**PYTHON CƠ BẢN**

**LỜI MỞ ĐẦU**

Hiện nay, Python là một trong những ngôn ngữ lập trình đang được chú ý bởi tính đa dạng về ứng dụng, thư viện phong phú và cộng đồng đông đảo. Cuốn sách nhỏ này được viết trong quá trình mình bắt đầu học Python và giải quyết các bài toán cơ bản theo nhu cầu của mình.

Hy vọng những ghi chép của mình cũng sẽ giúp ích cho những ai đang quan tâm đến việc ứng dụng Python vào công việc và xử lý hiện tại.

[CHƯƠNG I: TỔNG QUAN 1](#_Toc167915855)

[BÀI 1: NGÔN NGỮ LẬP TRÌNH PYTHON 1](#_Toc167915856)

[1. Ngôn ngữ lập trình Python 1](#_Toc167915857)

[2. Lý do nên học Python 1](#_Toc167915858)

[3. Ưu điểm của ngôn ngữ lập trình Python 2](#_Toc167915859)

[4. Phiên bản Python 2](#_Toc167915860)

[BÀI 2: CÀI ĐẶT PYTHON 3](#_Toc167915861)

[1. Download Python 3](#_Toc167915862)

[2. Cài đặt Python trên Win 10 3](#_Toc167915863)

[3. Kiểm tra phiên bản Python 5](#_Toc167915864)

[4. Kiểm tra vị trí cài đặt python 5](#_Toc167915865)

[BÀI 3: CHẠY CHƯƠNG TRÌNH PYTHON ĐẦU TIÊN 6](#_Toc167915866)

[1. Viết và chạy chương trình đầu tiên với Python trên Visual Studio Code 6](#_Toc167915867)

[1.1. Chạy chương trình trong CMD 9](#_Toc167915868)

[2. Lời Kết 11](#_Toc167915869)

[CHƯƠNG II: THÀNH PHẦN 1](#_Toc167915870)

[BÀI 1: KHAI BÁO BIẾN TRONG PYTHON 1](#_Toc167915871)

[1. Biến trong Python là gì? 1](#_Toc167915872)

[2. Cách khai báo biến trong Python 2](#_Toc167915873)

[3. Một số cách gán biến đặc biệt 2](#_Toc167915874)

[4. Một số ví dụ về đặt tên biến sai 2](#_Toc167915875)

[5. Thay đổi giá trị của biến trong python 3](#_Toc167915876)

[6. Xóa biến trong Python 3](#_Toc167915877)

[BÀI 2: CÁC KIỂU DỮ LIỆU TRONG PYTHON 4](#_Toc167915878)

[1. Các kiểu dữ liệu trong python 4](#_Toc167915879)

[2. Cách thay đổi dữ liệu cho biến Python 4](#_Toc167915880)

[2.1. Sử dụng toán tử gán 4](#_Toc167915881)

[2.2. Ép kiểu dữ liệu ngầm trong Python 5](#_Toc167915882)

[2.3. Ép kiểu trong Python tường minh 5](#_Toc167915883)

[2.4. Bảng các hàm ép kiểu dữ liệu trong Python 6](#_Toc167915884)

[3. Kiểm tra kiểu dữ liệu 7](#_Toc167915885)

[BÀI 3: KIỂU DỮ LIỆU NUMBERS 8](#_Toc167915886)

[1. Kiểu dữ liệu number là gì ? 8](#_Toc167915887)

[2. Cú pháp tạo number trong Python 8](#_Toc167915888)

[3. Số thập phân trong Python 9](#_Toc167915889)

[4. Khi nào sử dụng Decimal thay vì float 9](#_Toc167915890)

[5. Phân số 10](#_Toc167915891)

[6. Các hàm toán học trong kiểu dữ liệu number 10](#_Toc167915892)

[BÀI 4: KIỂU DỮ LIỆU STRING 12](#_Toc167915893)

[1. Kiểu dữ liệu string là gì ? 12](#_Toc167915894)

[2. Cú pháp tạo string trong Python 12](#_Toc167915895)

[3. Các ký tự đặc biệt trong chuỗi 12](#_Toc167915896)

[4. Hiển thị chuỗi trong python 13](#_Toc167915897)

[5. Truy cập các ký tự của chuỗi string trong Python 13](#_Toc167915898)

[6. Thay đổi và xóa chuỗi string trong Python 14](#_Toc167915899)

[7. Các phép toán với chuỗi string trong Python 15](#_Toc167915900)

[7.1. Nối hai hoặc nhiều chuỗi 15](#_Toc167915901)

[7.2. Lặp qua từng ký tự trong chuỗi 16](#_Toc167915902)

[7.3. Kiểm tra chuỗi con trong chuỗi cha 16](#_Toc167915903)

[BÀI 5: KIỂU DỮ LIỆU LIST 17](#_Toc167915904)

[1. Các thao tác trên mảng trong Python 17](#_Toc167915905)

[1.1. Thay đổi giá trị của phần tử 17](#_Toc167915906)

[1.2. Đếm chiều dài của mảng 17](#_Toc167915907)

[1.3. Lặp qua từng phần tử 18](#_Toc167915908)

[1.4. Kiểm tra một giá trị có tồn tại trong mảng 18](#_Toc167915909)

[1.5. Thêm phần tử vào mảng (List) 18](#_Toc167915910)

[1.6. Xóa phần tử ra khỏi mảng 18](#_Toc167915911)

[1.7. Thiết lập List rỗng 19](#_Toc167915912)

[2. List lồng nhau 19](#_Toc167915913)

[2.1. Kiểu dữ liệu Tuple trong Python 20](#_Toc167915914)

[2.2. Kiểu dữ liệu Dictionary trong Python 20](#_Toc167915915)

[BÀI 6: CHÚ THÍCH TRONG PYTHON 21](#_Toc167915916)

[1. Comment trong Python là gì? 21](#_Toc167915917)

[2. Cách tạo comment trong Python 21](#_Toc167915918)

[2.1. Tạo trên một dòng 21](#_Toc167915919)

[2.2. Tạo trên nhiều dòng 21](#_Toc167915920)

[BÀI 7: CÁC TOÁN TỬ TRONG PYTHON 22](#_Toc167915921)

[1. Toán tử toán học trong Python 22](#_Toc167915922)

[2. Toán tử so sánh 22](#_Toc167915923)

[3. Toán tử gán 23](#_Toc167915924)

[4. Toán tử logic trong Python 24](#_Toc167915925)

[CHƯƠNG III: VÒNG LẶP 25](#_Toc167915926)

[BÀI 1: CÂU LỆNH IF ELSE TRONG PYTHON 25](#_Toc167915927)

[1. Cú pháp lệnh if else trong Python 25](#_Toc167915928)

[1.1. Cú pháp if else 25](#_Toc167915929)

[1.2. Cú pháp if else lồng nhau 25](#_Toc167915930)

[1.3. Cú pháp elif 26](#_Toc167915931)

[2. Cách xác định dòng kết thúc của lệnh if else trong Python 26](#_Toc167915932)

[3. Một vài bài tập thực hành lệnh if else trong Python 27](#_Toc167915933)

[4. Kết luận 28](#_Toc167915934)

[BÀI 2: VÒNG LẶP FOR TRONG PYTHON 29](#_Toc167915935)

[1. Cú pháp vòng lặp for trong Python 29](#_Toc167915936)

[2. Kết hợp hàm range trong vòng lặp for Python 30](#_Toc167915937)

[2.1. Trường hợp có một tham số 30](#_Toc167915938)

[2.2. Trường hợp có hai tham số 30](#_Toc167915939)

[2.3. Trường hợp có ba tham số 30](#_Toc167915940)

[3. Ví dụ khác với vòng lặp for trong Python 31](#_Toc167915941)

[4. Vòng lặp for lồng nhau trong Python 31](#_Toc167915942)

[5. Sử dụng lệnh if else trong vòng lặp for 31](#_Toc167915943)

[6. Lời kết 32](#_Toc167915944)

[BÀI 3: VÒNG LẶP WHILE TRONG PYTHON 33](#_Toc167915945)

[1. Cú pháp vòng lặp While trong Python 33](#_Toc167915946)

[2. Ví dụ với vòng lặp wihle trong Python 33](#_Toc167915947)

[3. Cẩn thận vòng lặp While vô hạn 34](#_Toc167915948)

[4. Sử dụng else trong while 34](#_Toc167915949)

[5. Chuyển đổi từ vòng lặp for sang vòng lặp while 35](#_Toc167915950)

[6. Lời kết 36](#_Toc167915951)

[BÀI 4: LỆNH BREAK VÀ CONTINUE TRONG PYTHON 37](#_Toc167915952)

[1. Lệnh break trong Python 37](#_Toc167915953)

[2. Lệnh continue trong Python 38](#_Toc167915954)

[3. Lời kết 39](#_Toc167915955)

[BÀI 5: BÀI TẬP PYTHON VỀ LƯU ĐỒ THUẬT TOÁN CƠ BẢN 40](#_Toc167915956)

[CHƯƠNG IV: HÀM 1](#_Toc167915957)

[BÀI 1: CÁCH DÙNG FUNCTION TRONG PYTHON 1](#_Toc167915958)

[1. Function là gì trong Python? 1](#_Toc167915959)

[2. Cách gọi function trong Python 1](#_Toc167915960)

[3. Ý nghĩa của docstring trong function 2](#_Toc167915961)

[4. Hàm return trong Python 2](#_Toc167915962)

[5. Phạm vi của biến trong function 3](#_Toc167915963)

[6. Thực hành hàm (function) trong Python nâng cao 3](#_Toc167915964)

[7. Hiểu rõ về tham số của Function trong Python 4](#_Toc167915965)

[7.1. Giá trị mặc định của tham số 4](#_Toc167915966)

[7.2. Truyền tham số theo key name 5](#_Toc167915967)

[7.3. Tham số không giới hạn 5](#_Toc167915968)

[8. Lời kết 6](#_Toc167915969)

[BÀI 2: HÀM ĐỆ QUY TRONG PYTHON 7](#_Toc167915970)

[1. Đệ quy trong Python là gì? 7](#_Toc167915971)

[2. Khử đệ quy trong Python 7](#_Toc167915972)

[3. Ưu điểm và nhược điểm của đệ quy Python 8](#_Toc167915973)

[4. Lời kết 8](#_Toc167915974)

[BÀI 3: HÀM LAMBDA TRONG PYTHON 9](#_Toc167915975)

[1. Lambda Python là gì? 9](#_Toc167915976)

[2. Tại sao phải sử dụng lambda trong Python? 9](#_Toc167915977)

[3. Một vài kết hợp của hàm lambda trong Python 10](#_Toc167915978)

[3.1. Kết hợp với filter() 10](#_Toc167915979)

[3.2. Kết hợp với map() 10](#_Toc167915980)

[4. Lời kết 10](#_Toc167915981)

[BÀI 4: BIẾN TOÀN CỤC VÀ BIẾN CỤC BỘ TRONG PYTHON 12](#_Toc167915982)

[1. Biến toàn cục trong Python là gì? 12](#_Toc167915983)

[2. Biến cục bộ trong Python là gì? 13](#_Toc167915984)

[3. Từ khóa global trong Python 13](#_Toc167915985)

[4. Lời kết 14](#_Toc167915986)

[BÀI 5: BÀI TẬP PYTHON: THỰC HÀNH TẠO HÀM TRONG PYTHON 15](#_Toc167915987)

[CHƯƠNG V: KIỂU DỮ LIỆU TRONG PYTHON 1](#_Toc167915988)

[BÀI 1: KIỂU DỮ LIỆU NUMBER TRONG PYTHON 1](#_Toc167915989)

[1. Number trong Python là gì? 1](#_Toc167915990)

[2. Số thập phân trong Python 2](#_Toc167915991)

[2.1. Chúng ta đang sai ở đâu? 2](#_Toc167915992)

[3. Khi nào sử dụng Decimal thay vì float 3](#_Toc167915993)

[4. Phân số 3](#_Toc167915994)

[5. Các hàm toán học 4](#_Toc167915995)

[6. Lời kết 5](#_Toc167915996)

[BÀI 2: KIỂU DỮ LIỆU STRING TRONG PYTHON 6](#_Toc167915997)

[1. Kiểu dữ liệu string trong Python là gì? 6](#_Toc167915998)

[1.1. Cú pháp tạo string trong Python 6](#_Toc167915999)

[1.2. Truy cập các ký tự của chuỗi string trong Python 7](#_Toc167916000)

[2. Thay đổi và xóa chuỗi string trong Python 8](#_Toc167916001)

[3. Các phép toán với chuỗi string trong Python 8](#_Toc167916002)

[3.1. Nối hai hoặc nhiều chuỗi 9](#_Toc167916003)

[3.2. Lặp qua từng ký tự trong chuỗi 9](#_Toc167916004)

[3.3. Kiểm tra chuỗi con trong chuỗi cha 9](#_Toc167916005)

[4. Các định dạng chuỗi 10](#_Toc167916006)

[4.1. Ký tự nháy đơn và nháy kép 10](#_Toc167916007)

[4.2. Ký tự thoát 10](#_Toc167916008)

[5. Lời kết 10](#_Toc167916009)

[BÀI 3: KIỂU LIST TRONG PYTHON 11](#_Toc167916010)

[1. Mảng trong Python là gì? 11](#_Toc167916011)

[2. Các thao tác trên mảng trong Python 11](#_Toc167916012)

[2.1. Thay đổi giá trị của phần tử 12](#_Toc167916013)

[2.2. Đếm chiều dài của mảng 12](#_Toc167916014)

[2.3. Lặp qua từng phần tử 12](#_Toc167916015)

[2.4. Kiểm tra một giá trị có tồn tại trong mảng 12](#_Toc167916016)

[3. Thêm phần tử vào mảng 12](#_Toc167916017)

[3.1. Xóa phần tử ra khỏi mảng 12](#_Toc167916018)

[3.2. Thiết lập List rỗng 13](#_Toc167916019)

[4. Lời kết 13](#_Toc167916020)

[BÀI 4: KIỂU DỮ LIỆU TUPLE TRONG PYTHON 14](#_Toc167916021)

[1. Kiểu dữ liệu tuple trong Python là gì? 14](#_Toc167916022)

[1.1. Ưu điểm của tuple so với mảng 14](#_Toc167916023)

[2. Cách tạo Tuple trong Python 14](#_Toc167916024)

[3. Truy cập các phần tử của tuple trong Python 15](#_Toc167916025)

[3.1. Truy cập thông qua chỉ mục index 15](#_Toc167916026)

[3.2. Số chỉ mục âm 16](#_Toc167916027)

[3.3. Slicing 16](#_Toc167916028)

[4. Thay đổi giá trị cho các phần tử Tuple trong Python 16](#_Toc167916029)

[5. Xóa Tuple 17](#_Toc167916030)

[6. Các phương thức trong Tuple 17](#_Toc167916031)

[7. Một vài thao tác khác với Tuple 17](#_Toc167916032)

[7.1. Kiểm tra phần tử có xuất hiện trong Tuple 17](#_Toc167916033)

[7.2. Duyệt qua từng phần tử của Tuple 18](#_Toc167916034)

[BÀI 5: KIỂU DỮ LIỆU SET TRONG PYTHON 19](#_Toc167916035)

[1. Set trong Python là gì? 19](#_Toc167916036)

[1.1. Cú pháp 19](#_Toc167916037)

[1.2. Khởi tạo Set rỗng 20](#_Toc167916038)

[2. Làm thế nào để thay đổi Set trong Python? 20](#_Toc167916039)

[3. Làm sao để xóa phần tử ra khỏi Set trong Python 21](#_Toc167916040)

[4. Các hành phép tập hợp của Set trong Python 22](#_Toc167916041)

[4.1. Set union 22](#_Toc167916042)

[4.2. Set Intersection 22](#_Toc167916043)

[4.3. Set Difference 23](#_Toc167916044)

[4.4. Set Symmetric Difference 23](#_Toc167916045)

[5. Các thao tác khác với Set trong Python 24](#_Toc167916046)

[5.1. Kiểm tra phần tử tồn tại trong Set 24](#_Toc167916047)

[5.2. Lặp qua các phần tử của Set 24](#_Toc167916048)

[6. Lời kết 24](#_Toc167916049)

[BÀI 6: KIỂU DỮ LIỆU DICTIONARY TRONG PYTHON 25](#_Toc167916050)

[1. Tạo dictionary Python 25](#_Toc167916051)

[2. Truy xuất phần tử của Dictionary trong Python 25](#_Toc167916052)

[3. Thay đối giá trị phần tử của Dictionary Python 25](#_Toc167916053)

[4. Xóa phần tử ra khỏi Dictionary 26](#_Toc167916054)

[5. Thay đổi các phần tử dictionary qua vòng lặp 27](#_Toc167916055)

[BÀI 7: EXCEPTIONS TRONG PYTHON, XỬ LÝ LỖI ĐƠN GIẢN 28](#_Toc167916056)

[1. Các exception trong Python có sẵn 28](#_Toc167916057)

[2. Chuyện gì xảy ra nếu không dùng exception? 28](#_Toc167916058)

[3. Lệnh try ... except trong Python 29](#_Toc167916059)

[4. Except bị thiếu exception 30](#_Toc167916060)

[5. Khai báo nhiều exception 31](#_Toc167916061)

[6. Khối finaly trong Exception 31](#_Toc167916062)

[7. Nhảy exception với từ khóa raise 32](#_Toc167916063)

[8. Tự tạo exception trong Python 32](#_Toc167916064)

[BÀI 8: BÀI TẬP PYTHON: THỰC HÀNH VỚI CÁC KIỂU DỮ LIỆU 33](#_Toc167916065)

[CHƯƠNG VI: OBJECT & CLASS 1](#_Toc167916066)

[BÀI 1: MODULE TRONG PYTHON: CÁCH TẠO VÀ SỬ DỤNG MODULE 1](#_Toc167916067)

[1. Module trong Python là gì? 1](#_Toc167916068)

[2. Cách tạo module trong Python 2](#_Toc167916069)

[3. Lệnh import trong Python 3](#_Toc167916070)

[3.1. Sử dụng import 4](#_Toc167916071)

[3.2. Sử dụng from-import 4](#_Toc167916072)

[4. Đổi tên module với AS trong Python 5](#_Toc167916073)

[5. Xem thuộc tính và phương thức của module 6](#_Toc167916074)

[6. Tải lại module với hàm reload 6](#_Toc167916075)

[7. Phạm vi của biến của module trong Python 6](#_Toc167916076)

[8. Xem các module có sẵn trong Python 7](#_Toc167916077)

[BÀI 2: PACKAGE TRONG PYTHON: CÓ GÌ KHÁC VỚI MODULE 11](#_Toc167916078)

[1. Package trong Python là gì? 11](#_Toc167916079)

[2. Thực hành tạo một package trong Python 12](#_Toc167916080)

[3. Tạo module trong Package Python 13](#_Toc167916081)

[4. Tạo package bên trong một package khác 14](#_Toc167916082)

[BÀI 3: Class trong Python (lập trình hướng đối tượng OOP) 16](#_Toc167916083)

[1. Lập trình hướng đối tượng là gì? 16](#_Toc167916084)

[2. Class trong Python là gì? 17](#_Toc167916085)

[3. self của class trong Python 18](#_Toc167916086)

[4. Tạo một instance object trong Python 19](#_Toc167916087)

[5. Các loại phương thức của class trong Python 20](#_Toc167916088)

[5.1. Phương thức khởi tạo 20](#_Toc167916089)

[5.2. Phương thức tĩnh static 21](#_Toc167916090)

[6. Kế thừa class trong Python 22](#_Toc167916091)

[7. Mức độ truy cập public / protected và private 24](#_Toc167916092)

[BÀI 4: HÀM KHỞI TẠO TRONG PYTHON, KHI NÀO THÌ NÊN DÙNG? 26](#_Toc167916093)

[1. Hàm khởi tạo trong Python là gì? 26](#_Toc167916094)

[2. Tham số truyền vào hàm khởi tạo Python 27](#_Toc167916095)

[3. Khi nào sử dụng hàm khởi tạo trong Python 28](#_Toc167916096)

[BÀI 5: KẾ THỪA TRONG PYTHON 29](#_Toc167916097)

[1. Kế thừa trong Python là gì? 29](#_Toc167916098)

[2. 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. 30](#_Toc167916099)

[3. Kế thừa trong Python với hàm khởi tạo 31](#_Toc167916100)

[BÀI 6: ĐA KẾ THỪA TRONG PYTHON 34](#_Toc167916101)

[1. Class kế thừa nhiều lớp trong Python 34](#_Toc167916102)

[2. Kế thừa đa cấp trong Python 34](#_Toc167916103)

[3. Thứ tự kế thừa trong đa kế thừa Python 35](#_Toc167916104)

[4. Cấp độ truy cập của đa kế thừa trong Python 37](#_Toc167916105)

[BÀI 7: SETTER VÀ GETTER TRONG PYTHON 39](#_Toc167916106)

[1. Getter và setter là gì? 39](#_Toc167916107)

[2. Tạo getter và setter theo cách thông thường 39](#_Toc167916108)

[3. Cú pháp getter và setter trong Python 40](#_Toc167916109)

[BÀI 8: OVERRIDE TRONG PYTHON (GHI ĐÈ TRONG KẾ THỪA) 42](#_Toc167916110)

[1. Override trong Python là gì? 42](#_Toc167916111)

[2. Cách gọi đến phương thức bị Override ở lớp cha 43](#_Toc167916112)

[3. Override trong kế thừa nhiều lớp 44](#_Toc167916113)

[BÀI 9: INTERFACE TRONG PYTHON 46](#_Toc167916114)

[1. Interface trong Python là gì? 46](#_Toc167916115)

[2. Tại sao phải sử dụng Interface trong Python? 46](#_Toc167916116)

[3. Lời kết 47](#_Toc167916117)

[BÀI 10: BÀI TẬP PYTHON: MODULE / OBJECT / CLASS 48](#_Toc167916118)

[CHƯƠNG VII: ADVANCED TOPICS 49](#_Toc167916119)

[BÀI 1: ITERATORS TRONG PYTHON 49](#_Toc167916120)

[1. Iterators trong Python là gì? 49](#_Toc167916121)

[1.1. Lặp tuple với Iterators trong Python 49](#_Toc167916122)

[1.2. Lặp chuỗi bằng Iterators trong Python 50](#_Toc167916123)

[1.3. Lặp mảng bằng Iterators trong Python 51](#_Toc167916124)

[2. lặp Iterators trong Python 52](#_Toc167916125)

[3. Tự tạo ra Iterators trong Python 53](#_Toc167916126)

[4. StopIteration trong Python 54](#_Toc167916127)

[BÀI 2: GENERATOR TRONG PYTHON 56](#_Toc167916128)

[1. Generators trong Python là gì? 56](#_Toc167916129)

[2. Cách tạo Generators trong Python 56](#_Toc167916130)

[3. Sự khác nhau giữa hàm generator và hàm bình thường 56](#_Toc167916131)

[4. Sử dụng vòng lặp trong Generator Python 58](#_Toc167916132)

[5. Sử dụng generator expression trong Python 59](#_Toc167916133)

[6. Tại sao nên dùng generator trong Python? 60](#_Toc167916134)

[BÀI 3: LIST COMPREHENSION TRONG PYTHON 63](#_Toc167916135)

[1. List comprehension là gì? 63](#_Toc167916136)

[2. Ví dụ về list comprehension trong Python 64](#_Toc167916137)

[BÀI 4: HÀM CLOSURE TRONG PYTHON 66](#_Toc167916138)

[1. Nested function là gì? 66](#_Toc167916139)

[2. Cách khai báo closure function trong Python 67](#_Toc167916140)

[3. Tại sao nên sử dụng closure function trong Python 68](#_Toc167916141)

[BÀI 5: HIGHER ORDER FUNCTIONS TRONG PYTHON 70](#_Toc167916142)

[1. Higher order function trong Python là gì? 70](#_Toc167916143)

[2. Các ví dụ về higher order function trong Python 71](#_Toc167916144)

[2.1. Functions là objects 71](#_Toc167916145)

[2.2. Function là một tham số 71](#_Toc167916146)

[2.3. Return về một function 72](#_Toc167916147)

[3. Điểm qua một chút về Decorator 73](#_Toc167916148)

[BÀI 6: DECORATOR TRONG PYTHON 74](#_Toc167916149)

[1. Python decorator là gì? 74](#_Toc167916150)

[2. Tạo decorator trong Pyhton bằng dấu @ 75](#_Toc167916151)

[3. Python decorator với hàm có tham số 76](#_Toc167916152)

[4. Multi decorator trong Python 77](#_Toc167916153)

[5. Decorator với class trong Python 78](#_Toc167916154)

[BÀI 7: XỬ LÝ FILE VÀ THƯ MỤC TRONG PYTHON 80](#_Toc167916155)

[1. Sơ lược về xử lý file trong Python 80](#_Toc167916156)

[2. Mở file với open() trong Python? 80](#_Toc167916157)

[3. Đóng kế nối với close() trong Python 81](#_Toc167916158)

[4. Ghi file trong Python 82](#_Toc167916159)

[5. Đọc dữ liệu của file trong Python 82](#_Toc167916160)

[6. Lời kết 84](#_Toc167916161)

[BÀI 8: SƠ LƯỢC DATE/TIME TRONG PYTHON 85](#_Toc167916162)

[1. Mốc thời gian nhỏ nhất trong Python 85](#_Toc167916163)

[2. Cách lấy thời gian hiện tại - localtime() 86](#_Toc167916164)

[3. Time tuple là gì? 86](#_Toc167916165)

[4. Hàm asctime() thay đổi format time 86](#_Toc167916166)

[5. Hàm sleep() trong Python 87](#_Toc167916167)

[6. Module datetime trong Python 87](#_Toc167916168)

[6.1. Tạo date object 88](#_Toc167916169)

[6.2. 2. So sánh 2 ngày khác nhau 88](#_Toc167916170)

[7. Calendar module trong Python 88](#_Toc167916171)

# TỔNG QUAN

## NGÔN NGỮ LẬP TRÌNH PYTHON

### Ngôn ngữ lập trình Python

Python là một ngôn ngữ lập trình phổ biến. Nó được tạo ra bởi Guido van Rossum và phát hành vào năm 1991.

|  |  |
| --- | --- |
| A screen shot of a computer  Description automatically generated  **Bảng 1.1:** Bảng xếp hạng các ngôn ngữ năm 2016 | Đến nay thì cộng đồng người sử dụng ngôn ngữ này rất đông, nếu so sánh từ bảng xếp hạng các ngôn ngữ năm 2016 thì Python đứng tứ 3 trong top 10 ngôn ngữ phổ biến nhất. |

Python được dùng để:

* Phát triển web (phía máy chủ)
* Phát triển phần mềm
* Toán học
* Các cript hệ thống

### Lý do nên học Python

Python luôn được đánh giá là một trong những ngôn ngữ lập trình phổ biến nhất thế giới. Python khá dễ học, vì vậy nếu bạn đang bắt đầu học bất kỳ ngôn ngữ lập trình nào thì Python có thể là lựa chọn tuyệt vời dành cho bạn.

Ngày nay, nhiều trường học, cao đẳng và đại học khác nhau đang dạy Python làm ngôn ngữ lập trình chính của họ. Có nhiều lý do chính đáng khác khiến Python trở thành lựa chọn hàng đầu của bất kỳ lập trình viên nào:

* Python là mã nguồn mở có nghĩa là nó có sẵn miễn phí.
* Python rất đơn giản và rất dễ học
* Python rất linh hoạt và có thể được sử dụng để tạo ra nhiều thứ khác nhau.
* Python có các thư viện phát triển mạnh mẽ bao gồm AI, ML, v.v.
* Python đang có nhu cầu cao và đảm bảo mức lương cao

### Ưu điểm của ngôn ngữ lập trình Python

* Python có thể được sử dụng trên máy chủ để tạo các ứng dụng web.
* Python có thể được sử dụng cùng với phần mềm để tạo quy trình công việc.
* Python có thể kết nối với các hệ thống cơ sở dữ liệu.
* Python có thể được sử dụng để xử lý dữ liệu lớn và thực hiện các phép toán phức tạp.
* Python có thể được sử dụng để phát triển phần mềm sẵn sàng sản xuất.

### Phiên bản Python

Phiên bản chính gần đây nhất của Python là Python 3, chúng ta sẽ sử dụng phiên bản này trong hướng dẫn này. Tuy nhiên, Python 2, mặc dù không được cập nhật bất cứ thứ gì ngoài các bản cập nhật bảo mật, vẫn khá phổ biến.

Trong hướng dẫn này, Python sẽ được viết bằng trình soạn thảo văn bản. Có thể viết Python trong Môi trường phát triển tích hợp, chẳng hạn như Thonny, Pycharm, Netbeans hoặc Eclipse, đặc biệt hữu ích khi quản lý các tập hợp tệp Python lớn hơn.

## CÀI ĐẶT MÔI TRƯỜNG PYTHON

Python hỗ trợ hầu hết trên các hệ điều hành hiện nay, và cách cài đặt nó cũng rất đơn giản. Tuy nhiên ở series này mình sẽ chỉ hướng dẫn mọi người cài đặt python trên môi trường Window.

### Download Python

Hiện tại Python được chia làm hai nhóm chính như sau:

* Nhóm version 2.x thuộc phiên bản cũ, hiện ít được sử dụng, chỉ được dùng chủ yếu cho những dự án cũ.
* Nhóm version 3x thuộc phiên bản mới và tương lai, vì vậy ta sẽ chọn version 3x để học theo series này nhé.

Trước tiên bạn cần lên trang chủ [python.org](https://freetuts.net/huong-dan-download-va-cai-dat-python-tren-window-194.html) để download phiên bản mới nhất và cũng là của serie này đó là phiên bản 3.6.1

A screenshot of a computer

Description automatically generated

**Hình 1.1:** Giao diện trang chủ [python.org](https://freetuts.net/huong-dan-download-va-cai-dat-python-tren-window-194.html) để download python

### Cài đặt Python trên Win 10

Chạy file vừa tải về và cài đặt theo các bước như sau:

A screen shot of a computer

Description automatically generated

Đợi quá trình cài đặt. Màn hình báo thành công.

A screenshot of a computer

Description automatically generated

Để chắc chắn lại là đã thành công Mở CMD lên và gõ lệnh python. Nếu như xuất hiện cửa sổ như hình thì bạn đã cài đặt thành công Python.

A computer screen with white text

Description automatically generated

### Kiểm tra phiên bản Python

Sau khi cài đặt xong xong. Bạn mở cửa sổ CMD lên và gõ câu lệnh sau:

|  |
| --- |
| python --version |

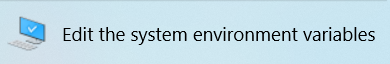
**Chú ý:**

Nếu bạn không có phiên bản Python cài đặt trên hệ thống của mình, thì cả hai câu lệnh bên trên sẽ khởi chạy Microsoft Store và chuyển hướng bạn đến trang ứng dụng của Python.

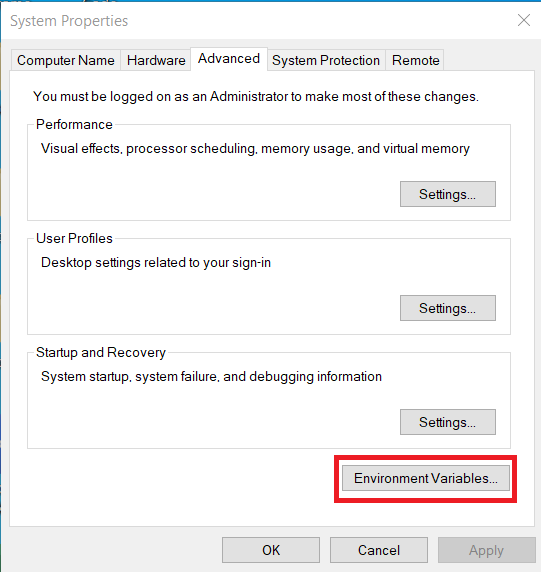
### Kiểm tra môi trường Python

Để kiểm tra môi trường trên máy tính, bạn làm như sau:

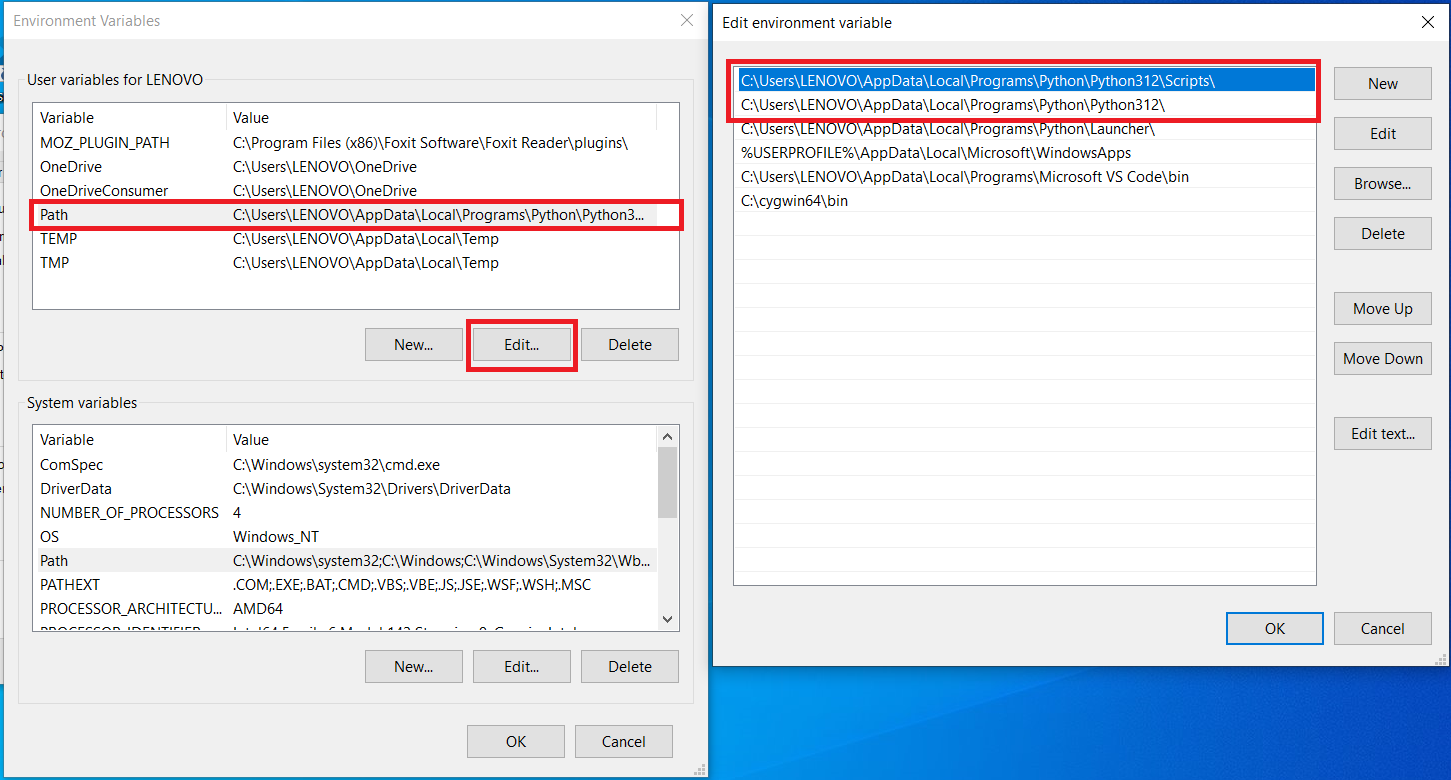
**Bước 1**: Tìm ***Edit the system environment variables***



**Bước 2**: Chọn ***Environment Variables …***



**Bước** 3: Nhấp vào **Path** Chọn **Edit**Quan sát thấy có đường dẫn như trên.



## PYTHON - VISUAL STUDIO CODE

Trong bài trước, chúng ta đã thực hiện cài đặt môi trường phát triển Python. Trong bài này, bạn sẽ bắt đầu biết sử dụng Visual Studio Code và viết một số chương trình trên ngôn ngữ Python và thực thi chạy nó trên Visual Studio Code.

Các bạn sẽ cài đặt Visual Studio Code nha =) **YouTube có hướng dẫn.**

Đầu tiên, chúng ta hãy mở Visual Studio Code lên nào !!!

A screenshot of a computer

Description automatically generated

Để bắt đầu lập trình một chương trình Python, trong Visual Studio Code, hãy tạo một tệp tin mới bằng cách gõ tổ hợp phím Ctrl + N hoặc bạn cũng có thể chọn từ Menu Tab File → New File như dưới đây:

A screenshot of a computer

Description automatically generated

Sau khi tạo tệp tin mới, giao diện của sổ của Visual Studio Code sẽ hiển thị như sau:

A screenshot of a computer

Description automatically generated

Mọi tệp tin tạo mới sẽ chưa lưu trong Visual Studio Code đều có tên mặc định là Untitled-x, với x là số hiệu nào đó.

Bây giờ, chúng ta đã có thể bắt đầu soạn thảo một chương trình Python đầu tiên. Ở đây, chúng ta sẽ tạo một chương trình đơn giản có mục đích là in ra một chuỗi "Hello world!" trên màn hình.

Nhập vào file vừa tạo đoạn mã sau:

|  |
| --- |
| print ('Hello world!') |

Sau khi gõ xong, chúng ta sẽ lưu file có tên hello.py. Bây giờ để có thể chạy được đoạn code trên thì các bạn di chuyển đến nơi file được lưu và mở CMD lên và gõ lệnh.

|  |
| --- |
| python hello.py |

Máy tính sẽ tự động mở 1 cửa sổ và hiển thị nội dung.

# THÀNH PHẦN

## KHAI BÁO BIẾN TRONG PYTHON

Trong bài trước, chúng ta đã làm quen với việc in một chuỗi ra màn hình với hàm print(). Với mỗi lần thay đổi giá trị muốn in, ta lại phải thay lại chuỗi cần in trong hàm print(). Điều này rất bất tiện, và trong nhiều trường hợp rất khó làm được, bởi người lập trình chẳng thể biết chuỗi cần in ra là gì!!!

Ví dụ, nếu có một yêu cầu hãy in ra tên của người đang đọc bài viết này thì chắc chắn là tôi sẽ không thể biết để đưa nó một cách thủ công vào hàm print được. Nếu máy tính chỉ hoạt động được như vậy thì nó chẳng có tác dụng gì. Đó là lí do chúng ta sinh ra các **biến**. Các ngôn ngữ lập trình cung cấp một phương thức để **lưu trữ và sử dụng lại dữ liệu** gọi là biến. Biến cho phép chúng ta lưu trữ các ngày tháng, ID của người dùng, các con số tính toán trung gian, các giá trị nhập vào bởi người dùng,...

Khi chúng ta muốn lưu lại giá trị nào đó để tiện cho việc sử dụng và thay thế sau này, chúng ta tạo ra một biến và gán cho nó giá trị muốn lưu trữ. Trong Python, để gán giá trị cho biến chúng ta sử dụng dấu bằng (=). Dấu bằng này còn được gọi là toán tử gán.

**Ví dụ 1:** Gán giá trị cho biến

|  |
| --- |
| account\_balance = 100  print(account\_balance) |

**Hiển thị**:

|  |
| --- |
| 100 |

Python là một ngôn ngữ đặc biệt, không cần phải đặt dấu chấm phẩy (;) đằng sau mỗi dòng lệnh.

### Biến trong Python là gì?

Trong Python, biến là một định danh dùng để lưu trữ dữ liệu trong quá trình biên dịch mã code Python. Tên biến hợp lệ chỉ gồm sự kết hợp của các chữ cái thường (a đến z) hoặc hoa (A đến Z), các chữ số (0 đến 9) và dấu gạch dưới \_.

**Lưu ý:**

* Tên biến bắt đầu bằng một chữ . **VD**: 1Number là một định danh ***KHÔNG*** hợp lệ, nhưng OneNumber là một định danh hợp lệ.
* Tên biến không được trùng với từ khóa của Python:  Từ khóa là những từ đặc trưng dành riêng trong ngôn ngữ lập trình, nó có tác dụng nhất định để chỉ định chương trình thực hiện một số chức năng nào đó. Chúng ta không thể sử dụng từ khóa để làm tên biến, tên hàm hoặc bất kỳ định danh nào khác bởi nó sẽ gây nhầm lẫn cho trình thông dịch Python làm cho chương trình thực hiện sai mục đích. Có 33 từ khóa trong Python 3.7. Con số này có thể thay đổi một chút theo thời gian, các phiên bản mới hơn có thể có thêm keyword bổ sung. Danh sách tất cả các từ khóa trong phiên bản mới nhất của Python được đưa ra bên dưới.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| False | await | else | import | pass |
| None | break | except | in | raise |
| True | class | finally | is | return |
| and | continue | for | lambda | try |
| as | def | from | nonlocal | while |
| assert | del | global | not | with |
| async | elif | if | or | yield |

* Tên biến không có khoảng trắng. Lí do, nếu bạn đặt tên biến có khoảng trắng thì chương trình dịch sẽ không hiểu và không phân biệt được đâu là từ khóa, đâu là tên biến hoặc các thành phần khác trong chương trình để dịch lại cho máy hiểu đúng.
* Tên biến trong Python có thể có độ dài bất kỳ. Bạn có thể đặt một tên cho đối tượng có 1 ký tự hoặc 1000 ký tự.
* Python là một ngôn ngữ phân biệt chữ hoa chữ thường. Điều này có nghĩa là, Variable và variable là hoàn toàn khác nhau và chỉ về hai đối tượng khác nhau.
* Luôn đặt tên có ý nghĩa. Mặc dù c = 10 là một tên hợp lệ, tuy nhiên nếu chúng ta viết count = 10 thì nó sẽ có ý nghĩa hơn và sẽ dễ dàng hơn để tìm ra nó đại diện cho điều gì khi bạn nhìn vào đoạn mã chương trình của mình sau một khoảng thời gian dài không đụng đến.
* Khi tên chứa nhiều từ, chúng ta có thể phân tách giữa các từ bằng dấu gạch dưới cho dễ hiểu, chẳng hạn như: Value\_Numbers

### Cách khai báo biến trong Python

Để khai báo biến thì ta sử dụng cú pháp như sau:

|  |
| --- |
| Tenbien = Giatri |

**Trong đó**:

* Tenbien là tên của biến mà các bạn muốn đặt.
* GiaTri là giá trị của biến mà bạn muốn gán.

**Lưu ý**: Trong Python, cú pháp khai báo biến sẽ không có khai báo kiểu dữ liệu. Vì vậy, phải luôn luôn gán một giá trị trong lúc khai báo biến, điều đó sẽ giúp Python nhận biết được biến đó sẽ lưu trữ kiểu dữ liệu gì.

**VD**: Khai báo các biến trong Python.

|  |
| --- |
| name = "NguyenPhiHao"  numbers = 50  pi = 3.14 |

### Một số cách gán biến đặc biệt

Ngoài cách gán biến chuẩn như trên, chúng ta cũng có thể thực hiện gán nhiều giá trị cho nhiều biến khác nhau và các giá trị, các biến được phân tách bởi dấu phẩy theo thứ tự lần lượt từ trái sang phải.

|  |
| --- |
| name, age, male = "NguyenPhiHao", 24 , "Male"  print(name)  print(age)  print(male) |

**Hiển thị:**

|  |
| --- |
| NguyenPhiHao  24  Male |

Nếu các biến cùng 1 giá trị thì có thể khai báo nhiều biến bằng 1 giá trị trên 1 lần khai báo.

|  |
| --- |
| a = b = c = 1999  print(a)  print(b)  print(c) |

**Hiển thị:**

|  |
| --- |
| 1999  1999  1999 |

### Một số ví dụ về đặt tên biến sai

**VD**: Khai báo các biến là từ khóa ***global***.

|  |
| --- |
| global = 1 |

**Hiển thị:**

|  |
| --- |
| SyntaxError: invalid syntax |

**VD**: Khai báo các biến sử dụng ký tự đặc biệt như: !, @, #, $, % ...

|  |
| --- |
| a@ = 0 |

**Hiển thị:**

|  |
| --- |
| We cannot  File "<interactive input>", line 1  a@ = 0  ^  SyntaxError: invalid syntax |

### Thay đổi giá trị của biến trong python

**VD**: Thay đổi giá trị của biến **Value**.

|  |
| --- |
| Value = 24  print(Value)  Value = "Nguyen Phi Hao"  print(Value) |

**Hiển thị:**

|  |
| --- |
| 24  Nguyen Phi Hao |

### Xóa biến trong Python

Để xóa một biến trong Python để giải phóng bộ nhớ thì bạn sử dụng lệnh **del**.

**VD:** Xóa biến name nên dòng lệnh print sẽ bị lỗi.

|  |
| --- |
| name = "NguyenPhiHao"  del name  print(name) |

## CÁC KIỂU DỮ LIỆU TRONG PYTHON

Python cũng giống như một số các ngôn ngữ bậc cao khác, khi ta khai báo biến thì kiểu dữ liệu của nó sẽ **tự động được detect**.

### Các kiểu dữ liệu trong python

Trong python, có tất cả 5 kiểu dữ liệu cơ bản trong python:

* Numbers
* String
* Lists
* Tuple
* Dictionary

**VD**: Các kiểu dữ liệu cơ bản trong python.

|  |
| --- |
| #string  age = 24  #integer  point = 8.9  #float  option = [1,2,3,4,5]  #lists  tuple = ('Nguyen Phi Hao', 24 , True)  #Tuple  dictionary = {"name": "Nguyen Phi Hao", "age": 24, "male": True}  #Dictionary |

### Cách thay đổi dữ liệu cho biến Python

Có 2 các để thay đổi kiểu dữ liệu trong python.

* Sử dụng toán tử gán.
* Ép kiểu dữ liệu trong python
* Ép kiểu dữ liệu ngầm trong Python
* Ép kiểu tường minh trong python
* Sử dụng các hàm ép kiểu

#### Sử dụng toán tử gán

Để thay đổi dữ liệu cho biến thì bạn chỉ việc sử dụng toán tử gán.

**VD:**

|  |
| --- |
| name = "NguyenPhiHao"  print(name)  name = "NguyenPhuHung"  print(name) |

**Hiển thị**:

|  |
| --- |
| NguyenPhiHao  NguyenPhuHung |

Trong bài này mình sẽ hướng dẫn các bạn cách chuyển đổi và ép kiểu trong Python, đây là thao tác thường được sử dụng trong thực tế khi làm việc với Python. Trước khi vào bài mới thì bạn hãy quay lại bài cũ để xem [Các kiểu dữ liệu trong Python](https://freetuts.net/cac-kieu-du-lieu-trong-python-3421.html) đã nhé, sau đó chúng ta sẽ tiếp tục với phần dưới đây.

À để kiểm tra kiểu dữ liệu của một biến thì ta dùng hàm type nhé các bạn, cú pháp như sau:

|  |
| --- |
| type(variable) |

**VD**: Khai báo thông tin cá nhân của mình.

|  |
| --- |
| name = "Nguyen Phi Hao"  print("Kiểu dữ liệu: ", type(name)) |

**Hiển thị**:

|  |
| --- |
| <class 'str'> |

#### Ép kiểu dữ liệu ngầm trong Python

Đây là thao tác tự động chuyển đổi một loại dữ liệu sang loại dữ liệu khác của Python, quá trình này không cần bất kỳ sự tham gia của lập trình viên.

Chúng ta hãy xem ví dụ dưới đây, trong đó Python chuyển đổi kiểu dữ liệu thấp hơn (số nguyên) sang kiểu dữ liệu cao hơn (float) để tránh mất dữ liệu.

|  |
| --- |
| num\_int = 123  num\_flo = 1.23    num\_new = num\_int + num\_flo    print("Kiểu dữ liệu của num\_int:",type(num\_int))  print("Kiểu dữ liệu của num\_flo:",type(num\_flo))    print("Giá trị của num\_new:",num\_new)  print("Kiểu dữ liệu của num\_new:",type(num\_new)) |

**Hiển thị**:

|  |
| --- |
| Kiểu dữ liệu của num\_int: <class 'int'>  Kiểu dữ liệu của num\_flo: <class 'float'>  Giá trị của num\_new: 124.23  Kiểu dữ liệu của num\_new: <class 'float'> |

Trong chương trình trên thì:

* Mình đã định nghĩa hai biến num\_int và num\_flo, sau đó tạo một biến num\_new để lưu trữ tổng của hai biến đó.
* Tiếp theo sẽ dùng hàm type để kiểm tra kiểu dữ liệu của cả ba biến, thật bất ngờ vì num\_new đã mang kiểu **float** vì đây là kiểu số lớn hơn kiểu **int**. Như vậy biến num\_new đã được chuyển đổi ngầm.

Bây giờ, hãy thử thêm một chuỗi và một số nguyên và xem Python xử lý thế nào.

**VD**: Bổ sung kiểu dữ liệu chuỗi (cao hơn) và kiểu dữ liệu số nguyên (thấp hơn)

|  |
| --- |
| num\_int = 123  num\_str = "456"    print("Kiểu dữ liệu của num\_int:",type(num\_int))  print("Kiểu dữ liệu của num\_str:",type(num\_str))    # Dòng này sẽ lỗi vì string và number không chuyển ngầm được  print(num\_int+num\_str) |

**Hiển thị**:

|  |
| --- |
| print(num\_int+num\_str)  TypeError: unsupported operand type(s) for +: 'int' and 'str'  Kiểu dữ liệu của num\_int: <class 'int'>  Kiểu dữ liệu của num\_str: <class 'str'> |

Như vậy mặc định Python không thể tự động chuyển đổi ngầm giữa string và number.

#### Ép kiểu trong Python tường minh

Trong ép kiểu tường minh thì lập trình viên sẽ chuyển đổi loại dữ liệu của một đối tượng thành loại dữ liệu cần thiết theo nhu cầu của họ.

Chúng ta sử dụng các hàm được xác định trước như int(), float(), str(), v.v để thực hiện chuyển đổi loại rõ ràng.

Chuyển đổi loại này cũng được gọi là **typecasting** vì người dùng ép (thay đổi) kiểu dữ liệu của các đối tượng. Cú pháp như sau:

|  |
| --- |
| required\_datatype (expression) |

Trong đó:

* required\_datatype: Là kiểu dữ liệu yêu cầu
* expression: Biến, biểu thức, … cần chuyển

**VD:** Ép kiểu từ string sang int trong Python.

|  |
| --- |
| num\_int = 123  num\_str = "456"    print("Kiểu dữ liệu của num\_int:",type(num\_int))  print("Kiểu dữ liệu của num\_str trước khi ép kiểu:",type(num\_str))    num\_str = int(num\_str)  print("Kiểu dữ liệu của num\_str sau khi ép kiểu:",type(num\_str))    num\_sum = num\_int + num\_str |

**Hiển thị**:

|  |
| --- |
| Kiểu dữ liệu của num\_int: <class 'int'>  Kiểu dữ liệu của num\_str trước khi ép kiểu: <class 'str'>  Kiểu dữ liệu của num\_str sau khi ép kiểu: <class 'int'>  Tổng của num\_int và num\_str: 579  Kiểu dữ liệu của sum: <class 'int'> |

Trong ví dụ trên thì bạn thấy mình đã sử dụng hàm int() để thực hiện chuyển đổi, ép kiểu một cách rõ ràng.

|  |
| --- |
| num\_str = int(num\_str) |

#### Bảng các hàm ép kiểu dữ liệu trong Python

Sau đây là bảng danh sách các hàm được sử dụng để kép kiểu trong Python, mời các bạn theo dõi.

|  |  |  |
| --- | --- | --- |
| **STT** | **Function** | **Description** |
| 1 | int(x [,base]) | Ép kiểu int |
| 2 | long(x [,base]) | Ép kiểu long int. |
| 3 | float(x) | Ép kiểu float. |
| 4 | complex(real ,[imag]) | Ép kiểu complex number. |
| 5 | str(x) | Ép kiểu string. |
| 6 | repr(x) | Ép thành chuỗi biểu thức. |
| 7 | eval(str) | Ép chuỗi sang object. |
| 8 | tuple(s) | Ép kiểu tuple. |
| 9 | list(s) | Ép kiểu list. |
| 10 | set(s) | Ép kiểu set. |
| 11 | dict(d) | Ép kiểu dictionary. |
| 12 | frozenset(s) | Ép kiểu frozen set. |
| 13 | chr(x) | Ép kiểu in sang kiểu char |
| 14 | unichr(x) | Ép kiểu int sang Unicode character. |
| 15 | ord(x) | Ép ký tự sang kiểu int. |
| 16 | hex(x) | Ép kiểu integer sang chuỗi thập lục phân. |
| 17 | oct(x) | Ép kiểu integer chuỗi bát phân. |

Như ví dụ trên bạn thấy 1 là kiểu số nguyên, 2.0 là kiểu số động và khi thực hiện phép toán thì kết quả sẽ mang kiểu số động.

Bạn cũng có thể sử dụng các hàm dựng sẵn như int (), float () và comlex () để chuyển đổi các kiểu dữ liệu một cách rõ ràng. Các hàm này thậm chí có thể chuyển đổi từ các chuỗi sang số và số sang chuỗi.

|  |
| --- |
| >>> int(2.3)  2  >>> int(-2.8)  -2  >>> float(5)  5.0  >>> complex('3+5j')  (3+5j) |

### Kiểm tra kiểu dữ liệu

Trong python, để kiểm tra kiểu dữ liệu của một biến thì chúng ta có thể sử dụng hàm type với cú pháp như sau:

|  |
| --- |
| type(data) |

Trong đó:

* data là biến mà chúng ta cần kiểm tra.

**VD**: Kiểm tra kiểu dữ liệu của các biến sau.

|  |
| --- |
| #string  name = "Nguyen Phi Hao"  print(type(name))  #int  age = 24  print(type(age))  #float  point = 8.9  print(type(point))  #list  option = [1,2,3,4,5]  print(type(option))  #Tuple  tuplet = ('Nguyen Phi Hao', 24 , True)  print(type(tuplet))  # dict  dictionary = {"name": "Nguyen Phi Hao", "age": 24, "male": True}  print(type(dictionary)) |

## KIỂU DỮ LIỆU NUMBERS

### Kiểu dữ liệu number là gì ?

Kiểu dữ liệu number dùng để lưu trữ những con số. Tuy nhiên, chúng ta có rất nhiều loại số khác nhau như số nguyên (int), số thực (float), số phức (complex).

* Integer và float được phân biệt bằng dấu chấm động (.). Nếu một số có dấu chấm động là kiểu float, ngược lại là kiểu int.
* Số phức được viết dưới dạng x + yj, trong đó **x** là phần thực và **y** là phần ảo.

Sử dụng hàm type() để kiểm tả một biến đang thuộc lớp nào.

### Cú pháp tạo number trong Python

**VD:** Khai báo kiểu dữ liệu numbers

|  |
| --- |
| # Output: <class 'int'>  print(type(5))  # Output: <class 'float'>  print(type(5.0))  # Output: <class 'complex'>  c = 5 + 3j  print(type(c)) |

Hàm **isinstance()** để kiểm tra xem nó có thuộc về một lớp cụ thể nào đó không.

**VD:** Khai báo kiểu dữ liệu numbers

|  |
| --- |
| # Output: True  print(isinstance(5, int))  # Output: True  print(isinstance(5.0, float))  # Output: True  print(isinstance(5 +3j, complex)) |

Trong khi số nguyên có thể có độ dài bất kỳ, số dấu phẩy động chỉ chính xác tối đa 15 vị trí thập phân (vị trí thứ 16 không chính xác).

Các số chúng ta xử dụng hàng ngày là hệ thống số thập phân (cơ sở 10). Nhưng các lập trình viên máy tính (nói chung là lập trình viên nhúng) cần phải làm việc với các hệ thống số nhị phân (cơ sở 2), hệ thập lục phân (cơ sở 16) và số bát phân (cơ sở 8).

Trong Python, chúng ta có thể biểu diễn các số này bằng cách đặt một tiền tố thích hợp trước số đó.

Bảng sau liệt kê các tiền tố này.

|  |  |
| --- | --- |
| **Number System** | **Prefix** |
| Binary | '0b' hoặc '0B' |
| Octal | '0o' hoặc '0O' |
| Hexadecimal | '0x' hoặc '0X' |

**VD:** Cách sử dụng các tiền tố prefix

|  |
| --- |
| # Output: 107  print(0B1101011)  # Output: 253 (251 + 2)  print(0XFB + 0b10)  # Output: 13  print(0O15) |

**Hiển thị**:

|  |
| --- |
| 107  253  13 |

Có thể chuyển đổi một từ cơ số này sang cơ số khác. Điều này còn được gọi là chuyển đổi cơ số.

**VD**: Các phép toán như phép cộng, phép trừ số nguyên sẽ tự động chuyển đổi cơ số sang kiểu float nếu một trong các toán hạng là float.

|  |
| --- |
| # Output: 3.0  number = 2 + 1.0  print(number)  print(type(number)) |

**Hiển thị**:

|  |
| --- |
| 3.0  <class 'float'> |

### Số thập phân trong Python

Lớp built-in trong Python có thể thực hiện một vài phép toán khiến chúng ta ngạc nhiên. Chúng ta đều biết rằng tổng của 1.1 và 2.2 là 3.3, tuy nhiên khi sử dụng toán tử so sánh bằng (==) thì điều đó là không đúng trong Python.

|  |
| --- |
| # Output: False  c = (1.1 + 2.2) == 3.3  print(c) |

**Tại sao lại là False ?**

Nó chỉ ra rằng các số dấu phẩy động được triển khai trong phần cứng máy tính dưới dạng phân số nhị phân, vì máy tính chỉ hiểu nhị phân (0 và 1). Vì lý do này, hầu hết các phân số thập phân mà chúng ta biết không thể được lưu trữ chính xác trong máy tính.

**Hãy lấy một ví dụ.** Chúng ta không thể biểu thị phân số 1/3 dưới dạng số thập phân. Điều này sẽ cung cấp cho 0.33333333 ... dài vô hạn và chúng ta chỉ có thể ước chừng nó.

Hóa ra phân số thập phân 0,1 sẽ dẫn đến phân số nhị phân dài vô hạn 0,000110011001100110011 ... và máy tính của chúng ta chỉ lưu trữ một số hữu hạn của nó.

Điều này sẽ chỉ xấp xỉ 0,1 nhưng không bao giờ bằng nhau. Do đó, đó là giới hạn của phần cứng máy tính của chúng tôi và không phải là lỗi trong Python.

|  |
| --- |
| # Output: 3.3000000000000003  1.1 + 2.2 |

Để khắc phục vấn đề này, chúng ta có thể sử dụng mô-đun **decimal.**

|  |
| --- |
| import decimal  # Output: 0.1  print(0.1)  # Output: Decimal('0.1000000000000000055511151231257827021181583404541015625')  print(decimal.Decimal(0.1)) |

Mô-đun này được sử dụng khi chúng ta muốn thực hiện các phép tính thập phân như đã học ở trường. Chúng ta biết 25,50 kg chính xác hơn 25,5 kg vì nó có hai chữ số thập phân đáng kể so với một.

|  |
| --- |
| from decimal import Decimal as D  # Output: Decimal('3.3')  print(D('1.1') + D('2.2'))  # Output: Decimal('3.000')  print(D('1.2') \* D('2.50')) |

Chúng ta có thể hỏi tại sao không thực hiện Decimal mỗi lần, thay vì float? Lý do chính là hiệu quả, float luôn luôn nhanh hơn.

### Khi nào sử dụng Decimal thay vì float

Chúng ta thường sử dụng Decimal trong các trường hợp sau.

* Khi đang làm cho các ứng dụng tài chính cần đại diện thập phân chính xác.
* Khi muốn kiểm soát mức độ chính xác cần thiết.
* Khi muốn thực hiện khái niệm về số thập phân có ý nghĩa.
* Khi muốn các hoạt động được thực hiện như chúng ta đã làm ở trường

### Phân số

Python cung cấp các hoạt động liên quan đến số phân số thông qua mô-đun **Fraction** của nó. Chúng ta có thể tạo các đối tượng Fraction theo nhiều cách khác nhau.

|  |
| --- |
| import fractions  # Output: 3/2  print(fractions.Fraction(1.5))  # Output: 5  print(fractions.Fraction(5))  # Output: 1/3  print(fractions.Fraction(1,3)) |

Trong khi tạo phân số từ kiểu float chúng ta có thể nhận được một số kết quả bất thường. Điều này là do biểu diễn số dấu phẩy động nhị phân không hoàn hảo như đã thảo luận trong phần trước.

May mắn thay, phân số cho phép chúng ta khởi tạo bằng chuỗi, đây là các tùy chọn ưa thích khi sử dụng số thập phân.

|  |
| --- |
| import fractions  # As float  # Output: 2476979795053773/2251799813685248  print(fractions.Fraction(1.1))  # As string  # Output: 11/10  print(fractions.Fraction('1.1')) |

Kiểu dữ liệu này hỗ trợ tất cả các toán tử cơ bản. Dưới đây là một vài ví dụ.

|  |
| --- |
| from fractions import Fraction as F  # Output: 2/3  print(F(1,3) + F(1,3))  # Output: 6/5  print(1 / F(5,6))  # Output: False  print(F(-3,10) > 0)  # Output: True  print(F(-3,10) < 0) |

### Các hàm toán học trong kiểu dữ liệu number

Python cung cấp các mô-đun như **math** và **random** để thực hiện các phép toán khác nhau như lượng giác, logarit, xác suất và thống kê, v.v.

**Module math**

|  |
| --- |
| import math  # Output: 3.141592653589793  print(math.pi)  # Output: -1.0  print(math.cos(math.pi))  # Output: 22026.465794806718  print(math.exp(10))  # Output: 3.0  print(math.log10(1000))  # Output: 1.1752011936438014  print(math.sinh(1))  # Output: 720  print(math.factorial(6)) |

**Module random**

|  |
| --- |
| import random  # Output: 16  print(random.randrange(10,20))  x = ['a', 'b', 'c', 'd', 'e']  # Get random choice  print(random.choice(x))  # Shuffle x  random.shuffle(x)  # Print the shuffled x  print(x)  # Print random element  print(random.random()) |

## KIỂU DỮ LIỆU STRING

### Kiểu dữ liệu string là gì ?

Một chuỗi ký tự hay  string là một tập hợp các ký tự (chữ, số, ký tự đặc biệt như !, @,#...) kết hợp với nhau. Các ký tự kết hợp với nhau và nằm trong một cặp dấu nháy đơn '' hoặc nháy kép "" được gọi là một chuỗi ký tự.

**Mở rộng:**

* Máy tính không hiểu ý nghĩa của các ký tự như con người mà chúng sẽ hiểu sẽ chuyển sang kiểu nhị phân (binary). Mặc dù bạn có thể thấy các ký tự trên màn hình của mình, nhưng bên trong nó được lưu trữ và thao tác dưới dạng kết hợp của 0 và 1.
* Việc chuyển đổi từ ký tự thành số được gọi là mã hóa và quá trình ngược lại gọi là giải mã. ASCII và Unicode là một số dạng mã hóa được sử dụng phổ biến.

### Cú pháp tạo string trong Python

Chuỗi có thể được tạo bằng cách đặt các ký tự bên trong cặp nháy đơn hoặc nháy kép. Nếu bạn muốn viết chuỗi trên nhiều dòng thì phải sử dụng ba dấu nháy, cách này thường được dùng để tạo **docstring**.

**VD:** Tạo chuỗi trong python

|  |
| --- |
| # Tất cả những ví dụ dưới đây là tương đương với nhau.  # Dùng nháy đơn  string = 'Hello World!'  print(string)  # Dùng nháy kép  string = "Hello World!"  print(string)  # Dùng ba nháy đơn trên nhiều dòng  string = '''Hello                  World!'''  print(string)  # Dùng ba nháy kép trên nhiều dòng  string = """Hello                  World!"""  print(string) |

**Hiển thị**:

|  |
| --- |
| Hello World!  Hello World!  Hello                  World!  Hello                  World! |

### Các ký tự đặc biệt trong chuỗi

Các ký tự đặc biệt khác:

* \n : Ngắt xuống dòng và bắt đầu dòng mời.
* \t : Đẩy nội dung phía sau nó cách 1 tab.
* \a : Chuông cảnh báo.
* \b : Xóa bỏ khoảng trắng phía trước nó.
* Ngoài ra bạn cũng có thể sử dụng để in ra các ký tự đặc biệt khác bằng việc sử dụng theo cú pháp \xnn, với n là 0->9, hoặc a->f hoặc A->F.

**VD:** Sử dụng ký tự đặc biệt **\t** trong chuỗi.

|  |
| --- |
| print("Website \t Hoccode.com") |

**Hiển thị**:

|  |
| --- |
| Website      Hoccode.com |

**VD:** Sử dụng ký tự đặc biệt **\n** trong chuỗi.

|  |
| --- |
| print("Name: Nguyen Phi Hao \nAge: 24") |

**Hiển thị**:

|  |
| --- |
| Name: Nguyen Phi Hao  Age: 24 |

### Hiển thị chuỗi trong python

Ngoài những cách in ra dữ liệu ở trên thì mọi người cũng có thể sử dụng các keyword định dạng cho kiểu giá trị và binding nó vào chuỗi.

**Cú pháp:**

|  |
| --- |
| print("%type" %(binding)) |

**Trong đó**:

* Type : Là các mã định dạng của kiểu dữ liệu các bạn muốn binding và thay thế vào vị trí đó.
* Binding : Là giá trị mà các bạn muốn binding vào vị trí được xác định trong chuỗi.

Type là các kiểu sau:

|  |  |
| --- | --- |
| **Định dạng** | **Mô tả** |
| %c | character |
| %s | chuỗi |
| %i | số nguyên |
| %d | số nguyên |
| %u | số nguyên |
| %o | bát phân |
| %x | thập lục phân (in thường) |
| %X | thập lục phân (in hoa) |
| %e | số mũ  (với e thường) |
| %E | số mũ  (với e hoa) |
| %f | số thực |
| %g | dạng rút gọn của %f and %e |
| %G | dạng rút gọn của %f and %E |

**VD:** Truyền binding vào chuỗi

|  |
| --- |
| Name = "Nguyen Phi Hao"  Full = "Chao mung ban %s da hoc toi bai String" %(Name)  print(Full) |

**Hiển thị**:

|  |
| --- |
| Chao mung ban Nguyen Phi Hao da hoc toi bai String |

Nếu bạn muốn truyền binding vào trong chuỗi thì mỗi giá trị bạn muốn binding cách nhau bởi 1 dấu ,

**VD:** Truyền nhiều binding vào trong chuỗi.

|  |
| --- |
| guy = "Ban"  doamin = "toidicode.com"  full = "Chao mung %s den voi website %s" %(guy, doamin)  print(full)  # Chao mung Ban den voi website toidicode.com |

**Hiển thị**:

|  |
| --- |
| Chao mung Ban den voi website toidicode.com |

### Truy cập các ký tự của chuỗi string trong Python

String được xem như là một **list** các ký tự nên bạn có thể truy cập đến các ký tự thông qua chỉ mục index của nó.

Chỉ mục bắt đầu từ 0, và nếu bạn cố gắng truy cập một ký tự nằm ngoài phạm vi chỉ mục thì sẽ xuất hiện lỗi **IndexError**.

Chỉ mục phải là một số nguyên, không thể sử dụng float hoặc các loại khác vì điều này sẽ dẫn đến **TypeError**.

Chỉ số -1 đề cập đến phần tử cuối cùng, -2 cho phần tử cuối cùng thứ hai, .. (đây là cách tính ngược của chuỗi).

Trong trường hợp các bạn muốn lấy nội dung của một đoạn chuỗi trong chuỗi đó thì có thể sử dụng cú pháp sau:

|  |
| --- |
| stringName[start:end] |

**Trong đó**:

* stringName : Là tên của biến chứa chuỗi, hoặc chuỗi.
* start : Là vị trí của ký tự bắt đầu lấy, nếu để trống start thì nó sẽ lấy từ 0.
* End : Là vị trí kết thúc (nó sẽ lấy trong khoảng từ start đến < end), nếu để trống end thì nó sẽ lấy đến hết chuỗi.

**VD:** Truy cập các phần tử trong chuỗi

|  |
| --- |
| str = 'Hello World!'  print('str = ', str)  # Ký tự đầu tiên  print('str[0] = ', str[0])  # Ký tự cuối cùng  print('str[-1] = ', str[-1])  #Lấy các kí tự thứ 2 đến thứ 6  print('str[1:5] = ', str[1:5])  # Lấy các kí tự từ thứ 6 đến ky tự thứ 3 tính từ cuối chuỗi  print('str[5:-2] = ', str[5:-2]) |

**Hiển thị**:

|  |
| --- |
| str =  Hello World!  str[0] =  H  str[-1] =  !  str[1:5] =  ello  str[5:-2] =   Worl |

Nếu truy cập phần tử không tồn tại, thì sẽ báo lỗi **IndexError**.

**VD:** Truy cập đến một phần tử không tồn tại trong chuỗi

|  |
| --- |
| str = 'Hello World!'  # Truy cập phần tử không tồn tại  print('str[15] = ', str[15]) |

**Hiển thị**:

|  |
| --- |
| IndexError: string index out of range |

Nếu truy cập phần tử với chỉ mục không phải là một số nguyên, thì sẽ báo lỗi **TypeError**.

**VD:** Truy cập phần tử với chỉ mục không phải là một số nguyên

|  |
| --- |
| str = 'Hello World!'  # Truy cập phần tử với chỉ mục không phải là một số nguyên  print('str[1.5] = ', str[1.5]) |

**Hiển thị**:

|  |
| --- |
| TypeError: string indices must be integers |

### Thay đổi và xóa chuỗi string trong Python

Để thay đổi chuỗi thì bạn có thể sử dụng chỉ mục index ... điều này là **sai** nhé các bạn. Chuỗi là bất biến nên bạn không thể thay đổi chuỗi, chỉ được phép gán một chuỗi khác vào biến.

**VD:** Thay đổi ký tự thứ 2 trong chuỗi.

|  |
| --- |
| # Thử thay đổi ký tự thứ 1 thành ký tự khác  str = 'Hello World!'  str[0] = "G" |

**Hiển thị**:

|  |
| --- |
| TypeError: 'str' object does not support item assignment |

Chúng ta không thể xóa một ký tự trong chuỗi nhưng có thể xóa hoàn toàn chuỗi bằng cách sử dụng từ khóa **del**, đây là từ khóa được sử dụng rất nhiều từ các bài học trước tới giờ.

**VD:** Thay đổi ký tự thứ 2 trong chuỗi.

|  |
| --- |
| # Thử thay đổi ký tự thứ 1 thành ký tự khác  str = 'Hello World!'  # Lệnh này sai, sẽ bị lỗi  del(str[1])  # Lệnh này đúng, nó xóa luôn biến str  del(str)  # Vì vậy khi in giá trị của str thì không tồn tại biến này  print(str) |

### Các phép toán với chuỗi string trong Python

Có nhiều phép toán được thực hiện với chuỗi string làm cho nó trở thành một trong những kiểu dữ liệu được sử dụng nhiều nhất trong Python.

#### Nối hai hoặc nhiều chuỗi

Phép toán nối hai hoặc nhiều chuỗi thành một chuỗi thì ta gọi là phép nối chuỗi.

* Để nối chuỗi thì ta sử dụng toán tử +.
* Nếu muốn lặp lại chuỗi nhiều lần thì dùng toán tử \*.

**VD:** Nối 2 chuỗi với nhau.

|  |
| --- |
| str1 = 'Hello'  str2 ='World!'  # Sử dụng +  print('str1 + str2 = ', str1 + str2)  # Sử dụng \*  print('str1 \* 3 =', str1 \* 3) |

**Hiển thị:**

|  |
| --- |
| str1 + str2 =  HelloWorld!  str1 \* 3 = HelloHelloHello |

Vì được xem là tập hợp của nhiều ký tự nên mỗi ký tự trong chuỗi được đánh một số chỉ mục và bắt đầu từ 0.

* Ký tự đầu tiên có chỉ mục = 0
* Ký tự cuối cùng có chỉ mục bằng tổng ký tự trừ đi 1.

Ngoài ra bạn có thể sử dụng các toán tử sau trên chuỗi:

* Bạn có thấy sử dụng cặp ngoặc vuông [] để lấy các ký tự trong chuỗi.
* Sử dụng dấu sao \* để lặp lại chuỗi.
* Sử dụng dấu cộng + để nối chuỗi.

**VD:**

|  |
| --- |
| string = 'Hello World!'    print(string)          # In ra toàn bộ chuỗi  print(string[0])       # In phân tử đầu tiên của chuỗi  print(string[2:5])     # In phần tử thứ 3 đến thứ 6, tại vì chỉ mục bắt đầu từ 0  print(string[2:])      # In từ vị trí thứ 3 đến cuối chuỗi  print(string \* 2)      # In chuỗi hai lần  print(string + "TEST") # Nối chuỗi |

Nếu viết hai chuỗi gần kề nhau thì cũng được xem là phép nối chuỗi. Còn nếu muốn nối hai chuỗi có nhiều dòng thì phải bổ sung thêm cặp ngoặc đơn.

**VD:**

|  |
| --- |
| # Đặt hai chuỗi gần nhau  print('Hello'' World!')  # Hai chuỗi gần nhau nhưng có xuống hàng  # Nên thêm cặp ngoặc đơn ()  print(('Hello'          ' World!')) |

**Hiển thị:**

|  |
| --- |
| Hello World!  Hello World! |

#### Lặp qua từng ký tự trong chuỗi

Nếu bạn muốn lặp qua từng ký tự trong chuỗi thì có thể kết hợp với ***vòng lặp for.***

**VD:**

|  |
| --- |
| count = 0  for letter in 'Freetuts.net':      if(letter == 't'):          count += 1  print(count,'ky tu t duoc tim thay') |

**Hiển thị:**

|  |
| --- |
| 3 ky tu t duoc tim thay |

#### Kiểm tra chuỗi con trong chuỗi cha

Tương tự, để kiểm tra một chuỗi con có xuất hiện trong chuỗi cha hay không thì ta dùng toán tử in và not in.

**VD:**

|  |
| --- |
| # Trả về True  print('f' in 'freetuts.net')  # Trả về False  print('f' not in 'freetuts.net') |

**Hiển thị:**

|  |
| --- |
| True  False |

## KIỂU DỮ LIỆU LIST

Trong Python, kiểu dữ liệu List hay còn gọi là Mảng; là một mảng danh sách các phần tử được sắp xếp với nhau, cho phép lưu trữ nhiều kiểu dữ liệu khác nhau bên trong nó, và có thể truy xuất đến các phần tử bên trong nó thông qua vị trí của phần tử đó trong list. Phần tử đầu tiên được đánh dấu là 0, phần tử cuối cùng được đánh dấu là tổng số phần tử trừ đi một.

Để khai báo kiểu List thì ta sử dụng cặp ngoặc vuông [], và mỗi phần tử sẽ được ngăn cách nhau bởi dấu phẩy.

**Cú pháp:**

|  |
| --- |
| TenList = [Gia tri 1, Gia tri 2,..., Gia tri N] |

**Trong đó:**

TenList là tên biến để lưu các giá trị của list.

Gia tri 1, Gia tri 2,..., Gia tri N là các giá trị của list.

**VD:** Khởi các một list

|  |
| --- |
| List1 = ["Nguyen Phi Hao", "Binh Thuan", 1999];  List2 = [1999, 2000, 2001, 20002, 2003];  List3 = ["Hao", "Hoa", "Hung"] |

**Hiển thị:**

|  |
| --- |
| ['Nguyen Phi Hao', 'Binh Thuan', 1999]  [1999, 2000, 2001, 20002, 2003]  ['Hao', 'Hoa', 'Hung'] |

Kiểu dữ liệu của các thành phần trong List không nhất thiết là phải giống nhau.

**VD:** Khai báo một List vừa có kiểu number vừa có kiểu string.

|  |
| --- |
| information = ["Nguyen Phi Hao", "Binh Thuan", "24"] |

Ngoài ra, bạn có thể tạo list thông qua đối tượng list của Python.

|  |
| --- |
| TheList = list(("apple", "banana", "cherry"))  print(TheList) |

**Hiển thị:**

|  |
| --- |
| ['apple', 'banana', 'cherry' |

### Các thao tác trên mảng trong Python

#### Thay đổi giá trị của phần tử

Như ta biết, list bản chất là array nên mỗi phần tử sẽ có một chỉ số đánh dấu riêng, vị trí đầu tiên là 0, tiếp theo là 1, 2, 3, 4, ... Ví dụ bạn có một List gồm 5 phần tử thì lần lược chỉ số đánh dấu của các phần tử là: 0, 1, 2, 3, 4. Như vậy để thay đổi giá trị của phần tử nào thì ta sẽ dựa vào số chỉ mục đánh dấu này

**VD:** Thay đổi phần tử thứ 1 trong mảng

|  |
| --- |
| TheList = ["apple", "banana", "cherry"]  print(TheList)  TheList [0] = "orange"  print(TheList) |

**Hiển thị:**

|  |
| --- |
| ['apple', 'banana', 'cherry']  ['orange', 'banana', 'cherry'] |

#### Đếm chiều dài của mảng

Để đếm chiều dài của mảng thì ta sử dụng hàm len.

**VD:** Đếm số lượng phần tử trong mảng TheList

|  |
| --- |
| TheList = ["apple", "banana", "cherry"]  print(len(TheList)) |

**Hiển thị:**

|  |
| --- |
| 3 |

#### Lặp qua từng phần tử

Để lặp qua từng phần tử thì ta sử dụng **vòng lặp for** hoặc **vòng lặp while**, kết hợp với hàm len để đếm tổng số phần tử của List.

**VD:** Lặp qua các phần tử trong mảng TheList

|  |
| --- |
| TheList = ["apple", "banana", "cherry"]  for index in range(len(TheList)):      print ('Current fruit :', TheList[index]) |

**Hiển thị:**

|  |
| --- |
| Current fruit : apple  Current fruit : banana  Current fruit : cherry |

**Lưu ý:** Biến index trong **VD** này chứa các giá trị là chỉ số của mảng ("0, 1, 2")

Có một cách đơn giản hơn là bạn sử dụng cú pháp basic như sau:

|  |
| --- |
| TheList = ["apple", "banana", "cherry"]  for index in TheList:      print ('Current fruit :',index) |

**Hiển thị:**

|  |
| --- |
| Current fruit : apple  Current fruit : banana  Current fruit : cherry |

**Lưu ý:** Biến index trong **VD** này chứa các giá trị của mảng ("apple", "banana", "cherry")

#### Kiểm tra một giá trị có tồn tại trong mảng

Ta phải sử dụng lênh if để kiểm tra.

**VD:** Kiểm tra chuỗi "apple" có tồn tại trong TheList không ?

|  |
| --- |
| TheList = ["apple", "banana", "cherry"]  if "apple" in TheList:      print("Does Apple exist in TheList") |

**Hiển thị:**

|  |
| --- |
| Does Apple exist in TheList |

#### Thêm phần tử vào mảng (List)

Trong Python phải sử dụng phương thức **append** của List. Phần tử được thêm sẽ nằm ở cuối List.

**VD:** Thêm phần tử "orange" vào trong TheList

|  |
| --- |
| TheList = ["apple", "banana", "cherry"]  TheList.append("orange")  print(TheList) |

**Hiển thị:**

|  |
| --- |
| ['apple', 'banana', 'cherry', 'orange'] |

#### Xóa phần tử ra khỏi mảng

**Cách 1**: Để xóa một phần tử ra khỏi List thì ta sử dụng phương thức remove() được tích hợp sẵn trong List.

**VD:** Chương trình này sẽ xóa phần tử có giá trị là "banana".

|  |
| --- |
| TheList = ["apple", "banana", "cherry"]  TheList.remove("banana")  print(TheList) |

**Hiển thị:**

|  |
| --- |
| ['apple', 'cherry'] |

**Cách 2**: Nếu bạn muốn xóa phần tử có số thứ tự index nào đó thì sử dụng phương thức pop(). Nếu bạn không truyền index vào thì mặc định nó sẽ xóa phần tử cuối cùng.

**VD:** Xóa phần tử "banana" sử dụng phương thức pop()

|  |
| --- |
| TheList = ["apple", "banana", "cherry"]  TheList.pop(1)  print(TheList) |

**Hiển thị:**

|  |
| --- |
| ['apple', 'cherry'] |

**Cách 3**: Sử dụng từ khóa del

**VD:** Sử dụng **del** còn có thể xóa phần tử thứ 2 "banana" trong mảng TheList.

|  |
| --- |
| TheList = ["apple", "banana", "cherry"]  del TheList[1]  print(TheList) |

**Hiển thị:**

|  |
| --- |
| ['apple', 'cherry'] |

Từ khóa del còn có thể xóa toàn bộ phần tử như sau.

**VD:** Sử dụng del còn có thể xóa toàn bộ phần tử trong mảng TheList.

***Lúc này, mảng TheList sẽ không tồn tại.***

|  |
| --- |
| TheList = ["apple", "banana", "cherry"]  del TheList  print(TheList) |

**Hiển thị:**

|  |
| --- |
| NameError: name 'TheList' is not defined |

#### Thiết lập List rỗng

Để thiết lập list rỗng thì bạn sử dụng phương thức clear().

**VD:** Sử dụng phương thức clear() để tạo một mảng TheList rỗng

***Lúc này, mảng TheList sẽ rỗng.***

|  |
| --- |
| TheList = ["apple", "banana", "cherry"]  TheList.clear()  print(TheList) |

**Hiển thị:**

|  |
| --- |
| [] |

### List lồng nhau

Do list có thể chứa nhiều kiểu dữ liệu khác nhau lên chúng ta hoàn toàn có thể khai báo một list chứa một hoặc nhiều list khác nhau.

**VD:**

|  |
| --- |
| Birth = [25, 12, 1999]  Address = ["Luong Son", "Bac Binh", "Binh Thuan"]  Information = ["Nguyen Phi Hao", Birth, Address]  print(Information) |

**Hiển thị:**

|  |
| --- |
| ['Nguyen Phi Hao', [25, 12, 1999], ['Luong Son', 'Bac Binh', 'Binh Thuan']] |

Đối với list lồng nhau như này thì chúng ta cũng truy xuất đến các phần tử như bình thường, theo cấp từ ngoài vào trong.

**VD**: Mình sẽ truy cập vào phần tử đầu tiên trong Birth

|  |
| --- |
| Birth = [25, 12, 1999]  Address = ["Luong Son", "Bac Binh", "Binh Thuan"]  Information = ["Nguyen Phi Hao", Birth, Address]  print(Information)  # Truy cập từ ngoài vào trong  Temp = Information[1]  Temp[0] = 20  print(Information)  # Hoặc có thể viết ngắn gọn như sau  Information[1][0] = 10  print(Information) |

**Hiển thị:**

|  |
| --- |
| ['Nguyen Phi Hao', [25, 12, 1999], ['Luong Son', 'Bac Binh', 'Binh Thuan']]  ['Nguyen Phi Hao', [20, 12, 1999], ['Luong Son', 'Bac Binh', 'Binh Thuan']]  ['Nguyen Phi Hao', [10, 12, 1999], ['Luong Son', 'Bac Binh', 'Binh Thuan']] |

#### Kiểu dữ liệu Tuple trong Python

Tuple cũng là một kiểu dữ liệu dạng danh sách giống như kiểu List, điểm khác biệt duy nhất là Tuple sử dụng **dấu ngoặc đơn**, còn List sử dụng **dấu ngoặc vuông**.

**VD**: Khai báo một kiểu Tuple.

|  |
| --- |
| information = ("Nguyen Phi Hao", "Binh Thuan", "Python") |

Điểm khác biệt thứ hai nữa đó là dữ liệu trong Tuple không thể thay đổi, nó giống như một hằng số vậy. Tuy nhiên nếu một phần tử của Typle là kiểu List thì ta có thể thay đổi dữ liệu cho phần tử đó.

Xem thêm tại bài [Tuple trong Python](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html).

#### Kiểu dữ liệu Dictionary trong Python

Kiểu từ điển (dictionary) là một loại bảng băm (hash table), nó sẽ lưu trữ dữ liệu ở dạng key => value nên việc truy xuất cực kì dễ dàng.

Nếu bạn đã từng học qua PHP thì có thể thấy nó giống với kiểu [Mảng kết hợp](https://freetuts.net/cac-kieu-du-lieu-trong-php-3.html), nó cũng giống với cấu trúc của [JSON](https://freetuts.net/csdl-can-ban/hoc-json) và [Object trong Javascript](https://freetuts.net/doi-tuong-object-trong-javascript-408.html).

**VD**: Khai báo thông tin cá nhân của mình.

|  |
| --- |
| information = {'name': "Nguyen Phi Hao", 'address': "Binh Thuan", 'age': "24"] |

Vì các key được đánh theo số hoặc chuỗi nên việc sắp xếp theo đúng thứ tự trong dictionary là không có. Nói cách khác là các phần tử trong dictionary là không có thứ tự.

Xem thêm kiểu dữ liệu [Dictoinary trong Python](https://freetuts.net/dictionary-trong-python-1961.html).

**VD**: Để in giá trị của biến ra màn hình thì ta sử dụng lệnh **print**.

|  |
| --- |
| name = "NguyenPhiHao"  print(name) |

## CHÚ THÍCH TRONG PYTHON

Trong bài này mình sẽ hướng dẫn các bạn cách tạo **comment trong Python**, đây là cách giúp bạn tạo ra những ghi chú trong lúc lập trình với ngôn ngữ Python.

Hầu hết các ngôn ngữ lập trình đều hỗ trợ tạo ghi chú chứ không riêng gì Python, vấn đề là cú pháp của mỗi ngôn ngữ sẽ khác nhau.

### Comment trong Python là gì?

Comment hay còn gọi là ghi chú, đây là những đoạn code mà trình biên dịch Python sẽ bỏ qua trong lúc chạy mã code.

Khi bạn phát triển dự án khổng lồ, hoặc bạn xây dựng một hàm nào đó thì cần lưu lại những lưu ý để sau này xem lại sẽ hiểu được những gì mà bạn đã code. Lúc này sử dụng cú pháp tạo comment sẽ giúp bạn giải quyết được vấn đề đó.

### Cách tạo comment trong Python

Chúng ta có hai cách để tạo comment, đó là tạo trên một dòng và tạo trên nhiều dòng.

#### Tạo trên một dòng

Để tạo trên một dòng thì ta sử dụng dấu thăng #.

|  |
| --- |
| # Nội dung comment |

Sau đây là một vài cách sử dụng.

**VD:**

|  |
| --- |
| # Comment 1  print("Say Hi!") # Comment 2 |

#### Tạo trên nhiều dòng

Để comment trên nhiều dòng, lúc này có hai giải pháp cho bạn.

Một là sử dụng nhiều dấu thăng.

|  |
| --- |
| # Dòng 1  # Dòng 2  # Dòng 3 |

Hai là sử dụng dấu ba chấm để bao quanh các dòng lại.

**VD:**

|  |
| --- |
| """  Đây là nội dung comment  trên nhiều dòng  """  print("Hi You!") |

Việc tạo comment sẽ giúp chương trình dễ đọc hơn.

Bản thân mình cũng vậy, nhiều lúc quên comment và hôm sau xem lại code phải mất một thời gian đọc lại từ đầu. Vì vậy hãy tập thói quen comment khi lập trình với Python nhé.

## CÁC TOÁN TỬ TRONG PYTHON

Trong bài này mình sẽ giới thiệu đến các bạn danh sách các toán tử trong Python, đây là những toán tử được sử dụng rất thường xuyên khi lập trình Python.

Vấn đề muôn thuở, khi nói đến toán tử của các ngôn ngữ lập trình thì ta thường nhắc đến toán tử toán học, toán tử so sánh, toán tử gán, toán tử logic, toán tử quan hệ .. và trong bài này ta sẽ sẽ tìm hiểu đến chúng.

Thông qua bảng danh sách các toán tử trong Python này sẽ giúp bạn hiểu rõ hơn về cách sử dụng.

### Toán tử toán học trong Python

Toán tử toán học là những toán tử về cộng / trừ / nhân / chia ... trong toán học.

|  |  |
| --- | --- |
| **Toán tử** | **Ý nghĩa** |
| + | Cộng |
| - | Trừ |
| \* | Nhân |
| / | Chia |
| % | Chia lấy dư |
| \*\* | Tính lũy thừa |

**VD**: Hãy chạy ví dụ sau để hiểu rõ hơn về toán tử toán học.

|  |
| --- |
| a = 10  b = 20  print("Cộng: " + str(a + b))  print("Trừ: " + str(a - b))  print("Nhân: " + str(a \* b))  print("Chia: " + str(a / b))  print("Chia lấy dư: " + str(a % b))  print("Số mũ: " + str(a \*\* b)) |

**Hiển thị**:

|  |
| --- |
| Cộng: 30  Trừ: -10  Nhân: 200  Chia: 0.5  Chia lấy dư: 10  Số mũ: 100000000000000000000 |

### Toán tử so sánh

**Lưu ý**:

* Trong Python thì giá trị True hoặc False phải ghi hoa chữ cái đầu tiên, nếu không Python sẽ hiểu nhầm đó là một biến.
* Toán tử so sánh được dùng để so sánh mối quan hệ giữa hai toán hàng, kết quả nó trả về là một kiểu boolean (True hoặc False).

Giả sử ta có hai toán hạng A và B, ta có bảng các toán tử so sánh dưới đây:

|  |  |
| --- | --- |
| **Toán tử** | **Ý nghĩa** |
| == | Trả về True nếu A bằng B, False nếu a khác B |
| != | Trả về True nếu A khác B, False nếu A bằng B |
| <> | Giống với toán tử != |
| > | Trả về True nếu A > B, False nếu A bé hơn hoặc bằng B |
| < | Trả về True nếu A < B, False nếu A lớn hơn hoặc bằng B |
| >= | Trả về True nếu A lớn hơn hoặc bằng B, ngược lại trả về False |
| <= | Trả về True nếu A bé hơn hoặc bằng B, ngược lại trả về False |

**VD:**

|  |
| --- |
| a = 10  b = 20    print("So sánh bằng: "              + str(a == b))  print("So sánh không bằng: "        + str(a != b))  print("So sánh lớn hơn: "           + str(a > b))  print("So sánh bé hơn: "            + str(a < b))  print("So sánh lớn hơn hoặc bằng: " + str(a >= b))  print("So sánh bé hơn hoặc bằng: "  + str(a <= b)) |

**Hiển thị**:

|  |
| --- |
| So sánh bằng: False  So sánh không bằng: True  So sánh lớn hơn: False  So sánh bé hơn: True  So sánh lớn hơn hoặc bằng: False  So sánh bé hơn hoặc bằng: True |

### Toán tử gán

Chúng ta đã từng sử dụng toán tử gán ở những bài học trước rồi đấy các bạn, đó là toán tử =.

Toán tử gán được dùng để gán dữ liệu cho một biến nào đó.

Xét ví dụ dưới đây.

|  |
| --- |
| a = 4  b = 2    c = a + b   # c = 6  print("Giá trị c: " + str(c)) |

**Hiển thị**:

|  |
| --- |
| Giá trị c: 6 |

Toán tử gán có thể kết hợp với Toán tử toán học được biến thể ra nhiều thể loại khác nhau.

|  |  |
| --- | --- |
| **Toán tử** | **Ý nghĩa** |
| c = a + b | Gán giá trị a + b vào biến c |
| a += b | Tương đương với a = a + b |
| a -= b | Tương đương với a = a - b |
| a \*= b | Tương đương với a = a \* b |
| a /= b | Tương đương với a = a / b |
| a %= b | Tương đương với a = a % b |
| a \*\*= b | Tương đương với a = a \*\* b |

**VD:**

|  |
| --- |
| a = 4  b = 2    c = a + b   # c = 6  a += b      # a = a + b  a -= b      # a = a - b  a \*= b      # a = a \* b  a /= b      # a = a / b  a %= b      # a = a % b  a \*\*= b     # a = a \*\* b |

**Hiển thị**:

|  |
| --- |
| So sánh bằng: False  So sánh không bằng: True  So sánh lớn hơn: False  So sánh bé hơn: True  So sánh lớn hơn hoặc bằng: False  So sánh bé hơn hoặc bằng: True |

### Toán tử logic trong Python

Toán tử logic là những loại toán tử được dùng để kiểm tra sự liên kết giữa hai biểu thức, kết quả nó trả về là True hoặc False.

Chúng ta có ba toán tử thôn dụng gồm:

|  |  |
| --- | --- |
| **Toán tử** | **Ý nghĩa** |
| A and B | Trả về True nếu cả A và B có giá trị True, ngược lại trả về False |
| A or B | Trả về True nếu một trong hai A hoặc B có giá trị True, ngược lại trả về False |
| not(A) | Trả về True nếu A là False, và trả về False nếu A là True. Đây là toán tử phủ định |

**VD:** Bạn hãy thử chạy các lện sau để xem kết quả.

|  |
| --- |
| A = True  B = False    print (A and B) # False  print (A or B)  # True  print (not A)   # False |

Trên là tổng hợp 4 nhóm toán tử trong Python được sử dụng nhiều nhất. Thực tế vẫn còn rất nhiều loại toán tử khác, nhưng do chúng ít được sử dụng nên mình sẽ không liệt kê ra ở đây nhé.

# VÒNG LẶP

## CÂU LỆNH IF ELSE TRONG PYTHON

Trong bài này chúng ta sẽ học cách sử dụng câu **lệnh if else trong Python**, đây là lệnh giúp rẻ nhánh chương trình và được sử dụng rất nhiều trong lúc lập trình với Python.

Nói đến lệnh if else thì mình xin nhắc lại một chút kiến thức về toán học, đó là **mệnh đề**.

Như ta biết thì mệnh đề thường sẽ trả lời các câu hỏi như "nếu đúng ... thì làm ... ", nghĩa là nếu thỏa mãn điều kiện thì làm công việc gì đó. Nếu như bạn đã học qua một số ngôn ngữ khác như [PHP](https://freetuts.net/hoc-php), [C++](https://freetuts.net/hoc-c/hoc-c++-can-ban-va-nang-cao), [C#](https://freetuts.net/hoc-csharp) thì chúng ta có cú pháp là:

|  |
| --- |
| if (expression){      //statement  }  else {      // statement  } |

Nhưng với Python thì cú pháp của lệnh if else hoàn toàn khác, và khác như thế nào thì chúng ta bắt đầu tìm hiểu nhé.

### Cú pháp lệnh if else trong Python

Vì trong Python không dùng dấu ngoặc nhọn {} để tạo ra các block nên cú pháp nó hơi khác biệt một chút.

Ta dùng dấu hai chấm : để ngăn cách lệnh if, các đoạn code phía dưới sẽ được thụt vào một tab và nó tạo thành một khối.

#### Cú pháp if else

**Cú pháp**

|  |
| --- |
| if expression:      #statement  else:      #statement |

**VD:**

|  |
| --- |
| Bien = 100  if Bien == 100:      print("Gia tri cua Bien = 100")  else :      print("Gia tri cua Bien khac 100") |

**Hiển thị**:

|  |
| --- |
| Gia tri cua Bien = 100 |

#### Cú pháp if else lồng nhau

**Cú pháp**

|  |
| --- |
| if expression1:      if expression2:          #statment      #statment  else:      #statment |

**VD:**

|  |
| --- |
| Bien = 100  #Câu if thứ nhất  if Bien == 100:      Ten = "thehalfheart"      # Câu if thứ hai      if Ten == "thehalfheart":          print("Gia tri cua ten la thehalfehart")      print("Gia tri cua Bien = 100")  else:      print("Gia tri cua Bien khac 100") |

**Hiển thị**:

|  |
| --- |
| Gia tri cua ten la thehalfehart  Gia tri cua Bien = 100 |

#### Cú pháp elif

Nghe **elif** có vẻ hơi khác thường nhỉ, thông thường trong các ngôn ngữ PHP, C, C++ thì **else if**chứ ko giống Python (**elif**)

**Cú pháp**

|  |
| --- |
| if expression1:      #statement  elif expression2:      #statement |

**VD:**

|  |
| --- |
| Bien = 200  if Bien == 100:      print("Gia tri Bien = 100")  elif Bien == 200:      print("Gia tri cua Bien = 200") |

**Hiển thị**:

|  |
| --- |
| Gia tri Bien = 200 |

Như vậy trong Python ta không nhất thiết phải dùng cặp dấu ngoặc () để bao quanh điều kiện trong lệnh if, và ta cũng không dùng cặp dấu {} để bao quanh khối lệnh, mà thay vào đó ta dùng dấu hai chấm (:) ở đầu để khai báo bắt đầu đoạn code và kết thúc không có gì cả.

**\* Lưu ý**:

Trong Python các đoạn code được hiểu là một khối nếu nó cùng một tab (thẳng hàng từ trên xuống) cho tới khi gặp tab ngắn hơn, nên nếu bạn tự ý tab vào thì trình biên dịch sẽ báo lỗi ngay.

### Cách xác định dòng kết thúc của lệnh if else trong Python

Trong Python, để báo cho trình biên dịch biết đoạn code nằm ngoài một khối lệnh nào đó thì ta sẽ tap thụt ngoài 1 ô, nghĩa là các lệnh được bao khối bởi cấu trúc cây (cây nằm sâu bên trong là con của cây ngoài).

**VD:**

|  |
| --- |
| Bien = 100  if Bien == 100:      print("Gia tri bien = 100")  elif Bien != 100:      print("Gia tri cua bien khac 100")  print("Doan code nam ngoai lenh if else")  print("Doan code nam ngoai lenh if else") |

Các đoạn high light màu vàng mô phỏng các đoạn code cùng cấp trong python

**VD 1**: Thụt vào 1 tab nên bị hiểu nhầm là nằm trong khối else

|  |
| --- |
| Bien = 200  if Bien == 100:      print("Gia tri Bien = 100")  else:      print("Gia tri cua Bien khac 100")      print("Doan code nam ngoai lenh if else") |

**Hiển thị**:

|  |
| --- |
| Gia tri cua Bien khac 100  Doan code nam ngoai lenh if else |

Mục đích mình muốn cái dòng cuối cùng là nằm ngoài lệnh else, nhưng vì mình cho nó cùng cấp với khối else nên trình biên dịch hiểu nhầm nó là code nằm trong lệnh else.

**VD 2**: Không thụt vào 1 tab nên trình biên dịch hiểu dòng cuối cùng không nằm trong khối if else.

|  |
| --- |
| Bien = 100  if Bien == 100:      print("Gia tri Bien = 100")  else:      print("Gia tri cua Bien khac 100")  print("Doan code nay muc dich trong lenh else") |

**Hiển thị**:

|  |
| --- |
| Gia tri Bien = 100  Doan code nay muc dich trong lenh else |

Tương tự, dòng code cuối cùng mục đích mình muốn nó nằm trong khối else, nhưng vì mình không thụt vào một tab nên trình biên dịch đã hiểu lầm nó nằm ngoài khối else nên được chạy.

### Một vài bài tập thực hành lệnh if else trong Python

Tiếp theo ta sẽ làm một vài bài tập nhé, thông qua những bài tập này bạn sẽ hiểu được cách sử dụng lệnh if else.

**1. Chương trình kiểm tra số chẵn hay lẻ trong Python**:

|  |
| --- |
| number = 20;  if (number % 2 == 0):      print (str(number) + " là số chẵn")  else:      print (str(number) + " là số lẻ") |

**Hiển thị**:

|  |
| --- |
| 20 là số chẵn |

**2. Chương trình tìm số lớn nhất trong 3 số bằng Python**:

|  |
| --- |
| number1 = 20  number2 = 300  number3 = 70    # Giả sử số number1 là lớn nhất  max = number1    # Kiểm tra xem số number2 có lớn hơn số max không  # Nếu có thì phải đổi số lớn nhất thành number2  if (max < number2) :      max = number2    # Tương tự, ta sẽ kiểm tra cho số thứ 3  if (max < number3) :      max = number3    print ("Số lớn nhất là " + str(max)) |

**Chương trình in ra các ngày trong tháng bằng Python**:

Hãy viết chương trình in ra các ngày trong tuần bằng lệnh if else.

|  |
| --- |
| # Bạn hãy thay đổi số này để test nhé  dayofweek = 2    if dayofweek == 2 :      print ("Thứ hai")  elif dayofweek == 3 :      print ("Thứ ba")  elif dayofweek == 4 :      print("Thứ 4")  elif dayofweek == 5 :      print("Thứ 5")  elif dayofweek == 6 :      print("Thứ 6")  elif dayofweek == 7 :      print ("Thứ 7")  else :      print("Chủ nhật") |

### Kết luận

Tới bài này mình cảm thấy Python bắt đầu thú vị rồi đấy, các cú pháp code của nó không giống như C hay C++ mà theo một cách riêng biệt.

Nếu bạn mới học lập trình thì không sao chứ bạn nào mà học qua các ngôn ngữ kia rồi thì sẽ cảm thấy Python khó chịu vì khó quản lý code. Tuy nhiên bản thân mình thấy nó rất khắt khe trong việc code, bạn không thể tự mình thụt vào (tab) mà phải tuân theo quy luật của nó.

Qua bày này hy vọng bạn đã biết cách sử dụng lệnh if else trong Python. Hẹn gặp lại bạn ở bài tiếp theo nhé.

## VÒNG LẶP FOR TRONG PYTHON

Trong bài này chúng ta sẽ học cách sử dụng vòng lặp for trong Python, đây là vòng lặp đơn giản và dễ hiểu nhất nên được học đầu tiên trong series học Python cơ bản này.

Nói về vòng lặp thì bạn phải một chút về khái niệm: Vòng lặp là những hành động được lặp đi lặp lại theo một chu kì cho tới khi điều kiện lặp bị sai. Python có hai vòng lặp chủ yếu đó là vòng lặp while và vòng lặp for, tuy nhiên trong bài này ta sẽ tìm hiểu vòng lặp for trước nhé.

### Cú pháp vòng lặp for trong Python

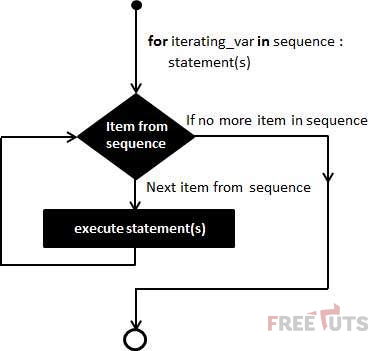
Nếu vòng lặp while thì sẽ không biết trước được tổng số lần lặp thì vòng lặp for lại khác, chúng ta có thể biết được tổng số lần lặp dựa vào cú pháp của nó như sau:

|  |
| --- |
| for iterating\_var in sequence:     statements(s) |

Trong đó:

* iterating\_var là biến dùng để lưu giá trị của các phần tử trong mỗi vòng lặp
* sequence la tập hợp các giá trị cần lặp
* statements(s) la các dòng lệnh xử lý trong thân vòng lặp

Hãy xem lược đồ của vòng lặp for dưới đây.



**VD**: Lặp và in ra từng ký tự của chữ freetuts.

|  |
| --- |
| for letter in 'Freetuts':     print ('Ký tự hiện tại :', letter) |

**Hiển thị**:

|  |
| --- |
| Ký tự hiện tại : F  Ký tự hiện tại : r  Ký tự hiện tại : e  Ký tự hiện tại : e  Ký tự hiện tại : t  Ký tự hiện tại : u  Ký tự hiện tại : t  Ký tự hiện tại : s |

Như bạn thấy, vòng lặp này sẽ lặp qua từng phần tử và in chúng ra màn hình.

**VD**: Lặp qua từng phần tử của mảng

|  |
| --- |
| fruits = ['banana', 'apple',  'mango']  for fruit in fruits:     print ('Fruit :', fruit) |

**Hiển thị**:

|  |
| --- |
| Fruit : banana  Fruit : apple  Fruit : mango |

Trong Python thì mảng hay còn gọi là List, vì vậy bạn đừng hiểu lần giữa hai khá niệm này nhé, tuy hai nhưng lại là một.

**\* Lưu ý**: Phạm vi bên trong vòng lặp phải được tab vào nhé các bạn, nếu bạn để cùng cấp với vòng lặp thì Python sẽ hiểu lầm là code ở bên ngoài.

### Kết hợp hàm range trong vòng lặp for Python

Mình sẽ không nói rõ hàm range trong bài này mà chỉ giải thích sơ lược và ứng dụng của nó trong vòng lặp thôi nhé.

Hàm range sẽ trả về một mảng trong đó tổng số phần tử sẽ phụ thuộc vào các tham số truyền vào. Cú pháp như sau:

|  |
| --- |
| range(start, end, step) |

Trong đó:

* start: là giá trị bắt đầu
* end là giá trị kết thúc
* step là khoảng cách giữa các phần tử, hay còn gọi la bước nhảy

#### Trường hợp có một tham số

Nếu bạn chỉ truyền một tham số n thì nó sẽ tạo một mảng từ 0 -> n - 1.

**Ví dụ**

|  |
| --- |
| for i in range(5):      print(i, end=', ') |

**Hiển thị**: Kết quả sẽ tạo một mảng gồm 5 phần tử có giá trị lần lượt từ 0 -> 5.

|  |
| --- |
| 0, 1, 2, 3, 4, |

#### Trường hợp có hai tham số

Nếu bạn truyền 2 tham số thì sẽ tạo một mảng với bước nhảy là 1, phần tử đầu của mảng là start, phần tử cuối cùng của mảng là end - 1.

**VD:**

|  |
| --- |
| for i in range(5, 10):      print(i, end=', ') |

Kết quả sẽ tạo một mảng gồm 5 phần tử có giá trị lần lượt là 5 -> 9

|  |
| --- |
| 5, 6, 7, 8, 9, |

#### Trường hợp có ba tham số

Trường hợp này sẽ tạo một mảng như trường hợp 2 nhưng vì bước nhảy là step nên tổng số phần tử sẽ nhỏ hơn.

**Ví dụ**

|  |
| --- |
| for i in range(1, 10, 2):      print(i, end=', ') |

Kết quả trả về một mảng 5 phần tử có giá lần lượt là 1, 3, 5, 7, 9 vì bước nhảy là 2.

|  |
| --- |
| 1, 3, 5, 7, 9, |

### Ví dụ khác với vòng lặp for trong Python

Bây giờ ta sẽ thực hành với một vài ví dụ khác với vòng lặp for nhé.

**Ví dụ**: Kết hợp hàm len để in ra các phần tử của mảng dựa vào chỉ số index.

|  |
| --- |
| fruits = ['banana', 'apple',  'mango']  for index in range(len(fruits)):     print 'Current fruit :', fruits[index] |

**Hiển thị**:

|  |
| --- |
| Current fruit : banana  Current fruit : apple  Current fruit : mango |

**VD**: Kết hợp với else xử lý lần lặp **bị từ chối** đầu tiên, tức là kết thúc vòng lặp thì chạy lệnh trong else.

|  |
| --- |
| for i in range(5):      print(i, end=', ')  else:      print ('Gia tri cua i la', (i)) |

**Hiển thị**:

|  |
| --- |
| 0, 1, 2, 3, 4, Gia tri cua i la 4 |

### Vòng lặp for lồng nhau trong Python

Không khác gì [lệnh if else trong Python](https://freetuts.net/if-else-trong-python-197.html), bạn có thể sử dụng vòng lặp for để tạo ra những đoạn code lồng nhau, nghĩa là vòng lặp này sẽ nằm bên trong một vòng lặp khác.

**VD**: Viết chương trình in ra bảng cửu chương bằng vòng lặp for.

|  |
| --- |
| for i in range(2, 10):      for j in range (2, 10):          print(i, " x ", j, ' = ', i \* j) |

**Hiển thị**:

|  |
| --- |
| 2  x  2  =  4  2  x  3  =  6  2  x  4  =  8  2  x  5  =  10  2  x  6  =  12  2  x  7  =  14  2  x  8  =  16  2  x  9  =  18  3  x  2  =  6  3  x  3  =  9  3  x  4  =  12  3  x  5  =  15  3  x  6  =  18  3  x  7  =  21  3  x  8  =  24  ... |

### Sử dụng lệnh if else trong vòng lặp for

Bạn có thể kết hợp sử dụng lệnh if else trong vòng lặp for để giải quyết bài toán, đây không phải là thao tác chỉ có ở mỗi Python mà hầu như ngôn ngữ nào cũng có.

**Ví dụ**: In ra các số chẵn và số lẻ từ 1 đến 100.

|  |
| --- |
| for i in range(1, 100):      if i % 2 == 0:          print(i, ' là số chẵn')      else :          print (i, ' là số lẻ'); |

**Hiển thị**:

|  |
| --- |
| 1  là số lẻ  2  là số chẵn  3  là số lẻ  4  là số chẵn  5  là số lẻ  6  là số chẵn  7  là số lẻ  8  là số chẵn  9  là số lẻ  10  là số chẵn  ... |

### Lời kết

Như vậy là bạn đã học xong cú pháp và cách sử dụng cơ bản của vòng lặp for trong Python. Vì đây là kiến thức cơ bản nên mình không giải thích quá nhiều mà thay vào đó là những ví dụ trực quan giúp bạn dễ học hơn. Chào tạm biệt và hẹn gặp lại các bạn ở bài tiếp theo đó là bài tìm hiểu number trong Python.

## VÒNG LẶP WHILE TRONG PYTHON

Trong bài này chúng ta sẽ tìm hiểu về **vòng lặp while trong Python**, đây là vòng lặp được sử dụng khá nhiều khi bạn làm các ứng dụng trong thực tế, không chỉ riêng ở Python mà ở các ngôn ngữ khác đều vậy.

Khác với vòng lặp for, vòng lặp while sẽ lặp trong những trường hợp bạn không xác định được tổng số lần lặp là bao nhiêu, nó phụ thuộc vào điều kiện dừng vòng lặp nằm trong lệnh while.

### Cú pháp vòng lặp While trong Python

Python là ngôn ngữ đơn giản nên cú pháp của nó cũng đơn giản. Sau đây là cú pháp chung của vòng lặp while.

|  |
| --- |
| while expression:     statement(s) |

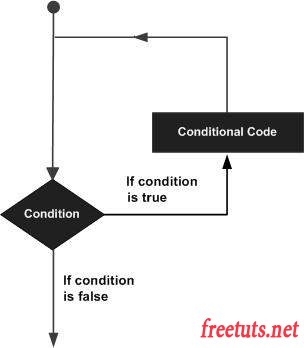
Trong đó:

* **statement(s)** là một lệnh đơn hoặc một tập lênh gồm nhiều lênh đơn dùng để xử lý chương trình.
* **expression**có thể là một biến hoặc một biểu thức, nhưng bắt buộc giá trị của nó phải là True hoặc False.

Như mình đã giới thiệu ở các bài trước, trong python sử dụng khoảng trắng để xác định các khối lệnh, điều này khác với các ngôn ngữ khác như PHP, C, C++ thì sử dụng cặp đóng mở {}. Nếu giá trị của expression là True thì những đoạn code trong khối lệnh của nó sẽ được thực thi, ngược lại nó sẽ bỏ qua và chạy các lệnh ở bên ngoài.

|  |
| --- |
| while expression:     // Lệnh trong vòng lặp     // Lệnh trong vòng lặp     // ...    // Các lệnh ở ngoài vòng lặp |

Hãy xem sơ đồ sau để hiểu rõ hơn.



### Ví dụ với vòng lặp wihle trong Python

Hãy thực hành một ví dụ nhỏ dưới đây để hiểu hơn về vòng lặp này: In ra các số từ 0 đến 9.

|  |
| --- |
| count = 0  while (count <= 9):      print ('Lượt đếm:', count)      count = count + 1    print ("Good bye!") |

**Hiển thị**:

|  |
| --- |
| Lượt đếm: 0  Lượt đếm: 1  Lượt đếm: 2  Lượt đếm: 3  Lượt đếm: 4  Lượt đếm: 5  Lượt đếm: 6  Lượt đếm: 7  Lượt đếm: 8  Lượt đếm: 9  Good bye! |

Như các bạn thấy, dòng Good bye! không bị lặp bởi vì nó nằm ngoài vòng lặp, còn lượt đếm sẽ bị lặp 10 lân biến count có giá trị ban đầu là 0 (count = 0), sau mỗi vòng lặp nó tăng lên 1 đơn vị (count = count + 1) và điều kiện dừng vòng lặp là count bé hơn hoặc bằng 9 (count <= 9).

### Cẩn thận vòng lặp While vô hạn

Vòng lặp while sẽ không biết chính xác tổng số lần lặp là bao nhiêu nên điều này rất nguy hiểm, vì nó có thể gây ra lặp vô hạn và dẫn đến ứng dụng bị chết, nên bạn phải tính toán thật kĩ điều kiện dừng vòng lặp.

Như ở ví dụ ở phần 2 thì điều kiện dừng là count <= 9, mà biến count có giá trị ban đầu là 0 nên sau mỗi vòng lặp mình khong tăng biến count lên 1 đơn vị thì sẽ dẫn đế lặp vô hạn.

Sau đây là ví dụ chương trình bị lặp vô hạn vì **expression**luôn trả về **True**.

**Bị lặp vô hạn**

|  |
| --- |
| # Chương trình Python return true  # Lặp vô hạn nên không được chạy test nhé  count = 0  while (count <= 9):     print ('Lượt đếm:', count)    print ("Good bye!") |

Hoặc bạn có thể tham khảo ví dụ dưới đây, mình yêu cầu người dùng nhập dữ liệu vô hạn vì điều kiện var == 1 luôn luôn đúng.

|  |
| --- |
| var = 1  while var == 1 :      num = input("Nhập số của bạn  :")      print ("Bạn đã nhập số: ", num)    print ("Good bye!") |

Ví dụ này cũng bị lặp vô hạn bởi vì điều kiện dừng luôn luôn đúng. Nếu bạn chạy mà không thoát được thì hãy nhấn CTR + C để thoát ra khỏi chương trình nhé.

### Sử dụng else trong while

Có một sự mới mẻ đối với vòng lặp while trong Python đó là bạn có thể kết hợp thêm từ khóa else để xử lý cho lần lặp đầu tiên **không được thực hiện**khi điều kiện lặp false.

|  |
| --- |
| count = 0  while count < 5:     print (count, " nhỏ hơn 5")     count = count + 1  else:     print (count, " lớn hơn 5") |

**Hiển thị**:

|  |
| --- |
| 0  nhỏ hơn 5  1  nhỏ hơn 5  2  nhỏ hơn 5  3  nhỏ hơn 5  4  nhỏ hơn 5  5  lớn hơn 5 |

Các bạn thấy đó, lần lặp thứ 6 sẽ không xảy ra nên code trong lệnh else sẽ được thực hiện.

### Chuyển đổi từ vòng lặp for sang vòng lặp while

Trong Python, bạn có thể chuyển đổi chương trình sử dụng vòng lặp for sang sử dụng vòng lặp while cực kì đơn giản.

Mình sẽ lấy một ví dụ mà ta đã thực hành ở bài trước nhé, đó là in ra bảng cửu chương.

**Đối với vòng lặp for:**

|  |
| --- |
| for i in range(2, 10):      for j in range (2, 10):          print(i, " x ", j, ' = ', i \* j) |

**Chuyển sang sử dụng vòng lặp while:**

|  |
| --- |
| i = 2;  j = 1;    while i <= 9:      while j <= 9:          print(i, " x ", j, ' = ', i \* j)          j += 1      i += 1      j = 1 # Thiết lập lại giá trị cho j cho bảng cửu chương tiếp theo |

**Hiển thị**:

|  |
| --- |
| 2  x  1  =  2  2  x  2  =  4  2  x  3  =  6  2  x  4  =  8  2  x  5  =  10  2  x  6  =  12  2  x  7  =  14  2  x  8  =  16  2  x  9  =  18  3  x  1  =  3  3  x  2  =  6  3  x  3  =  9  3  x  4  =  12  3  x  5  =  15  3  x  6  =  18  3  x  7  =  21  ... |

### Lời kết

Như vậy là bạn đã hiểu được cách sử dụng vòng lặp while trong python rồi phải không nào, qua bài học này hy vọng sẽ giúp ích được cho bạn khi tiếp cận với ngôn ngữ Python.

Và sau đây mình xin rút ra kết luận như sau:

* Vòng lặp while được dùng trong trường hợp không biết trước tổng số lần lặp, còn vòng lặp for thì biết trước.
* Vòng lặp while có điều kiện lặp là trả về True hoặc False nên sử dụng trong những bài toán phức tạp, còn vòng lặp while dùng trong trường hợp đơn giản.

Hẹn gặp lại bạn ở bài tiếp theo đó là bài vòng lặp For.

## LỆNH BREAK VÀ CONTINUE TRONG PYTHON

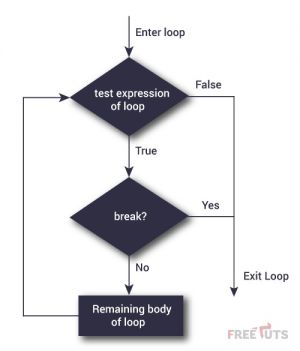
Trong bài viết này ta sẽ học cách sử dụng hai lệnh **break**và **continue**trong Python, công dụng của hai lệnh này trong Python là thay đổi luồng xử lý của vòng lặp.

Nếu bạn muốn kết thúc vòng lặp một cách đột ngột, hoặc bỏ qua một lần lặp nào đó thì có thể sử dụng hai lệnh này. Và cũng lưu ý là nó được sử dụng rất nhiều khi làm việc với Python, vì vậy bạn cần nắm vững nó nhé.

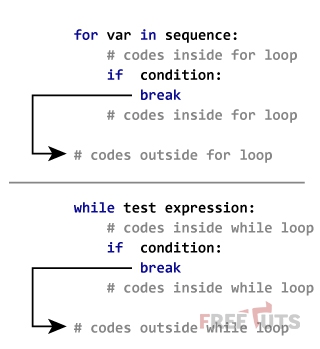
### Lệnh break trong Python

Lệnh break có công dụng là chấm dứt vòng lặp ngay lập tức, nó không quan tâm đến điều kiện dừng của vòng lặp như thế nào, miễn gặp lệnh này là vòng lặp sẽ kết thúc.

Trường hợp bạn sử dụng nhiều vòng lặp lồng nhau thì phạm vi tác dụng của break là vòng lặp gần nó nhất, tức là vòng lặp trong cùng. Hãy tham khảo lược đồ sau:



Và dưới đây là ví dụ sơ đồ hoạt động của nó ở trong hai vòng lặp for và while.



**Ví dụ**: Python break

|  |
| --- |
| # Use of break statement inside loop    for val in "string":      if val == "i":          break      print(val)    print("The end") |

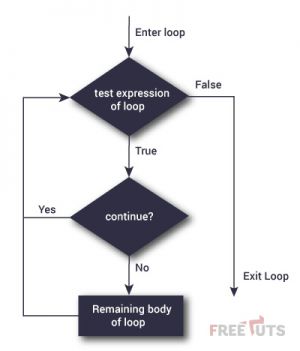
**Hiển thị**:

|  |
| --- |
| s  t  r  The end |

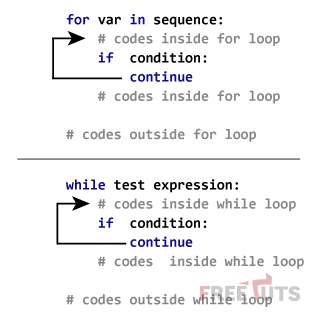
Chương trình này đã lặp qua lặp qua từng ký tự trong chuỗi "string", sau đó mỗi lần lặp sẽ kiểm tả đã đến kí tự "i" chưa, nếu đến rồi thì thoát khỏi vòng lặp, chưa thì tiếp tục lặp. Đấy là lý do tại sao nó không in đủ các ký tự trong chuỗi "string".

### Lệnh continue trong Python

Câu lệnh continue được sử dụng để bỏ qua phần còn lại của code bên trong thân vòng lặp và chỉ có tác dụng cho lần lặp hiện tại. Vòng lặp không kết thúc mà tiếp tục với lần lặp tiếp theo.



Lược đồ hoạt động ở trong hai vòng lặp while và for.



**VD**: Python continue

|  |
| --- |
| # Program to show the use of continue statement inside loops    for val in "string":      if val == "i":          continue      print(val)    print("The end") |

**Hiển thị**:

|  |
| --- |
| s  t  r  n  g  The end |

Giải thích tương tự như phần continue, chỉ có điểm khác là lần lặp cho ký tự "i" sẽ không chạy đoạn code print(val) ở phía dưới mà nhảy đến lần lặp tiếp theo nên sẽ không in ra giá trị này.

### Lời kết

Trên là cách sử dụng lệnh break và continue trong Python, đây là hai lệnh sử dụng khá nhiều nên rất quan trọng. Với lệnh break thì nó có nhiệm vụ là dừng mọi hoạt động của vòng lặp, thoát một cách đột ngột. Còn với lệnh continue thì nó sẽ bỏ qua lần lập hiện tại và nhảy đến lần lặp tiếp theo.

## BÀI TẬP PYTHON VỀ LƯU ĐỒ THUẬT TOÁN CƠ BẢN

Trong bài này mình sẽ giải một số bài tập Python cơ bản về phần lưu đồ thuật toán, đây là những bài thực hành giúp bạn nắm vững cấu trúc lệnh cơ bản của Python.

Danh sách các bài tập này được tổng hợp từ internet và mình sẽ sử dụng ngôn ngữ Python để giải nó. Mỗi bài tập mình sẽ đăng trên một bài riêng biệt nên bạn có thể click vào từng link và xem nhé.

**Danh sách bài tập Python cơ bản**

* [Kiểm tra dữ liệu là number hay string bằng Python](https://freetuts.net/ref/kiem-tra-du-lieu-la-number-hay-string-bang-python-394.html)
* [Tính tổng S(n) = 1 + 2 + 3 + … + n bằng Python](https://freetuts.net/ref/bai-tap-python-tinh-tong-sn-1-+-2-+-3-+-+-n-393.html)
* [Tính tổng S(n) = 1^2 + 2^2 + … + n^2 bằng Python](https://freetuts.net/ref/tinh-tong-sn-12-+-22-+-+-n2-bang-python-395.html)
* [Tính tổng S(n) = 1 + ½ + 1/3 + … + 1/n bằng Python](https://freetuts.net/ref/tinh-s-n-1-+-+-1-3-+-+-1-n-bang-python-396.html)
* [Tính tổng S(n) = ½ + ¼ + … + 1/2n bằng Python](https://freetuts.net/ref/tinh-s-n-+-+-12-n-bang-python-397.html)
* [Tính tổng S(n) = 1/3 + 1/5 + … + 1/(2n + 1)](https://freetuts.net/ref/tinh-tong-sn-1-+-1-3-+-15-+-+-1-2n-+-1-398.html)
* [Liệt kê tất cả các ước số của số nguyên dương n bằng Python](https://freetuts.net/ref/liet-ke-uoc-so-cua-so-nguyen-duong-n-bang-python-399.html)
* [Tính tổng tất cả các ước số của số nguyên dương N bằng Python](https://freetuts.net/ref/tinh-tong-tat-ca-cac-uoc-so-bang-python-400.html)
* [Tìm ước số lẻ lớn nhất của một số bằng Python](https://freetuts.net/ref/tim-uoc-so-le-lon-nhat-bang-python-401.html)
* [Kiểm tra một số có phải là số hoàn hảo bằng Python](https://freetuts.net/ref/kiem-tra-so-hoan-hao-bang-python-402.html)
* [Kiểm tra số chính phương bằng Python](https://freetuts.net/ref/kiem-tra-so-chinh-phuong-bang-python-403.html)
* [Kiểm tra số nguyên tố bằng Python](https://freetuts.net/ref/kiem-tra-so-nguyen-to-bang-python-404.html)
* [Đảo ngược một số bằng Python](https://freetuts.net/ref/dao-nguoc-mot-so-bang-python-406.html)
* [In ra từng ký tự của một số bằng Python](https://freetuts.net/ref/in-ra-tung-ky-tu-cua-mot-so-bang-python-405.html)
* [Giải phương trình bậc nhất một ẩn bằng Python (ax + b = 0)](https://freetuts.net/ref/giai-phuong-trinh-bac-nhat-mot-an-bang-python-407.html)
* [Giải phương trình bậc hai một ẩn bằng Python (ax^2 + bx + c = 0)](https://freetuts.net/ref/giai-phuong-trinh-bac-hai-mot-an-bang-python-408.html)

# HÀM

## CÁCH DÙNG FUNCTION TRONG PYTHON

Trong bài này chúng ta sẽ tìm hiểu về **function trong Python**, bạn sẽ được học khái niệm về function là gì? cách khai báo và đặt tên function như thế nào? Sau đó làm một vài ví dụ đơn giản để giúp bạn hiểu rõ hơn.

Trước khi học sâu vào function thì mình đưa ra một bất cập khi sử dụng cách code từ bài đầu đến giờ nhé. Giả sử bạn viết chương trình [Kiểm tra số nguyên tố bằng Python](https://freetuts.net/ref/kiem-tra-so-nguyen-to-bang-python-404.html), và bạn sử dụng theo cách ở bài viết này thì đã thực hiện thành công. Tuy nhiên, nếu trong một chương trình mình cần kiểm tra số nguyên tố ở **100 vị trí** khác nhau thì bạn phải code **100 lần** như vậy. Điều này gây ra tình trạng code bị dư thừa, khó bảo dưỡng và nâng cấp sau này.

Vậy có cách nào giải quyết không? Thật may mắn là các ngôn ngữ lập trình hiện nay đều hỗ trợ function giúp bạn xử lý nhanh gọn.

### Function là gì trong Python?

Trong Python, function hay còn gọi là hàm, nó là tổng hợp một đoạn code gồm một hoặc nhiều lệnh nhằm giải quyết một bài toán cụ thể nào đó.

Function ra đời giúp lập trình viên dễ dàng tối ưu chương trình hơn, bởi ta có thể sử dụng function nhiều lần mà không cần phải định nghĩa lại, ta gọi đây là hướng lập trình modun. Việc tái sử dụng này giúp code ngắn và gọn hơn rất nhiều.

**Cú pháp:**

|  |
| --- |
| def function\_name(parameters):      """docstring"""      statement(s) |

Trong đó các thành phần được giải thích như sau:

* def là từ khóa bắt buộc dùng để tạo function, nó được đặt tại vị trí đầu tiên của function
* function\_name là tên của function mà bạn muốn đặt, ví dụ bạn viết function kiểm tra số lớn nhất thì có thể đặt tên là number\_max.
* docstring là phần mô tả ý nghĩa của function, cái này do bạn tự nhập vào để sau này dễ dàng hiểu ý nghĩa của function mà không cần phải đọc lại toàn bộ code.
* statement(s) là những dòng lệnh xử lý bên trong function
* parameters là các tham số truyền vào của function

**Ví dụ**

|  |
| --- |
| def greet(name):      """Hàm này sẽ hiển thị câu chào!"""      print("Chào, " + name + ". Rất vui khi gặp bạn!") |

Quy tắc đặt tên của function giống như đặt tên biến vậy, nghĩa là dùng các chữ cái không dấu, không được chứa kí tự đặc biệt, có thể chứa số nhưng nó không được đứng ở vị trí đầu tiên.

Nếu bạn tạo nhiều function thì tên của chúng không được trùng nhau.

### Cách gọi function trong Python

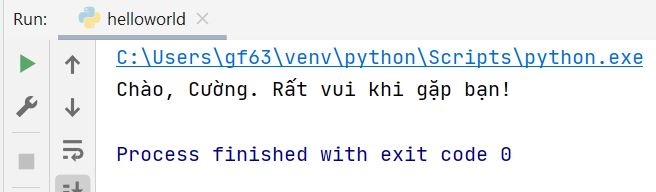
Trong Python, để gọi một function thì bạn chỉ việc nhập tên của function, sau đó truyền thêm các tham số nếu có. Bạn có thể gọi đến function ở bất kì đâu, miễn là trong phạm vi tồn tại của function.

Như ở ví dụ trên mình sẽ gọi đến function greet như sau:

|  |
| --- |
| greet('Cường') |

**Hiển thị**:

|  |
| --- |
| Chào, Cường. Rất vui khi gặp bạn! |



Khác với các ngôn ngữ khác, bạn không cần khai báo kiểu dữ liệu cho các tham số truyền vào.

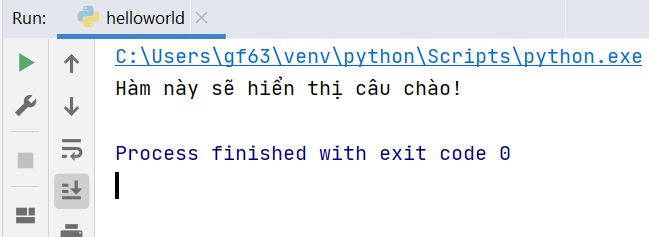
### Ý nghĩa của docstring trong function

Chuỗi đầu tiên sau tên hàm được gọi là docstring và viết tắt của document string. Nó được sử dụng để giải thích ngắn gọn ý nghĩa của function.

Mặc dù bạn có thể nhập docstring hoặc không cần cũng được. Tuy nhiên, document sẽ giúp bạn hiểu ý nghĩa của function mà không cần phải dò lại code, vì vậy với những lập trình viên có kinh nghiệm thì không bao giờ bỏ qua phần này.

Quay lại ví dụ trên, chúng ta có một docstring bên dưới tiêu đề hàm, và mình sử dụng đoạn code sau để xem ý nghĩa của hàm này.

|  |
| --- |
| print(greet.\_\_doc\_\_)  => Hàm này sẽ hiển thị câu chào! |



Như bạn thấy, kết quả nó trả về chính là phần mà mình đã comment lúc khai báo hàm.

### Hàm return trong Python

Mỗi function có thể có giá trị trả về hoặc không. Câu hỏi đặt ra là khi nào nên sử dụng hàm có giá trị trả về và khi nào thì không? Để trả lời câu hỏi này thì bạn hãy làm một vài ví dụ trước nhé.

Quay lại ví dụ ở phần 1, đó là một function không có giá trị trả về bởi không tồn tại lệnh **return** bên trong hàm. Nhưng với function dưới đây thì khác:

|  |
| --- |
| def absolute\_value(num):      """Hàm này sẽ trả về giá trị tuyệt đối của một số"""        if num >= 0:          return num      else:          return -num    # Output: 2  print(absolute\_value(2))    # Output: 4  print(absolute\_value(-4)) |

Function này có nhiệm vụ tính giá trị truyệt đối của một số, sau đó trả kết quả về, giá trị trả về chính là giá trị mà bạn đã gắn vào lệnh **return**. Vậy nếu bạn muốn viết một function mà có trả kết quả về thì sử dụng lệnh **return**.

Ví dụ bạn cần viết chương trình kiểm tra một số có phải là số nguyên tố hay không thì bạn có sử dụng lệnh return để trả về True hoặc False.

### Phạm vi của biến trong function

Trong function của Python bạn có thể tạo thêm bao nhiêu biến cũng được. Tuy nhiên, các biến đó chỉ tồn tại bên trong function mà thôi, bạn không thể sử dụng nó ở bên ngoài.

Hãy xem ví dụ dưới đây, mình đã tạo ra một function và biến x ở cả bên trong function và bên ngoài:

|  |
| --- |
| def my\_func():      x = 10      print("Giá trị bên trong function:", x)    x = 20  my\_func()  print("Giá trị bên ngoài function:", x) |

Tuy nhiên kết quả trong function sẽ trả về giá trị 10 bởi vì phạm vi của function là cục bộ, nó không nhận được biến x ở bên ngoài. Kết quả sẽ như sau:

|  |
| --- |
| Giá trị bên trong function: 10  Giá trị bên ngoài function: 20 |

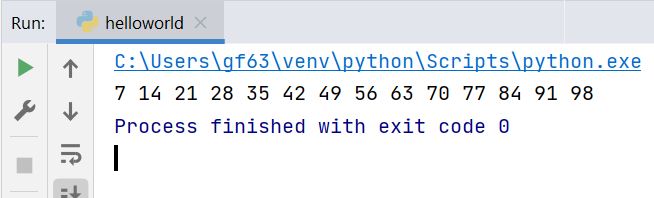
### Thực hành hàm (function) trong Python nâng cao

Bây giờ ta sẽ làm một vài ví dụ để bạn hiểu rõ hơn về cách sử dụng hàm trong Python nhé.

**Bài tập 1**: Viết chương trình in ra tất cả các số chia hết cho 7 từ 1 đến 100.

|  |
| --- |
| # HỌC PYTHON TẠI FREETUTS.NET  # TÁC GIẢ: CƯỜNG NGUYỄN  # Function trong Python  # Tìm các số chia hết cho 7 từ 1 đến 100    def chia\_het\_cho\_7(number):      if number % 7 == 0:          return True      else:          return False      for i in range(1, 101):      if chia\_het\_cho\_7(i):          print(i, end=' ') |

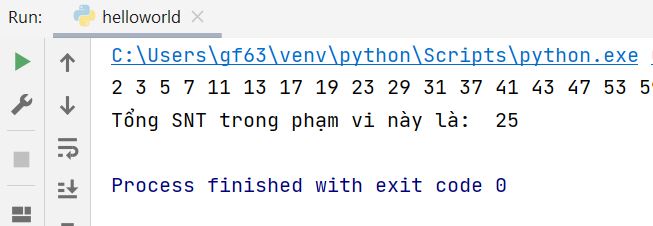
**Hiển thị**:



**Bài tập 2**: Đếm tổng và in ra các số nguyên tố từ 1 đến 100.

|  |
| --- |
| # HỌC PYTHON TẠI FREETUTS.NET  # TÁC GIẢ: CƯỜNG NGUYỄN  # Function trong Python  # Sử dụng hàm để kiểm tra SNT    def la\_SNT(n):      if (n < 2):          return False      elif (n == 2):          return True      elif (n % 2 == 0):          return False      else:          # Lặp qua các số lẻ nên bắt đầu từ 3 với bước nhảy là 2          for i in range(3, n, 2):              if (n % i == 0):                  return False      return True      counter = 0;    for i in range(1, 101):      if la\_SNT(i):          counter += 1          print(i, end=' ')  print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_")  print("Tổng SNT trong phạm vi này là: ", counter) |

**Hiển thị**:



### Hiểu rõ về tham số của Function trong Python

Trong phần này chúng ta sẽ học kỹ hơn về tham số truyền vào của hàm Python.

Trong Python bạn có thể tạo một hàm với nhiều tham số. Khi gọi hàm thì bắt buộc phải nhập giá trị cho tham số, tuy nhiên với những tham số có giá trị mặc định thì không cần.

#### Giá trị mặc định của tham số

Giá trị mặc định tức là nếu bạn không nhập giá trị lúc gọi hàm thì tham số sẽ lấy giá trị mặc định đó. Để tạo giá trị mặc định thì bạn dùng toán tử = và gán ngay lúc tạo hàm.

**VD:**

|  |
| --- |
| def greet(msg = "Good morning!"): |

Biến msg sẽ có giá trị mặc định là 'Good morning!". Hãy xem ví dụ đầy đủ sau để hiểu rõ hơn.

|  |
| --- |
| def greet(name, msg = "Chào buổi sáng!"):     print("Xin chào ",name + ', ' + msg)    greet("Kate")  greet("Bruce","Bạn có khỏe không?") |

Ở đoạn gọi hàm mình đã gọi với hai cách khác nhau,

* greet("Kate") là cách gọi chỉ truyền vào một tham số nên function sẽ lấy giá trị mặc định cho tham số thứ hai.
* greet("Bruce","Bạn có khỏe không?") là các gọi đầy đủ nên giá trị mặc định không được sử dụng.

Chạy chương trình này thì kết quả sẽ như sau:

|  |
| --- |
| Xin chào Kate, Chào buổi sáng!  Xin chào Bruce, Bạn có khỏe không? |

**Lưu ý**: Bạn không thể đặt một tham số có giá trị mặc định đằng trước những tham số không có giá trị mặc định, như ví dụ dưới đây là sai.

|  |
| --- |
| def greet(msg = "Đặt tại đây là sai!", name): |

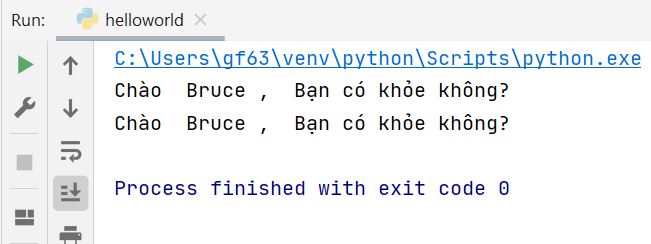
#### Truyền tham số theo key name

Truyền tham số theo **key**name tức là lúc gọi hàm và truyền tham số bạn sẽ nhập thêm tên của tham số, cách này có ưu điểm ở chỗ là bạn không cần phải nhớ chính xác thứ tự của các tham số, nhưng bù lại bạn phải nhớ tên của tham số.

Như ở ví dụ trên mình có thể gọi các cách như sau:

|  |
| --- |
| def greet(name, msg):      print("Chào ", name, ', ', msg)    greet(name = "Bruce", msg = "Bạn có khỏe không?")  greet(msg = "Bạn có khỏe không?", name = "Bruce") |

Chạy đoạn code này thì có kết quả giống nhau.



#### Tham số không giới hạn

Tham số không giới hạn hay còn gọi là tham số Arbitrary. Khi sử dụng tham số này thì bạn có thể truyền bao nhiêu cũng được lúc gọi hàm.

Để tạo một tham số không giới hạn thì bạn thêm ký tự \* đằng trước tham số đó. Hãy xem ví dụ dưới đây.

|  |
| --- |
| def greet(\*names):     """This function greets all     the person in the names tuple."""       # names is a tuple with arguments     for name in names:         print("Hello",name)    greet("Monica","Luke","Steve","John") |

Như bạn thấy, mình phải sử dụng vòng lặp để lặp qua từng tham số, và lúc gọi hàm thì nhập bao nhiêu cũng được.

**Hiển thị**:

|  |
| --- |
| Hello Monica  Hello Luke  Hello Steve  Hello John |

Đây là cách gọi hàm với nhiều đối số, khi bạn truyền vào thì nó sẽ tạo thành một Tuple nên bắt buộc phải sử dụng vòng lặp để lặp qua từng giá trị.

### Lời kết

Như vậy là bạn đã học được sơ lược về cách tạo hàm (function) trong Python. Vẫn còn rất nhiều vấn đề liên quan đến function nhưng mình sẽ viết nó ở một bài khác, việc phân tán bài học như vậy sẽ giúp bạn dễ dàng năm bắt hơn. Chúc bạn học tốt và hẹn gặp lại ở bài tiếp theo.

## HÀM ĐỆ QUY TRONG PYTHON

Trong bài này mình sẽ nói đến một kiến thức rất hay và thuộc dạng nâng cao đó là xây dựng hàm **đệ quy trong Python**, đây là phần mình đã giải thích rất rõ ở trong series học [PHP căn bản](https://freetuts.net/hoc-php/hoc-lap-trinh-php-can-ban), tuy nhiên mình cũng sẽ trình bày sơ lược lại ở trong bài học này.

Để biết được cách tạo đệ quy thì trước tiên bạn phải hiểu khái niệm về **đệ quy là gì** đã nhém sau đó ta sẽ thực hành một vài ví dụ cơ bản nhất.

### Đệ quy trong Python là gì?

Đệ quy trong Python hay còn gọi là **recursion python**. Nói về toán học thì đệ quy là thuật toán giải quyết bài toán bằng cách gọi lại chính thuật toán đó, thao tác này sẽ thực hiện liên tục cho đến khi gặp điều kiện dừng.

Đệ quy được thể hiện rất tốt khi áp dụng với hàm trong Python. Hàm đệ quy là chương trình sẽ gọi lại chính hàm đó và ngưng gọi khi gặp điều kiện dừng. Nếu quay lại kiến thức về vòng lặp thì bản chất đây cũng là một loại vòng lặp đặc biệt phải không các bạn.

Chương trình đệ quy sẽ có điều kiện dừng, nếu không nó sẽ tạo ra một vòng đời đệ quy vô hạn, điều này giống như vòng lặp while trong Python là bạn đã được học.

Hãy làm một ví dụ đơn giản đó là tính giai thừa của một số.

Ví dụ sử dụng đệ quy để tính giai thừa của 4 thì sẽ là 1\*2\*3\*4 = 24.

**Bước 1**: yêu cầu người dùng nhập vào số 4

**Bước 2**: Sử dụng đệ quy để lặp tính tích từ 4 trở về 1. Gọi x là giá trị cho mỗi lần lặp thì ta có điều kiện dừng là x = 1.

|  |
| --- |
| def calc\_factorial(x):      if x == 1:          return 1      else:          return (x \* calc\_factorial(x-1))    num = 4  print("Dãy fibo của ", num, "là ", calc\_factorial(num)) |

Bạn hãy để ý bên trong phần thân của hàm calc\_factorial nhé, điều kiện để dừng đệ quy là x == 1, ngược lại chương trình sẽ thực hiện lặp đệ quy bởi đoạn code return (x \* calc\_factorial(x-1)).

Quy trình hoạt động của nó như sau:

|  |
| --- |
| calc\_factorial(4)              # Lần 1 gọi với số 4  4 \* calc\_factorial(3)          # Lần 2 gọi với số 3  4 \* 3 \* calc\_factorial(2)      # Lần 3 gọi với số 2  4 \* 3 \* 2 \* calc\_factorial(1)  # Lần 4 gọi với số 1  4 \* 3 \* 2 \* 1                  # Cuối cùng ta được chuỗi này => kết quả là 24 |

Lần gọi đên quy cuối cùng vì giá trị tham số x truyền vào là 1 nên sẽ không thực hiện đệ quy nữa, sau đó trả kết quả là **24**.

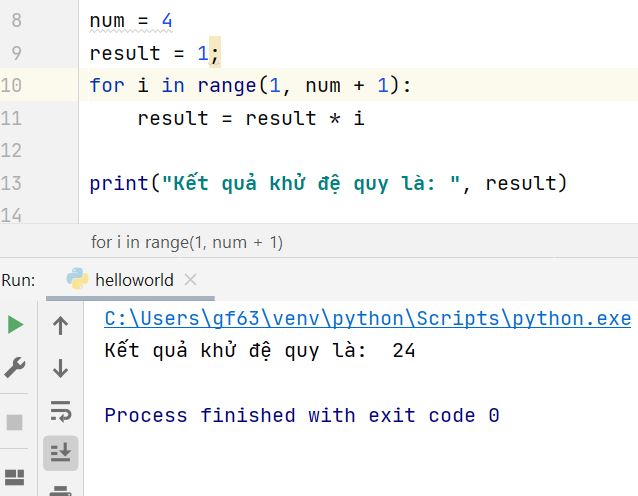
### Khử đệ quy trong Python

Thực tế thì sử dụng đệ quy sẽ tốn rất nhiều tài nguyên của máy tính, bởi nó sẽ phải lưu trữ khá nhiều thông tin để tạo ra một biểu thức cuối cùng. Vì vậy người ta thưởng sử dụng khử đệ quy để chuyển đổi từ đệ quy thành vòng lặp.

Như bài toán tính Fibo trên ta có thể sử dụng trong vòng lặp rất dễ dàng và nhanh chóng.

|  |
| --- |
| num = 4  result = 1;  for i in range(1, num + 1):      result = result \* i    print("Kết quả khử đệ quy là: ", result) |

Kết quả giống nhau:



### Ưu điểm và nhược điểm của đệ quy Python

Sau đây là một vài ưu điểm và nhược điểm của đệ quy trong lập trình Python.

**Ưu điểm**:

* Các hàm đệ quy làm cho mã trông sạch sẽ
* Một tác vụ phức tạp có thể được chia thành các vấn đề phụ đơn giản hơn bằng cách sử dụng đệ quy.
* Tạo trình tự dễ dàng với đệ quy hơn là sử dụng một số lần lặp lồng nhau.

**Nhược điểm**:

* Đôi khi logic đằng sau đệ quy rất khó theo dõi.
* Chi phí gọi đệ quy rất tốn kém (không hiệu quả) vì chúng chiếm rất nhiều bộ nhớ và thời gian.
* Các hàm đệ quy khó gỡ lỗi.

### Lời kết

Khi đi làm thực tế thì người ta rất ít khi chọn giải pháp đệ quy, trừ khi bắt buộc, bởi mất khá nhiều tài nguyên về bộ nhớ và thời gian để chạy một ứng dụng đệ quy, điều này là không tốt cho những ứng dụng trong thực tế. Bạn thử nghĩ xem nếu một website hoạt động quá chậm thì sẽ giảm đi phần trải nghiệm của người dùng một cách nghiêm trọng.

Có khá nhiều loại đệ quy như đệ quy tuyến tính, đệ quy hỗ tương, .. nhưng mình không đề cập đến trong bài viết, bạn tự tìm hiểu nhé.

## HÀM LAMBDA TRONG PYTHON

Trong bài viết này chúng ta sẽ tìm hiểu hàm **lambda trong Python**, nó còn được gọi là **hàm ẩn danh trong Python**. Đây là cách tạo hàm ngắn gọn và không có tên hàm.

Nếu bạn đã từng học các ngôn ngữ khác thì chắc chắn đã nghe qua khái niệm này rồi. Còn nếu bạn mới học Python lần đầu thì hãy tìm hiểu khái niệm lambda Python là gì đã nhé.

### Lambda Python là gì?

Lambda Python là một hàm ẩn danh được tạo ra mà không có tên hàm. Như chúng ta đã học, để tạo một hàm trong Python thì ta sử dụng từ khóa def ten\_ham, nhưng đối với hàm ẩn danh thì chúng ta sử dụng từ khóa lambda và đằng sau nó là danh sách các tham số và **một biểu thức**duy nhất.

Mặc dù các hàm bình thường được xác định bằng từ khóa def, nhưng trong các hàm ẩn danh Python được xác định bằng từ khóa lambda. Do đó, các hàm ẩn danh cũng được gọi là các hàm lambda.

Cú pháp tạo hàm lambda như sau:

|  |
| --- |
| lambda arguments: expression |

Trong đó:

* lambda là từ khóa khai báo hàm lambda
* arguments là danh sách các tham số truyền vào hàm
* expression là biểu thức tính toán của hàm

Lambda function có thể có nhiều tham số, tuy nhiên nội dung bên trong của nó chỉ có một biểu thức mà thôi. Mỗi hàm lambda đều có giá trị trả về bởi biểu thức bên trong mà không cần từ khóa return.

Hãy xem ví dụ dưới đây:

|  |
| --- |
| # Chương trình sử dụng lambda python function  double = lambda x: x \* 2    # Output: 10  print(double(5)) |

Trong ví dụ này mình đã tạo một lambda function có tên gọi là **double**, biểu thức bên trong của nó nhân hai giá trị của tham số truyên vào x \* 2.

Nếu viết theo hàm thông thường thì lambda function trên sẽ tương đương với hàm sau:

|  |
| --- |
| def double(x):     return x \* 2 |

Như bạn thấy, đối với hàm lanbda thì chúng ta không cần từ khóa return vì mặc định biểu thức của hàm sẽ là giá trị return. Còn hàm thông thường thì khác, bạn phải sử dụng từ khóa return nếu muốn trả về một giá trị cho hàm.

Vậy câu hỏi đặt ra là tại sao lại sử dụng lambda function? Chúng ta cùng tìm hiểu qua phần 2 nhé.

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

Sức mạnh nổi bậc nhất khi sử dụng lambda đó là sử dụng hàm lambda trong một function bình thường khác.

Giả sử bạn có một function có một tham số x, bạn muốn viết một function có chức năng khi truyền x lần đầu xong thì lưu trạng thái function đó lại, để những lần gọi sau sẽ kế thừa giá trị x đó.

|  |
| --- |
| def myfunc(n):    return lambda a : a \* n    # Biến mydoubler lúc này sẽ là một lambda function  mydoubler = myfunc(2)    # Vì vậy bạn có thể gọi thoải mái và nhiều lần ở nhièu vị trí  # Và vẫn kế thừa giá trị n của hàm myfunc  print(mydoubler(11)) # Kết quả 22  print(mydoubler(10)) # Kết quả 20 |

Qua ví dụ này thì bạn thấy rất dễ hiểu phải không nào? Lambda Python quả thật rất vi diệu.

### Một vài kết hợp của hàm lambda trong Python

Chúng ta sử dụng lambda function trong trường hợp cần một function trong thời gian ngắn, và thường được kết hợp với các hàm ở bậc cao hơn, chính xác hơn thì thường được sử dụng như một tham số.

Bây giờ hãy làm hai ví dụ thường thấy nhất khi sử dụng lambda function nhé, đó là kết hợp với **filter**và **map**.

#### Kết hợp với filter()

Hàm filter có công dụng là lọc dữ liệu theo tham số truyên vào, tham số đầu tiên là hàm kiểm tra điều kiện lọc, tham số thứ hai là dữ liệu cần lọc. Hãy xem ví dụ lọc các số chẵn trong list dưới đây:

|  |
| --- |
| # Chương trình lấy những số chẵn trong danh sách  my\_list = [1, 5, 4, 6, 8, 11, 3, 12]  # Hàm lambda sẽ trả về True nếu là số chẵn, False nếu là số lẻ  new\_list = list(filter(lambda x: (x%2 == 0) , my\_list))    # Output: [4, 6, 8, 12]  print(new\_list) |

**Hiển thị**

|  |
| --- |
| [4, 6, 8, 12] |

#### Kết hợp với map()

Hàm map có công dụng là lặp qua từng phần tử và thay đổi giá trị của nó dựa vào hai tham số, tham số đầu tiên là một hàm dùng để xử lý dữ liệu và trả kết quả về, tham số thứ hai là dữ liệu cần lặp.

Hãy xem ví dụ dưới đây, ví dụ này sẽ nhân đôi giá trị của các phần tử trong list.

|  |
| --- |
| # Chương trình nhân đôi giá trị các phần tử của List  my\_list = [1, 5, 4, 6, 8, 11, 3, 12]  new\_list = list(map(lambda x: x \* 2 , my\_list))    # Output: [2, 10, 8, 12, 16, 22, 6, 24]  print(new\_list) |

**Hiển thị**:

|  |
| --- |
| [2, 10, 8, 12, 16, 22, 6, 24] |

### Lời kết

Như vậy là mình đã giới thiệu sơ lược về hàm **lambda trong Python**. Ban phải hiểu rằng bản chất lambda function là một hàm ẩn danh, hoạt động trong một thời gian ngắn và tự mất đi khi phạm vi hoạt động của nó không còn nữa.

Trong thực tế chúng ta rất hay sử dụng lambda function, không chỉ ở Python mà ở hầu hết các ngôn ngữ hiện nay như PHP, Javascript đều hỗ trợ lambda function.

## BIẾN TOÀN CỤC VÀ BIẾN CỤC BỘ TRONG PYTHON

Trong bài này chúng ta sẽ tìm hiểu về **biến cục bộ** và **biến toàn cục trong Python**. Đây là hai phạm vi hoạt động chủ yếu của một biến trong Python, vì vậy bạn phải hiểu để sủ dụng cho đúng.

Đọc đến bài này thì chắc chắn bạn đã hiểu cách tạo hàm trong Python rồi nhỉ? Bạn cũng biết việc sử dụng biến trong một hàm là quá bình thường. Nhưng chuyện gì xảy ra nếu bạn khai báo tên biến trong hàm trùng với tên của một biến nằm bên ngoài hàm?

Rất đơn giản, chỉ cần biết phạm vi hoạt động của biến là bạn sẽ hiểu được cách hoạt động của Python.

### Biến toàn cục trong Python là gì?

Trong Python, một biến được khai báo **bên ngoài hàm** hoặc trong **phạm vi toàn cục** được gọi là biến toàn cục. Điều này có nghĩa, biến toàn cục có thể được truy cập bên trong hoặc bên ngoài hàm.

Biến toàn cục có tên gọi là **Global Variable**.

Hãy xem một ví dụ về cách tạo một biến toàn cục trong Python.

**Global Variable**

|  |
| --- |
| x = "Học Python tại Freetuts.net"    def foo():      print("Trong hàm:", x)  foo()  print("Ngoài hàm:", x) |

**Hiển thị**:

|  |
| --- |
| Trong hàm : Học Python tại Freetuts.net  Ngoài hàm: Học Python tại Freetuts.net |

Trong đoạn code trên thì biến x là **biến toàn cục** bởi nó được khai báo bên ngoài hàm, nằm tại vị trí cấp cao nhất của ứng dụng. Giống như Javascript, trong hàm có thể sử dụng biến cục bộ mà không cần phải thông qua việc truyền tham số.

Khi sử dụng biến toàn cục bên trong một hàm thì bạn không được thay đổi giá trị cho biến. Như ví dụ dưới đây là sai, bởi vì mình đã thay đổi giá trị cho biến x.

|  |
| --- |
| x = 10    def foo():      x = x \* 2      print(x)  foo() |

**Hiển thị**:

|  |
| --- |
| UnboundLocalError: local variable 'x' referenced before assignment |

**Lý do là đây**: Vì bên trong hàm foo() bạn đã thay đổi giá trị cho biến x nên mặc nhiên nó sẽ hiểu biến x là biệc cục bộ, mà biến cục bộ thì đoạn code x = x \* 2 sẽ sai vì biến x chưa được khai báo.

Để giải quyết vấn đề này thì bạn sử dụng từ khóa **global** nhé.

|  |
| --- |
| x = 10    def foo():      global x      x = x \* 2      print(x)    # In giá trị 20  foo()    # In giá trị 20 vì x = 20  print(x) |

Như bạn thấy, biến cục bộ x đã bị thay đổi bên trong hàm **foo**.

### Biến cục bộ trong Python là gì?

Trong Python, biến cục bộ được hiểu là biến được khai báo bên trong hàm hoặc phạm vi cục bộ, những biến này gọi là biến cục bộ.

Biến cục bộ có tên gọi là **Local Variable**.

Hãy xem ví dụ dưới đây:

|  |
| --- |
| def foo():      y = "local"    foo()    # Lệnh này sai vì biến y không tồn tại  print(y) |

Do biến y là biến cục bộ nằm bên trong hàm foo() nên đoạn code print(y) là sai, bạn không thể sử dụng biến cục bộ ở bên ngoài hàm được.

**Hiển thị**:

|  |
| --- |
| NameError: name 'y' is not defined |

### Từ khóa global trong Python

Mình đã lấy một ví dụ về từ khóa Global ở phần 1 rồi, tuy nhiên mình sẽ giải thích nó kỹ hơn một chút qua phần này nhé.

Để tường minh trong việc sử dụng biến cục bộ trong hàm thì Python cung cấp từ khóa **global**, khi khai báo từ khóa này cho một biến bất kì thì trình biên dịch sẽ tự hiểu biến đó sẽ trỏ tới một biến cục bộ nằm bên ngoài hàm.

**Ví dụ**: Sử dụng từ khóa global

|  |
| --- |
| x = "global"    def foo():      global x      y = "local"      x = x \* 2      print(x)      print(y)    foo() |

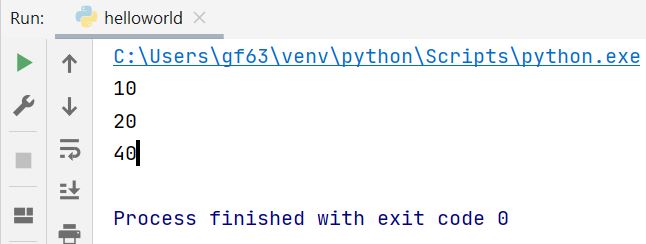
**Hiển thị**:

|  |
| --- |
| global global  local |

Khi chúng ta thay đổi giá trị cho biến được khai báo global nằm ở trong hàm thì giá trị của biến đó sẽ thay đổi theo. Xem ví dụ dưới đây để hiểu hơn.

|  |
| --- |
| x = 5    def plusNum():      global x      x = x \* 2    plusNum()  print(x) # kết quả là 10    plusNum()  print(x) # kết quả là 20    plusNum()  print(x) # kết quả là 40 |

Xem hình:



### Lời kết

Khó khăn nhất là việc khai báo trùng tên ở biến toàn cục và biến cục bộ, vì vậy lời khuyên bạn nên sử dụng biến một cách khoa học, khai báo bằng tham số truyền vào nếu muốn sử dụng biến toàn cục ở bên trong hàm.

Như vậy là mình đã giới thiệu xong khái niệm về biến cục bộ và biến toàn cục trong Python, hy vọng bạn sẽ hiểu ý nghĩa của từng loại biến để sử dụng cho đúng mục đích.

## BÀI TẬP PYTHON: THỰC HÀNH TẠO HÀM TRONG PYTHON

Trong bài này mình sẽ tổng hợp một số bài tập thực hành về hàm trong Python. Qua những bài tập tạo hàm này sẽ giúp bạn vững hơn trong việc phân chia module khi lập trình Python.

Nếu bạn đã từng học qua ngôn ngữ khác rồi thì có thể bỏ qua bài này. Nhưng nếu bạn là người mới tìm hiểu Python thì nên làm những bài tập này, nó sẽ giúp ích cho bạn rất nhiều trong việc rèn luyện tư duy logic.

Sau này khi làm việc với lập trình hướng đối tượng (OOP) thì bạn phải hiểu về hàm trước, bởi vì trong OOP có khái niệm về phương thức (mothod) có cách sử dụng rất giống với hàm.

**Câu hỏi thường gặp liên quan:**

* [Viết hàm Python tính điểm trung bình học sinh](https://freetuts.net/ref/viet-ham-python-tinh-diem-trung-binh-hoc-sinh-412.html)
* [Viết hàm Python tính tổng các số nguyên tô từ 0 - 1000](https://freetuts.net/ref/viet-ham-python-tinh-tong-cac-so-nguyen-to-413.html)
* [Viết hàm Python tìm giá trị lớn nhất trong 3 số](https://freetuts.net/ref/chua-khoi-tao-slug-414.html)
* [Viết hàm Lambda Python in ra thông tin sinh viên](https://freetuts.net/ref/viet-ham-lambda-python-in-ra-thong-tin-sinh-vien-415.html)
* [Viết hàm đệ quy Python tính tổng S = 1 + 2 + 3 + 4 + 5 ... + n](https://freetuts.net/ref/viet-ham-de-quy-python-tinh-tong-s-1-+-2-+-3-+-4-+-5-+-n-416.html)

# KIỂU DỮ LIỆU TRONG PYTHON

## KIỂU DỮ LIỆU NUMBER TRONG PYTHON

Trong bài này mình sẽ giới thiệu kiểu dữ liệu Number trong Python, tìm hiểu về các loại number khác nhau được sử dụng trong Python, cách chuyển đổi từ loại dữ liệu này sang loại dữ liệu khác và các hoạt động toán học được hỗ trợ trong Python.

Nói đến Number thì chắc ai cũng biết đây là kiểu số. Tuy nhiên, chúng ta có rất nhiều loại số khác nhau như số nguyên (int), số thực (float), số phức (complex number). Vì vậy bạn phải nắm vững cách chuyển đổi giữa chúng.

**Mục lục**

* [1. Number trong Python là gì?](https://freetuts.net/kieu-du-lieu-number-trong-python-1709.html#goto-h2-0)
* [2. Số thập phân trong Python](https://freetuts.net/kieu-du-lieu-number-trong-python-1709.html#goto-h2-1)
  + [Chúng ta đang sai ở đâu?](https://freetuts.net/kieu-du-lieu-number-trong-python-1709.html#goto-h3-0)
* [3. Khi nào sử dụng Decimal thay vì float](https://freetuts.net/kieu-du-lieu-number-trong-python-1709.html#goto-h2-2)
* [4. Phân số](https://freetuts.net/kieu-du-lieu-number-trong-python-1709.html#goto-h2-3)
* [5. Các hàm toán học](https://freetuts.net/kieu-du-lieu-number-trong-python-1709.html#goto-h2-4)
* [6. Lời kết](https://freetuts.net/kieu-du-lieu-number-trong-python-1709.html#goto-h2-5)

### Number trong Python là gì?

Number là một nhóm dữ liệu thường dùng trong Python, nó dùng để lưu trữ hầu hết các kiểu về số như số thực, số phức, số nguyên.

Python hỗ trợ số nguyên, số thực và số phức. Chúng được định nghĩa trong các lớp int, float và lớp complex của Python.

Integer và float được phân biệt bằng dấu chấm động (.). Nếu một số có dấu chấm động là kiểu float, ngược lại là kiểu int.

**Ví dụ**: Số 5 thuộc kiểu int, trong khi 5.0 thuộc kiểu float.

Số phức được viết dưới dạng x + yj, trong đó **x** là phần thực và y là phần ảo.

Chúng ta có thể sử dụng hàm type() để biết một biến đang thuộc lớp nào, và hàm isinstance() để kiểm tra xem nó có thuộc về một lớp cụ thể nào đó không.

**Ví dụ**

|  |
| --- |
| # Output: <class 'int'>  print(type(a))    # Output: <class 'float'>  print(type(5.0))    # Output: (8+3j)  c = 5 + 3j  print(c + 3)    # Output: True  print(isinstance(c, complex)) |

Trong khi số nguyên có thể có độ dài bất kỳ, số dấu phẩy động chỉ chính xác tối đa 15 vị trí thập phân (vị trí thứ 16 không chính xác).

Các số chúng ta xử dụng hàng ngày là hệ thống số thập phân (cơ sở 10). Nhưng các lập trình viên máy tính (nói chung là lập trình viên nhúng) cần phải làm việc với các hệ thống số nhị phân (cơ sở 2), hệ thập lục phân (cơ sở 16) và số bát phân (cơ sở 8).

Trong Python, chúng ta có thể biểu diễn các số này bằng cách đặt một tiền tố thích hợp trước số đó.

Bảng sau liệt kê các tiền tố này.

|  |  |
| --- | --- |
| **Number System** | **Prefix** |
| Binary | '0b' or '0B' |
| Octal | '0o' or '0O' |
| Hexadecimal | '0x' or '0X' |

Dưới đây là một vài ví dụ cách sử dụng các tiền tố prefix:

|  |
| --- |
| # Output: 107  print(0b1101011)    # Output: 253 (251 + 2)  print(0xFB + 0b10)    # Output: 13  print(0o15) |

**Hiển thị**:

|  |
| --- |
| 107  253  13 |

Chúng ta có thể chuyển đổi một từ cơ số này sang cơ số khác. Điều này còn được gọi là chuyển đổi cơ số.

**Ví dụ**: Các phép toán như phép cộng, phép trừ số nguyên sẽ tự động chuyển đổi cơ số sang kiểu float nếu một trong các toán hạng là float.

|  |
| --- |
| >>> 1 + 2.0  3.0 |

Như ví dụ trên bạn thấy 1 là kiểu số nguyên, 2.0 là kiểu số động và khi thực hiện phép toán thì kết quả sẽ mang kiểu số động.

Bạn cũng có thể sử dụng các hàm dựng sẵn như int (), float () và comlex () để chuyển đổi các kiểu dữ liệu một cách rõ ràng. Các hàm này thậm chí có thể chuyển đổi từ các chuỗi sang số và số sang chuỗi.

|  |
| --- |
| >>> int(2.3)  2  >>> int(-2.8)  -2  >>> float(5)  5.0  >>> complex('3+5j')  (3+5j) |

### Số thập phân trong Python

Lớp built-in trong Python có thể thực hiện một vài phép toán khiến chúng ta ngạc nhiên. Chúng ta đều biết rằng tổng của 1.1 và 2.2 là 3.3, tuy nhiên khi sử dụng toán tử so sánh bằng (==) thì điều đó là không đúng trong Python.

|  |
| --- |
| >>> (1.1 + 2.2) == 3.3  False |

#### Chúng ta đang sai ở đâu?

Nó chỉ ra rằng các số dấu phẩy động được triển khai trong phần cứng máy tính dưới dạng phân số nhị phân, vì máy tính chỉ hiểu nhị phân (0 và 1). Vì lý do này, hầu hết các phân số thập phân mà chúng ta biết không thể được lưu trữ chính xác trong máy tính.

**Hãy lấy một ví dụ.** Chúng ta không thể biểu thị phân số 1/3 dưới dạng số thập phân. Điều này sẽ cung cấp cho 0.33333333 ... dài vô hạn và chúng ta chỉ có thể ước chừng nó.

Hóa ra phân số thập phân 0,1 sẽ dẫn đến phân số nhị phân dài vô hạn 0,000110011001100110011 ... và máy tính của chúng ta chỉ lưu trữ một số hữu hạn của nó.

Điều này sẽ chỉ xấp xỉ 0,1 nhưng không bao giờ bằng nhau. Do đó, đó là giới hạn của phần cứng máy tính của chúng tôi và không phải là lỗi trong Python.

|  |
| --- |
| >>> 1.1 + 2.2  3.3000000000000003 |

Để khắc phục vấn đề này, chúng ta có thể sử dụng mô-đun **decimal**.

|  |
| --- |
| import decimal    # Output: 0.1  print(0.1)    # Output: Decimal('0.1000000000000000055511151231257827021181583404541015625')  print(decimal.Decimal(0.1)) |

Mô-đun này được sử dụng khi chúng ta muốn thực hiện các phép tính thập phân như đã học ở trường. Chúng ta biết 25,50 kg chính xác hơn 25,5 kg vì nó có hai chữ số thập phân đáng kể so với một.

|  |
| --- |
| from decimal import Decimal as D  # Output: Decimal('3.3')  print(D('1.1') + D('2.2'))    # Output: Decimal('3.000')  print(D('1.2') \* D('2.50')) |

Chúng ta có thể hỏi tại sao không thực hiện Decimal mỗi lần, thay vì float? Lý do chính là hiệu quả, float luôn luôn nhanh hơn.

### Khi nào sử dụng Decimal thay vì float

Chúng ta thường sử dụng Decimal trong các trường hợp sau.

* Khi đang làm cho các ứng dụng tài chính cần đại diện thập phân chính xác.
* Khi muốn kiểm soát mức độ chính xác cần thiết.
* Khi muốn thực hiện khái niệm về số thập phân có ý nghĩa.
* Khi muốn các hoạt động được thực hiện như chúng ta đã làm ở trường

### Phân số

Python cung cấp các hoạt động liên quan đến số phân số thông qua mô-đun **Fraction** của nó. Chúng ta có thể tạo các đối tượng Fraction theo nhiều cách khác nhau.

|  |
| --- |
| import fractions    # Output: 3/2  print(fractions.Fraction(1.5))    # Output: 5  print(fractions.Fraction(5))    # Output: 1/3  print(fractions.Fraction(1,3)) |

Trong khi tạo phân số từ kiểu float chúng ta có thể nhận được một số kết quả bất thường. Điều này là do biểu diễn số dấu phẩy động nhị phân không hoàn hảo như đã thảo luận trong phần trước.

May mắn thay, phân số cho phép chúng ta khởi tạo bằng chuỗi, đây là các tùy chọn ưa thích khi sử dụng số thập phân.

|  |
| --- |
| import fractions    # As float  # Output: 2476979795053773/2251799813685248  print(fractions.Fraction(1.1))    # As string  # Output: 11/10  print(fractions.Fraction('1.1')) |

Kiểu dữ liệu này hỗ trợ tất cả các toán tử cơ bản. Dưới đây là một vài ví dụ.

|  |
| --- |
| from fractions import Fraction as F    # Output: 2/3  print(F(1,3) + F(1,3))    # Output: 6/5  print(1 / F(5,6))    # Output: False  print(F(-3,10) > 0)    # Output: True  print(F(-3,10) < 0) |

### Các hàm toán học

Python cung cấp các mô-đun như **math**và **random** để thực hiện các phép toán khác nhau như lượng giác, logarit, xác suất và thống kê, v.v.

**Module math**

|  |
| --- |
| import math    # Output: 3.141592653589793  print(math.pi)    # Output: -1.0  print(math.cos(math.pi))    # Output: 22026.465794806718  print(math.exp(10))    # Output: 3.0  print(math.log10(1000))    # Output: 1.1752011936438014  print(math.sinh(1))    # Output: 720  print(math.factorial(6)) |

**Module random**

|  |
| --- |
| import random    # Output: 16  print(random.randrange(10,20))    x = ['a', 'b', 'c', 'd', 'e']    # Get random choice  print(random.choice(x))    # Shuffle x  random.shuffle(x)    # Print the shuffled x  print(x)    # Print random element  print(random.random()) |

### Lời kết

Như vậy là mình đã giới thiệu xong toàn bộ kiến thức cơ bản về xử lý number trong Python, có rất nhiều hàm bổ trợ cho number và mình sẽ trình bày nó ở một bài khác. Chúc bạn học tốt.

## KIỂU DỮ LIỆU STRING TRONG PYTHON

Trong bài này chúng ta sẽ tìm hiểu kiểu dữ liệu String trong Python, đây là kiểu chuỗi, được dùng để lưu trữ các chuỗi kí tự trong Python.

Mình sẽ hướng dẫn các bạn cách tạo, định dạng, sửa đổi và xóa các chuỗi trong Python. Ngoài ra, bạn sẽ được giới thiệu các thao tác và chức năng liên quan đến chuỗi.

**Mục lục**

* [1. Kiểu dữ liệu string trong Python là gì?](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h2-0)
  + [Cú pháp tạo string trong Python](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h3-0)
  + [Truy cập các ký tự của chuỗi string trong Python](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h3-1)
  + [Thay đổi và xóa chuỗi string trong Python](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h3-2)
* [2. Các phép toán với chuỗi string trong Python](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h2-1)
  + [Nối hai hoặc nhiều chuỗi](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h3-3)
  + [Lặp qua từng ký tự trong chuỗi](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h3-4)
  + [Kiểm tra chuỗi con trong chuỗi cha](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h3-5)
* [3. Các định dạng chuỗi](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h2-2)
  + [Ký tự nháy đơn và nháy kép](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h3-6)
  + [Ký tự thoát](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h3-7)
* [4. Lời kết](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html#goto-h2-3)

### Kiểu dữ liệu string trong Python là gì?

Một chuỗi là một dãy các ký tự được sắp xếp với nhau, một ký tự đơn giản là một biểu tượng và máy tính có thể đọc hiểu được ký tự đó thông qua mã máy ASCII.

Máy tính không hiểu ý nghĩa của các ký tự như con người mà chúng sẽ hiểu sẽ chuyển sang kiểu nhị phân (binary). Mặc dù bạn có thể thấy các ký tự trên màn hình của mình, nhưng bên trong nó được lưu trữ và thao tác dưới dạng kết hợp của 0 và 1.

Việc chuyển đổi từ ký tự thành số được gọi là mã hóa và quá trình ngược lại gọi là giải mã. ASCII và Unicode là một số dạng mã hóa được sử dụng phổ biến.

#### Cú pháp tạo string trong Python

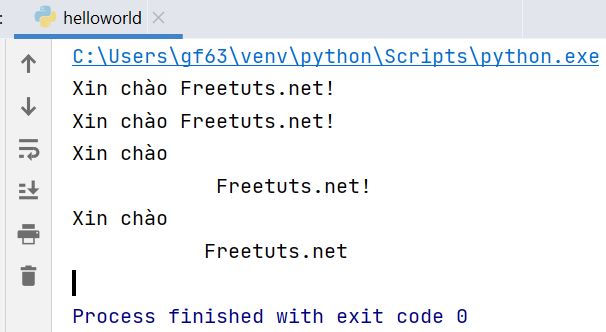
Chuỗi có thể được tạo bằng cách đặt các ký tự bên trong cặp nháy đơn hoặc nháy kép. Nếu bạn muốn viết chuỗi trên nhiều dòng thì phải sử dụng ba dấu nháy, cách này thường được dùng để tạo **docstring**.

**Ví dụ tạo chuỗi**

|  |
| --- |
| # Tất cả những ví dụ dưới đây là tương đương với nhau.    # Dùng nháy đơn  my\_string = 'Xin chào Freetuts.net!'  print(my\_string)    # Dùng nháy kép  my\_string = "Xin chào Freetuts.net!"  print(my\_string)    # Dùng ba nháy đơn trên nhiều dòng  my\_string = '''Xin chào             Freetuts.net!'''  print(my\_string)    # Dùng ba nháy kép trên nhiều dòng  my\_string = """Xin chào             Freetuts.net"""  print(my\_string) |

**Hiển thị:**

|  |
| --- |
| Xin chào Freetuts.net!  Xin chào Freetuts.net!  Xin chào              Freetuts.net!  Xin chào             Freetuts.net |



Ví dụ trên là một vài cách tạo chuỗi thông thường với nháy đơn và nháy kép,

#### Truy cập các ký tự của chuỗi string trong Python

String được xem như là một **list** các ký tự nên bạn có thể truy cập đến các ký tự thông qua chỉ mục index của nó.

Chỉ mục bắt đầu từ 0, và nếu bạn cố gắng truy cập một ký tự nằm ngoài phạm vi chỉ mục thì sẽ xuất hiện lỗi **IndexError**.

Chỉ mục phải là một số nguyên, không thể sử dụng float hoặc các loại khác vì điều này sẽ dẫn đến **TypeError**.

Chỉ số -1 đề cập đến phần tử cuối cùng, -2 cho phần tử cuối cùng thứ hai, .. (đây là cách tính ngược của chuỗi).

Chúng ta có thể truy cập một loạt các phần tử trong một chuỗi bằng cách sử dụng toán tử slicing (dấu hai chấm).

**Ví dụ**

|  |
| --- |
| str = 'freetuts.net'  print('str = ', str)    # Ký tự đầu tiên  print('str[0] = ', str[0])    # Ký tự cuối cùng  print('str[-1] = ', str[-1])    #Lấy các kí tự thứ 2 đến thứ 6  print('str[1:5] = ', str[1:5])    # Lấy các kí tự từ thứ 6 đến ky tự thứ 3 tính từ cuối chuỗi  print('str[5:-2] = ', str[5:-2]) |

**Hiển thị**:

|  |
| --- |
| str =  freetuts.net  str[0] =  f  str[-1] =  t  str[1:5] =  reet  str[5:-2] =  uts.n |

Nhưng nếu bạn cố gắng truy cập đến một phần tử không tồn tại, thì sẽ báo lỗi như ví dụ dưới đây.

|  |
| --- |
| # index must be in range  >>> my\_string[15]  ...  IndexError: string index out of range    # index must be an integer  >>> my\_string[1.5]  ...  TypeError: string indices must be integers |

### Thay đổi và xóa chuỗi string trong Python

Để thay đổi chuỗi thì bạn có thể sử dụng chỉ mục index ... điều này là **sai**nhé các bạn. Chuỗi là bất biến nên bạn không thể thay đổi chuỗi, chỉ được phép gán một chuỗi khác vào biến.

Ví dụ mình thay đổi ký tự thứ 2 trong chuỗi.

**Lỗi đổi giá trị của chuỗi**

|  |
| --- |
| # Thử thay đổi ký tự thứ 2 thành ký tự khác  str = 'freetuts.net'  str[1] = "A" |

Thì nhận được kết quả lỗi như sau:



Chúng ta không thể xóa một ký tự trong chuỗi nhưng có thể xóa hoàn toàn chuỗi bằng cách sử dụng từ khóa del, đây là từ khóa được sử dụng rất nhiều từ các bài học trước tới giờ.

**Lỗi xóa một ký tự trong chuỗi**

|  |
| --- |
| str = 'freetuts.net'    # Lệnh này sai, sẽ bị lỗi  del(str[1])    # Lệnh này đúng, nó xóa luôn biến str  del(str)    # Vì vậy khi in giá trị của str thì không tồn tại biến này  print(str) |

### Các phép toán với chuỗi string trong Python

Có nhiều phép toán được thực hiện với chuỗi string làm cho nó trở thành một trong những kiểu dữ liệu được sử dụng nhiều nhất trong Python.

#### Nối hai hoặc nhiều chuỗi

Phép toán nối hai hoặc nhiều chuỗi thành một chuỗi thì ta gọi là phép nối chuỗi.

* Để nối chuỗi thì ta sử dụng toán tử +.
* Nếu muốn lặp lại chuỗi nhiều lần thì dùng toán tử \*.

Hãy xem ví dụ dưới đây:

|  |
| --- |
| str1 = 'Hello'  str2 ='World!'    # Sử dụng +  print('str1 + str2 = ', str1 + str2)    # Sử dụng \*  print('str1 \* 3 =', str1 \* 3) |

**Hiển thị:**

|  |
| --- |
| str1 + str2 =  HelloWorld!  str1 \* 3 = HelloHelloHello |

Nếu viết hai chuỗi gần kề nhau thì cũng được xem là phép nối chuỗi. Còn nếu muốn nối hai chuỗi có nhiều dòng thì phải bổ sung thêm cặp ngoặc đơn.

**Ví dụ**

|  |
| --- |
| # Đặt hai chuỗi gần nhau  print('Xin chào'' Freetuts')    # Hai chuỗi gần nhau nhưng có xuống hàng  # Nên thêm cặp ngoặc đơn ()  print(('Xin chào'         ' Freetuts')) |

**Hiển thị**

|  |
| --- |
| Xin chào Freetuts  Xin chào Freetuts |

#### Lặp qua từng ký tự trong chuỗi

Nếu bạn muốn lặp qua từng ký tự trong chuỗi thì có thể kết hợp với [vòng lặp for](https://freetuts.net/vong-lap-for-trong-python-1689.html).

**Ví dụ**

|  |
| --- |
| count = 0  for letter in 'Freetuts.net':      if(letter == 't'):          count += 1  print(count,'ký tự t được tìm thấy') |

**Hiển thị**

|  |
| --- |
| 3 ký tự t được tìm thấy |

#### Kiểm tra chuỗi con trong chuỗi cha

Tương tự, để kiểm tra một chuỗi con có xuất hiện trong chuỗi cha hay không thì ta dùng toán tử in và not in.

**Ví dụ**

|  |
| --- |
| # Trả về True  print('f' in 'freetuts.net')    # Trả về False  print('f' not in 'freetuts.net') |

### Các định dạng chuỗi

#### Ký tự nháy đơn và nháy kép

Nếu trong chuỗi có ký tự nháy đơn hoặc nháy kép thì bạn phải bổ thêm đằng trước nó dấu \, nếu không Python sẽ không hiểu đó là một dãy chuỗi. Việc sử dụng nháy đơn hay kép phụ thuộc vào chuỗi được bao quanh bởi nháy đơn hay kép:

* Nếu bao quanh bởi nháy đơn thì thêm \ vào nháy đơn
* Nếu bao quanh bởi nháy kép thì thêm \ vào nháy kép

**VD:**

|  |
| --- |
| # Sử dụng ba dấu nháy  print('''He said, "What's there?"''')    # Trường hợp nháy đơn  print('He said, "What's there?"')    # Trường hợp nháy kép  print("He said, \"What's there?\"") |

#### Ký tự thoát

Chúng ta có rất nhiều ký tự thoát trong Python, nó là một dạng chuỗi đặc biệt. Ví dụ bạn thêm \n cuối chuỗi thì python sẽ hiểu đó là ký tự xuống hàng.

Dưới đây là danh sách các ký tự thoát.

|  |  |
| --- | --- |
| Escape Sequence | Description |
| \newline | Backslash and newline ignored |
| \\ | Backslash |
| ' | Single quote |
| \" | Double quote |
| \a | ASCII Bell |
| \b | ASCII Backspace |
| \f | ASCII Formfeed |
| \n | ASCII Linefeed |
| \r | ASCII Carriage Return |
| \t | ASCII Horizontal Tab |
| \v | ASCII Vertical Tab |
| \ooo | Character with octal value ooo |
| \xHH | Character with hexadecimal value HH |

### Lời kết

Như vậy là mình đã giới thiệu sơ lược các kiến thức về xử lý chuỗi string trong Python, bài này giải thích rõ phần lý thuyết giúp bạn hiểu một cách tường tận để sau này không phải bở ngỡ trước những bài toán phức tạp. Chúc ban học tốt.

## KIỂU LIST TRONG PYTHON

Trong bài này chúng ta sẽ tìm hiểu kiểu dữ liệu List (mảng) trong Python. Đây là kiểu dữ liệu được dùng rất nhiều, nhất là trong những bài tập học kỹ thuật lập trình Python.

Kiểu dữ liệu List hay còn được gọi là array (tức là mảng). Mảng là một kiểu dữ liệu rất quan trọng bất kì ngôn ngữ nào như PHP, C/C++ nên việc nắm vững nó là rất cần thiết.

**Mục lục**

* [1. Mảng trong Python là gì?](https://freetuts.net/list-mang-trong-python-1690.html#goto-h2-0)
* [2. Các thao tác trên mảng trong Python](https://freetuts.net/list-mang-trong-python-1690.html#goto-h2-1)
  + [Thay đổi giá trị của phần tử](https://freetuts.net/list-mang-trong-python-1690.html#goto-h3-0)
  + [Đếm chiều dài của mảng](https://freetuts.net/list-mang-trong-python-1690.html#goto-h3-1)
  + [Lặp qua từng phần tử](https://freetuts.net/list-mang-trong-python-1690.html#goto-h3-2)
  + [Kiểm tra một giá trị có tồn tại trong mảng](https://freetuts.net/list-mang-trong-python-1690.html#goto-h3-3)
  + [Thêm phần tử vào mảng](https://freetuts.net/list-mang-trong-python-1690.html#goto-h3-4)
  + [Xóa phần tử ra khỏi mảng](https://freetuts.net/list-mang-trong-python-1690.html#goto-h3-5)
  + [Thiết lập List rỗng](https://freetuts.net/list-mang-trong-python-1690.html#goto-h3-6)
* [3. Lời kết](https://freetuts.net/list-mang-trong-python-1690.html#goto-h2-2)

### Mảng trong Python là gì?

Trong Python, mảng là một loại dữ liệu đặc biệt, nó gồm nhiều phần tử và mỗi phần tử là một dữ liệu riêng biệt.

Ví dụ bạn cần lưu trữ danh sách sinh viên thì có thể sử dụng mảng, mỗi phần tử của mảng là một sinh viên.

Bạn cũng có thể tưởng tượng một chiếc tủ có nhiều ngăn xếp, lúc này chiếc tủ được xem là mảng và các ngăn xếp là các phần tử của mảng.

**\* Lưu ý**: Trong Python, ngoài tên gọi là mảng ra thì tên chính của nó là **List**nhé các bạn.

**Cách tạo mảng**

Trước tiên hãy tìm hiểu cú pháp khởi tạo một list, xem các ví dụ dưới đây.

|  |
| --- |
| list1 = ['freetuts', 'blog', 1997, 2000];  list2 = [1, 2, 3, 4, 5 ];  list3 = ["a", "b", "c", "d"] |

Trong ví dụ này mình đã tạo ra ba biến lưu trữ 3 tập hợp List khác nhau, mỗi phần tư trong List không bắt buộc phải có cùng kiểu dữ liệu. Ví dụ dưới đây sẽ in ra giá trị của các phần tử trong List.

|  |
| --- |
| thislist = ["apple", "banana", "cherry"]  print(thislist) |

**Hiển thị**

|  |
| --- |
| ['apple', 'banana', 'cherry'] |

Ngoài ra, bạn có thể tạo list thông qua đối tượng list của Python.

|  |
| --- |
| thislist = list(("apple", "banana", "cherry"))  print(thislist) |

**Hiển thị**

|  |
| --- |
| ['apple', 'banana', 'cherry'] |

### Các thao tác trên mảng trong Python

Bây giờ ta sẽ tìm hiểu những hành động liên quan đến List nhé.

#### Thay đổi giá trị của phần tử

Như ta biết, list bản chất là array nên mỗi phần tử sẽ có một chỉ số đánh dấu riêng, vị trí đầu tiên là 0, tiếp theo là 1, 2, 3, 4, ... Ví dụ bạn có một List gồm 5 phần tử thì lần lược chỉ số đánh dấu của các phần tử là: 0, 1, 2, 3, 4. Như vậy để thay đổi giá trị của phần tử nào thì ta sẽ dựa vào số chỉ mục đánh dấu này.

**Ví dụ**

|  |
| --- |
| thislist = ["apple", "banana", "cherry"]  thislist[1] = "blackcurrant"  print(thislist) |

**Hiển thị**

|  |
| --- |
| ['apple', 'blackcurrant', 'cherry'] |

#### Đếm chiều dài của mảng

Để đếm chiều dài của mảng thì ta sử dụng hàm len.

|  |
| --- |
| thislist = ["apple", "banana", "cherry"]  print(len(thislist)) |

**Hiển thị**

|  |
| --- |
| 3 |

#### Lặp qua từng phần tử

Để lặp qua từng phần tử thì ta sư dụng [vòng lặp for](https://freetuts.net/vong-lap-for-trong-python-1689.html) hoặc [vòng lặp while](https://freetuts.net/vong-lap-while-trong-python-685.html), kết hợp với hàm len để đếm tổng số phần tử của List.

**Ví dụ**

|  |
| --- |
| fruits = ['banana', 'apple',  'mango']  for index in range(len(fruits)):     print 'Current fruit :', fruits[index] |

Có một cách đơn giản hơn là bạn sử dụng cú pháp basic như sau:

|  |
| --- |
| fruits = ['banana', 'apple',  'mango']  for fruit in fruits:     print 'Fruit :', fruit |

#### Kiểm tra một giá trị có tồn tại trong mảng

Ta phải sử dụng lênh if để kiểm tra. Như ví dụ dưới đây kiểm tra chuỗi "Apple" có tồn tại trong list không.

|  |
| --- |
| thislist = ["apple", "banana", "cherry"]  if "apple" in thislist:    print("Apple có trong danh sách này") |

### Thêm phần tử vào mảng

Nếu PHP sử dụng cú pháp $var[] = value để thêm một phần tử vào array thì trong Python phải sử dụng phương thức **append** của List.

|  |
| --- |
| thislist = ["apple", "banana", "cherry"]  thislist.append("orange")  print(thislist) |

**Hiển thị**

|  |
| --- |
| ['apple', 'banana', 'cherry', 'orange'] |

#### Xóa phần tử ra khỏi mảng

**Cách 1**: Để xóa một phần tử ra khỏi List thì ta sử dụng phương thức remove() được tích hợp sẵn trong List.

|  |
| --- |
| thislist = ["apple", "banana", "cherry"]  thislist.remove("banana")  print(thislist) |

Chương trình này sẽ xóa phần tử có giá trji là "banana".

**Cách 2**: Nếu bạn muốn xóa phần tử có số thứ tự index nào đó thì sử dụng phương thức pop(). Nếu bạn không truyền index vào thì mặc định nó sẽ xóa phần tử cuối cùng.

|  |
| --- |
| thislist = ["apple", "banana", "cherry"]  thislist.pop(1)  print(thislist) |

Chương trình này sẽ xóa phần tử có index = 1, tức là "banana".

**Cách 3**: Sử dụng từ khóa del

|  |
| --- |
| thislist = ["apple", "banana", "cherry"]  del thislist[<em>0</em>]  print(thislist) |

Từ khóa del còn có thể xóa toàn bộ phần tử như sau.

|  |
| --- |
| thislist = ["apple", "banana", "cherry"]  del thislist |

#### Thiết lập List rỗng

Để thiết lập list rỗng thì bạn sử dụng phương thức clear().

|  |
| --- |
| thislist = ["apple", "banana", "cherry"]  thislist.clear()  print(thislist) |

### Lời kết

Trên là cách sử dụng mảng trong Python. Nếu bạn muốn tham khảo thêm các function thì hãy xem ở chuyên mục [List Methods](https://freetuts.net/python-function/list-methods) nhé, ở đó có rất nhiều ví dụ về cách sử dụng các phương thức có sẵn trong List như: [clear](https://freetuts.net/list-clear-trong-python-5298.html), [copy](https://freetuts.net/list-copy-trong-python-5297.html), [sort](https://freetuts.net/list-sort-trong-python-5296.html), ...

Như vậy là mình đã giới thiệu xong cách sử dụng List trong Python, và mình cũng có nói sơ lược về các phương thức có sẵn trong List dùng để xử lý những hành động thường gặp. Hẹn gặp bạn ở bài tiếp theo nhé.

## KIỂU DỮ LIỆU TUPLE TRONG PYTHON

Trong bài viết này chúng ta sẽ học cách sử dụng **kiểu dữ liệu tuple trong Python**. Cụ thể hơn sẽ tìm hiểu khái niệm tuple là gì? Cách sử dụng nó như thế nào? Các phương thức được tích hợp sẵn trong tuple.

Ở bài trước chúng ta đã học kiểu dữ liệu mảng và bạn cũng đã biết cách sử dụng rồi phải không nào?

Thực ra tuple cũng không có gì đặc biệt, cách khai báo và sử dụng nó không khác gì mảng. Nhưng tại sao Python lại bổ sung thêm tuple trong khi đã có mảng? Hãy cùng mình tìm hiểu ngay nhé.

**Mục lục**

* [1. Kiểu dữ liệu tuple trong Python là gì?](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h2-0)
  + [Ưu điểm của tuple so với mảng](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h3-0)
* [2. Cách tạo Tuple trong Python](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h2-1)
* [3. Truy cập các phần tử của tuple trong Python](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h2-2)
  + [Truy cập thông qua chỉ mục index](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h3-1)
  + [Số chỉ mục âm](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h3-2)
  + [Slicing](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h3-3)
* [4. Thay đổi giá trị cho các phần tử Tuple trong Python](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h2-3)
* [5. Xóa Tuple](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h2-4)
* [6. Các phương thức trong Tuple](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h2-5)
* [7. Một vài thao tác khác với Tuple](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h2-6)
  + [Kiểm tra phần tử có xuất hiện trong Tuple](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h3-4)
  + [Duyệt qua từng phần tử của Tuple](https://freetuts.net/kieu-du-lieu-tuple-trong-python-1710.html#goto-h3-5)

### Kiểu dữ liệu tuple trong Python là gì?

Trong Python, kiểu dữ liệu Tuple tương tự như [mảng](https://freetuts.net/list-mang-trong-python-1690.html), sự khác biệt giữa chúng là ta **không thể thay đổi** các phần tử của một tuple, trừ khi phần tử đó là một mảng.

#### Ưu điểm của tuple so với mảng

Các phần tử của Tuple rất giống với List nên về cơ bản cả hai đều có thể sử dụng trong một tình huống tương tự nhau. Tuy nhiên, vẫn có một số lợi thế của Tuple so với List:

* Chúng ta thường dùng Tuple cho các kiểu dữ liệu không đồng nhất (các phần tử khác kiểu dữ liệu) và List cho các kiểu dữ liệu đồng nhất (các phần tử cùng kiểu dữ liệu).
* Vì tuple là bất biến, nên việc lặp qua tuple nhanh hơn so với List.
* Tuple có thể được sử dụng làm khóa cho [Dictionary](https://freetuts.net/dictionary-trong-python-1961.html), trong khi list thì không thể.
* Nếu bạn có dữ liệu ít thay đổi, việc triển khai nó dưới dạng Tuple sẽ đảm bảo không bị thay đổi.

### Cách tạo Tuple trong Python

Một tuple được tạo bằng cách đặt tất cả các mục (phần tử) bên trong dấu ngoặc đơn (), được phân tách bằng dấu phẩy. Các dấu ngoặc là tùy chọn, có hoặc không đều được, nhưng tốt nhất là nên sử dụng nó.

**Cú pháp**

|  |
| --- |
| my\_tuple = (element1, element2, ...) |

Một tuple có thể có nhiều phần tử và mỗi phần tử có thể thuộc các kiểu dữ liệu khác nhau (int, float, list, string, v.v.).

**Ví dụ**

|  |
| --- |
| # Các phần tử Tuple có thể có các kiểu dữ liệu khác nhau  my\_tuple = ("freetuts", [8, 4, 6], (1, 2, 3))  print(my\_tuple)  # **Hiển thị** ("freetuts", [8, 4, 6], (1, 2, 3))    # tuple có thể được tạo bằng cách bỏ đi cặp ngoặc đơn  my\_tuple = 3, 4.6, "blog"  print(my\_tuple)  # **Hiển thị** 3, 4.6, "blog"    # Bạn có thể tách các phần tử tuple thành nhiều biến nhỏ  a, b, c = my\_tuple  print(a)  print(b)  print(c)  # **Hiển thị**  # 3  # 4.6  # blog |

Việc tạo một Tuple **chỉ có một phần tử** sẽ phức tạp hơn một xíu, bởi vì trường hợp này thì Python sẽ dễ bị nhầm lẫn sang kiểu [string](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html), vì vậy ta phải bổ sung thêm dấu phẩy ở cuối để trình biên dịch Python nhận biết đó là một Tuple.

**Ví dụ**

|  |
| --- |
| # Chỉ có một phần tử nên Python hiểu nhầm qua kiểu string  my\_tuple = ("hello")  print(type(my\_tuple))  # **Hiển thị** <class 'str'>    # Bạn thêm dấu phẩy để Python biết đây là 1 tuple  my\_tuple = ("hello",)  print(type(my\_tuple))  # **Hiển thị** <class 'tuple'>    # Bạn cũng có thể bỏ cặp dấu ngoặc đơn nhưng phải có dấu phẩy  my\_tuple = "hello",  print(type(my\_tuple))  # Output: <class 'tuple'> |

### Truy cập các phần tử của tuple trong Python

Có nhiều cách khác nhau để truy cập các phần tử của tuple.

#### Truy cập thông qua chỉ mục index

Chúng ta có thể sử dụng toán tử chỉ mục [] để truy cập đến một phần tử trong một tuple, chỉ mục bắt đầu từ 0.

Ví dụ, một tuple có 6 phần tử thì chỉ mục bắt đầu từ 0 đến 5, nên nếu bạn cố gắng truy cập đến cá phần tử khác như (6, 7, ...) thì sẽ gây ra lỗi **IndexError**.

Chỉ mục phải là một số nguyên (int), vì vậy bạn không thể sử dụng kiểu float hoặc các loại khác, điều này sẽ dẫn đến TypeError.

Tương tự, tuple lồng nhau được truy cập bằng cách sử dụng lập chỉ mục lồng nhau, như trong ví dụ dưới đây.

|  |
| --- |
| # Danh sách Tuple  my\_tuple = ('p','e','r','m','i','t')    # Lấy phần tử đầu tiên trong tuple  print(my\_tuple[0])  # Output: 'p'    # Lấy phần tử thứ 6 trong tuple  print(my\_tuple[5])  # Output: 't'    # Mỗi phần tử của tuple là một mảng hoặc 1 tuple khác  n\_tuple = ("mouse", [8, 4, 6], (1, 2, 3))    # Lấy phần tử thứ 4 của phần tử thứ nhất  print(n\_tuple[0][3])  # Output: 's'    # Lấy phần tử thứ 2 của phần tử thứ 2  print(n\_tuple[1][1])  # Output: 4 |

#### Số chỉ mục âm

Python cho phép lập chỉ mục số âm, chỉ số -1 đề cập đến phần tử cuối cùng, -2 cho phần tử cuối cùng thứ hai, v.v.

|  |
| --- |
| my\_tuple = ('p','e','r','m','i','t')    # Phần tử cuối cùng  print(my\_tuple[-1])  # Output: 't'    # Phần tử thứ 6 tính từ cuối lên  print(my\_tuple[-6])  # Output: 'p' |

#### Slicing

Chúng ta có thể truy cập vào một loạt các phần tử trong một tuple bằng cách sử dụng **toán tử slicing**, tức là dấu hai chấm ":".

* [begin:end] : Trong đó end là biên, tức sẽ ko lấy phần tử **end**mà lấy từ begin đến **end - 1**.

**VD:**

|  |
| --- |
| my\_tuple = ('p','r','o','g','r','a','m','i','z')    # Lấy phần tử thứ 2 đến thứ 4  print(my\_tuple[1:4])  # Output: ('r', 'o', 'g')    # Phần tử đầu tiên đến thứ hai (tức thứ 7 tính từ sau tới)  print(my\_tuple[:-7])  # Output: ('p', 'r')    # Phần tử thứ 8 đến cuối  print(my\_tuple[7:])  # Output: ('i', 'z')    # Lấy toàn bộ phần tử  print(my\_tuple[:])  # Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z') |

Slicing có thể được xem là tốt nhất nếu bạn dùng nó để lấy một dãy các phần tử trong Tuple.

### Thay đổi giá trị cho các phần tử Tuple trong Python

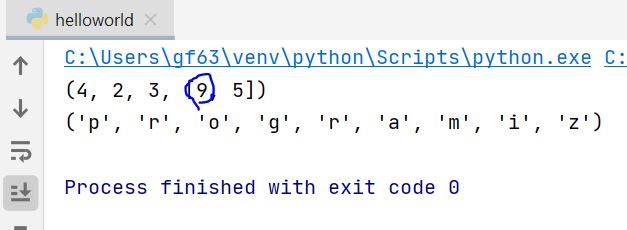
Không giống như mảng, bạn không thể thay đổi các phần tử của Tuple, điều này có nghĩa là các phần tử của một tuple không thể thay đổi một khi nó đã được gán giá trị.

Nhưng nếu phần tử tự nó là một kiểu dữ liệu giống như mảng thì ta có thể thay đổi được.

**Ví dụ**

|  |
| --- |
| my\_tuple = (4, 2, 3, [6, 5])    # Thay đổi giá trị cho phần tử thứ nhất của phần tử thứ 4 trong my\_tuple.  my\_tuple[3][0] = 9  print(my\_tuple)    # Chương trình này sẽ bị lỗi  my\_tuple = ('p','r','o','g','r','a','m','i','z')    # Bỏ dòng này sẽ không có lỗi như hình  # Bạn hãy thử mở lại dòng này sẽ thấy bị lỗi do ta cố tình thay đổi ..  # giá trị cho phần tử tuple  # my\_tuple[1] = 'a'    print(my\_tuple) |

**Hiển thị**



### Xóa Tuple

Như đã thảo luận ở trên, chúng ta không thể thay đổi các thành phần trong một tuple, điều đó cũng có nghĩa là chúng ta không thể xóa các phần tử của nó, nhưng hoàn toàn có thể xóa một tuple bằng cách sử dụng từ khóa del.

**Ví dụ**

|  |
| --- |
| my\_tuple = ('p','r','o','g','r','a','m','i','z')    # Xóa biến my\_tuple  del my\_tuple    # Lệnh này lỗi vì biến my\_tuple đã bị xóa  my\_tuple |

### Các phương thức trong Tuple

Trong lớp đối tượng Tuple hỗ trợ hai phương thức, đó là:

|  |  |
| --- | --- |
| **Method** | **Description** |
| [count(x)](https://freetuts.net/tuple-count-trong-python-5316.html) | Đếm số lần xuất hiện của một phần tử |
| [index(x)](https://freetuts.net/tuple-index-trong-python-5317.html) | Trả về chỉ mục của phần tử cần tìm |

### Một vài thao tác khác với Tuple

Sau đây là một vài thao tác khác với Tuple thường gặp nhất.

#### Kiểm tra phần tử có xuất hiện trong Tuple

Để kiểm tra một phần tử có xuất hiện trong tuple không thì ta dùng toán tử in.

**Ví dụ**

|  |
| --- |
| my\_tuple = ('a','p','p','l','e',)    # Toán tử in  print('a' in my\_tuple)  # Output: True    print('b' in my\_tuple)  # Output: False    # Toán tử not in  print('g' not in my\_tuple)  # Output: True |

#### Duyệt qua từng phần tử của Tuple

Để lặp qua từng phần tử của Tuple thì ta có thể sử dụng vòng lặp for.

**Vi dụ**

|  |
| --- |
| for name in ('John','Kate'):       print("Hello",name)  # Output:  # Hello John  # Hello Kate |

Ngoài ra vẫn còn một vài function khác thuộc lớp [Build-in function](https://freetuts.net/python-function/built-in-functions).

## KIỂU DỮ LIỆU SET TRONG PYTHON

Trong bài này chúng ta sẽ tìm hiểu **kiểu dữ liệu Set trong Python**với các thao tác như: Cách khởi tạo Set, thêm hoặc xóa các thành phần tử ra khỏi Set, và các thao tác thường dùng trên Set.

Các kiểu dữ liệu trong Python quả thật rườm rà, tại sao có mảng, tuple rồi mà lại thêm kiểu tập hợp Set làm gì nữa? Cái gì cũng có lý do của nó các bạn ạ. Sau đây chúng ta sẽ tìm hiểu về Set nhé.

**Mục lục**

* [1. Set trong Python là gì?](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h2-0)
  + [Cú pháp](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h3-0)
  + [Khởi tạo Set rỗng](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h3-1)
* [2. Làm thế nào để thay đổi Set trong Python?](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h2-1)
* [3. Làm sao để xóa phần tử ra khỏi Set trong Python](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h2-2)
* [4. Các hành phép tập hợp của Set trong Python](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h2-3)
  + [Set union](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h3-2)
  + [Set Intersection](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h3-3)
  + [Set Difference](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h3-4)
  + [Set Symmetric Difference](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h3-5)
* [5. Các thao tác khác với Set trong Python](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h2-4)
  + [Kiểm tra phần tử tồn tại trong Set](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h3-6)
  + [Lặp qua các phần tử của Set](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h3-7)
* [6. Lời kết](https://freetuts.net/kieu-du-lieu-set-trong-python-1712.html#goto-h2-5)

### Set trong Python là gì?

Set trong Python là tập các phần tử dữ liệu không có thứ tự, mỗi phần tử là duy nhất (không trùng lặp) và phải bất biến (không thể thay đổi). Tuy nhiên chúng ta có thể thêm hoặc xóa các phần tử ra khỏi Set một cách dễ dàng.

#### Cú pháp

Sau đây là cú pháp tạo Set trong Python.

|  |
| --- |
| my\_set = {element\_1, element\_2, element\_3, ...} |

Trong đó mỗi element là một phần tử và có thể có thuộc nhiều kiểu dữ liệu khác nhau, không nhất thiết phải đồng nhất.

Vì các phần tử không thể thay đổi giá trị nên **Set không cho phép lưu trữ kiểu List hoặc Dictionary**.

**Ví dụ**

|  |
| --- |
| # Tập hợp set kiểu int  my\_set = {1, 2, 3}  print(my\_set)    # Tập hợp set đa dạng kiểu khác  my\_set = {1.0, "Hello", (1, 2, 3)}  print(my\_set) |

Như bạn thấy, ở ví dụ đầu tiên thì mình truyền dữ liệu đơn giản, còn ở ví dụ thứ hai thì có thểm kiểu Tuple.

**Ví dụ khác**

|  |
| --- |
| # Set không chứa dữ liệu trùng nhau  # Như trong tập hợp này mình cố tình cho trùng giá trị 3  # Nhưng khi in ra thì nó chỉ lấy 1 số 3 mà thôi  my\_set = {1,2,3,4,3,2}  print(my\_set)  # Output: {1, 2, 3, 4}    # Set không thể chứa tập hợp có thể thay đổi như Dictionary và List  # Nên đoạn code này sẽ bị lỗi TypeError  # Do nó chứa mảng [3,4]  my\_set = {1, 2, [3, 4]}  # TypeError: unhashable type: 'list'    # Ta có thể khởi tạo giá trị của set qua ...  # đối tượng Set() trong Python  # bằng cách truyền vào một list (array)  my\_set = set([1,2,3,2])  print(my\_set)  # Output: {1, 2, 3} |

Ở ví dụ này mình đã cố tình truyền dữ liệu bị trùng và kết quả là nó lọc hết chỉ giữ lại 1 giá trị bị trùng mà thôi. Ở ví dụ cuối cùng là mình đã khởi tạo một Set từ List chứ không phải tạo phần tử Set chứa kiểu dữ liệu List đâu nhé, đừng nhầm lẫn đoạn đó.

#### Khởi tạo Set rỗng

Việc khởi tạo Set rỗng sẽ có chút nhầm lẫn, việc sử dụng cặp {} sẽ gây hiểu lầm với kiểu dữ liệu [Dictionary](https://freetuts.net/dictionary-trong-python-1961.html), vì vậy để khởi tạo một Set rỗng thì nên sử dụng đối tượng **Set()** và không truyền bất kì đối số nào vào.

**Ví dụ**

|  |
| --- |
| # Khởi tạo bằng cặp {}  a = {}    # Kiểm tra kiểu dữ liệu thì nó là một dictionary  print(type(a))  # Output: <class 'dict'>    # Khởi tạo bằng set()  a = set()    # Kiểm tra kiểu dữ liệu thì là một Set  print(type(a))  # Output: <class 'set'> |

### Làm thế nào để thay đổi Set trong Python?

Set không phải là kiểu dữ liệu **có chỉ mục** nên việc truy xuất thông qua chỉ mục index là điều không thể. Tuy nhiên nó hỗ trợ một số phương thức giúp ban làm điều này.

* add() dùng để thêm phần tử
* update() dùng để cập nhật phần tử

Hãy đọc kỹ phần ví dụ dưới đây sẽ giúp bạn hiểu rõ hơn.

|  |
| --- |
| # Khởi tạo set my\_set  my\_set = {1,3}  print(my\_set)    # Nếu bạn bỏ comment ở dòng 9,  # bạn set nhận một lỗi là  # TypeError: 'set' object does not support indexing    #my\_set[0]    # Thêm một phần tử vào set  my\_set.add(2)  print(my\_set)  # Output: {1, 2, 3}    # Thêm nhiều phần tử vào Set, ta dùng mảng để chứa các phần tử đó  my\_set.update([2,3,4])  print(my\_set)  # Output: {1, 2, 3, 4}    # Cập nhật set bằng một list và set mới  # Nó sẽ duyệt và loại bỏ bớt giá trị trùng nhau  my\_set.update([4,5], {1,6,8})  print(my\_set)  # Output: {1, 2, 3, 4, 5, 6, 8} |

**Hiển thị:**

|  |
| --- |
| {1, 3}  {1, 2, 3}  {1, 2, 3, 4}  {1, 2, 3, 4, 5, 6, 8} |

### Làm sao để xóa phần tử ra khỏi Set trong Python

Để loại bỏ phần tử ra khỏi Set thì ta sử dụng hai phương thức discard() và remove(). Cả hai đều có chung một mục đích, tuy nhiên điểm khác biệt là với discard() thì sẽ không sinh lỗi nếu giá trị cần xóa không tồn tại, còn remove() thì sẽ sinh lỗi.

Ví dụ dưới đây sẽ minh họa cho điều này:

|  |
| --- |
| # Khởi tạo my\_set  my\_set = {1, 3, 4, 5, 6}  print(my\_set)    # Xóa phần tử có giá trị 4  my\_set.discard(4)  print(my\_set)  # Output: {1, 3, 5, 6}    # Xóa phần tử có giá trị 6  my\_set.remove(6)  print(my\_set)  # Output: {1, 3, 5}    # Xóa phần tử  # không tồn tại trong my\_set  my\_set.discard(2)  print(my\_set)  # Output: {1, 3, 5} => Không bị lỗi    # Xóa một phần tử  # không tồn tại trong my\_set  # thì sẽ bị lỗi, do dùng phương thức remove  my\_set.remove(2)  # Output: KeyError: 2 |

Tương tự, chúng ta có thể xóa và trả về phần tử đã xóa bằng phươn thức pop(), hoặc xóa toàn bộ phần tử bằng phương thức clear().

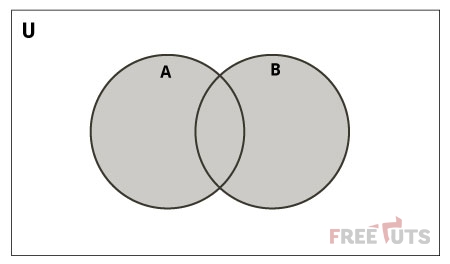
|  |
| --- |
| # Khởi tạo my\_set  my\_set = set("HelloWorld")  print(my\_set)  # Output: {'r', 'd', 'o', 'e', 'W', 'H', 'l'}    # pop một phần tử ngẫu nhiên  print(my\_set.pop())  # Output: Tùy thuộc vào giá trị mà nó xóa mà in kết quả khác nhau    # pop thêm một phần tử ngẫu nhiên khác  my\_set.pop()  print(my\_set)  # Output: Cũng là random    # clear my\_set, kết quả trả về Set rỗng  my\_set.clear()  print(my\_set)  #Output: set() |

### Các hành phép tập hợp của Set trong Python

Set có thể sử dụng với nhiều phép toán như liên kết, hợp, ... bằng cách sử dụng các toán tử và phương thức mà nó hỗ trợ sẵn.

#### Set union

Toán tử union kí hiệu là |, nó sẽ trả về hợp của hai Set

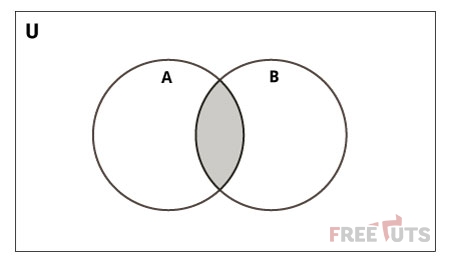


**Ví dụ**

|  |
| --- |
| # Khởi tạo set A và B  A = {1, 2, 3, 4, 5}  B = {4, 5, 6, 7, 8}    # Sử dụng toán tử union |  print(A | B)  # Output: {1, 2, 3, 4, 5, 6, 7, 8} |

#### Set Intersection

Phép giao intersection ký hiệu là &, nó sẽ trả về phần giao của hai set

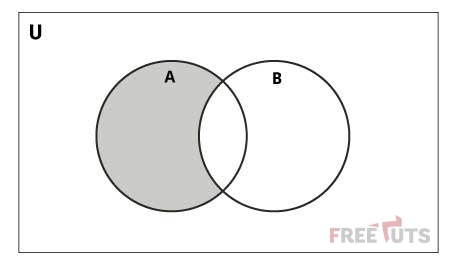


**Ví dụ**

|  |
| --- |
| # Khởi tạo set A và B  A = {1, 2, 3, 4, 5}  B = {4, 5, 6, 7, 8}    # Sử dụng toán tử &  print(A & B)  # Output: {4, 5} |

#### Set Difference

Phép Difference kí hiệu là -, nó sẽ trả về các phần tử của A không có trong B

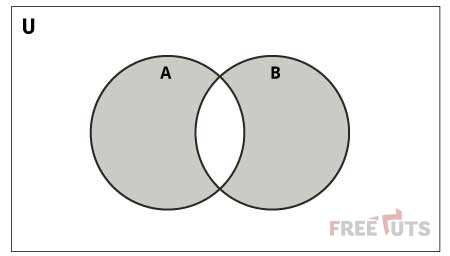


**Ví dụ**

|  |
| --- |
| # Khởi tạo set A và B  A = {1, 2, 3, 4, 5}  B = {4, 5, 6, 7, 8}    # Sử dụng toán tử -  print(A - B)  # Output: {1, 2, 3} |

#### Set Symmetric Difference

Toán tử Set Symmetric Difference kí hiệu là ^, Nó sẽ trả về phần tử A không có trong B và phần tử B không có trong A.



**Ví dụ**

|  |
| --- |
| # Khởi tạo A và B  A = {1, 2, 3, 4, 5}  B = {4, 5, 6, 7, 8}    # Sử dụng toán tử ^  print(A ^ B)  # Output: {1, 2, 3, 6, 7, 8} |

### Các thao tác khác với Set trong Python

Bây giờ mình sẽ làm một vài ví dụ về các thao tác khác thường gặp trên Set, đây là những thao tác được sử dụng rất nhiều khi bạn làm việc với Python.

#### Kiểm tra phần tử tồn tại trong Set

Để kiểm tra một phần tử có tồn tại không trong Set thì ta sử dụng toán tử in, toán tử này được sử dụng khá nhiều, hầu như tất cả các loại dữ liệu đều có.

**Ví dụ**

|  |
| --- |
| # Khởi tạo my\_set  my\_set = set("apple")    # Kiểm tra xem 'a' có tồn tại trong set không  print('a' in my\_set)  # Output: True    # Kiểm tra xem 'p' có tồn tại trong set không  print('p' not in my\_set)  # Output: False |

#### Lặp qua các phần tử của Set

Để lặp qua các phần tử của Set thì ta có thể sử dụng [vòng lặp for](https://freetuts.net/vong-lap-for-trong-python-1689.html), đây là vòng lặp được sử dụng nhiều nhất trong Python.

**Ví dụ**

|  |
| --- |
| for letter in set("apple"):      print(letter)    # Result:  # p  # l  # a  # e |

### Lời kết

Như vậy là mình đã giới thiệu xong kiến thức cơ bản về tập hợp Set trong Python.

Bạn cần lưu ý rằng với Set thì không thể thay đổi giá trị của các phần tử, bản chất khi sử dụng phương thức update là bạn đã bổ sung các phần tử. Tuy nhiên bạn có thể thực hiện các thao tác xóa phần tử ra khỏi set

## KIỂU DỮ LIỆU DICTIONARY TRONG PYTHON

Trong bài này chúng ta sẽ tìm hiểu kiểu dữ liệu từ điển **Dictionary trong Python**, đây là kiểu dữ liệu rất hay, thường được dùng để kết hợp với JSON để xử lý dữ liệu.

Nếu bạn đã từng học qua PHP thì có thể xem dictionary python là một mảng kết hợp gôm các cặp key : value. Đây cũng là cấu trúc phù hợp với chuỗi JSON.

**Mục lục**

* [1. Tạo dictionary Python](https://freetuts.net/dictionary-trong-python-1961.html#goto-h2-0)
* [2. Truy xuất phần tử của Dictionary trong Python](https://freetuts.net/dictionary-trong-python-1961.html#goto-h2-1)
* [3. Thay đối giá trị phần tử của Dictionary Python](https://freetuts.net/dictionary-trong-python-1961.html#goto-h2-2)
* [4. Xóa phần tử ra khỏi Dictionary](https://freetuts.net/dictionary-trong-python-1961.html#goto-h2-3)
* [5. Thay đổi các phần tử dictionary qua vòng lặp](https://freetuts.net/dictionary-trong-python-1961.html#goto-h2-4)

### Tạo dictionary Python

Kiểu từ điển dictionary là loại dữ liệu tổng hợp được sắp xếp không có thứ tự, mỗi phần tử sẽ có hai phần đó là key và value.

Nếu bạn đã từng học qua PHP thì nó giống như một mảng Array, còn với Javascript thì nó giống như là một Object.

**Ví dụ**

|  |
| --- |
| my\_dict = {1: 'apple', 2: 'ball'} |

Như trong ví dụ này thì chúng ta có hai phần tử và mỗi phần tử có hai thông số key : value, đó là 1 : 'apple' và 2 : 'ball'.

Khi bạn truy xuất một phần tử trong dictionary sẽ rất nhanh nếu biết trước tên khóa đó là gì.

Giả sử mình cần lấy phần tử thứ có key là 1 thì sẽ lấy như sau:

|  |
| --- |
| my\_dict[1] |

Các key có thể là một số tự nhiên hoặc một chuỗi được đặt trong ngoặc đơn hoặc ngoặc kép.

### Truy xuất phần tử của Dictionary trong Python

Để truy xuất đến các phần tử trong dictionary thì ta dựa vào tên key của phần tử đó, vì vậy bạn phải biết tên key nhé.

Chúng ta có hai cách lấy, cách thứ nhất là thông qua cặp dấu ngoặc vuông và cách thứ hai là thông qua phương thức get() được tích hợp sẵn trong dictionary.

Ví dụ dưới đây thể hiện cho hai cách này.

|  |
| --- |
| my\_dict = {'name':'Jack', 'age': 26}    # Output: Jack  print(my\_dict['name'])    # Output: 26  print(my\_dict.get('age')) |

Sự khác nhau giữa chúng là gì? Thực tế kết quả không có gì khác nhau, chỉ là với phương thức get() thì sẽ trả về none nếu key không tồn tại, còn cặp ngoặc vuông sẽ trả về lỗi.

**Ví dụ**

|  |
| --- |
| # Error vì key abc không tồn tại  print(my\_dict['abc']) |

### Thay đối giá trị phần tử của Dictionary Python

Chúng ta có hai thao tác chủ yếu đó là thêm hoặc thay đổi giá trị cho một phần tử, và đương nhiên để thay đổi thì ta phải biết tên key.

Dictionary là kiểu dữ liệu cho phép ta thay đổi các phần tử. Để thêm một phần tử mới thì ta sử dụng toán tử gán (=), nếu phần tử cần thêm chưa tồn tai thì dictionary sẽ bổ sung nó vào, ngược lại nếu đã tồn tại thì sẽ được cập nhật giá trị.

Hãy xem ví dụ dưới đây.

|  |
| --- |
| my\_dict = {'name':'Jack', 'age': 26}    # Thay đổi giá trị  my\_dict['age'] = 27    #Output: {'age': 27, 'name': 'Jack'}  print(my\_dict)    # Thêm phần tử  my\_dict['address'] = 'Downtown'    # Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}  print(my\_dict) |

**Hiển thị:**

|  |
| --- |
| {'name': 'Jack', 'age': 27}  {'name': 'Jack', 'age': 27, 'address': 'Downtown'} |

### Xóa phần tử ra khỏi Dictionary

Chúng ta có thể xóa một phần tử ra khỏi dictionary bằng cách sử dụng phương thức pop(), tham số truyền vào là tên key cần xóa, kết quả trả về là giá trị của phần tử đã xóa.

Hoặc sử dụng phương thức popitem() để xóa phần tử ra khỏi dictionary và kết quả trả về là cặp giá trị (key, value) đã xóa, và sử dụng phương thức clear() để xóa toàn bộ phần tử.

Ngoài ra nếu bạn muốn xóa luôn dictionary đó thì có thể sử dụng từ khóa del, chi tiết sẽ xem ở ví dụ dưới đây.

|  |
| --- |
| # Xóa sử dụng phương thức pop()  # Output: 16  print(squares.pop(4))    # Output: {1: 1, 2: 4, 3: 9, 5: 25}  print(squares)    # Xóa sử dụng phương thức popitem()  # Output: (1, 1)  print(squares.popitem())    # Output: {2: 4, 3: 9, 5: 25}  print(squares)    # Xóa sử dụng từ khóa del  del squares[5]    # Output: {2: 4, 3: 9}  print(squares)    # Xóa tất cả phần tử sử dụng phương thức clear()  squares.clear()    # Output: {}  print(squares)    # Xóa biến squares ra khỏi bộ nhớ  del squares    # Throws Error bởi vì biến squares đã bị xóa nên không tồn tại  # print(squares) |

**Hiển thị:**

|  |
| --- |
| 16  {1: 1, 2: 4, 3: 9, 5: 25}  (1, 1)  {2: 4, 3: 9, 5: 25}  {2: 4, 3: 9}  {} |

### Thay đổi các phần tử dictionary qua vòng lặp

Trường hợp bạn muốn tất cả các phần tử được thay đổi thông qua một công thức nào đó thì có thể kết hợp sử dụng vòng lặp for.

**Ví dụ**: cho dictionary sau:

|  |
| --- |
| squares = {1:1, 2:4, 3:9, 4:16, 5:25} |

Bây giờ mình muốn tạo thành một squares mới với mỗi phần tử sẽ được nhân double lên. Với cách làm thông thường thì mình sẽ làm như sau:

|  |
| --- |
| squares = {1:1, 2:4, 3:9, 4:16, 5:25}  squares[1] = squares[1]\*2  squares[2] = squares[2]\*2  squares[3] = squares[3]\*2  squares[4] = squares[4]\*2    #Output: {1: 2, 2: 8, 3: 18, 4: 32, 5: 25}  print(squares) |

Trên là một vài chia sẻ về cách sử dụng kiểu từ điển dictionary trong Python, hy vọng qua bài này bạn hiểu được cách khởi tạo, thêm phần tử, xóa phần tử, cũng như thay đổi giá trị của các phần tử trong dictionary.

## EXCEPTIONS TRONG PYTHON, XỬ LÝ LỖI ĐƠN GIẢN

Trong bài này chúng ta sẽ tìm hiểu Exception trong Python, đây là mã lệnh giúp xử lý các lỗi từ cơ bản đến nâng cao trong quá trình lập trình Python.

Một exception có thể được xem là một đoạn code bất thường trong một chương trình dẫn đến sự gián đoạn trong dòng chảy của chương trình đó, hay còn gọi là lỗi.

Bất cứ khi nào có exception xảy ra, chương trình sẽ dừng thực thi nên các đoạn mã phía dưới không được biên dịch. Do đó, một exception là lỗi mà trình biên dịch python không thể chạy được.

**Mục lục**

* [1. Các exception trong Python có sẵn](https://freetuts.net/exceptions-trong-python-2060.html#goto-h2-0)
* [2. Chuyện gì xảy ra nếu không dùng exception?](https://freetuts.net/exceptions-trong-python-2060.html#goto-h2-1)
* [3. Lệnh try ... except trong Python](https://freetuts.net/exceptions-trong-python-2060.html#goto-h2-2)
* [4. Except bị thiếu exception](https://freetuts.net/exceptions-trong-python-2060.html#goto-h2-3)
* [5. Khai báo nhiều exception](https://freetuts.net/exceptions-trong-python-2060.html#goto-h2-4)
* [6. Khối finaly trong Exception](https://freetuts.net/exceptions-trong-python-2060.html#goto-h2-5)
* [7. Nhảy exception với từ khóa raise](https://freetuts.net/exceptions-trong-python-2060.html#goto-h2-6)
* [8. Tự tạo exception trong Python](https://freetuts.net/exceptions-trong-python-2060.html#goto-h2-7)

### Các exception trong Python có sẵn

Python cung cấp cho chúng ta cách xử lý exception để giúp chương trình không bị gián đoạn. Vì vậy với những đoạn code bạn cảm thấy không an toàn thì hãy đưa nó vào một exception.

Dưới đây là danh sách những exception thường được sử dụng trong lập trình Python:

1. **ZeroDivisionError**: Xảy ra khi một số được chia cho số không.
2. **NameError**: Xảy ra khi tên không tồn tại, có thể là cục bộ hoặc toàn cục.
3. **ValueError**: Xảy ra khi bạn gán dữ liệu cho biến không chính xác.
4. **IndentationError**: Xảy ra khi chương trình thụt hàng không đúng.
5. **IOError**: Xảy ra khi xử lý nhập xuất bị lỗi.
6. **EOFError**: Xảy ra khi kết thúc tập tin mà các thao tác vẫn còn thực hiện trên đó.
7. **KeyError**: Xảy ra khi một key không tồn tại trong dictionary
8. **ImportError**: Xảy ra khi bạn import một package / module không thành công
9. **SyntaxError**: Xảy ra khi lỗi liên quan đến cú pháp
10. **ZeroDivisionError**: Xảy ra khi bạn chia một số cho 0
11. **ArithmeticError**: Lỗi liên quan đến toán học

Tuy nhiên, bạn có thể tự tạo ra các exception trong Python bằng cách sử dụng từ khóa except (mình sẽ trình bày nó ở phần cuối cùng của bài viết này).

### Chuyện gì xảy ra nếu không dùng exception?

Như chúng ta đã thảo luận ở trên, exception là một lệnh điều kiện bắt lỗi trong quá trình thực thi chương trình.

Hãy xem xét ví dụ sau.

|  |
| --- |
| a = int(input("Enter a:"))  b = int(input("Enter b:"))  c = a/b;  print("a/b = %d"%c)    #other code:  print("Vị trí khác của chương trình") |

Giả sử mình nhập số a = 10, b = 0, chạy chương trình lên sẽ có kết quả như sau:

|  |
| --- |
| Enter a:10  Enter b:0  Traceback (most recent call last):    File "exception-test.py", line 3, in <module>      c = a/b;  ZeroDivisionError: division by zero |

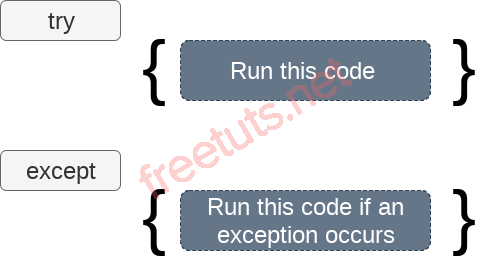
Như vậy bạn sẽ nhận được một thông báo lỗi **ZeroDivisionError**.

Để bắt lỗi này và giúp chương trình không bị gián đoạn thì ta sử dụng lệnh try ... except.

### Lệnh try ... except trong Python

Nếu chương trình python chứa những đoạn code đáng ngờ thì bạn hãy đặt chúng vào một exception, bằng cách ném chúng vào một khối lệnh **try .. except**.

Thường trong try là đoạn code nghi ngờ có lỗi, trong except là đoạn code thông báo lỗi hoặc trả về lỗi. Bạn có thể định nghĩa nhiều except.

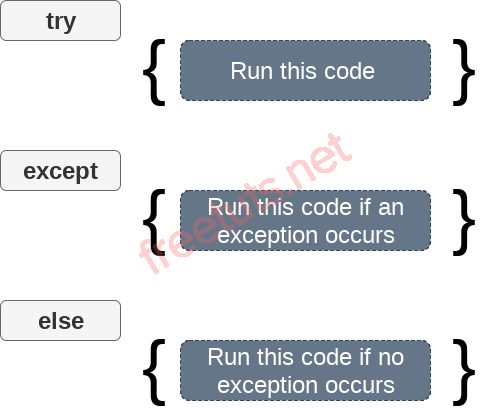


**Cú pháp**

|  |
| --- |
| try:      #block of code    except Exception1:      #block of code    except Exception2:      #block of code    #other code |

Chúng ta cũng có thể sử dụng từ khóa ELSE kết hợp trong **try-except** để thực thi những đoạn code trong trường hợp **except-block** không chạy.

|  |
| --- |
| try:      #block of code    except Exception1:      #block of code    else:      #Đoạn này sẽ chạy nếu except block không chạy |



**Ví dụ**

|  |
| --- |
| try:      a = int(input("Enter a:"))      b = int(input("Enter b:"))      c = a/b;      print("a/b = %d"%c)  except Exception:      print("Không thể chia cho 0")  else:      print("Đây là đoạn code ELSE") |

**Hiển thị**

|  |
| --- |
| Enter a:10  Enter b:2  a/b = 5  Đây là đoạn code ELSE |

### Except bị thiếu exception

Ở các ví dụ trên mình đã xác định rõ lỗi cho các except bằng cách đặt tên lỗi phía sau, lúc này nếu đoạn code nào trong **try block** bị lỗi nào thì sẽ chạy đoạn code ở **except** tương ứng với lỗi đó.

|  |
| --- |
| except Exception: |

Tuy nhiên bạn hoàn toàn có thể bỏ từ khóa Exception như ở ví dụ dưới đây.

|  |
| --- |
| try:      a = int(input("Enter a:"))      b = int(input("Enter b:"))      c = a/b;      print("a/b = %d"%c)  except:      print("Không thể chia cho 0")  else:      print("Đoạn code trong Else") |

Lúc này dù thể loại lỗi là gì đi nữa thì lỗi đó vẫn được ném vào except.

**Tóm lại bạn cần phải ghi nhớ những điều sau:**

* Python cho phép chúng ta không cần phải chỉ định **exception**ở lệnh **except**.
* Có thể khai báo nhiều **except,**vì khối **try**có thể ném ra nhiều loại **exception**khác nhau.
* Chúng ta cũng có thể chỉ định một khối **else** cùng với câu lệnh **try-except** và sẽ được thực thi nếu không có **exception**nào sinh ra trong khối **try**.
* Các câu lệnh không ném exception nên đặt bên trong khối **else**.

**Ví dụ:**

|  |
| --- |
| try:      #Đoạn này sẽ sinh ra lỗi nếu file không tồn tại.      fileptr = open("file.txt","r")  except IOError:      print("File không tồn tại")  else:      print("File mở thành công")      fileptr.close() |

### Khai báo nhiều exception

Nếu bạn muốn nhiều exception cùng thực hiện chung một đoạn code thì sử dụng cú pháp sau để kai báo.

|  |
| --- |
| except (<Exception 1>,<Exception 2>,<Exception 3>,...<Exception n>) |

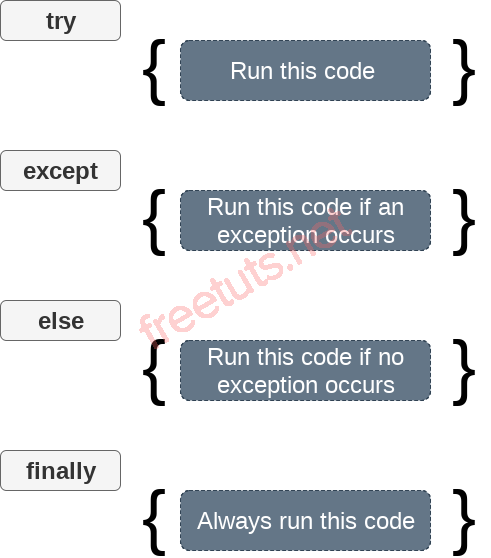
**Ví dụ**

|  |
| --- |
| try:      a=10/0;  except ArithmeticError,StandardError:      print "Lỗi tính toán"  else:      print "Thành công" |

### Khối finaly trong Exception

Nếu khối else sẽ được thực hiện nếu không có exception nào được sinh ra thì khối finany sẽ **luôn luôn được thực thi** vì nó là một khối đặc biệt. Cú pháp của nó như sau:

|  |
| --- |
| try:      # Khối chứa đoạn code      # có khả năng sinh lỗi  finally:      # Khối chứa đoạn code      # luôn luôn thực thi |



**Ví dụ**

|  |
| --- |
| try:      fileptr = open("file.txt","r")      try:          fileptr.write("Nội dung ghi vào file")      finally:          fileptr.close()          print("Đóng file")  except:      print("Mở file lỗi") |

### Nhảy exception với từ khóa raise

Ở những ví dụ trên sẽ sinh ra những exception mà Python có hỗ trợ. Trường hợp bạn muốn tự định nghĩa exception, hoặc nhảy đến exception nào đó thì có thể sử dụng từ khóa **raise**.

**Cú pháp**:

|  |
| --- |
| raise Exception\_class,<value> |

**Ví dụ**:

|  |
| --- |
| try:      age = int(input("Nhập tuổi của bạn?"))      if age<18:          raise ValueError      else:          print("Tuổi không hợp lệ")  except ValueError:      print("Tuổi quá nhỏ") |

Nếu bạn chạy và nhập tuổi bé hơn 18 thì sẽ xuất hiện lỗi "Tuổi quá nhỏ".

### Tự tạo exception trong Python

Python cho phép chúng ta tạo ra các exception bằng cách sử dụng từ khóa except.

Mình khuyên bạn nên đọc phần này sau khi học xong kiến thức về lớp, bởi để định nghĩa một exception thì bạn phải sử dụng **class**.

**Ví dụ**: Tự tạo ra exception **ErrorInCode**.

|  |
| --- |
| class ErrorInCode(Exception):      def \_\_init\_\_(self, data):          self.data = data      def \_\_str\_\_(self):          return repr(self.data)    try:      raise ErrorInCode(2000)  except ErrorInCode as ae:      print("Received error:", ae.data) |

Như mình đã nói ở trên, vì trong ví dụ này có sử dụng class để tạo ra một đối tượng exception nên bạn phải hiểu về class-object thì mới hiểu được.

Trên là cách sử dụng exception để xử lý ngoại lệ trong Python. Hy vọng qua bài này bạn sẽ hiểu và vận dụng trong quá trình học tập và làm việc với Python.

## BÀI TẬP PYTHON: THỰC HÀNH VỚI CÁC KIỂU DỮ LIỆU

Trong bài này mình sẽ tổng hợp các bài tập Python thực hành với các kiểu dữ liệu như kiểu mảng list, tuple, set, dictionary. Đây là những bài tập ở mức cơ bản dành cho newbie.

Để làm được những bài này thì bạn phải có kiến thức nền tảng về Python, vì vậy đừng quên học toàn bộ những nội dung kiến thức ở các bài trước nhé, nhất là kiến thức lệnh rẻ nhánh và vòng lặp.

**Câu hỏi thường gặp liên quan:**

* [In ra các số chẵn trong mảng bằng Python](https://freetuts.net/ref/in-ra-cac-so-chan-trong-mang-bang-python-419.html)
* [Tìm kiếm phần tử trong mảng bằng Python](https://freetuts.net/ref/tim-kiem-phan-tu-trong-mang-bang-python-420.html)
* [Sự khác nhau giữa mảng, tuple, dictionary và set trong Python](https://freetuts.net/ref/su-khac-nhau-giua-mang-tuple-dictionary-va-set-python-422.html)
* [Sắp xếp mảng tăng / giảm dần bằng Python](https://freetuts.net/ref/sap-xep-mang-tang-giam-dan-bang-python-423.html)
* [Chương trình quản lý sinh viên Python lưu mảng và dictionary](https://freetuts.net/ref/chuong-trinh-quan-ly-sinh-vien-python-421.html)

# OBJECT & CLASS

## MODULE TRONG PYTHON: CÁCH TẠO VÀ SỬ DỤNG MODULE

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.

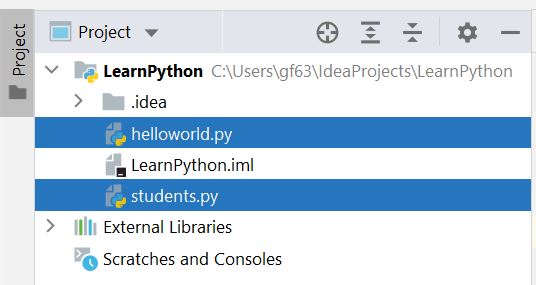
Các module giúp chúng ta code một cách linh hoạt hơn, mỗi file sẽ chứa những đoạn code phục vụ cho một chức năng cụ thể, được sắp xếp tùy vào cách thiết kế của mỗi lập trình viên.

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.

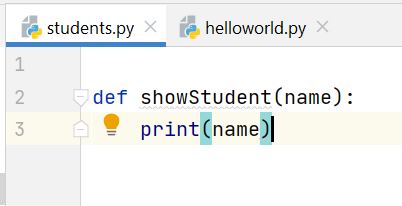
Giả sử mình cần tạo một thư viện chứa những hàm dùng để quản lý sinh viên, lúc này mình sẽ tạo một file tên là students.py cùng cấp với file chạy chính như sau:



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.

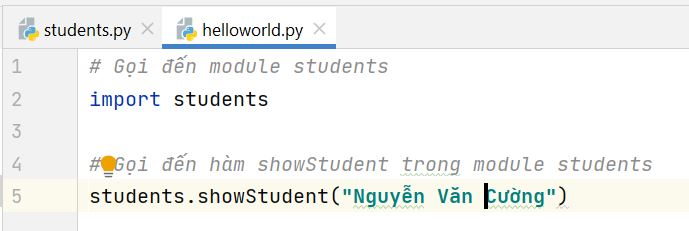
Trong file students.py bạn hãy dùng lệnh def để tạo một hàm như sau:

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

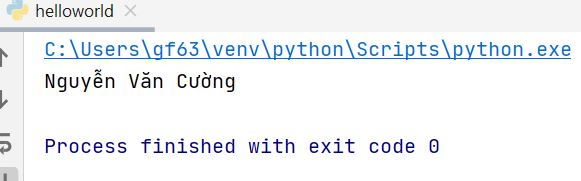


Trong file helloworld.py bạn hãy viết đoạn code sau:

|  |
| --- |
| # 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:

|  |
| --- |
| 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.

|  |
| --- |
| import module |

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

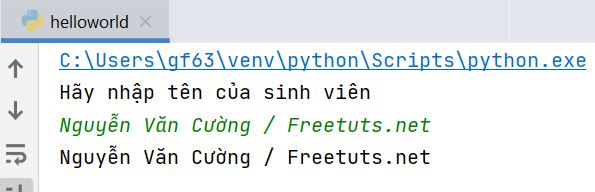
|  |
| --- |
| import module1,module2,........ module n |

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

**Ví dụ**

|  |
| --- |
| # 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**

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

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

**calculation.py**

|  |
| --- |
| #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**

|  |
| --- |
| 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)) |

**Hiển thị:**

|  |
| --- |
| 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 \*.

|  |
| --- |
| 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:**

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

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

|  |
| --- |
| #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**.

|  |
| --- |
| import json    List = dir(json)    print(List) |

**Hiển thị**

|  |
| --- |
| ['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().

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

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

|  |
| --- |
| 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:

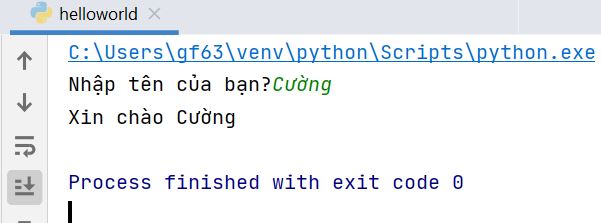
* **Biến toàn cục**: là biến có thể sử dụng ở tất cả các vị trí trong chương trình chính, trừ trong hàm.
* **Biến cục bộ**: Là biến chỉ sử dụng được trong một phạm vi nhất định, ví dụ trong hàm.

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.

|  |
| --- |
| # 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) |

**Hiển thị**



### Xem các module có sẵn trong Python

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

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

Kết quả nó sẽ trả về danh sách như sau:

|  |
| --- |
| \_\_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**.



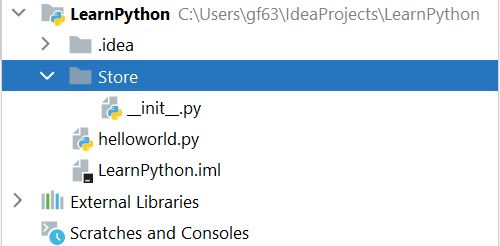
**Như bạn thấy**:

* 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:

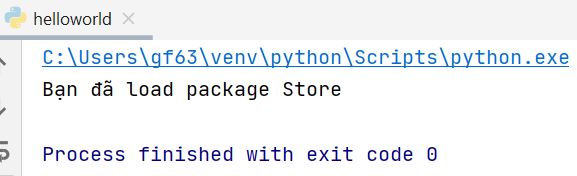
**Store/\_\_init\_\_.py**

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

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

|  |
| --- |
| import Store |

Chạy lên bạn sẽ nhận được kết quả như sau:

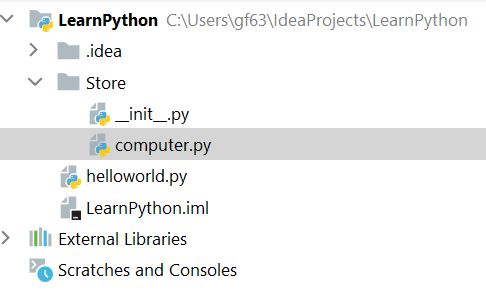


Như vậy đoạn code trong file \_\_init\_\_.py đã tự động được gọi.

### 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:

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

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

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

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

|  |
| --- |
| 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.

|  |
| --- |
| 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.

|  |
| --- |
| 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:

|  |
| --- |
| 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.

|  |
| --- |
| 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:

|  |
| --- |
| 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.

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

**Hiển thị**



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.

|  |
| --- |
| s1 = Student()  s2 = Student()  s3 = Student() |

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

|  |
| --- |
| 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() |

**Hiển thị**



### 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:

|  |
| --- |
| class Dog:      name = "",      country = "" |

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

|  |
| --- |
| 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.

|  |
| --- |
| 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.

|  |
| --- |
| 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.

|  |
| --- |
| 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.

|  |
| --- |
| 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.

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

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

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

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

|  |
| --- |
| 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).

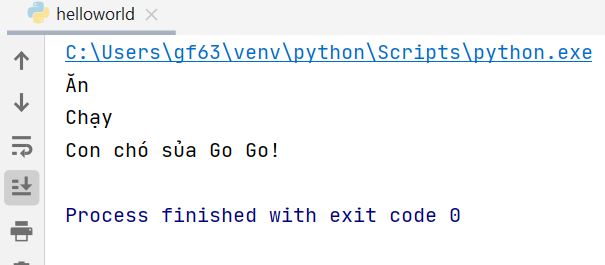
Việc kế thừa này là rất hay và thực tế.

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.

|  |
| --- |
| # 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() |

**Hiển thị**



### 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:

|  |
| --- |
| # 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.

|  |
| --- |
| 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:

|  |
| --- |
| 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.

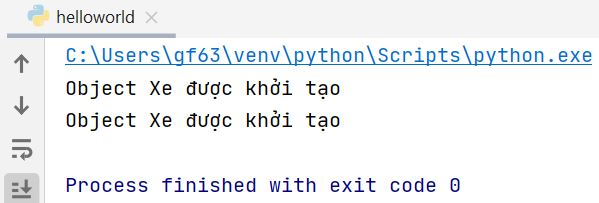
Cú pháp như sau:

|  |
| --- |
| 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.

|  |
| --- |
| class Xe:      def \_\_init\_\_(self):          print("Object Xe được khởi tạo")    x1 = Xe()  x2 = Xe() |

**Hiển thị**



### Tham số truyền vào hàm khởi tạo Python

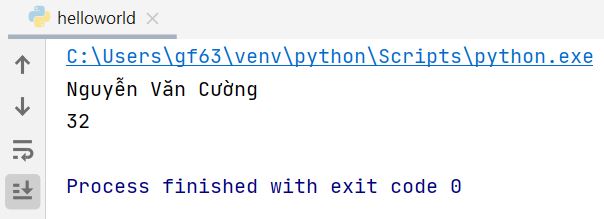
Hàm khởi tạo cũng là một phương thức bình thường nên bạn cũng có thể truyền tham số cho nó.

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.

|  |
| --- |
| 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) |

**Hiển thị:**



### 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:

|  |
| --- |
| class lopCha:      # Code của lớp cha  class lopCon(lopCha):      # Code của lớp con |

**VD**: Khao báo lớp XeDap kế thừa từ lớp Xe.

|  |
| --- |
| class Xe:      # Code ...    class XeDap(Xe):      # Code ... |

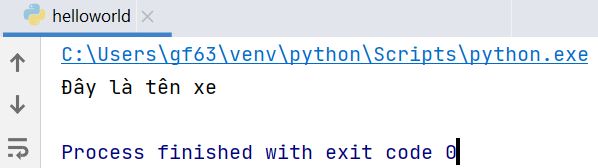
Phần này bạn chỉ cần hiểu khái niệm và cú pháp thôi nhé, còn chi tiết cách dùng kế thừa trong Python thì chúng ta sẽ học ở những phần tiếp theo dưới đây.

### 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

|  |
| --- |
| 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() |

**Hiển thị**



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

|  |
| --- |
| 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") |

**Hiển thị**



### 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.

|  |
| --- |
| 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() |

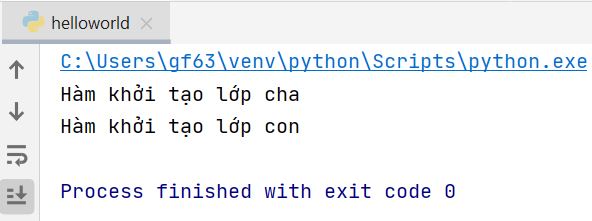
**Hiển thị**



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

|  |
| --- |
| 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() |

**Hiển thị**



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

|  |
| --- |
| 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:

|  |
| --- |
| 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:

|  |
| --- |
| 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**.

|  |
| --- |
| # 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.

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

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

|  |
| --- |
| 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() # **Hiển thị** Hi, lớp con |

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

|  |
| --- |
| 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() # **Hiển thị** Hi, lớp cha |

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

|  |
| --- |
| class LopOngNoi:      def sayHi(self):          print("Hi, lớp ông nội")    class LopCha(LopOngNoi):      pass    class LopCon (LopCha):      pass    # Main  c = LopCon()  c.sayHi() # **Hiển thị** 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.

|  |
| --- |
| 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

|  |
| --- |
| 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.

|  |
| --- |
| 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.

|  |
| --- |
| 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())  # **Hiển thị** 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).

|  |
| --- |
| 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:

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

**Hiển thị:**



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 ...

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

|  |
| --- |
| 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**.

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

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

|  |
| --- |
| 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

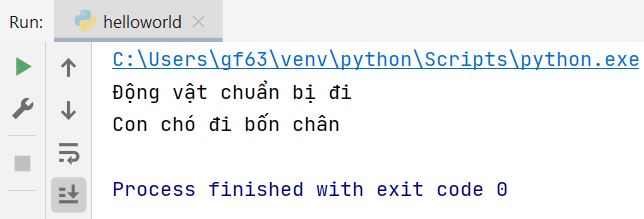
Nếu sử dụng chung hàm **move**ở **Animal** thì không đáp ứng được yêu cầu của bài toán. Đây là một *ví dụ điển hình để trả lời cho câu hỏi "*khi nào nên sử dụng override trong Python?*".*

### 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:

|  |
| --- |
| 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() |

**Hiển thị:**



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

|  |
| --- |
| 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.

**VD:**

|  |
| --- |
| 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() |

**Hiển thị:**



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.

|  |
| --- |
| class Animal:      def move(self):          pass        def eat(self):          pass |

Bây giờ ta sẽ tạo một class **Dog**:

|  |
| --- |
| 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**:

|  |
| --- |
| 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)

# ADVANCED TOPICS

## ITERATORS TRONG PYTHON

Trong bài này chúng ta sẽ tìm hiểu khái niệm **iterators trong Python**, đây là những đối tượng có những hàm hoặc phương thức hỗ trợ lặp riêng, thường là những kiểu dữ liệu dạng danh sách như mảng.

Mặc dù có thể sử dụng vòng lặp để lặp qua các phần tử của các iterators. Tuy nhiên, việc sử dụng những hàm hỗ trợ riêng của nó sẽ giúp ta lặp một cách an toàn hơn, điển hình nhất là việc ghi nhớ lại vị trí của phần đã lặp cuối cùng.

**Mục lục**

* [1. Iterators trong Python là gì?](https://freetuts.net/iterators-trong-python-3492.html#goto-h2-0)
  + [Lặp tuple với Iterators trong Python](https://freetuts.net/iterators-trong-python-3492.html#goto-h3-0)
  + [Lặp chuỗi bằng Iterators trong Python](https://freetuts.net/iterators-trong-python-3492.html#goto-h3-1)
  + [Lặp mảng bằng Iterators trong Python](https://freetuts.net/iterators-trong-python-3492.html#goto-h3-2)
* [2. lặp Iterators trong Python](https://freetuts.net/iterators-trong-python-3492.html#goto-h2-1)
* [3. Tự tạo ra Iterators trong Python](https://freetuts.net/iterators-trong-python-3492.html#goto-h2-2)
* [4. StopIteration trong Python](https://freetuts.net/iterators-trong-python-3492.html#goto-h2-3)

### Iterators trong Python là gì?

Trong Python, iterators là những đối tượng (object) được hỗ trợ những tính năng lặp riêng biệt, không cần phải sử dụng những vòng lặp thông thường như [vòng lặp for](https://freetuts.net/vong-lap-for-trong-python-1689.html) hay [vòng lặp while](https://freetuts.net/vong-lap-while-trong-python-685.html).

Một số đối tượng như [list](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html), [tuple](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html), [string](https://freetuts.net/kieu-du-lieu-string-trong-python-1711.html) là những **iterators**, bởi bạn có thể sử dụng hàm **iter()** để tạo chúng thành một iterators, sau đó sử dụng những hàm như **next()** để lặp qua từng phần tử.

Về mặt kỹ thuật thì một iterator object phải implement từ hai phương thức \_\_iter\_\_() và \_\_next\_\_(), được gọi chung là **iterator protocol**. Và để giúp lập trình viên sử dụng dễ dàng hơn thì hai build-in function ra đời, đó là **iter()** và **next()**.

#### Lặp tuple với Iterators trong Python

**VD:**

|  |
| --- |
| # Danh sách dạng tuple  words = ("Xin", "chào", "các", "bạn")    # Chuyển nó thành Iterators  iword = iter(words)    # Sử dụng next để lấy từng phần tử  print(next(iword)) # Xin  print(next(iword)) # Chào  print(next(iword)) # Các  print(next(iword)) # Bạn |

**Hiển thị:**



Hàm next sẽ trả về phần tử tiếp theo tính từ vị trí được đánh dấu là đã lặp cuối cùng. Chính vì vậy mình đã dễ dàng lấy ra 4 phần tử bằng bốn lệnh **next()**.

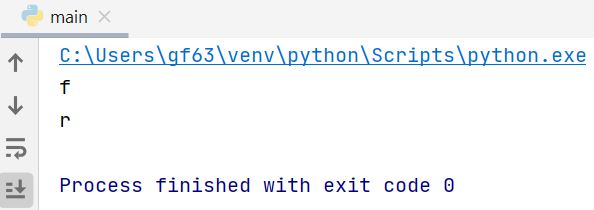
#### Lặp chuỗi bằng Iterators trong Python

Ta sẽ làm một ví dụ rất đơn giản, đó là in ra lần lượt từng kí tự của chuỗi "freetuts.net".

Đương nhiên bạn có thể sử dụng vòng lặp để làm bài này, nhưng cách đơn giản nhất là sử dụng iterators.

|  |
| --- |
| # Chuỗi  domain = "freetuts.net"    # Chuyển thành Iterator  idomain = iter(domain)    # Lấy ký tự đầu tiên  print(next(idomain))    # Lấy ký tự thứ 2  print(next(idomain)) |

**Hiển thị**



#### Lặp mảng bằng Iterators trong Python

|  |
| --- |
| # Danh sách mảng  my\_list = [4, 7, 0, 3]    # Chuyển sang iterator  my\_iter = iter(my\_list)    # Lấy phần tử đầu tiên  print(next(my\_iter))    # Lấy phần tử tiếp theo (thứ 2)  print(next(my\_iter))    # Lấy phần tử tiếp theo (thứ 3)  print(my\_iter.\_\_next\_\_())    # Lấy phần tử tiếp theo (thứ 4)  print(my\_iter.\_\_next\_\_()) |

Trong ví dụ này mình có sử dụng my\_iter.\_\_next\_\_(), thực ra nó có công dụng giống với hàm next(my\_iter).

### lặp Iterators trong Python

Bạn có thể sử dụng vòng lặp for hoặc while để lặp qua các phần tử của đối tượng Iterators.

**Ví dụ vòng lặp for in 1**:

|  |
| --- |
| # Danh sách mảng  my\_list = [4, 7, 0, 3]    # Iterator  my\_iter = iter(my\_list)    # Lặp  for i in range(0, len(my\_list)):      print(next(my\_iter)) |

**Ví dụ vòng lặp for in 2**:

|  |
| --- |
| # Danh sách mảng  my\_list = [4, 7, 0, 3]    # Iterator  my\_iter = iter(my\_list)    # Lặp  for item in my\_iter:      print(item) |

**Ví dụ vòng lặp while**:

Bạn phải kết hợp với lênh **try .. except** để bắt lỗi khi lặp hết phần tử nhé.

|  |
| --- |
| # Danh sách mảng  my\_list = [4, 7, 0, 3]    # Iterator  my\_iter = iter(my\_list)    # Lặp  while True:      try:          print(next(my\_iter))      except StopIteration:          break |

### Tự tạo ra Iterators trong Python

Để có thể tự tạo ra những đối tượng Iterators thì trong class bạn phải khai báo hai phương thức \_iter\_\_() và \_\_next\_\_().

* Phương thức \_\_iter\_\_() phải trả về chính đối tượng của lớp đó.
* Phương thức \_\_next\_\_() phải trả về phần tử tiếp theo trong dãy.

**VD**: Tạo một Iterator là danh sách các số chẵn lớn hơn 1.

|  |
| --- |
| class MyNumbers:      def \_\_iter\_\_(self):          self.a = 2 # Số chẵn bắt đầu bằng 2          return self        def \_\_next\_\_(self):          x = self.a          self.a += 2 # Tăng lên 2 đơn vị          return x      mynumber = MyNumbers()  inumber = iter(mynumber)    print(next(inumber))  print(next(inumber))  print(next(inumber))  print(next(inumber))  print(next(inumber)) |

**Hiển thị**



### StopIteration trong Python

Bạn có thể sử dụng StopIteration để để thoát khỏi chương trình lặp.

Xét lại ví dụ ở trên, mình muốn MyNumbers chỉ in các số chẵn lớn hơn 1 và bé hơn 11.

|  |
| --- |
| class MyNumbers:      def \_\_iter\_\_(self):          self.a = 2 # Số chẵn bắt đầu bằng 2          return self        def \_\_next\_\_(self):          x = self.a          self.a += 2 # Tăng lên 2 đơn vị          if (self.a < 11):              return x          else :              StopIteration    mynumber = MyNumbers()  inumber = iter(mynumber)    print(next(inumber))  print(next(inumber))  print(next(inumber))  print(next(inumber))  print(next(inumber)) # None |

Lệnh print cuối cùng nó trả về None vì kết quả đã vượt khỏi số 10.

Trên là cách sử dụng và tạo Iterators trong Python. Chúc bạn thành công nhé!

## GENERATOR TRONG PYTHON

Trong bài này chúng ta sẽ tìm hiểu về Generator trong Python, đây là cách giúp bạn tạo ra một đối tượng iterator cực kì dễ dàng. Bên cạn đó mình cũng phân tích giúp bạn hiểu được sự khác nhau giữa một hàm bình thường và một hàm generator.

**Mục lục**

* [1. Generators trong Python là gì?](https://freetuts.net/generator-trong-python-3494.html#goto-h2-0)
* [2. Cách tạo Generators trong Python](https://freetuts.net/generator-trong-python-3494.html#goto-h2-1)
* [3. Sự khác nhau giữa hàm generator và hàm bình thường](https://freetuts.net/generator-trong-python-3494.html#goto-h2-2)
* [4. Sử dụng vòng lặp trong Generator Python](https://freetuts.net/generator-trong-python-3494.html#goto-h2-3)
* [5. Sử dụng generator expression trong Python](https://freetuts.net/generator-trong-python-3494.html#goto-h2-4)
* [6. Tại sao nên dùng generator trong Python?](https://freetuts.net/generator-trong-python-3494.html#goto-h2-5)

### Generators trong Python là gì?

Generator là cách tạo ra một mô hình lặp [iterator](https://freetuts.net/iterators-trong-python-3492.html) trong Python, bằng cách sử dụng từ khóa **yield** để tạo ra những trình lặp một cách đơn giản và hiệu quả nhất.

Một hàm khi sử dụng yield thì bản thân nó đã tự kế thừa hai phương thức \_\_iter\_\_() và \_\_next\_\_() nên ta có thể sử dụng hàm next() mà không cần phải sử dụng hàm iter() để khởi tạo iterator.

### Cách tạo Generators trong Python

Việc tạo một generator trong Python khá là đơn giản, nó giống như một hàm bình thường, nhưng thay vì sử dụng lệnh return để trả về thì ta sử dụng lệnh [yield](https://freetuts.net/ref/yield-trong-python-dung-de-lam-gi-427.html).

Nếu một function (hàm) có chứa một hoặc nhiều lệnh yield thì nó là một generator. Một hàm có thể có một hoặc nhiều lệnh yield đặt tại nhiều vị trí khác nhau trong hàm.

Sự khác nhau giữa lệnh return và yield là trong khi return sẽ trả về một giá trị và kết thúc hàm, nhưng yield thì sẽ trả về nhưng vẫn giữ lại trạng thái của các biến, sau đó nếu được gọi tiếp thì nó sẽ tiếp tục xử lý ngay tại vị trí tạm dừng đó.

### Sự khác nhau giữa hàm generator và hàm bình thường

Dưới đây là một vài điểm khác nhau giữa hàm generator và hàm bình thường.

* Generator chứa một hoặc nhiều lệnh yield.
* Khi được gọi, hàm generator sẽ trả về một đối tượng iterator nhưng nó không thực thi liền.
* Hai phương thức \_\_iter\_\_() và \_\_next\_\_() được kế thừa tự động, vì vậy bạn có thể sử dụng hàm next() mà không cần dùng hàm iter() để tạo iterator .
* Trạng thái của các biến được lưu trữ lại giữa những lần gọi.
* Cuối cùng khi hàm kết thúc thì StopIteration sẽ bung ra cho lần gọi hàm tiếp theo, đây là lỗi cho thấy iterator đã duyệt đến phần tử cuối cùng.

Hãy xem ví dụ dưới đây để hiểu rõ hơn.

|  |
| --- |
| # HỌC PYTHON TAI FREETUTS.NET  # AUTHOR: CƯỜNG NGUYỄN  def generateNumber():      n = 1      print("Lần gọi thứ nhất trả về n = ", n)      yield n        n += 1      print("Lần gọi thứ hai trả về n = ", n)      yield n        n += 1      print("Lần gọi thứ ba trả về n = ", n)      yield n    # Chương trình chính  num = generateNumber()    # Kiểm tra xe num là gì?  # => Nó là một generate object  # <generator object generateNumber at 0x000001CD0A939510>  print(num)    # Gọi đến generate => trả về yield đầu tiên  # **Hiển thị** 1  print(next(num))    # Gọi đến generate => trả về yield thứ hai  # **Hiển thị** 2  print(next(num))    # Gọi đến generate => trả về yield thứ ba  # **Hiển thị** 3  print(next(num)) |

Như bạn thấy, biến n bên trong hàm đã được nhớ sau mỗi lần gọi.

### Sử dụng vòng lặp trong Generator Python

Ở ví dụ trên mình chỉ giải thích cách hoạt động của generator chứ thực tế không ai làm như vậy.

Chúng ta thường sử dụng vòng lặp để tạo ra những generator.

Hãy xem ví dụ dưới đây, mình sẽ tạo ra một generator các số từ 1 đến 10.

|  |
| --- |
| # Tạo generator  def generateNumber():      for i in range(1, 11):          yield i    # Lặp qua generator  num = generateNumber()  for n in num:      print(n) |

**Hiển thị:**



### Sử dụng generator expression trong Python

Ngoài những cách trên thì bạn có thể tạo ra generator bằng biểu thức expression.

Cách hoạt động của nó giống như [list comprehension](https://freetuts.net/list-comprehension-trong-python-3496.html), chỉ có điều một bên sử dụng dấu ngoặc vuông, một bên sử dụng dấu ngoặc nhọn.

Cách này chỉ phù hợp với những trường hợp có list data sẵn, và bạn muốn tạo ra một generator dựa trên list đó.

|  |
| --- |
| # Tạo list  my\_list = [1, 3, 6, 10]    # Sử dụng expression để tạo generator  generator = (x\*\*2 for x in my\_list)    for item in generator:      print(item) |

**Hiển thị:**

|  |
| --- |
|  |



Ta có thể viết lại ví dụ trên bằng cách sử dụng vòng lặp như sau:

|  |
| --- |
| # Tạo list  my\_list = [1, 3, 6, 10]    # Sử dụng vòng lặp  def newList(my\_list):      for item in my\_list:          yield item    generator = newList(my\_list)    for item in generator:      print(item) |

### Tại sao nên dùng generator trong Python?

Lý do đơn giản và thuyết phục nhất là generator rất đơn giản và dễ thực hiện.

Generator triển khai dễ dàng và ngắn gọn hơn nhiều so với việc sử dụng trình lặp của iterator.

**VD**: Xây dựng một iterator tính lũy thừa của 2.

|  |
| --- |
| class LuyThua2():      def \_\_init\_\_(self, max = 0):          # Thuộc tính lưu số lũy thừa hiện tại          self.n = 0          # Thuộc tính lưu số lũy thừa tối đa          self.max = max        def \_\_iter\_\_(self):          return self        def \_\_next\_\_(self):          if self.n > self.max:              raise StopIteration          else:              result = 2 \*\* self.n              self.n += 1              return result      l = LuyThua2(5)    print(next(l)) # n = 0  print(next(l)) # n = 1  print(next(l)) # n = 2  print(next(l)) # n = 3  print(next(l)) # n = 4  print(next(l)) # n = 5    print(next(l)) # Trả về lỗi StopIteration |

Nhưng nếu viết bằng Generator thì chương trình rất ngắn gọn như sau:

|  |
| --- |
| def LuyThua2(max = 0):      n = 0      while n <= max:          yield 2 \*\* n          n += 1    l = LuyThua2(5)  print(next(l)) # n = 1  print(next(l)) # n = 2  print(next(l)) # n = 3  print(next(l)) # n = 4  print(next(l)) # n = 5  print(next(l)) # n = 6  print(next(l)) # Lỗi StopIteration |

Ngoài ra, generator thân thiện hơn bởi nó chỉ tạo ra một mục tại một thời điểm gọi.

Trên là những chia sẻ cơ bản về cách sử dụng generator trong Python.

## LIST COMPREHENSION TRONG PYTHON

Trong bài này chúng ta sẽ tìm hiểu cách sử dụng list comprehension trong Python, đây là cách tạo ra một list mới dựa vào một list có sẵn dựa vào một điều kiện nào đó.

**Mục lục**

* [1. List comprehension là gì?](https://freetuts.net/list-comprehension-trong-python-3496.html#goto-h2-0)
* [2. Ví dụ về list comprehension trong Python](https://freetuts.net/list-comprehension-trong-python-3496.html#goto-h2-1)

### List comprehension là gì?

List comprehension là cách tạo ra một list dữ liệu mới dựa vào dữ liệu cũ theo một điều kiện nào đó.

Vậy, công thức chung của list comprehension là:

|  |
| --- |
| newlist = [expression for item in iterable if condition == True] |

Trong đó:

* expression chính là biểu thức trả về
* iterable là tập dữ liệu iterator
* condition là điều kiện trả về, phần tử nào có condition = True thì biểu thức expression tương ứng sẽ được trả về.

Giả sử mình có một mảng các số như sau:

|  |
| --- |
| numbers = [1,3,86,4,54,54,45,65,6,57,67,87,82] |

Bây giờ mình muốn tạo một mảng mới từ mảng này, với điều kiện là chỉ lấy các số chẵn mà thôi.

**Cách thứ nhất**: Sử dụng vòng lặp bình thường.

|  |
| --- |
| numbers = [1,3,86,4,54,54,45,65,6,57,67,87,82]    new\_numbers = []  for n in numbers:      if n % 2 == 0:          new\_numbers.append(n)    # **Hiển thị** [86, 4, 54, 54, 6, 82]  print(new\_numbers) |

**Cách thứ hai**: Sử dụng list comprehension.

|  |
| --- |
| numbers = [1,3,86,4,54,54,45,65,6,57,67,87,82]    new\_numbers = [x for x in numbers if x % 2 == 0]    # **Hiển thị** [86, 4, 54, 54, 6, 82]  print(new\_numbers) |

Bạn có thể không cần phải truyền condition.

**VD**: Tạo mảng mới với giá trị gấp đôi mảng cũ.

|  |
| --- |
| numbers = [1,3,5,7,9]    new\_numbers = [x\*2 for x in numbers]    # **Hiển thị** [2, 6, 10, 14, 18]  print(new\_numbers) |

### Ví dụ về list comprehension trong Python

Bây giờ mình sẽ làm thêm một vài ví dụ để bạn dễ hiểu hơn nhé.

**VD**: Tạo danh sách các số nguyên tố từ một danh sách có sẵn.

Để giải bài này thì ta cần tạo một hàm kiểm tra số nguyên tố trước, sau đó sẽ sử dụng nó trong phần **condition**.

|  |
| --- |
| # Hàm kiểm tra số nguyên tố  def laSNT(n):      if n < 2: return False      elif n == 2: return True      elif n % 2 == 0: return False      else:          for i in range(3, n, 2):              if n % i == 0: return False      return True    # Dữ liệu  data = [1, 2, 3, 5, 45, 46, 5, 6, 76, 9, 11]    # Lấy danh sách các số nguyên tố trong data  data\_snt = [x for x in data if laSNT(x)]  print(data\_snt) |

Đây chính là cách sử dụng list comprehension để tạo ra danh sách mảng mới trong Python.

## HÀM CLOSURE TRONG PYTHON

Trong bài này chúng ta sẽ tìm hiểu hàm closure trong Python, cách tạo ra một hàm closure, các ví dụ cách sử dụng, cũng như lý do tại sao nên dùng closure Python.

Trước khi tìm hiểu về closure thì bạn phải hiểu hai khái niệm, đó là hàm lồng nhau (nested function) và biến không cục bộ (non-local variable). Nào, hãy bắt đầu ngay nhé.

**Mục lục**

* [1. Nested function là gì?](https://freetuts.net/closure-trong-python-3497.html#goto-h2-0)
* [2. Cách khai báo closure function trong Python](https://freetuts.net/closure-trong-python-3497.html#goto-h2-1)
* [3. Tại sao nên sử dụng closure function trong Python](https://freetuts.net/closure-trong-python-3497.html#goto-h2-2)

### Nested function là gì?

Nested function Python hay còn gọi là **hàm lồng nhau** trong Python.

Một hàm được định nghĩa bên trong một hàm khác thì ta gọi đó là nested function. Các hàm nested function có thể sử dụng biến trong phạm vi hàm cha của nó nhưng không được thay đổi giá trị của chúng.

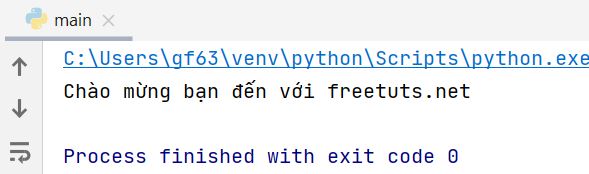
Non-local variable (biến không cục bộ) có thể được tạm hiểu là một biến được khai báo trong một hàm A, và nó cũng có thể được sử dụng bên trong những hàm nested function khai báo bên trong hàm A đó.

Bạn **không được thay đổi giá trị**của biến không cục bộ.

Ví dụ dưới đây thì hàm **printer** là một nested function và chúng truy cập vào một biến không cục bộ tên là **msg**.

|  |
| --- |
| def sayHi(msg):        # Phạm vi code bên ngoài        def printer():          # Đây là nested function          print(msg)        printer()    # Thực hiện hàm cha  # Trong thâm hàm cha gọi đến một nested fucntion  # có tên là printer  sayHi("Chào mừng bạn đến với freetuts.net") |

Kết quả như sau:



Như bạn thấy, nested function tên là **printer**đã sử dụng biến của hàm cha của nó tên là **msg**. Biến **msg**nó có phạm vi bên trong hàm **sayHi**, và vì **printer**là hàm được khai báo trong hàm **sayHi**nên nó có thể sử dụng biến **msg**.

Dưới đây là một ví dụ mình đa cố tình thay đổi giá trị cho biến không cục bộ, đây là một cú pháp sai.

|  |
| --- |
| def createNum():      n = 1        def plusOne() :          n += 1 # Sai vì đã thay đổi giá trị cho biến n |

### Cách khai báo closure function trong Python

Xét lại ví dụ ở trên, chuyện gì sẽ xảy ra nếu ta đặt lệnh gọi đến hàm printer() bên trong lệnh **return**? Điều này có nghĩa là ta đang trả về một hàm printer chứ không phải là gọi thực thi nó.

Lúc này việc sử dụng hàm **sayHi**sẽ qua hai công đoạn:

* Thứ nhất: Gọi hàm sayHi, hàm này sẽ trả về hàm printer
* Thứ hai: Gọi đến hàm printer bằng cách gọi đến biến lưu trữ giá trị trả về từ hàm sayHi.

|  |
| --- |
| def sayHi(msg):      # Phạm vi code bên ngoài        def printer():          # Đây là nested function          print(msg)        return printer    # Gọi đến hàm sayHi  s = sayHi("Welcome to freetuts.net")    # In thử xem nó là cái gì  print(s)  # **Hiển thị** <function sayHi.<locals>.printer ...>    # Vì nó là một hàm printer nên ta gọi thêm một lần nữa  s() |

Bạn có nhận thấy điều bất thường không? Khi gọi đến hàm **sayHi**thì biến **msg**đã được gán giá trị là "Welcome to freetuts.net", và sau đó biến đó **vẫn được lưu trữ giá trị** cho đến khi mình gọi đến hàm printer bằng lệnh s().

Cách viết như thế này ta gọi là **closure function**. Vậy closure function hàm mà giá trị trả về một hàm khác.

### Tại sao nên sử dụng closure function trong Python

Từ ví dụ trên thì ta có kết luận rằng các biến không cục bộ trong closure function sẽ **được ghi nhớ**, vì vậy ta có thể tận dụng nó để làm những công việc sau đây.

**Đầu tiên** là có thể thay thế trong lập trình hướng đối tượng, thay vì phải tạo class dài dòng thì bạn có thể dùng closure.

|  |
| --- |
| # Chương trình sẽ tính n ^ x  def soMu(n):      def cal(x):          return n \*\* x      return cal    # Gọi đến hàm soMu  # Hàm này trả về hàm cal  mu = soMu(2)    # Tính 2 mũ 5  print(mu(5))    # Tính 2 mũ 8  print(mu(8)) |

Ta có thể viết bằng class như sau:

|  |
| --- |
| class soMu:      def \_\_init\_\_(self, n):          self.n = n        def cal(self, x):          return self.n \*\* x    mu = soMu(2)    print(mu.cal(5)) # 2 ^ 5  print(mu.cal(8)) # 2 ^ 8 |

Trên là cách sử dụng closure trong Python, hy vọng qua bài viết này sẽ giúp bạn hiểu khái niệm closure python là gì, cách khai báo nó như thế nào, cũng như nắm vững được khái niệm **nested function** và **non-local variable**.

## HIGHER ORDER FUNCTIONS TRONG PYTHON

Trong bài này chúng ta sẽ tìm hiểu Higher Order Functions trong Python, qua đó mình cũng nói qua cách sử dụng Decorators, một cách sử dụng khá phổ biến của Higher Order Functions.

Trước khi tìm hiểu thì mình muốn nhắc các bạn rằng bản thân hàm (function) trong Python là một object, được kế thừa tự động một số phương thức như \_\_call\_\_, cũng như bạn có thể gán nó vào một biến và gọi lại sau đó.

**Mục lục**

* [1. Higher order function trong Python là gì?](https://freetuts.net/higher-order-functions-3500.html#goto-h2-0)
* [2. Các ví dụ về higher order function trong Python](https://freetuts.net/higher-order-functions-3500.html#goto-h2-1)
  + [Functions là objects](https://freetuts.net/higher-order-functions-3500.html#goto-h3-0)
  + [Function là một tham số](https://freetuts.net/higher-order-functions-3500.html#goto-h3-1)
  + [Return về một function](https://freetuts.net/higher-order-functions-3500.html#goto-h3-2)
* [3. Điểm qua một chút về Decorator](https://freetuts.net/higher-order-functions-3500.html#goto-h2-2)

### Higher order function trong Python là gì?

Trong Python, một hàm được gọi là higher order function (hàm bậc cao) nếu hàm đó có tham số truyền vào là một hàm khác, và bản thân hàm này cũng có thể return về một function (ta gọi đó là [*closure function*](https://freetuts.net/closure-trong-python-3497.html)).

Sau đây là một số đặc điểm:

* Một function là một instance của một object
* Bạn có thể lưu trữ function trong một biến
* Bạn có thể truyền một function vào tham số của một function khác
* Bạn có thể return về một function
* Bạn có thể lưu trữ function trong các kiểu dữ liệu dạng cấu trúc như list, hash tables.

Ta hãy thử **print**xem **type**của một function là gì nhé.

|  |
| --- |
| def abc():      pass    t = abc()  print(type(t)) |

**Hiển thị**

|  |
| --- |
| <class 'NoneType'> |

Như vậy biến t chính là một class.

### Các ví dụ về higher order function trong Python

Bây giờ ta sẽ làm một vài ví dụ về những đặc điểm của funciton cấp cao trong Python nhé.

#### Functions là objects

Function là object nên bạn có thể lưu trữ nó trong một biến, sau đó gọi thông qua biến đó.

|  |
| --- |
| def sayHi(msg):      print(msg)    # Gán function vào biến  say = sayHi    # Gọi function  say("freetuts.net") |

**Hiển thị**



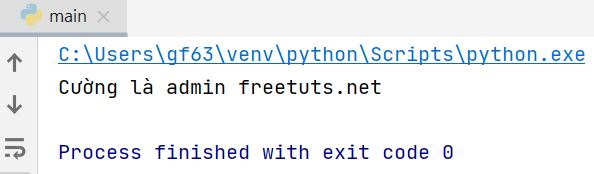
Như trong ví dụ này thì bạn thấy mình đã lưu trữ function **sayHi**vào biến **say**, sau đó gọi đến function bằng cú pháp **say()**.

#### Function là một tham số

Ta có thể truyền function vào như là một tham số, bên trong hàm cha sẽ gọi để kích hoạt function tham số đó.

|  |
| --- |
| # Hàm cha có tham số getMsg là một function  def sayHi(getMsg, msg):      # Gọi đến hàm getMsg, và truyền tham số cho nó      print(getMsg(msg))    # Hàm getMsg có một tham số truyền vào  def getMsg(msg):      return msg + " freetuts.net"    # Chương trình chính  msg = getMsg  sayHi(msg, "Cường là admin") |

**Hiển thị**

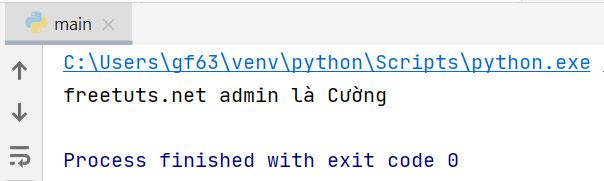


#### Return về một function

Dạng này ta gọi là closure function nhé các bạn. Nếu bạn muốn tìm hiểu sâu hơn thì hãy đọc bài đó trong series Python nâng cao này nhé.

|  |
| --- |
| def sayHi(domain):      def getMsg(msg):          return domain + msg      return getMsg    hi = sayHi("freetuts.net")  print(hi(" admin là Cường")) |

**Hiển thị**



### Điểm qua một chút về Decorator

Decorator là kỹ thuật dùng higher order function để trang trí thêm nội dung cho một function có sẵn.

**VD**:

|  |
| --- |
| def enclose(fun):        def wrapper():            print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")          fun()          print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")        return wrapper    def myfun():      print("myfun")    enc = enclose(myfun)  enc() |

Trong ví dụ này mình đã sử dụng hàm enclose là một decorator, bởi vì mình dùng nó để gọi đến hàm fun(), và đặt hàm func() bên trong hai hàng star.

Như vậy khi chạy thì hai hàng star được trang trí thêm cho hàm func (in nội dung my fun).

Xem bài viết [decorator trong Python](https://freetuts.net/decorator-trong-python-3501.html) để hiểu thêm nhé.

Trên là bài hướng dẫn cách sử dụng hàm bậc cao "higher order function trong Python", hy vọng qua bài này sẽ giúp bạn hiểu được cách truyền tham số vào hàm là một tham số khác. Chúc bạn thành công.

## DECORATOR TRONG PYTHON

Trong bài này chúng ta sẽ tìm hiểu cách dùng Decorator trong Python, dựa vào cấu trúc của nested function ta có thể tạo ra những decorator một cách dễ dàng.

Các hàm trong Python là first-class (lớp đầu tiên), điều này có nghĩa là các hàm sẽ có trạng thái bình đẳng với những đối tượng object khác trong Python. Điều này có nghĩa là các hàm có thể được gán cho biến, được truyền vào như một tham số, được lưu trữ trong các collection.

**Mục lục**

* [1. Python decorator là gì?](https://freetuts.net/decorator-trong-python-3501.html#goto-h2-0)
* [2 Tạo decorator trong Pyhton bằng dấu @](https://freetuts.net/decorator-trong-python-3501.html#goto-h2-1)
* [3. Python decorator với hàm có tham số](https://freetuts.net/decorator-trong-python-3501.html#goto-h2-2)
* [4. Multi decorator trong Python](https://freetuts.net/decorator-trong-python-3501.html#goto-h2-3)
* [5. Decorator với class trong Python](https://freetuts.net/decorator-trong-python-3501.html#goto-h2-4)

### Python decorator là gì?

Python decorator là cách thay đổi hành vi (behavior) của một object [callable](https://freetuts.net/ref/callable-trong-python-428.html) mà không cần can thiệp và chỉnh sửa object đó. Decorator là một hàm có nhiệm vụ bao bọc một hàm khác (ta tạm gọi là hàm B), hàm B sẽ được gọi bên trong thân của hàm decorator, và đương nhiên lúc này hàm decorator có thể bổ sung những đoạn code phía trên và phía dưới vị trí gọi hàm B.

**Hãy lấy một ví dụ cho dễ hiểu**: Giả sử bạn viết một hàm in ra các số từ 1 đến 10 như sau:

|  |
| --- |
| def showNumbers():      for i in range( 1, 11):          print(i, end=' ')    showNumbers() |

Nhưng hôm sau bạn muốn in thêm một dòng dấu sao ở phía trên và phía dưới của các số đó nữa thì làm thế nào? Cách đơn giản nhất là sửa lại hàm đó.

|  |
| --- |
| def showNumbers():      print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')      for i in range( 1, 11):          print(i, end=' ')      print('')      print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')    showNumbers() |

Nhưng cách làm này sẽ không hiệu quả nếu function showNumbers mình đã sử dụng ở rất nhiều nơi, nếu sửa trực tiếp như vậy sẽ làm ảnh hưởng đến những vị trí đó.

Để giải quyết thì mình sẽ kết hợp với [higher order function](https://freetuts.net/higher-order-functions-3500.html) để tạo ra decorator.

|  |
| --- |
| # Đây là hàm decorator  def decoratorFunc(func):      # Đây là hàm decorator gọi,      # ta gọi là extend decorated function      def addStar():          print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")          func()          print('')          print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")        return addStar    def showNumbers():      for i in range( 1, 11):          print(i, end=' ')      add\_star = decoratorFunc(showNumbers)  add\_star() |

Trong ví dụ này thì hàm decoratorFunc là một **decorator**, bên trong được gọi đến hàm mở rộng addStar dùng để trang trí thêm mấu dấu sao.

### Tạo decorator trong Pyhton bằng dấu @

Python cho phép bạn sử dụng dấu @ để đánh dấu một phương thức sẽ được trang trí (decorator) bằng một phương thức khác.

Quay lại ví dụ trên thì mình sẽ viết lại chương trình decorator như sau:

|  |
| --- |
| def decoratorFunc(func):      def addStar():          print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")          func()          print('')          print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")        return addStar    @decoratorFunc  def showNumbers():      for i in range( 1, 11):          print(i, end=' ')    # Gọi kiểu này đơn giản hơn  showNumbers() |

**Hiển thị:**



Kể từ bây giờ chúng ta nên sử dụng cách này nhé các bạn, bởi trông chuyên nghiệp và ảo diệu hơn rất nhiều, một người không rành về Python sẽ không thể hiểu là bạn đang code cái gì :3

### Python decorator với hàm có tham số

Trường hợp hàm showNumbers có tham số thì cách viết như thế nào nhỉ?

Giả sử mình fix lại hàm đó với một tham số **n** truyền vào, và nội dung sẽ in ra dãy số từ 1 đến **n**.

|  |
| --- |
| def decoratorFunc(func):      def addStar(n):          print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")          func(n)          print('')          print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")        return addStar    @decoratorFunc  def showNumbers(n):      for i in range( 1, n + 1):          print(i, end=' ')    # Gọi có tham số truyền vào  showNumbers(20) |

Các bạn chú ý nhé, đoạn code trên mình có truyền vào tham số n = 20 nên có hơi chút khác biệt so với các ví dụ ở trên.

### Multi decorator trong Python

Bạn có thể áp dụng nhiều decorator cho một function.

**VD**: Giả sử mình có một hàm

|  |
| --- |
| def addStrongTag(func):      def add(msg):          print('<strong>')          func(msg)          print('</strong>')      return add    def addSpanTag(func):      def add(msg):          print('<span>')          func(msg)          print('</span>')      return add    @addStrongTag  @addSpanTag  def showMessage(msg):      print(msg)    showMessage("Freetuts.net") |

**Hiển thị**



### Decorator với class trong Python

Bạn có thể tạo decorator bằng một class với điều kiện trong class đó phải khai báo phương thức \_\_call\_\_ nhé.

|  |
| --- |
| class addTag():        def \_\_init\_\_(self, func):          self.func = func        def \_\_call\_\_(self, msg):          print('<strong>')          self.func(msg)          print('</strong>')    @addTag  def showMessage(msg):      print(msg)    showMessage("freetuts.net") |

**Hiển thị**

|  |
| --- |
| freetuts.net |

**Kết luận**: Qua bài này ta thấy việc sử dụng decorator trong Python thực sự rất hữu ích. Có nhiều trường hợp trong thực tế có thể áp dụng nó như: Chương trình quản lý đăng nhập, quản lý việc hiển thị nội dung ra bên ngoài một trang web ...

Cuối cùng, chúc bạn nắm vững được kỹ thuật decorator này nhé.

## XỬ LÝ FILE VÀ THƯ MỤC TRONG PYTHON

Trong bài viết này ta sẽ tìm hiểu về các hoạt động với file trong Python như: mở file, đọc file, ghi file, đóng file, ... và các phương thức xử lý file khác.

**Mục lục**

* [1. Sơ lược về xử lý file trong Python](https://freetuts.net/xu-ly-file-trong-python-1714.html#goto-h2-0)
* [2. Mở file với open() trong Python?](https://freetuts.net/xu-ly-file-trong-python-1714.html#goto-h2-1)
* [3. Đóng kế nối với close() trong Python](https://freetuts.net/xu-ly-file-trong-python-1714.html#goto-h2-2)
* [4. Ghi file trong Python](https://freetuts.net/xu-ly-file-trong-python-1714.html#goto-h2-3)
* [5. Đọc dữ liệu của file trong Python](https://freetuts.net/xu-ly-file-trong-python-1714.html#goto-h2-4)
* [6. Lời kết](https://freetuts.net/xu-ly-file-trong-python-1714.html#goto-h2-5)

### Sơ lược về xử lý file trong Python

File là một tệp được người dùng máy tính sử dụng để lưu trữ dữ liệu của họ, nó sẽ được lưu trữ trên ổ đĩa vĩnh viễn trừ khi ổ đĩa bị hỏng hoặc bạn xóa nó.

Khi làm việc với file thì chúng ta sẽ có những hành động chính như sau:

* Mở, tạo file
* Thay đổi dữ liệu của file
* Xóa file
* Di chuyển file
* Copy file
* ...

Đối với một phiên làm việc với file trong Python thì sẽ có ba bước chính như sau:

* Mở file, nếu file cần mở không tồn tại thì tạo file mới.
* Thay đổi dữ liệu của file
* Đóng file

Bây giờ chúng ta sẽ tìm hiểu chi tiết hơn từng thao tác nhé.

### Mở file với open() trong Python?

Python cung cấp cho chúng ta một hàm có tên là open(), hàm này có cú pháp như sau:

|  |
| --- |
| f = open("location\_file", "mode") |

Trong đó location\_file là đường dẫn trỏ đến file cần mở, còn mode là chế độ mở file. Bảng dưới đây là danh sách các chế độ mở file thường gặp.

|  |  |
| --- | --- |
| **Python File Modes** | |
| **Mode** | **Description** |
| 'r' | Mở file ở chế độ đọc (mặc định) |
| 'w' | Mở file ở chế độ ghi, tạo file mới nếu file không tồn tại, làm sạch file nếu nó đã tồn tại |
| 'x' | Mở file ở chế độ tạo mới, nếu file đã tồn tại thì sẽ bị báo lỗi |
| 'a' | Mở file ở chế độ ghi tiếp theo ở phân cuối của file, tạo mới nếu nó không tồn tại |
| 't' | Mở file ở chế độ dữ liệu kiểu text (mặc định) |
| 'b' | Mở file ở chế độ dữ liệu kiểu nhị phần binary |
| '+' | Mở file ở cả hai chế độ đọc và ghi |

**VD**:

|  |
| --- |
| f = open("test.txt")      # equivalent to 'r' or 'rt'  f = open("test.txt",'w')  # write in text mode  f = open("img.bmp",'r+b') # read and write in binary mode |

Như vậy bạn có thể sử dụng một hoặc nhiều chế độ trong cùng một phiên mở file.

Không giống như các ngôn ngữ khác, ký tự 'a' không phải là số 97 trong bảng mã ASCII (hoặc mã hóa tương đương khác), nó sẽ phụ thuộc vào nền tảng mà bạn đang sư dụng. Trong Window thì là 'cp1252' nhưng trong Linux thì 'utf-8', vì vậy, chúng ta không nên sử dụng mã hóa mặc định mà chỉ rõ trong lúc mở file.

**VD**:

|  |
| --- |
| f = open("test.txt",mode = 'r',encoding = 'utf-8') |

### Đóng kế nối với close() trong Python

Khi ta mở một file và xử lý xong thì nên đóng file đó lại để đảm bảo tính an toàn, và để làm điều này thì ta sử dụng hàm close().

**VD**:

|  |
| --- |
| f = open("test.txt",encoding = 'utf-8')  # perform file operations  f.close( |

Phương thức này sẽ không an toàn trong một số trường hợp, nó sẽ không tự đóng file nếu trong quá trình mở và ghi bị lỗi. Để khắc phục vấn đề này thì ta sử dụng từ khóa with, nó sẽ đảm bảo răng file sẽ luôn được đóng khi lệnh kết thúc.

**VD**:

|  |
| --- |
| with open("test.txt",encoding = 'utf-8') as f:     # perform file operations |

Và ta cũng không cần phải gọi hàm close() vì nó tự động thực hiện ngầm.

### Ghi file trong Python

Để ghi vào một file bằng Python thì bắt buộc chúng ta cần mở nó ở chế độ ghi 'w', nối thêm 'a' hoặc tạo mới 'x'. Chúng ta cần cẩn thận với chế độ 'w' vì nó sẽ ghi đè lên tệp nếu nó đã tồn tại, tất cả dữ liệu trước đó bị xóa.

**VD**:

|  |
| --- |
| with open("test.txt",'w',encoding = 'utf-8') as f:     f.write("my first file\n")     f.write("This file\n\n")     f.write("contains three lines\n") |

Chương trình này sẽ tạo một file tên là test.txt nếu nó chưa tồn tại, còn nếu đã tồn tại thì dữ liệu trước đây sẽ bị xóa hết.

### Đọc dữ liệu của file trong Python

Để đọc dữ liệu của file thì chúng ta phải mở nó ở chế độ reading (r), sau đó có thể sử dụng hàm read(size) để để đọc dữ liệu của file, nếu bạn không truyền size thì nó sẽ đọc từ đầu đến cuối file.

**VD**:

|  |
| --- |
| >>> f = open("test.txt",'r',encoding = 'utf-8')  >>> f.read(4)    # read the first 4 data  'This'    >>> f.read(4)    # read the next 4 data  ' is '    >>> f.read()     # read in the rest till end of file  'my first file\nThis file\ncontains three lines\n'    >>> f.read()  # further reading returns empty sting  '' |

Làm thế nào để Python biết được là xuống dòng? Nó sẽ nhận biết thông qua ký tự thoát \n.

Để lấy số dòng hiện tại đang đọc thì ta dùng hàm tell(), và dùng hàm seek() để di chuyển con trỏ đọc tới một vị trí nào đó.

**VD**:

|  |
| --- |
| >>> f.tell()    # get the current file position  56    >>> f.seek(0)   # bring file cursor to initial position  0    >>> print(f.read())  # read the entire file  This is my first file  This file  contains three lines |

Ngoài ra để đọc từng dòng thì ta có thể kết hơp với vòng lặp như ví dụ dưới đây.

|  |
| --- |
| >>> for line in f:  ...     print(line, end = '')  ...  This is my first file  This file  contains three lines |

Mỗi lần lặp chính là một dòng dữ liệu của file.

Để đọc một dòng duy nhất thì ta dùng hàm readline(), hàm này sẽ đọc dòng mới và sẽ ngưng khi kết thúc dòng đó.

**VD**:

|  |
| --- |
| >>> f.readline()  'This is my first file\n'    >>> f.readline()  'This file\n'    >>> f.readline()  'contains three lines\n'    >>> f.readline()  '' |

Để đọc tất cả các dòng và trả về dạng list thì ta dùng hàm readlines().

|  |
| --- |
| >>> f.readlines()  ['This is my first file\n', 'This file\n', 'contains three lines\n'] |

### Lời kết

Trên là những thao tác cơ bản xử lý file trong Python, những thao tác này rất đơn giản nhưng lại rất hữu ích, hy vọng bài viết sẽ giúp được bạn trong quá trình học Python.

## SƠ LƯỢC DATE/TIME TRONG PYTHON

Trong bài này chúng ta sẽ tìm hiểu cách xử lý thời gian (date - time) trong Python như: Cách import object date, lấy thời gian hiện tai, định dạng thời gian, ...

Trong các ứng dụng thực tế, có những lúc chúng ta cần phải làm việc với ngày tháng và thời gian. Ví dụ cần viết một ứng dụng đặt lịch hẹn bằng ngôn ngữ Python thì ta phải biết cách lấy thời gian trên hệ thống để xử lý.

Trong python, date không phải là kiểu dữ liệu, nhưng chúng ta có thể làm việc với các đối tượng ngày bằng cách import mô-đun có tên là datetime, time và calendar.

**Mục lục**

* [1. Mốc thời gian nhỏ nhất trong Python](https://freetuts.net/so-luoc-datetime-trong-python-2061.html#goto-h2-0)
* [2. Cách lấy thời gian hiện tại - localtime()](https://freetuts.net/so-luoc-datetime-trong-python-2061.html#goto-h2-1)
* [3. Time tuple là gì?](https://freetuts.net/so-luoc-datetime-trong-python-2061.html#goto-h2-2)
* [4. Hàm asctime() thay đổi format time](https://freetuts.net/so-luoc-datetime-trong-python-2061.html#goto-h2-3)
* [5. Hàm sleep() trong Python](https://freetuts.net/so-luoc-datetime-trong-python-2061.html#goto-h2-4)
* [6. Module datetime trong Python](https://freetuts.net/so-luoc-datetime-trong-python-2061.html#goto-h2-5)
  + [1. Tạo date object](https://freetuts.net/so-luoc-datetime-trong-python-2061.html#goto-h3-0)
  + [2. So sánh 2 ngày khác nhau](https://freetuts.net/so-luoc-datetime-trong-python-2061.html#goto-h3-1)
* [7. Calendar module trong Python](https://freetuts.net/so-luoc-datetime-trong-python-2061.html#goto-h2-6)

### Mốc thời gian nhỏ nhất trong Python

Cũng như hầu hết các ngôn ngữ lập trình khác, Python lấy ngày**1/1/1970** làm mốc nhỏ nhất. Hàm time() sẽ trả về tổng số mili giây đã trôi qua tính từ ngày 1/1/1970.

Xem ví dụ dưới đây để hiểu rõ hơn:

|  |
| --- |
| import time;    #In ra tổng số mili giây kể từ ngày 1/1/1970    print(time.time()) |

Kết quả trả về tính đến thời điểm hiện tại:

|  |
| --- |
| 1545124470.9151752 |

### Cách lấy thời gian hiện tại - localtime()

Để lấy thời gian hiện tại trên hệ thống thì ta sử dụng hàm localtime(). Cách sử dụng rất đơn giản, bạn chỉ cần import module time và gọi hàm này một cách bình thường.

Xem ví dụ sau:

|  |
| --- |
| import time;    #returns a time tuple    print(time.localtime(time.time())) |

**Hiển thị:** Kết quả nó trả về là một tuple.

|  |
| --- |
| time.struct\_time(tm\_year=2018, tm\_mon=12, tm\_mday=18, tm\_hour=15, tm\_min=1,  tm\_sec=32, tm\_wday=1, tm\_yday=352, tm\_isdst=0) |

### Time tuple là gì?

Time tuple là bộ các đơn vị thời gian, gồm 9 thông số như sau:

|  |  |  |
| --- | --- | --- |
| **Index** | **Attribute** | **Values** |
| 0 | Year | 4 số nguyên(for example 2018) |
| 1 | Month | 1 đến 12 |
| 2 | Day | 1 đến 31 |
| 3 | Hour | 0 đến 23 |
| 4 | Minute | 0 đến 59 |
| 5 | Second | 0 đến 60 |
| 6 | Day of weak | 0 đến 6 |
| 7 | Day of year | 1 đến 366 |
| 8 | Daylight savings | -1, 0, 1 , or -1 |

### Hàm asctime() thay đổi format time

Bạn có thể thay đổi định dạng thời gian bằng cách sử dụng hàm asctime() năm trong **module time**. Nó sẽ trả về thời gian được định dạng của đối tượng time đang sử dụng hàm này.

|  |
| --- |
| import time;    #Trả về định dạng time    print(time.asctime(time.localtime(time.time()))) |

**Hiển thị**

|  |
| --- |
| Tue Dec 18 15:31:39 2019 |

### Hàm sleep() trong Python

Hàm sleep() trong Python được dùng để dừng thực thi chương trình trong một khoảng thời gian nhất định. Số thời gian bị hoãn phụ thuộc vào giá trị mà bạn truyền vào hàm này.

Đơn vị của tham số thời gian đầu vào tính bằng giây, có thể số nguyên hoặt số thực.

**VD**: Tạo vòng lặp, mỗi lần lặp cho nghỉ 1 giây.

|  |
| --- |
| import time  for i in range(0,5):      print(i)      #Mỗi lần lặp sẽ nghỉ 1 giây, sau đó in kết quả ra màn hình      time.sleep(1) |

### Module datetime trong Python

**Module datetime** cho phép ta tạo đối tượng thời gian và có thể tùy chỉnh theo ý mình. Đối tượng này được sử dụng rất nhiều trong thực tế. Ví dụ bạn làm lịch hẹn giờ thì chắc chắn phải dùng module date để lấy thời gian hiện tai, datetime để lấy thời gian hẹn, sau đó so sánh và xử lý hiển thị kết quả.

Tương tự, ta phải import datetime thì mới sử dụng các phương thức - hàm ở trong module này được.

**VD**: In ra thời gian hiện tại

|  |
| --- |
| import datetime;    #Trả về object time hiện tại    print(datetime.datetime.now()) |

**Hiển thị**

|  |
| --- |
| 2018-12-18 16:16:45.462778 |

#### Tạo date object

Ta có thể tạo một date object bằng cách truyền giá trị ngày vào ba tham số của hàm datetime.

Như ví dụ dưới đây mình truyền vào 10/12/2018.

|  |
| --- |
| import datetime;    print(datetime.datetime(2018,12,10)) |

Hoặc truyền nhiều hơn 3 tham số:

|  |
| --- |
| print(datetime.datetime(2018,12,10,14,15,10)) |

#### 2. So sánh 2 ngày khác nhau

Để so sánh hai ngày hơn kém nhau thì ta có thể sử dụng các toán tử so sánh (>, >=, <, <=, =).

**VD:**

|  |
| --- |
| from datetime import datetime as dt  #So sánh thời gian. Nếu thời gian nằm trong khoảng 8AM và 4PM, thì in ra màn hình thời gian việc, ngược lai là thời gian nghỉ ngơi  if dt(dt.now().year,dt.now().month,dt.now().day,8)<dt.now()<dt(dt.now().year,dt.now().month,dt.now().day,16):      print("Thời gian lam việc ....")  else:      print("Thời gian nghỉ ngơi") |

### Calendar module trong Python

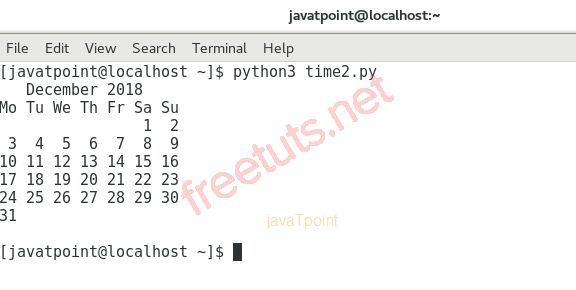
Python cung cấp một module về xử lý lịch (calendar) chứa các phương thức khác nhau để làm việc với lịch.

Hãy xem xét ví dụ sau để in lịch của tháng cuối năm 2018.

**Ví dụ**

|  |
| --- |
| import calendar;  cal = calendar.month(2018,12)  #In lịch năm 2018  print(cal) |

**Hiển thị**



Nếu bạn muốn in ra lịch của cả năm thì làm như sau:

|  |
| --- |
| import calendar    #In lịch năm 2019  calendar.prcal(2019) |

