất cả các function, còn một bên chỉ gọi một function nào đó thôi.

**Cú pháp**

|  |  |
| --- | --- |
| 1 | from < module-name> import <name 1>, <name 2>..,<name n> |

**Ví dụ**: Xem cách sử dụng from-import dưới đây.

**calculation.py**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | #Đoạn code trong file calculation.py  def summation(a,b):      return a+b  def multiplication(a,b):      return a\*b  def divide(a,b):      return a/b |

**Main.py**

|  |  |
| --- | --- |
| 1  2  3  4  5 | from calculation import summation  # Nó sẽ import duy nhất hàm summation() trong file calculation.py  a = int(input("Nhập số thứ nhất: "))  b = int(input("Nhập số thứ hai: "))  print("Sum = ",summation(a,b)) |

Chạy lên kết quả sẽ như sau:

|  |  |
| --- | --- |
| 1  2  3 | Nhập số thứ nhất: 10  Nhập số thứ hai: 20  Sum =  30 |

Lệnh from ... import sử dụng trong trường hợp bạn biết chính xác tên function muốn sử dụng trong module. Nó sẽ không khiến chương trình nặng hơn, vì vậy cứ yên tâm mà sử dụng.

Trường hợp bạn muốn import tất cả các function thì sử dụng dấu sao \*.

|  |  |
| --- | --- |
| 1 | from <module> import \* |

### Đổi tên module với AS trong Python

Nếu bạn muốn đổi tên module cho ngắn gọn và dễ hiểu thì có thể sử dụng từ khóa AS. Từ khóa này rất hữu ích vì giúp bạn tiết kiệm được thời gian nhập những module có tên quá dài, thay vao đó chỉ cần một cai tên thật đặc biệt.

**Cú pháp như sau:**

|  |  |
| --- | --- |
| 1 | import <module-name> as <specific-name> |

**Ví dụ**: Thay đổi tên module **calculation**thành **cal**

|  |  |
| --- | --- |
| 1  2  3  4  5 | #Tên của module calculation ở ví dụ trước sẽ đổi thành cal.  import calculation as cal  a = int(input("Enter a?"))  b = int(input("Enter b?"))  print("Sum = ",cal.summation(a,b)) |

### Xem thuộc tính và phương thức của module

Hàm dir() có công dụng trả về danh sách các thuộc tính và phương thức của module.

**Ví dụ**: Xem những tên biến, hàm có sẵn trong module **json**.

|  |  |
| --- | --- |
| 1  2  3  4  5 | import json    List = dir(json)    print(List) |

**Kết quả**:

|  |
| --- |
| ['JSONDecoder', 'JSONEncoder', '\_\_all\_\_', '\_\_author\_\_', '\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_',  '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_path\_\_', '\_\_spec\_\_', '\_\_version\_\_',  '\_default\_decoder', '\_default\_encoder', 'decoder', 'dump', 'dumps', 'encoder', 'load', 'loads', 'scanner'] |

### Tải lại module với hàm reload

Trong Phython, khi bạn import một module thì nó sẽ thực hiện một lần duy nhất, cho dù bạn sử dụng đoạn code import bao nhiêu lần đi nữa. Tuy nhiên có một số trường hợp bạn muốn tải lại dữ liệu mới nhất của module đó thì có thể sử dụng hàm reload().

|  |  |
| --- | --- |
| 1 | reload(<module-name>) |

**Ví dụ**: Tải lại module calculation đã được định nghĩa ở các ví dụ trước.

|  |  |
| --- | --- |
| 1 | reload(calculation) |

### Phạm vi của biến của module trong Python

Như tất cả các ngôn ngữ lập trình khác, chúng ta có hai dạng biến như sau:

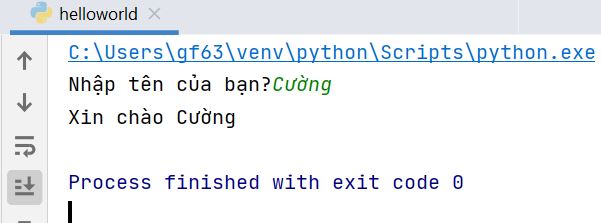
* **Biến toàn cục**: là biến có thể sử dụng ở tất cả các vị trí trong chương trình chính, trừ trong hàm.
* **Biến cục bộ**: Là biến chỉ sử dụng được trong một phạm vi nhất định, ví dụ trong hàm.

Vậy khi bạn khai báo một biến trong module thì có thể sử dụng tai mọi vị trí trong module đó. Tuy nhiên trong hàm thì không thể gọi đến biến cục bộ đó.

Hãy xem ví dụ dưới đây, biến name đã được khai báo ở cấp ngoài cùng và trong hàm, đây là 2 biến hoàn toàn khác nhau.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | # Đây là biến toàn cục  name = "john"  def print\_name(name):      # Biến này là biến cục bộ, giá trị của nó được truyền vào      print("Xin chào",name)  name = input("Nhập tên của bạn?")  print\_name(name) |

**Kết quả**:



### Xem các module có sẵn trong Python

Để xem danh sách các module có sẵn trong Python thì ta sử dụng lệnh sau:

|  |  |
| --- | --- |
| 1 | print (help('modules') ) |

Kết quả nó sẽ trả về danh sách như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76 | \_\_future\_\_          \_threading\_local    getpass             sched  \_abc                \_tkinter            gettext             secrets  \_aix\_support        \_tracemalloc        glob                select  \_ast                \_uuid               graphlib            selectors  \_asyncio            \_warnings           gzip                setuptools  \_bisect             \_weakref            hashlib             shelve  \_blake2             \_weakrefset         heapq               shlex  \_bootlocale         \_winapi             helloworld          shutil  \_bootsubprocess     \_xxsubinterpreters  hmac                signal  \_bz2                \_zoneinfo           html                site  \_codecs             abc                 http                smtpd  \_codecs\_cn          aifc                idlelib             smtplib  \_codecs\_hk          antigravity         imaplib             sndhdr  \_codecs\_iso2022     argparse            imghdr              socket  \_codecs\_jp          array               imp                 socketserver  \_codecs\_kr          ast                 importlib           sqlite3  \_codecs\_tw          asynchat            inspect             sre\_compile  \_collections        asyncio             io                  sre\_constants  \_collections\_abc    asyncore            ipaddress           sre\_parse  \_compat\_pickle      atexit              itertools           ssl  \_compression        audioop             json                stat  \_contextvars        base64              keyword             statistics  \_csv                bdb                 lib2to3             string  \_ctypes             binascii            linecache           stringprep  \_ctypes\_test        binhex              locale              struct  \_datetime           bisect              logging             students  \_decimal            builtins            lzma                subprocess  \_distutils\_hack     bz2                 mailbox             sunau  \_elementtree        cProfile            mailcap             symbol  \_functools          calendar            marshal             symtable  \_hashlib            cgi                 math                sys  \_heapq              cgitb               mimetypes           sysconfig  \_imp                chunk               mmap                tabnanny  \_io                 cmath               modulefinder        tarfile  \_json               cmd                 msilib              telnetlib  \_locale             code                msvcrt              tempfile  \_lsprof             codecs              multiprocessing     test  \_lzma               codeop              netrc               textwrap  \_markupbase         collections         nntplib             this  \_md5                colorsys            nt                  threading  \_msi                compileall          ntpath              time  \_multibytecodec     concurrent          nturl2path          timeit  \_multiprocessing    configparser        numbers             tkinter  \_opcode             contextlib          opcode              token  \_operator           contextvars         operator            tokenize  \_osx\_support        copy                optparse            trace  \_overlapped         copyreg             os                  traceback  \_peg\_parser         crypt               parser              tracemalloc  \_pickle             csv                 pathlib             tty  \_py\_abc             ctypes              pdb                 turtle  \_pydecimal          curses              pickle              turtledemo  \_pyio               dataclasses         pickletools         types  \_queue              datetime            pip                 typing  \_random             dbm                 pipes               unicodedata  \_sha1               decimal             pkg\_resources       unittest  \_sha256             difflib             pkgutil             urllib  \_sha3               dis                 platform            uu  \_sha512             distutils           plistlib            uuid  \_signal             doctest             poplib              venv  \_sitebuiltins       easy\_install        posixpath           warnings  \_socket             email               pprint              wave  \_sqlite3            encodings           profile             weakref  \_sre                ensurepip           pstats              webbrowser  \_ssl                enum                pty                 winreg  \_stat               errno               py\_compile          winsound  \_statistics         faulthandler        pyclbr              wsgiref  \_string             filecmp             pydoc               xdrlib  \_strptime           fileinput           pydoc\_data          xml  \_struct             fnmatch             pyexpat             xmlrpc  \_symtable           formatter           queue               xxsubtype  \_testbuffer         fractions           quopri              zipapp  \_testcapi           ftplib              random              zipfile  \_testconsole        functools           re                  zipimport  \_testimportmultiple gc                  reprlib             zlib  \_testmultiphase     genericpath         rlcompleter         zoneinfo  \_thread             getopt              runpy |

Mình chạy trên **Python 3.8**.

Như vậy là mình đã hướng dẫn xong cách sử dụng lệnh import trong Python để gọi đến một module nào đó. Ngoài ra bạn cũng biết cách tạo mới module và gọi đến một module vừa tạo đó.

## Package trong Python: Có gì khác với module

Trong bài này chúng ta sẽ tìm hiểu package trong Python, đây là phương pháp quản lý các module theo package, giúp việc xây dựng cấu trúc của ứng dụng Python trở nên đơn giản hơn rất nhiều.

Trong bài [module](https://freetuts.net/module-trong-python-2058.html) mình đã hướng dẫn bạn cách tạo một module đơn giản trong Python, và nó nằm cùng cấp thư mục với file chính của chương trình. Giả sử trường hợp bạn tạo ra 100 modules thì phải tạo ra 100 files, như vậy việc quản lý các modules sẽ rất phức tạp.

Để đơn giản hóa thì trong Python xuất hiện thêm khái niệm package. Chi tiết thế nào thì chúng ta cùng tìm hiểu ngay nhé.

**Mục lục**

* [1. Package trong Python là gì?](https://freetuts.net/package-trong-python-3481.html#goto-h2-0)
* [2. Thực hành tạo một package trong Python](https://freetuts.net/package-trong-python-3481.html#goto-h2-1)
* [3. Tạo module trong Package Python](https://freetuts.net/package-trong-python-3481.html#goto-h2-2)
* [4. Tạo package bên trong một package khác](https://freetuts.net/package-trong-python-3481.html#goto-h2-3)

### Package trong Python là gì?

Package là cách tạo ra các gói và mỗi gói sẽ chứa những module liên quan với nhau. Phương pháp này giúp việc phân chia modules trong Python một cách dễ dàng.

Bạn có thể hiểu một package giống như folder trong máy tính vậy. Bên trong folder này có thể chứa thêm một folder khác (tức là package khác), và cũng có thể chứa thêm những file khác (tức là các modules).

Hình ảnh dưới đây là cấu trúc của package **Game**.



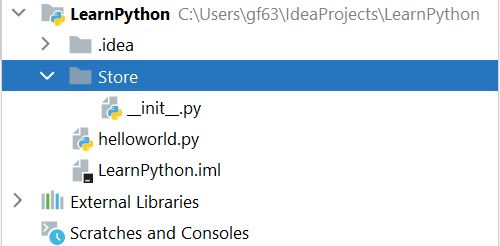
**Như bạn thấy**:

* Bên trong Game có thêm 3 package con là **Sound**, **Image**, **Level**
* Bên trong 3 package con này chứa các module có liên quan với nhau.

Điều đặc biệt là mỗi package **bắt buộc** phải có file \_\_init\_\_.py, đây là file được gọi đầu tiên mỗi khi bạn import package nó vào chương trình. Nếu bạn đã học lập trình hướng đối tượng thì file này giống như **hàm khởi tạo** của một lớp vậy đó.

### Thực hành tạo một package trong Python

Giả sử mình cần tạo package có tên là **Store**.



Như bạn thấy trong hình, mình đã tạo một folder Store và một file \_\_init\_\_.py, bây giờ mình sẽ viết code trong file này như sau:

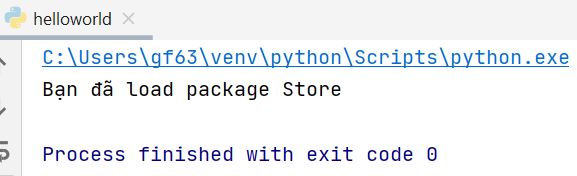
**Store/\_\_init\_\_.py**

|  |  |
| --- | --- |
| 1 | print("Bạn đã load package Store") |

Để gọi đến package này trong file helloworld.py thì ta sử dụng lệnh sau

|  |  |
| --- | --- |
| 1 | import Store |

Chạy lên bạn sẽ nhận được kết quả như sau:

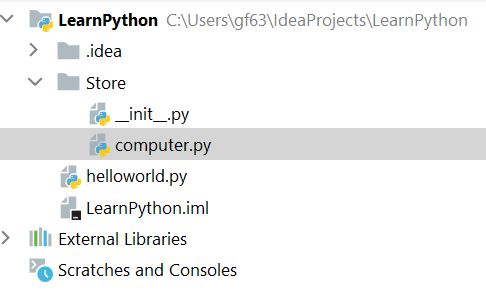


Như vậy đoạn code trong file \_\_init\_\_.py đã tự động được gọi.

### Tạo module trong Package Python

Để tạo một module nằm trong một package A thì bạn chỉ việc tạo file module\_name.py nằm trong thư mục package A là được.

Giả sử mình cần tạo module computer.py nằm trong package Store thì cấu trúc sẽ như sau:



Tạo một hàm trong package **computer** như sau:

|  |  |
| --- | --- |
| 1  2 | def showComputerMsg(msg):      print(msg) |

Bây giờ để sử dụng package computer trong file helloworld.py thì ta sử dụng lệnh **import**như sau:

|  |  |
| --- | --- |
| 1  2 | import Store.computer  Store.computer.showComputerMsg('Máy tính của Freetuts.net') |

Hoặc sử dụng lệnh **from ... import** như sau:

|  |  |
| --- | --- |
| 1  2 | from Store import computer  computer.showComputerMsg('Máy tính của Freetuts.net') |

### Tạo package bên trong một package khác

Package trong Python có thể tạo chồng lên nhau. Ví dụ trong package A bạn muốn chia thành 2 package nhỏ là B và C thì hoàn toàn được, bằng cách sử dụng cấu trúc sub folder như phần 1.

Điều lưu ý duy nhất là mỗi package bắt buộc phải có file \_\_init\_\_.py nhé.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | packageA      \_\_init\_\_.py      packageB          \_\_init\_\_.py      packageC          \_\_init\_\_.py |

Và để truy xuất đến các package này thì chỉ việc sử dụng dấu chấm để nối giữa các package là được.

**Ví dụ**: Giả sử mình có cấu trúc package như sau trong một ứng dụng Python.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | Store      \_\_init\_\_.py      Computer          \_\_init\_\_.py          Ram              \_\_init\_\_.py              raminfor.py |

Lúc này để gọi đến module raminfor thì ta sử dụng cú pháp sau:

|  |  |
| --- | --- |
| 1 | import Store.Computer.Ram.raminfor |

Các cách gọi khác thì tương tự như phần 3 nhé.

Như vậy là qua bài này bạn đã biết cách sử dụng package trong Python rồi nhỉ. Mình hy vọng bạn hiểu được các vấn đề sau:

* Cách tạo và gọi đến các package
* Hiểu ý nghĩa của file **\_\_init\_\_.py trong Python**
* Và nhiều kiến thức liên quan khác.

## Class trong Python (lập trình hướng đối tượng OOP)

Trong bài này mình sẽ hướng dẫn các bạn cách tạo **class trong Python**, và các kiến thức liên quan như hàm khởi tạo - tính kế thừa - các mực độ truy cập public / private và public.

Nói đến class thì không thể không nhắc đến đối tượng, và cách tốt nhất là liên tưởng đến những đối tượng và sự vật trong thực tế. Hãy cùng minh tìm hiểu ngay nhé!

**Mục lục**

* [1. Lập trình hướng đối tượng là gì?](https://freetuts.net/class-trong-python-3482.html#goto-h2-0)
* [2. Class trong Python là gì?](https://freetuts.net/class-trong-python-3482.html#goto-h2-1)
* [3. self của class trong Python](https://freetuts.net/class-trong-python-3482.html#goto-h2-2)
* [4. Tạo một instance object trong Python](https://freetuts.net/class-trong-python-3482.html#goto-h2-3)
* [5. Các loại phương thức của class trong Python](https://freetuts.net/class-trong-python-3482.html#goto-h2-4)
  + [Phương thức khởi tạo](https://freetuts.net/class-trong-python-3482.html#goto-h3-0)
  + [Phương thức tĩnh static](https://freetuts.net/class-trong-python-3482.html#goto-h3-1)
* [6. Kế thừa class trong Python](https://freetuts.net/class-trong-python-3482.html#goto-h2-5)
* [7. Mức độ truy cập public / protected và private](https://freetuts.net/class-trong-python-3482.html#goto-h2-6)

### Lập trình hướng đối tượng là gì?

Lập trình hướng đối tượng (object-oriented programming) là phương pháp lập trình phân chia theo từng đối tượng, nó khác hoàn toàn với lập trình hướng module, thay vì mỗi module sẽ sử dụng hàm thì OOP sẽ được quản lý trong một lớp đối tượng.

Mỗi đối tượng sẽ có hai nhóm thông tin chính, thứ nhất là các **thuộc tính**, thứ hai là những **hành động**. Ví dụ đối tượng **Con Chó** thì nó có các thông tin như chủng loại, màu sắc, cân nặng, ... và các hành động như ăn uống, đi vệ sinh, chạy nhảy ...

Đây là đối tượng trong thực tế, còn đối tượng trong lập trình thì bạn phải quy về một bài toán cụ thể.

**Vi dụ**: Bạn cần viết chương trình quản lý sinh viên thì hãy xem sinh viên là một đối tượng, nó có các thông tin như tên sinh viên, năm sinh, tuổi, học khoa nào ... và có các hành động trên đối tượng này như thêm sinh viên, xóa sinh viên, sửa sinh viên.

### Class trong Python là gì?

Class là một lớp trong Puython, dùng để khai báo cấu trúc thuộc tính và hành động cho một đối tượng nào đó. Bởi vì lập trình không thể diễn tả bằng lời nói được nên phải mượn class để thể hiện.

**Ví dụ**: Khai báo class cho đối tượng con chó, đặt tên là **Dog**, nó có thuộc tính màu lông và hành động chạy.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | class Dog:        # Thuộc tính      color : ""        # Hành động      def run(self):          print("Con chó đang chạy") |

**Ví dụ**: Tạo class **Students**gồm cấc thông tin như sau:

* Sinh viên có hai thuộc tính gồm tên sinh viên và mã sinh viên
* Có 4 hành động chính trên sinh viên, đó là thêm sinh viên, sửa sinh viên, xóa sinh viên, và hiển thị thông tin sinh viên

Ta sẽ tạo class cho đối tượng sinh viên như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | class Student:        # Danh sách thuộc tính      id = ''      name = ''        # Phương thức thêm sinh viên      def add(self, id, name):          print("Hàm add")          self.id = id          self.name = name        # Phương thức xóa sinh viên      def remove(self, id):          print("hàm xóa")        # Phương thức sửa sinh viên      def edit(self, id, name):          print("Hàm sửa")        # Phương thức hiện thị thông tin      def show(self):          print("ID: ", self.id)          print("NAME: ", self.name) |

Như vậy, việc khai báo thuộc tính và phương thức rất giống với khai bái biến và hàm.

### self của class trong Python

Như bạn thấy ở ví dụ trên, tất cả các phương thức đều có một tham số truyền vào đầu tiên đó là **self**. Vậy **self trong Python là gì**?

Self là tham số ảo **bắt buộc** truyền vào khi khai báo phương thức, nó **không được tính** là một tham số phải truyền vào khi sử dụng phương thức. **Self**chính là biến trỏ đến chính đối tượng đó, nếu trong C, C++, C# hay PHP thì nó được gọi là **this**.

**Ví dụ**: Tạo một đối tượng **student**và sử dụng phương thức **add**, sau đó sử dung phương thức show.

|  |  |
| --- | --- |
| 1  2  3 | s = Student()  s.add('001', 'Cường Nguyễn Freetuts')  s.show() |

Kết quả:



Mặc dù lúc khai báo phương thức add() mình có thêm tham số self nhưng khi sử dụng thì Python đã loại bỏ tham số này. Mục đích là giúp bạn có thể thao tác với dữ liệu của chính class đó.

### Tạo một instance object trong Python

Object là một khái niệm cơ bản của lập trình hướng đối tượng Python.

Khi bạn khai báo class tức là bạn khai báo cấu trúc dữ liệu cho một nhóm đối tượng. Như ví dụ ở phần 2 thì **Dog**là nhóm đối tượng dành cho **Chó**, **Student**là nhóm đối tượng dành cho **sinh viên**.

Để tạo một đối tượng trong nhóm đối tượng đó thì ta phải khởi tạo chúng bằng cách tạo một **instance object**.

**Ví dụ**: Tạo ra ba instance object của đối tương sinh viên.

|  |  |
| --- | --- |
| 1  2  3 | s1 = Student()  s2 = Student()  s3 = Student() |

Lúc này, việc sử dụng những phương thức và thuộc tính trên mỗi instance sẽ riêng biệt.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | s1 = Student()  s1.add('001', 'Cường Nguyễn Freetuts')  s1.show()    s2 = Student()  s2.add('002', 'Kính Nguyễn Freetuts')  s2.show()      s3 = Student()  s3.add('003', 'Chính Nguyễn Freetuts')  s3.show() |

**Kết quả**:



### Các loại phương thức của class trong Python

Phương thức có nhiều loại khác nhau, mỗi loại có một công dụng và ý nghĩa riêng, và thường được áp dụng cho một trường hợp cụ thể nào đó. Vì vậy bạn phải hiểu và phân biệt giữa chúng để sử dụng cho đúng.

#### Phương thức khởi tạo

Phương thức khởi tạo hay còn gọi là **hàm khởi tạo**, đây là hàm sẽ được tự động gọi khi bạn tạo mới một đối tượng (instance object).

Ví dụ mình có class Dog như sau:

|  |  |
| --- | --- |
| 1  2  3 | class Dog:      name = "",      country = "" |

Nếu khởi tạo object Dog theo cách thông thường thì thế này:

|  |  |
| --- | --- |
| 1 | d = Dog() |

Lúc này hai thuộc tính **name**và **country**sẽ có giá trị rỗng. Bạn sẽ phải định nghĩa thêm một phương thức **setData**để thiết lập giá trị cho chúng.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | class Dog:      name = "",      country = ""        def setData(self, name, country):          self.name = name          self.country = country |

Và gọi đến phương thức này cho đối tương vừa tạo.

|  |  |
| --- | --- |
| 1  2 | d = Dog()  d.setData("Chăn cừu", "Đức") |

Mình thấy quá rườm rà, bây giờ mình muốn khi khởi tạo đối tượng mới thì phải truyền luôn hai giá trị name và country thì lúc này sẽ sử dụng hàm khởi tạo.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | class Dog:      name = "",      country = ""        def \_\_init\_\_(self, name, country):          self.name = name          self.country =country |

Và bây giờ mỗi khi khởi tạo đối tượng Dog thì bạn phải nhập thông tin dữ liêu cho chúng luôn.

|  |  |
| --- | --- |
| 1  2 | d = Dog("Chăn cừu", "Đức")  print(d.name) # Chăn cừu |

Bạn có thể thiết lập bao nhiêu tham số cho hàm khởi tạo đều được nhé, vì bản chất nó là một phương thức bình thường, chỉ là sẽ được gọi khi tạo mới object.

**Xem thêm**: [Hàm khởi tạo trong Python](https://freetuts.net/ham-khoi-tao-trong-python-3483.html).

#### Phương thức tĩnh static

Phương thức tĩnh là loại phương thức có thể được gọi mà không cần phải khởi tạo đối tượng class trong Python.

**Ở những ví dụ ở trên bạn có thể thấy rằng**: Để sử dụng một phương thức nào thì ta sẽ phải khởi tạo đối tượng rồi mới dùng. Tuy nhiên, có một loại phương thức có thể gọi đến mà không cần phải khởi tạo đối tượng, đó chính là **staticmethod**.

**Ví dụ**: Mình có lớp Mathematics như sau.

|  |  |
| --- | --- |
| 1  2  3 | class Mathematics:      def addNumbers(self, x, y):          return x + y |

Để sử dụng method **addNumbers**thì mình làm như sau:

|  |  |
| --- | --- |
| 1  2 | m = Mathematics()  print(m.addNumbers(12, 15)) |

Quá phiền hà, ta có thể đổi addNumbers thành một dạng static method như sau.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | class Mathematics:        @staticmethod      def addNumbers(x, y):          return x + y    # Cách dùng  print(Mathematics.addNumbers(12, 15)) |

Quan sát kỹ hơn thì ta sẽ thấy những kết luận như sau:

* Static method không có tham số ảo **Self**.
* Sử dụng @staticmethod đặt trước method để khai báo đó là một static method.
* Sử dụng cú pháp className.methodName() để gọi đến một static method.

Vì static method không khởi tạo đối tượng nên không có tham số Self, vì vậy ta chỉ sử dụng trong trường hợp bạn muốn gắn một hàm xử lý chung chung nào đó vào class để tiện cho việc quản lý code.

### Kế thừa class trong Python

Một class A có thể được kế thừa từ một class B khác, lúc này ta gọi A là **lớp con** và B là **lớp cha**.

Lớp con có thể kế thừa các thuộc tính và phương thức của lớp cha, trừ trường hợp đó là một private method (sẽ tìm hiểu tiếp ở phần 7).

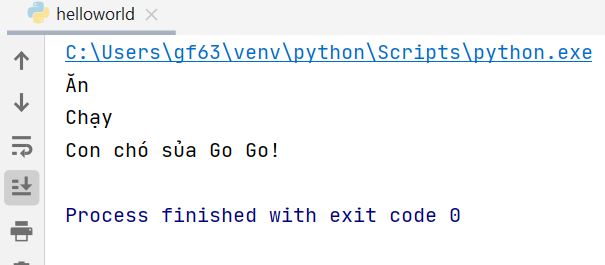
Việc kế thừa này là rất hay và thực tế.

Ví dụ, bạn khai báo lớp **Animal**chung chung dành cho động vật, nó có các thuộc tính tên, giới tính, ... và hành động ăn, uống, đi vệ sinh, chạy, ...

Bây giờ bạn muốn khai báo thêm lớp **Dog**, đây cũng là một động vật nên cũng có các phương thức và thuộc tính như của **Animal**. Vì vậy ta có thể thiết lập **Dog**kế thừa **Animal**để tiết kiệm chương trình.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | # Animal  class Animal:      ten = ''      gioitinh = ''        def an(self):          print("Ăn")      def chay(self):          print("Chạy")    # Dog kế thừa từ Animal  class Dog(Animal):        def sua(self):          print("Con chó sủa Go Go!")    # Cách dùng  d = Dog()  d.an()  d.chay()  d.sua() |

Kết quả:



### Mức độ truy cập public / protected và private

Trong lập trình hướng đối tượng thì các thuộc tính và phương thức sẽ có 3 mức độ truy cập khác nhau.

* **Public**: là câp độ phổ biến nhất, có thể sử dụng ở cả bên trong và ngoài lớp
* **Protected**: Là cấp độ được bảo vệ, chỉ dùng trong nội bộ của lớp đó và lớp con kế thừa.
* **Private**: Là cấp độ bảo mật nhất, nó chỉ được dùng bên trong chính lớp đó mà thôi.

**Cách khai báo**:

* Sử dụng hai dấu gạch dưới (\_\_) để khai báo cho mức private
* Sử dụng một dấu gạch dưới (\_) để khai báo cho mức protected
* Không sử dụng dấu gạch dưới là public.

Xem ví dụ cách khai báo dưới đây:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | # Animal  class className:        # Thuộc tính      name = '' # public      \_name = '' # protected      \_\_name = '' # private        # Phương thức      def getName(self): # public          return 0        def \_getName(self): # protected          return 0        def \_\_getName(self): # private          return 0 |

Bây giờ là một ví dụ rõ ràng hơn về các mức độ truy cập trong Python.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | class Dog:      \_\_name = ''      name = ''        def setName(self, name):          # Đúng vì thuộc tính private có thể truy cập trong class          self.\_\_name = name        def showName(self):          print(self.\_\_name)    d = Dog()    # Đoạn code này sai vì \_\_name ở mức private  d.\_\_name = 'Chó Bull'    # Đúng vì name là public  d.name = 'Chó Bull'    # Đúng vì hàm setName là public  d.setName("Chó Bull")    # Đúng vì showName là public  d.showName(); |

Trên là những thông tin cơ bản nhất về lập trình hướng đối tượng trong Python, hay nói đơn giản hơn là cách sử dụng class trong Python. Sắp tới mình sẽ có những bài viết giải thích chi tiết từng mục nhỏ trong bài này để giúp các bạn dễ hiểu hơn nhé.

## Hàm khởi tạo trong Python, khi nào thì nên dùng?

Trong bài này chúng ta sẽ tìm hiểu **hàm khởi tạo trong Python**, trong tiếng Anh ta gọi nó là **Constructors method**. Đây là loại hàm khá đặc biệt, được dùng để chạy những đoạn code quan trọng khi khởi tạo một đối tượng.

Có một số trường hợp bạn muốn chạy một đoạn code nào đó khi tạo mới một đối tượng. Ta có thể xử lý bằng cách đặt nó ở bên trong một phương thức nào đó, và gọi sau khi khởi tạo đối tượng:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | class className:      def config(self):          print("Cấu hình cho lớp")      c = className()  c.config() |

Tuy nhiên, vì bạn đang làm việc nhóm nên việc bắt buộc những người khác phải gọi đến hàm **config** này, và sẽ có người nhớ và có người quên. Vì vậy cần có một giải pháp nào tự động gọi đến hàm **config** mỗi khi tạo mới đối tượng? Đó chính là hàm khởi tạo.

**Mục lục**

* [1. Hàm khởi tạo trong Python là gì?](https://freetuts.net/ham-khoi-tao-trong-python-3483.html#goto-h2-0)
* [2. Tham số truyền vào hàm khởi tạo Python](https://freetuts.net/ham-khoi-tao-trong-python-3483.html#goto-h2-1)
* [3. Khi nào sử dụng hàm khởi tạo trong Python](https://freetuts.net/ham-khoi-tao-trong-python-3483.html#goto-h2-2)

### Hàm khởi tạo trong Python là gì?

Hàm khởi tạo trong Python là một phương thức đặc biệt, nó có tên là \_\_init\_\_ và được gọi tự động mỗi khi bạn tạo mới một instance object của class.

*Bài viết này được đăng tại [free tuts .net]*

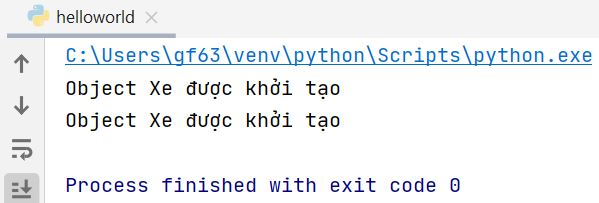
Cú pháp như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | class className:      def \_\_init\_\_(self, var1, var2, ...):          # Code sẽ chạy khi tạo mới một className    # Cách dùng  c = className(var1, var2, ...) |

**Ví dụ**: Tạo class Xe và có hàm khởi tạo.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | class Xe:      def \_\_init\_\_(self):          print("Object Xe được khởi tạo")    x1 = Xe()  x2 = Xe() |

Kết quả:



### Tham số truyền vào hàm khởi tạo Python

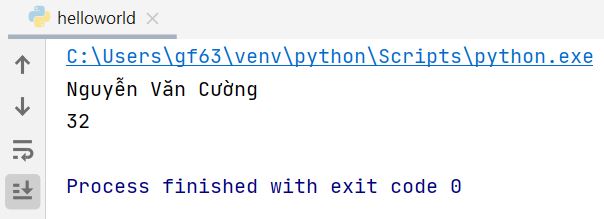
Hàm khởi tạo cũng là một phương thức bình thường nên bạn cũng có thể truyền tham số cho nó.

Lúc này, khi khởi tạo object thì bạn phải truyền tham số cho nó nhé.

**Vi dụ**: Hãy tạo một lớp **Student**gồm hai thông tin tên và tuổi. Khi tạo mới một student thì bắt buộc phải nhập hai thông tin này.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | class Student:      name = ''      age = 0      def \_\_init\_\_(self, name, age):          self.name = name          self.age = age    c = Student("Nguyễn Văn Cường", 32)    print(c.name)  print(c.age) |

kết quả:



### Khi nào sử dụng hàm khởi tạo trong Python

Bạn nên sử dụng hàm khởi tạo trong trường hợp muốn chạy một đoạn code nào đó mỗi khi khởi tạo một object, đó có thể là đoạn code cấu hình quan trọng cho đối tượng.

**Ví dụ**: Bạn viết class xử lý các thao tác thêm / xóa / sửa cho sinh viên thì bắt buộc phải kết nối vào CSDL. Vì vậy mỗi khởi tạo đối tượng sinh viên thì ta sẽ chạy đoạn code kết nối DB ngầm.

Về lý thuyết thì như vậy, còn thực tế thì tùy vào khả năng và ý tưởng của lập trình viên.

Như vậy là chúng ta đã học xong về hàm khởi tạo Python. Hẹn gặp lại các bạn ở bài tiếp theo nhé.

## Kế thừa trong Python

Trong bài này mình sẽ giải thích sâu hơn về **kế thừa trong Python**, nhất là kế thừa liên quan đến các mức độ truy cập private / protected và public trong Python.

Trong [lập trình hướng đối tượng Python](https://freetuts.net/class-trong-python-3482.html) thì tính kế thừa rất quan trọng, nó giúp bạn thiết kế cấu trúc ứng dụng một cách khoa học, phân chia các lớp một cách tối ưu, lớp này có thể tái sử dụng dữ liệu của lớp khác.

**Mục lục**

* [1. Kế thừa trong Python là gì?](https://freetuts.net/ke-thua-trong-python-3484.html#goto-h2-0)
* [2. Kế thừa thuộc tính và phương thức của lớp cha](https://freetuts.net/ke-thua-trong-python-3484.html#goto-h2-1)
* [3. Kế thừa trong Python với hàm khởi tạo](https://freetuts.net/ke-thua-trong-python-3484.html#goto-h2-2)

### Kế thừa trong Python là gì?

Kế thừa (Inheritance) là việc một lớp được khai báo kế thừa toàn bộ thuộc tính và phương thức của một lớp khác. Lớp được kế thừa ta gọi là lớp con, và lớp kế thừa ta gọi là lớp cha.

Lớp con có thể sử dụng toàn bộ dữ liệu khai báo ở mức độ **protected**và **public**ở lớp cha. Riêng với **private**thì không được, vì đó là mức độ bảo mật cao nhất, chỉ sử dụng bên trong nội bộ của lớp cha.

*Bài viết này được đăng tại [free tuts .net]*

Nếu bạn chưa biêt về ba mức độ này thì hãy quay lại bài viết [class trong Python](https://freetuts.net/class-trong-python-3482.html) để đọc nhé.

Cách khai báo kế thừa:

|  |  |
| --- | --- |
| 1  2  3  4  5 | class lopCha:      # Code của lớp cha    class lopCon(lopCha):      # Code của lớp con |

**Ví dụ**: Khao báo lớp XeDap kế thừa từ lớp Xe.

|  |  |
| --- | --- |
| 1  2  3  4  5 | class Xe:      # Code ...    class XeDap(Xe):      # Code ... |

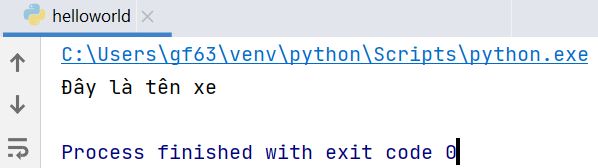
Phần này bạn chỉ cần hiểu khái niệm và cú pháp thôi nhé, còn chi tiết cách dùng kế thừa trong Python thì chúng ta sẽ học ở những phần tiếp theo dưới đây.

### Trong kế thừa Python thì đặc điểm hay nhất là lớp con co thể sử dụng dữ liệu của lớp cha, trừ trường hợp dữ liệu đó là **private**.

**Ví dụ 1**: Sử dụng thuộc tính của lớp cha

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | class Xe:      name = 'Đây là tên xe'    class XeDap(Xe):      def showName(self):          # sử dụng thuộc tính name của lớp cha          print(self.name)    # Cách dùng  d = XeDap()  d.showName() |

**Kết quả**:



**Ví dụ 2**: Sử dụng phương thức của lớp cha

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | class Xe:      name = ''      def setName(self, name):          self.name = name    class XeDap(Xe):      def showName(self, name):          # Sử dụng phương thức của lớp cha          self.setName("Xe đạp")            # sử dụng thuộc tính name của lớp cha          print(self.name)    # Cách dùng  d = XeDap()  d.showName("Xe đạp") |

Kết quả:



### Kế thừa trong Python với hàm khởi tạo

[Hàm khởi tạo](https://freetuts.net/ham-khoi-tao-trong-python-3483.html) trong Python sẽ được tự động gọi mỗi khi bạn tạo mới đối tượng. Nếu là class đơn thì quá dễ, trường hợp class có kế thừa thì câu hỏi đặt ra như sau:

Nếu cả lớp cha và lớp con đều có hàm khởi tạo thì Python sẽ sử dụng hàm nào?

Theo mức độ ưu tiên thì Python sẽ chạy hàm khởi tạo ở lớp con nhé các bạn.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | class Xe:      def \_\_init\_\_(self):          print("Hàm khởi tạo lớp cha")    class XeDap(Xe):      def \_\_init\_\_(self):          print("Hàm khởi tạo lớp con")    # Cách dùng  d = XeDap() |

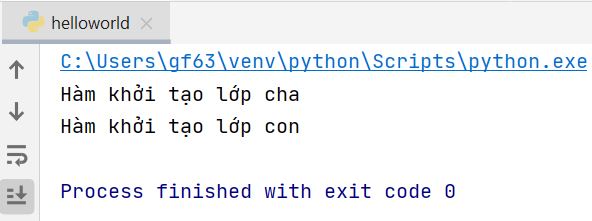
Kết quả:



Nếu bạn muốn chạy hàm khởi tạo của lớp cha thì hãy gọi nó ở lớp con thông qua hàm **supper()** nhé.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | class Xe:      def \_\_init\_\_(self):          print("Hàm khởi tạo lớp cha")    class XeDap(Xe):      def \_\_init\_\_(self):          # Gọi hàm khởi tạo lớp cha          super().\_\_init\_\_()            print("Hàm khởi tạo lớp con")    # Cách dùng  d = XeDap() |

Kết quả:



Hoặc bạn cũng có thể sử dụng thông qua tên của lớp cha như ví dụ dưới đây.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | class Xe:      def \_\_init\_\_(self):          print("Hàm khởi tạo lớp cha")    class XeDap(Xe):      def \_\_init\_\_(self):          # Gọi hàm khởi tạo lớp cha          Xe.\_\_init\_\_(self)            print("Hàm khởi tạo lớp con")    # Cách dùng  d = XeDap() |

Trên là cách sử dụng tính kế thừa trong Python nói chung và trong lâp trình hướng đối tương Python nói riêng. Bài này mình sẽ dừng ở đây, bài tiếp theo chúng ta sẽ tìm hiểu đa kế thừa trong Python nhé.

## Đa kế thừa trong Python

Trong bài này chúng ta sẽ tìm hiểu đa kế thừa trong Python, đây là cách kế thừa nhiều cấp với nhau, class B kế thừa class A, class C kế thừa class B. Ngoài ra một class có thể kế thừa từ nhiều class khác.

Trước khi đọc bài này thì phải chắ chắn rằng bạn đã xem bài [kế thừa trong Python](https://freetuts.net/ke-thua-trong-python-3484.html) nhé, vì bài đó mình giải thích khá kỹ về các trường hợp khác nhau khi kế thừa class. Trước tiên hãy tìm hiểu về một lớp kế thừa từ nhiều lớp khác.

**Mục lục**

* [1. Class kế thừa nhiều lớp trong Python](https://freetuts.net/da-ke-thua-trong-python-3485.html#goto-h2-0)
* [2. Kế thừa đa cấp trong Python](https://freetuts.net/da-ke-thua-trong-python-3485.html#goto-h2-1)
* [3. Thứ tự kế thừa trong đa kế thừa Python](https://freetuts.net/da-ke-thua-trong-python-3485.html#goto-h2-2)
* [4. Cấp độ truy cập của đa kế thừa trong Python](https://freetuts.net/da-ke-thua-trong-python-3485.html#goto-h2-3)

### Class kế thừa nhiều lớp trong Python

Một class được kế thừa từ nhiều khác là trường hợp đầu tiên của **đa kế thừa trong Python**.

Trong bài học trước chúng ta chỉ kế thừa từ một lớp, nhưng thực tế thì bạn có thể kế thừa từ nhiều lớp bằng cách sử dụng cú pháp sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | class Lop1:      # code    class Lop2:      # code    class Lopcon(Lop1, Lop2):      # code |

Câu hỏi đặt ra là khi kế thừa kiểu này thì việc sử dụng các thuộc tính và phương thức từ lớp cha trong lớp con sẽ như thế nào?

Nó sẽ tuân theo khái niệm kế thừa, nghĩa là lớp con sẽ có toàn bộ các tính năng của các lớp cha.

### Kế thừa đa cấp trong Python

Trường hợp thú hai trong đa kế thừa Python đó là **kế thừa đa cấp**, đây cũng là một loại đa kế thừa trong Python.

Dạng kế thừa này được sử dụng khá nhiều, lớp con được kế thừa từ lớp cha, lớp cha kế thừa từ lớp ông nội, ... cú pháp như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | class LopOngNoi:      pass    class LopCha(LopOngNoi):      pass    class LopCon(LopCha):      pass |

Ví dụ này mình kế thừa 3 cấp, thực tế thì bạn có thể kế thừa từ nhiều cấp.

Tương tự như trên, lớp con sẽ kế thừa toàn bộ những tính năng của các lớp cha và ông nội.

### Thứ tự kế thừa trong đa kế thừa Python

Vì được kế thừa từ nhiều lớp nên việc sử dụng tài nguyên phải tuân theo những quy luật dưới đây.

Theo mặc định thì mọi kiểu dữ liệu trong Python đều được xuất phát từ class object, vì vậy về mặt kỹ thuật thì tất cả các class dù là người dùng tự định nghĩa hay có sẵn trong Python đều là một thể hiện (instance) của class tên là **object**.

|  |  |
| --- | --- |
| 1  2  3 | # class object  o = object()  print(o) |

**Trường hợp kế thừa đa cấp**: Khi lớp con gọi đến một thuộc tính hoặc phương thức thì Python sẽ dò tìm trong lớp con trước, nếu không có thì sẽ dò tiếp lớp cha, nếu vẫn không có thì nó sẽ dò ở lớp ông nội ... cứ như vậy cho đến level cuối cùng.

**Trường hợp kế thừa từ nhiều lớp**: Cách hoạt động vẫn như trên, nhưng nó sẽ duyệt từ trái qua phải theo thứ tự mà bạn liệt kê trong lúc kế thừa.

Như ví dụ dưới đây nó sẽ ưu tiên tìm ở Lopcon trước, sau đó đến Lop1, tiếp là Lop2.

|  |  |
| --- | --- |
| 1 | class Lopcon(Lop1, Lop2): |

**Ví dụ 1**: Trường hợp lớp con tồn tại

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | class LopOngNoi:      def sayHi(self):          print("Hi, lớp ông nội")    class LopCha(LopOngNoi):      def sayHi(self):          print("Hi, lớp cha")    class LopCon (LopCha):      def sayHi(self):          print("Hi, lớp con")    # Main  c = LopCon()  c.sayHi() # Kết quả: Hi, lớp con |

**Ví dụ 2**: Trường hợp lớp cha tồn tại

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | class LopOngNoi:      def sayHi(self):          print("Hi, lớp ông nội")    class LopCha(LopOngNoi):      def sayHi(self):          print("Hi, lớp cha")    class LopCon (LopCha):      pass    # Main  c = LopCon()  c.sayHi() # Kết quả: Hi, lớp cha |

**Ví dụ 3**: Trường hợp lớp ông nội tồn tại

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | class LopOngNoi:      def sayHi(self):          print("Hi, lớp ông nội")    class LopCha(LopOngNoi):      pass    class LopCon (LopCha):      pass    # Main  c = LopCon()  c.sayHi() # Kết quả: Hi, lớp ông nội |

### Cấp độ truy cập của đa kế thừa trong Python

Như đơn kế thừa, một lớp chỉ có thể sử dụng tài nguyên ở cấp độ **public**hoặc **protected**. Nếu ở ngoài lớp thì mỗi public là sử dụng được.

**Ví dụ**: Sử dụng ngoài lớp.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | class LopOngNoi:      def \_\_sayHi(self):          print("private")        def \_sayHi(self):          print("protected")        def sayHi(self):          print("public")      class LopCha(LopOngNoi):      pass    class LopCon (LopCha):      pass    # Main  c = LopCon()  c.sayHi() # Đúng, vì public  c.\_sayHi() # Sai, vì protected  c.\_\_sayHi() # Sai, vì private |

**Ví dụ 2**: Sử dụng trong lớp con

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | class LopOngNoi:      def \_\_sayHi(self):          print("private")        def \_sayHi(self):          print("protected")        def sayHi(self):          print("public")      class LopCha(LopOngNoi):      pass    class LopCon (LopCha):      def sayHi(self):          self.\_sayHi() # Đúng, vì protected          self.\_sayHi() # Sai, vì private |

Bạn hãy tự thay đổi nội dung của các ví dụ để hiểu rõ hơn về đa kế thừa trong Python nhé.

## Setter và Getter trong Python

Trong bài này chúng ta sẽ tìm hiểu về getter và setter trong Python, đây là cách **thiết lập giá trị** và **lấy giá trị** các thuộc tính của class mà không vi phạm đến tính an toàn của chúng.

Trong Python, getter và setter không giống với những ngôn ngữ lập trình hướng đối tượng khác, nó sử dụng từ khóa @property để khai báo getter, và @name.setter để khai báo setter.

**Mục lục**

* [1. Getter và setter là gì?](https://freetuts.net/setter-va-getter-trong-python-3486.html#goto-h2-0)
* [2. Tạo getter và setter theo cách thông thường](https://freetuts.net/setter-va-getter-trong-python-3486.html#goto-h2-1)
* [3. Cú pháp getter và setter trong Python](https://freetuts.net/setter-va-getter-trong-python-3486.html#goto-h2-2)

### Getter và setter là gì?

Getter là phương thức dùng để lấy dữ liệu thuộc tính của một lớp. Còn Setter là phương thức dùng để thiết lập giá trị cho thuộc tính.

Thực tế bạn có thể truy cập trực tiếp đến thuộc tính bằng cách khai báo cho nó là public, sau đó gọi đến một cách bình thường.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | class className:      name = ''    c = className()  c.name = 'freetuts.net'  print(c.name) |

Tuy nhiên, việc cho phép bên ngoài truy xuất đến dữ liệu nhạy cảm như vậy sẽ không tốt. Vì vậy giải pháp là tạo ra những hàm hỗ trợ việc xử lý truy cập đến những thuộc tính đó, và ta gọi nó là setter và getter.

### Tạo getter và setter theo cách thông thường

Trong Python, để viết những phương thức **getter**và **setter**theo cách thông thường thì bạn chỉ việc tạo ra hai phương thức có tên khác nhau.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | class className:        # Thuộc tính name      \_\_name = ''        # Setter cho name      def setName(self, name):          self.\_\_name = name        # Getter cho name      def getName(self):          return self.\_\_name    # Sử dụng  c = className()  c.setName("Cường")  print(c.getName())  # Kết quả: Cường |

Thuộc tính \_\_name đang ở cấp độ private, đây cũng chính là cấp độ nên sử dụng khi khai báo các thuộc tính trong lập trình hướng đối tượng.

### Cú pháp getter và setter trong Python

Để khai báo một hàm là setter thì ta sử dụng từ khóa **@name.setter**, còn getter thì sử dụng từ khóa **@property**, cả hai đều được đặt phía trước khai báo hàm (phương thức).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | class Freetuts:        # Thuộc tính name      \_\_domain = ''        # Getter      @property      def domain(self):          print("Getter được gọi")          return self.\_\_domain        # Setter      @domain.setter      def domain(self, domain):          print("Setter được gọi")          self.\_\_domain = domain |

Như bạn thấy, mình đã khai báo hai hàm trùng tên **domain** luôn nhưng không bị Python báo lỗi, lý do là nó hiểu đây chính là hai hàm getter và setter.

Lúc này tùy vào **số lượng tham số** truyền vào hàm mà nó gọi đúng đến hàm có số lượng tham số tương ứng.

Dưới đây là cách sử dụng:

|  |  |
| --- | --- |
| 1  2  3  4 | # Sử dụng  c = Freetuts()  c.domain = "[https://freetuts.net](https://freetuts.net/)"  print(c.domain) |

Kết quả như sau:



Như ví dụ này thì rõ ràng tên của thuộc tính là \_\_domain và nó ở dạng private. Nhưng khi sử dụng thì bạn có thể đổi thành domain vì mình đã khai báo đó là những setter và getter.

Trên là những chia sẻ về cách sử dụng setter và getter trong Python.

## Override trong Python (Ghi đè trong kế thừa)

Trong bài này chúng ta sẽ học cách sử dụng **Override trong Python**, đây là phương pháp ghi đè phương thức trong các lớp kế thừa của Python.

Ghi đè phương thức là một tính năng rất quan trọng trong lập trình hướng đối tượng OOP, nó cho phép một lớp con có thể viết lại các phương thức của lớp cha, tức là tạo một phương thức ở lớp con **trùng tên** với phương thức của lớp cha.

**Mục lục**

* [1. Override trong Python là gì?](https://freetuts.net/override-trong-python-3487.html#goto-h2-0)
* [2. Cách gọi đến phương thức bị Override ở lớp cha](https://freetuts.net/override-trong-python-3487.html#goto-h2-1)
* [3. Override trong kế thừa nhiều lớp](https://freetuts.net/override-trong-python-3487.html#goto-h2-2)

### Override trong Python là gì?

Override là cách viết lại các method ở lớp cha trong lớp con, bằng cách áp dụng quy tắt gọi method trong kế thừa đó là: Khi bạn gọi một method thì Python sẽ tìm và gọi trong lớp con trước, nếu không có thì gọi ở lớp cha, nếu vẫn không có thì gọi đến lớp ông nội ...

**Ví dụ**: Giả sử mình khai báo lớp động vật Animal như sau.

*Bài viết này được đăng tại [free tuts .net]*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | class Animal:      name : ""        def move(self):          pass        def eat(self):          pass |

Bây giờ mình tạo một lớp Dog để kế thừa lớp Animal, sau đó viết lại phương thức **move**.

|  |  |
| --- | --- |
| 1  2  3 | class Dog(Animal):      def move(self):          print('Con chó đi bốn chân') |

Tiêp tục, mình viêt class Duck kế thừa từ lớp Animal.

|  |  |
| --- | --- |
| 1  2  3 | class Duck(Animal):      def move(self):          print('Con vịt đi hai chân') |

Như vậy, phương thức **move**ở lớp **Animal**sẽ có cách hoạt động khác nhau và chúng tùy vào loại động vật.

* Chó thì đi 4 chân
* Vịt thì đi 2 chân

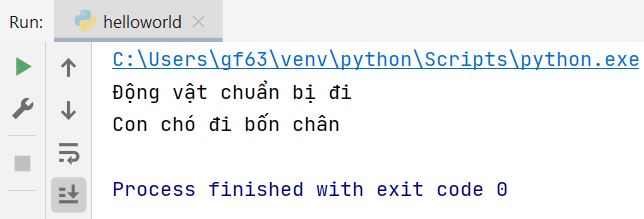
Nếu sử dụng chung hàm **move**ở **Animal** thì không đáp ứng được yêu cầu của bài toán. Đây là một ví dụ điển hình để trả lời cho câu hỏi "khi nào nên sử dụng override trong Python?".

### Cách gọi đến phương thức bị Override ở lớp cha

Trong Python, khi bạn ghi đè overide thì method bị override ở lớp cha sẽ hết tác dụng. Tuy nhiên, có một số trường hợp bạn cần gọi đến method bị override đó thì hãy sử dụng cách sau:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | class Animal:      name : ""        def move(self):          print('Động vật chuẩn bị đi')        def eat(self):          pass    class Dog(Animal):      def move(self):          Animal.move(self) # Gọi đến method move của lớp cha          print('Con chó đi bốn chân')    # Cách dùng  t = Dog()  t.move() |

Kết quả:



Ngoài ra bạn cũng có thể sử dụng hàm supper() như sau:

|  |  |
| --- | --- |
| 1 | super().move() # Gọi đến method move của lớp cha |

### Override trong kế thừa nhiều lớp

Như mình đa nói ở những bài học trước, việc gọi đến các method trong kế thừa sẽ áp dụng quy tắc: Lớp con -> Lớp cha -> Lớp ông nội.

Nên khi bạn muốn override method ở lớp ông nội thì cũng làm như bình thường, có điều nếu bạn muốn gọi đếm hàm bị override đó thì sử dụng tên class và trỏ tới đúng method đó.

Xem ví dụ dưới đây để hiểu rõ hơn:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | class Parent():        def show(self):          print("Cha")    class Child(Parent):      def show(self):          print("Con")    class GrandChild(Child):      def show(self):          Parent.show(self)          print("Cháu")    # Chương trình chính  g = GrandChild()  g.show() |

Kết quả:



Trên là cách sử dụng Override trong Python, hy vọng qua bài này sẽ giúp bạn hiểu rõ hơn về phương pháp ghi đè method trong Python.

## Interface trong Python

Trong bài này chúng ta sẽ tìm hiểu Interface trong Python, đây là kỹ thuật đóng vai trò rất quan trọng trong kỹ thuật phần mềm, nhất là việc quản lý mã code của Python.

Có một số trường hợp bạn cần tạo ra nhiều class có những phương thức **rất giống nhau**, nhưng về cách tiếp cận thì **có chút khác nhau**, điều này có thể gây ra một số nhầm lẫn. Trong trường hợp này bạn có thể sử dụng Interface trong Python để giải quyết.

**Lưu ý**: Trong Python không có khái niệm interface, nên bài này mình chỉ muốn giải thích cho bạn hiểu ý nghĩa của interface mà thôi.

**Mục lục**

* [1. Interface trong Python là gì?](https://freetuts.net/interface-trong-python-3488.html#goto-h2-0)
* [2. Tại sao phải sử dụng Interface trong Python?](https://freetuts.net/interface-trong-python-3488.html#goto-h2-1)
* [3. Lời kết](https://freetuts.net/interface-trong-python-3488.html#goto-h2-2)

### Interface trong Python là gì?

Interface là một bản thiết kế cho các class trong Python. Nó cũng là một class nhưng cách hoạt động khác với những class bình thường. Bạn có thể khai báo các phương thức bên trong interface, nhưng những phương thức này là trừu tượng (abstract), tức là chỉ khai báo chứ không xây dựng nội dung.

Bạn có thể tưởng tượng **Interface**là một mẫu chuẩn thiết kế cho các loại nhà (đặt tên nó là A), và:

* Mẫu nhà cấp 4 là một **abstract class**, bởi nó được triển khai từ A.
* Mẫu nhà mái thái là một **abstract class**, bởi nó được triển khai từ A.

Cách tiếp cận Interface trong Python hơi khác so với các ngôn ngữ khác. Hầu hết những ngôn ngữ đó đều sử dụng từ khóa interface để định nghĩa, còn trong Python thì không có, bạn sẽ phải sử dụng một module tên là **Abstract Base Classes** (ABC) để tạo Interface.

### Tại sao phải sử dụng Interface trong Python?

Để trả lời cho câu hỏi này thì ta hãy làm một ví dụ nhé.

**Ví dụ**: Thiết kế chương trình quản lý các loại động vật như chó, mèo, vịt.

**Nhận xét**: Đây đều là những loại động vật, nó có các hành động chung như di chuyển, ăn ... Vì vậy ta sẽ khai báo một interface **Animal**, sau đó abstract class sẽ implement (triểu khai) từ Animal.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | class Animal:      def move(self):          pass        def eat(self):          pass |

Bây giờ ta sẽ tạo một class **Dog**:

|  |  |
| --- | --- |
| 1  2  3  4  5 | class Dog(Animal):      def move(self):          print("Con chó di chuyển 4 chân")      def eat(self):          print("Con chó ăn thịt") |

Và đây là class **Duck**:

|  |  |
| --- | --- |
| 1  2  3  4  5 | class Duck(Animal):      def move(self):          print("Con vịt di chuyển 2 chân")      def eat(self):          print("Con chó ăn cám") |

Trong ví dụ này thì class Animal có thể được xem là một Interface, còn các class Dog và Duck được triển khai (implement) từ Animal.

### Lời kết

Mình cũng không rõ tại sao Python lại không có khái niệm Interface. Có ý kiến cho rằng bản chất Python không cần interface bởi nó đã có [đa kế thừa](https://freetuts.net/da-ke-thua-trong-python-3485.html). Nhưng dù sao đi nữa thì đây vẫn là một ngôn ngữ có cấu trúc rất đặc biệt.

## Bài tập Python: Module / Object / Class

Trong bài này chúng ta sẽ làm bài tập Python liên quan đến **hướng đối tượng** như: Thao tác với class, kế thừa, tạo ra những chương trình đơn giản có sử dụng class.

Qua những bài tập này sẽ giúp bạn nắm vững hơn về cách tạo class trong Python, cũng như những phương pháp tiếp cận kỹ thuật lập trình Python nâng cao.

**Danh sách bài tập**

* [Chương trình quản lý sinh viên với Module trong Python](https://freetuts.net/ref/quan-ly-sinh-vien-voi-module-trong-python-424.html)
* [Chương trình quản lý sinh viên bằng class trong Python](https://freetuts.net/ref/quan-ly-sinh-vien-bang-class-trong-python-425.html)