TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NÔI

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



**ĐỀ TÀI**

**TÌM HIỂU NGÔN NGỮ PYTHON**

GIẢNG VIÊN HƯỚNG DẪN: THẦY THÂN QUANG KHOÁT

SINH VIÊN THỰC HIỆN: ĐẶNG XUÂN TRƯỜNG 20154018

# Lời nói đầu

Công nghiệp 4.0 đang bùng nổ với xu hướng chính trong ngành công nghệ thông tin là lĩnh vực trí tuệ nhân tạo. Theo đó các ngôn ngữ lập trình cũng đang phát triển hơn nưa để phù hợp với xu thế. Trong đó không thể không nhắc tới ngôn ngữ lập trình Python.

Python là ngôn ngữ lập trình thông dịch, bậc cao đơn giản dễ hiểu giúp các lập trình viên dễ dàng tiếp cận với trí tuệ nhân tạo. Python đang không ngừng phát triển và đang ngày càng trở thành ngôn ngữ thông dụng và ứng dụng rộng rãi trong IT. Python được tích hợp để chạy được trên nhiều hệ điều hành khác nhau như: Windows, MS-DOS, MAC OS,OS/2, Linux và các hệ điều hành họ Unix. Có lẽ vì vậy mà Python được ứng dụng trong nhiều lĩnh vực trong công nghệ thông tin như: lập trình Web, bigdata,xử lí toán học, an toàn bảo mật, xử lí ảnh, xử lí ngôn ngữ tự nhiên,...

Nhận thấy được sự hữu ích của Python nên em chọn đề tài cho môn học là: “Tìm hiểu ngôn ngữ lập trình Python”. Bài tìm hiểu chỉ giới hạn trên những tài liệu tiếng việt thu thập được trên mạng Internet, do đó bài viết khó tránh những thiếu xót mong được thầy chỉ dẫn thêm.

# Mục lục

[Lời nói đầu 2](#_Toc509488222)

[Mục lục 3](#_Toc509488223)

[I. GIới thiệu chung 5](#_Toc509488224)

[I.1. Các khái niệm 6](#_Toc509488225)

[I.2. Lịch sử hình thành và phát triển 8](#_Toc509488226)

[I.3. Đặc điểm 9](#_Toc509488227)

[Tính dễ học, dễ đọc 10](#_Toc509488228)

[Các bản hiện thực 11](#_Toc509488229)

[Khả năng mở rộng 11](#_Toc509488230)

[Trình thông dịch 12](#_Toc509488231)

[Module hóa 12](#_Toc509488232)

[Đa năng 13](#_Toc509488233)

[Đa biến hóa 15](#_Toc509488234)

[II. Python cơ bản 16](#_Toc509488235)

[II.1. Cú pháp cơ bản và tham số dòng lệnh 16](#_Toc509488236)

[Cú pháp cơ bản 16](#_Toc509488237)

[Tham số dòng lệnh 17](#_Toc509488238)

[II.2. Các kiểu biến và toán tử 18](#_Toc509488239)

[Các kiểu biến 18](#_Toc509488240)

[Toán tử 19](#_Toc509488241)

[II.3. Cấu trúc điều khiển, vòng lặp 20](#_Toc509488242)

[II.4. Cấu trúc dữ liệu 27](#_Toc509488243)

[Kiểu dữ liệu NUMBER 27](#_Toc509488244)

[Kiểu dữ liệu String 29](#_Toc509488245)

[LIST 38](#_Toc509488246)

[Tuple 44](#_Toc509488247)

[Dictionary 49](#_Toc509488248)

[II.5. Hàm, module, file 54](#_Toc509488249)

[Date and time 54](#_Toc509488250)

[Hàm 59](#_Toc509488251)

[File I/O 72](#_Toc509488252)

[II.6. Xử lí ngoại lệ 84](#_Toc509488253)

[III. Python nâng cao 92](#_Toc509488254)

[III.1, Hướng đối tượng 92](#_Toc509488255)

[III.2. Truy cập Databases 105](#_Toc509488256)

[III.3. Gửi email 114](#_Toc509488257)

[III.4. Lập trình mạng 120](#_Toc509488258)

[III.5. Xử lí đa luồng 124](#_Toc509488259)

[IV. Các đặc điểm mới trong Python 3.x 133](#_Toc509488260)

[Kết luận 135](#_Toc509488261)

[Tài liệu tham khảo 136](#_Toc509488262)

# I. GIới thiệu chung

**Python** là một [ngôn ngữ lập trình](https://vi.wikipedia.org/wiki/Ng%C3%B4n_ng%E1%BB%AF_l%E1%BA%ADp_tr%C3%ACnh) [thông dịch](https://vi.wikipedia.org/wiki/Tr%C3%ACnh_th%C3%B4ng_d%E1%BB%8Bch) do [Guido van Rossum](https://vi.wikipedia.org/w/index.php?title=Guido_van_Rossum&action=edit&redlink=1) tạo ra năm [1990](https://vi.wikipedia.org/wiki/1990). Python hoàn toàn [tạo kiểu động](https://vi.wikipedia.org/w/index.php?title=T%E1%BA%A1o_ki%E1%BB%83u_%C4%91%E1%BB%99ng&action=edit&redlink=1) và dùng cơ chế [cấp phát bộ nhớ tự động](https://vi.wikipedia.org/wiki/Qu%E1%BA%A3n_l%C3%BD_b%E1%BB%99_nh%E1%BB%9B); do vậy nó tương tự như [Perl](https://vi.wikipedia.org/wiki/Perl), [Ruby](https://vi.wikipedia.org/wiki/Ruby_(ng%C3%B4n_ng%E1%BB%AF_l%E1%BA%ADp_tr%C3%ACnh)), [Scheme](https://vi.wikipedia.org/wiki/Scheme), [Smalltalk](https://vi.wikipedia.org/wiki/Smalltalk), và [Tcl](https://vi.wikipedia.org/wiki/Tcl). Python được phát triển trong một dự án [mã mở](http://www.opensource.org/), do tổ chức phi lợi nhuận [Python Software Foundation](http://www.python.org/psf/about/) quản lý.

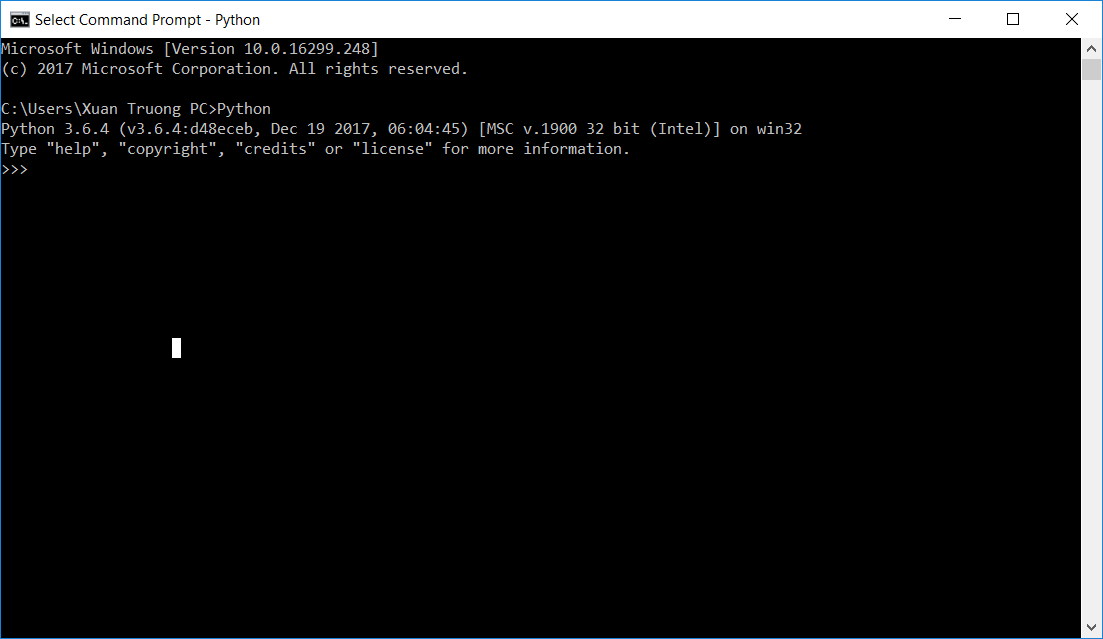
Theo đánh giá của [Eric S. Raymond](https://vi.wikipedia.org/w/index.php?title=Eric_Raymond&action=edit&redlink=1), Python là ngôn ngữ có hình thức rất sáng sủa, cấu trúc rõ ràng, thuận tiện cho người mới học lập trình. Cấu trúc của Python còn cho phép người sử dụng viết mã lệnh với số lần gõ phím tối thiểu, như nhận định của chính [Guido van Rossum](https://vi.wikipedia.org/w/index.php?title=Guido_van_Rossum&action=edit&redlink=1) trong một bài phỏng vấn ông.

Ban đầu, Python được phát triển để chạy trên nền [Unix](https://vi.wikipedia.org/wiki/Unix). Nhưng rồi theo thời gian, nó đã "bành trướng" sang mọi [hệ điều hành](https://vi.wikipedia.org/wiki/H%E1%BB%87_%C4%91i%E1%BB%81u_h%C3%A0nh) từ [MS-DOS](https://vi.wikipedia.org/wiki/MS-DOS) đến [Mac OS](https://vi.wikipedia.org/wiki/Mac_OS), OS/2, [Windows](https://vi.wikipedia.org/wiki/Microsoft_Windows), [Linux](https://vi.wikipedia.org/wiki/Linux) và các hệ điều hành khác thuộc họ Unix. Mặc dù sự phát triển của Python có sự đóng góp của rất nhiều cá nhân, nhưng Guido van Rossum hiện nay vẫn là tác giả chủ yếu của Python. Ông giữ vai trò chủ chốt trong việc quyết định hướng phát triển của Python.

Python là một ngôn ngữ lập trình bậc cao, thông dịch, hướng đối tượng, đa mục đích và cũng là một ngôn ngữ lập trình động.

Cú pháp của Python là khá dễ dàng để học và ngôn ngữ này cũng mạnh mẽ và linh hoạt không kém các ngôn ngữ khác trong việc phát triển các ứng dụng. Python hỗ trợ mẫu đa lập trình, bao gồm lập trình hướng đối tượng, lập trình hàm và mệnh lệnh hoặc là các phong cách lập trình theo thủ tục.

Để bắt đâu học và luyện tập vơi ngôn ngữ Python ta cần tài và cài đặt trình thông dịch Python. Trong bài tìm hiểu này em xin được sử dụng Python trên nền Windows.



## I.1. Các khái niệm

Để thuận tiện cho những người mới học em xin được đưa ra một số định nghĩa cơ bản.

1) trình thông dịch: biên dịch và chạy chương trình thành từng đoạn

2) trình biên dịch: biên dịch cả chương trình rồi mới chạy

3) module: là một đoạn code trong chương trình có thể chay độc lập. Các chương trình thường được module hóa để có thể áp dụng những kĩ thuật xử lí song song.

4) Hướng đối tượng; là một [mẫu hình lập trình](https://vi.wikipedia.org/wiki/M%E1%BA%ABu_h%C3%ACnh_l%E1%BA%ADp_tr%C3%ACnh) dựa trên khái niệm "[công nghệ đối tượng](https://vi.wikipedia.org/wiki/%C4%90%E1%BB%91i_t%C6%B0%E1%BB%A3ng_(khoa_h%E1%BB%8Dc_m%C3%A1y_t%C3%ADnh))", mà trong đó, đối tượng chứa đựng các [dữ liệu](https://vi.wikipedia.org/wiki/D%E1%BB%AF_li%E1%BB%87u), trên các [trường](https://vi.wikipedia.org/wiki/Tr%C6%B0%E1%BB%9Dng_(khoa_h%E1%BB%8Dc_m%C3%A1y_t%C3%ADnh)), thường được gọi là các *thuộc tính;* và mã nguồn, được tổ chức thành các [*phương thức*](https://vi.wikipedia.org/wiki/Ph%C6%B0%C6%A1ng_th%E1%BB%A9c_(l%E1%BA%ADp_tr%C3%ACnh_m%C3%A1y_t%C3%ADnh))*.* Phương thức giúp cho đối tượng có thể truy xuất và hiệu chỉnh các trường dữ liệu của đối tượng khác, mà đối tượng hiện tại có tương tác (đối tượng được hỗ trợ các phương thức "[this](https://vi.wikipedia.org/w/index.php?title=This_(l%E1%BA%ADp_tr%C3%ACnh_m%C3%A1y_t%C3%ADnh)&action=edit&redlink=1)" hoặc "self"). Trong lập trình hướng đối tượng, chương trình máy tính được thiết kế bằng cách tách nó ra khỏi phạm vi các đối tượng tương tác với nhau.

5) Các khái niệm liên quan đến hướng đối tượng:

**Lớp**: Một nguyên mẫu được định nghĩa bởi người dùng cho một đối tượng mà định nghĩa một tập hợp các thuộc tính mà xác định rõ bất kỳ đối tượng nào của lớp đó. Các thuộc tính là các thành viên dữ liệu (các biến class và biến instance) và các phương thức được truy cập thông qua toán tử dot (dấu chấm .).

Biến **class**: Đây là một biến được chia sẻ bởi tất cả các instance (sự thể hiện) của một lớp. Các biến class được định nghĩa bên trong một lớp nhưng ở bên ngoài bất cứ phương thức nào của lớp đó. Biến class không được sử dụng thường xuyên như biến instance.

**Thành viên dữ liệu**: Là một biến class hoặc biến instance mà giữ dữ liệu được liên kết với một lớp và các đối tượng của nó.

**Nạp chồng hàm (overloading):** Là phép gán của nhiều hơn một hành vi tới một hàm cụ thể. Hoạt động được thực hiện là đa dạng do các kiểu của các đối tượng hoặc tham số liên quan.

Biến **instance:** Là một biến được định nghĩa bên trong một phương thức và chỉ thuộc sở hữu của instance hiện tại của một lớp đó.

**Tính kế thừa**: Là việc truyền các đặc trưng của một lớp cho các lớp khác mà kế thừa từ lớp ban đầu.

**Instance**: Là một đối tượng riêng của một lớp nào đó. Một đối tượng obj mà thuộc một lớp Circle là một instance (sự thể hiện) của lớp Circle.

**Trình khởi tạo**: Là trình tạo một sự thể hiện của một lớp.

**Phương thức**: Một loại hàm đặc biệt mà được định nghĩa trong một phần định nghĩa lớp.

**Đối tượng**: Một instance duy nhất của một cấu trúc dữ liệu mà được định nghĩa bởi lớp của nó. Một đối tượng gồm các thành viên dữ liệu (biến class và biến instance) và các phương thức.

**Nạp chồng toán tử**: Là phép gán của nhiều hơn một hàm cho một toán tử cụ thể.

## I.2. Lịch sử hình thành và phát triển

**Python** đã được hình thành vào cuối những năm 1980, và việc thực hiện nó vào tháng 12 năm 1989 bởi **Guido van Rossum** tại**Centrum Wiskunde & Informatica (CWI)** ở **Hà Lan** như là một kế thừa cho ngôn ngữ **ABC** (tự lấy cảm hứng từ SETL) có khả năng xử lý ngoại lệ và giao tiếp với Hệ điều hành Amoeba. **Van Rossum** là tác giả chính của **Python**, và vai trò trung tâm của ông trong việc quyết định hướng phát triển của **Python**.

*Về nguồn gốc của****Python****,****Van Rossum****đã viết vào năm 1996:*

*“Hơn sáu năm trước, vào tháng 12 năm 1989, tôi đã tìm kiếm một dự án lập trình "sở thích" mà nó đã chiếm đóng tâm trí tôi trong suốt tuần lễ Giáng sinh. Văn phòng của tôi ... sẽ đóng cửa, nhưng tôi đã có một máy tính ở nhà, và không có nhiều thứ khác trên tay. Tôi quyết định viết một bộ thông dịch (interprester) cho ngôn ngữ kịch bản mới mà tôi đã từng nghĩ đến: một hậu duệ của****ABC****có thể hấp dẫn các hacker****Unix/C****. Tôi đã chọn****Python****như là một tiêu đề làm việc cho dự án.”*

Sự phát triển Python đến nay có thể chia làm các giai đoạn:

* **Python 1**: bao gồm các bản phát hành 1.x. Giai đoạn này, kéo dài từ đầu đến cuối [thập niên 1990](https://vi.wikipedia.org/wiki/Th%E1%BA%ADp_ni%C3%AAn_1990). Từ năm 1990 đến 1995, Guido làm việc tại [CWI](https://vi.wikipedia.org/w/index.php?title=CWI&action=edit&redlink=1)(*Centrum voor Wiskunde en Informatica* - Trung tâm Toán-Tin học tại [Amsterdam](https://vi.wikipedia.org/wiki/Amsterdam), [Hà Lan](https://vi.wikipedia.org/wiki/H%C3%A0_Lan)). Vì vậy, các phiên bản Python đầu tiên đều do CWI phát hành. Phiên bản cuối cùng phát hành tại CWI là 1.2.
  + Vào năm 1995, Guido chuyển sang [CNRI](https://vi.wikipedia.org/w/index.php?title=CNRI&action=edit&redlink=1) (*Corporation for National Research Initiatives*) ở [Reston](https://vi.wikipedia.org/w/index.php?title=Reston&action=edit&redlink=1), [Virginia](https://vi.wikipedia.org/wiki/Virginia). Tại đây, ông phát hành một số phiên bản khác. Python 1.6 là phiên bản cuối cùng phát hành tại CNRI.
  + Sau bản phát hành 1.6, Guido rời bỏ CNRI để làm việc với các lập trình viên chuyên viết phần mềm thương mại. Tại đây, ông có ý tưởng sử dụng Python với các phần mềm tuân theo chuẩn [GPL](https://vi.wikipedia.org/wiki/Gi%E1%BA%A5y_ph%C3%A9p_C%C3%B4ng_c%E1%BB%99ng_GNU). Sau đó, CNRI và [FSF](https://vi.wikipedia.org/wiki/Qu%E1%BB%B9_Ph%E1%BA%A7n_m%E1%BB%81m_T%E1%BB%B1_do) (*Free Software Foundation* - Tổ chức phần mềm tự do) đã cùng nhau hợp tác để làm bản quyền Python phù hợp với GPL. Cùng năm đó, Guido được nhận Giải thưởng FSF vì Sự phát triển Phần mềm tự do (*Award for the Advancement of Free Software*).
  + Phiên bản 1.6.1 ra đời sau đó là phiên bản đầu tiên tuân theo bản quyền GPL. Tuy nhiên, bản này hoàn toàn giống bản 1.6, trừ một số sửa lỗi cần thiết.
* **Python 2**: vào năm 2000, Guido và nhóm phát triển Python dời đến [BeOpen.com](https://vi.wikipedia.org/w/index.php?title=BeOpen.com&action=edit&redlink=1) và thành lập [BeOpen PythonLabs team](https://vi.wikipedia.org/w/index.php?title=BeOpen_PythonLabs_team&action=edit&redlink=1). Phiên bản Python 2.0 được phát hành tại đây. Sau khi phát hành Python 2.0, Guido và các thành viên PythonLabs gia nhập [Digital Creations](https://vi.wikipedia.org/w/index.php?title=Digital_Creations&action=edit&redlink=1).
  + Python 2.1 ra đời kế thừa từ Python 1.6.1 và Python 2.0. Bản quyền của phiên bản này được đổi thành Python Software Foundation License. Từ thời điểm này trở đi, Python thuộc sở hữu của [Python Software Foundation](https://vi.wikipedia.org/w/index.php?title=Python_Software_Foundation&action=edit&redlink=1) (PSF), một tổ chức phi lợi nhuận được thành lập theo mẫu [Apache Software Foundation](https://vi.wikipedia.org/w/index.php?title=Apache_Software_Foundation&action=edit&redlink=1).
* **Python 3**, còn gọi là **Python 3000** hoặc **Py3K**: Dòng 3.x sẽ không hoàn toàn tương thích với dòng 2.x, tuy vậy có công cụ hỗ trợ chuyển đổi từ các phiên bản 2.x sang 3.x. Nguyên tắc chủ đạo để phát triển Python 3.x là "bỏ cách làm việc cũ nhằm hạn chế trùng lặp về mặt chức năng của Python". Trong PEP (*Python Enhancement Proposal*) có mô tả chi tiết các thay đổi trong Python . Các đặc điểm mới của Python 3.0 sẽ được trình bày phần cuối bài này.

## I.3. Đặc điểm

Triết lý căn bản của ngôn ngữ Python được trình bày trong tài liệu*The*[*Zen of Python*](https://en.wikipedia.org/wiki/Zen_of_Python) (*PEP 20*):

Beautiful is better than ugly,

Explicit is better than implicit.

Simple is better than complex.

Complex is better is than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren’t special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity,

Refuse the temptation to quess.

There should be one – and preferably only one- obvious at first unless you’re Dutch.

Now is better than never.

Although never is often better than right now.

If the implementation is hard to explain, it’s a bad idea.

If the implementation is easy to explain, it may be a good idea

Namespaces are one honking great idea- let’s do more of those!

Và có thể tóm gọn là:

* Đẹp đẽ tốt hơn xấu xí
* Minh bạch tốt hơn che đậy
* Đơn giản tốt hơn phức tạp
* Phức tạp tốt hơn rắc rối
* Dễ đọc

### Tính dễ học, dễ đọc

Tính dễ học, dễ đọc được thể hiện qua các yếu tố:

+) Hệ thống từ khóa: tăng cường sử dụng cá từ kháo bằng tiếng Anh, hạn chế cá kí hiệu và cấu trúc cú pháp so với các ngôn ngữ khác; Python là một ngôn ngữ phân biết kiểu chữ Hoa và chữ thường; Như C/C++, các từ khóa của Python đều ở dạng chữ thường

+) Khối lệnh: Trong ác ngôn ngữ khác, khối lệnh thường được đánh dấu bằng cặp ký hiệu hoặc từ khóa. Ví dụ: trong C/C++, cặp ngoặc nhon {} được dùng để bao bọc một khối lệnh. Với Python để tạo một khối lệnh chỉ cần thụt các câu lệnh trong khối vào sâu hơn (về phái bên phải) so với các câu lệnh của khối lệnh cha chứa nó.

### Các bản hiện thực

Python được viết từ những ngôn ngữ khác, tạo ra những bản hiện thực khác nhau. Bản hiện thực Python chính, còn gọi là CPython, được viết bằng C, và được phân phối kèm một thư viện chuẩn lớn được viết hỗn hợp bằng C và Python. CPython có thể chạy trên nhiều nền và khả chuyển trên nhiều nền khác. Dưới đây là các nền trên đó, CPython có thể chạy.

Các hệ điều hành họ Unix: AIX, Darwin, FreeBSD, Mac OS X, NetBSD, Linux, OpenBSD, Solaris,…

Các hệ điều hành dành cho máy desktop: Amiga, AROS, BeOS, Mac OS 9, Microsoft Windows, OS/2, RISC OS.

Các hệ thống nhúng và các hệ đặc biệt: GP2X, Máy ảo Java, Nokia 770 Internet Tablet, Palm OS, PlayStation 2, PlayStation Portable, Psion, QNX, Sharp Zaurus, Symbian OS, Windows CE/Pocket PC, Xbox/XBMC, VxWorks.

Các hệ máy tính lớn và các hệ khác: AS/400, OS/390, Plan 9 from Bell Labs, VMS, z/OS.

Ngoài CPython, còn có hai hiện thực Python khác: Jython cho môi trường Java và IronPython cho môi trường .NET và Mono.

### Khả năng mở rộng

Python có thể được mở rộng: nếu ta biết sử dụng C, ta có thể dễ dàng viết và tích hợp vào Python nhiều hàm tùy theo nhu cầu. Các hàm này sẽ trở thành hàm xây dựng sẵn (built-in) của Python. Ta cũng có thể mở rộng chức năng của trình thông dịch, hoặc liên kết các chương trình Python với các thư viện chỉ ở dạng nhị phân (như các thư viện đồ họa do nhà sản xuất thiết bị cung cấp). Hơn thế nữa, ta cũng có thể liên kết trình thông dịch của Python với các ứng dụng viết từ C và sử dụng nó như là một mở rộng hoặc một ngôn ngữ dòng lệnh phụ trợ cho ứng dụng đó.

### Trình thông dịch

Python là một ngôn ngữ lập trình dạng thông dịch, do đó có ưu điểm tiết kiệm thời gian phát triển ứng dụng vì không cần phải thực hiện biên dịch và liên kết. Trình thông dịch có thể được sử dụng để chạy file script, hoặc cũng có thể được sử dụng theo cách tương tác. Ở chế độ tương tác, trình thông dịch Python tương tự shell của các hệ điều hành họ Unix, tại đó, ta có thể nhập vào từng biểu thức rồi gõ Enter, và kết quả thực thi sẽ được hiển thị ngay lập tức. Đặc điểm này rất hữu ích cho người mới học, giúp họ nghiên cứu tính năng của ngôn ngữ; hoặc để các lập trình viên chạy thử mã lệnh trong suốt quá trình phát triển phần mềm. Ngoài ra, cũng có thể tận dụng đặc điểm này để thực hiện các phép tính như với máy tính bỏ túi.

### Module hóa

Python cho phép chia chương trình thành các module để có thể sử dụng lại trong các chương trình khác. Nó cũng cung cấp sẵn một tập hợp các modules chuẩn mà lập trình viên có thể sử dụng lại trong chương trình của họ. Các module này cung cấp nhiều chức năng hữu ích, như các hàm truy xuất tập tin, các lời gọi hệ thống, trợ giúp lập trình mạng (socket),…

### Đa năng

Python là một ngôn ngữ lập trình đơn giản nhưng rất hiệu quả.

+ So với Unix shell, Python hỗ trợ các chương trình lớn hơn và cung cấp nhiều cấu trúc hơn.

+ So với C, Python cung cấp nhiều cơ chế kiểm tra lỗi hơn. Nó cũng có sẵn nhiều kiểu dữ liệu cấp cao, ví dụ như các mảng (array) linh hoạt và từ điển (dictionary) mà ta sẽ phải mất nhiều thời gian nếu viết bằng C.

Python là một ngôn ngữ lập trình cấp cao có thể đáp ứng phần lớn yêu cầu của lập trình viên:

+ Python thích hợp với các chương trình lớn hơn cả AWK và Perl.

+ Python được sử dụng để lập trình Web. Nó có thể được sử dụng như một ngôn ngữ kịch bản.

+ Python được thiết kế để có thể nhúng và phục vụ như một ngôn ngữ kịch bản để tuỳ biến và mở rộng các ứng dụng lớn hơn.

+ Python được tích hợp sẵn nhiều công cụ và có một thư viện chuẩn phong phú, Python cho phép người dùng dễ dàng tạo ra các dịch vụ Web, sử dụng các thành phần COM hay CORBA, hỗ trợ các loại định dạng dữ liệu Internet như email, HTML, XML và các ngôn ngữ đánh dấu khác. Python cũng được cung cấp các thư viện xử lý các giao thức Internet thông dụng như HTTP, FTP,…

+ Python có khả năng giao tiếp đến hầu hết các loại cơ sở dữ liệu, có khả năng xử lý văn bản, tài liệu hiệu quả, và có thể làm việc tốt với các công nghệ Web khác.

+ Python đặc biệt hiệu quả trong lập trình tính toán khoa học nhờ các công cụ Python Imaging Library, pyVTK, MayaVi 3D Visualization Toolkits, Numeric Python, ScientificPython,…

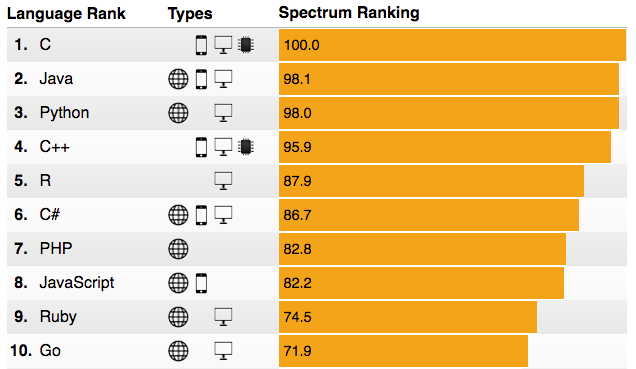
+ Python có thể được sử dụng để phát triển các ứng dụng desktop. Lập trình viên có thể dùng wxPython, PyQt, PyGtk để phát triển các ứng dụng giao diện đồ họa (GUI) chất lượng cao. Python còn hỗ trợ các nền tảng phát triển phần mềm khác như MFC, Carbon, Delphi, X11, Motif, Tk, Fox, FLTK, …

+ Python cũng có sẵn một unit testing framework để tạo ra các bộ test (test suites).

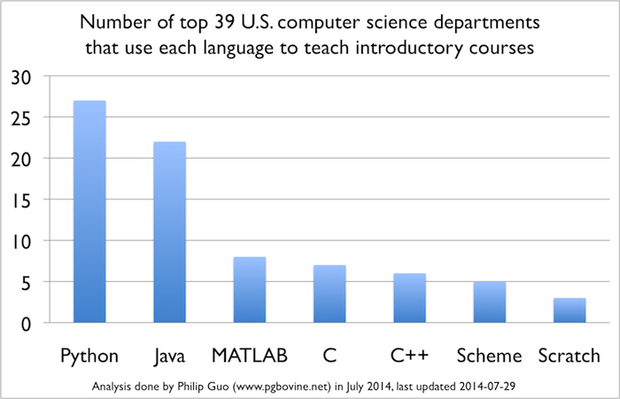
### Đa biến hóa

Python là một ngôn ngữ đa biến hóa (multiple paradigms). Có nghĩa là, thay vì ép buộc mọi người phải sử dụng duy nhất một phương pháp lập trình, Python lại cho phép sử dụng nhiều phương pháp lập trình khác nhau: hướng đối tượng, có cấu trúc, chức năng, hoặc chỉ hướng đến một khía cạnh. Python kiểu kiểu động và sử dụng bộ thu gom rác để quản lý bộ nhớ. Một đặc điểm quan trọng nữa của Python là giải pháp tên động, kết nối tên biến và tên phương thức lại với nhau trong suốt thực thi của chương trình.

Có lẽ vì những đặc điểm đó mà Python trở thành ngôn ngữ thông dụng có tính ứng dụng cao và có tỉ lệ cao được giảng dạy ở nhiều trường đại học.



Mức độ thông dụng của các ngôn ngữ lập trình



Tỉ lệ các ngôn ngữ lập trình được giảng dạy tại các trường đại học

# II. Python cơ bản

## II.1. Cú pháp cơ bản và tham số dòng lệnh

### Cú pháp cơ bản

**Định danh**: một định danh trong python là một tên được sử dụng đẻ nhận diện một biến, một hàm, một lớp, một đối tượng. Một định danh bắt đầu với một chữ cái từ A đến Z hoặc từ a đến z hoặc một dấu gạch dưới them sau bởi 0 hoặc nhiều kí tự, dấu gạch dưới hoặc các chữ số từ 0 đến 9. Python không hỗ trợ các kí tự đặc biệt như @.$,% bên trong các định danh. Hơn nữa, các định danh trong Python phân biệt chữ hóa chữ thường, ví dụ 2 định danh “dhbkhn” và “DHBKHN” là 2 định danh khác nhau.

Một sô quy tắc đặt tên:

+ Một định danh là một dãy ký tự hoặc chữ số

+ không có ký tự đặc biệt nào được sử dụng trừ dấu gạch dưới. Ký tự đầu tiên có thể là chữ cái, dấu gạch dưới, nhưng không được sử dụng chữ số làm ký tự đầu tiên.

+ từ khóa không nên được sử dụng như là một tên định danh

+ tên lớp bắt đầu với một chữ cái hoa. Tất cả định danh khác bắt đầu với một chữ cái thường.

+ bắt đầu một định danh với một dấu gạch dưới đơn chỉ rằng định danh đó là private

+ bắt đầu một định danh với hai dấu gạch dưới chỉ rằng định danh đó thực sự là private

+ Nếu định danh cũng kết thúc với hai dâu gạch dưới, thì định danh này là một tên đặc biết được định nghĩa bởi ngôn ngữ.

**Từ khóa**: and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, with, yield.

**Khối lệnh**: các dòng lệnh đều thụt vào so với đầu dòng một khoảng như nhau sẽ tạo thành các khối lệnh.

**Chú thích**: dấu # bắt đầu một chú thích đơn dòng và dấu cặp dấu “““ ””” dùng để ghi dòng chú thích đa dòng.

### Tham số dòng lệnh

Python cung cấp getopt Module giúp bạn phân tích cú pháp các tùy chọn và tham số dòng lệnh. sys Module trong Python cung cấp sự truy cập tới bất kỳ tham số dòng lệnh nào thông qua sys.argv. Phục vụ hai mục đích: **sys.argv** là danh sách các tham số dòng lệnh; **len(sys.argv)** là số tham số dòng lệnh.

Parse các tham số dòng lệnh trong Python: Python cung cấp getopt Module giúp bạn phân tích cú pháp các tùy chọn và tham số dòng lệnh. Module này cung cấp hai hàm và một exception để kích hoạt việc phân tích cú pháp các tham số dòng lệnh.

Phương thức getopt.getopt trong Python: Phương thức này phân tích cú pháp danh sách tham số và các tùy chọn tham số dòng lệnh Cú pháp là: **getopt.getopt(args, option, [long\_option])**

Chi tiết về tham số:

+ args: Đây là danh sách tham số để được phân tích.

+ option: Đây là chuỗi các tùy chọn mà script muốn để nhận ra. Với các tùy chọn mà yêu cầu một tham số thì nên được theo sau bởi một dấu hai chấm (:).

+ long\_option: Đây là tham số tùy ý và nếu được xác định, phải là một danh sách các chuỗi là tên các tùy chọn dài, mà được hỗ trợ. Với các tùy chọn dài yêu cầu một tham số thì nên được theo sau bởi một dấu bằng (=). Để chỉ chấp nhận các tùy chọn dài, các tùy chọn nên là một chuỗi trống.

Phương thức này trả về trả trị bao gồm hai phần tử: phần tử đầu là một danh sách các cặp (option, value). Phần tử thứ hai là danh sách các tham số chương trình.

Cặp option-value được trả về có một dấu gạch nối ngắn ở trước (ví dụ -x) là tùy chọn ngắn, có hai dấu gạch nối là tùy chọn dài (ví dụ --long-option).

Ngoại lệ **getopt.GetoptError trong Python**: Đây là một exception và nó được tạo khi thấy một tùy chọn không được nhận ra trong danh sách tham số hoặc khi một tùy chọn cần một tham số mà không cung cấp tham số nào.

Tham số cho exception là một chuỗi chỉ nguyên nhân gây ra lỗi. Các thuộc tính msg và opt cung cấp thông điệp lỗi và tùy chọn có liên quan.

## II.2. Các kiểu biến và toán tử

### Các kiểu biến

Biến là không gì khác ngoài các vị trị bộ nhớ được dành riêng để lưu trữ dữ liệu. Một khi một biến đã được lưu trữ, nghĩa là một khoảng không gian đã được cấp phát trong bộ nhớ đó.

Dựa trên kiểu dữ liệu của một biến, trình thông dịch cấp phát bộ nhớ và quyết định những gì có thể được lưu trữ trong khu nhớ dành riêng đó. Vì thế, bằng việc gán các kiểu dữ liệu khác nhau cho các biến, bạn có thể lưu trữ số nguyên, thập phân hoặc ký tự trong các biến này.

Python cho phép đồng thời gán giá trị đơn vào các biến

Trong Python có 5 kiểu dữ liệu chính: Number, String, List, Tuple, Dictionary; có 2 giá trị logic là True và False.

Chuyển đổi kiểu trong Python

|  |  |
| --- | --- |
| Hàm | Miêu tả |
| int(x,[base]) | Chuyển x thành một số nguyên, tham số base xác định cơ sở nếu x là một chuỗi |
| long(x,[base]) | Chuyển x thành một số nguyên, tham số base xác định cơ sở nếu x là một chuỗi |
| float(x) | Chuyển x thành một số thực |
| complex(real,imag) | Chuyển x thành số phức |
| str(x) | Chuyển x thành chuỗi |
| repr(x) | Chuyển đối tượng x thành một chuỗi biểu thức |
| eval(str) | Ước lượng một chuỗi và trả về 1 đối tượng |
| tuple(s) | Chuyển đổi s thành 1 Tuple |
| set(s) | Chuyển s thành 1 Set |
| list(s) | Chuyển s thành 1 List |
| dict(s) | Tạo một Dictionary và s là 1 dãy tuple của cặp (key, value) |
| frozenset(s) | Chuyển s thành 1 Fronzen Set |
| chr(x) | Chuyển x thành 1 ki tự |
| unichr(x) | Chuyển thành kí tự unicode |
| ord(x) | Chuyển đổi một ký tự đơn thành giá trị nguyên của nó |
| hex(x) | Chuyển 1 số nguyên thành 1 chuỗi thập luc phân |
| oct(x) | Chuyển 1 số nguyên thành 1 chuỗi bát phân |

### Toán tử

Toán tử số học: +, -, \*, // (chia lấy nguyên), / (chia), % (chia lấy dư), \*\* (lũy thừa)

Toán tử quan hệ: =, <, >, <=, >=, ==, != hoặc <>

Toán tử gán: =

Toán tử logic: and, or, not

Toán tử so sanh bit: | (hoặc), & (và), ^ (xor), << (dịch trái), >> (dịch phải), ~ (đảo bit).

Toán tử membership kiểm tra xem đối tượng có nằm trong dãy hay ko: in, not in.

Toán tử identify so sánh vị trí ô nhớ của 2 đối tượng: is, not is.

Thứ tự ưu tiên: bảng sắp xếp thứ thự ưu tiên từ sao xuống thấp

|  |  |
| --- | --- |
| Toán tử | Miêu tả |
| \*\* | Lũy thừa |
| ~ + - | Phần bù, cộng trừ 1 ngôi |
| \* , /, %, // | Nhân, chia, lấy dư, lấy nguyên |
| + ,- | Cộng trừ |
| >> , << | Dịch phải, dịch trái |
| & | Và bit |
| ^, | | Xor bít, hoặc bit |
| <= , < , >, >= | Toán tử so sánh |
| <>, ==, != | Toán tử so sánh bằng |
| is, is not | Toán tử identify |
| In, not in | Toán tử membership |
| Not or and | Toán tử logic |

## II.3. Cấu trúc điều khiển, vòng lặp

Ngôn ngữ lập trình Python coi các giá trị khác null và khác 0 là true, và coi các giá trị là null hoặc 0 là false.

**Câu lệnh if**

Lệnh if trong Python là giống như trong ngôn ngữ C. Lệnh này được sử dụng để kiểm tra một điều kiện, nếu điều kiện là true thì lệnh của khối if sẽ được thực thi, nếu không nó sẽ bị bỏ qua.

Cú pháp của lệnh if là:

**if bieu\_thuc:**

**cac\_lenh**

Ví dụ cho lệnh if:

a=20

if (a==20):

print ("gia tri bien a la 20")

Kết quả cho ra màn hình: gia tri bien a la 20



**Câu lệnh if ... elif ... else**

Một lệnh else có thể được sử dụng kết hợp với lệnh if. Một lệnh else chứa khối code mà thực thi nếu biểu thức điều kiện trong lệnh if được ước lượng là 0 hoặc một giá trị false. Lệnh else là lệnh tùy ý và chỉ có duy nhất một lệnh else sau lệnh if.

Cú pháp của lệnh if...else là:

**if bieu\_thuc:**

**cac\_lenh**

**else:**

**cac\_lenh**

Dưới đây là ví dụ minh họa lệnh if...else trong Python:

a=201

if (a==20):

print ("gia tri bien a la 20")

else:

print ("gia tri bien a khong la 20")

Kết quả là: gia tri bien a khong la 20



Lệnh elif cho phép bạn kiểm tra nhiều điều kiện và thực thi khối code ngay khi một trong các điều kiện được ước lượng là true. Cũng giống như lệnh else, lệnh elif là tùy ý. Tuy nhiên, không giống else mà chỉ có một lệnh được theo sau if, thì bạn có thể sử dụng nhiều elif theo sau if.

Cú pháp của lệnh elif là:

**if bieu\_thuc1:**

**cac\_lenh**

**elif bieu\_thuc2:**

**cac\_lenh**

**elif bieu\_thuc3:**

**cac\_lenh**

**else:**

**cac\_lenh**

Python không cung cấp các lệnh switch hoặc case như trong các ngôn ngữ lập trình khác, tuy nhiên bạn có thể sử dụng các lệnh if…elif để thực hiện vai trò như của switch hoặc case như trong ví dụ trên.

Dưới đây là ví dụ của lệnh elif trong Python:

a=100

if a==100:

print ("gia tri cua a la 100")

elif a==111:

print ("gia tri cua a la 111")

else:

print ("gia tri cua a khong phai la 100 hoac 111")

Kết quả là:



Đôi khi có một tình huống là khi ta muốn kiểm tra thêm một điều kiện khác sau khi một điều kiện đã được ước lượng là true. Trong tình huống như vậy, bạn có thể sử dụng các lệnh if lồng nhau trong Python.

Trong cấu trúc các lệnh if lồng nhau, bạn có thể có cấu trúc if…elif…else bên trong cấu trúc if…elif…else khác.

Cú pháp của cấu trúc lồng các lệnh if như sau:

**if bieu\_thuc1:**

**cac\_lenh**

**if bieu\_thuc2:**

**cac\_lenh**

**elif bieu\_thuc3:**

**cac\_lenh**

**else**

**cac\_lenh**

**elif bieu\_thuc4:**

**cac\_lenh**

**else:**

**cac\_lenh**

Dưới đây là ví dụ minh họa cho cấu trúc các lệnh if lồng nhau trong Python:

a=111

if a>=10:

if a>=100:

print ("gia tri cua a lon hon 100")

else:

print ("gia tri cua a lon hon 10 nho hon 100")

else:

print ("gia tri cua a nho hon 10")

Kết quả:



**Vòng lặp While**

Vòng lặp while trong Python thực thi lặp đi lặp lại các lệnh hoặc phần thân của vòng lặp miễn là điều kiện đã cho là true. Khi điều kiện là false, thì điều khiển sẽ thoát ra khỏi vòng lặp. Dưới đây là cú pháp của vòng lặp while trong Python:

**while bieu\_thuc:**

**cac\_lenh**

Ở đây, cac\_lenh có thể là một lệnh đơn hoặc một khối lệnh. Bieu\_thuc có thể là bất kỳ biểu thức nào. Điều đáng chú ý về vòng lặp while là vòng lặp này có thể không chạy. Bởi vì khi điều kiện được kiểm tra là false, thì phần thân vòng lặp sẽ bị bỏ qua và lệnh đầu tiên ngay sau vòng lặp sẽ được thực thi.

Ví dụ minh họa cho vòng lặp while trong Python:

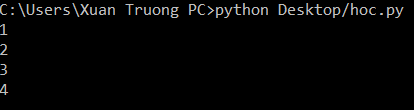
a=1

while a<5:

print (a)

a+=1

Kết quả:



Vòng lặp vô hạn trong Python

Một vòng lặp vô hạn là vòng lặp mà điều kiện của nó là luôn true. Bạn phải đặc biệt chú ý khi sử dụng các vòng lặp while bởi vì tồn tại khả năng là điều kiện của nó sẽ không bao giờ false, tức là làm cho vòng lặp không bao giờ kết thúc.

Một vòng lặp vô hạn có thể là rất hữu ích trong lập trình client/server, tại đó server cần chạy liên tục để mà các chương trình client có thể giao tiếp với nó khi cần thiết.

Sử dụng lệnh else với vòng lặp while trong Python

Python cho phép bạn có một lệnh else được sử dụng kết hợp với một lệnh vòng lặp. Khi else được sử dụng với một vòng lặp while, thì lệnh else được thực thi khi điều kiện là false.

**Vòng lặp FOR**

Vòng lặp for được sử dụng để lặp một biến qua một dãy (List hoặc String) theo thứ tự mà chúng xuất hiện. Sau đây là cú pháp của vòng lặp for:

for bien\_vong\_lap in day\_sequense:

cac\_lenh

Nếu một dãy day\_sequense gồm một danh sách các biểu thức, nó được ước lượng đầu tiên. Sau đó, item đầu tiên trong dãy được gán cho biến vòng lặp bien\_vong\_lap. Tiếp theo, các khối lệnh cac\_lenh được thực thi và khối lệnh này được thực thi tới khi dãy này đã được lặp xong.

Dưới đây là ví dụ minh họa vòng lặp for trong Python:

for i in 'DHBKHN':

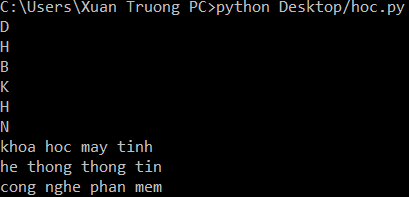
print (i)

bomon=['khoa hoc may tinh','he thong thong tin','cong nghe phan mem']

for i in bomon:

print (i)

Kết quả:



Sử dụng lệnh else với vòng lặp for trong Python

Python cho phép bạn có một lệnh else để liên hợp với một lệnh vòng lặp. Với vòng lặp for, lệnh else được thực thi khi vòng lặp đã lặp qua hết các phần tử trong list.

Lồng vòng lặp

Ngôn ngữ lập trình Python cho phép bạn sử dụng một vòng lặp bên trong một vòng lặp khác.

Cú pháp lồng vòng lặp for trong Python

for bien\_vong\_lap in day\_seq:

for bien\_vong\_lap in day\_seq:

cac\_lenh

cac\_lenh

Cú pháp lồng vòng lặp while trong Python

while bieu\_thuc:

while bieu\_thuc:

cac\_lenh

cac\_lenh

Một ghi chú khác trong lồng vòng lặp là bạn có thể đặt bất cứ kiểu vòng lặp nào bên trong kiểu vòng lặp khác. Ví dụ như bạn có thể đặt một vòng lặp for bên trong một vòng lặp while hoặc ngược lại.

**Các lệnh điều khiển lặp**

**Lệnh break**: lệnh break trong Python là giống như lệnh break trong C. Lệnh này kết thúc vòng lặp hiện tại và truyền điều khiển tới cuối vòng lặp. Lệnh break này có thể được sử dụng trong vòng lặp while và vòng lặp for. Nếu bạn đang sử dụng lồng vòng lặp, thì lệnh break kết thúc sự thực thi của vòng lặp bên trong và bắt đầu thực thi dòng code tiếp theo của khối.

**Lệnh continue**: lệnh continue trả về điều khiển tới phần ban đầu của vòng lặp. Lệnh này bỏ qua lần lặp hiện tại và bắt buộc lần lặp tiếp theo của vòng lặp diễn ra. Lệnh continue có thể được sử dụng trong vòng lặp while hoặc vòng lặp for.

**Lệnh pass**: lệnh pass, giống như tên của nó, được sử dụng khi một lệnh là cần thiết theo cú pháp nhưng bạn không muốn bất cứ lệnh hoặc khối code nào được thực thi. Lệnh pass là một hoạt động null và không có gì xảy ra khi nó thực thi.

## II.4. Cấu trúc dữ liệu

### Kiểu dữ liệu NUMBER

Kiểu dữ liệu Number lưu trữ các giá trị số. Chúng là các kiểu dữ liệu immutable, hay là kiểu dữ liệu không thay đổi, nghĩa là các thay đổi về giá trị của kiểu dữ liệu số này sẽ tạo ra một đối tượng được cấp phát mới. Các đối tượng Number được tạo khi bạn gán một giá trị cho chúng. Bạn cũng có thể xóa tham chiếu tới một đối tượng Number bởi sử dụng lệnh **del**.

Python hỗ trợ 4 kiểu dữ liệu số, đó là:

Kiểu **int**: kiểu số nguyên không có dấu thập phân.

Kiểu **long**: là các số nguyên không giới hạn kích cỡ, được theo sau bởi một chữ l hoặc chữ L.

Kiểu **float**: số thực với dấu thập phân. Kiểu này cũng có thể được viết ở dạng số mũ của 10 với E hoặc e như (2.5e2 = 2.5 x 102 = 250).

Kiểu **số phức**: là trong dạng a + bJ, với a và b là số thực và J (hoặc j) biểu diễn căn bậc hai của -1. Phần thực là a và phần ảo là b. Nói chung, số phức không được sử dụng nhiều trong lập trình Python.

**Chuyển đổi kiểu số trong Python**

Python chuyển đổi các số một cách nội tại bên trong một biểu thức chứa các kiểu phức tạp thành một kiểu chung để ước lượng. Tuy nhiên có đôi khi bạn cần chuyển đổi tường minh một số từ kiểu này sang kiểu khác để thỏa mãn yêu cầu của một toán tử hoặc một hàm.

Để chuyển đổi số x thành số thuần nguyên, bạn gõ **int(x)**.

Để chuyển đổi số x thành số long, bạn gõ **long(x)**.

Để chuyển đổi số x thành số thực, bạn gõ **float(x)**.

Để chuyển đổi số x thành số phức với phần thực là x và phần ảo là 0, bạn gõ **complex(x).**

Để chuyển đổi số x và y thành số phức với phần thực là x và phần ảo là y, bạn gõ **complex(x, y).**

**Hằng toán học trong Python**

Python cũng định nghĩa hai hằng toán học là: hằng toán học **pi** và hằng toán học **e**.

Để làm việc và thao tác với các số, Python cũng cung cấp cho bạn một danh sách các hàm xử lý số đa dạng. Dưới đây là danh sách các hàm.

**Một số hàm toán học trong Python**

|  |  |
| --- | --- |
| Hàm | Miêu tả |
| abs(x) | Lấy giá trị tuyệt đói của x |
| ceil(x) | Số nguyên nhỏ nhất mà không nhỏ hơn x |
| cmp(x,y) | Trả về -1 nếu x<y, trả về 0 nếu x==y hoặc 1 nếu x>y |
| exp(x) | Trả về e­­x |
| fabs(x) | Giá trị tuyệt đối của x |
| floor(x) | Số nguyên lớn nhất mà không lớn hơn x |
| log(x) | Trả về lnx với x>0 |
| pow(x,y) | Trả về giá trị của x\*\*y |
| round(x,n) | Làm tròn x về n chữ số sau dấu thập phân |
| sqrt(x) | Trả về căn bậc hai của x với x>0 |

**Một số hàm xử lí số ngẫu nhiên**

|  |  |
| --- | --- |
| Hàm | Miêu tả |
| choice(seq) | Một item ngẫu nhiên trong một list, tuple, hoặc một |
| randrange(start,stop,step) | Một phần tử được lựa chọn một cách ngẫu nhiên từ dãy (start,stop,step) |
| random() | Một số thực ngẫu nhiên r trong dãy  0 <= r < 1 |
| seed(x) | Thiết lập giá trị nguyên bắt đầu mà được sử dụng trong bộ sinh số ngẫu nhiên. Bạn nên gọi hàm này trước khi gọi bất cứ hàm ngẫu nhiên nào khacs. Hàm này trả về NONE |
| shuffle(list) | Sắp xếp các item trong list một cách ngẫu nhiên |
| uniform(x,y) | Trả về một số thực ngẫu nhiên r trong dãy x <= r <y |

**Một số hàm lượng giác**

|  |  |
| --- | --- |
| Hàm | Miêu tả |
| acos(x) | Trả về giá trị arccos(x) với đơn vị radian |
| asin(x) | Trả về giá trị arcsin(x) với đơn vị radian |
| atan(x) | Trả vể giá trị arctan(x) với đơn vị radian |
| atan2(y,x) | Trả về giá trị arctan(y/x) với đơn vị radian |
| cos(x) | Trả về giá trị cos(x) |
| sin(x) | Trả về giá trị sin(x) |
| tan(x) | Trả về giá trị tan(x) |
| degrees(x) | Chuyển góc x từ radian thành độ |

### Kiểu dữ liệu String

String là một trong các kiểu phổ biến nhất trong Python. String trong Python là immutable. Chúng ta có thể tạo các chuỗi bằng cách bao một text trong một trích dẫn đơn hoặc trích dẫn kép. Python coi các lệnh trích dẫn đơn và kép là như nhau.

**Truy cập các giá trị trong String**

Python không hỗ trợ một kiểu chữ cái; chúng được coi như các chuỗi có độ dài là 1. Trong Python, String được lưu giữ dưới dạng các ký tự đơn trong vị trí ô nhớ liên tiếp nhau. Lợi thế của sử dụng String là nó có thể được truy cập từ cả hai hướng (tiến về trước forward hoặc ngược về sau backward).

Việc lập chỉ mục của cả hai hướng đều được cung cấp bởi sử dụng String trong Python:

Chỉ mục với hướng forward bắt đầu với 0,1,2,3,…

Chỉ mục với hướng backward bắt đầu với -1,-2,-3,…

Để truy cập các giá trị trong String, bạn sử dụng các dấu ngoặc vuông có chỉ mục ở bên trong.

**Cập nhật String trong Python**

Bạn có thể cập nhật một chuỗi đang tồn tại bằng cách gán (hoặc tái gán) một biến cho string khác. Giá trị mới có thể liên quan hoặc khác hoàn toàn giá trị trước đó.

**Các kí tự thoát trong Python**

Ký tự thoát Biểu diễn trong hệ 16 Miêu tả

\a 0x07 Bell hoặc alert

\b 0x08 Backspace

\cx Control-x

\C-x Control-x

\e 0x1b Escape

\f 0x0c Formfeed

\M-\C-x Meta-Control-x

\n 0x0a Newline

\nnn Notation trong hệ cơ số 8, ở đây n là trong dãy từ 0 tới 7

\r 0x0d Carriage return

\s 0x20 Space

\t 0x09 Tab

\v 0x0b Tab dọc

\x Ký tự x

\xnn Notation trong hệ thập lục phân, ở đây n là trong dãy từ 0.9, a.f, hoặc A.F

**Các toán tử để thao tác với String trong Python**

Có ba kiểu toán tử được hỗ trợ bởi String, đó là:

+ Toán tử cơ bản

+ Toán tử membership

+Toán tử quan hệ

**Các toán tử cơ bản để thao tác với String**

Có hai loại toán tử cơ bản có thể được sử dụng với String, đó là toán tử **nối chuỗi +** và toán tử **lặp chuỗi \***.

Toán tử nối chuỗi + được sử dụng để nối hai chuỗi với nhau và tạo nên một chuỗi mới.

Chú ý: Cả hai toán hạng được truyền cho phép nối chuỗi này phải cùng kiểu, nếu không sẽ tạo một lỗi.

Toán tử lặp chuỗi \* sử dụng hai tham số. Một tham số là giá trị nguyên và tham số khác là chuỗi. Toán tử lặp chuỗi này được sử dụng để lặp đi lặp lại một chuỗi một số lần nào đó.

Ghi chú: Bạn có thể sử dụng toán tử lặp chuỗi \* này theo bất kỳ cách nào như int \* string hoặc string \* int. Cả hai tham số được truyền cho toán tử này phải không trong cùng một kiểu.

**Các toán tử membership để thao tác với String**

Toán tử **in**: trả về true nếu một ký tự là có mặt trong chuỗi đã cho, nếu không nó trả về false.

Toán tử **not in**: trả về true nếu một ký tự là không tồn tại trong chuỗi đã cho, nếu không nó trả về false.

**Các toán tử quan hệ để thao tác với String**

Tất cả các toán tử quan hệ (như <,>, <=, >=, ==, !=, <>) cũng có thể áp dụng cho các String. Các chuỗi được so sánh dựa trên giá trị ASCII hoặc Unicode.

**Dấu chia chuỗi [] trong Python**

Có nhiều cách để chia một chuỗi. Khi chuỗi có thể được truy cập hoặc được lập chỉ mục từ cả hai hướng forward và backward thì chuỗi cũng có thể được chia theo hai hướng này. Dưới đây là cú pháp của dấu chia chuỗi [] trong Python:

<ten\_chuoi>[chi\_muc\_bat\_dau:chi\_muc\_ket\_thuc]

hoac

<ten\_chuoi>[:chi\_muc\_ket\_thuc]

hoac

<ten\_chuoi>[chi\_muc\_bat\_dau:]

Chẳng hạn với cú pháp <ten\_chuoi>[chi\_muc\_bat\_dau:chi\_muc\_ket\_thuc], thì toán tử này sẽ trả về các ký tự nằm trong dãy chỉ mục đã cho.

Ghi chú: chi\_muc\_bat\_dau trong String là inclusive, tức là bao gồm cả ký tự tại vị trí chỉ mục đó. Còn chi\_muc\_ket\_thuc là exclusive, tức là không bao gồm ký tự tại chỉ mục đó.

**Ví dụ cho phân String:**

name='abcdef'

print ("chuoi co gia tri: ",name[:6])

name='truong'

print ("chuoi co gia tri: ",name[:6])

print ("\t cong nghe\nthong tin")

name1=name+name

name2=name\*3

print ("vi du nối chuoi: ",name1)

print ("vi du lap chuoi: ",name2)

c='k'

if c in name:

print ("ki tu ",c,"khong co trong ",name)

else:

print ("ki tu ",c,"co trong ",name)

name3='thinh'

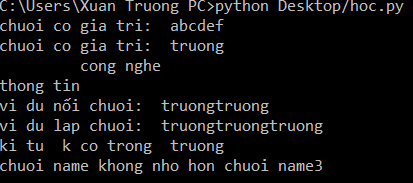
if name<name3:

print ("chuoi name nho hon chuoi name3")

else:

print ("chuoi name khong nho hon chuoi name3")

Kết quả trên màn hình:



**Toán tử định dạng chuỗi trong Python**

Một trong những đặc điểm hay nhất trong Python là toán tử định dạng chuỗi %. Toán tử này là duy nhất cho các String và được sử dụng với hàm print().

**Trích dẫn tam (triple quote) trong Python**

Trích dẫn tam trong Python cho phép các chuỗi có thể trải rộng trên nhiều dòng, bao gồm đúng nguyên văn của các newline (dòng mới), tab và bất kỳ ký tự đặc biệt nào khác. Bạn theo dõi đoạn code sau:

para\_str = """day la mot chuoi day gom nhieu dong

va gom mot so ky tu khong in duoc chang han nhu

TAB ( \t ) chung se duoc hien thi dung nguyen van nhu the."""

print para\_str

Khi code trên được thực thi, nó cho kết quả như dưới đây. Bạn chú ý cách mỗi ký tự đặc biệt đã được chuyển đổi thành dạng được in của nó.

day la mot chuoi day gom nhieu dong

va gom mot so ky tu khong in duoc chang han nhu

TAB ( \t ) chung se duoc hien thi dung nguyen van nhu the.

Các chuỗi thô (raw string) không coi dấu \ như là một ký tự đặc biệt. Mỗi ký tự bạn đặt vào trong một chuỗi thô sẽ tồn tại giống như cách bạn đã viết nó.

Để hiểu rõ vấn đề này, trước hết bạn theo dõi ví dụ:

print 'C:\\nowhere'

Khi code trên được thực thi sẽ cho kết quả:

C:\nowhere

Bây giờ sử dụng chuỗi thô. Chúng ta đã đặt biểu thức trong r'bieu\_thuc' như sau:

print r'C:\\nowhere'

Khi code trên được thực thi sẽ cho kết quả:

C:\\nowhere

**Chuỗi dạng Unicode trong Python**

Các chuỗi thông thường trong Python được lưu trữ nội tại dưới dạng ASCII 8 bit, trong khi các chuỗi Unicode được lưu trữ dưới dạng Unicode 16 bit. Điều này cho phép để có một tập hợp các ký tự đa dạng hơn, bao gồm các ký tự đặc biệt từ hầu hết các ngôn ngữ trên thế giới. Bạn theo dõi ví dụ:

print u'Hello, world!'

Khi code trên được thực thi sẽ cho kết quả:

Hello, world!

Như bạn có thể thấy, các chuỗi dạng Unicode sử dụng tiền tố u, trong khi các chuỗi thô sử dụng tiền tố r.

**Một số phương thức và hàm đã xây dựng sẵn để xử lý chuỗi trong Python**

**STT Phương thức và Miêu tả**

**1 Phương thức capitalize(): Viết hoa chữ cái đầu tiên của chuỗi**

**2 Phương thức center(width, fillchar): Trả về một chuỗi mới, trong đó chuỗi ban đầu đã được cho vào trung tâm và hai bên đó là các fillchar sao cho tổng số ký tự của chuỗi mới là width**

**3 Phương thức count(str, beg= 0,end=len(string)): Đếm xem chuỗi str này xuất hiện bao nhiêu lần trong chuỗi string hoặc chuỗi con của string nếu bạn cung cấp chỉ mục ban đầu start và chỉ mục kết thúc end**

**4 Phương thức decode(encoding='UTF-8',errors='strict')**

**Giải mã chuỗi bởi sử dụng encoding đã cho**

**5 Phương thức encode(encoding='UTF-8',errors='strict')**

**Trả về phiên bản chuỗi đã được mã hóa của chuỗi ban đầu. Nếu có lỗi xảy ra, thì chương trình sẽ tạo một ValueError trừ khi các lỗi này được cung cấp với ignore hoặc replace**

**6 Phương thức endswith(suffix, beg=0, end=len(string))**

**Xác định xem nếu chuỗi string hoặc chuỗi con đã cho của string (nếu bạn cung cấp chỉ mục bắt đầu beg và chỉ mục kết thúc end) kết thúc với hậu tố suffix thì trả về true, nếu không thì phương thức này trả về false**

**7 Phương thức expandtabs(tabsize=8): Mở rộng các tab trong chuỗi tới số khoảng trống đã cho; mặc định là 8 space cho mỗi tab nếu bạn không cung cấp tabsize**

**8 Phương thức find(str, beg=0 end=len(string)): Xác định xem chuỗi str có xuất hiện trong chuỗi string hoặc chuỗi con đã cho của string (nếu bạn cung cấp chỉ mục bắt đầu beg và chỉ mục kết thúc end), nếu xuất hiện thì trả về chỉ mục của str, còn không thì trả về -1**

**9 Phương thức index(str, beg=0, end=len(string)): Tương tự như find(), nhưng tạo ra một ngoại lệ nếu str là không được tìm thấy**

**10 Phương thức isalnum():Trả về true nếu chuỗi có ít nhất một ký tự và tất cả ký tự là chữ-số. Nếu không hàm sẽ trả về false**

**11 Phương thức isalpha(): Trả về true nếu chuỗi có ít nhất 1 ký tự và tất cả ký tự là chữ cái. Nếu không phương thức sẽ trả về false**

**12 Phương thức isdigit(): Trả về true nếu chuỗi chỉ chứa các chữ số, nếu không là false**

**13 Phương thức islower(): Trả về true nếu tất cả ký tự trong chuỗi là ở dạng chữ thường, nếu không là false**

**14 Phương thức isnumeric(): Trả về true nếu một chuỗi dạng Unicode chỉ chứa các ký tự số, nếu không là false**

**15 Phương thức isspace(): Trả về true nếu chuỗi chỉ chứa các ký tự khoảng trắng whitespace, nếu không là false**

**16 Phương thức istitle(): Trả về true nếu chuỗi là ở dạng titlecase, nếu không là false**

**17 Phương thức isupper(): Trả về true nếu tất cả ký tự trong chuỗi là chữ hoa**

**18 Phương thức join(seq): Nối chuỗi các biểu diễn chuỗi của các phần tử trong dãy seq thành một chuỗi**

**19 Phương thức len(string): Trả về độ dài của chuỗi**

**20 Phương thức ljust(width[, fillchar]): Trả về một chuỗi mới, trong đó có chuỗi ban đầu được căn chỉnh vào bên trái và bên phải là các fillchar sao cho tổng số ký tự là width**

**21 Phương thức lower(): Chuyển đối tất cả chữ hoa trong chuỗi sang kiểu chữ thường**

**22 Phương thức lstrip(): Xóa tất cả các khoảng trống trắng ban đầu (leading) trong chuỗi**

**23 Phương thức maketrans(): Trả về một bảng thông dịch được sử dụng trong hàm translate**

**24 Phương thức max(str): Trả về ký tự chữ cái lớn nhất từ chuỗi str đã cho**

**25 Phương thức min(str): Trả về ký tự chữ cái nhỏ nhất từ chuỗi str đã cho**

**26 Phương thức replace(old, new [, max]): Thay thế tất cả sự xuất hiện của old trong chuỗi với new với số lần xuất hiện max (nếu cung cấp)**

**27 Phương thức rfind(str, beg=0,end=len(string)): Tương tự hàm find(), nhưng trả về chỉ mục cuối cùng**

**28 Phương thức rindex( str, beg=0, end=len(string)): Giống index(), nhưng trả về chỉ mục cuối cùng nếu tìm thấy**

**29 Phương thức rjust(width,[, fillchar]): Trả về một chuỗi mới, trong đó có chuỗi ban đầu được căn chỉnh vào bên phải và bên trái là các fillchar sao cho tổng số ký tự là width**

**30 Phương thức rstrip(): Xóa bỏ tất cả các khoảng trống trắng ở cuối (trailing) của chuỗi**

**31 Phương thức split(str="", num=string.count(str)): Chia chuỗi theo delimeter đã cho (là space nếu không được cung cấp) và trả về danh sách các chuỗi con; nếu bạn cung cấp num thì chia chuỗi thành num chuỗi con**

**32 Phương thức splitlines( num=string.count('\n')): Trả về một List gồm tất cả các dòng trong chuỗi, và tùy ý xác định các ngắt dòng (nếu num được cung cấp và là true).**

**33 Phương thức startswith(str, beg=0,end=len(string): Xác định xem chuỗi hoặc chuỗi con (nếu bạn cung cấp chỉ mục bắt đầu beg và chỉ mục kết thúc end) có bắt đầu với chuỗi con str không, nếu có trả về true, nếu không là false**

**34 Phương thức strip([chars]): Thực hiện cả hai phương thức lstrip() và rstrip() trên chuỗi**

**35 Phương thức swapcase(): Đảo ngược kiểu của tất cả ký tự trong chuỗi**

**36 Phương thức title(): Trả về một bản sao của chuỗi trong đó tất cả ký tự đầu tiên của tất cả các từ là ở kiểu chữ hoa.**

**37 Phương thức translate(table, deletechars=""): Trả về một bản sao đã được thông dịch của chuỗi**

**38 Phương thức upper(): Chuyển đổi các chữ thường trong chuỗi thành chữ hoa**

**39 Phương thức zfill (width): Trả về một chuỗi mới, trong đó bao gồm chuỗi ban đầu và được đệm thêm với các số 0 vào bên trái sao cho tổng ký tự là width**

**40 Phương thức isdecimal(): Trả về true nếu một chuỗi dạng Unicode chỉ chứa các ký tự thập phân, nếu không là false**

### LIST

Cấu trúc dữ liệu cơ bản nhất trong Python là dãy (sequence). Mỗi phần tử trong dãy được gán một số, là vị trí hoặc chỉ mục của nó. Chỉ mục đầu tiên là 0, chỉ mục thứ hai là 1, và …

Python có sáu kiểu dãy đã được xây dựng sẵn, và trong loạt bài này chúng ta sẽ tìm hiểu hai kiểu được sử dụng phổ biến nhất là List và Tuple.

List trong Python là cấu trúc dữ liệu mà có khả năng lưu giữ các kiểu dữ liệu khác nhau.

List trong Python là thay đổi (mutable), nghĩa là Python sẽ không tạo một List mới nếu bạn sửa đổi một phần tử trong List.

List là một container mà giữ các đối tượng khác nhau trong một thứ tự đã cho. Các hoạt động khác nhau như chèn hoặc xóa có thể được thực hiện trên List.

Một List có thể được tạo ra bởi lưu trữ một dãy các kiểu giá trị khác nhau được phân biệt bởi các dấu phảy. Dưới đây là cú pháp để tạo List:

<ten\_list>=[giatri1, giatri2, ..., giatriN];

Ví dụ:

list1 = ['vatly', 'hoahoc', 1997, 2000];

list2 = [1, 2, 3, 4, 5 ];

list3 = ["a", "b", "c", "d"];

Một List trong Python được bao xung quanh bởi các dấu ngoặc vuông [].

Tương tự như chỉ mục của chuỗi, chỉ mục của List bắt đầu từ 0.

**Truy cập các giá trị trong List trong Python**

Để truy cập các giá trị trong List, bạn sử dụng cú pháp sau:

<ten\_list>[index]

để lấy giá trị có sẵn tại chỉ mục đó.

Ví dụ:

list1 = ['vatly', 'hoahoc', 1997, 2000];

list2 = [1, 2, 3, 4, 5, 6, 7 ];

print "list1[0]: ", list1[0]

print "list2[1:5]: ", list2[1:5]

Khi code trên được thực thi sẽ cho kết quả:

list1[0]: vatly

list2[1:5]: [2, 3, 4, 5]

Ghi chú: Trình tổ chức bộ nhớ nội tại

List không lưu trữ các phần tử một cách trực tiếp tại chỉ mục. Sự thực là một tham chiếu được lưu trữ tại mỗi chỉ mục mà tham chiếu tới đối tượng được lưu trữ ở đâu đó trong bộ nhớ. Điều này là do một số đối tượng có thể lớn hơn một số đối tượng khác và vì thế chúng được lưu trữ tại một vị trí bộ nhớ khác.

**Các hoạt động cơ bản trên List trong Python**

Bạn có thể thực hiện các hoạt động nối với toán tử + hoặc hoạt động lặp với \* như trong các chuỗi. Điểm khác biệt là ở đây nó tạo một List mới, không phải là một chuỗi.

Ví dụ cho nối List:

list1=[10,20]

list2=[30,40]

list3=list1+list2

print list3

Kết quả là:

[10, 20, 30, 40]

Ghi chú: Toán tử + ngụ ý rằng cả hai toán hạng được truyền cho nó phải là List, nếu không sẽ cho một lỗi.

**Cập nhật List trong Python**

Bạn có thể cập nhật một hoặc nhiều phần tử của List bởi gán giá trị cho chỉ mục cụ thể đó. Cú pháp:

<ten\_list>[index]=<giatri>

Ví dụ:

list = ['vatly', 'hoahoc', 1997, 2000];

print "Gia tri co san tai chi muc thu 2 : "

print list[2]

list[2] = 2001;

print "Gia tri moi tai chi muc thu 2 : "

print list[2]

Khi code trên được thực thi sẽ cho kết quả:

Gia tri co san tai chi muc thu 2 :

1997

Gia tri moi tai chi muc thu 2 :

2001

Phụ thêm phần tử vào cuối một List

Phương thức append() được sử dụng để phụ thêm phần tử vào cuối một List. Cú pháp:

<ten\_list>.append(item)

Ví dụ:

list1=[10,"hoang",'z']

print "Cac phan tu cua List la: "

print list1

list1.append(10.45)

print "Cac phan tu cua List sau khi phu them la: "

print list1

Khi code trên được thực thi sẽ cho kết quả:

Cac phan tu cua List la:

[10, 'hoang', 'z']

Cac phan tu cua List sau khi phu them la:

[10, 'hoang', 'z', 10.45]

**Xóa phần tử trong List**

Để xóa một phần tử trong List, bạn có thể sử dụng lệnh del nếu bạn biết chính xác phần tử nào bạn muốn xóa hoặc sử dụng phương thức **remove()** nếu bạn không biết. Ví dụ:

list1 = ['vatly', 'hoahoc', 1997, 2000];

print list1

del list1[2];

print "Cac phan tu cua List sau khi xoa gia tri tai chi muc 2 : "

print list1

Khi code trên được thực thi sẽ cho kết quả:

['vatly', 'hoahoc', 1997, 2000]

Cac phan tu cua List sau khi xoa gia tri tai chi muc 2 :

['vatly', 'hoahoc', 2000]

Bạn cũng có thể sử dụng del để xóa tất cả phần tử từ chi\_muc\_bat\_dau tới chi\_muc\_ket\_thuc như sau:

list1=[10,'hoang',50.8,'a',20,30]

print list1

del list1[0]

print list1

del list1[0:3]

print list1

Khi code trên được thực thi sẽ cho kết quả:

[10, 'hoang', 50.8, 'a', 20, 30]

['hoang', 50.8, 'a', 20, 30]

[20, 30]

Các hàm và phương thức đã xây dựng sẵn để xử lý List trong Python

Ngoài các phương thức kể trên, Python còn xây dựng sẵn rất nhiều hàm và phương thức để bạn có thể sử dụng khi làm việc với List. Bảng dưới đây liệt kê các phương thức này.

**Danh sách các hàm xử lý List trong Python:**

STT Hàm và Miêu tả

1 Hàm cmp(list1, list2): So sánh các phần tử trong cả hai list

2 Hàm len(list): Trả về độ dài của list

3 Hàm max(list): Trả về phần tử có giá trị lớn nhất trong list

4 Hàm min(list): Trả về phần tử có giá trị nhỏ nhất trong list

5 Hàm list(seq): Chuyển đổi một tuple thành list

**Danh sách các phương thức xử lý List trong Python:**

STT Phương thức và Miêu tả

1 Phương thức list.append(obj): Phụ thêm đối tượng obj vào cuối list

2 Phương thức list.count(obj): Đếm xem có bao nhiêu lần mà obj xuất hiện trong list

3 Phương thức list.extend(seq): Phụ thêm các nội dung của seq vào cuối list

4 Phương thức list.index(obj): Trả về chỉ mục thấp nhất trong list mà tại đó obj xuất hiện

5 Phương thức list.insert(index, obj): Chèn đối tượng obj vào trong list tại index đã cho

6 Phương thức list.pop(obj=list[-1]): Xóa và trả về phần tử cuối cùng hoặc đối tượng obj có chỉ mục đã cung cấp từ list đã cho

7 Phương thức list.remove(obj): Xóa đối tượng obj từ list

8 Phương thức list.reverse(): Đảo ngược thứ tự các đối tượng trong list

9 Phương thức list.sort([func]): Sắp xếp các đối tượng của list, sử dụng hàm so sánh nếu được cung cấp

### Tuple

Một tuple là một dãy các đối tượng không thay đổi (immutable) trong Python, vì thế tuple không thể bị thay đổi. Các tuple cũng là các dãy giống như List.

Không giống List mà sử dụng các dấu ngoặc vuông, thì tuple sử dụng các dấu ngoặc đơn. Các đối tượng trong tuple được phân biệt bởi dấu phảy và được bao quanh bởi dấu ngoặc đơn (). Giống như chỉ mục của chuỗi, chỉ mục của tuple bắt đầu từ 0.

Ví dụ:

data=(10,20,'ram',56.8)

data2="a",10,20.9

data

(10, 20, 'ram', 56.8)

data2

('a', 10, 20.9)

Ghi chú: Nếu các dấu ngoặc đơn không được cung cấp với một dãy, thì nó được coi như là tuple.

Một tuple trống không chứa phần tử nào, ví dụ:

tup1 = ();

Với một tuple chỉ có một giá trị đơn, thì phải có một dấu phảy ở cuối, ví dụ:

tup1 = (50,);

Các tuple cũng có thể được lồng vào nhau, ví dụ:

tupl1='a','hoang',10.56

tupl2=tupl1,(10,20,30)

print tupl1

print tupl2

Kết quả:

('a', 'hoang', 10.56)

(('a', 'hoang', 10.56), (10, 20, 30))

**Truy cập các giá trị trong tuple trong Python**

Để truy cập các giá trị trong tuple, bạn sử dụng cách tương tự như khi truy cập các phần tử trong List. Ví dụ:

tup1 = ('vatly', 'hoahoc', 1997, 2000);

tup2 = (1, 2, 3, 4, 5, 6, 7 );

print "tup1[0]: ", tup1[0]

print "tup2[1:5]: ", tup2[1:5]

Khi code trên được thực thi sẽ cho kết quả:

tup1[0]: vatly

tup2[1:5]: [2, 3, 4, 5]

**Các hoạt động cơ bản trên tuple trong Python**

Giống như String và List, bạn cũng có thể sử dụng toán tử nối + và toán tử lặp \* với tuple. Điểm khác biệt là nó tạo ra một tuple mới, không tạo ra một chuỗi hay list.

Ví dụ cho toán tử +:

data1=(1,2,3,4)

data2=('x','y','z')

data3=data1+data2

print data1

print data2

print data3

Kết quả là:

(1, 2, 3, 4)

('x', 'y', 'z')

(1, 2, 3, 4, 'x', 'y', 'z')

Ghi chú: Dãy mới được tạo là một Tuple mới.

Ví dụ cho toán tử +:

tuple1=(10,20,30);

tuple2=(40,50,60);

print tuple1\*2

print tuple2\*3

Kết quả là:

(10, 20, 30, 10, 20, 30)

(40, 50, 60, 40, 50, 60, 40, 50, 60)

**Xóa các phần tử của tuple trong Python**

Xóa các phần tử đơn của tuple là điều không thể. Bạn chỉ có thể xóa toàn bộ tuple với lệnh del như ví dụ sau:

data=(10,20,'hoang',40.6,'z')

print data

del data #se xoa du lieu cua tuple

print data #se hien thi mot error boi vi tuple da bi xoa

Code trên sẽ cho kết quả sau. Bạn chú ý rằng sẽ có một exception được tạo ra, đó là bởi vì sau khi xóa tup thì tuple này không tồn tại nữa.

(10, 20, 'hoang', 40.6, 'z')

Traceback (most recent call last):

File "C:/Python27/t.py", line 4, in >module<

print data

NameError: name 'data' is not defined

**Cập nhật phần tử trong tuple trong Python**

Các phần tử của Tuple không thể được cập nhật. Đó là bởi vì tuple là không thay đổi (immutable). Tuy nhiên, các tuple có thể được sử dụng để tạo nên một tuple mới.

Ví dụ sau sẽ tạo một exception:

data=(10,20,30)

data[0]=100

print data

Khi code trên được thực thi sẽ cho kết quả:

Traceback (most recent call last):

File "C:/Python27/t.py", line 2, in >module<

data[0]=100

TypeError: 'tuple' object does not support item assignment

Ví dụ tạo một tuple mới từ các tuple đang tồn tại:

data1=(10,20,30)

data2=(40,50,60)

data3=data1+data2

print data3

Khi code trên được thực thi sẽ cho kết quả:

(10, 20, 30, 40, 50, 60)

Tập hợp đối tượng mà không có dấu giới hạn

Bất kỳ tập hợp nào gồm nhiều đối tượng, được phân biệt bởi dấu phảy, được viết mà không có các biểu tượng nhận diện (chẳng hạn như dấu ngoặc vuông cho List, dấu ngoặc đơn cho Tuple, …) thì Python mặc định chúng là Tuple. Ví dụ:

print 'abc', -4.24e93, 18+6.6j, 'xyz'

x, y = 1, 2;

print "Gia tri cua x , y : ", x,y

Khi code trên được thực thi sẽ cho kết quả:

abc -4.24e+93 (18+6.6j) xyz

Gia tri cua x , y : 1 2

**Các hàm được xây dựng sẵn cho tuple trong Python**

STT Hàm và Miêu tả

1 Hàm cmp(tuple1, tuple2): So sánh hai tuple với nhau

2 Hàm len(tuple): Trả về độ dài của tuple

3 Hàm max(tuple): Trả về item có giá trị lớn nhất từ một tuple đã cho

4 Hàm min(tuple): Trả về item có giá trị nhỏ nhất từ một tuple đã cho

5 Hàm tuple(seq): Chuyển đổi một dãy thành tuple

### Dictionary

Dictionary trong Python là một tập hợp các cặp key và value không có thứ tự. Nó là một container mà chứa dữ liệu, được bao quanh bởi các dấu ngoặc móc đơn {}. Mỗi cặp key-value được xem như là một item. Key mà đã truyền cho item đó phải là duy nhất, trong khi đó value có thể là bất kỳ kiểu giá trị nào. Key phải là một kiểu dữ liệu không thay đổi (immutable) như chuỗi, số hoặc tuple.

Key và value được phân biệt riêng rẽ bởi một dấu hai chấm (:). Các item phân biệt nhau bởi một dấu phảy (,). Các item khác nhau được bao quanh bên trong một cặp dấu ngoặc móc đơn tạo nên một Dictionary trong Python

Ví dụ:

data={100:'Hoang' ,101:'Nam' ,102:'Binh'}

print data

Kết quả là:

{100: 'Hoang', 101: 'Nam', 102: 'Binh'}

**Các thuộc tính của key trong Dictionary**

Không có hạn chế nào với các value trong Dictionary, tuy nhiên với key thì bạn cần chú ý các điểm sau:

(a) Nhiều hơn một entry cho mỗi key là không được phép. Nghĩa là không cho phép bản sao các key được xuất hiện. Khi bắt gặp nhiều bản sao key trong phép gán, thì phép gán cuối cùng được thực hiện. Ví dụ:

dict = {'Ten': 'Hoang', 'Tuoi': 7, 'Ten': 'Nam'};

print "dict['Ten']: ", dict['Ten']

Kết quả là:

dict['Ten']: Nam

(b) Key phải là immutable. Nghĩa là bạn chỉ có thể sử dụng chuỗi, số hoặc tuple làm key của Dictionary. Dưới đây là ví dụ đơn giản:

dict = {['Ten']: 'Hoang', 'Tuoi': 7};

print "dict['Ten']: ", dict['Ten']

Khi code trên được thực thi sẽ cho kết quả:

Traceback (most recent call last):

File "test.py", line 3, in <module>

dict = {['Ten']: 'Hoang', 'Tuoi': 7};

TypeError: list objects are unhashable

Truy cập các giá trị trong Dictionary trong Python

Khi chỉ mục không được định nghĩa với Dictionary, thì các giá trị trong Dictionary có thể được truy cập thông qua các key của chúng. Cú pháp:

<ten\_dictionary>[key]

Ví dụ:

data1={'Id':100, 'Ten':'Thanh', 'Nghenghiep':'Developer'}

data2={'Id':101, 'Ten':'Chinh', 'Nghenghiep':'Trainer'}

print "Id cua nhan vien dau tien la",data1['Id']

print "Id cua nhan vien thu hai la",data2['Id']

print "Ten cua nhan vien dau tien la:",data1['Ten']

print "Nghe nghiep cua nhan vien thu hai la:",data2['Nghenghiep']

Kết quả là:

Id cua nhan vien dau tien la 100

Id cua nhan vien thu hai la 101

Ten cua nhan vien dau tien la is Thanh

Nghe nghiep cua nhan vien thu hai la Trainer

Nếu bạn cố gắng truy cập một item với một key nào mà không là một phần của Dictionary nào, thì bạn sẽ nhận một lỗi như sau:

dict = {'Ten': 'Hoang', 'Tuoi': 7, 'Lop': 'Lop1'};

print "dict['Huong']: ", dict['Huong']

Code trên sẽ cho một lỗi là:

dict['Hoang']:

Traceback (most recent call last):

File "test.py", line 4, in <module>

print "dict['Huong']: ", dict['Huong'];

KeyError: 'Huong'

Cập nhật Dictionary trong Python

Item (cặp key-value) có thể được cập nhật. Bạn cập nhật một Dictionary bằng cách thêm một entry mới hoặc một cặp key-value mới, sửa đổi một entry đã tồn tại, hoặc xóa một entry đang tồn tại như trong ví dụ đơn giản sau:

data1={'Id':100, 'Ten':'Thanh', 'Nghenghiep':'Developer'}

data2={'Id':101, 'Ten':'Chinh', 'Nghenghiep':'Trainer'}

data1['Nghenghiep']='Manager'

data2['Mucluong']=17000000

data1['Mucluong']=12000000

print data1

print data2

Khi code trên được thực thi sẽ cho kết quả:

{'Mucluong': 12000000, 'Nghenghiep': 'Manager','Id': 100, 'Ten': 'Thanh'}

{'Mucluong': 17000000, 'Nghenghiep': 'Trainer', 'Id': 101, 'Ten': 'Chinh'}

**Xóa phần tử từ Dictionary trong Python**

Với Dictionary, bạn có thể xóa một phần tử đơn hoặc xóa toàn bộ nội dung của Dictionary đó. Bạn sử dụng lệnh del để thực hiện các hoạt động này.

**Cú pháp để xóa một item từ Dictionary:**

**del ten\_dictionary[key]**

**Để xóa cả Dictionary, bạn sử dụng cú pháp:**

**del ten\_dictionary**

Ví dụ:

data={100:'Hoang', 101:'Thanh', 102:'Nam'}

del data[102]

print data

del data

print data #se hien thi mot error boi vi Dictionary da bi xoa.

Code trên sẽ cho kết quả như dưới đây. Bạn có thể thấy một ngoại lệ được tạo ra bởi vì sau khi xóa data thì Dictionary này không tồn tại nữa.

{100: 'Hoang', 101: 'Thanh'}

Traceback (most recent call last):

File "C:/Python27/dict.py", line 5, in >module<

print data

NameError: name 'data' is not defined

Các hàm và phương thức đã được xây dựng sẵn cho Dictionary trong Python

Python đã xây dựng sẵn các hàm sau để được sử dụng với Dictionary. Bạn có thể theo dõi ví dụ chi tiết về các hàm này ở phần dưới đây.

STT Hàm và Miêu tả

1 Hàm cmp(dict1, dict2): So sánh các phần tử của cả hai dict

2 Hàm len(dict): Độ dài của dict. Nó sẽ là số item trong Dictionary này

3 Hàm str(dict): Tạo ra một biểu diễn chuỗi có thể in được của một dict

4 Hàm type(variable): Trả về kiểu của biến đã truyền. Nếu biến đã truyền là Dictionary, thì nó sẽ trả về một kiểu Dictionary

**Các phương thức đã được xây dựng sẵn cho Dictionary trong Python**:

STT Phương thức và Miêu tả

1 Phương thức dict.clear(): Xóa tất cả phần tử của dict

2 Phương thức dict.copy(): Trả về bản sao của dict

3 Phương thức fromkeys(seq,value1)/ fromkeys(seq): Được sử dụng để tạo một Dictionary mới từ dãy seq và value1. Trong đó dãy seq tạo nên các key và tất cả các key chia sẻ các giá trị từ value1. Trong trường hợp value1 không được cung cấp thì value của các key được thiết lập là None

4 Phương thức dict.get(key, default=None): Trả về giá trị của key đã cho. Nếu key không có mặt thì phương thức này trả về None

5 Phương thức dict.has\_key(key): Trả về true nếu key là có mặt trong Dictionary, nếu không là false

6 Phương thức dict.items(): Trả về tất cả các cặp (key-value) của một Dictionary

7 Phương thức dict.keys(): Trả về tất cả các key của một Dictionary

8 Phương thức dict.setdefault(key, default=None): Tương tự get(), nhưng sẽ thiết lập dict[key]=default nếu key là không tồn tại trong dict

9 Phương thức dict.update(dict2): Được sử dụng để thêm các item của dictionary 2 vào Dictionary đầu tiên

10 Phương thức dict.values(): Trả về tất cả các value của một Dictionary

## II.5. Hàm, module, file

### Date and time

Với Python, bạn có thể dễ dàng thu được Date và Time hiện tại. Chương này sẽ giới thiệu một số phương thức được phổ biến trong khi làm việc với Date và Time bởi sử dụng Python.

Lấy Time hiện tại trong Python

Để lấy Time hiện tại, bạn sử dụng hàm tiền định nghĩa localtime(). Hàm localtime() này nhận một tham số là time.time(). Ở đây, time là module, time() là một hàm mà trả về system time hiện tại được biểu diễn dưới dạng số tick (số tích tắc) từ 12:00 am, 1/1/1970. Về cơ bản, tick là một số thực.

import time;

localtime = time.localtime(time.time())

print "Thoi gian hien tai la :", localtime

Kết quả là:

Thoi gian hien tai la : time.struct\_time(tm\_year=2015, tm\_mon=11, tm\_mday=29, tm\_hour=19, tm\_min=16, tm\_sec=54, tm\_wday=6, tm\_yday=333, tm\_isdst=0)

Dưới đây là phần giải thích:

Time được trả về là một cấu trúc gồm 9 thuộc tính. Như trong bảng sau:

**Thuộc tính Miêu tả**

tm\_year Trả về năm hiện tại (ví dụ: 2015)

tm\_mon Trả về tháng hiện tại (1-12)

tm\_mday Trả về ngày hiện tại (1-31)

tm\_hour Trả về giờ hiện tại (0-23)

tm\_min Trả về phút hiện tại (0-59)

tm\_sec Trả về giây hiện tại (0-61 với 60 và 61 là các dây nhuận)

tm\_wday Trả về ngày trong tuần (0-6 với 0 là Monday)

tm\_yday Trả về ngày trong năm (1-366 kể cả năm nhuận)

tm\_isdst Trả về -1, 0 hoặc 1 tức là có xác định DST không

**Lấy Time đã được định dạng trong Python**

Bạn có thể định dạng bất kỳ time nào theo yêu cầu của bạn, nhưng phương thức đơn giản nhất là asctime(). Đây là một hàm đã được định nghĩa trong time module. Hàm này trả về một time đã được định dạng bao gồm ngày trong tuần, tháng, ngày trong tháng, thời gian và năm. Ví dụ:

import time;

localtime = time.asctime( time.localtime(time.time()) )

print "Thoi gian da duoc dinh dang la :", localtime

Kết quả là:

Thoi gian da duoc dinh dang la : Sun Nov 29 19:16:30 2015

time module trong Python

Có nhiều hàm được định nghĩa sẵn trong time Module mà bạn có thể được sử dụng để làm việc với time.

STT Hàm và Miêu tả

1 Hàm time.altzone: Trả về offset của DST timezone (số giây)

2 Hàm time.asctime([tupletime]): Chấp nhận một time-tuple và trả về một chuỗi gồm 24 ký tự có thể đọc được ví dụ như Mon Dec 11 18:07:14 2015

3 Hàm time.clock( ): Trả về CPU time hiện tại dưới dạng số giây dạng số thực

4 Hàm time.ctime([secs]): Giống asctime(localtime(secs)) và nếu không có tham số thì giống như asctime( )

5 Hàm time.gmtime([secs]): Chuyển đổi một time được biểu diễn là số giây từ epoch sang một struct\_time trong UTC

6 Hàm time.localtime([secs]): Tương tự như gmtime(), nhưng nó chuyển đổi số giây thành local time.

7 Hàm time.mktime(tupletime): Là ngược với hàm localtime(). Trả về một số thực để tương thích với time()

8 Hàm time.sleep(secs): Dừng trình thực thi trong số giây đã cho là secs

9 Hàm time.strftime(fmt[,tupletime]): Chuyển đổi một tuple hoặc struct\_time thành một chuỗi được xác định bởi tham số format

10 Hàm time.strptime(str,fmt='%a %b %d %H:%M:%S %Y'): Parse một chuỗi biểu diễn time theo một định dạng đã cho

11 Hàm time.time( ): Trả về time dưới dạng một số thực được diễn đạt bởi số giây từ epoch, trong UTC

12 Hàm time.tzset(): Phục hồi các qui ước về thời gian được sử dụng bởi các chương trình con của thư viện. Biến môi trường TZ xác định cách được thực hiện

**Có hai thuộc tính quan trọng có sẵn với time Module là:**

**time.timezone**: Thuộc tính time.timezone là số giây trong local timezone (không DST) từ UTC (>0 trong Americas; <=0 trong Europe, Asia, Africa).

**time.tzname**: Thuộc tính time.tzname là một cặp các chuỗi biểu diễn locale và biểu diễn phụ thuộc, mà tương ứng là tên của local timezone với và không với DST.

**calendar Module trong Python**

Python cung cấp calendar Module để giúp bạn hiển thị Calendar. Ví dụ:

import calendar

print "Thang hien tai la:"

cal = calendar.month(2014, 6)

print cal

Kết quả là:

Thang hien tai la:

November 2015

Mo Tu We Th Fr Sa Su

1

2 3 4 5 6 7 8

9 10 11 12 13 14 15

16 17 18 19 20 21 22

23 24 25 26 27 28 29

30

Có rất nhiều hàm và phương thức đã được xây dựng sẵn trong calendar Module giúp bạn làm việc với Calendar. Dưới đây là một số hàm và phương thức:

Hàm Miêu tả

**Hàm prcal(year)** In cả calendar của năm

**Hàm firstweekday()** Trả về ngày trong tuần đầu tiên. Theo mặc định là 0 mà xác định là Monday

**Hàm isleap(year)** Trả về true nếu năm đã cho là năm nhuận, nếu không là false

**Hàm monthcalendar(year,month)** Trả về một list gồm các ngày trong tháng đã cho của năm dưới dạng các tuần

**Hàm leapdays(year1,year2)** Trả về số ngày nhuận từ năm year1 tới năm year2

**Hàm prmonth(year,month)** In ra tháng đã cho của năm đã cung cấp

Dưới đây là ví dụ cho một số hàm:

Hàm firstweekday()

import calendar

print calendar.firstweekday()

Kết quả là:

0

Hàm isleap(year)

import calendar

print calendar.isleap(2000)

Kết quả là:

True

Hàm monthcalendar(year,month)

import calendar

print calendar.monthcalendar(2015,11)

Kết quả là:

[[0, 0, 0, 0, 0, 0, 1], [2, 3, 4, 5, 6, 7, 8], [9, 10, 11, 12, 13, 14, 15], [16, 17, 18, 19, 20, 21, 22], [23, 24, 25, 26, 27, 28, 29], [30, 0, 0, 0, 0, 0, 0]]

Hàm prmonth(year,month)

import calendar

print calendar.prmonth(2015,11)

Kết quả là:

November 2015

Mo Tu We Th Fr Sa Su

1

2 3 4 5 6 7 8

9 10 11 12 13 14 15

16 17 18 19 20 21 22

23 24 25 26 27 28 29

30

None

### Hàm

Hàm, là một khối code được tổ chức và có thể tái sử dụng, để thực hiện một hành động nào đó. Trong các chương trước, bạn đã làm quen với một số hàm đã được xây dựng sẵn trong Python, điển hình như hàm print(). Tuy nhiên bạn cũng có thể tạo riêng cho mình một hàm với cách định nghĩa và kiểu giá trị cho riêng bạn. Các hàm này được gọi là user-defined function.

**Định nghĩa một hàm trong Python**

Khi định nghĩa các hàm để cung cấp một tính năng nào đó, bạn cần theo các qui tắc sau:

Từ khóa **def** được sử dụng để bắt đầu phần định nghĩa hàm. Def xác định phần bắt đầu của khối hàm.

Def được theo sau bởi ten\_ham được theo sau bởi các dấu ngoặc đơn ().Các tham số được truyền vào bên trong các dấu ngoặc đơn. Ở cuối là dấu hai chấm.

Trước khi viết một code, một độ thụt dòng được cung cấp trước mỗi lệnh. Độ thụt dòng này nên giống nhau cho tất cả các lệnh bên trong hàm đó.

Lệnh đầu tiên của hàm là tùy ý, và nó là Documentation String của một hàm đó.

Sau đó là lệnh để được thực thi.

Cú pháp

def ten\_ham( cac\_tham\_so ):

"function\_docstring"

function\_suite

return [bieu\_thuc]

Ví dụ

Hàm sau nhận một chuỗi là tham số input và in nó trên màn hình chuẩn:

def printme( str ):

"Chuoi nay duoc truyen vao trong ham"

print str

return

Triệu hồi một hàm trong Python

Để thực thi một hàm, bạn cần gọi hàm đó. Phần định nghĩa hàm cung cấp thông tin về tên hàm các tham số và định nghĩa những hoạt động nào được thực hiện bởi hàm đó. Để thực thi phần định nghĩa của hàm, bạn cần gọi hàm đó. Cú pháp như sau:

**ten\_ham( cac\_tham\_so )**

Ví dụ sau minh họa cách gọi hàm printme():

# Phan dinh nghia ham o day

def printme( str ):

"Chuoi nay duoc truyen vao trong ham"

print str

return;

# Bay gio ban co the goi ham printme

printme("Loi goi dau tien toi custom func!")

printme("Loi goi thu hai toi custom func")

Khi code trên được thực thi sẽ cho kết quả:

Loi goi dau tien toi custom func!

Loi goi thu hai toi custom func

Hàm return() trong Python

Hàm return(bieu\_thuc) được sử dụng để gửi điều khiển quay trở lại người gọi với bieu\_thuc đã cho. Trong trường hợp không cung cấp bieu\_thuc, thì hàm return này sẽ trả về None. Nói cách khác, lệnh return được sử dụng để thoát khỏi định nghĩa hàm.

Các ví dụ trên không trả về bất cứ giá trị nào. Bạn có thể trả về một giá trị từ một hàm như sau:

# Phan dinh nghia ham o day

def sum( arg1, arg2 ):

# Cong hai tham so va tra ve ket qua."

total = arg1 + arg2

print "Ben trong ham : ", total

return total;

# Bay gio ban co the goi ham sum nay

total = sum( 10, 20 );

print "Ben ngoai ham : ", total

Kết quả là:

Ben trong ham : 30

Ben ngoai ham : 30

Phân biệt argument và parameter trong Python

Có hai kiểu dữ liệu được truyền trong hàm:

Kiểu dữ liệu đầu tiên là dữ liệu được truyền trong lời gọi hàm. Dữ liệu này được gọi là argument. Argument có thể là hằng, biến hoặc biểu thức.

Kiểu dữ liệu thứ hai là dữ liệu được nhận trong phần định nghĩa hàm. Dữ liệu này được gọi là parameter. Parameter phải là biến để giữ các giá trị đang đến.

Ví dụ:

def addition(x,y):

print x+y

x=15

addition(x ,10)

addition(x,x)

y=20

addition(x,y)

Kết quả là:

25

30

35

Với cách phân biệt trên, bạn có thể hiểu hơn về đâu là kiểu tham số bạn đang dùng, còn thực sự thì mình nghĩ nó cũng không quan trọng lắm. Do đó, trong phần tiếp theo, chúng tôi sẽ gọi chung chúng là tham số.

Truyền bởi tham chiếu vs bởi giá trị trong Python

Tất cả parameter (argument) trong Python được truyền bởi tham chiếu. Nghĩa là nếu bạn thay đổi những gì mà một parameter tham chiếu tới bên trong một hàm, thì thay đổi này cũng phản ánh trong hàm đang gọi. Ví dụ:

# Phan dinh nghia ham o day

def changeme( mylist ):

"Thay doi list da truyen cho ham nay"

mylist.append([1,2,3,4]);

print "Cac gia tri ben trong ham la: ", mylist

return

# Bay gio ban co the goi ham changeme function

mylist = [10,20,30];

changeme( mylist );

print "Cac gia tri ben ngoai ham la: ", mylist

Ở đây, chúng ta đang duy trì tham chiếu của đối tượng đang truyền và đang phụ thêm các giá trị trong cùng đối tượng đó. Vì thế đoạn code này sẽ cho kết quả:

Cac gia tri ben trong ham la: [10, 20, 30, [1, 2, 3, 4]]

Cac gia tri ben ngoai ham la: [10, 20, 30, [1, 2, 3, 4]]

Bạn theo dõi một ví dụ nữa, tại đây tham số đang được truyền bởi tham chiếu và tham chiếu đang được ghi đè bên trong hàm được gọi.

# Phan dinh nghia ham o day

def changeme( mylist ):

"Thay doi list da truyen cho ham nay"

mylist = [1,2,3,4]; # Lenh nay gan mot tham chieu moi cho mylist

print "Cac gia tri ben trong ham la: ", mylist

return

# Bay gio ban co the goi ham changeme

mylist = [10,20,30];

changeme( mylist );

print "Cac gia tri ben ngoai ham la: ", mylist

Tham số mylist là cục bộ (local) với hàm changeme. Việc thay đổi mylist bên trong hàm này không ảnh hưởng tới mylist. Và cuối cùng code trên sẽ cho kết quả:

Cac gia tri ben trong ham la: [1, 2, 3, 4]

Cac gia tri ben ngoai ham la: [10, 20, 30]

Phạm vi biến trong Python

Tất cả các biến trong một chương trình không phải là có thể truy cập tại tất cả vị trí ở trong chương trình đó. Điều này phụ thuộc vào nơi bạn đã khai báo một biến.

Phạm vi biến quyết định nơi nào của chương trình bạn có thể truy cập một định danh cụ thể. Trong Python, có hai khái niệm về phạm vi biến là:

**Biến toàn cục**

**Biến cục bộ**

**Biến cục bộ trong Python**

Các biến được khai báo bên trong một thân hàm là biến cục bộ. Tức là các biến cục bộ này chỉ có thể được truy cập ở bên trong hàm mà bạn đã khai báo, không thể được truy cập ở bên ngoài thân hàm đó. Ví dụ:

def msg():

a=10

print "Gia tri cua a la",a

return

msg()

print a #no se cho mot error vi bien la cuc bo

Kết quả sẽ cho một lỗi vì biến là cục bộ.

Gia tri cua a la 10

Traceback (most recent call last):

File "C:/Python27/lam.py", line 7, in >module<

print a #no se cho mot error vi bien la cuc bo

TenError: name 'a' is not defined

**Biến toàn cục trong Python**

Biến được định nghĩa bên ngoài hàm được gọi là biến toàn cục. Biến toàn cục có thể được truy cập bởi tất cả các hàm ở khắp nơi trong chương trình. Do đó phạm vi của biến toàn cục là rộng nhất. Ví dụ:

b=20

def msg():

a=10

print "Gia tri cua a la",a

print "Gia tri cua b la",b

return

msg()

print b

Kết quả là:

Gia tri cua a la 10

Gia tri cua b la 20

20

Tham số hàm trong Python

Sau khi đã tìm hiểu về phạm vi biến, tiếp theo chúng ta cùng tìm hiểu về các loại tham số hàm. Python hỗ trợ các kiểu tham số chính thức sau:

Tham số bắt buộc

Tham số mặc định

Tham số từ khóa (tham số được đặt tên)

Số tham số thay đổi

Tham số bắt buộc trong Python

Các tham số bắt buộc là các tham số được truyền tới một hàm theo một thứ tự chính xác. Ở đây, số tham số trong lời gọi hàm nên kết nối chính xác với phần định nghĩa hàm.

Bạn theo dõi ví dụ dưới đây:

#Phan dinh nghia cua ham sum

def sum(a,b):

"Ham co hai tham so"

c=a+b

print c

sum(10,20)

sum(20)

Khi code trên được thực thi sẽ cho một lỗi như sau:

30

Traceback (most recent call last):

File "C:/Python27/su.py", line 8, in >module<

sum(20)

TypeError: sum() takes exactly 2 arguments (1 given)

Giải thích:

1, Trong trường hợp đầu tiên, khi hàm sum() được gọi đã được truyền hai giá trị là 10 và 20, đầu tiên Python so khớp với phần định nghĩa hàm, sau đó 10 và 20 được gán tương ứng cho a và b. Do đó hàm sum được tính toán và được in.

2, Trong trường hợp thứ hai, khi bạn chỉ truyền cho hàm sum() một giá trị là 20, giá trị này được truyền tới phần định nghĩa hàm. Tuy nhiên phần định nghĩa hàm chấp nhận hai tham số trong khi chỉ có một giá trị được truyền, do đó sẽ tạo ra một lỗi như trên.

Tham số mặc định trong Python

Tham số mặc định là tham số mà cung cấp các giá trị mặc định cho các tham số được truyền trong phần định nghĩa hàm, trong trường hợp mà giá trị không được cung cấp trong lời gọi hàm. Ví dụ:

#Phan dinh nghia ham

def msg(Id,Ten,Age=21):

"In gia tri da truyen"

print Id

print Ten

print Tuoi

return

#Function call

msg(Id=100,Ten='Hoang',Tuoi=20)

msg(Id=101,Ten='Thanh')

Kết quả là:

100

Hoang

20

101

Thanh

21

Giải thích:

1, Trong trường hợp đầu tiên, khi hàm msg() được gọi đang truyền ba giá trị là 100, Hoang, và 20, thì các giá trị này sẽ được gán tương ứng cho các tham số và do đó chúng được in ra tương ứng.

2, Trong trường hợp thứ hai, khi bạn chỉ truyền hai tham số cho hàm msg() được gọi là 101 và Thanh, thì các giá trị này được gán tương ứng cho ID và Ten. Không có giá trị nào được gán cho tham số thứ ba trong lời gọi hàm, và vì thế hàm sẽ lấy giá trị mặc định là 21.

Tham số từ khóa trong Python

Sử dụng tham số từ khóa, tham số được truyền trong lời gọi hàm được kết nối với phần định nghĩa hàm dựa trên tên của tham số. Với trường hợp này, vị trí của các tham số trong lời gọi hàm là tùy ý. Ví dụ:

def msg(id,name):

"In gia tri da truyen"

print id

print ten

return

msg(id=100,ten='Hoang')

msg(ten='Thanh',id=101)

Kết quả là:

100

Hoang

101

Thanh

Giải thích:

1, Trong trường hợp đầu tiên, trong lời gọi hàm msg(), bạn đã truyền hai giá trị và truyền vị trí giống như của chúng trong phần định nghĩa hàm. Sau khi so khớp với phần định nghĩa hàm, thì các giá trị này được truyền tương ứng với các tham số trong phần định nghĩa hàm. Điều này được thực hiện dựa trên tên tham số.

2, Trong trường hợp thứ hai, trong lời gọi hàm msg(), bạn cũng cũng truyền hai giá trị nhưng với vị trí khác với phần định nghĩa hàm. Thì ở đây, dựa vào tên của tham số, các giá trị này cũng được truyền tương ứng cho các tham số trong phần định nghĩa hàm.

Hàm với số tham số thay đổi trong Python

Bạn có thể cần xử lý một hàm mà có số tham nhiều hơn là bạn đã xác định trong khi định nghĩa hàm. Những tham số này được gọi là các tham số có số tham số thay đổi (variable-length args) và không được đặt tên trong định nghĩa hàm, không giống như các tham số bắt buộc và tham số mặc định.

Cú pháp cho một hàm có số thay đổi là:

def tenham([tham\_so\_chinh\_thuc,] \*var\_args\_tuple ):

"function\_docstring"

function\_suite

return [bieu\_thuc]

Một dấu \* được đặt trước tên biến để giữ các giá trị của các tham số loại này. Tuple này vẫn là trống nếu không có tham số bổ sung nào được xác định trong khi gọi hàm. Dưới đây là ví dụ đơn giản.

# Phan dinh nghia ham o day

def printinfo( arg1, \*vartuple ):

"In mot tham so da truyen"

print "Ket qua la: "

print arg1

for var in vartuple:

print var

return;

# Bay gio ban co the goi ham printinfo

printinfo( 10 )

printinfo( 70, 60, 50 )

Kết quả là:

Ket qua la:

10

Ket qua la:

70

60

50

Hàm nặc danh trong Python

Hàm nặc danh (hàm vô danh), hiểu theo cách đơn giản, là hàm không có tên và chúng không được khai báo theo cách chính thức bởi từ khóa def. Để khai báo hàm này, bạn sử dụng từ khóa lambda. Lambda nhận bất kỳ lượng tham số nào và chỉ trả về một giá trị trong dạng một biểu thức đã được ước lượng. Bạn không thể gọi trực tiếp gọi hàm nặc danh để in bởi vì labda cần một biểu thức. Ngoài ra, các hàm lambda có namespace cục bộ của chúng. Dưới đây là cú pháp của hàm lamda:

lambda [arg1 [,arg2,.....argn]]:bieu\_thuc

Bạn theo dõi ví dụ sau để hiểu cách hàm lambda làm việc:

#Phan dinh nghia ham

square=lambda x1: x1\*x1

#Goi square nhu la mot ham

print "Binh phuong cua so la",square(10)

Kết quả là:

Binh phuong cua so la 100

So sánh hàm chính thức và hàm nặc danh trong Python?

Bạn theo dõi hai ví dụ sau:

Ví dụ cho hàm chính thức:

#Phan dinh nghia ham

def square(x):

return x\*x

#Goi ham square

print "Binh phuong cua so la",square(10)

Ví dụ cho hàm nặc danh:

#Phan dinh nghia ham

square=lambda x1: x1\*x1

#Goi square nhu la mot ham

print "Binh phuong cua so la",square(10)

Giải thích:

Hàm nặc danh được tạo bởi sử dụng từ khóa lambda, không phải bởi từ khóa def. Hàm này chỉ trả về một giá trị dưới dạng một biểu thức đã được ước lượng.

### File I/O

**In kết quả ra màn hình trong Python**

Đến đây, chắc bạn đã quá quen thuộc về cách sử dụng của lệnh print. Lệnh này được sử dụng để in kết quả trên màn hình. Hàm này chuyển đổi biểu thức mà bạn đã truyền cho nó thành dạng chuỗi và ghi kết quả trên đầu ra chuẩn Standard Output. Cú pháp của lệnh print là:

print "Hoc Python la kha don gian,", "ban co thay vay khong?"

Kết quả là:

Hoc Python la kha don gian, ban co thay vay khong?

Đọc input từ bàn phím trong Python

Python cung cấp hai hàm đã được xây dựng sẵn để nhận input từ người dùng. Hai hàm đó là:

**+ Hàm input()**

**+ Hàm raw\_input()**

**Hàm input() trong Python**

Hàm này được sử dụng để nhận input từ người dùng. Hàm này giống hàm raw\_input(), nhưng với hàm input() này thì bất cứ biểu thức nào được nhập từ người dùng thì nó ước lượng và sau đó trả về kết quả. Ví dụ:

str = input("Nhap dau vao cua ban: ");

print "Dau vao da nhan la : ", str

Code trên sẽ cho kết quả sau tùy thuộc vào input bạn đã nhập:

Nhap dau vao cua ban: [x\*5 for x in range(2,10,2)]

Dau vao da nhan la : [10, 20, 30, 40]

**Hàm raw\_input() trong Python**

Hàm raw\_input() được sử dụng để nhận đầu vào từ người dùng. Nó nhận đầu vào từ Standard Input dưới dạng một chuỗi và đọc dữ liệu từ từng dòng một. Ví dụ:

str = raw\_input("Nhap dau vao cua ban: ");

print "Dau vao da nhan la : ", str

Kết quả khi mình nhập "Hello Python!" là:

Nhap dau vao cua ban: Hello Python

Dau vao da nhan la : Hello Python

Ghi chú: Hàm raw\_input() trả về một chuỗi. Vì thế trong trường hợp một biểu thức cần được ước lượng, thì nó phải ép kiểu sang kiểu dữ liệu sau của nó. Bạn theo dõi một số ví dụ dưới đây.

Ví dụ 1:

prn=int(raw\_input("Trang VietJack"))

r=int(raw\_input("Thu Tu"))

t=int(raw\_input("Vi Tri"))

si=(prn\*r\*t)/100

print "Chao Ban ",si

Kết quả là:

Trang VietJack1000

Thu Tu10

Vi Tri2

Chao Ban 200

Ví dụ 2:

name=raw\_input("Nhap ten ban ")

math=float(raw\_input("Nhap diem mon Toan"))

physics=float(raw\_input("Nhap diem mon Vat Ly"))

chemistry=float(raw\_input("Nhap diem mon Hoa Hoc"))

rollno=int(raw\_input("Nhap mssv"))

print "Welcome ",name

print "MSSV cua ban la ",rollno

print "Diem mon Toan la ",math

print "Diem mon Vat Ly la ",physics

print "Diem mon Hoa Hoc la ",chemistry

print "Diem trung binh la ",(math+physics+chemistry)/3

Kết quả là:

Nhap ten ban Hoang

Nhap diem mon Toan7.68

Nhap diem mon Vat Ly7.14

Nhap diem mon Hoa Hoc8.84

Nhap mssv0987645672

Welcome Hoang

MSSV cua ban la 987645672

Diem mon Toan la 7.68

Diem mon Vat Ly la 7.14

Diem mon Hoa Hoc la 8.84

Diem trung binh la 7.8866666667

**Làm việc với File trong Python**

Python cung cấp nhiều cách tiện lợi để bạn làm việc với file. Ở trên, bạn đã đọc dữ liệu từ Standard Input và ghi dữ liệu tới Standard Output. Bây giờ chúng ta tìm hiểu cách sử dụng các file dữ liệu thực sự. Một file là một nơi lưu trữ ngoại vi trên hard disk, tại đó dữ liệu có thể được lưu trữ và thu nhận. Dưới đây là các hoạt động trên File:

**Mở file trong Python**

Trước khi làm việc với bất cứ File nào, bạn phải mở File đó. Để mở một File, Python cung cấp hàm open(). Nó trả về một đối tượng File mà được sử dụng với các hàm khác. Với File đã mở, bạn có thể thực hiện các hoạt động đọc, ghi, … trên File đó. Cú pháp của hàm open() là:

**doi\_tuong\_file = open(ten\_file [, access\_mode][, buffer])**

Ở đây,

ten\_file là tên File bạn muốn truy cập.

access\_mode xác định chế độ của File đã được mở. Có nhiều mode sẽ được trình bày trong phần dưới. Bạn nên xác định mode này phụ thuộc vào các hoạt động mà bạn muốn thực hiện trên File đó. Chế độ truy cập mặc định là read.

buffer Nếu buffer được thiết lập là 0, nghĩa là sẽ không có trình đệm nào diễn ra. Nếu xác định là 1, thì trình đệm dòng được thực hiện trong khi truy cập một File. Nếu là số nguyên lớn hơn 1, thì hoạt động đệm được thực hiện với kích cỡ bộ đệm đã cho. Nếu là số âm, thì kích cỡ bộ đệm sẽ là mặc định (hành vi mặc định).

**Đóng một File trong Python**

Khi bạn đã thực hiện xong các hoạt động trên File thì cuối cùng bạn cần đóng File đó. Python tự động đóng một File khi đối tượng tham chiếu của một File đã được tái gán cho một file khác. Tuy nhiên, sử dụng phương thức close() để đóng một file là một sự thực hành tốt cho bạn. Phương thức close() có cú pháp như sau:

**fileObject.close();**

**Đọc một File trong Python**

Để đọc một File, bạn sử dụng phương thức read() trong Python. Cú pháp là:

**doi\_tuong\_file.read(giatri);**

Ở đây, value là số byte để được đọc từ file đã mở. Phương thức này bắt đầu đọc từ phần đầu file và nếu bạn không cung cấp tham số value thì phương thức này cố gắng đọc nhiều dữ liệu nhất có thể, có thể tới cuối File.

**Ghi tới một File trong Python**

Phương thức write() được sử dụng để ghi bất kỳ chuỗi nào tới một File đã mở. Bạn chú ý là phương thức write này không thêm một ký tự newline (dòng mới) ('\n') vào cuối chuỗi. Cú pháp của write() là:

**doi\_tuong\_file.write(string);**

Dưới đây là chương trình ví dụ để đọc và ghi dữ liệu từ một File trong Python:

obj=open("abcd.txt","w")

obj.write("Python xin chao cac ban")

obj.close()

obj1=open("abcd.txt","r")

s=obj1.read()

print s

obj1.close()

obj2=open("abcd.txt","r")

s1=obj2.read(20)

print s1

obj2.close()

Kết quả là:

Python xin chao cac ban

Chao mung ban den voi the gioi Python

**Các thuộc tính của File trong Python**

Đối tượng File có các thuộc tính sau:

Thuộc tính Miêu tả

file.closed Trả về true nếu file đã được đóng, nếu không là false

file.mode Trả về chế độ truy cập nào mà file đã mở với

file.name Trả về tên file

file.softspace Trả về false nếu space được yêu cầu tường minh với print, nếu không là true

Ví dụ

Chúng ta tạo foo.txt có nội dung sau:

# Mo mot file

fo = open("foo.txt", "wb")

fo.write( "Python la mot ngon ngu lap trinh tuyet voi.\nMinh cung nghi nhu the!!\n");

# Dong file da mo

fo.close()

Giờ chúng ta kiểm tra các thuộc tính của nó:

# Mo mot file

fo = open("foo.txt", "wb")

print "Ten cua file la: ", fo.name

print "File da duoc dong hay chua : ", fo.closed

print "Che do mode la : ", fo.mode

print "Softspace la : ", fo.softspace

Kết quả là:

Ten cua file la: foo.txt

File da duoc dong hay chua : False

Che do mode la : wb

Softspace la : 0

Các chế độ truy cập (mode) của File trong Python

File có thể được mở với các chế độ truy cập khác nhau. File có thể được mở trong Text Mode hoặc Binary Mode. Bảng dưới liệt kê và giới thiệu các chế độ này:

Mode Miêu tả

r Mở file trong chế độ đọc, đây là chế độ mặc định. Con trỏ tại phần bắt đầu của File

rb Mở file trong chế độ đọc cho định dạng nhị phân, đây là chế độ mặc định. Con trỏ tại phần bắt đầu của File

r+ Mở file để đọc và ghi. Con trỏ tại phần bắt đầu của File

rb+ Mở file để đọc và ghi trong định dạng nhị phân. Con trỏ tại phần bắt đầu của File

w Mở File trong chế độ ghi. Nếu file đã tồn tại, thì ghi đè nội dung của file đó, nếu không thì tạo một file mới

wb Mở File trong chế độ ghi trong định dạng nhị phân. Nếu file đã tồn tại, thì ghi đè nội dung của file đó, nếu không thì tạo một file mới

w+ Mở file để đọc và ghi. Nếu file tồn tại thì ghi đè nội dung của nó, nếu file không tồn tại thì tạo một file mới để đọc và ghi

wb+ Mở file để đọc và ghi trong định dạng nhị phân. Nếu file tồn tại thì ghi đè nội dung của nó, nếu file không tồn tại thì tạo một file mới để đọc và ghi

a Mở file trong chế độ append. Con trỏ là ở cuối file nếu file này đã tồn tại. Nếu file không tồn tại, thì tạo một file mới để ghi

ab Mở file trong chế độ append trong chế độ nhị phân. Con trỏ là ở cuối file nếu file này đã tồn tại. Nếu file không tồn tại, thì tạo một file mới để ghi

a+ Mở file trong để đọc và append. Con trỏ file tại cuối nếu file đã tồn tại. Nếu không tồn tại thì tạo một file mới để đọc và ghi

ab+ Mở file trong để đọc và append trong định dạng nhị phân. Con trỏ file tại cuối nếu file đã tồn tại. Nếu không tồn tại thì tạo một file mới để đọc và ghi

**Thay tên file trong Python**

Phương thức **rename()** trong os Module được sử dụng để thay tên file. Phương thức này nhận hai tham số là tên file cũ và tên file mới.

Cú pháp

**os.rename(ten\_file\_hien\_tai, ten\_file\_moi)**

Ví dụ sau thay tên test1.txt thành test2.txt:

import os

# Thay ten tu test1.txt thanh test2.txt

os.rename( "test1.txt", "test2.txt" )

**Xóa file trong Python**

Bạn có thể sử dụng phương thức remove() của os Module để xóa các file với tham số là tên file bạn cần xóa.

Cú pháp

**os.remove(ten\_file)**

Ví dụ sau sẽ xóa test2.txt:

import os

# Xoa test2.txt

os.remove("text2.txt")

**Vị trí File trong Python**

Phương thức tell() nói cho bạn biết vị trí hiện tại bên trong file. Nói cách khác, việc đọc và ghi tiếp theo sẽ diễn ra trên các byte đó.

Phương **thức seek(offset[, from])** thay đổi vị trí hiện tại bên trong file. Tham số offset chỉ số byte để được di chuyển. Tham số from xác định vị trí tham chiếu mà từ đó byte được di chuyển.

Nếu from được thiết lập là 0 nghĩa là sử dụng phần đầu file như là vị trí tham chiếu và 1 nghĩa là sử dụng vị trí hiện tại như là vị trí tham chiếu và nếu là 2 thì sử dụng phần cuối file như là vị trí tham chiếu.

Ví dụ

Sử dụng foo.txt đã tạo ở trên để minh họa các hàm tell và seek:

# Mo mot file

fo = open("foo.txt", "r+")

str = fo.read(10);

print "Chuoi da doc la : ", str

# Kiem tra con tro hien tai

position = fo.tell();

print "Con tro file hien tai : ", position

# Dat lai vi tri con tro tai vi tri ban dau mot lan nua

position = fo.seek(0, 0);

str = fo.read(10);

print "Chuoi da doc la : ", str

# Dong file da mo

fo.close()

Kết quả là:

Chuoi da doc la : Python is

Con tro file hien tai : 10

Chuoi da doc la : Python la

**Thư mục trong Python**

Tất cả file được chứa trong các thư mục đa dạng và Python cũng cung cấp rất nhiều phương thức để xử lý các hoạt động đa dạng liên quan tới thư mục. os Module có một số phương thức giúp bạn tạo, xóa, và thay đổi các thư mục.

**Phương thức mkdir() trong Python**

Bạn có thể sử dụng phương thức mkdir() của os Module để tạo các thư mục trong thư mục hiện tại. Bạn cần cung cấp một tham số là tên thư mục cho phương thức này.

Cú pháp

**os.mkdir("thu\_muc\_moi")**

Ví dụ sau tạo một thư mục test trong thư mục hiện tại.

import os

# Tao mot thu muc la "test"

os.mkdir("test")

**Phương thức chdir() trong Python**

Bạn có thể sử dụng phương thức chdir() để thay đổi thư mục hiện tại. Phương thức chdir() nhận một tham số là tên của thư mục bạn muốn tới từ thư mục hiện tại.

Cú pháp

**os.chdir("thu\_muc\_moi")**

Ví dụ sau tới thư mục /home/newdir.

import os

# Thay doi mot thu muc toi "/home/newdir"

os.chdir("/home/newdir")

**Phương thức getcwd() trong Python**

Phương thức getcwd() hiển thị thư mục đang làm việc hiện tại.

Cú pháp

**os.getcwd()**

Ví dụ sau hiển thị thư mục đang làm việc hiện tại.

import os

# Lenh nay se cung cap vi tri thu muc hien tai

os.getcwd()

**Phương thức rmdir() trong Python**

Phương thức rmdir() xóa thư mục mà có tên được truyền như là một tham cố cho phương thức này.

Trước khi xóa thư mục, tất cả nội dung trong nó nên được xóa.

Cú pháp

**os.rmdir('ten\_thu\_muc')**

Ví dụ sau sẽ xóa thư mục /tmp/test. Bạn phải cung cấp tên đầy đủ của thư mục, nếu không phương thức này sẽ không tìm thấy thư mục đó và sẽ không có hoạt động xóa diễn ra.

import os

# Xoa thu muc "/tmp/test" .

os.rmdir( "/tmp/test" )

Các phương thức xử lý File và thư mục trong Python

Đối tượng File và OS cung cấp rất nhiều phương thức tiện ích để xử lý và thao tác với File và thư mục trên hệ điều hành Windows và Unix. Bạn truy cập đường link sau để tìm hiểu các phương thức này.

Đối tượng File: cung cấp các phương thức để thao tác File.

os Module: cung cấp rất nhiều phương thức để thao tác File và thư mục.

## II.6. Xử lí ngoại lệ

**Exception là gì?**

Ngoại lệ có thể là bất kỳ điều kiện bất thường nào trong chương trình mà phá vỡ luồng thực thi chương trình đó. Bất cứ khi nào một ngoại lệ xuất hiện, mà không được xử lý, thì chương trình ngừng thực thi và vì thế code không được thực thi.

Python đã định nghĩa sẵn rất nhiều ngoại lệ, mà đã được trình bày trong chương Standard Exception. Trong chương này chúng ta sẽ tìm hiểu cách xử lý ngoại lệ cũng như cách tạo các Custom Exception như thế nào.

**Xử lý ngoại lệ trong Python**

Nếu bạn thấy bất cứ code nào là khả nghi (có thể gây ra ngoại lệ) thì bạn có thể phòng thủ chương trình của mình bằng cách đặt các khối code khả nghi này trong một khối try. Khối try này được theo sau bởi lệnh except. Sau đó, nó được theo sau bởi các lệnh mà xử lý vấn đề đó.

Cú pháp

Dưới đây là cú pháp của khối try....except...else trong Python:

try:

Ban thuc hien cac hoat dong cua minh tai day;

Va day la phan code co the tao exception;

......................

except ExceptionI:

Neu co ExceptionI, thi thuc thi khoi code nay

except ExceptionII:

Neu co ExceptionII, thi thuc thi khoi code nay

......................

else:

Neu khong co exception nao thi thuc thi khoi code nay

Dưới đây là một số điểm bạn cần lưu ý:

Phần code khả nghi mà có khả năng tạo exception cần được bao quanh trong khối try.

Khối try được theo sau bởi lệnh except. Có thể có một hoặc nhiều lệnh except với một khối try đơn.

Lệnh except xác định exception mà xảy ra. Trong trường hợp mà exception đó xảy ra, thì lệnh tương ứng được thực thi.

Ở cuối khối try, bạn có thể cung cấp lệnh else. Nó được thực thi khi không có exception nào xảy ra. Khối else là địa điểm tốt cho code mà không cần sự bảo vệ của khối try.

Ví dụ

Ví dụ sau mở một file, ghi nội dung vào file này và sau đó đóng file, tất cả hoạt động đều thành công:

try:

fh = open("testfile", "w")

fh.write("Day la mot kiem tra nho ve xu ly ngoai le!!")

except IOError:

print "Error: Khong tim thay file"

else:

print "Thanh cong ghi noi dung vao file"

fh.close()

Kết quả là:

Thanh cong ghi noi dung vao file

Ví dụ

Ví dụ sau mở một file để ghi trong khi bạn không có quyền ghi, do đó nó sẽ tạo một exception:

try:

fh = open("testfile", "r")

fh.write("Day la mot kiem tra nho ve xu ly ngoai le!!")

except IOError:

print "Error: Khong tim thay file"

else:

print "Thanh cong ghi noi dung vao file"

Kết quả là:

Error: Khong tim thay file

Mệnh đề except mà không xác định Exception trong Python

Lệnh except cũng có thể được sử dụng mà không xác định exception nào. Lệnh try-except này bắt tất cả exception mà xuất hiện. Sử dụng loại lệnh try-except này không phải là sự thực hành lập trình tốt, bởi vì nó bắt tất cả exception nhưng không làm cho lập trình viên biết được căn nguyên của vấn đề làm xuất hiện exception đó.

Cú pháp

try:

Ban thuc hien cac hoat dong cua minh tai day;

Va day la phan code co the tao exception;

......................

except:

Neu co bat ky exception nao, thi thuc thi khoi code nay

......................

else:

Neu khong co exception nao, thi thuc thi khoi code nay

Mệnh đề except với nhiều exception trong Python

Sử dụng cùng lệnh except như trên, bạn có thể khai báo nhiều exception như sau:

try:

Ban thuc hien cac hoat dong cua minh tai day;

Va day la phan code co the tao exception;

......................

except(Exception1[, Exception2[,...ExceptionN]]]):

Neu co bat ky exeption nao trong danh sach,

thi thuc thi khoi code nay

......................

else:

Neu khong co exception nao, thi thuc thi khoi code nay

Khối try-finally trong Python

Trong trường hợp nếu có bất kỳ code nào mà người dùng muốn được thực thi, dù cho có xuất hiện exception hay không thì khối code đó có thể được đặt trong khối finally. Khối finally sẽ luôn luôn được thực thi bất chấp có hay không exception. Cú pháp của khối try-finally là:

try:

Ban thuc hien cac hoat dong cua minh tai day;

Va day la phan code co the tao exception;

......................

Do co exeption nen khoi nay bi bo qua

finally:

Khoi nay nen duoc thuc thi

......................

Ghi chú: Bạn có thể cung cấp một hoặc nhiều mệnh đề except, hoặc một mệnh đề finally, nhưng không được cung cấp cả hai. Ngoài ra bạn cũng không thể sử dụng mệnh đề else với một mệnh đề finally.

Ví dụ

try:

fh = open("testfile", "w")

fh.write("Day la mot kiem tra nho ve xu ly ngoai le!!")

finally:

print "Error: Khong tim thay file"

Nếu bạn không có quyền mở file trong chế độ ghi, thì code trên sẽ cho kết quả:

Error: Khong tim thay file

Ví dụ trên có thể được viết rõ ràng hơn như sau:

try:

fh = open("testfile", "w")

try:

fh.write("Day la mot kiem tra nho ve xu ly ngoai le!!")

finally:

print "Chuan bi dong file"

fh.close()

except IOError:

print "Error: Khong tim thay file"

Khi một exception được ném trong khối try, thì trình thực thi ngay lập tức truyền tới khối finally. Sau đó tất cả các lệnh trong khối finally được thực thi, exception được tạo lại lần nữa và được xử lý bởi lệnh except nếu có mặt trong lớp trên tiếp theo của lệnh try-except.

Tham số của một Exception trong Python

Một Exception có thể có một tham số, mà là một giá trị mà cung cấp thông tin bổ sung về vấn đề. Nội dung của tham số là đa dạng tùy vào các exception. Dưới đây là cú pháp:

try:

Ban thuc hien cac hoat dong cua minh tai day;

Va day la phan code co the tao exception;

......................

except Kieu\_exception, Tham\_so:

Ban co the in gia tri cua Tham\_so tai day...

Nếu bạn viết code trên để xử lý một exception đơn, bạn có thể có một biến theo sau tên của exception trong lệnh except. Nếu bạn đang khai báo nhiều exception thì bạn có một biến theo sau tuple của các exception đó.

Ví dụ

Ví dụ của một exception đơn:

# Dinh nghia mot ham o day.

def temp\_convert(var):

try:

return int(var)

except ValueError, Argument:

print "Tham so khong chua cac so\n", Argument

# Goi ham tren.

temp\_convert("xyz");

Tạo một Exception trong Python

Bạn có thể ném tường minh một Exception trong Python bởi sử dụng lệnh raise. Cú pháp của lệnh raise như sau:

raise [Exception [, args ]]

Ở đây, Lop\_Exception là kiểu của exception và tham số value (tùy ý) là một giá trị. Để truy cập giá trị này thì từ khóa as được sử dụng.

Ví dụ

try:

a=10

print a

raise NameError("Hello")

except NameError as e:

print "Mot Exception xuat hien"

print e

Trong ví dụ trên, e được sử dụng như là một biến tham chiếu mà lưu trữ giá trị của exception.

Ghi chú: để bắt một exception, một mệnh đề except phải tham chiếu tới cùng exception đã được ném. Ví dụ, để bắt exception trên, chúng ta phải viết mệnh đề except như sau:

try:

Ban thuc hien cac hoat dong cua minh tai day;

Va day la phan code co the tao exception;

...

except "Invalid level!":

Xu ly ngoai le o day...

else:

Phan con lai cua code...

Custom Exception trong Python

Python cho phép bạn tạo riêng cho mình các exception bằng cách kế thừa các lớp từ các Standard Exception.

Ví dụ dưới đây liên quan tới RuntimeError. Ở đây, một lớp đã tạo là lớp con của của RuntimeError. Trong khối try, exception được định nghĩa bởi người dùng được tạo và được bắt trong khối except. Biến e được sử dụng để tạo một instance của lớp Networkerror.

class Networkerror(RuntimeError):

def \_\_init\_\_(self, arg):

self.args = arg

Sau khi định nghĩa lớp trên, bạn có thể tạo exception như sau:

try:

raise Networkerror("Bad hostname")

except Networkerror,e:

print e.args

# III. Python nâng cao

## III.1, Hướng đối tượng

Python là một ngôn ngữ lập trình hướng đối tượng. Do đó với những bạn đã học qua C++ chắc rằng đã khá quen thuộc với các khái niệm về hướng đối tượng này. Chương này sẽ trình bày sơ qua về các thuật ngữ liên quan đến hướng đối tượng cùng với các ví dụ minh họa giúp bạn dễ hiểu hơn về vấn đề đã trình bày.

**Tạo các lớp trong Python**

Trong Python, lệnh class được sử dụng để tạo một lớp mới. Tên của lớp theo ngay sau từ khóa class và được theo sau bởi dấu hai chấm, như sau:

class TenLop:

'Phan documentation string cho lop la tuy y'

class\_suite

Lớp có một Documentation String mà có thể được truy cập thông qua TenLop.\_\_doc\_\_.

class\_suite gồm tất cả các lệnh thành phần mà định nghĩa các thành viên lớp, cấu trúc dữ liệu và hàm.

Dưới đây là ví dụ đơn giản về một lớp trong Python:

class Sinhvien:

'Class co so chung cho tat ca sinh vien'

svCount = 0

def \_\_init\_\_(self, ten, hocphi):

self.ten = ten

self.hocphi = hocphi

Sinhvien.svCount += 1

def displayCount(self):

print "Tong so Sinh vien %d" % Sinhvien.svCount

def displaySinhvien(self):

print "Ten : ", self.ten, ", Hoc phi: ", self.hocphi

Biến svCount là một biến class có giá trị được chia sẻ trong tất cả instance của lớp Sinhvien này. Biến này có thể được truy cập dưới dạng Sinhvien.svCount từ bên trong lớp hoặc bên ngoài lớp.

**Phương thức đầu tiên \_\_init\_\_()** là một phương thức đặc biệt, là constructor của lớp hoặc phương thức khởi tạo mà Python gọi khi bạn tạo một instance mới của lớp này.

Bạn khai báo các phương thức khác như các hàm thông thường với exception là tham số đầu tiên cho mỗi phương thức là self. Python thêm tham số self tới List cho bạn; bạn không cần bao nó khi bạn gọi các phương thức.

**Tạo Instance trong Python**

Để tạo các instance của một lớp, bạn gọi lớp này bởi sử dụng tên lớp và truyền vào bất kỳ tham số nào mà phương thức \_\_init\_\_ của nó chấp nhận. Bạn theo dõi ví dụ sau:

"Lenh nay tao doi tuong dau tien cua lop Sinhvien"

sv1 = Sinhvien("Hoang", 4000000)

"Lenh nay tao doi tuong thu hai cua lop Sinhvien"

sv2 = Sinhvien("Huong", 4500000)

Truy cập các thuộc tính trong Python

Bạn truy cập các thuộc tính của đối tượng bởi sử dụng toán tử dot (dấu chấm) với đối tượng. Biến class sẽ được truy cập bởi sử dụng tên lớp như sau:

sv1.displaySinhvien()

sv2.displaySinhvien()

print "Tong so Sinh vien %d" % Sinhvien.svCount

Bây giờ đặt tất cả khái niệm cùng với nhau:

class Sinhvien:

'Class co so chung cho tat ca sinh vien'

svCount = 0

def \_\_init\_\_(self, ten, hocphi):

self.ten = ten

self.hocphi = hocphi

Sinhvien.svCount += 1

def displayCount(self):

print "Tong so Sinh vien %d" % Sinhvien.svCount

def displaySinhvien(self):

print "Ten : ", self.ten, ", Hoc phi: ", self.hocphi

"Lenh nay tao doi tuong dau tien cua lop Sinhvien"

sv1 = Sinhvien("Hoang", 4000000)

"Lenh nay tao doi tuong thu hai cua lop Sinhvien"

sv2 = Sinhvien("Huong", 4500000)

sv1.displaySinhvien()

sv2.displaySinhvien()

print "Tong so Sinh vien %d" % Sinhvien.svCount

Khi code trên được thực thi sẽ cho kết quả:

Ten : Hoang ,Hoc phi: 4000000

Ten : Huong ,Hoc phi: 4500000

Tong so Sinh vien 2

Bạn có thể thêm, xóa, hoặc sửa đổi các thuộc tính của các lớp và đối tượng tại bất cứ thời điểm nào.

sv1.tuoi = 21 # Them mot thuoc tinh 'tuoi'.

sv1.tuoi = 20 # Sua doi thuoc tinh 'tuoi'.

del sv1.tuoi # Xoa thuoc tinh 'tuoi'.

Thay vì sử dụng các lệnh chính thức để truy cập các thuộc tính, bạn có thể sử dụng các hàm sau:

Hàm getattr(obj, name[, default]) : Để truy cập thuộc tính của đối tượng.

Hàm hasattr(obj,name) : Để kiểm tra xem một thuộc tính có tồn tại hay không.

Hàm setattr(obj,name,value) : Để thiết lập một thuộc tính. Nếu thuộc tính không tồn tại, thì nó sẽ được tạo.

Hàm delattr(obj, name) : Để xóa một thuộc tính.

Ví dụ:

hasattr(sv1, 'tuoi') # Tra ve true neu thuoc tinh 'tuoi' ton tai

getattr(sv1, 'tuoi') # Tra ve gia tri cua thuoc tinh 'tuoi'

setattr(sv1, 'tuoi', 20) # Thiet lap thuoc tinh 'tuoi' la 20

delattr(sv1, 'tuoi') # Xoa thuoc tinh 'tuoi'

**Các thuộc tính đã có sẵn cho lớp trong Python**

Mỗi lớp Python đều giữ các thuộc tính đã được xây dựng sẵn sau và chúng có thể được truy cập bởi sử dụng toán tử dot (dấu chấm .) như bất kỳ thuộc tính khác:

\_\_dict\_\_: Là Dictionary chứa namespace của lớp.

\_\_doc\_\_: Được sử dụng để truy cập Documentation String của lớp nếu có.

\_\_name\_\_: Là tên lớp.

\_\_module\_\_: Là tên Module trong đó lớp được định nghĩa. Thuộc tính là \_\_main\_\_ trong chế độ tương tác.

\_\_bases\_\_: Là một Tuple chứa các lớp cơ sở.

Với lớp Sinhvien trên, chúng ta sẽ thử truy cập tất cả các thuộc tính này.

class Sinhvien:

'Class co so chung cho tat ca sinh vien'

svCount = 0

def \_\_init\_\_(self, ten, hocphi):

self.ten = ten

self.hocphi = hocphi

Sinhvien.svCount += 1

def displayCount(self):

print "Tong so Sinh vien %d" % Sinhvien.svCount

def displaySinhvien(self):

print "Ten : ", self.ten, ", Hoc phi: ", self.hocphi

print "Sinhvien.\_\_doc\_\_:", Sinhvien.\_\_doc\_\_

print "Sinhvien.\_\_name\_\_:", Sinhvien.\_\_name\_\_

print "Sinhvien.\_\_module\_\_:", Sinhvien.\_\_module\_\_

print "Sinhvien.\_\_bases\_\_:", Sinhvien.\_\_bases\_\_

print "Sinhvien.\_\_dict\_\_:", Sinhvien.\_\_dict\_\_

Khi code trên được thực thi sẽ cho kết quả:

Sinhvien.\_\_doc\_\_: Class co so chung cho tat ca sinh vien

Sinhvien.\_\_name\_\_: Sinhvien

Sinhvien.\_\_module\_\_: \_\_main\_\_

Sinhvien.\_\_bases\_\_: ()

Sinhvien.\_\_dict\_\_: {'\_\_module\_\_': '\_\_main\_\_', 'displayCount':

<function displayCount at 0xb7c84994>, 'svCount': 2,

'displaySinhvien': <function displaySinhvien at 0xb7c8441c>,

'\_\_doc\_\_': 'Class co so chung cho tat ca sinh vien',

'\_\_init\_\_': <function \_\_init\_\_ at 0xb7c846bc>}

**Hủy đối tượng (Trình dọn rác) trong Python**

Python sẽ hủy các đối tượng mà không cần đến nữa (các kiểu đã được xây dựng sẵn hoặc instance của lớp) một cách tự động để giải phóng không gian bộ nhớ. Tiến trình này được gọi là Garbage Collection được thực hiện bởi trình dọn rác Garbage Collector.

Trình dọn rác của Python chạy trong khi thực thi chương trình và được kích hoạt khi số tham chiếu của một đối tượng tiến về 0. Số tham chiếu của một đối tượng thay đổi khi số alias mà trỏ tới nó thay đổi.

Số tham chiếu của một đối tượng tăng khi nó được gán một tên mới hoặc được đặt trong một container (chẳng hạn như List, Tuple, Dictionary). Số tham chiếu của một đối tượng giảm khi nó bị xóa với lệnh del, tham chiếu của nó được tái gán, hoặc tham chiếu của nó thoát ra khỏi phạm vi. Khi số tham chiếu của một đối tượng tiến về 0, thì Python thu thập nó một cách tự động. Ví dụ:

a = 40 # Tao doi tuong <40>

b = a # Tang so tham chieu cua <40>

c = [b] # Tang so tham chieu cua <40>

del a # Giam so tham chieu cua <40>

b = 100 # Giam so tham chieu cua <40>

c[0] = -1 # Giam so tham chieu cua <40>

Thường thì bạn sẽ không chú ý khi trình dọn rác hủy một instance và giải phóng bộ nhớ. Nhưng một lớp có thể triển khai phương thức đặc biệt là \_\_del\_\_(), được gọi là một destructor, mà được triệu hồi khi instance là chuẩn bị được hủy. Phương thức này có thể được sử dụng để xóa bất kỳ nguồn bộ nhớ nào được sử dụng bởi một instance.

Ví dụ

\_\_del\_\_() destructor này in tên lớp của một instance mà chuẩn bị được hủy.

class Point:

def \_\_init( self, x=0, y=0):

self.x = x

self.y = y

def \_\_del\_\_(self):

class\_name = self.\_\_class\_\_.\_\_name\_\_

print class\_name, "destroyed"

pt1 = Point()

pt2 = pt1

pt3 = pt1

print id(pt1), id(pt2), id(pt3) # in id cua doi tuong

del pt1

del pt2

del pt3

Khi code trên được thực thi sẽ cho kết quả sau:

3083401324 3083401324 3083401324

Point destroyed

Ghi chú: Một cách lý tưởng nhất là bạn nên định nghĩa các lớp của bạn trong file riêng biệt, sau đó bạn nên import chúng trong file chương trình chính bởi sử dụng lệnh import.

**Kế thừa lớp trong Python**

Thay vì bắt đầu viết code cho một lớp mới, bạn có thể tạo một lớp bằng việc kế thừa nó từ một lớp đã tồn tại trước đó bằng cách liệt kê lớp cha trong cặp dấu ngoặc đơn sau tên lớp mới.

Lớp con kế thừa các thuộc tính của lớp cha của nó, và bạn có thể sử dụng các thuộc tính như thể là chúng đã được định nghĩa trong lớp con đó. Một lớp con cũng có thể ghi đè các thành viên dữ liệu và các phương thức từ lớp cha.

Cú pháp

Các lớp kế thừa được khai báo khá giống như lớp cha của nó; tuy nhiên, một danh sách lớp cơ sở để kế thừa từ đó được cung cấp sau tên lớp mới.

class Tenlopcon (LopCha1[, LopCha2, ...]):

'Phan documentation string cua Class la tuy y'

class\_suite

Ví dụ

class Parent: # dinh nghia lop cha

parentAttr = 100

def \_\_init\_\_(self):

print "Goi constructor cua lop cha"

def parentMethod(self):

print 'Goi phuong thuc cua lop cha'

def setAttr(self, attr):

Parent.parentAttr = attr

def getAttr(self):

print "Thuoc tinh cua lop cha :", Parent.parentAttr

class Child(Parent): # dinh nghia lop con

def \_\_init\_\_(self):

print "Goi constructor cua lop con"

def childMethod(self):

print 'Goi phuong thuc cua lop con'

c = Child() # instance cua lop con

c.childMethod() # lop con goi phuong thuc cua no

c.parentMethod() # goi phuong thuc cua lop cha

c.setAttr(200) # tiep tuc goi phuong thuc cua lop cha

c.getAttr() # tiep tuc goi phuong thuc cua lop cha

Khi code trên được thực thi sẽ cho kết quả sau:

Goi constructor cua lop con

Goi phuong thuc cua lop con

Goi phuong thuc cua lop cha

Thuoc tinh cua lop cha : 200

Theo cách tương tự, bạn có thể kế thừa một lớp từ nhiều lớp cha như sau:

class A: # dinh nghia lop A

.....

class B: # dinh nghia lop B

.....

class C(A, B): # lop con cua A va B

.....

Bạn có thể sử dụng các hàm issubclass() hoặc isinstance() để kiểm tra mối quan hệ của hai lớp và instance.

Hàm issubclass(sub, sup) trả về true nếu lớp con sub đã cho thực sự là lớp con của lớp cha sup.

Hàm isinstance(obj, Class) trả về true nếu obj là một instance của lớp Class hoặc là một instance của lớp con của Class.

Ghi đè phương thức trong Python

Bạn có thể ghi đè các phương thức của lớp cha. Một trong các lý do để thực hiện việc ghi đè phương thức của lớp cha là bạn muốn có tính năng khác biệt hoặc đặc biệt trong lớp con.

Ví dụ

class Parent: # dinh nghia lop cha

def myMethod(self):

print 'Goi phuong thuc cua lop cha'

class Child(Parent): # dinh nghia lop con

def myMethod(self):

print 'Goi phuong thuc cua lop con'

c = Child() # instance cua lop con

c.myMethod() # lop con goi phuong thuc duoc ghi de

Kết quả là:

Goi phuong thuc cua lop con

**Nạp chồng phương thức trong Python**

Bảng dưới đây liệt kê một số tính năng chung mà bạn có thể ghi đè trong các lớp riêng của bạn.

STT Phương thức, Miêu tả và Lời gọi mẫu

1 \_\_init\_\_ ( self [,args...] )

Là constructor (với bất kỳ tham số tùy ý nào)

Lời gọi mẫu : obj = tenLop(args)

2 \_\_del\_\_( self )

Là destructor, xóa một đối tượng

Lời gọi mẫu : del obj

3 \_\_repr\_\_( self )

Biểu diễn chuỗi có thể ước lượng

Lời gọi mẫu : repr(obj)

4 \_\_str\_\_( self )

Biểu diễn chuỗi có thể in được

Lời gọi mẫu : str(obj)

5 \_\_cmp\_\_ ( self, x )

So sánh đối tượng

Lời gọi mẫu : cmp(obj, x)

Nạp chồng toán tử trong Python

Giả sử bạn đã tạo một lớp Vector để biểu diễn các vector hai chiều. Điều gì xảy ra khi bạn sử dụng toán tử cộng (+) để cộng chúng? Có thể nói vui rằng, lúc đó Python sẽ la hét vào mặt bạn.

Tuy nhiên, bạn có thể định nghĩa phương thức \_\_add\_ trong lớp của bạn để thực hiện phép cộng vector và sau đó phép cộng vector sẽ vận hành như bạn mong đợi.

Ví dụ

class Vector:

def \_\_init\_\_(self, a, b):

self.a = a

self.b = b

def \_\_str\_\_(self):

return 'Vector (%d, %d)' % (self.a, self.b)

def \_\_add\_\_(self,other):

return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(2,10)

v2 = Vector(5,-2)

print v1 + v2

Kết quả là:

Vector(7,8)

Ẩn dữ liệu (Data Hiding) trong Python

Các thuộc tính của một đối tượng có thể hoặc không thể là nhìn thấy với bên ngoài phần định nghĩa lớp. Bạn cần đặc tên các thuộc tính với một tiền tố là hai dấu gạch dưới, và sau đó các thuộc tính này sẽ không là nhìn thấy với bên ngoài.

Ví dụ

class JustCounter:

\_\_secretCount = 0

def count(self):

self.\_\_secretCount += 1

print self.\_\_secretCount

counter = JustCounter()

counter.count()

counter.count()

print counter.\_\_secretCount

Kết quả là:

1

2

Traceback (most recent call last):

File "test.py", line 12, in <module>

print counter.\_\_secretCount

AttributeError: JustCounter instance has no attribute '\_\_secretCount'

Python bảo vệ các thành viên đó bằng cách thay đổi nội tại tên để bao tên lớp. Bạn có thể truy cập các thuộc tính này dưới dạng như doi\_tuong.\_tenLop\_\_tenThuocTinh. Nếu bạn thay thế dòng cuối cùng như sau, thì nó sẽ làm việc cho bạn.

.........................

print counter.\_JustCounter\_\_secretCount

Kết quả là:

1

2

2

## III.2. Truy cập Databases

Chương này sẽ giới thiệu khái quát cho bạn về cách truy cập Database bởi sử dụng Python và giới thiệu qua về một số hoạt động cơ bản trên Database như INSERT, UPDATE, DELETE ...

Trước khi theo dõi và thực hành chương này, bạn cần phải tải một DB API Module riêng cho mỗi Database bạn cần truy cập. DB API cung cấp một chuẩn tối thiểu để làm việc với Database bởi sử dụng Python. Trong chương này, chúng ta sẽ sử dụng MySQL, vì thế trước hết chúng ta hãy tìm hiểu qua về MySQLdb Module.

MySQLdb là gì?

MySQLdb là một Interface để kết nối tới một MySQL Database Server từ Python. Nó triển khai Python Database API 2.0 và được xây dựng trên cùng của MySQL C API.

Cách cài đặt MySQLdb?

Trước khi tiến hành, bạn cần cài đặt MySQL trên thiết bị của bạn. Sau đó gõ dòng Python script sau và thực thi nó:

import MySQLdb

Nếu nó cho kết quả sau, thì nghĩa là MySQLdb Module đã không được cài đặt:

Traceback (most recent call last):

File "test.py", line 3, in <module>

import MySQLdb

ImportError: No module named MySQLdb

Để cài đặt MySQLdb Module, tải nó từ MySQLdb Download và tiến hành như sau:

$ gunzip MySQL-python-1.2.2.tar.gz

$ tar -xvf MySQL-python-1.2.2.tar

$ cd MySQL-python-1.2.2

$ python setup.py build

$ python setup.py install

Kết nối Database trong Python

Trước khi kết nối với một MySQL Database, đảm bảo:

Bạn đã tạo một Database có tên là TESTDB.

Bạn đã tạo một bảng là SINHVIEN trong TESTDB.

Bảng này có các trường HO, TEN, TUOI, GIOITINH và HOCPHI

User ID là testuser và password là test123 được thiết lập để truy cập TESTDB.

MySQLdb Module được cài đặt phù hợp trên thiết bị của bạn.

Dưới đây là ví dụ về kết nối với TESTDB.

import MySQLdb

# mo ket noi toi Database

db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# chuan bi mot doi tuong cursor boi su dung phuong thuc cursor()

cursor = db.cursor()

# Thuc thi truy van SQL boi su dung phuong thuc execute().

cursor.execute("SELECT VERSION()")

# Lay mot hang boi su dung phuong thuc fetchone().

data = cursor.fetchone()

print "Database version : %s " % data

# ngat ket noi voi server

db.close()

Khi chạy script này, nó sẽ cho kết quả sau trên thiết bị Linux.

Database version : 5.0.45

Nếu một kết nối được thành lập, thì một đối tượng Connection được trả về và được lưu giữ vào trong db, nếu không db được thiết lập là None. Tiếp đó, đối tượng db được sử dụng để tạo đối tượng cursor, mà tiếp đó được sử dụng để thực thi các truy vấn SQL. Cuối cùng, trước khi thoát ra, nó bảo đảm rằng kết nối tới Database được đóng và các resource được giải phóng.

Tạo bảng dữ liệu trong Python

Khi một kết nối tới Database đã được thành lập, chúng ta có thể tạo các bảng hoặc bản ghi vào trong bảng đó bởi sử dụng phương thức execute của đối tượng cursor đã được tạo.

Bạn theo dõi ví dụ để tạo bảng SINHVIEN:

import MySQLdb

# mo ket noi toi Database

db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# chuan bi mot doi tuong cursor boi su dung phuong thuc cursor()

cursor = db.cursor()

# Xoa bang neu no da ton tai boi su dung phuong thuc execute().

cursor.execute("DROP TABLE IF EXISTS SINHVIEN")

# Tao mot bang

sql = """CREATE TABLE SINHVIEN (

HO CHAR(20) NOT NULL,

TEN CHAR(20),

TUOI INT,

GIOITINH CHAR(1),

HOCPHI FLOAT )"""

cursor.execute(sql)

# ngat ket noi voi server

db.close()

Hoạt động INSERT trong Python

Đây là hoạt động bắt buộc khi bạn muốn tạo các bản ghi vào trong bảng đã tạo.

Ví dụ sau sẽ thực thi lệnh SQL INSERT để tạo một bản ghi vào trong bảng SINHVIEN.

import MySQLdb

# mo ket noi toi Database

db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# chuan bi mot doi tuong cursor boi su dung phuong thuc cursor()

cursor = db.cursor()

# Truy van SQL de INSERT mot ban ghi vao trong database.

sql = """INSERT INTO SINHVIEN(HO,

TEN, TUOI, GIOITINH, HOCPHI)

VALUES ('Nguyen', 'Hoang', 20, 'M', 4000000)"""

try:

# Thuc thi lenh SQL

cursor.execute(sql)

# Commit cac thay doi vao trong Database

db.commit()

except:

# Rollback trong tinh huong co bat ky error nao

db.rollback()

# ngat ket noi voi server

db.close()

Ví dụ trên có thể được viết như sau để tạo các truy vấn SQL hay hơn:

import MySQLdb

# mo ket noi toi Database

db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# chuan bi mot doi tuong cursor boi su dung phuong thuc cursor()

cursor = db.cursor()

# Truy van SQL de INSERT mot ban ghi vao trong database.

sql = "INSERT INTO SINHVIEN(HO, \

TEN, TUOI, GIOITINH, HOCPHI) \

VALUES ('%s', '%s', '%d', '%c', '%d' )" % \

('Nguyen', 'Hoang', 20, 'M', 4000000)

try:

# Thuc thi lenh SQL

cursor.execute(sql)

# Commit cac thay doi vao trong Database

db.commit()

except:

# Rollback trong tinh huong co bat ky error nao

db.rollback()

# ngat ket noi voi server

db.close()

Ví dụ

Đoạn code sau là form thực thi khác, tại đây bạn có thể truyền các tham số một cách trực tiếp.

..................................

user\_id = "test123"

password = "password"

con.execute('chen cac gia tri de dang nhap ("%s", "%s")' % \

(user\_id, password))

..................................

Hoạt động đọc trong Python

Hoạt động đọc trên bất cứ Database nào nghĩa là lấy một số thông tin hữu ích từ Database.

Khi kết nối với Databse được thiết lập, bạn có thể tạo một truy vấn vào trong Database này. Bạn có thể sử dụng hoặc phương thức fetchone() để lấy bản ghi đơn hoặc phương thức fetchall() để lấy nhiều giá trị từ một bảng.

Hàm fetchone() lấy hàng tiếp theo của một tập kết quả truy vấn. Một tập kết quả là một đối tượng được trả về khi một đối tượng cursor được sử dụng để truy vấn một bảng.

Hàm fetchall() lấy tất cả các hàng trong một tập kết quả. Nếu một số hàng đã sẵn sàng được trích từ tập kết quả đó, thì nó thu nhận các hàng còn lại từ tập kết quả.

Thuộc tính rowcount là một thuộc tính read-only và trả về số hàng đã bị ảnh hưởng bởi phương thức excute().

Ví dụ sau truy vấn tất cả bản ghi từ bảng SINHVIEN mà có salary lớn hơn 1000:

import MySQLdb

# mo ket noi toi Database

db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# chuan bi mot doi tuong cursor boi su dung phuong thuc cursor()

cursor = db.cursor()

# Chuan bi truy van SQl de INSERT mot ban ghi vao trong database.

sql = "SELECT \* FROM SINHVIEN \

WHERE HOCPHI > '%d'" % (1000)

try:

# Thuc thi lenh SQL

cursor.execute(sql)

# Lay tat ca cac hang trong list.

results = cursor.fetchall()

for row in results:

ho = row[0]

ten = row[1]

tuoi = row[2]

gioitinh = row[3]

hocphi = row[4]

# Bay gio in ket qua

print "ho=%s,ten=%s,tuoi=%d,gioitinh=%s,hocphi=%d" % \

(ho, ten, tuoi, gioitinh, hocphi )

except:

print "Error: khong lay duoc du lieu"

# ngat ket noi voi server

db.close()

Kết quả là:

ho=Nguyen, ten=Hoang, tuoi=20, gioitinh=M, hocphi=4000000

Hoạt động Update trong Python

Hoạt động UPDATE trên bất cứ Database nào nghĩa là để cập nhật một hoặc nhiều bản ghi mà đã có sẵn trong Database.

Thủ tục sau cập nhật tất cả bản ghi có GIOITINH là M. Ở đây, chúng ta tăng tất cả TUOI của male thêm 1 năm.

Ví dụ

import MySQLdb

# mo ket noi toi Database

db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# chuan bi mot doi tuong cursor boi su dung phuong thuc cursor()

cursor = db.cursor()

# Truy van SQL de UPDATE cac ban ghi

sql = "UPDATE SINHVIEN SET TUOI = TUOI + 1

WHERE GIOITINH = '%c'" % ('M')

try:

# Thuc thi lenh SQL

cursor.execute(sql)

# Commit cac thay doi vao trong Database

db.commit()

except:

# Rollback trong tinh huong co bat ky error nao

db.rollback()

# ngat ket noi voi server

db.close()

Hoạt động DELETE trong Python

Hoạt động DELETE là cần thiết khi bạn muốn xóa một số bản ghi từ Database. Dưới đây là thủ tục để xóa tất cả bản ghi từ bảng SINHVIEN với điều kiện là TUOI lớn hơn 20:

Ví dụ

import MySQLdb

# mo ket noi toi Database

db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# chuan bi mot doi tuong cursor boi su dung phuong thuc cursor()

cursor = db.cursor()

# Chuan bi truy van SQL de DELETE cac ban ghi can thiet

sql = "DELETE FROM SINHVIEN WHERE TUOI > '%d'" % (20)

try:

# Thuc thi lenh SQL

cursor.execute(sql)

# Commit cac thay doi vao trong Database

db.commit()

except:

# Rollback trong tinh huong co bat ky error nao

db.rollback()

# ngat ket noi voi server

db.close()

Ngắt kết nối tới Database trong Python

Để ngắt kết nối tới Database, bạn sử dụng phương thức close(), có cú pháp như sau:

db.close()

## III.3. Gửi email

SMTP (là viết tắt của Simple Mail Transfer Protocol) là một giao thức để xử lý trình gửi và định tuyến email giữa các Mail Server. Python cung cấp smtplib Module, mà định nghĩa một đối tượng SMTP Client Session có thể được sử dụng để gửi email tới bất kỳ thiết bị internet nào với một SMTP hoặc ESMTP Listener.

Dưới đây là cú pháp cơ bản để tạo một đối tượng SMTP:

import smtplib

smtpObj = smtplib.SMTP( [host [, port [, local\_hostname]]] )

Chi tiết về tham số:

host: Đây là host đang chạy SMTP Server của bạn. Bạn có thể xác định địa chỉ IP của host hoặc một tên miền như vietjack.com. Đây là tham số tùy ý.

port: Nếu bạn cung cấp tham số host, thì bạn cần xác định một port, đây là nơi SMTP Server nghe yêu cầu. Thường thì port này sẽ là 25.

ten\_localhost: Nếu SMTP Server của bạn đang chạy trên thiết bị local, thì bạn có thể xác định là localhost cho tùy chọn này.

Đối tượng SMTP có một phương thức instance là sendmail, được sử dụng để gửi một thông điệp. Nó nhận ba tham số:

sender - Là một chuỗi chỉ địa chỉ của người gửi.

receivers - Một danh sách các chuỗi, mỗi chuỗi là địa chỉ của người nhận.

message - Là một thông điệp dưới định dạng chuỗi.

Ví dụ

Dưới đây là cách đơn giản để gửi một email bởi sử dụng Python.

import smtplib

sender = 'from@fromdomain.com'

receivers = ['to@todomain.com']

message = """From: Tu <from@fromdomain.com>

To: Toi <to@todomain.com>

Subject: SMTP e-mail test

Day la vi du ve gui email.

"""

try:

smtpObj = smtplib.SMTP('localhost')

smtpObj.sendmail(sender, receivers, message)

print "Gui email thanh cong"

except SMTPException:

print "Error: khong the gui email"

Trong ví dụ trên, chúng ta đã đặt một email cơ bản trong message, bởi sử dụng trích dẫn tam. Bạn cần định dạng các trường header một cách chính xác. Một email cần một From, To, và Subject header, được phân biệt với phần thân email bởi một dòng trắng.

Để gửi một email, bạn sử dụng smtpObj để kết nối tới SMTP Server trên thiết bị local và sau đó sử dụng phương thức sendmail cùng với thông điệp message, địa chỉ người gửi, địa chỉ người nhận là các tham số.

Nếu bạn không chạy SMTP Server trên thiết bị local, bạn có thể sử dụng smtplib Client để giao tiếp với một SMTP Server từ xa. Ví dụ:

smtplib.SMTP('mail.your-domain.com', 25)

Gửi HTML email bởi sử dụng Python

Khi bạn gửi một text message bởi sử dụng Python, thì tất cả nội dung được xem như dưới dạng text đơn giản. Ngay cả khi bạn bao các HTML tag trong thông điệp text này, thì nó cũng chỉ hiển thị dưới dạng text đơn giản và các thẻ HTML này sẽ không được định dạng tương ứng với cú pháp HTML. Nhưng Python cung cấp tùy chọn để gửi một HTML message dưới dạng một thông điệp HTML thực sự.

Trong khi gửi một email message, bạn có thể xác định một Mine version, kiểu nội dung, và bộ ký tự để gửi một HTML email.

Ví dụ

Ví dụ đơn giản này gửi một HTML content dưới dạng một email.

import smtplib

message = """From: Tu <from@fromdomain.com>

To: Toi <to@todomain.com>

MIME-Version: 1.0

Content-type: text/html

Subject: SMTP HTML e-mail test

Day la vi du gui email trong dinh dang HTML

<b>Day la HTML message.</b>

<h1>Day la headline.</h1>

"""

try:

smtpObj = smtplib.SMTP('localhost')

smtpObj.sendmail(sender, receivers, message)

print "Gui email thanh cong"

except SMTPException:

print "Error: khong the gui email"

Gửi đính kèm với email bởi sử dụng Python

Để làm điều này, bạn cần thiết lập trường Content-type header thành multipart/mixed. Sau đó, các phần text và attachment có thể được xác định bên trong boundaries.

Một boundary được bắt đầu với hai dấu gạch nối (--) được theo sau bởi một số duy nhất, mà không thể xuất hiện trong phần thông điệp của email. Một final boundary, biểu thị khu vực cuối cùng của email đó, cũng phải kết thúc với hai dấu gạch dưới.

Các file đính kèm nên được bao quanh bởi hàm pack("m") để có mã hóa base64 trước khi truyền tải.

Ví dụ

Ví dụ sau sẽ gửi một file có tên là /tmp/test.txt dưới dạng một đính kèm.

import smtplib

import base64

filename = "/tmp/test.txt"

# Doc mot file va ma hoa no trong dinh dang base64

fo = open(filename, "rb")

filecontent = fo.read()

encodedcontent = base64.b64encode(filecontent) # base64

sender = 'webmaster@vietjack.com'

reciever = 'amrood.admin@gmail.com'

marker = "AUNIQUEMARKER"

body ="""

Day la vi du de gui email voi attachment.

"""

# Dinh nghia cac header.

part1 = """From: Tu <me@fromdomain.net>

To: Toi <amrood.admin@gmail.com>

Subject: Sending Attachement

MIME-Version: 1.0

Content-Type: multipart/mixed; boundary=%s

--%s

""" % (marker, marker)

# Dinh nghia message action

part2 = """Content-Type: text/plain

Content-Transfer-Encoding:8bit

%s

--%s

""" % (body,marker)

# Dinh nghia phan attachment

part3 = """Content-Type: multipart/mixed; name=\"%s\"

Content-Transfer-Encoding:base64

Content-Disposition: attachment; filename=%s

%s

--%s--

""" %(filename, filename, encodedcontent, marker)

message = part1 + part2 + part3

try:

smtpObj = smtplib.SMTP('localhost')

smtpObj.sendmail(sender, reciever, message)

print "Gui email thanh cong"

except Exception:

print "Error: khong the gui email"

## III.4. Lập trình mạng

Chương này sẽ trình bày cho bạn hiểu về khái niệm quan trọng nhất trong lập trình mạng, đó là Lập trình Socket.

Socket là gì?

Socket là các điểm đầu nút (endpoint) của một kênh giao tiếp song hướng. Các Socket có thể giao tiếp bên trong một tiến trình, giữa các tiến trình trên cùng một thiết bị hoặc giữa các tiến trình trên các lục địa khác nhau.

Các Socket có thể được triển khai thông qua các kênh khác nhau: domain, TCP, UDP, … Thư viện socket cung cấp các lớp riêng để xử lý các trình truyền tải cũng như một Interface chung để xử lý phần còn lại.

Socket có các khái niệm riêng như sau:

Khái niệm Miêu tả

domain Là family của các giao thức protocol được sử dụng như là kỹ thuật truyền tải. Các giá trị này là các hằng như AF\_INET, PF\_INET, PF\_UNIX, PF\_X25, …

type Kiểu giao tiếp giữa hai endpoint, đặc trưng là SOCK\_STREAM cho các giao thức hướng kêt nối (connection-oriented) và SOCK\_DGRAM cho các giao thức không hướng kết nối

protocol Đặc trưng là 0, mà có thể được sử dụng để nhận diện một biến thể của một giao thức bên trong một domain hoặc type

hostname Định danh của một network interface:

Một chuỗi, có thể là tên một host, địa chỉ IPV6, …

Một chuỗi "<broadcast>", xác định một địa chỉ INADDR\_BROADCAST

Một chuỗi có độ dài là 0, xác định INADDR\_ANY, hoặc

Một số nguyên, được thông dịch dưới dạng một địa chỉ nhị phân trong thứ tự host byte

port Mỗi Server nghe các lời gọi từ Client trên một hoặc nhiều cổng (port). Một port có thể là một chuỗi chứa số hiệu của port, một tên của một dịch vụ, …

socket Module trong Python

Để tạo một Socket, bạn phải sử dụng hàm socket.socket() có sẵn trong socket Module, có cú pháp chung như sau:

s = socket.socket (socket\_family, socket\_type, protocol=0)

Chi tiết về tham số:

socket\_family: Đây hoặc là AF\_UNIX hoặc AF\_INET.

socket\_type: Đây hoặc là SOCK\_STREAM hoặc SOCK\_DGRAM.

protocol: Thường được để trống, mặc định là 0.

Khi bạn đã có đối tượng socket, bạn có thể sử dụng các hàm để tạo chương trình cho Client hoặc Server. Dưới đây là danh sách các hàm:

Các phương thức sử dụng cho Server Socket

Phương thức Miêu tả

s.bind() Phương thức này gắn kết địa chỉ (hostname, port number) tới Socket

s.listen() Phương thức này thiết lập và bắt đầu TCP Listener.

s.accept() Phương thức này chấp nhận một cách thụ động kết nối TCP Client, đợi cho tới khi kết nối tới.

Các phương thức sử dụng cho Client Socket

s.connect(): Phương thức này khởi tạo kết nối TCP Server.

Các phương thức chung cho Socket

Phương thức Miêu tả

s.recv() Phương thức này nhận TCP message.

s.send() Phương thức này truyền TCP message.

s.recvfrom() Phương thức này nhận UDP message.

s.sendto() Phương thức này truyền UDP message.

s.close() Phương thức này đóng Socket.

socket.gethostname() Trả về hostname.

Ví dụ viết một chương Server đơn giản trong Python

Để viết một Server, bạn sử dụng hàm socket có trong socket Module để tạo một đối tượng socket. Sau đó, đối tượng socket được sử dụng để gọi các hàm khác để thiết lập một Socket Server.

Bây giờ gọi hàm bind(hostname, port) để xác định một port cho dịch vụ của bạn trên host đã cho.

Tiếp đó, gọi phương thức accept của đối tượng được trả về. Phương thức này đợi tới khi một Client kết nối tới port mà bạn đã xác định, và sau đó trả về một đối tượng connection mà biểu diễn kết nối tới Client đó.

# Day la server.py file

import socket # Import socket module

s = socket.socket() # Tao mot doi tuong socket

host = socket.gethostname() # Lay ten thiet bi local

port = 12345 # Danh rieng mot port cho dich vu cua ban.

s.bind((host, port)) # Ket noi toi port

s.listen(5) # Doi 5 s de ket noi voi client.

while True:

c, addr = s.accept() # Thiet lap ket noi voi client.

print 'Da ket noi voi', addr

c.send('Cam on ban da ket noi')

c.close() # Ngat ket noi

Ví dụ viết một chương trình Client đơn giản trong Python

Chúng ta viết một chương trình Client đơn giản để mở một kết nối tới một port có số hiệu đã cho là 12345 và với host đã xác định.

Hàm socket.connect(hosname, port ) mở một kết nối TCP tới hostname trên port đã cho. Khi bạn có một socket đã được mở, bạn có thể đọc từ nó giống như bất kỳ đối tượng IO nào.

Code sau là một Client rất đơn giản để kết nối tới host và port đã cho, đọc bất cứ dữ liệu nào có sẵn từ Socket đó và sau đó thoát.

# This is client.py file

import socket # Import socket module

s = socket.socket() # Tao mot doi tuong socket

host = socket.gethostname() # Lay ten thiet bi local

port = 12345 # Danh rieng mot port cho dich vu cua ban.

s.connect((host, port))

print s.recv(1024)

s.close # Dong socket

Bây giờ chạy server.py trong background và sau đó chạy client.py trên để xem kết quả.

# Chay server trong background.

$ python server.py &

# Mot khi server da bat dau, ban chay client nhu sau:

$ python client.py

Kết quả là:

Da ket noi voi ('127.0.0.1', 48437)

Cam on ban da ket noi

Các Module quan trọng trong lập trình mạng

Protocol Tính năng chung Port No Python module

HTTP Web page 80 httplib, urllib, xmlrpclib

NNTP Usenet new 119 nntplib

FTP Truyền tải file 20 ftplib, urllib

SMTP Gửi email 25 smtplib

POP3 Lấy email 110 poplib

IMAP4 Lấy email 143 imaplib

Telnet Command line 23 telnetlib

Gopher Truyền tải Document 70 gopherlib, urllib

## III.5. Xử lí đa luồng

Một chương trình đa luồng chứa hai hoặc nhiều phần mà có thể chạy đồng thời và mỗi phần có thể xử lý tác vụ khác nhau tại cùng một thời điểm, để sử dụng tốt nhất các nguồn có sẵn, đặc biệt khi máy tính của bạn có nhiều CPU.

Python cung cấp thread Module và threading Module để bạn có thể bắt đầu một thread mới cũng như một số tác vụ khác trong khi lập trình đa luồng. Mỗi một Thread đều có vòng đời chung là bắt đầu, chạy và kết thúc. Một Thread có thể bị ngắt (interrupt), hoặc tạm thời bị dừng (sleeping) trong khi các Thread khác đang chạy – được gọi là yielding.

Bắt đầu một Thread mới trong Python

Phương thức dưới đây có sẵn trong thread Module được sử dụng để bắt đầu một Thread mới:

thread.start\_new\_thread ( function, args[, kwargs] )

Lời gọi phương thức này được trả về ngay lập tức và Thread con bắt đầu và gọi hàm function với danh sách các tham số args đã truyền. Khi hàm function trả về, thì Thread kết thúc.

Ở đây, args là một Tuple của các tham số, sử dụng một Tuple trống để gọi hàm function mà không truyền cho nó bất kỳ tham số nào. Tham số kwargs là một Dictionary của các tham số từ khóa tùy ý. (bạn thao khảo chương Hàm trong Python để biết chi tiết tham số từ khóa là gì)

Ví dụ

import thread

import time# Dinh nghia mot ham cho thread

def print\_time( threadName, delay):

count = 0

while count < 5:

time.sleep(delay)

count += 1

print "%s: %s" % ( threadName, time.ctime(time.time()) )# Tao hai thread nhu sau

try:

thread.start\_new\_thread( print\_time, ("Thread-1", 2, ) )

thread.start\_new\_thread( print\_time, ("Thread-2", 4, ) )

except:

print "Error: khong the bat dau thread"while 1:

pass

Kết quả là:

Thread-1: Mon Nov 21 15:42:17 2015

Thread-1: Mon Nov 21 15:42:19 2015

Thread-2: Mon Nov 21 15:42:19 2015

Thread-1: Mon Nov 21 15:42:21 2015

Thread-2: Mon Nov 21 15:42:23 2015

Thread-1: Mon Nov 21 15:42:23 2015

Thread-1: Mon Nov 21 15:42:25 2015

Thread-2: Mon Nov 21 15:42:27 2015

Thread-2: Mon Nov 21 15:42:31 2015

Thread-2: Mon Nov 21 15:42:35 2015

Mặc dù thread Module rất hiệu quả với đa luồng tầm thấp nhưng khi so sánh với threading Module thì nó có nhiều điểm hạn chế. Phần tiếp theo giới thiệu về threading Module.

threading Module trong Python

Module mới này được bao với Python 2.4 nhằm cung cấp nhiều hỗ trợ mạnh mẽ và cấp độ cao hơn cho các Thread trong khi so sánh với thread Module ở trên. Ngoài các phương thức có trong thread Module, thì threading Module còn bổ sung thêm một số phương thức khác, đó là:

threading.activeCount(): Trả về số đối tượng thread mà là active.

threading.currentThread(): Trả về số đối tượng thread trong Thread control của Caller.

threading.enumerate(): Trả về một danh sách tất cả đối tượng thread mà hiện tại là active.

Bên cạnh đó, threading Module có lớp Thread để triển khai đa luồng. Lớp này có các phương thức sau:

run(): Là entry point cho một Thread.

start(): Bắt đầu một thread bởi gọi phương thức run().

join([time]): Đợi cho các thread kết thúc.

isAlive(): Kiểm tra xem một thread có đang thực thi hay không.

getName(): Trả về tên của một thread.

setName(): Thiết lập tên của một thread.

Tạo Thread bởi sử dụng threading Module trong Python

Để triển khai một thread mới bởi sử dụng threading Module, bạn phải thực hiện:

Định nghĩa một lớp con của lớp Thread.

Ghi đè phương thức \_\_init\_\_(self [,args]) để bổ sung thêm các tham số.

Sau đó, ghi đè phương thức run(self [,args]) để triển khai những gì thread cần thực hiện khi được bắt đầu.

Một khi bạn đã tạo lớp con Thread mới, bạn có thể tạo một instance của nó và sau đó bắt đầu một Thread bởi triệu hồi phương thức start().

Ví dụ

import threading

import timeexitFlag = 0class myThread (threading.Thread):

def \_\_init\_\_(self, threadID, name, counter):

threading.Thread.\_\_init\_\_(self)

self.threadID = threadID

self.name = name

self.counter = counter

def run(self):

print "Bat dau " + self.name

print\_time(self.name, self.counter, 5)

print "Ket thuc " + self.namedef print\_time(threadName, delay, counter):

while counter:

if exitFlag:

threadName.exit()

time.sleep(delay)

print "%s: %s" % (threadName, time.ctime(time.time()))

counter -= 1# Tao cac thread moi

thread1 = myThread(1, "Thread-1", 1)

thread2 = myThread(2, "Thread-2", 2)# Bat dau cac thread moi

thread1.start()

thread2.start()print "Ket thuc Main Thread"

Kết quả là:

Bat dau Thread-1

Bat dau Thread-2

Ket thuc Main Thread

Thread-1: Mon Nov 21 09:10:03 2015

Thread-1: Mon Nov 21 09:10:04 2015

Thread-2: Mon Nov 21 09:10:04 2015

Thread-1: Mon Nov 21 09:10:05 2015

Thread-1: Mon Nov 21 09:10:06 2015

Thread-2: Mon Nov 21 09:10:06 2015

Thread-1: Mon Nov 21 09:10:07 2015

Ket thuc Thread-1

Thread-2: Mon Nov 21 09:10:08 2015

Thread-2: Mon Nov 21 09:10:10 2015

Thread-2: Mon Nov 21 09:10:12 2015

Ket thuc Thread-2

Đồng bộ hóa các Thread trong Python

Python cung cấp threading Module, mà bao gồm một kỹ thuật locking cho phép bạn đồng bộ hóa các Thread một cách dễ dàng. Một lock mới được tạo bởi gọi phương thức Lock().

Phương thức acquire(blocking) của đối tượng lock mới này được sử dụng để ép các Thread chạy một cách đồng bộ. Tham số blocking tùy ý cho bạn khả năng điều khiển để xem một Thread có cần đợi để đạt được lock hay không.

Nếu tham số blocking được thiết lập là 0, tức là Thread ngay lập tức trả về một giá trị 0 nếu không thu được lock và trả về giá trị 1 nếu thu được lock. Nếu blocking được thiết lập là 1, thì Thread cần đợi cho đến khi lock được giải phóng.

Phương thức release() của đối tượng lock được sử dụng để giải phóng lock khi nó không cần nữa.

Ví dụ

import threading

import timeclass myThread (threading.Thread):

def \_\_init\_\_(self, threadID, name, counter):

threading.Thread.\_\_init\_\_(self)

self.threadID = threadID

self.name = name

self.counter = counter

def run(self):

print "Bat dau " + self.name

# Lay lock de dong bo hoa cac thread

threadLock.acquire()

print\_time(self.name, self.counter, 3)

# Giai phong lock cho thread ke tiep

threadLock.release()def print\_time(threadName, delay, counter):

while counter:

time.sleep(delay)

print "%s: %s" % (threadName, time.ctime(time.time()))

counter -= 1threadLock = threading.Lock()

threads = []# Tao cac thread moi

thread1 = myThread(1, "Thread-1", 1)

thread2 = myThread(2, "Thread-2", 2)# Bat dau cac thread moi

thread1.start()

thread2.start()# Them cac thread vao list

threads.append(thread1)

threads.append(thread2)# Doi cho tat ca thread ket thuc

for t in threads:

t.join()

print "Ket thuc Main Thread"

Kết quả là:

Bat dau Thread-1

Bat dau Thread-2

Thread-1: Mon Nov 21 09:11:28 2015

Thread-1: Mon Nov 21 09:11:29 2015

Thread-1: Mon Nov 21 09:11:30 2015

Thread-2: Mon Nov 21 09:11:32 2015

Thread-2: Mon Nov 21 09:11:34 2015

Thread-2: Mon Nov 21 09:11:36 2015

Ket thuc Main Thread

Queue Module: quyền ưu tiên đa luồng trong Python

Queue Module cho phép bạn tạo một đối tượng queue mới mà có thể giữ một số lượng item nào đó. Dưới đây là các phương thức:

get(): Xóa và trả về một item từ queue.

put(): Thêm một item tới một queue.

qsize() : Trả về số item mà hiện tại đang trong queue.

empty(): Trả về true nếu queue là trống, nếu không thì trả về false.

full(): Trả về true nếu queue là đầy, nếu không thì trả về false.

Ví dụ

import Queue

import threading

import timeexitFlag = 0class myThread (threading.Thread):

def \_\_init\_\_(self, threadID, name, q):

threading.Thread.\_\_init\_\_(self)

self.threadID = threadID

self.name = name

self.q = q

def run(self):

print "Bat dau " + self.name

process\_data(self.name, self.q)

print "Ket thuc " + self.namedef process\_data(threadName, q):

while not exitFlag:

queueLock.acquire()

if not workQueue.empty():

data = q.get()

queueLock.release()

print "%s dang xu ly %s" % (threadName, data)

else:

queueLock.release()

time.sleep(1)threadList = ["Thread-1", "Thread-2", "Thread-3"]

nameList = ["One", "Two", "Three", "Four", "Five"]

queueLock = threading.Lock()

workQueue = Queue.Queue(10)

threads = []

threadID = 1# Tao cac thread moi

for tName in threadList:

thread = myThread(threadID, tName, workQueue)

thread.start()

threads.append(thread)

threadID += 1# Dien vao queue

queueLock.acquire()

for word in nameList:

workQueue.put(word)

queueLock.release()# Doi den khi queue la trong

while not workQueue.empty():

pass# Thong bao cho thread do la thoi gian de ket thuc

exitFlag = 1# Doi cho tat ca thread ket thuc

for t in threads:

t.join()

print "Ket thuc Main Thread"

Kết quả là:

Bat dau Thread-1

Bat dau Thread-2

Bat dau Thread-3

Thread-1 dang xu ly One

Thread-2 dang xu ly Two

Thread-3 dang xu ly Three

Thread-1 dang xu ly Four

Thread-2 dang xu ly Five

Ket thuc Thread-3

Ket thuc Thread-1

Ket thuc Thread-2

Ket thuc Main Thread

# IV. Các đặc điểm mới trong Python 3.x

Một số thay đổi cần lưu ý nhất[sửa | sửa mã nguồn]

Lệnh print trở thành hàm print(). Theo đó sau print() ta cần nhớ gõ vào cặp ngoặc ():

print("Hello")

print(2+3)

Trả lại kết quả không còn là list trong một số trường hợp.

dict.keys(), dict.items(), dict.values() kết quả cho ra các "view" thay vì list.

map và filter trả lại các iterator.

range bây giờ có tác dụng như xrange, và không trả lại list.

So sánh

Không còn hàm cmp, và cmp(a, b) có thể được thay bằng (a > b) - (a < b)

Số nguyên

Kiểu long được đổi tên thành int.

1/2 cho ta kết quả là số thực chứ không phải số nguyên.

Không còn hằng số sys.maxint

Kiểu bát phân được ký hiệu bằng 0o thay vì 0, chẳng hạn 0o26.

Phân biệt văn bản - dữ liệu nhị phân thay vì Unicode - chuỗi 8-bit

Tất cả chuỗi văn bản đều dưới dạng Unicode, nhưng chuỗi Unicode mã hóa lại là dạng dữ liệu nhị phân. Dạng mặc định là UTF-8.

Không thể viết u"a string" để biểu diễn chuỗi như trong các phiên bản 2.x

Các thay đổi về cú pháp[sửa | sửa mã nguồn]

Cú pháp mới[sửa | sửa mã nguồn]

Các tham biến chỉ chấp nhận keyword: Các tham biến phía sau \*args phải được gọi theo dạng keyword.

Từ khóa mới nonlocal. Muốn khai báo một biến x với có phạm vi ảnh hưởng rộng hơn, nhưng chưa đến mức toàn cục, ta dùng nonlocal x.

Gán giá trị vào các phần tử tuple một cách thông minh, chẳng hạn có thể viết (a, \*rest, b) = range(5) để có được a = 0; b = [1,2,3]; c = 4.

Dictionary comprehension, chẳng hạn {k: v for k, v in stuff} thay vì dict(stuff).

Kiểu nhị phân, chẳng hạn b110001.

Cú pháp được thay đổi[sửa | sửa mã nguồn]

raise [biểu\_thức [from biểu\_thức]]

except lệnh as biến

Sử dụng metaclass trong đối tượng:

class C(metaclass=M):

pass

Cách dùng biến \_\_metaclass\_\_ không còn được hỗ trợ.

Cú pháp bị loại bỏ[sửa | sửa mã nguồn]

Không còn dấu ``, thay vì đó, dùng repr.

Không còn so sánh <> (dùng !=).

Không còn các lớp kiểu classic.

# Kết luận

Qua những tìm hiểu về Python,ta có thể thấy được đó là một ngôn ngữ thông dịch dễ tiếp cận, dễ học, dễ sử dụng và thật sự hữu ích,Python với những hỗ trợ tuyệt vời, hướng đối tượng,hướng thủ tuc với những bộ cài đặc biệt, đã giúp cho lập trình viên dễ dàng tiếp cận những lĩnh vực mới trong ngành như AI, web, an toàn mạng, v.v. Cũng có lẽ vậy mà chúng ta nên tiếp cận và sử dụng ngôn ngữ này. Nhiều khả năng trong tương lai, Python sẽ ngày càng được phát triển và ứng dụng nhiều hơn nữa trong lĩnh vực công nghệ thông tin bởi những khả năng của nó.

# Tài liệu tham khảo

Các trang web:

+) <http://vietjack.com/python/>

+) <https://vi.wikipedia.org/wiki/Python>

Bài chỉ dẫn Python - Guido van Rossum - Python Software Foundation - Thư điện tử: docs@python.org - Fred L. Drake, Jr., - biên tập viên do Nguyễn Thành Nam, Lê Hồng Việt và Lương Trọng Đức của nhóm Python cho người Việt dịch