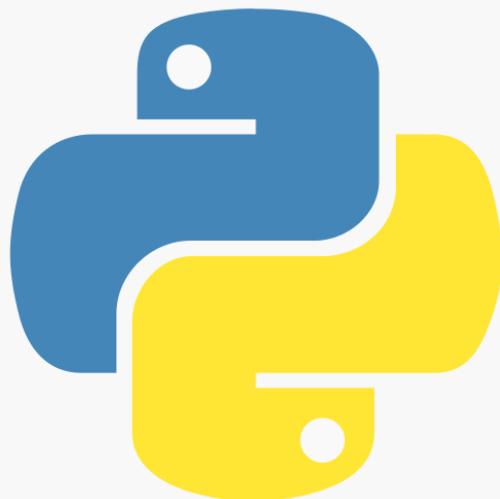


HIENLTH

**UUI HỌC
LẬP TRÌNH
PYTHON**



05.2021

STUDY

TRUNG TÂM ĐÀO TẠO CNTT NHẤT NGHỆ



Tài liệu

Lập trình Python

Tổng hợp và Biên soạn:
Lương Trần Hy Hiến

TP.HCM
05, 2021



python

LỜI MỞ ĐẦU

Python là ngôn ngữ lập trình bậc cao ra đời từ năm 1991. Đây là ngôn ngữ rất gần gũi với ngôn ngữ tự nhiên (tiếng Anh), với cấu trúc rõ ràng, dễ đọc, dễ học, dễ hiểu. Python hiện nay là ngôn ngữ lập trình được sử dụng phổ biến rộng rãi ở châu Âu, châu Mỹ và được sử dụng rộng rãi trong trường học.

Tài liệu "Lập trình Python" này được tổng hợp từ nhiều nguồn khác nhau, nhằm giúp bạn đọc hiểu nhanh, dễ làm thông qua các nội dung của tài liệu.

Tài liệu bao gồm 17 chương, từ những khái niệm cơ bản đến các chủ đề gợi mở nhằm giúp người đọc có cái nhìn tổng quát về ngôn ngữ Python.

Trong quá trình thực hiện, chắc chắn sẽ không tránh khỏi sai sót, rất mong nhận được sự góp ý từ bạn đọc. Mọi ý kiến đóng góp, trao đổi xin gửi về: pynhatnghe@gmail.com.

MỤC LỤC

LỜI MỞ ĐẦU.....	1
MỤC LỤC	2
Bài 1: Tổng quan Lập trình Python.....	10
1.1 Lập trình là gì?.....	10
1.2 Python là gì	11
1.2.1 Giới thiệu.....	11
1.2.2 Lịch sử của Python.....	11
1.2.3 Các phiên bản Python đã phát hành	12
1.2.4 Python được dùng ở đâu?	14
1.2.5 Đặc điểm của Python	15
1.3 Cài đặt môi trường.....	15
1.3.1 Cài đặt Python	15
1.3.2 Giới thiệu Pycharm IDE	19
1.4 Phân khõi và chú thích.....	28
1.4.1 Phân khõi	28
1.4.2 Chú thích.....	29
1.5 Python keyword.....	29
1.6 Biến	35
1.6.1 Biến là gì	35
1.6.2 Gán giá trị cho biến.....	36
1.6.3 Xóa biến khỏi bộ nhớ	36
1.6.4 Quy tắc đặt tên biến	36

1.7	Nhập xuất dữ liệu trên shell	37
1.7.1	Nhập với input.....	37
1.7.2	Xuất với print.....	37
Bài 2:	Kiểu dữ liệu, Toán tử	41
2.1	Các kiểu dữ liệu.....	41
2.1.1	Xác định id đối tượng.....	42
2.1.2	Xác định kiểu dữ liệu.....	42
2.1.3	Kiểu số.....	43
2.1.4	Kiểu chuỗi.....	44
2.1.5	Kiểu Boolean.....	46
2.1.6	Kiểu None.....	47
2.2	Các loại toán tử	47
2.2.1	Toán tử số học	47
2.2.2	Toán tử so sánh.....	48
2.2.3	Toán tử Gán	49
2.2.4	Toán tử Logic	51
2.2.5	Toán tử định danh	51
2.2.6	Toán tử membership.....	52
2.2.7	Độ ưu tiên toán tử	52
2.3	Các hàm toán học.....	53
2.3.1	Type Conversion	53
2.3.2	Iterables and Iterators	54
2.3.3	Composite Data Type.....	55
Bài 3:	Cấu trúc Điều khiển, Lặp.....	56

3.1	Lệnh rẽ nhánh	56
3.1.1	Câu lệnh If	56
3.1.2	Câu lệnh If .. Else ..	56
3.1.3	Câu lệnh If .. ElIf .. Else ..	56
3.2	Lệnh lặp while	57
3.3	Lệnh lặp for	58
3.4	Sử dụng break, continue, pass statement	60
3.4.1	Câu lệnh break	60
3.4.2	Câu lệnh continue	61
3.4.3	Câu lệnh pass	61
Bài 4:	String, Date & Time	63
4.1	Kiểu chuỗi (string)	63
4.1.1	Chuỗi và truy xuất chuỗi	63
4.1.2	Các hàm xử lý chuỗi	64
4.1.3	Tìm chuỗi con	70
4.1.4	Toán tử trên string	72
4.1.5	Thay thế chuỗi	73
4.1.6	Tách, nối chuỗi	73
4.1.7	Chữ HOA và chữ thường	75
4.1.8	Một số thao tác khác	76
4.1.9	Định dạng chuỗi	77
4.2	Kiểu Date, Time	80
4.2.1	Kiểu datetime	80
4.2.2	Khoảng thời gian	81

4.2.3	Định dạng datetime	82
4.2.4	Module calendar	83
4.2.5	Module pytz.....	85
4.2.6	Module dateutil.....	86
Bài 5:	Kiểu cấu trúc	88
5.1	Kiểu List.....	88
5.1.1	Giới thiệu và khai báo	88
5.1.2	Các phép tính trên list.....	91
5.1.3	Một số hàm trên kiểu dữ liệu List.....	92
5.1.4	Sắp xếp một List.....	94
5.1.5	Thêm phần tử mới vào list	95
5.1.6	Lỗi IndexError, TypeError.....	96
5.1.7	Xóa phần tử trong list	97
5.1.8	Thay đổi giá trị phần tử trong list	99
5.1.9	Sao chép một list	99
5.1.10	Chỉ số trong list.....	101
5.1.11	Duyệt chuỗi	102
5.1.12	Đếm số lượng phần tử trong list.....	103
5.1.13	Lồng các list lại với nhau	104
5.1.14	Sắp xếp list.....	105
5.1.15	Hàm map() và hàm filter()	106
5.2	Kiểu Tuple	107
5.2.1	Giới thiệu và khai báo	107
5.2.2	Truy cập các phần tử	108

5.2.3	Các phép tính trên Tuple.....	108
5.3	Kiểu Dictionary	109
5.3.1	Giới thiệu.....	109
5.3.2	Khởi tạo dictionary.....	109
5.3.3	Các phép toán cơ bản	111
5.3.4	Làm việc với khóa và giá trị.....	114
5.3.5	Duyệt dictionary.....	115
5.4	Kiểu Set	118
5.4.1	Giới thiệu.....	119
5.4.2	Các phép tính trên Set	119
5.4.3	Một số ví dụ minh họa.....	120
Bài 6:	Hàm (function)	122
6.1	Xây dựng, gọi sử dụng phương thức	122
6.1.1	Giới thiệu.....	122
6.1.2	Khai báo hàm	122
6.1.3	Tham số của hàm	123
6.2	Anonymous Function (lambda).....	123
6.2.1	Cú pháp.....	123
6.2.2	Hàm Lambda với filter()	124
6.2.3	Hàm Lambda với map().....	124
Bài 7:	Module và làm việc với thư viện.....	125
7.1	Giới thiệu module	125
7.2	Ví dụ cách tạo và sử dụng module.....	125
7.3	Package	127

Bài 8: Xử lý ngoại lệ	128
8.1 Giới thiệu.....	128
8.2 Các exception có sẵn trong Python.....	129
8.3 Custom Exception trong Python.....	131
Bài 9: File I/O.....	133
9.1 Làm việc với tập tin (text/nhị phân).....	133
9.1.1 Mở file.....	133
9.1.2 Đọc file.....	133
9.1.3 Ghi file:	134
9.1.4 Đóng file.....	134
9.1.5 Truy xuất file với cấu trúc 'with'.....	134
9.1.6 Một số ví dụ minh họa.....	135
9.1.7 Một số hàm xử lý khác.....	136
9.2 Làm việc với thư mục.....	137
Bài 10: Lập trình hướng đối tượng.....	138
Bài 11: Làm việc với dữ liệu JSON/CSV	139
11.1 Làm việc với JSON	139
11.1.1 Giới thiệu.....	139
11.1.2 Làm việc với JSON bằng Python	139
11.2 Làm việc với file CSV.....	142
11.2.1 Module csv trong python	142
11.2.2 Cách đọc csv	143
11.2.3 Cấu tạo 1 file csv	144
11.2.4 Sử dụng csv trong python	144

11.2.5 SỬ DỤNG CSV.READER ()	144
11.2.6 SỬ DỤNG LỚP CSV.DICTREADER ().....	145
11.3 Thư viện request, urllib3	146
11.3.1 Thư viện request.....	146
11.3.2 Thư viện urllib3.....	147
Bài 12: Làm việc với cơ sở dữ liệu.....	148
12.1 Giới thiệu hệ quản trị CSDL SQLite	148
12.2 Làm việc với MySQL.....	151
Bài 13: Thread - multi thread	153
13.1 Cách hoạt động của đa luồng trong python	153
13.2 Cách bắt đầu luồng mới trong Python.....	153
13.3 Các luồng mô đun trong Python	154
13.4 Tạo luồng sử dụng threading module	155
13.5 Queue đa luồng ưu tiên trong python.....	155
Bài 14: Regular Expression.....	156
14.1 Các hàm regex trong python.....	156
14.2 Xây dựng regex trong python.....	157
14.3 Regex trong python - xây dựng bằng set.....	157
14.4 Regex trong python - xây dựng bằng ký tự đặc biệt.....	158
Bài 15: Xây dựng ứng dụng GUI với Tkinter.....	160
15.1 Tkinter	160
15.2 Tạo ứng dụng đầu tiên	160
15.3 Các widgets của Tkinter python.....	161
15.4 Quản lý hình học trong Tkinter python	162

Bài 16: Một số ứng dụng Python	163
16.1 Thư viện NumPy	163
16.1.1 Giới thiệu	163
16.1.2 Một số hàm thường sử dụng của numpy	163
16.1.3 Truy nhập phần tử của mảng trong numpy	166
16.1.4 Các phép tính trên mảng dữ liệu numpy	167
16.2 Làm việc với Matplotlib để vẽ đồ thị	169
16.2.1 Khái niệm chung	169
16.2.2 Bắt đầu với Pyplot	170
16.3 Scraping data	185
TÀI LIỆU THAM KHẢO	186

Bài 1: Tổng quan Lập trình Python

1.1 Lập trình là gì?

Lập trình là một công việc mà lập trình viên sử dụng những ngôn ngữ lập trình, các đoạn code, những tiện ích có sẵn để xây dựng nên các phần mềm, chương trình, ứng dụng, trò chơi, các trang web, ... nhằm giúp người dùng có thể thực hiện các mệnh lệnh với máy tính hay tương tác qua lại với nhau thông qua các thiết bị điện tử. Lập trình là một phần trong ngành công nghệ thông tin chứ không phải là công nghệ thông tin. Khi chưa có máy tính, con người vẫn giải các bài toán thường gặp trong cuộc sống bằng cách tính nhẩm trong đầu đối với bài toán đơn giản, còn các bài toán phức tạp hơn cần đến giấy và bút.

Ví dụ:

Một người đi làm bằng xe máy, quãng đường từ nhà đến nơi làm việc là 13 km. Trung bình cứ 50km thì xe máy tiêu thụ hết 1 lít xăng. Giá xăng là 15000 đồng/lít. Một tháng người đó phải đi làm 25 ngày. Hỏi tiền xăng cho việc đi lại trong 1 tháng của người đó là bao nhiêu?

Lời giải:

Quãng đường đi lại trong 1 ngày:

$$2 \times 13 = 26 \text{ (km)}$$

Quãng đường đi lại trong 1 tháng:

$$26 \times 25 = 650 \text{ (km)}$$

Số lít xăng tiêu thụ trong 1 tháng:

$$650/50 = 13 \text{ (lít)}$$

Tiền xăng tiêu thụ trong 1 tháng:

$$13 \times 15000 = 195000 \text{ (đồng)}$$

Đáp số: 195000 đồng

Kể từ khi có máy tính, con người đã sử dụng máy trong việc giải các bài toán. Việc giải toán trên máy tính nhanh hơn và chính xác hơn so với việc giải bằng tay.

1.2 Python là gì

1.2.1 Giới thiệu

Python là ngôn ngữ lập trình hướng đối tượng, thông dịch, mã nguồn mở, đa mục đích và luôn nằm trong top 10 ngôn ngữ lập trình phổ biến nhất ở tất cả các bảng xếp hạng lớn (TIOBE, RedMonk, PYPL).

Python nằm trong top các ngôn ngữ lập trình được sử dụng nhiều nhất bởi các nhà khoa học dữ liệu (data scientists), đó là những chuyên gia đang phát triển các giải pháp thông minh cho nhiều thách thức khác nhau như việc tìm kiếm phương pháp chữa trị cho bệnh ung thư, lập bản đồ hành vi khủng bố, và cải thiện khả năng nhận thức của trẻ em. Các bộ tool mạnh mẽ được phát triển bởi Google, Facebook trong lĩnh vực AI (Artificial Intelligence), Machine Learning, Deep Learning đều phát hành các mã nguồn được viết bằng Python.

1.2.2 Lịch sử của Python

Python 1: bao gồm các bản phát hành 1.x. Giai đoạn này, kéo dài từ đầu đến cuối thập niên 1990. Từ năm 1990 đến 1995, Guido van Rossum làm việc tại CWI (Centrum voor Wiskunde en Informatica - Trung tâm Toán-Tin học tại Amsterdam, Hà 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 (Corporation for National Research Initiatives) ở Reston, 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. Sau đó, CNRI và FSF (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 và thành lập BeOpen PythonLabs team. 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.

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 (PSF), một tổ chức phi lợi nhuận được thành lập theo mẫu Apache Software Foundation.

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.

1.2.3 Các phiên bản Python đã phát hành

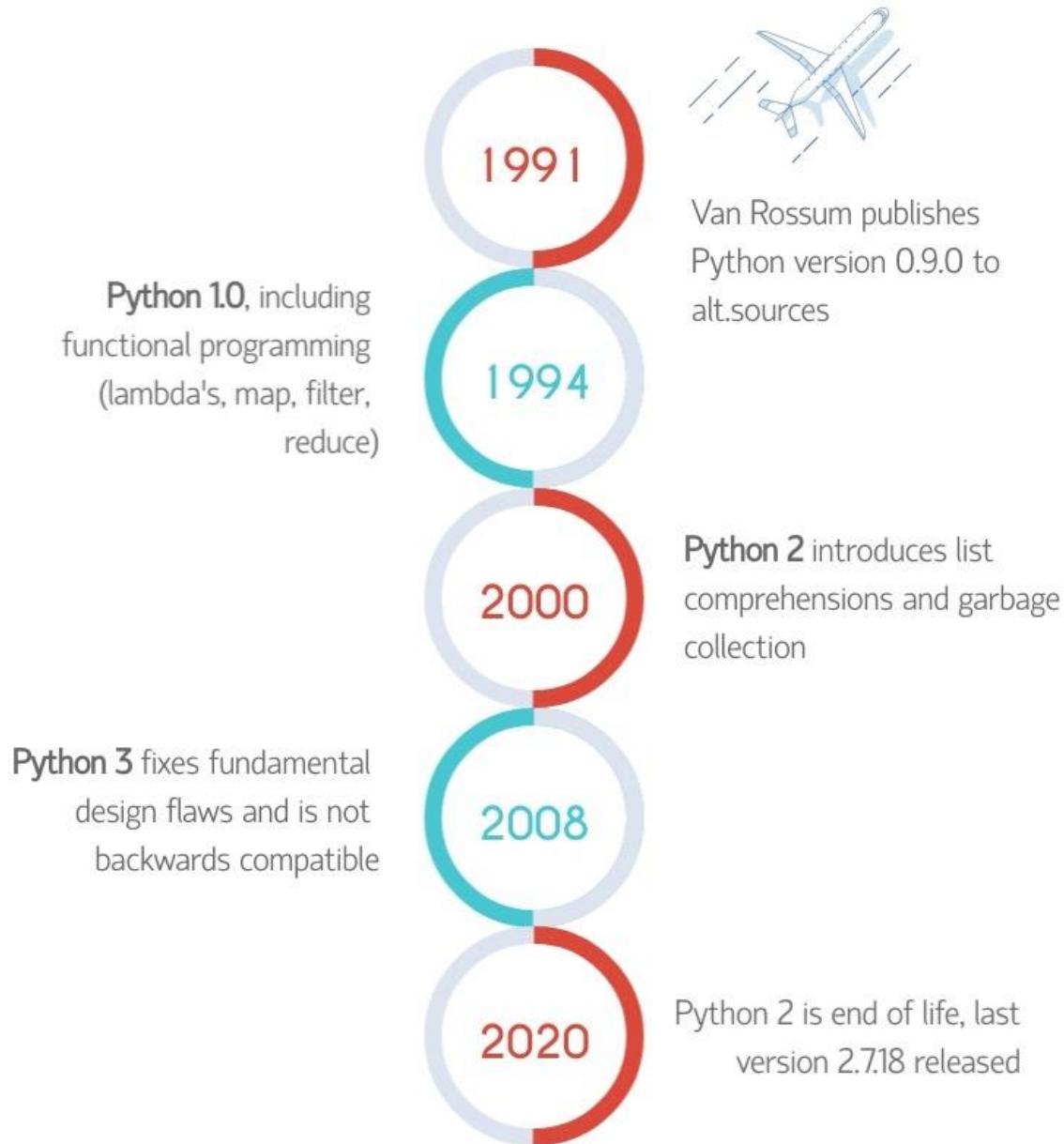
Phiên bản	Ngày phát hành
Python 1.0 (bản phát hành chuẩn đầu tiên) Python 1.6 (Phiên bản 1.x cuối cùng)	01/1994 05/09/2000
Python 2.0 (Giới thiệu list comprehension) Python 2.7 (Phiên bản 2.x cuối cùng)	16/10/2000 03/07/2010
Python 3.0 (Loại bỏ cấu trúc và mô-đun trùng lặp) Python 3.8.5 (Bản mới nhất tính đến thời điểm viết bài)	03/12/2008 20/07/2020

Bạn có thể xem các version tại trang chính <https://www.python.org/doc/versions/>

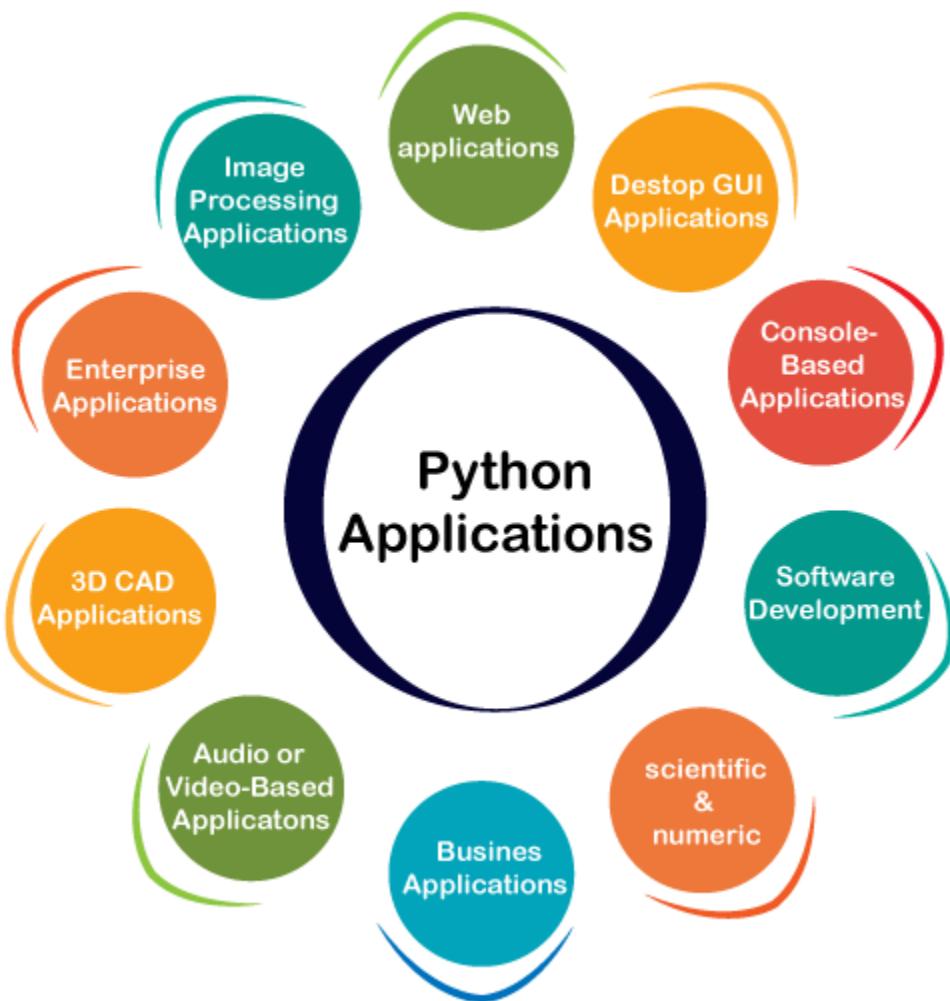
The screenshot shows a web browser window displaying the Python Documentation by Version page. The URL in the address bar is <https://www.python.org/doc/versions/>. The page has a dark blue header with navigation links for About, Downloads, Documentation, Community, Success Stories, News, and Events. On the left side, there is a sidebar with a tweet from @ThePSF about a fundraiser. The main content area is titled "Python Documentation by Version" and lists previous versions of the documentation available online. A sidebar on the right lists unreleased (in development) documentation versions. The visible list of released versions includes:

- [Python 3.9.5](#), documentation released on 3 May 2021.
- [Python 3.9.4](#), documentation released on 4 April 2021.
- [Python 3.9.3](#), documentation released on 2 April 2021.
- [Python 3.9.2](#), documentation released on 19 February 2021.
- [Python 3.9.1](#), documentation released on 8 December 2020.
- [Python 3.9.0](#), documentation released on 5 October 2020.
- [Python 3.8.10](#), documentation released on 3 May 2021.
- [Python 3.8.9](#), documentation released on 2 April 2021.

Python History



1.2.4 Python được dùng ở đâu?



Lập trình ứng dụng web (Web Application): Bạn có thể tạo web app có khả năng mở rộng (scalable) được bằng cách sử dụng framework và CMS (Hệ thống quản trị nội dung) được tích hợp trong Python. Vài nền tảng phổ biến để tạo web app là: Django, Flask, Pyramid, Plone, Django CMS. Các trang như Mozilla, Reddit, Instagram và PBS đều được viết bằng Python. Ngày nay với sự ra đời của FastAPI (từ Python 3.6+) bạn có thể xây dựng các API backend cung cấp cho các ứng dụng khác thật dễ dàng.

Khoa học và tính toán (Scientific & numeric): Có nhiều thư viện trong Python cho khoa học và tính toán số liệu, như SciPy và NumPy, được sử dụng cho những mục đích chung chung trong tính toán. Và, có những thư viện cụ thể như: EarthPy cho khoa học trái đất, AstroPy cho Thiên văn học, ... Ngoài ra, Python còn được sử dụng nhiều trong machine learning, khai thác dữ liệu và deep learning.

Tạo nguyên mẫu phần mềm: Python chậm hơn khi so sánh với các ngôn ngữ được biên dịch như C++ và Java. Nó có thể không phải là lựa chọn tốt nếu nguồn lực bị giới hạn và yêu cầu về hiệu quả là bắt buộc. Tuy nhiên, Python là ngôn ngữ tuyệt vời để tạo những nguyên mẫu (bản chạy thử - prototype). Ví dụ, bạn có thể sử dụng Pygame (thư viện viết game) để tạo nguyên mẫu game trước. Nếu thích nguyên mẫu đó có thể dùng C++ để viết game thực sự.

Ngôn ngữ tốt để dạy lập trình: Python được nhiều công ty, trường học sử dụng để dạy lập trình cho trẻ em và những người mới lần đầu học lập trình. Bên cạnh những tính năng tuyệt vời thì cú pháp đơn giản và dễ sử dụng của nó là lý do chính cho việc lựa chọn này.

1.2.5 Đặc điểm của Python

Python có ưu điểm:

- Cú pháp ngắn gọn, dễ hiểu, dễ đọc, phù hợp cho người mới học.
- Hỗ trợ nhiều nền tảng, nhiều hệ điều hành khác nhau.
- Cộng đồng phát triển rất đông đảo (thư viện nhiều, tài liệu đầy đủ).

Bên cạnh đó Python có một số nhược điểm:

- Là ngôn ngữ thông dịch nên tốc độ thực thi chương trình Python chậm hơn so với các ngôn ngữ biên dịch.
- Là ngôn ngữ kiểu động (dynamic type) nên không kiểm tra được hết các lỗi khi viết chương trình.
- Python không phải là ngôn ngữ tốt trong xử lý đa tác vụ (Multi-threading)

1.3 Cài đặt môi trường

1.3.1 Cài đặt Python

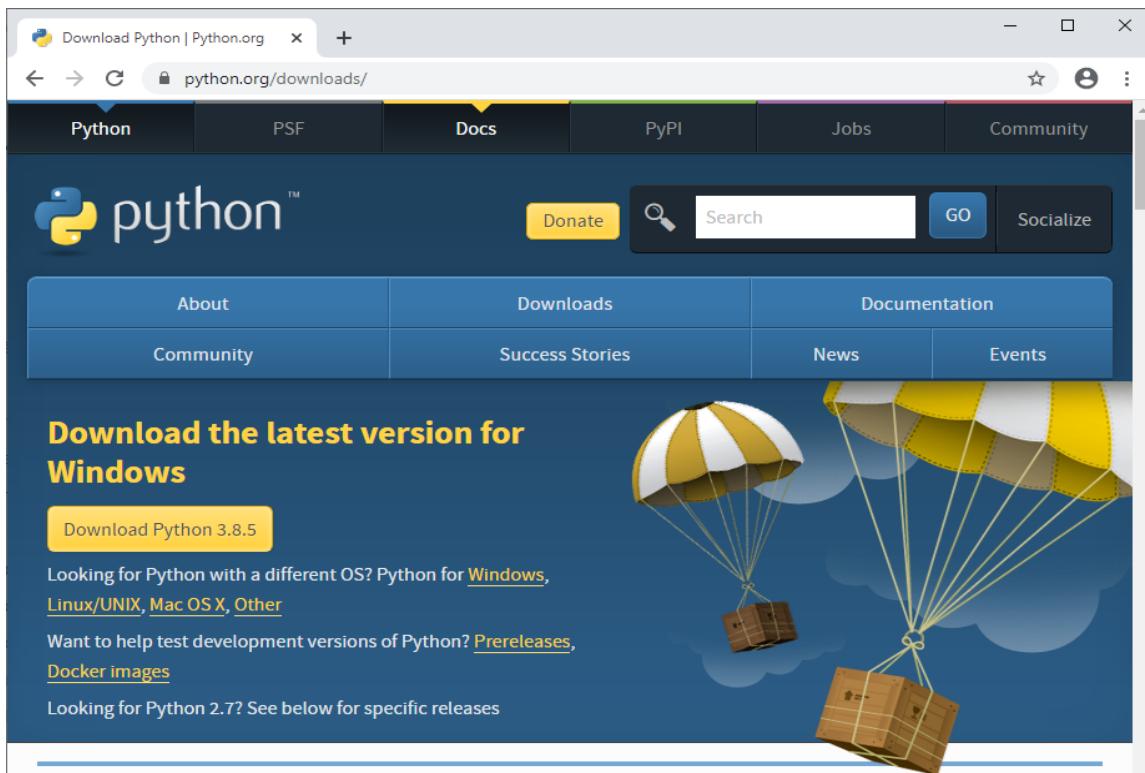
Python là ngôn ngữ được hỗ trợ trên cả 3 nền tảng Window, Linux và Mac. Trong bài viết này, tôi sẽ hướng dẫn cách cài đặt trình thông dịch Python và Pycharm IDE cũng như cách cài thêm các packages bên ngoài.

Bạn có thể tải và cài đặt **python-xyz.exe** tại website

<https://www.python.org/downloads/>. (với xyz là version)

1.3.1.1 Bước 1: Downloading Python

Click [Python Download](#). Version cài đặt trong bài viết này là 3.8.5 (truy cập 9/9/2020) nhưng có thể ta nên bạn nên chọn lùi một vài version.



1. Chọn mục Download → Windows.

The screenshot shows a list of download options for Python 3.8.5:

- No files for this release.
- [Python 3.8.5 - July 20, 2020](#)

Note that Python 3.8.5 cannot be used on Windows XP or earlier.

 - Download [Windows help file](#)
 - Download [Windows x86-64 embeddable zip file](#)
 - **Download Windows x86-64 executable installer**
 - Download [Windows x86-64 web-based installer](#)
 - Download [Windows x86 embeddable zip file](#)
 - Download [Windows x86 executable installer](#)
 - Download [Windows x86 web-based installer](#)
- [Python 3.8.4 - July 13, 2020](#)

Chọn Download **Windows x86-64 executable installer**

2. Click **Download python 3.8.5.amd64.exe** .
3. Sau đó mở file **python-3.8.5.amd64.exe** để bắt đầu **Installing**.

1.3.1.2 Bước 2: Cài đặt Python

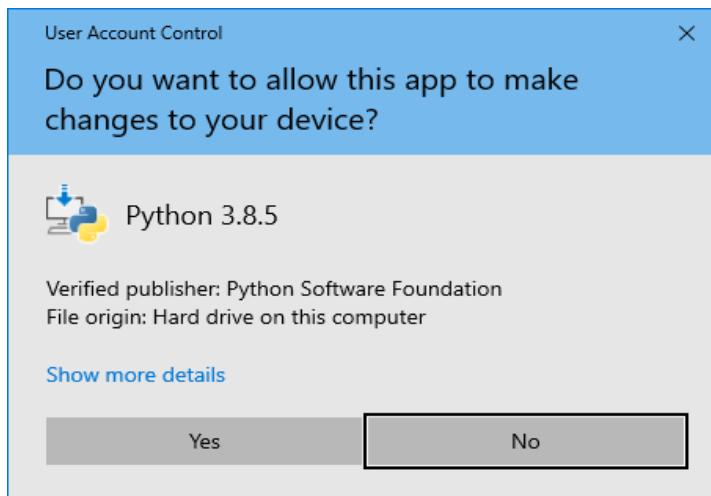
Double-click file cài đặt **python-3.8.5.exe**. Một màn hình xuất hiện.



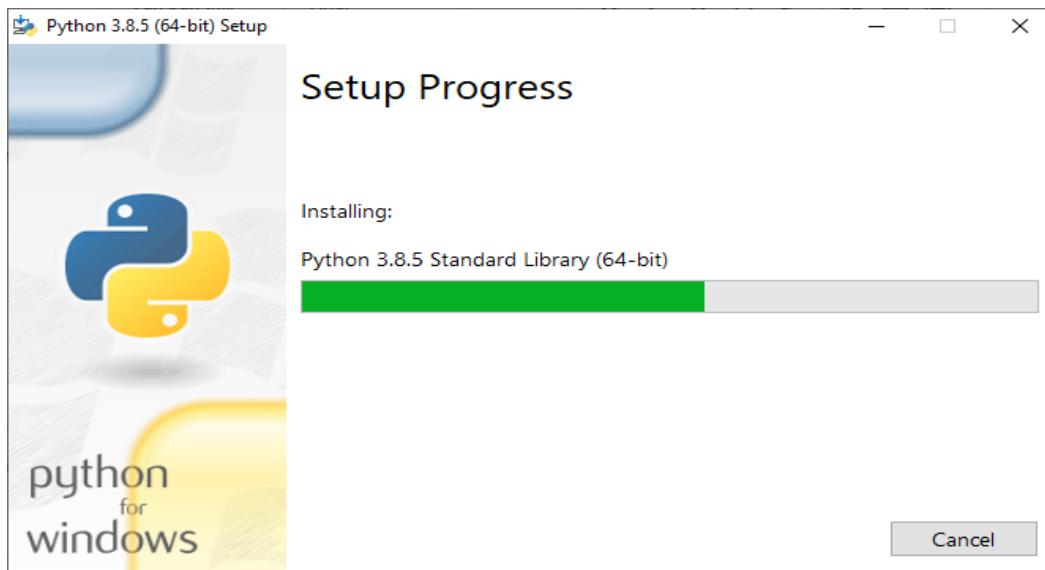
Nên chọn cài Python cho toàn bộ user **Install launcher for all users (recommended)** và chọn **Add Python 3.8 to PATH** để tự setup đường dẫn PATH (để chạy trực tiếp python bằng lệnh python ở bất kỳ đâu).

1. Click chọn **Install Now**.

Khi chạy, cửa sổ **User Account Control** sẽ pop-up để xác nhận **Do you want to allow this app to make changes to your device?**



2. Click chọn **Yes**. Cửa sổ tiền trình cài đặt **Python 3.8.5 (64-bit) Setup** sẽ xuất hiện **Setup Progress**.



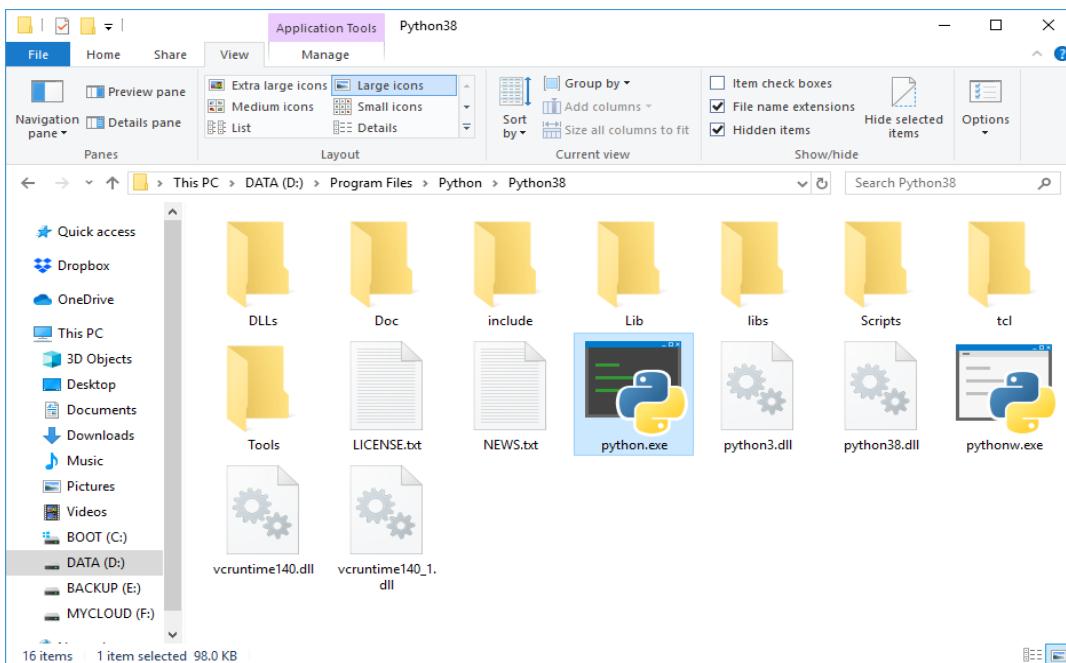
Cuối cùng thông báo cài thành công **Setup was successful**.

3. Click **Close** để kết thúc.

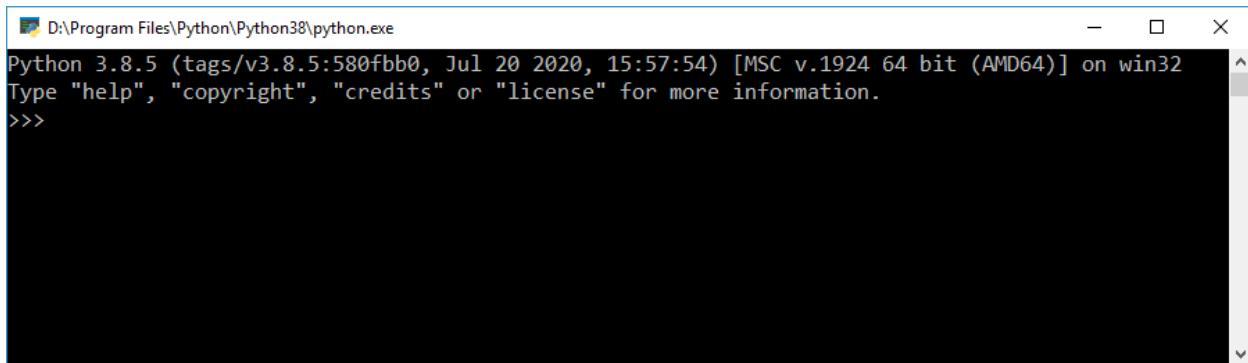
1.3.1.3 Bước 3: Kiểm tra cài đặt thành công



1. Di chuyển đến thư mục Python đã cài đặt. Ví dụ: *D:\Program Files\Python\Python38*



2. Nhấp đúp vào file **python.exe**. Một cửa sổ xuất hiện:



Tại con trỏ **>>>**: gõ lệnh **exit()** để kết thúc.

1.3.2 Giới thiệu Pycharm IDE

Để viết mã nguồn Python, ta có thể sử dụng bất kỳ một trình soạn thảo nào, kể cả những trình soạn thảo đơn giản nhất như NotePad. Bạn có thể dùng các IDE sau: Visual Code, Thony <https://thonny.org/>, Code with Mu <https://codewith.mu/>, PyCharm, Atom, Anaconda, Sublime Text, ...

Tuy nhiên, để phát triển các ứng dụng một cách hiệu quả hơn, ta nên sử dụng một IDE (Môi trường phát triển tích hợp), để có thể tiết kiệm thời gian và công sức viết code.

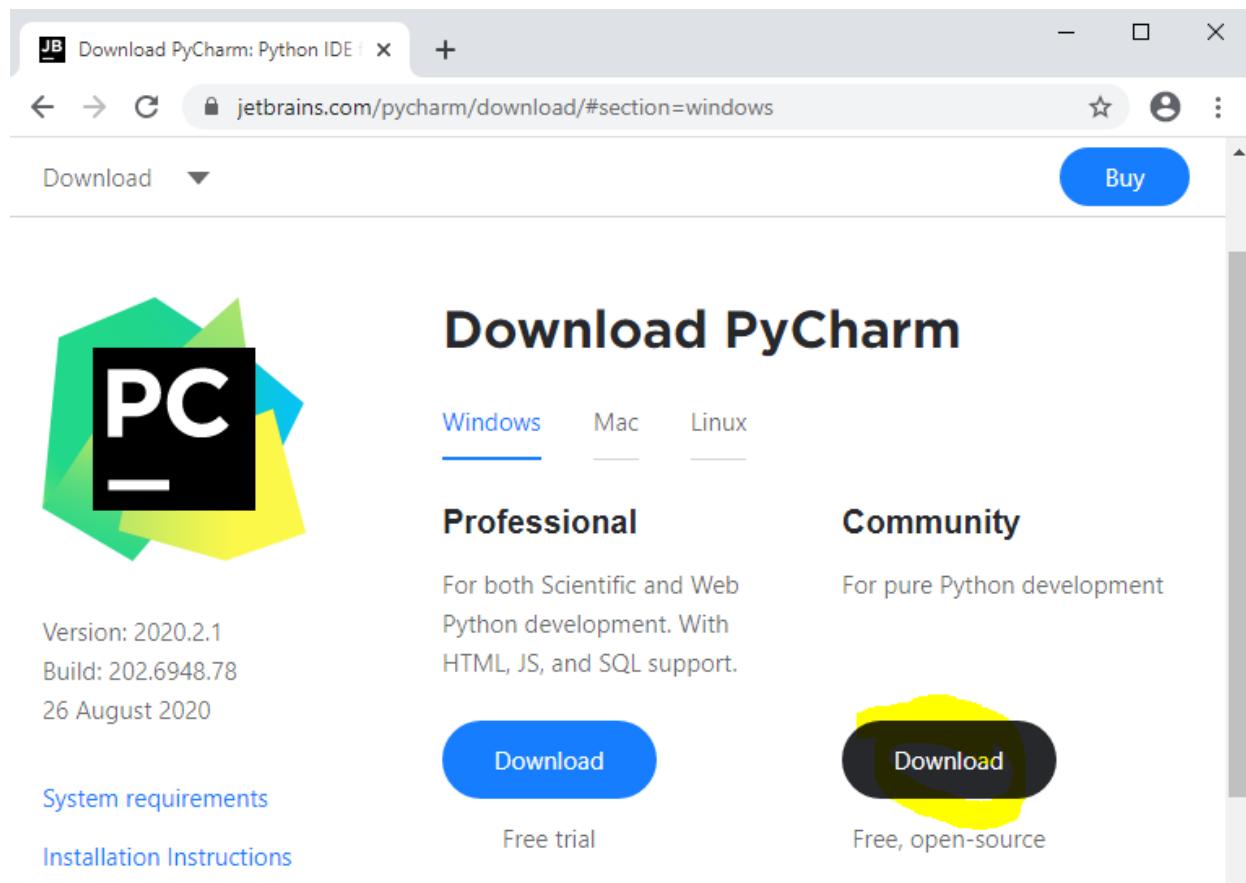
1.3.2.1 Bước 1: Download và cài đặt PyCharm IDE

Bạn thực hiện theo các hướng dẫn tuần tự bên dưới để cài đặt PyCharm IDE về máy của mình.

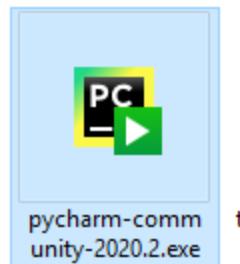
- Ta vào website www.jetbrains.com/pycharm
- Download để tải PyCharm IDE về máy tính cá nhân như hình bên dưới.

Có 2 phiên bản PyCharm:

- Bản **Professional**: Có đầy đủ tất cả các tính năng từ cơ bản đến nâng cao để phát triển Python, nhưng ta phải mua bản quyền. Ta có thể download bản dùng thử.
- Bản **Community**: Là bản chứa các tính năng cơ bản, để có thể phát triển Python. Bản này được tải miễn phí.

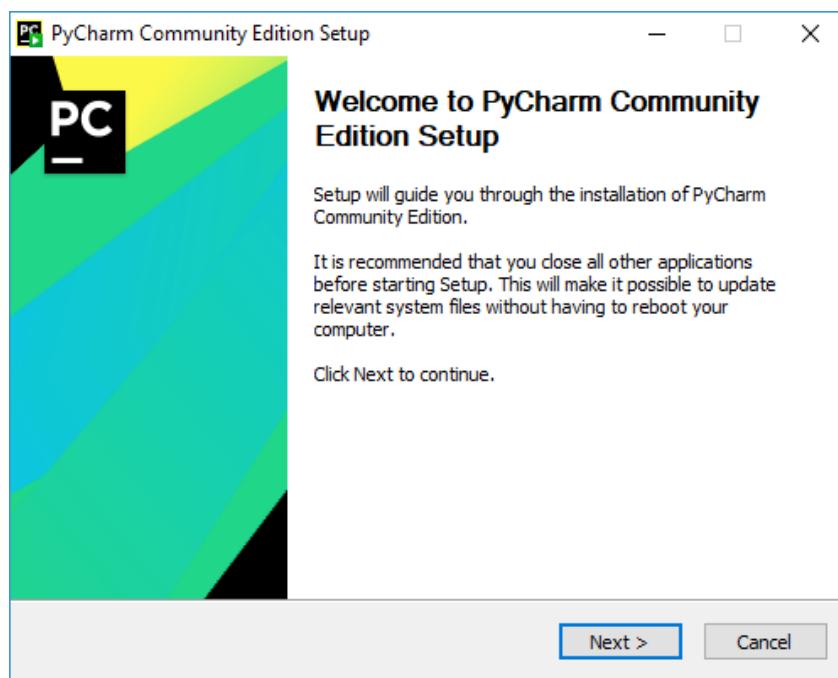


Sau khi download thành công, PyCharm sẽ được lưu tại một thư mục Download của máy tính.



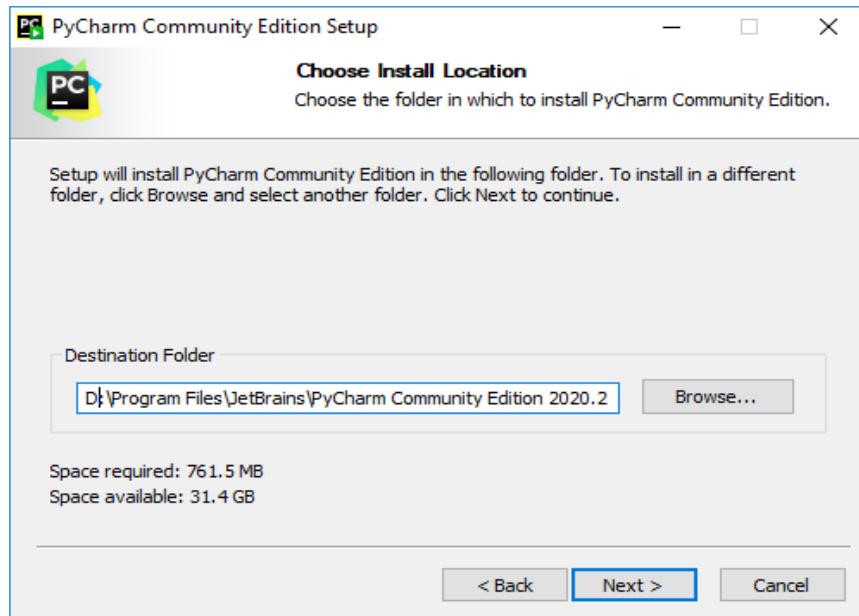
Ta click đúp lên file cài, để tiến hành cài đặt PyCharm.

Màn hình chào mừng được hiển thị, ta nhấn Next để tiếp tục.



Giao diện cài Đặt PyCharm đầu tiên - Chọn Next

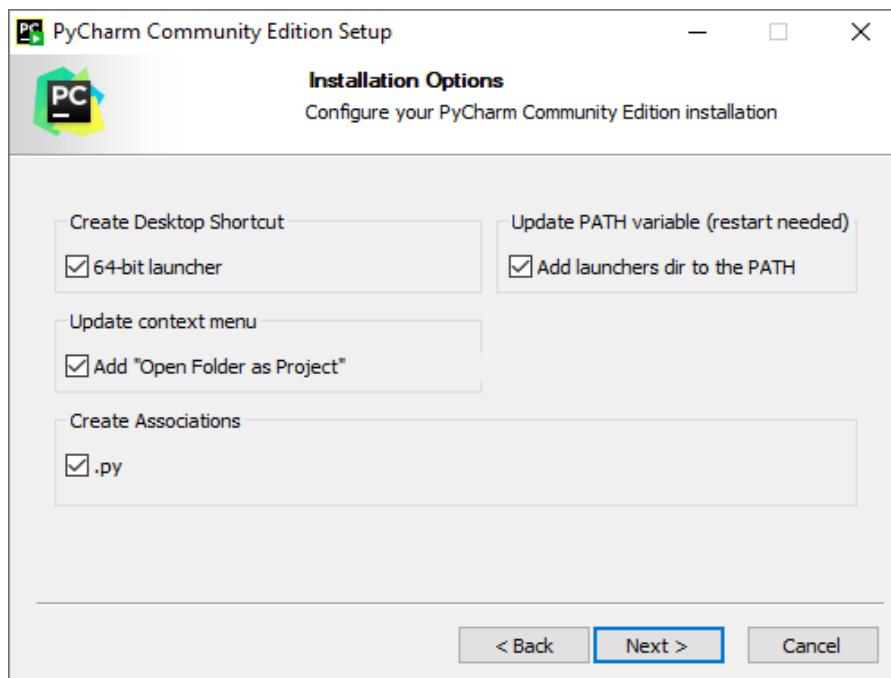
Sau đó, ta chọn đường dẫn thư mục chứa bộ cài nói trên.



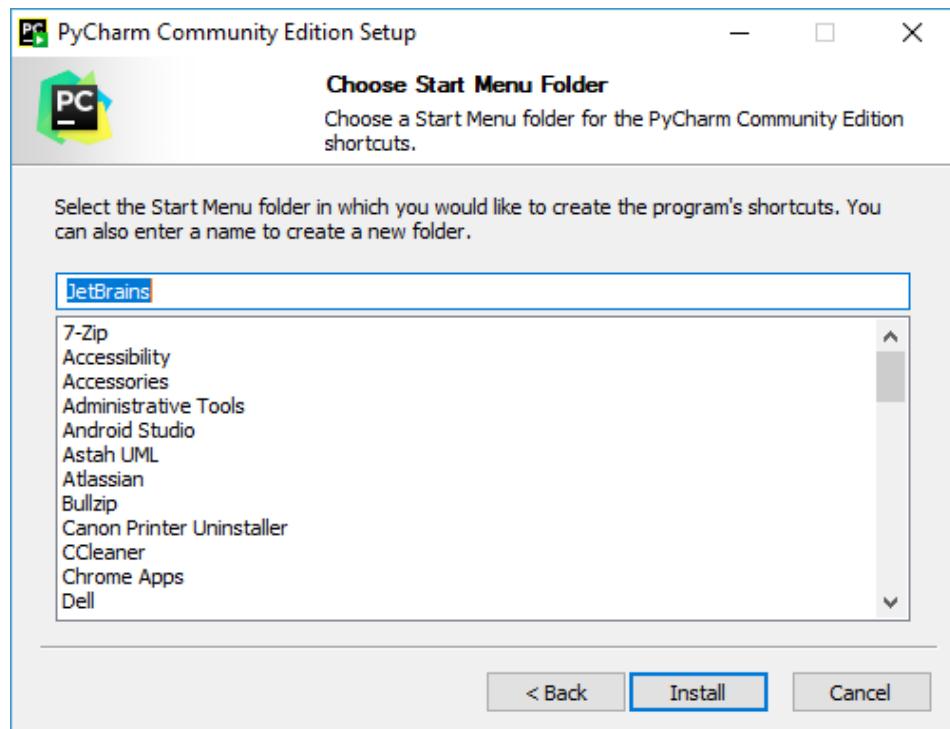
Chọn đường dẫn lưu PyCharm

Tiếp theo, ta chọn các tùy chọn cho việc cài đặt.

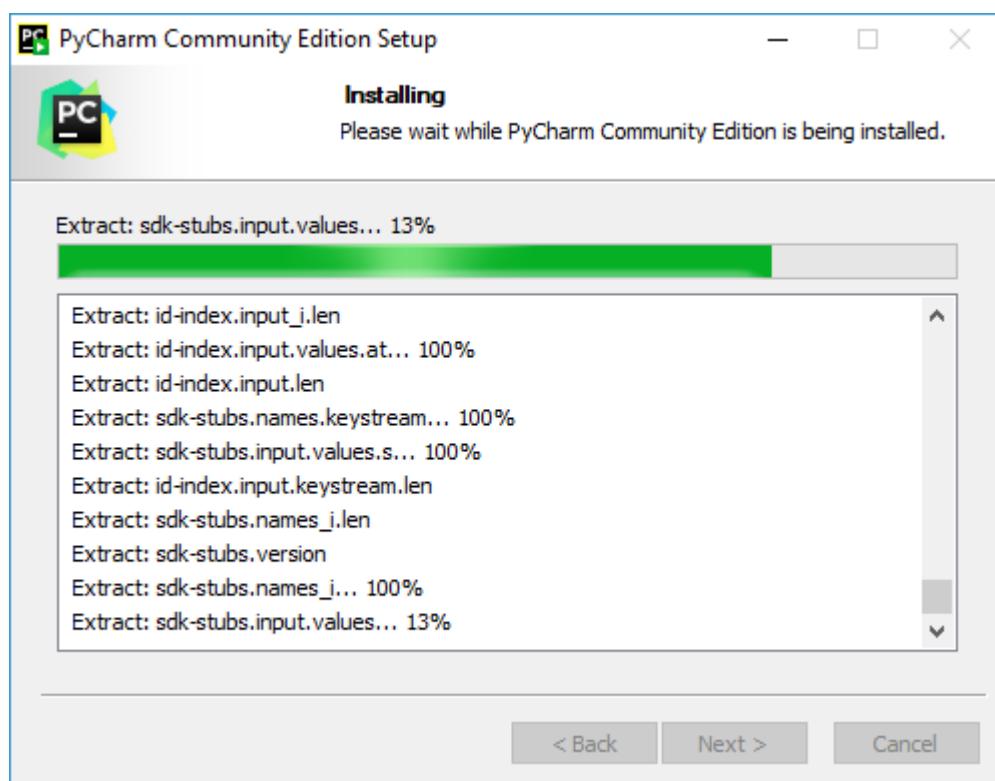
Nếu máy chưa cài đặt Java thì ta tích vào tất cả các tùy chọn trên màn hình này.



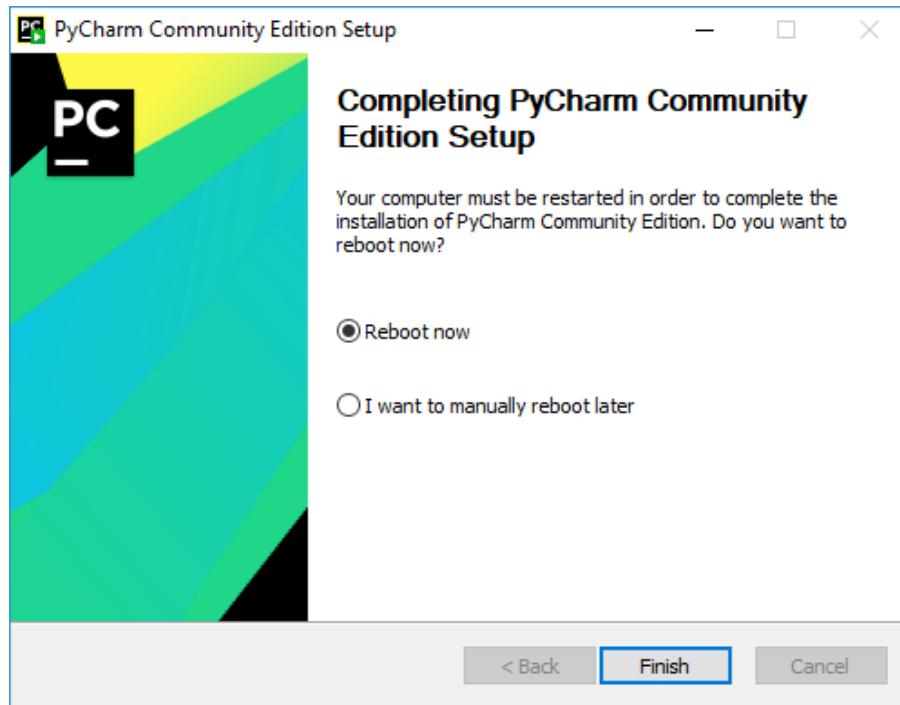
Sau đó ta chọn Install trong màn hình tiếp theo, để bắt đầu tiến hành cài đặt PyCharm.



Lựa chọn Install để cài đặt PyCharm



Sau khi cài đặt xong, PyCharm sẽ hỏi ta có muốn khởi động lại máy luôn hay không. Ta có thể chọn **Reboot now** để khởi động lại máy tính nhằm hoàn tất quá trình cài đặt.

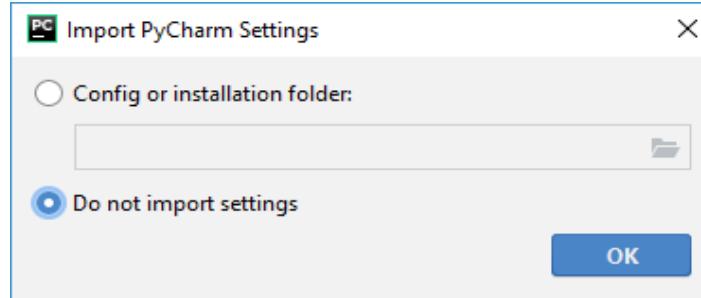


Chọn Reboot now để khởi động lại và hoàn tất cài đặt PyCharm

Như vậy là về cơ bản chúng ta đã cài đặt xong Pycharm.

Sau khi cài xong, mở PyCharm, ta sẽ được hỏi "Có muốn Import các thiết lập đã có từ trước hay không?".

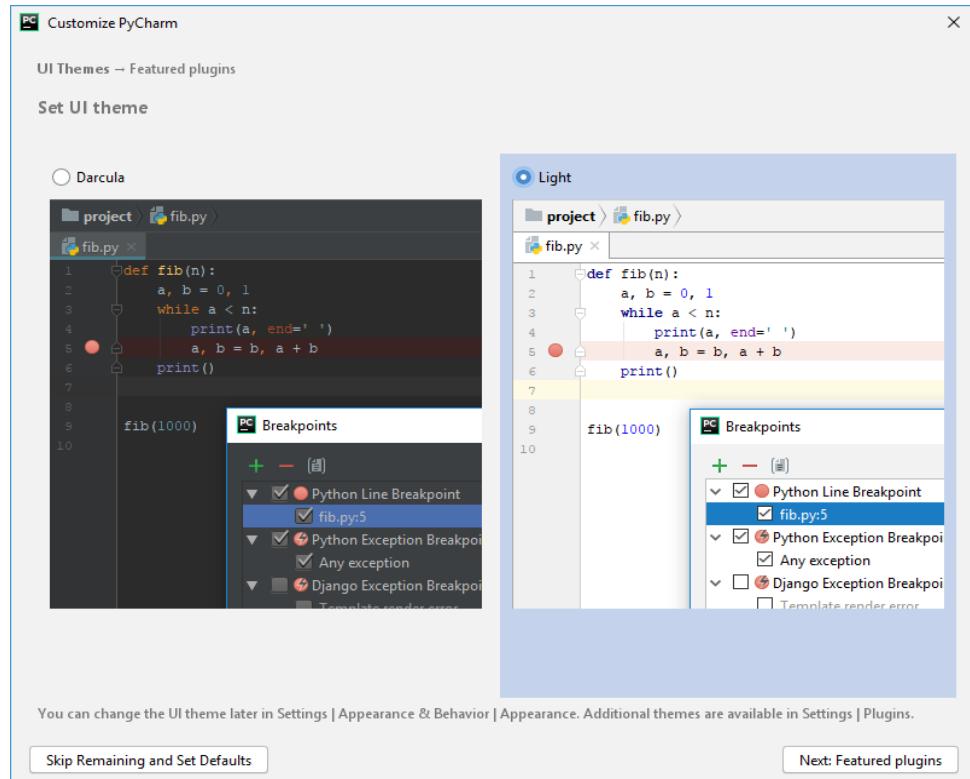
Nếu cài mới hoàn toàn, ta chọn mục **Do not import settings**, rồi nhấn OK.



Chọn Có hay Không Import các cài đặt trước

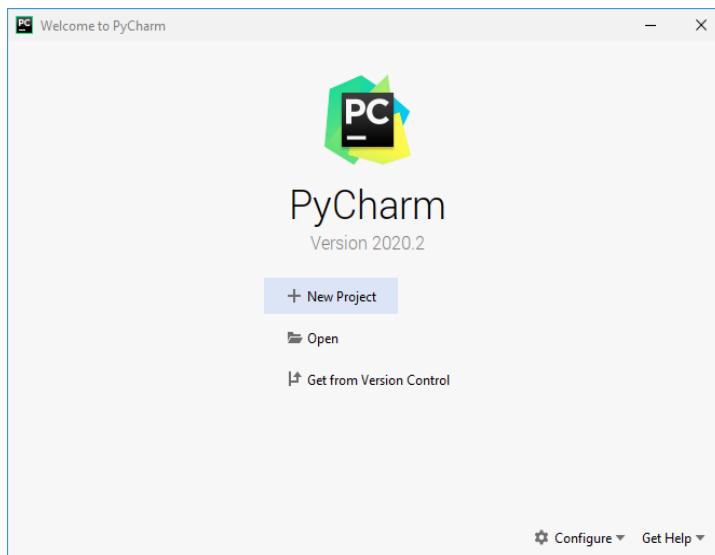
Trong phần chính sách bảo mật, ta nhấn xác nhận và nhấn Continue để tiếp tục.

Trong màn hình Tùy biến PyCharm, ta chọn **Skip Remaining and Set Defaults** để lựa chọn các thiết lập mặc định.



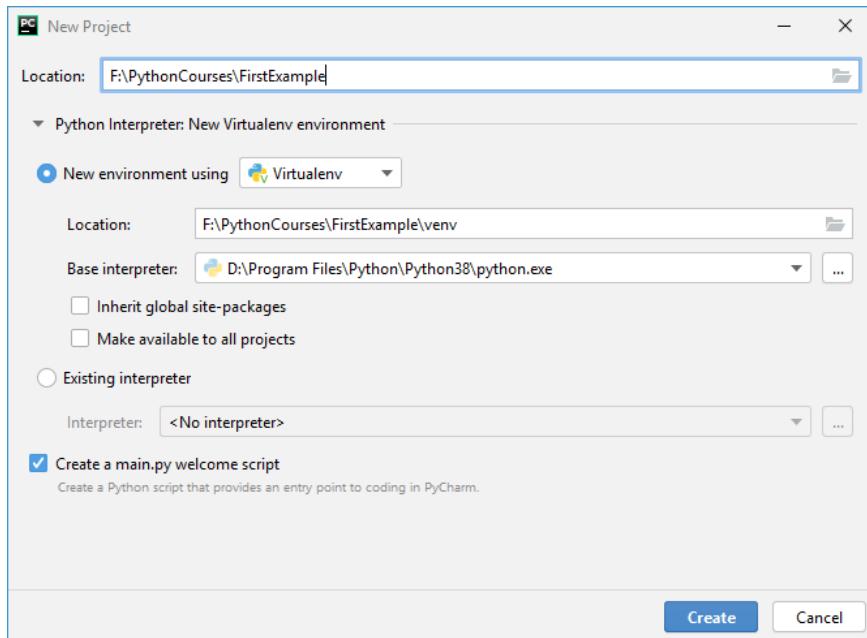
Sử dụng các thiết lập mặc định của PyCharm

Sau đó là màn hình chào hỏi của PyCharm, ta chọn mục **New Project** để tạo một Project mới.



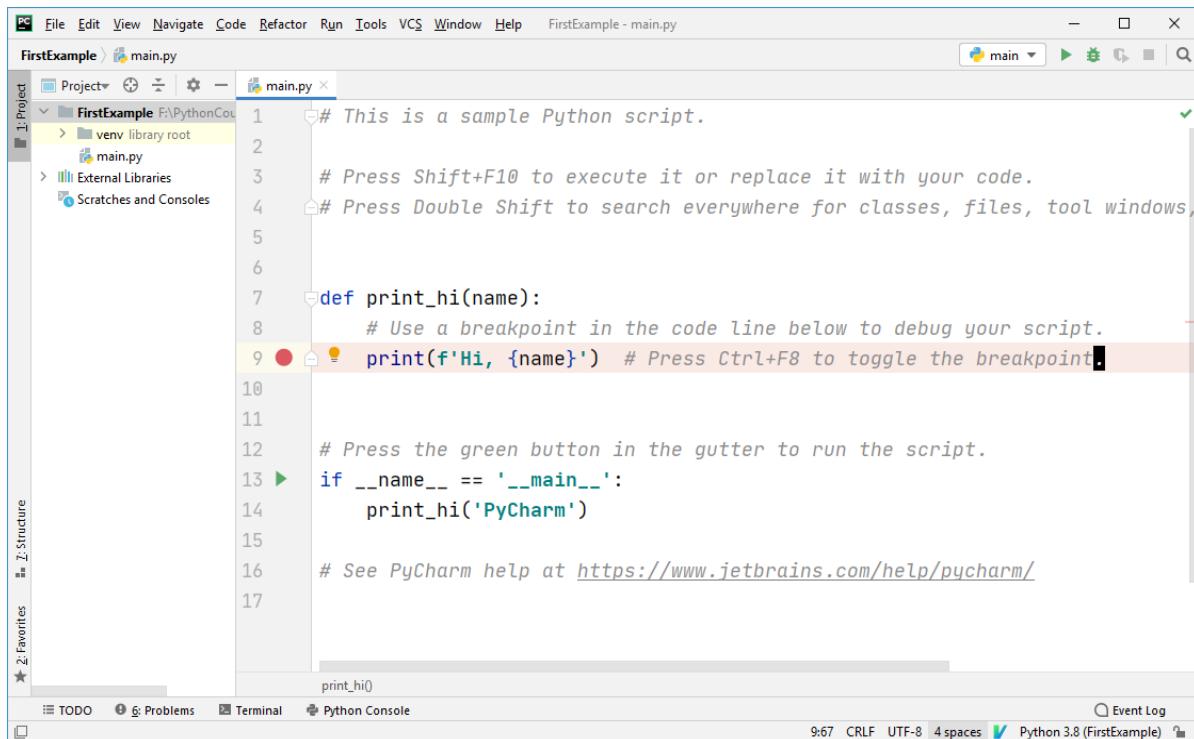
Tạo 1 Project mới trong PyCharm

Ta chọn thư mục chứa Project mới được tạo. Sau đó nhấn **Create**



Chọn thư mục chứa Project và Create

Sau khi quá trình trên được hoàn tất, Project mới sẽ được tạo ra tại PyCharm như hình bên dưới.



Project Python mới được tạo

1.3.2.2 Bước 2: Viết mã Python trên PyCharm

Sau khi tạo xong Project, ta thấy có sẵn file **main.py**. Ta có thể viết thêm lệnh nhập đơn giản mời người dùng nhập vào một chuỗi từ bàn phím, rồi sau đó in ra chuỗi vừa nhập.

Chuỗi này được gán cho một biến có tên là **name**.

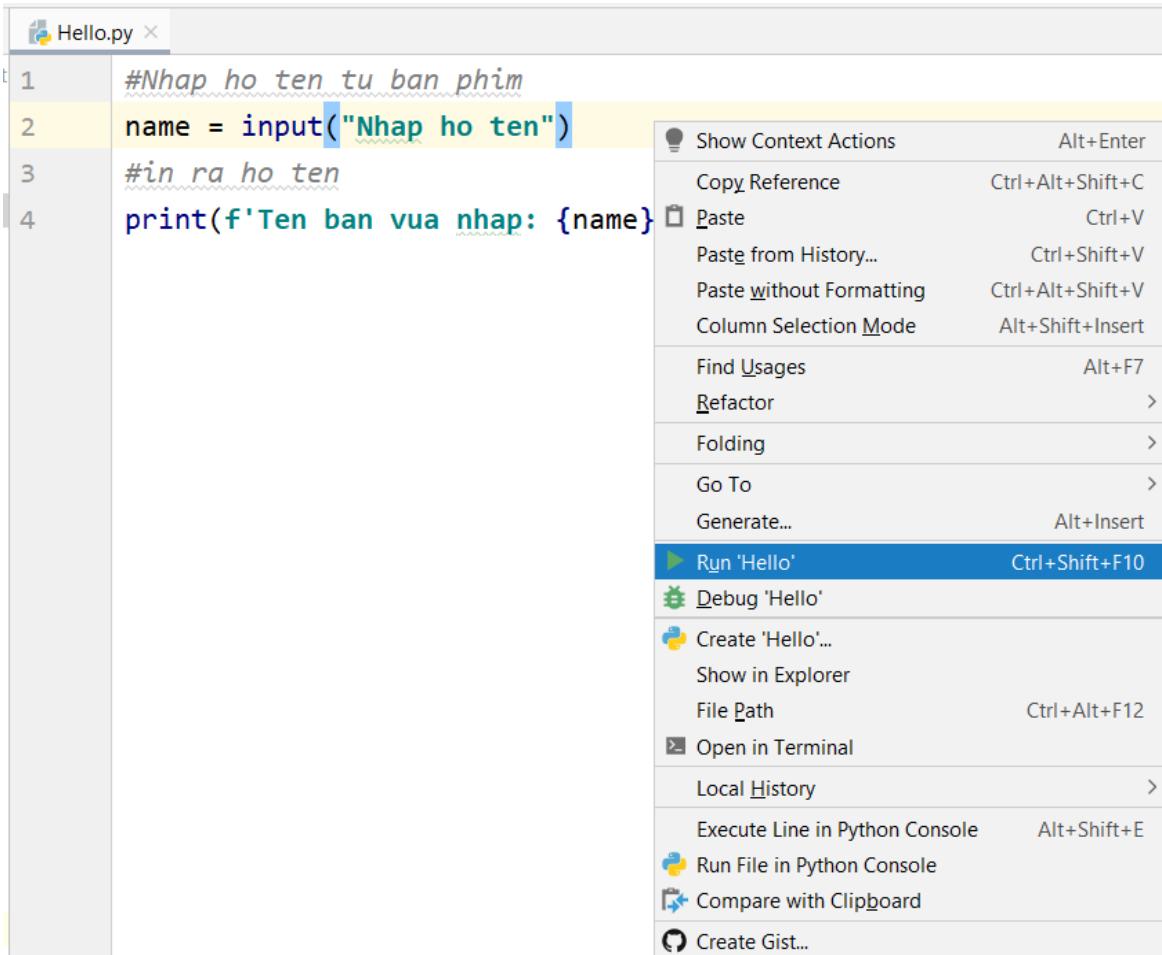
Ta gọi hàm **input()** để đợi người dùng nhập một chuỗi vào từ bàn phím.

```
# Nhập họ tên từ bàn phím
name = input("Nhập họ tên")
# in ra họ tên
print(f'Tên ban vua nhap: {name}')
```

1.3.2.3 Bước 3: Thực thi file Python bằng PyCharm và xem kết quả

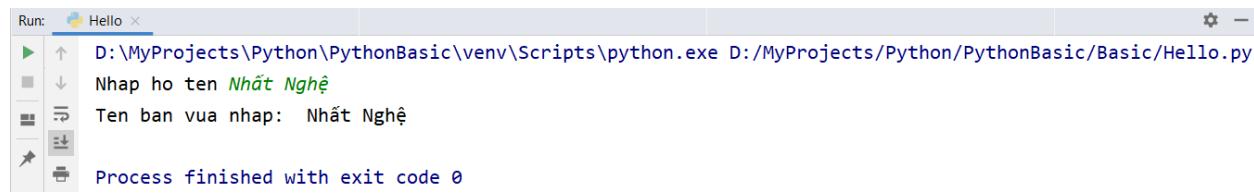
Sau khi đã viết mã xong, ta click phải chuột lên cửa sổ soạn thảo, rồi chọn mục **Run** để thực thi file.

Ta có thể sử dụng tổ hợp phím tắt **Ctrl + Shift + F10** để thực thi file.



Click Run để chạy File Python

Ta có thể thấy kết quả của chương trình được hiển thị như hình bên dưới.



```
Run: Hello ×
▶ D:\MyProjects\Python\PythonBasic\venv\Scripts\python.exe D:/MyProjects/Python/PythonBasic/Basic/Hello.py
Nhập họ tên Nhất Nghệ
Tên bạn vừa nhập: Nhất Nghệ
Process finished with exit code 0
```

Kết quả chạy thử nghiệm File Python bằng PyCharm

1.4 Phân khối và chú thích

1.4.1 Phân khối

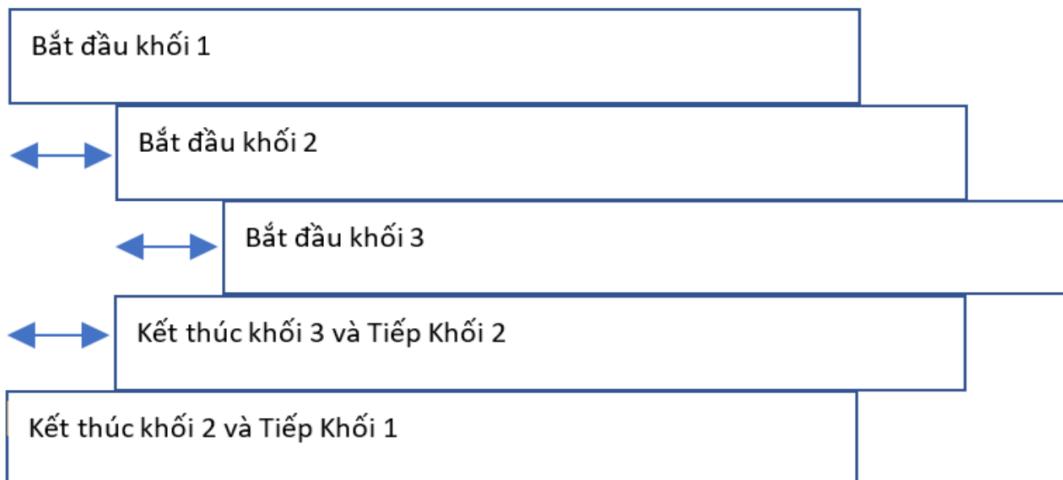
Phần lớn ngôn ngữ sử dụng vài kí tự đặc biệt hoặc từ khóa để nhóm các khối:

- begin ... end
- do ... done
- { ... }
- if ... if

Python sử dụng một nguyên lý khác. Chương trình tổ chức qua thụt lề (sử dụng tab), nghĩa là các khối code được phân chia dựa vào thụt lề. Cách tổ chức này buộc người viết code tạo ra đoạn mã có “coding style” dễ đọc, dễ bảo trì và cũng là mong đợi của bất cứ chương trình nào.

Hình dưới đây mô tả cách phân chia khối trong python.

↔ ↔ 4 dấu cách (hoặc 2 dấu cách)



1.4.2 Chú thích

Khi viết chương trình, bạn có thể thêm các chú thích để giải thích ý nghĩa của các dòng lệnh. Trong Python, các chú thích được đặt sau dấu #, tức toàn bộ phần sau dấu # đến hết cuối dòng sẽ được Python bỏ qua khi chạy chương trình. Ví dụ:

```
print(4 + 5) # Ghi chu 1 dong
```

Để chú thích một khối sử dụng 3 cặp nháy đôi (""" """) hoặc 3 cặp nháy đơn ('' '')

```
'''  
numbers = [15, 21, 24, 30, 84]  
primes = timSoNguyenToLonNhat(numbers)
```

1.5 Python keyword

Python 3.8 cung cấp 35 keywords, bạn có thể xem từng keyword chi tiết bằng cách click vào keyword nhé:

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

Xem thêm tại: <https://realpython.com/python-keywords/>
hay <https://www.programiz.com/python-programming/keyword-list>

No	Keyword	Description	Example
1	False	instance of class bool.	x = False
	True	instance of bool class. This keyword is used to represent a boolean true. If a statement is true, "True" is printed.	x = True
2	class	keyword to define a class.	class Foo: pass
3	from	clause to import class from module	from collections import OrderedDict
4	or	Boolean operator	x = True or False
5	None	instance of NoneType object	x = None
6	continue	continue statement, used in the nested for and while loop. It continues with the next cycle of the nearest enclosing loop.	numbers = range(1,11) for number in numbers: if number == 7: continue
7	global	global statement allows us to modify the variables outside the current scope. This keyword is used to define a variable inside the function to be of a global scope.	x = 0 def add(): global x x = x + 10 add() print(x) # 10
8	pass	Python pass statement is used to do nothing. It is useful when we require some statement but we don't want to execute any code.	def foo(): pass
10	def	keyword used to define a function.	def bar(): print("Hello")
11	if	if statement is used to write conditional code block.	x = 10 if x%2 == 0: print("x is even")

No	Keyword	Description	Example
12	raise	The raise statement is used to throw exceptions in the program.	<pre>def square(x): if type(x) is not int: raise TypeError("Require int argument") print(x * x)</pre>
13	and	Boolean operator for and operation.	<pre>x = True y = False print(x and y) # False</pre>
14	del	The del keyword is used to delete objects such as variables, list, objects, etc.	<pre>s1 = "Hello" print(s1) # Hello del s1 print(s1) # NameError: name 's1' is not defined</pre>
15	import	The import statement is used to import modules and classes into our program.	<pre># importing class from a module from collections import OrderedDict # import module import math</pre>
16	return	The return statement is used in the function to return a value.	<pre>def add(x,y): return x+y</pre>
17	as	Python as keyword is used to provide name for import, except, and with statement.	<pre>from collections import OrderedDict as od import math as m with open('data.csv') as file: pass # do some processing on file try: pass except TypeError as e: pass</pre>
18	elif	The elif statement is always used with if statement for “else if” operation.	<pre>x = 10 if x > 10: print('x is greater than 10') elif x > 100: print('x is greater than 100') elif x == 10:</pre>

No	Keyword	Description	Example
			<pre>print('x is equal to 10') else: print('x is less than 10')</pre>
19	in	Python in keyword is used to test membership.	<pre>l1 = [1, 2, 3, 4, 5] if 2 in l1: print('list contains 2') s = 'abcd' if 'a' in s: print('string contains a')</pre>
20	try	Python try statement is used to write exception handling code.	<pre>x = " try: i = int(x) except ValueError as ae: print(ae) # invalid literal for int() with base 10: "</pre>
21	assert	The assert statement allows us to insert debugging assertions in the program. If the assertion is True, the program continues to run. Otherwise AssertionError is thrown.	<pre>def divide(a, b): assert b != 0 return a / b</pre>
22	else	The else statement is used with if-elif conditions. It is used to execute statements when none of the earlier conditions are True.	<pre>if False: pass else: print('this will always print')</pre>
23	is	Python is keyword is used to test if two variables refer to the same object. This is same as using == operator.	<pre>fruits = ['apple'] fruits1 = ['apple'] f = fruits print(f is fruits) # True print(fruits1 is fruits) # False</pre>
24	while	The while statement is used to run a block of statements till the expression is True.	<pre>i = 0 while i < 3: print(i)</pre>

No	Keyword	Description	Example
			<pre>i+=1 # Output # 0 # 1 # 2</pre>
25	async	New keyword introduced in Python 3.5. This keyword is always used in coroutine function body. It's used with asyncio module and await keywords.	<pre>import asyncio import time async def ping(url): print(f'Ping Started for {url}') await asyncio.sleep(1) print(f'Ping Finished for {url}') async def main(): await asyncio.gather(ping('nhatnghe.co m'),ping('hienlth.info'),) if __name__ == '__main__': then = time.time() loop = asyncio.get_event_loop() loop.run_until_complete(main()) now = time.time() excute_time = now - then print(f'Execution Time = {excute_time}') # Output Ping Started for nhatnghe.com Ping Started for hienlth.info Ping Finished for nhatnghe.com Ping Finished for hienlth.info Execution Time = 1.0181794166564941</pre>
26	await	New keyword in Python 3.5 for asynchronous processing.	Above example demonstrates the use of async and await keywords.
27	lambda	The lambda keyword is used to create lambda expressions.	<pre>multiply = lambda a, b: a * b print(multiply(8, 6)) # 48</pre>

No	Keyword	Description	Example
28	with	Python with statement is used to wrap the execution of a block with methods defined by a context manager. The object must implement <code>__enter__()</code> and <code>__exit__()</code> functions.	<code>with open('data.csv') as file: file.read()</code>
29	except	Python except keyword is used to catch the exceptions thrown in try block and process it.	Please check the try keyword example.
30	finally	The finally statement is used with try-except statements. The code in finally block is always executed. It's mainly used to close resources.	<pre>def division(x, y): try: return x / y except ZeroDivisionError as e: print(e) return -1 finally: print("this will always execute")</pre> <pre>print(division(10, 2)) print(division(10, 0)) # Output this will always execute 5.0 division by zero this will always execute -1</pre>
31	nonlocal	The nonlocal keyword is used to access the variables defined outside the scope of the block. This is always used in the nested functions to access variables defined outside.	<pre>def outer(): v = 'outer' def inner(): nonlocal v v = 'inner' inner() print(v) outer()</pre>

No	Keyword	Description	Example
32	yield	Python yield keyword is a replacement of return keyword. This is used to return values one by one from the function.	<pre>def firstn(n): num = 0 while num <= n: yield num num += 1 print(sum(firstn(5))) # Output 15</pre>
33	break	The break statement is used with nested “for” and “while” loops. It stops the current loop execution and passes the control to the start of the loop.	<pre>number = 1 while True: print(number) number += 2 if number > 5: break # Output 1 3 5</pre>
34	for	Python for keyword is used to iterate over the elements of a sequence or iterable object.	<pre>s1 = 'Nhat Nghe' for c in s1: print(c)</pre>
35	not	The not keyword is used for boolean not operation.	<pre>x = True print(not x) # False</pre>

Python luôn được cập nhật version mới, danh sách các từ khóa trên có thể sẽ không giống với phiên bản bạn đang dùng.

1.6 Biến

1.6.1 Biến là gì

Biến số là khái niệm cơ bản nhất trong các chương trình. Chúng được dùng để lưu giá trị trung gian trong quá trình tính toán. Biến được đặt tên duy nhất để phân biệt giữa các vị trí bộ nhớ khác nhau.

Trong Python, bạn không cần khai báo biến trước khi sử dụng, chỉ cần gán cho biến một giá trị và nó sẽ tồn tại. Cũng không cần phải khai báo kiểu biến, kiểu biến sẽ được nhận tự động dựa vào giá trị mà bạn đã gán cho biến.

1.6.2 Gán giá trị cho biến

Để gán giá trị cho biến ta sử dụng toán tử `=`. Bất kỳ loại giá trị nào cũng có thể gán cho biến hợp lệ.

Ví dụ:

```
ten = "Nhất Nghệ"  
tuoi = 17  
pythonCourse = [2.5, 'Python', 'Nhất Nghệ', 54]
```

Gán nhiều giá trị:

Trong Python bạn có thể thực hiện gán nhiều giá trị trong một lệnh như sau:

```
tenLop, thoiluong, hocPhi = "Lập trình Python cơ bản", 54, 2500
```

1.6.3 Xóa biến khỏi bộ nhớ

Để xóa một biến số khỏi bộ nhớ, chúng ta dùng lệnh:

```
del <ten_bien>
```

1.6.4 Quy tắc đặt tên biến

Một biến có thể có tên ngắn (như `x` và `y`) hoặc tên mô tả đầy đủ hơn (`tuoi`, `tenNhanVien`, `luong`). Quy tắc cho các biến Python:

- Tên biến phải bắt đầu bằng một chữ cái hoặc ký tự gạch dưới (`_`).
- Tên biến không được bắt đầu bằng số.
- Tên biến chỉ có thể chứa các ký tự chữ-số và dấu gạch dưới (A-z, 0-9 và `_`).
- Tên biến phân biệt chữ hoa chữ thường (`age`, `Age` và `AGE` là ba biến khác nhau).
- Không sử dụng các từ khóa có sẵn trong Python để đặt tên biến.

1.7 Nhập xuất dữ liệu trên shell

1.7.1 Nhập với input

Sử dụng lệnh **input()** để nhập dữ liệu từ bàn phím.

```
username = input("Nhập vào họ tên: ")  
print("Tên của bạn là: " + username)
```

 Nhập vào họ tên: *Anh Đào*
 Tên của bạn là: Anh Đào


Trong trường hợp muốn lấy giá trị vào ở dạng số, chúng ta sử dụng các lệnh sau để chuyển đổi từ dữ liệu String sang dữ liệu số:

- **int(text)** : chuyển giá trị text có kiểu String thành số nguyên.
- **float(text)** : chuyển giá trị text có kiểu String thành số thập phân.

Ví dụ:

_Viết chương trình nhập vào từ bàn phím 2 số và in ra tổng của 2 số đó.

```
a = input('Số thứ nhất : ')  
a = float(a)  
  
b = input('Số thứ hai : ')  
b = float(b)  
  
print('Tổng của 2 số là : ', a + b)
```

1.7.2 Xuất với print

1.7.2.1 Lệnh xuất cơ bản

Sau khi tính toán xong một bài toán, chúng ta cần in kết quả ra màn hình. Để thực hiện việc này chúng ta dùng lệnh **print** theo cú pháp:

print(<danh sách giá trị ngăn cách nhau bởi dấu phẩy>)

Ví dụ:

```
x = 1  
y = 2  
z = x + y  
print(x, '+', y, '=', z)
```

Ở dòng cuối trong chương trình trên, danh sách các giá trị được in ra nằm trong lệnh print, theo thứ tự là:

```
x    →  1  
'+' → +  
y    →  2  
'=' → =  
z    →  3
```

Như vậy, thông tin được in ra trên màn hình là:

```
1 + 2 = 3
```

Ngoài ra có thể dùng lệnh sau:

```
print(f'{x} + {y} = {z}')  
print(f'{x} + {y} = {x+y}')
```

1.7.2.2 Python f-string

Python f-string (được support từ Python 3.6) là cú pháp Python mới nhất để thực hiện định dạng chuỗi. f-string cung cấp cách định dạng chuỗi nhanh hơn, dễ đọc hơn, ngắn gọn hơn và ít mắc lỗi hơn trong Python.

f-string có tiền tố f và sử dụng dấu ngoặc {} để đánh giá giá trị.

f-string dạng cho biến:

```
print(f'{name} is {age} years old')
```

f-string dạng biểu thức:

```
bags = 5  
apples_in_bag = 15  
print(f'There are total of {bags * apples_in_bag} apples')
```

f-string dạng từ điển (dictionaries):

```
user = {'name': 'HIENLTH', 'occupation': 'trainner'}  
print(f"{user['name']} is a {user['occupation']}")
```

f-string dạng debug: tự tính toán giá trị biểu thức bằng toán tử =

```
import math  
x = 0.8  
print(f'{math.cos(x) = }')  
print(f'{math.sin(x) = }')
```

f-string dạng multiline:

```

name = 'HIENLTH'
age = 39
occupation = 'trainner'

msg = (
    f'Name: {name}\n'
    f'Age: {age}\n'
    f'Occupation: {occupation}'
)

print(msg)

```

f-string dạng gọi **hàm**:

```

def find_max(x, y):
    return x if x > y else y

a = 3
b = 4
print(f'Max of {a} and {b} is {find_max(a, b)}')

```

f-string dạng **object**: f-string chấp nhận object; các object phải có hoặc là `__str__()` hoặc `__repr__()`

```

class User:
    def __init__(self, name, occupation):
        self.name = name
        self.occupation = occupation

    def __repr__(self):
        return f"{self.name} is a {self.occupation}"

u = User('HIENLTH', 'trainner')
print(f'{u}')

```

f-string dạng format datetime:

```

import datetime
now = datetime.datetime.now()
print(f'{now:%Y-%m-%d %H:%M}')

```

f-string định dạng float:

```
val = 12.3

print(f'{val:.2f}')
print(f'{val:.5f}')
```

f-string định dạng độ rộng:

```
for x in range(1, 11):
    print(f'{x:02} {x*x:3} {x*x*x:4}')
```

f-string định dạng căn lề:

```
s1 = 'a'
s2 = 'ab'
s3 = 'abc'
s4 = 'abcd'

print(f'{s1:>10}')
print(f'{s2:>10}')
print(f'{s3:>10}')
print(f'{s4:>10}')
```

f-string định dạng biểu diễn số:

```
a = 300

# hexadecimal
print(f"{a:x}")

# octal
print(f"{a:o}")

# scientific
print(f"{a:e}")
```

Bài 2: Kiểu dữ liệu, Toán tử

2.1 Các kiểu dữ liệu

Trong Python cũng như các ngôn ngữ lập trình khác, mỗi biến số sau khi được khai báo sẽ chứa giá trị nhất định, gọi là dữ liệu. Giá trị dữ liệu này sẽ thuộc vào một trong các kiểu dữ liệu mà Python hỗ trợ. Python hỗ trợ các loại kiểu dữ liệu sau:

Kiểu dữ liệu

`str`

Kiểu Numeric

`int, float, complex`

Kiểu Sequence

`list, tuple, range`

Kiểu Mapping

`dict`

Kiểu Set

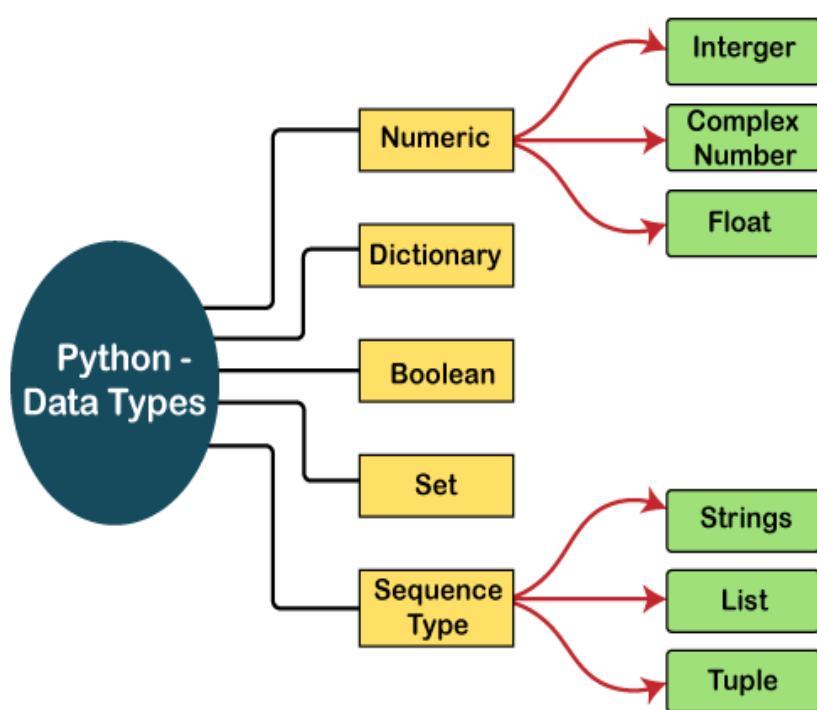
`set, frozenset`

Kiểu Boolean

`bool`

Kiểu Binary

`bytes, bytearray, memoryview`



2.1.1 Xác định id đối tượng

Tất cả các đối tượng trong Python đều có id duy nhất của riêng nó. Id được gán cho đối tượng khi nó được tạo. Để xác định id, sử dụng hàm id()

```
x = 5  
print(id(x))
```

2.1.2 Xác định kiểu dữ liệu

Sử dụng hàm type() để xác định kiểu dữ liệu.

```
x = 5  
print(type(x))
```

Ví dụ:

Ví dụ	Kiểu (type(x))
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = range(6)	range
x = {"name" : "John", "age" : 36}	dict
x = {"apple", "banana", "cherry"}	set
x = frozenset({"apple", "banana", "cherry"})	frozenset
x = True	bool
x = b"Hello"	bytes
x = bytearray(5)	bytearray
x = memoryview(bytes(5))	memoryview

2.1.3 Kiểu số

Đây là kiểu dữ liệu sử dụng nhiều nhất. Python hỗ trợ số nguyên (int), số thập phân (float) và số phức (complex). Số nguyên và số thập phân được phân biệt bằng sự có mặt hoặc vắng mặt của dấu thập phân. Ví dụ: 5 là số nguyên, 5.0 là số thập phân.

Kiểu nguyên (int) trong Python không giới hạn số ký tự mà chỉ phụ thuộc vào bộ nhớ máy tính. Kiểu thực (float) trong Python có giới hạn tối đa 15 chữ số phần thập phân.

Python cũng hỗ trợ số phức và sử dụng hậu tố j hoặc J để chỉ phần ảo. Ví dụ: 3+5j. Ngoài int và float, Python hỗ trợ thêm 2 loại số nữa là Decimal và Fraction.

Ta sẽ dùng hàm type() để kiểm tra xem biến hoặc giá trị thuộc lớp số nào và hàm isinstance() để kiểm tra xem chúng có thuộc về một class cụ thể nào không.

Ví dụ:

```
a = 5
print("kiểu dữ liệu của", a, "là", type(a))

a = 2.0
print("kiểu dữ liệu của", a, "là", type(a))

a = 1+2j
print("kiểu dữ liệu của", a, "là", type(a))
print(a, "là kiểu số phức ==>", isinstance(1+2j,complex))
```

Kết quả chạy:

```
kiểu dữ liệu của 5 là <class 'int'>
kiểu dữ liệu của 2.0 là <class 'float'>
kiểu dữ liệu của (1+2j) là <class 'complex'>
(1+2j) là kiểu số phức ==> True
```

Các hàm chuyển đổi kiểu:

- String sang Integer: int(chuỗi)
- Integer sang string: str(số)
- Float sang Integer: int(số_thực)

2.1.4 Kiểu chuỗi

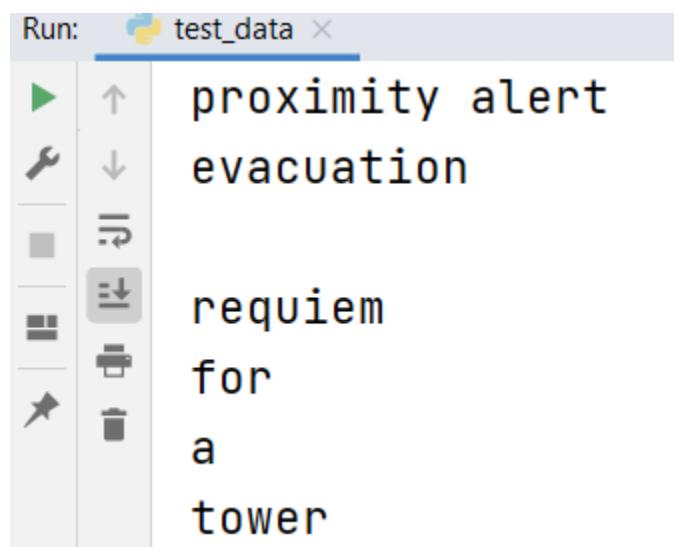
String là kiểu dữ liệu lưu trữ văn bản.

Chúng ta có thể tạo ra một string bằng dấu nháy đơn, nháy kép hay 3 dấu nháy kép. Khi dùng 3 dấu nháy kép, chúng ta cũng có thể ghi một chuỗi trên nhiều dòng mà không cần dùng dấu \.

```
a = "proximity alert"
b = 'evacuation'
c = """
requiem
for
a
tower
"""

print (a)
print (b)
print (c)
```

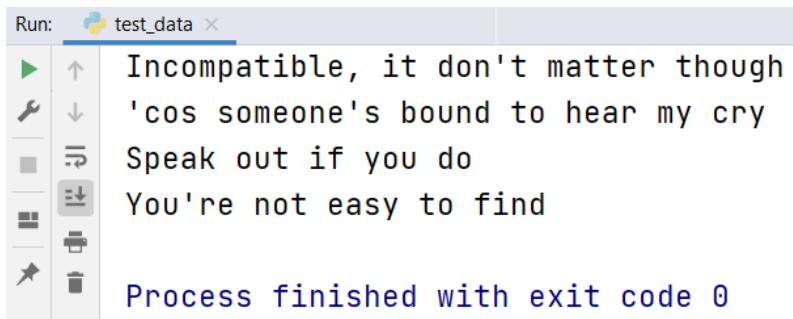
Trong ví dụ trên chúng ta gán 3 chuỗi vào 3 biến a, b, c rồi in ra màn hình.



Trong một chuỗi chúng ta có thể dùng các ký tự thoát. Ký tự thoát là các ký tự đặc biệt dùng cho nhiều mục đích khác nhau. Xem ví dụ.

```
print ("Incompatible, it don't matter though\n'cos someone's bound to
hear my cry")
print ("Speak out if you do\nYou're not easy to find")
```

Ký tự `\n` là ký tự xuống dòng, các đoạn text sau ký tự này sẽ tự động xuống dòng.



The screenshot shows a Python terminal window titled "test_data". The code printed is:

```
Incompatible, it don't matter though
'cos someone's bound to hear my cry
Speak out if you do
You're not easy to find

Process finished with exit code 0
```

Tiếp theo chúng ta tìm hiểu về ký tự xóa.

```
print ("Python\b\b\booo") # prints Pytoo
```

Ký tự `\b` xóa 1 ký tự, trong ví dụ trên, chúng ta dùng 3 ký tự `\b`, do đó 3 ký tự “hon” sẽ bị xóa để nhường chỗ cho 3 ký tự “ooo”.

```
print ("Towering\tinferno") # prints Towering
inferno
```

Dòng code trên ví dụ về ký tự tab `\t`, nó hoạt động giống như khi bạn bấm phím Tab vậy.

```
"Johnie's dog"
'Johnie\'s dog'
```

Đôi khi bạn ghi chuỗi trong cặp dấu nháy đơn, và bản thân bên trong chuỗi này bạn cũng cần dùng một dấu nháy đơn khác, lúc này bạn phải thêm một dấu `\` trước dấu nháy đơn đó, nếu không trình biên dịch sẽ báo lỗi.

Nếu bạn không muốn sử dụng các ký tự thoát thì bạn thêm `r` vào trước chuỗi của mình. Các ký tự thoát sẽ được in ra như các ký tự bình thường.

```
print (r"Another world\nhas come")
```

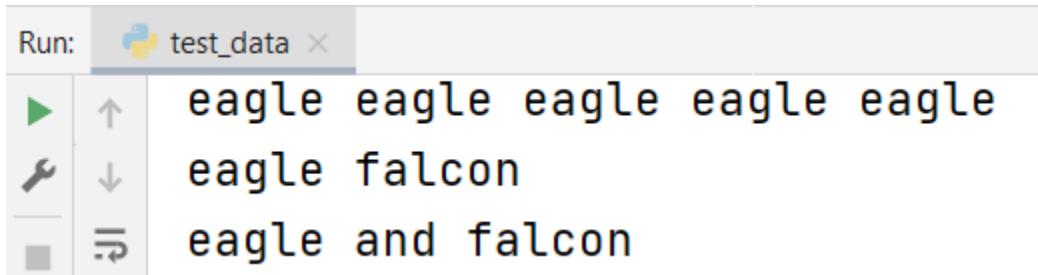
```
Another world\nhas come
```

Như ở trên dòng chữ Another world\n sẽ được in ra.

Tiếp theo chúng ta sẽ tìm hiểu về cách nhân chuỗi và nối chuỗi.

```
print ("eagle " * 5)
print ("eagle " "falcon")
print ("eagle " + "and " + "falcon")
```

Phép nhân * có thể được dùng cho một chuỗi, lúc này nội dung chuỗi sẽ được lặp lại n lần, trong đoạn code trên chữ “eagle” được lặp lại 5 lần. Hai chuỗi đẽ sát nhau sẽ ngầm tự động được nối vào. Và nếu bạn muốn nối chuỗi một cách rõ ràng hơn thì bạn có thể dùng toán tử +.



```
Run: test_data
▶ eagle eagle eagle eagle eagle
🔗 eagle falcon
🔗 eagle and falcon
```

Để chuyển kiểu dữ liệu bất kỳ sang kiểu chuỗi dùng hàm str():

Ví dụ:

```
>>> str(100)
'100'
```

2.1.5 Kiểu Boolean

Còn được gọi là kiểu logic, hay kiểu “đúng sai”. Tức là kiểu giá trị này chỉ có hai giá trị là đúng (**True**) và sai (**False**), nhưng thực ra thì đối với những người đã lập trình lâu năm thì họ thường gọi là kiểu boolean (hoặc kiểu bool) luôn cho tiện. Hầu như tất cả các ngôn ngữ lập trình đều hỗ trợ kiểu này.

Ví dụ.

```
import random

male = False
male = bool(random.randint(0, 1))

print(male)
```

Đoạn code trên có sử dụng module random để tạo số ngẫu nhiên.

```
import random
```

Để sử dụng module này thì chúng ta thêm dòng import random vào đầu chương trình.

```
import random
male = bool(random.randint(0, 1))
print(male)
```

Và ở đây chúng ta sẽ sử dụng phương thức randint(x, y), phương thức này sẽ trả về giá trị ngẫu nhiên từ x→y, ở đây là 0→1. Sau đó chúng ta chuyển kiểu dữ liệu từ kiểu int sang kiểu bool bằng cách bao bọc lấy phương thức này bằng bool(). Nếu giá trị trả về là 0 thì sẽ được chuyển thành False, ngược lại là True.

Đoạn code dưới đây sẽ cho chúng ta biết các dạng dữ liệu khác khi được chuyển sang kiểu bool sẽ có giá trị nào.

print(bool(True))	True
print(bool(False))	False
print(bool("text"))	True
print(bool(""))	False
print(bool(' '))	True
print(bool(0))	False
print(bool())	False
print(bool(3))	True
print(bool(None))	False
print(bool(["apple", "cherry", "banana"]))	True
print(bool([]))	False

2.1.6 Kiểu None

Đây là một kiểu đặc biệt trong Python. Ý nghĩa của kiểu này là không có giá trị gì cả, không tồn tại, rỗng...v..v.

```
def function():
    pass
```

```
print(function())
```

Trong đoạn code trên, chúng ta định nghĩa một hàm. Chúng ta sẽ tìm hiểu về hàm ở các bài sau. Hàm này cũng không trả về giá trị gì cả nên nó sẽ tự động ngầm trả về giá trị **None**.

2.2 Các loại toán tử

2.2.1 Toán tử số học

Python cũng hỗ trợ một số toán tử toán học thông dụng như:

Toán tử	Công dụng	Ví dụ

+	Toán tử cộng 2 giá trị.	$7 + 3 = 10$
-	Toán tử trừ 2 giá trị.	$7 - 3 = 4$
*	Toán tử nhân 2 giá trị.	$7 * 3 = 21$
/	Toán tử chia 2 giá trị.	$7 / 3 = 2.3333333333333335$
//	Toán tử chia lấy phần nguyên.	$7 // 3 = 2$ $10 // 6 = 1$
%	Toán tử chia lấy phần dư (modulo).	$7 \% 3 = 1$ $10 \% 6 = 4$
**	Toán tử lũy thừa/mũ ($a^{**}b = a^b$)	$2^{**} 3 = 8$ $5^{**} 7 = 78125$

2.2.2 Toán tử so sánh

Dùng để so sánh 2 giá trị với nhau, kết quả của phép toán này là True hoặc False (đúng hoặc sai).

Toán tử	Mô tả	Ví dụ
>	Toán thử lớn hơn - nếu số hạng bên trái lớn hơn số hạng bên phải thì kết quả sẽ là True	$3 > 5$ (False)
<	Toán tử nhỏ hơn - nếu số hạng bên trái nhỏ hơn số hạng bên phải thì kết quả sẽ là True	$3 < 5$ (True)
==	Toán tử bằng với - nếu hai số hạng có giá trị bằng nhau thì kết quả sẽ là True.	$3 == 3$ (True)
!=	Toán tử khác bằng - nếu hai số hạng có giá trị khác nhau thì kết quả sẽ là True.	$3 != 3$ (False)

<code>>=</code>	Toán tử lớn hơn hoặc bằng - nếu số hạng bên trái lớn hơn hoặc bằng số hạng bên phải thì kết quả sẽ là True	<code>7 >= 6</code> (True)
<code><=</code>	Toán tử nhỏ hơn hoặc bằng - nếu số hạng bên trái nhỏ hơn hoặc bằng số hạng bên phải thì kết quả sẽ là True	<code>5 <= 6</code> (true)

2.2.3 Toán tử Gán

Python cho phép sử dụng các phép tính dạng rút gọn: `+=`, `-=`, `*=`, `/=`, `//=`, `%=`, `**=`

Loại toán tử	Mục đích	Cách dùng
<code>=</code>	Gán giá trị của về phải cho về trái	<code>x = 5</code>
<code>+=</code>	Tăng giá trị của về phải sau đó gán cho về trái	<code>x += 5</code> <code>(x = x + 5)</code>
<code>-=</code>	Giảm giá trị của về phải sau đó gán cho về trái	<code>x -= 5</code> <code>(x = x - 5)</code>
<code>*=</code>	Nhân giá trị của về phải trước sau đó gán cho về trái	<code>x *= 5</code> <code>(x = x * 5)</code>
<code>/=</code>	Chia giá trị của về phải sau đó gán cho về trái (chia nguyên)	<code>x /= 5</code> <code>(x = x / 5)</code>

<code>%=</code>	Chia giá trị của vế phải sau đó gán cho vế trái (chia lấy dư)	$x \%= 5$ $(x = x \% 5)$
<code>//=</code>	Phép chia lấy phần nguyên.	$x //= 5$ $(x = x // 5)$
<code>**=</code>	Tính số mũ của vế phải sau đó gán giá trị của vế trái	$x **= 5$ $(x = x ** 5)$
<code>&=</code>	Thực hiện phép toán của toán tử AND của vế phải sau đó gán cho vế trái	$x &= 5$ $(x = x & 5)$
<code> =</code>	Thực hiện phép toán của toán tử OR của vế phải sau đó gán cho vế trái	$x = 5$ $(x = x 5)$
<code>^=</code>	Thực hiện phép toán của toán tử XOR của vế phải sau đó gán cho vế trái	$x ^= 5$ $(x = x ^ 5)$
<code>>>=</code>	Thực hiện phép toán dịch phải của vế phải sau đó gán cho vế trái	$x >>= 5$ $(x = x >> 5)$
<code><<=</code>	Thực hiện phép toán dịch trái của vế phải sau đó gán cho vế trái	$x <<= 5$ $(x = x << 5)$

2.2.4 Toán tử Logic

Toán tử logic `not`, `or` và `and` là các toán tử được dùng để kết hợp các mệnh đề lại với nhau. Bảng thể hiện toán tử logic:

Loại toán tử	Mục đích	Cách dùng
and	Trả về True nếu hai điều kiện cùng đúng, ngược lại trả về False	a and b
or	Trả về True nếu có ít nhất một điều kiện đúng, ngược lại nếu cả hai điều kiện đều sai thì trả về False	a or b
not	Toán tử phủ định, toán tử này trả về False nếu điều kiện là True, ngược lại nếu điều kiện là False thì trả về True	not a

2.2.5 Toán tử định danh

Toán tử định danh (`identity`) được dùng để xác định xem hai biến có đang trỏ tới cùng một đối tượng hay không. Với các kiểu dữ liệu như `int`, `str`, `float`,... thì toán tử này tương đương với toán tử `==`. Bạn sẽ được học về sự khác nhau giữa hai toán tử này ở các bài sau.

Toán tử	Mô tả	Ví dụ
is	Trả về true nếu 2 biến cùng trỏ về 1 đối tượng giống nhau.	x is y, trả về true nếu <code>id(x) = id(y)</code> .
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, trả về false nếu <code>id(x) != id(y)</code> .

Ví dụ:

`a = 5`

`b = 7`

```
print(a is b)
print(a is not b)
```

Kết quả khi chạy chương trình:

False

True

2.2.6 Toán tử membership

Thứ tự ưu tiên của các toán tử là điều rất quan trọng các bạn cần nắm được khi thực hiện các phép toán kết hợp nhiều toán tử. Các toán tử được liệt kê trong bảng theo thứ tự ưu tiên từ cao đến thấp.

Toán tử	Mô tả	Ví dụ
in	Trả về true nếu tìm thấy biến a trong tập các giá trị sau in, ngược lại trả về false.	x in y, trả về true .
not in	Trả về false nếu tìm thấy biến a trong tập các giá trị sau in, ngược lại trả về false.	x in y, trả về false. Vì x tồn tại trong tập y.

2.2.7 Độ ưu tiên toán tử

Thứ tự ưu tiên của các toán tử là điều rất quan trọng các bạn cần nắm được khi thực hiện các phép toán kết hợp nhiều toán tử. Các toán tử được liệt kê trong bảng theo thứ tự ưu tiên từ cao đến thấp.

	Toán tử	Mô tả
1	**	Toán tử mũ
2	~ + -	Đảo bit (lấy phần bù); phép cộng, trừ một ngôi
3	~ / % //	Phép nhân, chia, lấy phần dư và phép chia //
4	+ -	Toán tử cộng, trừ
5	>> <<	Dịch bit phải và dịch bit trái
6	&	Phép Và Bit
7	^	Phép XOR và OR
8	<= < > >=	Các toán tử so sánh
9	<> == !=	Các toán tử so sánh bằng
10	= %= /= //=- -= += *= **=	Các toán tử gán

11	in not in	Các toán tử Membership
12	not or and	Các toán tử logic

2.3 Các hàm toán học

Các hàm toán học (trong thư viện math) cho phép thực hiện một số hàm lượng giác và giải tích trên số thập phân:

- Các hàm lượng giác : sin, cos, tan, asin, acos, atan
- Hàm căn bậc 2 : sqrt
- Hàm phần nguyên : floor, hàm phần nguyên trên : ceil
- Hàm logarithm : log10 (cơ số 10), log (cơ số e)

Để sử dụng các hàm toán học, cần khai báo thư viện math : import math

Ví dụ :

```
import math
print(math.sin(math.pi/2))
print(math.sqrt(4))
```

Function	Description
abs()	Returns absolute value of a number
divmod()	Returns quotient and remainder of integer division
max()	Returns the largest of the given arguments or items in an iterable
min()	Returns the smallest of the given arguments or items in an iterable
pow()	Raises a number to a power
round()	Rounds a floating-point value
sum()	Sums the items of an iterable

2.3.1 Type Conversion

Function	Description
ascii()	Returns a string containing a printable representation of an object
bin()	Converts an integer to a binary string
bool()	Converts an argument to a Boolean value
chr()	Returns string representation of character given by integer argument

Function	Description
<code>complex()</code>	Returns a complex number constructed from arguments
<code>float()</code>	Returns a floating-point object constructed from a number or string
<code>hex()</code>	Converts an integer to a hexadecimal string
<code>int()</code>	Returns an integer object constructed from a number or string
<code>oct()</code>	Converts an integer to an octal string
<code>ord()</code>	Returns integer representation of a character
<code>repr()</code>	Returns a string containing a printable representation of an object
<code>str()</code>	Returns a string version of an object
<code>type()</code>	Returns the type of an object or creates a new type object

2.3.2 Iterables and Iterators

Function	Description
<code>all()</code>	Returns True if all elements of an iterable are true
<code>any()</code>	Returns True if any elements of an iterable are true
<code>enumerate()</code>	Returns a list of tuples containing indices and values from an iterable
<code>filter()</code>	Filters elements from an iterable
<code>iter()</code>	Returns an iterator object
<code>len()</code>	Returns the length of an object
<code>map()</code>	Applies a function to every item of an iterable
<code>next()</code>	Retrieves the next item from an iterator

Function	Description
<code>range()</code>	Generates a range of integer values
<code>reversed()</code>	Returns a reverse iterator
<code>slice()</code>	Returns a <code>slice</code> object
<code>sorted()</code>	Returns a sorted list from an iterable
<code>zip()</code>	Creates an iterator that aggregates elements from iterables

2.3.3 Composite Data Type

Function	Description
<code>bytearray()</code>	Creates and returns an object of the <code>bytearray</code> class
<code>bytes()</code>	Creates and returns a <code>bytes</code> object (similar to <code>bytearray</code> , but immutable)
<code>dict()</code>	Creates a <code>dict</code> object
<code>frozenset()</code>	Creates a <code>frozenset</code> object
<code>list()</code>	Creates a <code>list</code> object
<code>object()</code>	Creates a new featureless object
<code>set()</code>	Creates a <code>set</code> object
<code>tuple()</code>	Creates a <code>tuple</code> object

Bài 3: Cấu trúc Điều khiển, Lặp

3.1 Lệnh rẽ nhánh

Khi giải các bài toán trên giấy, thứ tự thực hiện các phép tính là từ trên xuống dưới. Tuy nhiên, với các bài toán trên máy tính, có hiện tượng "rẽ nhánh", tức là tùy vào kết quả của phép tính trước mà một số phép tính sau có thể được thực hiện hay không.

Để thực hiện việc này, Python sử dụng lệnh **if** theo các cấu trúc:

3.1.1 Câu lệnh If

```
if <điều kiện> :  
    <Lệnh>
```

Cấu trúc trên có nghĩa: Nếu điều kiện đúng thì thực hiện lệnh, không thì không thực hiện lệnh. Ví dụ:

```
so_nguyen = 8  
if (so_nguyen % 2 == 0):  
    print (so_nguyen, "là số chẵn")
```

3.1.2 Câu lệnh If .. Else

```
if <điều kiện> :  
    <Lệnh 1>  
else:  
    <Lệnh 2>
```

Cấu trúc trên có nghĩa : Nếu điều kiện đúng thì thực hiện lệnh 1, không thì thực hiện lệnh 2. Ví dụ:

```
so_nguyen = 15  
if (so_nguyen % 2 == 0):  
    print (so_nguyen, "là số chẵn")  
else:  
    print(so_nguyen, "là số lẻ")
```

3.1.3 Câu lệnh If .. ElIf .. Else

```
if <điều kiện 1> :  
    <Lệnh 1>  
elif <điều kiện 2> :  
    <Lệnh 2>  
elif <điều kiện 3> :
```

<Lệnh 3>

else:

<Lệnh 4>

Cấu trúc trên có nghĩa : Nếu điều kiện 1 đúng thì thực hiện lệnh 1, không thì kiểm tra điều kiện 2, nếu điều kiện 2 đúng thì thực hiện lệnh 2, nếu không tiếp tục kiểm tra các điều kiện tiếp theo.

Ví dụ:

```
time = 10

if (time < 10):
    print ("Good morning")

elif (time < 18):
    print("Good afternoon")

else:
    print("Good evening")
```

3.2 Lệnh lặp while

Vòng lặp while thường dùng khi bạn chưa biết trước số lượng vòng lặp cần dùng.

while condition:

Khởi lệnh này sẽ được thực thi nếu condition còn đúng

Ví dụ về chương trình hiển thị ra màn hình các số từ 1 tới 5 sử dụng vòng lặp while:

```
i = 1
while i <= 5:
    print(i)
    i += 1
```

Kết quả chạy:

```
1
2
3
4
5
```

Có thể thấy cách sử dụng vòng lặp `while` rất đơn giản, bạn có thể sử dụng vòng lặp `while` để tính tổng các số từ `1` tới `n` giống như sau:

```
n = int(input())
i = 1
answer = 0
while i <= n:
    answer += i
    i += 1
print(answer)
```

3.3 Lệnh lặp for

Vòng lặp for thường được dùng khi bạn đã biết trước số lượng vòng lặp cần thực hiện.

1 for biến lặp in tập hợp:
2 các câu lệnh

Biến lặp có thể là bất cứ biến nào. Bạn chỉ cần đưa vào một cái tên là Python sẽ tự ngầm hiểu kiểu dữ liệu. Còn tập hợp có thể là bất kỳ kiểu tập hợp nào mà chúng ta đã học trong bài trước, hoặc cũng có thể là một string.

```
for letter in 'Python':
    print('Current Letter :', letter)

fruits = ['banana', 'apple', 'mango']
for fruit in fruits:
    print ('Current fruit :', fruit)
```

Kết quả chạy:

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : h
Current Letter : o
Current Letter : n
Current fruit : banana
Current fruit : apple
Current fruit : mango
```

Bạn cũng có thể dùng vòng lặp for theo chỉ số.

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

Kết quả chạy:

```
Current fruit : banana
Current fruit : apple
Current fruit : mango
```

Cũng giống như vòng lặp `while`, bạn cũng có thể dùng `else` cho vòng lặp `for`.

```
1   for num in range(10,20):
2       for i in range(2,num):
3           if num%i == 0:
4               j=num/i
5               print('%d equals %d * %d' % (num,i,j))
6               break
7       else:
8           print (num, 'is a prime number')
```

Ví dụ trên tìm ước số của các số khác 1 và chính nó của từ 10 đến 20. Nếu không tìm thấy thì thông báo số đó là số nguyên tố. Và cũng giống như các câu lệnh điều kiện, bạn cũng có thể lồng các vòng lặp vào nhau.

```
1   10 equals 2 * 5
2   11 is a prime number
3   12 equals 2 * 6
4   13 is a prime number
5   14 equals 2 * 7
6   15 equals 3 * 5
7   16 equals 2 * 8
8   17 is a prime number
9   18 equals 2 * 9
10  19 is a prime number
```

Cách biểu diễn tập giá trị lặp trong vòng lặp for

- Liệt kê các thành phần của tập giá trị. Ví dụ [1, 2, 3, 4, 5]. Đây thực chất là dữ liệu kiểu List (sẽ được mô tả ở phần sau)
- Sử dụng cấu trúc :
`range(<end>)`

Biểu thức này thể hiện tập các số nguyên từ 0 đến trước giá trị `<end>` (`<end-1>`)

Ví dụ:

range(5) → 0, 1, 2, 3, 4

- Sử dụng cấu trúc :

range(start, end)

Biểu thức này thể hiện tập các số nguyên từ <start> đến <end-1>

Ví dụ:

range(1, 5) → 1, 2, 3, 4

- Sử dụng cấu trúc :

range(start, end, increment)

Biểu thức này thể hiện tập các số nguyên từ <start> đến <end-1> và tăng đều với khoảng cách increment.

Ví dụ:

range(0, 10, 2) → 0, 2, 4, 6, 8

3.4 Sử dụng break, continue, pass statement

3.4.1 Câu lệnh break

Khi bạn muốn dừng vòng lặp giữa chừng thì dùng câu lệnh break.

```
1  for letter in 'Python':  
2      if letter == 'h':  
3          break  
4      print ('Current Letter :', letter)  
5  
6  var = 10  
7  while var > 0:  
8      print ('Current variable value :', var)  
9      var = var -1  
10     if var == 5:  
11         break
```

Trong đoạn code trên, ở vòng lặp for chúng ta sẽ dừng vòng lặp nếu tìm thấy kí tự 'h' trong chuỗi "Python", ở vòng lặp while thì sẽ dừng vòng lặp khi biến var giảm bằng 5.

```
1  Current Letter : P  
2  Current Letter : y  
3  Current Letter : t  
4  Current variable value : 10  
5  Current variable value : 9  
6  Current variable value : 8  
7  Current variable value : 7  
8  Current variable value : 6
```

3.4.2 Câu lệnh continue

Câu lệnh `continue` có tác dụng nhảy sang lần lặp kế tiếp. Các câu lệnh phía sau `continue` sẽ không được thực thi.

```
1  for letter in 'Python':  
2      if letter == 'h':  
3          continue  
4      print ('Current Letter :', letter)  
5  
6  var = 10  
7  while var > 0:  
8      var = var -1  
9      if var == 5:  
10         continue  
11     print ('Current variable value :', var)
```

Ở ví dụ trên, ở vòng lặp `for`, chúng ta cho lặp tiếp khi tìm thấy kí tự 'h' mà không in kí tự đó ra màn hình. Còn ở vòng lặp `while` thì chúng ta cho lặp tiếp khi biến `var` bằng 5.

```
1  Current Letter : P  
2  Current Letter : y  
3  Current Letter : t  
4  Current Letter : o  
5  Current Letter : n  
6  Current variable value : 9  
7  Current variable value : 8  
8  Current variable value : 7  
9  Current variable value : 6  
10 Current variable value : 4  
11 Current variable value : 3  
12 Current variable value : 2  
13 Current variable value : 1  
14 Current variable value : 0
```

3.4.3 Câu lệnh pass

Trong Python, `pass` là một lệnh trống, không làm gì cả, nó chỉ giữ chỗ cho các hàm, vòng lặp mà bạn đã thêm vào, nhưng hiện tại chưa dùng đến mà để "dành cho con cháu" mở rộng trong tương lai.

```
def function(args):  
    pass
```

Lệnh pass khác với comment ở chỗ lệnh comment thì trình biên dịch bỏ qua, còn lệnh pass xem như vẫn thực hiện nhưng không làm gì cả.

Ví dụ:

```
# pass chỉ giữ chỗ cho for
# hàm sẽ được thêm vào sau.
sequence = ['n', 'h', 'a', 't', 'n', 'g', 'h', 'e']
for val in sequence:
    pass
```

Bài 4: String, Date & Time

4.1 Kiểu chuỗi (string)

4.1.1 Chuỗi và truy xuất chuỗi

Một chuỗi có thể khai báo bằng dấu nháy đôi " hoặc đơn ' . Ví dụ các chuỗi sau:

```
str1 = "Hello"
```

```
str2 = 'world'
```

Có thể truy xuất từng ký tự trong một chuỗi theo hình thức index, ví dụ: str1[0] , str1[1] ...

Có thể sử dụng 3 dấu nháy (đôi hoặc đơn) để khai báo chuỗi trên nhiều dòng. Ví dụ:

```
paragraph = """This is line 1
```

```
This is line 2
```

```
This is line 3"""
```

Chuỗi là được xem là một mảng các byte đại diện cho các ký tự unicode.

Để truy cập các phần tử của chuỗi, ta dùng dấu ngoặc vuông []. Ký tự đầu tiên có chỉ số là 0.

```
str = "HELLO"
```

H	E	L	L	O
0	1	2	3	4

```
str[0] = 'H'
```

```
str[1] = 'E'
```

```
str[2] = 'L'
```

```
str[3] = 'L'
```

```
str[4] = 'O'
```

Ta có thể lấy chuỗi con bằng cách chỉ định chỉ mục bắt đầu và chỉ mục kết thúc, được phân tách bằng dấu hai chấm, để trả về một phần của chuỗi.

str = "HELLO"

H	E	L	L	O
0	1	2	3	4

str[0] = 'H' str[:] = 'HELLO'

str[1] = 'E' str[0:] = 'HELLO'

str[2] = 'L' str[:5] = 'HELLO'

str[3] = 'L' str[:3] = 'HEL'

str[4] = 'O' str[0:2] = 'HE'

str[1:4] = 'ELL'

Ngoài ra ta có thể truy cập chuỗi bằng chỉ mục âm để lấy ra chuỗi con bắt đầu từ cuối chuỗi: Ví dụ:

```
str1 = "Hello World!"
```

```
print(str1[-5:-2])
```

4.1.2 Các hàm xử lý chuỗi

4.1.2.1 Một số hàm cơ bản

- Hàm **str** : chuyển từ các kiểu dữ liệu khác sang dữ liệu string

Ví dụ:

```
>>> str(100)  
'100'
```

- Phép cộng 2 giá trị string :

Ví dụ:

```
>>> 'Chào ' + 'bạn'  
'Chào bạn'
```

- Hàm **lower** : chuyển giá trị string sang dạng chữ thường

Ví dụ:

```
>>> 'Chào bạn'.lower()
'chào bạn'
```

- Hàm **upper** : chuyển giá trị string sang dạng chữ hoa

Ví dụ:

```
>>> 'Chào bạn'.upper()
'CHÀO BẠN'
```

- Hàm **replace** : Thay giá trị một đoạn con trong một string bằng một giá trị khác

Ví dụ:

```
>>> 'Tôi sống ở Hà Nội'.replace('Hà Nội', 'Huế')
'Tôi sống ở Huế'
```

- Hàm **split** : Tách một giá trị string thành nhiều đoạn nhỏ

Ví dụ:

```
>>> 'Hà Nội'.split()
['Hà', 'Nội']
>>> 'Hà Nội, Việt Nam'.split(',')
['Hà Nội', ' Việt Nam']
```

- Hàm **strip** : Bỏ đi các kí tự trắng (khoảng trắng) ở đầu và cuối string:

Ví dụ:

```
>>> '    Hà Nội      '.strip()
'Hà Nội'
```

Ngoài ra có thể dùng hàm lstrip(), rstrip() để loại bỏ khoảng trắng thừa bên trái hoặc bên phải chuỗi.

- Hàm **len** : trả về số lượng ký tự (độ dài) của một chuỗi

Ví dụ:

```
>>> len('Chào bạn')
8
```

4.1.2.2 Nối chuỗi

Dùng toán tử + để nối chuỗi và toán tử * để lặp chuỗi.

Ví dụ: Nối 2 chuỗi ký tự

```
message = "Xin chào" + " " + "Nhất Nghệ"
```

```
print(message)
```

Kết quả:

Xin chào Nhất Nghệ

Ví dụ: Lặp chuỗi ký tự giống nhau

```
print("Happy " * 3)
```

Kết quả in ra:

Happy Happy Happy

4.1.2.3 Các ký tự thoát

Đây là các ký tự đặc biệt dùng để điều khiển string.

```
1   print ("Incompatible, it don't matter though\n'cos someone's  
2     bound to hear my cry")  
3   print ("Speak out if you do\nYou're not easy to find")
```

Kí tự \n mang ý nghĩa xuống dòng, kí tự nào đứng đằng sau kí tự này sẽ tự động xuống dòng.

```
1   Incompatible, it don't matter though  
2     'cos someone's bound to hear my cry  
3   Speak out if you do  
4   You're not easy to find
```

Tiếp theo là kí tự backspace \b.

```
1   print ("Python\b\b\booo") # prints Pytoo
```

Kí tự này đưa con trỏ về trước 1 kí tự. Trong ví dụ trên, chúng ta có 3 kí tự \b nên con trỏ được đưa lùi về 3 kí tự nên 3 kí tự hon được thay thế bằng 3 kí tự ooo.

```
1   print ("Towering\tinferno") # prints Towering      inferno
```

Kí tự \t có tác dụng in ra một dấu tab.

Nếu chúng ta đưa r vào trước dấu nháy. Các ký tự thoát trong string sẽ không được sử dụng và chúng sẽ được in ra như các kí tự bình thường.

```

1     print (r"Another world\n")
1     Another world\n

```

Python còn nhiều kiểu kí tự thoát khác nữa, bạn có thể tham khảo ở bảng dưới đây.

Ký tự thoát	Miêu tả
\a	Bell hoặc alert
\b	Backspace
\cx	Control-x
\C-x	Control-x
\e	Escape
\f	Formfeed
\M-\C-x	Meta-Control-x
\n	Newline
\nnn	Notation trong hệ cơ số 8, ở đây n là trong dãy từ 0 tới 7
\r	Carriage return
\s	Space
\t	Tab
\v	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

4.1.2.4 So sánh string

Chúng ta có thể so sánh 2 string bằng các toán tử ==, !=... và kết quả trả về là một giá trị boolean.

```

1     print ("12" == "12")
2     print ("17" == "9")
3     print ("aa" == "ab")
4
5     print ("abc" != "bce")
6     print ("efg" != "efg")

```

Trong ví dụ trên, chúng ta so sánh 2 string.

```

1     print ("12" == "12")

```

Hai string trên bằng nhau, kết quả là True.

```
1     print ("aa" == "ab")
```

Hai string trên khác nhau, hai kí tự đầu tiên bằng nhau nhưng kí tự thứ hai khác nhau, trong đó kí tự a lớn hơn kí tự b. Kết quả trả về là False.

```
1     print ("abc" != "bce")
```

Hai string này khác nhau hoàn toàn. Kết quả trả về là True.

```
1     True  
2     False  
3     False  
4     True  
5     False
```

4.1.2.5 Truy xuất phần tử trong string

Một giá trị string được tạo thành từ một dãy các kí tự, để lấy kí tự ở vị trí index (bắt đầu từ 0) của một giá trị string, chúng ta dùng cú pháp:

c = text[index]

Các cách để lấy ra substring trong Python:

Cú pháp:

text[start:end]

Biểu thức này trả về đoạn kí tự từ vị trí start đến end-1 của string gốc.

Ví dụ:

```
>>> text = 'Chào bạn'; print(text[0:4])
```

Chào

Cú pháp:

text[start:]

Biểu thức này trả về đoạn kí tự từ vị trí start đến hết string gốc.

Ví dụ:

```
>>> text = 'Chào bạn'; print(text[5:])
```

bạn

Cú pháp:

text[:end]

Biểu thức này trả về đoạn kí tự từ đầu đến vị trí end của string gốc.

Ví dụ:

```
>>> text = 'Chào bạn'; print(text[:4])
```

Chào

Trong các cú pháp trên, nếu các giá trị start, end là âm thì có nghĩa vị trí được tính từ cuối string trở về (-1 tương ứng với vị trí cuối cùng của string)

Ví dụ:

```
>>> text = 'Chào bạn'; print(text[-3:])
    bạn
```

Cũng như tuple, list... chúng ta có thể truy xuất các phần tử trong string.

```
1     s = "Eagle"
2
3     print (s[0])
4     print (s[4])
5     print (s[-1])
6     print (s[-2])
7
8     print ("*****")
9
10    print (s[0:4])
11    print (s[1:3])
12    print (s[:])
```

Ở trên chúng ta truy xuất thông qua chỉ số.

```
1     print (s[0])
2     print (s[4])
```

Chỉ số ở đây được đánh số từ 0, nếu string có 5 kí tự thì chúng được đánh số từ 0..4.

```
1     print (s[-1])
2     print (s[-2])
```

Nếu chúng ta đưa vào chỉ số là số âm, chúng ta lấy được các phần tử từ cuối string.

```
1     print (s[0:4])
```

Chúng ta có thể đưa các phạm vi lấy từ đâu đến đâu vào để lấy một tập các phần tử chứ không chỉ lấy được các phần tử riêng biệt.

```
1     print (s[:])
```

Nếu chúng ta đưa vào chỉ số như phía trên thì sẽ lấy toàn bộ string.

```
1     E
2     e
3     e
4     l
5     *****
6     Eagl
7     ag
8     Eagle
```

Chúng ta có thể dùng vòng lặp để duyệt qua string.

```
1     s = "PyNhatNghe"
2
3     for i in s:
4         print (i,)
```

Đoạn code trên in toàn bộ các phần tử trong string ra màn hình.

```
1     PyNhatNghe
```

4.1.3 Tìm chuỗi con

Các phương thức `find()`, `rfind()`, `index()` và `rindex()` là các phương thức dùng để tìm string con. Kết quả trả về là vị trí của kí tự đầu tiên của string con được tìm thấy trong string cha. Phương thức `find()` và `index()` tìm từ đầu string, phương thức `rfind()` và `rindex()` tìm từ cuối string.

Sự khác nhau của 2 phương thức `find()` và `index()` là nếu không tìm thấy string con thì phương thức đầu tiên trả về -1, còn phương thức thứ 2 trả về một `ValueError exception`. Exception là một khái niệm quan trọng và chúng ta sẽ tìm hiểu ở các bài cuối.

```
1     find(str, beg=0, end=len(string))
2     rfind(str, beg=0, end=len(string))
3     index(str, beg=0, end=len(string))
4     rindex(str, beg=0, end=len(string))
```

Trong đoạn code trên, các phương thức này nhận vào 3 tham số, tham số đầu tiên `str` là string con cần tìm, tham số thứ 2 và thứ 3 là vị trí bắt đầu tìm và vị trí kết thúc để tìm trong string cha. Chúng ta có thể không đưa 2 tham số này vào và các phương thức trên sẽ mặc định tìm từ vị trí đầu tiên là 0 và vị trí cuối cùng là độ dài của string cha.

```
1     a = "I saw a wolf in the forest. A lone wolf."
2
3     print (a.find("wolf"))
4     print (a.find("wolf", 10, 20))
5     print (a.find("wolf", 15))
6
7     print (a.rfind("wolf"))
```

Ví dụ trên, chúng ta có một string và chúng ta thử tìm string con từ các khoảng vị trí khác nhau.

```
1     print (a.find("wolf"))
```

Dòng trên tìm vị trí xuất hiện đầu tiên của string “wolf”.

```
1     print (a.find("wolf", 10, 20))
```

Dòng trên tìm vị trí xuất hiện đầu tiên của string “wolf” từ vị trí 10 đến 20 trong string gốc, kết quả trả về -1 vì không tìm thấy.

```
1     print (a.find("wolf", 15))
```

Dòng trên chúng ta cũng tìm string “wolf” từ vị trí 15 đến cuối string và tìm ra vị trí thứ hai mà string này xuất hiện trong string cha.

```
1     print (a.rfind("wolf"))
```

Phương thức `rfind()` tìm string gốc từ cuối string cha, do đó cũng tìm ra vị trí thứ hai của string này trong string cha.

```
1     8
2     -1
3     35
4     35
```

Tiếp theo chúng ta sẽ tìm hiểu về phương thức `index()`.

```
1     a = "I saw a wolf in the forest. A lone wolf."
2
3     print (a.index("wolf"))
4     print (a.rindex("wolf"))
5
6     try:
7         print (a.rindex("fox"))
8     except ValueError as e:
9         print ("Could not find substring")
```

Như đã nói ở trên, điểm khác nhau của phương thức này với phương thức `find()` là cách chúng gửi giá trị trả về khi không tìm thấy.

```
1     print (a.index("wolf"))
2     print (a.rindex("wolf"))
```

Hai dòng trên tìm vị trí bắt đầu của string “wolf” từ vị trí đầu và cuối string cha.

```
1     try:
2         print (a.rindex("fox"))
3     except ValueError as e:
4         print ("Could not find substring")
```

Ở đây chúng ta thử tìm string “fox” và tất nhiên là không tìm thấy, phương thức này trả về một `ValueError exception`.

```
1     8
2     35
3 Could not find substring
```

4.1.4 Toán tử trên string

Ví dụ dưới đây sử dụng toán tử nhân * và toán tử nối chuỗi.

```
1 print ("eagle " * 5)
2 print ("eagle " "falcon")
3 print ("eagle " + "and " + "falcon")
```

Toán tử * sẽ lặp lại string n lần. Trong ví dụ trên là 5 lần. Hai string được đặt sát nhau sẽ tự động ngầm nối vào nhau. Bạn cũng có thể dùng dấu + để nối chuỗi một cách rõ ràng.

```
1 eagle eagle eagle eagle eagle
2 eagle falcon
3 eagle and falcon
```

Phương thức `len()` có tác dụng tính số chữ cái trong string.

```
1 var = 'eagle'
2
3 print (var, "has", len(var), "characters")
```

Ví dụ trên tính số lượng các kí tự có trong string.

```
1 eagle has 5 characters
```

Trong một số ngôn ngữ khác như Java, C#, kiểu string và kiểu số có thể được ngầm nối thành một string, chẳng hạn trong Java, bạn có thể gõ `System.out.println("Number: " + 12)` để in chuỗi “Number: 12” ra màn hình. Python không cho phép điều này, bạn phải chuyển đổi kiểu dữ liệu một cách rõ ràng.

```
1 print (int("12") + 12)
2 print ("There are " + str(22) + " oranges.")
3 print (float('22.33') + 22.55)
```

Chúng ta có thể dùng phương thức `int()` để chuyển một string thành một số, `float()` để chuyển một string thành một số thực, hoặc hàm `str()` để chuyển một số thành một string,

4.1.5 Thay thế chuỗi

Phương thức `replace()` sẽ thay thế một chuỗi con trong chuỗi cha thành một chuỗi mới.

```
1     replace(old, new [, max])
```

Mặc định thì phương thức này sẽ thay thế tất cả các chuỗi con mà nó tìm thấy được. Nhưng bạn cũng có thể giới hạn phạm vi tìm kiếm.

```
1     a = "I saw a wolf in the forest. A lonely wolf."
2
3     b = a.replace("wolf", "fox")
4     print (b)
5
6     c = a.replace("wolf", "fox", 1)
7     print (c)
```

Ở ví dụ trên chúng ta thay thế string “wolf” thành “fox”.

```
1     b = a.replace("wolf", "fox")
```

Dòng trên sẽ thay thế tất cả các string “wolf” thành “fox”.

```
1     c = a.replace("wolf", "fox", 1)
```

Dòng này thì chỉ thay thế string “wolf” đầu tiên mà nó tìm thấy.

```
1     I saw a fox in the forest. A lonely fox.
2     I saw a fox in the forest. A lonely wolf.
```

4.1.6 Tách, nối chuỗi

Chúng ta có thể tách một string bằng phương thức `split()` hoặc `rsplit()`. Kết quả trả về là một đối tượng kiểu `list` có các phần tử là các kí tự được tách ra từ string gốc, string gốc được tách dựa trên các kí tự phân cách mà chúng ta cung cấp. Ngoài ra 2 phương thức trên còn có 1 tham số thứ 2 là số lượng lần tách tối đa.

```
1     nums = "1,5,6,8,2,3,1,9"
2
3     k = nums.split(",")
4     print (k)
5
6     l = nums.split(", ", 5)
7     print (l)
8
9     m = nums.rsplit(", ", 3)
10    print (m)
```

Trong ví dụ trên, chúng ta sử dụng kí tự phân cách là dấu phẩy “,” để tách string thành các string con.

```
1     k = nums.split(",")
```

String gốc của chúng ta có 8 kí tự, nên list trả về sẽ có 8 phần tử.

```
1     l = nums.split(", ", 5)
```

Dòng code trên quy định số lần tách tối đa là 5 lần, do đó list trả về của chúng ta sẽ có 6 phần tử.

```
1     m = nums.rsplit(", ", 3)
```

Ở đây chúng ta tách string gốc thành 4 phần tử nhưng quá trình tách bắt đầu từ cuối string.

```
1     ['1', '5', '6', '8', '2', '3', '1', '9']
2     ['1', '5', '6', '8', '2', '3,1,9']
3     ['1,5,6,8,2', '3', '1', '9']
```

Chúng ta có thể nối các string bằng hàm `join()`. Các string được nối lại cũng có thể có các kí tự phân tách.

```
1     nums = "1,5,6,8,2,3,1,9"
2
3     n = nums.split(",")
4     print (n)
5
6     m = ':'.join(n)
7     print (m)
```

Trong ví dụ trên chúng ta tách string `nums` ra dựa trên dấu phẩy “,”, sau đó nối các phần tử mới trong list `n` thành một string và cách nhau bởi dấu hai chấm “:”.

```
1     m = ':'.join(n)
```

Phương thức `join()` sẽ tạo ra một string mới từ một list các string con, cách nhau bởi dấu “:”.

```
1     ['1', '5', '6', '8', '2', '3', '1', '9']
2     1:5:6:8:2:3:1:9
```

Ngoài phương thức `split()` dùng để tách chuỗi, chúng ta có một phương thức khác có chức năng tương tự là `partition()`, phương thức này chỉ cắt chuỗi 1 lần khi nó gặp kí tự phân cách, do đó nó cũng giống như là khi dùng `split(str, 1)` vậy. Thế nên kết quả trả về của phương thức này luôn là một list chứa 3 phần tử, phần tử đầu tiên là đoạn string nằm phía trước kí tự phân cách, phần tử thứ 2 là chính bản thân kí tự phân cách, phần tử thứ 3 là đoạn string nằm phía sau kí tự phân cách.

```
1     s = "1 + 2 + 3 = 6"
2
3     a = s.partition(">")
4
5     print(a)
```

Đoạn code trên ví dụ về phương thức `partition()`.

```
1     a = s.partition(">")
1     ('1 + 2 + 3 ', '=', ' 6')
```

4.1.7 Chữ HOA và chữ thường

Python có các phương thức để làm việc với từng trường hợp.

```
1     a = "PyNhatNghe"
2     print(a.upper())
3     print(a.lower())
4     print(a.swapcase())
5     print(a.title())
```

Ví dụ trên sử dụng bốn phương thức.

```
1     print(a.upper())
```

Phương thức `upper()` sẽ chuyển toàn bộ các kí tự trong string thành viết HOA.

```
1     print(a.lower())
```

Phương thức `lower()` sẽ chuyển toàn bộ các kí tự trong string thành viết thường.

```
1     print(a.swapcase())
```

Phương thức `swapcase()` đảo ngược kiểu HOA thành kiểu thường và ngược lại.

```
1     print(a.title())
```

Phương thức `title()` sẽ viết HOA chữ cái đầu tiên, các chữ cái còn lại sẽ chuyển thành kiểu thường.

```
1     PyNhatNghe
2     pynhatnghe
3     pYnHATnGHE
4     Pynhatnghe
```

4.1.8 Một số thao tác khác

```
1     sentence = "There are 22 apples"
2
3     alphas = 0
4     digits = 0
5     spaces = 0
6
7     for i in sentence:
8         if i.isalpha():
9             alphas += 1
10        if i.isdigit():
11            digits += 1
12        if iisspace():
13            spaces += 1
14
15    print ("There are", len(sentence), "characters")
16    print ("There are", alphas, "alphabetic characters")
17    print ("There are", digits, "digits")
18    print ("There are", spaces, "spaces")
```

Ở đây chúng ta sử dụng các phương thức `isalpha()`, `isdigit()`, `isspace()` để xác định xem kí tự nào là chữ cái, chữ số hay dấu cách.

```
1     There are 19 characters
2     There are 14 alphabetic characters
3     There are 2 digits
4     There are 3 spaces
```

Ví dụ dưới đây chúng ta sẽ căn chỉnh lề khi in string ra màn hình.

```
1     print ("Ajax Amsterdam" " - " + "Inter Milano " "2:3")
2     print ("Real Madridi" " - " "AC Milano " "3:3")
3     print ("Dortmund" " - " "Sparta Praha " "2:1")
```

Bạn đã biết là chúng ta có thể nối chuỗi bằng cách đặt hai chuỗi sát nhau hoặc dùng toán tử +.

```
1     Ajax Amsterdam - Inter Milano 2:3
2     Real Madridi - AC Milano 3:3
3     Dortmund - Sparta Praha 2:1
```

Chúng ta sẽ căn chỉnh lề cho đẹp hơn.

```

1     teams = {
2         0: ("Ajax Amsterdam", "Inter Milano"),
3         1: ("Real Madrid", "AC Milano"),
4         2: ("Dortmund", "Sparta Praha")
5     }
6
7     results = ("2:3", "3:3", "2:1")
8
9
10    for i in teams:
11        print (teams[i][0].ljust(16) + "-".ljust(5) + \
12              teams[i][1].ljust(16) + results[i].ljust(3))

```

Ở ví dụ trên chúng ta có một kiểu từ điển và một tuple. Phương thức `ljust()` sẽ đưa các string về lề trái, phương thức `rjust()` để đưa các string về lề phải. Chúng ta có thể quy định số lượng kí tự của mỗi string, nếu string không đủ số lượng thì các dấu cách sẽ tự động được thêm vào.

1	Ajax Amsterdam	-	Inter Milano	2 : 3
2	Real Madrid	-	AC Milano	3 : 3
3	Dortmund	-	Sparta Praha	2 : 1

4.1.9 Định dạng chuỗi

Định dạng chuỗi tức là bạn đưa các tham số/biến vào bên trong string và tùy theo các tham số này mang giá trị gì mà string sẽ hiển thị ra giá trị đó. Định dạng kiểu này rất được hay dùng không chỉ trong Python mà các ngôn ngữ lập trình khác. Các tham số định dạng sử dụng toán tử %.

```
1     print ('There are %d oranges in the basket' % 32)
```

Ví dụ trên chúng ta sử dụng định dạng `%d`, đây là định dạng cho số nguyên. Bên trong chuỗi, chúng ta ghi `%d` tại vị trí mà chúng ta muốn hiển thị tham số, sau khi kết thúc chuỗi, chúng ta thêm `%` vào sau chuỗi và các giá trị và chúng ta muốn đưa vào.

```
1     There are 32 oranges in the basket
```

Nếu chúng ta dùng nhiều tham số bên trong chuỗi thì ở sau chuỗi các giá trị đưa vào phải nằm trong cặp dấu (), tức là giống như một tuple vậy.

```
1     print ('There are %d oranges and %d apples in the basket' % (12, 23))
```

```
1     There are 12 oranges and 23 apples in the basket
```

Ví dụ dưới đây dùng tham số được định dạng kiểu số thực và kiểu string.

```
1     print ('Height: %f %s' % (172.3, 'cm'))
```

`%f` dùng cho số thực, còn `%s` là một string.

```
1 Height: 172.300000 cm
```

Mặc định định dạng số thực có 6 chữ số thập phân sau. Chúng ta có thể quy định số lượng chữ số này.

```
1 print ('Height: %.1f %s' % (172.3, 'cm'))
```

Bằng cách thêm dấu . và số phần thập phân vào giữa % và f. Python sẽ tự động loại bỏ các chữ số 0 thừa cho chúng ta.

```
1 Height: 172.3 cm
```

Dưới đây là một số kiểu định dạng khác.

```
1 # hexadecimal
2 print ("%x" % 300)
3 print ("%#x" % 300)
4
5 # octal
6 print ("%o" % 300)
7
8 # scientific
9 print ("%e" % 300000)
```

%x sẽ chuyển một số từ hệ bất kì sang hệ thập lục phân, nếu thêm kí tự # vào giữa %x thì được thêm cặp chữ 0x. Định dạng %o là hiển thị số ở hệ bát phân (octal). %e thì hiển thị theo định dạng số mũ.

```
1 12c
2 0x12c
3 454
4 3.000000e+05
```

Ví dụ dưới đây chúng ta sẽ in ra các chữ số trên 3 cột.

```
1 for x in range(1,11):
2     print ('%d %d %d' % (x, x*x, x*x*x))
```

```
1   1 1 1
2   2 4 8
3   3 9 27
4   4 16 64
5   5 25 125
6   6 36 216
7   7 49 343
8   8 64 512
9   9 81 729
10  10 100 1000
```

Như bạn thấy, các cột không được đều nhau nhìn không đẹp. Chúng ta sẽ thêm các đặc tả độ rộng vào. Bạn xem ví dụ để biết cách làm.

```
1   for x in range(1,11):
2       print ('%2d %03d %4d' % (x, x*x, x*x*x))
```

Nếu như chúng ta ghi là %2d thì sẽ có thêm một dấu cách vào cho đủ 2 kí tự, nếu ghi là 03 thì các số 0 sẽ được thêm vào cho đủ 3 kí tự...

```
1   1 001 1
2   2 004     8
3   3 009    27
4   4 016    64
5   5 025   125
6   6 036   216
7   7 049   343
8   8 064   512
9   9 081   729
10  10 100 1000
```

Tuy nhiên việc cộng nhiều string sẽ làm chương trình khó theo dõi, do đó có thể dùng hàm format của kiểu dữ liệu string để ghép nhiều string (và các kiểu dữ liệu khác) với nhau. Cách thực hiện như sau:

```
x = 1
y = 2
z = 3
st = 'Tổng của {} và {} là {}'.format(x, y, z)
print(st)
```

Mỗi cặp {} trong template của string sẽ tương ứng với một biến/biểu thức nằm bên trong hàm format.

Ngoài ra, có thể viết tên biến/biểu thức ngay bên trong cặp {} nếu sử dụng tiền tố f ở trước template của string, cách thực hiện như sau:

```
x = 1
y = 2
z = 3
st = f'Tổng của {x} và {y} là {z}'
print(st)
```

4.2 Kiểu Date, Time

4.2.1 Kiểu datetime

Dữ liệu kiểu Date & Time được cung cấp bởi thư viện datetime của hệ thống:

```
from datetime import date, datetime
```

Kiểu dữ liệu date chứa thông tin về ngày, tháng, năm. Kiểu dữ liệu datetime chứa thông tin về ngày, tháng, năm, giờ, phút, giây, mili-giây.

Để tạo mới một đối tượng kiểu date/datetime:

```
from datetime import date, datetime

d = date(2020,1,1)                      # 01/01/2020
dt = datetime(2020,1,1,23,59,59)          # 23:59:59 01/01/2020
```

Lấy ngày giờ hiện tại của hệ thống:

```
from datetime import date, datetime

d = date.today()
dt = datetime.now()
```

Truy nhập đến các thành phần của date/datetime :

```
from datetime import date, datetime

dt = datetime(2019,1,1,23,59,59)
print(dt.year, dt.month, dt.day, dt.hour, dt.minute,
      dt.second)

d = date(2019,1,1)
print(d.year, d.month, d.day)
```

Chuyển đổi từ datetime sang String :

```
from datetime import datetime

now = datetime.now()

print(now.strftime('%d-%m-%Y'))
print(now.strftime('%d/%m/%Y %H:%M:%S'))
```

Chuyển đổi từ String sang datetime :

```

from datetime import datetime

dt1 = datetime.strptime('01/01/2019', '%d/%m/%Y')
print(dt1)

dt2 = datetime.strptime('01-01-2019 23:59:59', '%d-%m-%Y
%H:%M:%S')
print(dt2)

```

4.2.2 Khoảng thời gian

Để đo sự chênh lệch giữa các mốc thời gian, thư viện datetime cung cấp kiểu dữ liệu timedelta. Kiểu dữ liệu này có thể hiểu là một bộ gồm 2 thành phần là số ngày (days) và số giây (seconds).

```

from datetime import timedelta

duration = timedelta(days=30, seconds=3600)

```

Tính khoảng thời gian giữa 2 thời điểm :

```

from datetime import datetime

dt1 = datetime(2019, 2, 1)
dt2 = datetime(2019, 2, 28, 23, 59)

duration = dt2 - dt1

days = duration.days
hours = duration.seconds // 3600
minutes = (duration.seconds % 3600) // 60
seconds = duration.seconds % 60

print(f'Duration : {days} days, {hours} hours, {minutes}
minutes, {seconds} seconds')

```

Cộng một mốc thời điểm với một khoảng thời gian :

```

from datetime import datetime, timedelta

dt1 = datetime(2019, 2, 1)
duration = timedelta(days=30, seconds=3600)
dt2 = dt1 + duration

print(dt2)

```

4.2.3 Định dạng datetime

Sử dụng hàm strftime() để trả về một chuỗi biểu diễn giá trị ngày, giờ và thời gian bằng cách sử dụng các đối tượng date, time và datetime.

```
import datetime
```

```
x = datetime.datetime(2020, 9, 19)
print(x.strftime("%B")) #Trả về September
```

Một số mã định dạng ngày tháng

Mã chỉ thị	Miêu tả	Ví dụ
%a	Thứ trong tuần, dạng ngắn	Wed
%A	Thứ trong tuần, dạng đầy đủ	Wednesday
%w	Ngày trong tuần dạng số từ 0-6, chủ nhật là ngày 0	3
%d	Ngày trong tháng thứ 01-31	30
%b	Tháng dạng ngắn	Dec
%B	Tháng dạng đầy đủ	December
%m	Tháng dạng số từ 01-12	12
%y	Năm dạng ngắn, không bao gồm thế kỷ (2 số cuối)	18
%Y	Năm dạng chuẩn	2018
%H	Giờ từ 00-23	12
%I	Giờ từ 00-12	11
%p	AM/PM	AM
%M	Phút từ 00-59	10
%S	Giây từ 00-59	35
%f	Mini giây từ 000000-999999	142123
%z	Giá trị utc	+0100
%Z	Timezone	CST
%j	Ngày trong năm từ 00-366	350
%U	Số tuần trong năm, chủ nhật là ngày đầu tuần từ 00-53	50
%W	Số tuần trong năm, thứ hai là ngày đầu tuần từ 00- 50	50
%c	Ngày giờ địa phương	Mon Dec 31 17:41:00 2018

Mã chỉ thị	Miêu tả	Ví dụ
%x	Ngày địa phương	12/31/18
%X	Giờ địa phương	17:41:00
%%	Ký tự %	%

4.2.4 Module calendar

Module Calendar (lịch) trong Python chứa lớp calendar cho phép thực hiện việc tính toán nhiều tác vụ khác nhau dựa theo ngày, tháng và năm. Trong đó, lớp TextCalendar và HTMLCalendar cho phép bạn thay đổi và sử dụng chúng tùy theo nhu cầu của mình.

Ví dụ: In lịch tháng 7/2025

The screenshot shows a code editor window with a script named 'Lich.py'. The code is as follows:

```

1 import calendar
2
3 c = calendar.TextCalendar(calendar.SUNDAY)
4 str = c.formatmonth(2025, 7)
5 print(str)

```

Below the code editor is a terminal window titled 'Run: Lich'. The terminal shows the output of the script:

```

E:\NhatNghe\python\venv\Scripts\python.exe
July 2025
Su Mo Tu We Th Fr Sa
      1  2  3  4  5
  6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31

```

Chú ý chọn ngày bắt đầu tuần của lịch:

c = calendar.TextCalendar(calendar.MONDAY)

July 2025

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	31			

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
prcal(year)	In cả calendar của năm
firstweekday()	Trả về ngày trong tuần đầu tiên. Theo mặc định là 0 mà xác định là Monday <code>print(calendar.firstweekday())</code> # Kết quả: 0
isleap(year)	Trả về True nếu năm đã cho là năm nhuận, nếu không là False <code>print(calendar.isleap(2020))</code> # True
monthcalendar(year,month)	Trả về một list gồm các ngày trong tháng đã cho của năm dưới dạng các tuần <code>print(calendar.monthcalendar(2025, 5))</code> [[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, 31, 0]]
leapdays(year1,year2)	Trả về số ngày nhuận từ năm year1 tới năm year2
prmonth(year,month)	In ra tháng đã cho của năm đã cung cấp <code>print(calendar.prmonth(2025, 5))</code> May 2025 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 31 None

4.2.5 Module pytz

Cài module: pip install pytz

Múi giờ (timezone) là gì? Đôi lúc, các bạn cần lấy dữ liệu múi giờ chính xác tại từng vùng, ví dụ muốn lấy dữ liệu múi giờ của “Asia/Ho_Chi_Minh”:

```
from datetime import datetime
from pytz import timezone

datetime_format = "%Y-%m-%d %H:%M:%S %Z %z"

# Current time in UTC
now_utc = datetime.now(timezone('UTC'))
print(now_utc.strftime(datetime_format))

# Convert to Asia/Ho_Chi_Minhh time zone
now_asia =
now_utc.astimezone(timezone('Asia/Ho_Chi_Minhh'))
print(now_asia.strftime(datetime_format))
```

Kết quả:

2021-05-05 15:26:07 UTC +0000

2021-05-05 22:26:07 +07 +0700

Lấy múi giờ theo mã quốc gia:

```
print(pytz.country_timezones('VN'))
```

sẽ cho ra:

['Asia/Ho_Chi_Minhh']

Nhưng:

```
print(pytz.country_timezones('US'))
```

sẽ cho:

```
['America/New_York', 'America/Detroit', 'America/Kentucky/Louisville',
'America/Kentucky/Monticello', 'America/Indiana/Indianapolis',
'America/Indiana/Vincennes', 'America/Indiana/Winamac',
'America/Indiana/Marengo', 'America/Indiana/Petersburg',
'America/Indiana/Vevay', 'America/Chicago',
```

```
'America/Indiana/Tell_City',           'America/Indiana/Knox',
'America/Menominee',                   'America/North_Dakota/Center',
'America/North_Dakota/New_Salem',      'America/North_Dakota/Beulah',
'America/Denver',                     'America/Boise',          'America/Phoenix',
'America/Los_Angeles',                'America/Anchorage',     'America/Juneau',
'America/Sitka',                      'America/Metlakatla',    'America/Yakutat',
'America/Nome',  'America/Adak',  'Pacific/Honolulu']
```

Ngoài ra ta có thể sử dụng:

- pytz.all_timezones: Lấy tất cả các timezone.
- pytz.country_timezones.items () : Lấy danh sách các quốc gia
- `for key, val in pytz.country_timezones.items () :`
 `print(key, '=', val, end=',')`

```
AD = ['Europe/Andorra'],
AE = ['Asia/Dubai'],
AU = ['Australia/Lord_Howe', 'Antarctica/Macquarie',
      'Australia/Hobart', 'Australia/Melbourne', 'Australia/Sydney',
      'Australia/Broken_Hill', 'Australia/Brisbane', 'Australia/Lindeman',
      'Australia/Adelaide', 'Australia/Darwin', 'Australia/Perth',
      'Australia/Eucla'],
...
SG = ['Asia/Singapore'],
VI = ['America/St_Thomas'],
VN = ['Asia/Ho_Chi_Minh'],
```

4.2.6 Module dateutil

Cài đặt: pip install python-dateutil

Giả sử hôm nay là ngày 06/06/2021. Các kết quả dưới đây tính từ ngày 06/06/2021.

Chèn thư viện

```
from datetime import *
from dateutil.relativedelta import *
import calendar
```

Lấy ngày giờ hiện tại

```
NOW = datetime.now()
print(NOW) # 2021-06-06 13:06:18.602744
```

Lấy ngày hiện tại

```
TODAY = date.today()
print(TODAY) # 2021-06-06
```

Lấy tuần kế

```
next_week = TODAY + relativedelta(weeks=+1)
print(next_week) # 2021-06-13
```

Lấy tháng kế

```
next_month = TODAY + relativedelta(months=+1)
print(next_month) # 2021-07-06
```

Lấy kết hợp ngày, tháng

```
next_month_plus_one_week = TODAY +
    relativedelta(months=+1, weeks=+1)
print(next_month_plus_one_week) # 2021-07-13
```

Bổ sung thêm time

```
added_time = TODAY + relativedelta(months=+1, weeks=+1,
hour=13)
print(added_time) # 2021-07-13 13:00:00
```

Lấy ngày thứ 6 của tuần tới

```
this_TGIF = TODAY + relativedelta(weekday=FR)
print(this_TGIF) # 2021-06-11
```

Lấy ngày thứ 3 của tuần tới

```
next_tuesday = TODAY + relativedelta(weeks=+1,
weekday=TU)
print(next_tuesday) # 2021-06-15
```

Lấy ngày thứ 3 của tuần tới (dùng thư viện calendar xác định ngày)

```
next_tuesday_calendar = TODAY + relativedelta(weeks=+1,
weekday=calendar.TUESDAY)
print(next_tuesday_calendar) # 2021-06-15
```

Bài 5: Kiểu cấu trúc

5.1 Kiểu List

5.1.1 Giới thiệu và khai báo

5.1.1.1 Giới thiệu

List là một danh sách các phần tử, các phần tử trong một list có thể có nhiều kiểu dữ liệu khác nhau. Các phần tử trong list được lưu trữ theo thứ tự. Giá trị của các phần tử có thể giống nhau và thay đổi được.

Các phần tử trong list có thể được truy xuất thông qua chỉ số. Cũng như các kiểu tập hợp khác, chỉ số trong list được đánh số từ 0.

```
num = [0, 2, 5, 4, 6, 7]

print (num[0])
print (num[2:])
print (len(num))
print (num + [8, 9])
```

Trong ví dụ trên, chúng ta có một list có 6 phần tử. List được khởi tạo trong cặp dấu []. Các phần tử của list được phân cách bởi dấu phẩy.

```
0
[5, 4, 6, 7]
6
[0, 2, 5, 4, 6, 7, 8, 9]
```

Các phần tử trong một list có thể mang nhiều kiểu dữ liệu khác nhau.

```
1 class Being:
2     pass
3
4     objects = [1, -2, 3.4, None, False, [1, 2], "Python",
5         (2, 3), Being(), {}]
6     print (objects)
```

Ví dụ trên tạo ra một list có nhiều kiểu dữ liệu khác nhau, bao gồm kiểu số nguyên, kiểu bool, một string, một tuple, một kiểu từ điển và nguyên cả một list khác.

```
1 [1, -2, 3.4, None, False, [1, 2], 'Python', (2, 3),  
<__main__.Being instance at 0xb76a186c>, {}]
```

5.1.1.2 Khởi tạo list

Cách khởi tạo ở trên là cách khởi tạo bằng cách đưa giá trị trực tiếp, ngoài ra bạn có thể khởi tạo mà không cần biết giá trị cụ thể:

```
1 n1 = [0 for i in range(15)]  
2 n2 = [0] * 15  
3  
4 print (n1)  
5 print (n2)  
6  
7 n1[0:11] = [10] * 10  
8  
9 print (n1)
```

Ví dụ trên mô tả hai cách khởi tạo một list. Cách đầu tiên dùng vòng lặp for với cú pháp **comprehension** bên trong cặp dấu [], cách thứ hai là dùng toán tử *.

```
1 n1 = [0 for i in range(15)]  
2 n2 = [0] * 15
```

Hai dòng code trên ví dụ về cách khởi tạo list dùng toán tử * và cú pháp **comprehension**, cú pháp comprehension sẽ được giải thích ở gần cuối bài này. Kết quả đều cho ra một list các phần tử có giá trị là 0.

```
1 n1[0:11] = [10] * 10
```

Dòng code trên gán 10 phần tử đầu tiên với giá trị 10.

```
1 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
2 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
3 [10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 0, 0, 0]
```

5.1.1.3 Hàm list()

Hàm **list()** có tác dụng khởi tạo một list từ một đối tượng tập hợp, có thể là 1 tuple, 1 list khác... hoặc tạo một list rỗng.

```
1 a = []  
2 b = list()  
3  
4 print (a == b)  
5  
6 print (list((1, 2, 3)))  
7 print (list("NhatNghe"))
```

```
8     print (list(['PHP', 'Python', 'ASP.NET Core']))
```

Trong ví dụ trên, chúng ta khởi tạo một list rỗng, một list từ một string, một list từ một tuple và một list từ một list khác.

```
1     a = []
2     b = list()
```

Hai dòng trên là 2 cách để khởi tạo một list rỗng.

```
1     print (a == b)
```

Dòng trên sẽ cho ra kết quả True vì a và b bằng nhau.

```
1     print (list((1, 2, 3)))
```

Dòng trên tạo một list từ một tuple.

```
1     print (list("NhatNghe"))
```

Dòng code trên tạo một list từ một string.

```
1     print (list(['PHP', 'Python', 'ASP.NET Core']))
```

Cuối cùng là tạo một list từ một list khác.

```
1     True
2     [1, 2, 3]
3     ['Z', 'e', 't', 'C', 'o', 'd', 'e']
4     ['PHP', 'Python', 'ASP.NET Core']
```

5.1.1.4 Khởi tạo list bằng comprehension

Comprehension là một kiểu cú pháp dùng khi khởi tạo list từ một list khác. Bạn đã thấy cú pháp này ở các phần trên và trong các bài trước.

```
1     L = [<biểu thức> for <biến> in <danh sách> [câu lệnh if]]
```

Đoạn code mã giả trên là cú pháp comprehension. Khi chúng ta khởi tạo list theo cú pháp này. Một vòng for sẽ lặp qua danh sách cho trước, cứ mỗi lần lặp, nó sẽ kiểm tra xem nếu câu lệnh if có thỏa điều kiện không, nếu thỏa thì thực hiện biểu thức và gán giá trị trả về của biểu thức cho list L.

```
1     a = [1, 2, 3, 4, 5, 6, 7, 8, 9]
2     b = [e + 2 for e in a if e % 2]
3     print (b)
```

Trong ví dụ trên, chúng ta tạo list a gồm các con số. Chúng ta tạo một list b gồm các phần tử là các số nguyên không thể chia hết cho 2 trong list a và cộng thêm 2 cho số đó.

```
1     b = [e + 2 for e in a if e % 2]
```

Trong vòng lặp for, mỗi lần lặp sẽ kiểm tra xem nếu e có chia hết cho 2 hay không, nếu không thì thực hiện biểu thức, biểu thức ở đây là cộng e thêm 2, sau đó thêm vào list b.

Cú pháp comprehension có thể viết một cách tương tự như sau:

```
1     a = [1, 2, 3, 4, 5, 6, 7, 8, 9]
2
3     b = list()
4     for e in a:
5         if (e % 2):
6             b.append(e + 2)
7
8     print (b)
```

Đoạn code trên tương đương với cú pháp comprehension.

```
1     [3, 5, 7, 9, 11]
```

5.1.2 Các phép tính trên list

Đoạn code dưới đây ví dụ về các phép tính trên list.

```
1     n1 = [1, 2, 3, 4, 5]
2     n2 = [3, 4, 5, 6, 7]
3
4     print (n1 == n2)
5     print (n1 + n2)
6
7     print (n1 * 3)
8
9     print (2 in n1)
10    print (2 in n2)
```

Chúng ta khởi tạo 2 list gồm các số nguyên và thực hiện một số phép tính.

```
1     print (n1 == n2)
```

Dòng code trên thực hiện phép so sánh thông qua toán tử `==`. Kết quả ra `False` vì các phần tử trong 2 list là khác nhau.

```
1 print (n1 + n2)
```

Toán tử + sẽ gộp hai list lại với nhau và tạo thành một list với chứa các phần tử của 2 list con.

```
1 print (n1 * 3)
```

Toán tử * sẽ nhân số lượng phần tử trong list lên n lần, trong trường hợp trên là 3 lần.

```
1 print (2 in n1)
```

Toán tử in kiểm tra xem một giá trị nào đó có tồn tại trong list hay không. Giá trị trả về là True hoặc False.

```
1 False
2 [1, 2, 3, 4, 5, 3, 4, 5, 6, 7]
3 [1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
4 True
5 False
```

5.1.3 Một số hàm trên kiểu dữ liệu List

5.1.3.1 Hàm len():

trả về số phần tử của một List

Ví dụ:

```
>>> ds = [1, 2, 3, 4, 5] ; print(len(ds))
5
```

5.1.3.2 Hàm append():

Thêm mới một phần tử vào List

Ví dụ:

```
>>> ds = [1, 2, 3, 4, 5] ; ds.append(6); print(ds)
[1, 2, 3, 4, 5, 6]
```

5.1.3.3 Hàm remove():

Xóa một phần tử khỏi List

Ví dụ:

```
>>> ds = [1, 2, 3, 4, 5] ; ds.remove(3); print(ds)
[1, 2, 4, 5]
```

5.1.3.4 Hàm extend():

Ghép một List con vào cuối một List

Ví dụ:

```
>>> ds = [1, 2, 3, 4, 5] ; ds.extend([6, 7]); print(ds)
[1, 2, 3, 4, 5, 6, 7]
```

5.1.3.5 Phép cộng 2 List:

Trả về một List mới bằng cách ghép 2 List với nhau

Ví dụ:

```
>>> ds = [1, 2, 3, 4, 5] ; ds += [6, 7]; print(ds)
[1, 2, 3, 4, 5, 6, 7]
```

5.1.3.6 Hàm reverse():

Đảo ngược một List

Ví dụ:

```
>>> ds = [1, 2, 3, 4, 5] ; ds.reverse(); print(ds)
[5, 4, 3, 2, 1]
```

5.1.3.7 Phép toán in:

Kiểm tra một phần tử có nằm trong List

Ví dụ:

```
>>> ds = [1, 2, 3, 4, 5] ; print(1 in ds)
True
```

5.1.3.8 Phép toán not in:

Kiểm tra một phần tử có không nằm trong List

Ví dụ:

```
>>> ds = [1, 2, 3, 4, 5] ; print(3 not in ds)
False
```

5.1.3.9 Hàm index():

Tìm vị trí của một phần tử trong List

Ví dụ:

```
>>> ds = [1, 2, 3, 4, 5] ; print(ds.index(3))
2
```

5.1.3.10 Hàm min():

Trả về giá trị nhỏ nhất của một List

Ví dụ:

```
>>> ds = [1, 3, 2, 5, 4] ; print(min(ds))
1
```

5.1.3.11 Hàm max():

Trả về giá trị lớn nhất của một List

Ví dụ:

```
>>> ds = [1, 3, 2, 5, 4] ; print(max(ds))
5
```

5.1.3.12 Hàm sum():

Trả về tổng giá trị của một List số

Ví dụ:

```
>>> ds = [1, 3, 2, 5, 4] ; print(sum(ds))
15
```

Truy nhập đến từng phần tử của List

Tương tự kiểu String, kiểu List cũng có thể truy nhập đến từng phần tử hoặc lấy ra các đoạn con bằng cách sử dụng các cấu trúc:

- `lst[i]` : phần tử thứ `i` của List
- `lst[start:end]` : List con chứa các phần tử từ chỉ số `start` đến `end-1`
- `lst[start:]` : List con chứa các phần tử từ chỉ số `start` đến cuối List
- `lst[:end]` : List con chứa các phần tử từ chỉ số `0` đến `end-1`

5.1.4 Sắp xếp một List

Để sắp xếp các phần tử trong một List theo một thứ tự tăng dần hoặc giảm dần có thể sử dụng hàm:

```
sorted(lst [, key=key_function, reverse=True/False])
```

Các tham số cần truyền vào cho hàm sắp xếp:

- `lst` : danh sách cần sắp xếp
- `key_function` (không bắt buộc) : Hàm thuộc tính để sắp xếp (có nghĩa sắp xếp các phần tử theo giá trị của thuộc tính này). Nếu tham số này không truyền vào thì nó sẽ có giá trị mặc định là hàm đồng nhất ($f(x)=x$), tức sắp xếp các phần tử theo giá trị của các phần tử.
- `reverse` (không bắt buộc): Nếu giá trị này bằng `True` thì sẽ sắp xếp theo thứ tự giảm dần, nếu giá trị này không được truyền vào thì sẽ sắp xếp theo thứ tự tăng dần

Ví dụ:

```
>>> print(sorted([1, 3, 2, 5, 4]))  
[1, 2, 3, 4, 5]  
>>> print(sorted([1, 3, 2, 5, 4], reverse=True))  
[5, 4, 3, 2, 1]  
>>> def number_of_element(x): return len(x)  
...  
>>> print(sorted([[1,2], [3,4,6],[1]], key=number_of_element)) # sắp xếp  
theo kích thước các phần tử trong list  
[[1], [1, 2], [3, 4, 6]]
```

Giá trị hàm thuộc tính `key_function` đôi khi có thể được truyền vào hàm `sorted` dưới dạng "*lambda function*:

```
print(sorted([[1,2], [3,4,6],[1]], key=lambda x : len(x)))
```

Biểu thức

```
lambda x : len(x)
```

Tương đương với giá trị của một hàm `f` trong đó `f` được định nghĩa bằng

```
def f(x) : return len(x)
```

Ví dụ:

Chuyển một số trong phạm vi 0-99 thành phát âm tiếng Việt. Ví dụ : 85 → tám mươi lăm

```
bangso = ['không', 'một', 'hai', 'ba', 'bốn', 'năm', 'sáu', 'bảy',  
'tám', 'chín']  
x = int(input('x='))  
if x < 10:  
    text = bangso[x]  
else:  
    chuc = x // 10  
    donvi = x % 10  
    text = (bangso[chuc] + ' mươi') if chuc > 1 else 'mười'  
    if donvi > 0:  
        text += ' '  
        if donvi == 5:  
            text += 'lăm'  
        elif donvi == 1 and chuc > 1:  
            text += 'mốt'  
        else:  
            text += bangso[donvi]  
print(text)
```

5.1.5 Thêm phần tử mới vào list

Để thêm một phần tử chúng ta dùng phương thức `append()` hoặc `insert()`, mặc định `append()` thêm phần tử mới vào cuối list, còn `insert()` có thể thêm phần tử vào vị trí bất kỳ do chúng ta quy định.

```
1 langs = list()
2
3 langs.append("Python")
4 langs.append("Perl")
5 print (langs)
6
7 langs.insert(0, "PHP")
8 langs.insert(2, "Lua")
9 print (langs)
10
11 langs.extend(("JavaScript", "ActionScript"))
12 print (langs)
```

Có 3 phương thức để thêm một phần tử vào list, đó là các phương thức append(), insert() và extend().

```
1 langs.append("Python")
2 langs.append("Perl")
```

Phương thức append() thêm một phần tử vào cuối list.

```
1 langs.insert(0, "PHP")
2 langs.insert(2, "Lua")
```

Phương thức insert() có thể thêm phần tử vào vị trí bất kỳ trong list.

```
1 langs.extend(("JavaScript", "ActionScript"))
```

Trong khi phương thức append() chỉ thêm một phần tử vào cuối list thì phương thức extend() có thể thêm một dãy các phần tử vào cuối list.

```
1 ['Python', 'Perl']
2 ['PHP', 'Python', 'Lua', 'Perl']
3 ['PHP', 'Python', 'Lua', 'Perl', 'JavaScript',
   'ActionScript']
```

5.1.6 Lỗi IndexError, TypeError

Lỗi exception IndexError là lỗi khi chúng ta thực hiện các thao tác có sử dụng chỉ số mà chỉ số vượt ngoài phạm vi list. Loại lỗi exception sẽ được đề cập ở các bài sau.

```
1 n = [1, 2, 3, 4, 5]
2
3 try:
4     n[0] = 10
5     n[6] = 60
6 except IndexError as e:
7     (print e)
```

Trong ví dụ trên, chúng ta có một list có 5 phần tử, các phần tử được đánh số từ 0..4

```
1 n[6] = 60
```

Nếu chúng ta truy xuất phần tử vượt quá 4 thì lỗi IndexError sẽ xảy ra, trong ví dụ trên chúng ta truy xuất phần tử thứ 6.

```
1 list assignment index out of range
```

Một lỗi exception khác là TypeError, lỗi này xảy ra khi chỉ số chúng ta đưa vào không phải là kiểu số.

```
1 n = [1, 2, 3, 4, 5]
2
3 try:
4     print (n[1])
5     print (n['2'])
6 except TypeError as e:
7     print ("Error in file %s" % __file__)
8     print ("Message: %s" % e)
```

Ví dụ trên sẽ xảy ra lỗi TypeError.

```
1 print (n['2'])
```

Chỉ số bắt buộc phải là kiểu số, tất cả các kiểu dữ liệu còn lại đều không hợp lệ.

```
1 2
2 Error in file ./typeerror.py
3 Message: list indices must be integers, not str
```

5.1.7 Xóa phần tử trong list

```
1 langs = ["Python", "Ruby", "Perl", "Lua", "JavaScript"]
2 print (langs)
3
4 langs.pop(3)
5 langs.pop()
6 print (langs)
7
8 langs.remove("Ruby")
9 print (langs)
```

Chúng ta có hai phương thức để xóa phần tử trong một list là phương thức pop() và phương thức remove(). Phương thức pop() xóa phần tử thông qua chỉ số, phương thức remove() xóa phần tử thông qua giá trị.

```
1 langs.pop(3)
2 langs.pop()
```

Ở đoạn code trên chúng ta xóa phần tử có chỉ số là 3, nếu chúng ta không đưa chỉ số vô thì Python sẽ tự động xóa phần tử cuối cùng.

```
1 langs.remove("Ruby")
```

Dòng trên xóa phần tử theo giá trị, phần tử đầu tiên trong list có giá trị "Ruby" sẽ bị xóa.

```
1 ['Python', 'Ruby', 'Perl', 'Lua', 'JavaScript']
2 ['Python', 'Ruby', 'Perl']
3 ['Python', 'Perl']
```

Ngoài ra chúng ta có thẻ xóa phần tử của list bằng từ khóa del.

```
1 langs = ["Python", "Ruby", "Perl", "Lua", "JavaScript"]
2 print (langs)
3
4 del langs[1]
5 print (langs)
6
7 #del langs[15]
8
9 del langs[:]
10 print (langs)
```

Trong ví dụ trên chúng ta có một list các string, chúng ta sẽ xóa các phần tử bằng từ khóa del.

```
1 del langs[1]
```

Dòng trên xóa phần tử thứ 1.

```
1 #del langs[15]
```

Nếu chúng ta xóa các phần tử có chỉ số vượt ngoài phạm vi của list thì lỗi IndexError sẽ xảy ra.

```
1 del langs[:]
```

Dòng code trên xóa toàn bộ list.

```
1 ['Python', 'Ruby', 'Perl', 'Lua', 'JavaScript']
2 ['Python', 'Perl', 'Lua', 'JavaScript']
3 []
```

Để xóa nguyên list dùng hàm clear().

5.1.8 Thay đổi giá trị phần tử trong list

```
1 langs = ["Python", "Ruby", "Perl"]
2
3 langs.pop(2)
4 langs.insert(2, "PHP")
5 print (langs)
6
7 langs[2] = "Perl"
8 print (langs)
```

Đoạn code trên sửa đổi giá trị của phần tử cuối cùng trong list.

```
1 langs.pop(2)
2 langs.insert(2, "PHP")
```

Trong hai dòng code trên, chúng ta xóa phần tử cuối cùng sau đó thêm một phần tử mới vào, cách này khá rườm rà và mất thời gian.

```
1 langs[2] = "Perl"
```

Thay vào đó bạn chỉ đơn giản là gán cho phần tử đó một giá trị mới là được.

```
1 ['Python', 'Ruby', 'PHP']
2 ['Python', 'Ruby', 'Perl']
```

5.1.9 Sao chép một list

```
1 import copy
2
3 w = ["Python", "Ruby", "Perl"]
4
5 c1 = w[:]
6 c2 = list(w)
7 c3 = copy.copy(w)
8 c4 = copy.deepcopy(w)
9 c5 = [e for e in w]
10
11 c6 = []
12 for e in w:
13     c6.append(e)
14
15 c7 = []
16 c7.extend(w)
17
18 print (c1, c2, c3, c4, c5, c6, c7)
```

Đoạn code trên chúng ta có một list chứa 3 string. Chúng ta sao chép list này sang 7 list khác.

```
1 import copy
```

Trong Python có module copy, module này có 2 phương thức dùng cho việc sao chép.

```
1 c1 = w[:]
```

Cách một bạn dùng cú pháp như trên sẽ copy được một list.

```
1 c2 = list(w)
```

Cách thứ 2 là dùng hàm list() để khởi tạo ngay một list.

```
1 c3 = copy.copy(w)
2 c4 = copy.deepcopy(w)
```

Cả hai phương thức copy() và deepcopy() đều có tác dụng là tạo ra bản sao của một list. Đến đây nếu bạn không muốn nghe giải thích về sự khác nhau của 2 phương thức này thì có thể bỏ qua đoạn dưới vì cũng không cần thiết lắm :D.

Nếu đã từng học hướng đối tượng thì bạn đã biết là một đối tượng có một phần bộ nhớ dành riêng cho chúng trong máy tính. Khi chúng ta tạo một list trong Python, chẳng hạn như `a = [1, 2, 3]`, một ô nhớ sẽ được cấp cho biến `a` để lưu trữ thông tin về 3 phần tử của nó, và 3 ô nhớ riêng được cấp cho 3 phần tử của list `a`. Khi chúng ta dùng phương thức `copy()`, vd `b = copy.copy(a)`, thì biến `b` được cấp một vùng nhớ riêng, còn 3 phần tử của biến `b` cũng được cấp vùng nhớ riêng nhưng chúng không trực tiếp lưu trữ giá trị như biến `a` mà bản thân chúng là một con trỏ tham chiếu đến 3 phần tử của biến `a`. Còn nếu dùng phương thức `deepcopy()` thì biến `b` cũng được cấp một vùng nhớ riêng, và 3 phần tử của biến `b` cũng được cấp vùng nhớ riêng và lưu trữ giá trị giống như biến `a`, có nghĩa là lúc này chúng là các thực thể độc lập, không liên quan đến nhau.

```
1 c5 = [e for e in w]
```

Đoạn code trên tạo ra một bản copy khác bằng cách dùng vòng lặp.

```
1 c6 = []
2 for e in w:
3     c6.append(e)
```

Dòng trên cũng không có gì khó hiểu.

```
1 c7 = []
2 c7.extend(w)
```

Phương thức `extend` cũng có thể được dùng để tạo bản sao.

```
1  ['Python', 'Ruby', 'Perl'] ['Python', 'Ruby', 'Perl']
2  ['Python', 'Ruby', 'Perl']
3  ['Python', 'Ruby', 'Perl'] ['Python', 'Ruby', 'Perl']
4  ['Python', 'Ruby', 'Perl']
5  ['Python', 'Ruby', 'Perl']
```

5.1.10 Chỉ số trong list

Như chúng ta đã biết, chúng ta có thể truy xuất phần tử thông qua chỉ số. Chỉ số trong Python được đánh số từ 0. Ngoài ra chúng ta có thể dùng chỉ số âm để lấy lùi.

```
1  n = [1, 2, 3, 4, 5, 6, 7, 8]
2
3  print (n[0])
4  print (n[-1])
5  print (n[-2])
6
7  print (n[3])
8  print (n[5])
```

Chỉ số được đặt bên trong cặp dấu [].

```
1  1
2  8
3  7
4  4
5  6
```

Phương thức `index(e, start, end)` tìm xem có phần tử nào có giá trị giống như tham số e hay không trong phạm vi từ start đến end. Chúng ta có thể không cần đưa hai tham số sau vào và Python sẽ tự ngầm tìm từ đầu đến cuối list.

```
1  n = [1, 2, 3, 4, 1, 2, 3, 1, 2]
2
3  print (n.index(1))
4  print (n.index(2))
5
6  print (n.index(1, 1))
7  print (n.index(2, 2))
8
9  print (n.index(1, 2, 5))
10 print (n.index(3, 4, 8))
```

Đoạn code trên ví dụ về phương thức `index()`.

```
1  print (n.index(1))
2  print (n.index(2))
```

Hai dòng code trên tìm vị trí của phần tử đầu tiên có giá trị 1 và 2.

```
1 print (n.index(1, 1))
2 print (n.index(2, 2))
```

Hai dòng code trên cũng tìm phần tử có giá trị 1 và 2 bắt đầu từ vị trí số 1.

```
1 print (n.index(1, 2, 5))
```

Đoạn code trên tìm phần tử có giá trị 1 trong phạm vi từ 2 đến 5.

```
1 0
2 1
3 4
4 5
5 4
6 6
```

5.1.11 Duyệt chuỗi

Tiếp theo chúng ta sẽ tìm hiểu cách duyệt một chuỗi.

```
1 n = [1, 2, 3, 4, 5]
2
3 for e in n:
4     print (e,)
5
6 print ()
```

Đoạn code trên duyệt chuỗi bằng vòng lặp for khá đơn giản.

```
1 1 2 3 4 5
```

Đoạn code dưới đây hơi rườm rà hơn một tí.

```
1 n = [1, 2, 3, 4, 5]
2
3 i = 0
4 s = len(n)
5
6 while i < s:
7     print (n[i],)
8     i = i + 1
9
10 print()
```

Chúng ta duyệt qua chuỗi bằng vòng lặp while.

```
1 i = 0
2 l = len(n)
```

Đầu tiên chúng ta định nghĩa biến đếm và biến lưu trữ chiều dài của list.

```
1 while i < s:  
2     print (n[i],)  
3     i = i + 1
```

Cứ mỗi lần lặp, chúng ta tăng biến đếm lên.

Khi dùng vòng lặp for với hàm enumerate(), chúng ta có thể lấy được cả chỉ số và giá trị của các phần tử trong list.

```
1 n = [1, 2, 3, 4, 5]  
2  
3 for e, i in enumerate(n):  
4     print ("n[%d] = %d" % (e, i))
```

```
1 n[0] = 1  
2 n[1] = 2  
3 n[2] = 3  
4 n[3] = 4  
5 n[4] = 5
```

5.1.12 Đếm số lượng phần tử trong list

Không những đếm số lượng phần tử của list mà còn đếm số lần xuất hiện của các phần tử nhất định trong list.

```
1 n = [1, 1, 2, 3, 4, 4, 4, 5]  
2  
3 print (n.count(4))  
4 print (n.count(1))  
5 print (n.count(2))  
6 print (n.count(6))
```

Chúng ta dùng phương thức count(), trong ví dụ trên, chúng ta đếm số lần xuất hiện của một vài chữ số trong list.

```
1 n = [1, 1, 2, 3, 4, 4, 4, 5]
```

Chúng ta tạo ra một list. Trong đó số 1 và số 4 được xuất hiện nhiều lần.

```
1 print (n.count(4))  
2 print (n.count(1))  
3 print (n.count(2))  
4 print (n.count(6))
```

Ở đây chúng ta đếm số lần xuất hiện của số 4, 1, 2, và 6.

```
1    3  
2    2  
3    1  
4    0
```

5.1.13 Lồng các list lại với nhau

Một list không chỉ chứa các phần tử đơn lẻ mà có thể chứa nguyên cả một list khác.

```
1     nums = [[1, 2], [3, 4], [5, 6]]  
2  
3     print (nums[0])  
4     print (nums[1])  
5     print (nums[2])  
6  
7     print (nums[0][0])  
8     print (nums[0][1])  
9  
10    print (nums[1][0])  
11    print (nums[2][1])  
12  
13    print (len(nums))
```

Trong ví dụ trên, chúng ta tạo ra một list chứa 3 list khác.

```
1     print (nums[0][0])  
2     print (nums[0][1])
```

Để truy xuất phần tử của list con thì chúng ta dùng hai cặp dấu ngoặc vuông [].

```
1     [1, 2]  
2     [3, 4]  
3     [5, 6]  
4     1  
5     2  
6     3  
7     6  
8     3
```

Đoạn code dưới đây lồng 4 list lại với nhau.

```
1     nums = [[1, 2, [3, 4, [5, 6]]]]  
2  
3     print (nums[0])  
4     print (nums[0][2])  
5     print (nums[0][2][2])  
6  
7     print (nums[0][0])
```

```
8     print (nums[0][2][1])
9     print (nums[0][2][2][0])
```

```
1     print (nums[0][0])
2     print (nums[0][2][1])
3     print (nums[0][2][2][0])
```

Để truy xuất các phần tử con thì chúng ta cứ dùng thêm các cặp dấu ngoặc vuông [].

```
1     [1, 2, [3, 4, [5, 6]]]
2     [3, 4, [5, 6]]
3     [5, 6]
4     1
5     4
6     5
```

5.1.14 Sắp xếp list

```
1     n = [3, 4, 7, 1, 2, 8, 9, 5, 6]
2     print (n)
3
4     n.sort()
5     print (n)
6
7     n.sort(reverse=True)
8     print (n)
```

Trong ví dụ trên, chúng ta có một list chứa các con số không theo một thứ tự nào cả. Chúng ta sẽ sắp xếp chúng bằng phương thức sort().

```
1     n.sort()
```

Mặc định phương thức sort() sẽ sắp xếp từ bé đến lớn.

```
1     n.sort(reverse=True)
```

Nếu muốn sắp xếp theo chiều ngược lại, bạn cho tham số reverse là True.

```
1     [3, 4, 7, 1, 2, 8, 9, 5, 6]
2     [1, 2, 3, 4, 5, 6, 7, 8, 9]
3     [9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Phương thức sort() thay đổi chính list gốc, nếu bạn không muốn thay đổi list gốc mà chỉ muốn có một bản copy list mới đã được sắp xếp thì dùng phương thức sorted().

```
1 n = [3, 4, 1, 7, 2, 5, 8, 6]
2
3 print (n)
4 print (sorted(n))
5 print (n)
```

```
1 [3, 4, 1, 7, 2, 5, 8, 6]
2 [1, 2, 3, 4, 5, 6, 7, 8]
3 [3, 4, 1, 7, 2, 5, 8, 6]
```

Lưu ý chúng ta không thể sắp xếp một list chứa nhiều kiểu dữ liệu khác nhau được, chẳng hạn [1, "Hello"] không thể nào được sắp xếp, vì chúng không cùng kiểu, bạn chỉ có thể sắp xếp các phần tử có cùng kiểu dữ liệu.

5.1.15 Hàm map() và hàm filter()

Hai hàm map() và filter() là hai hàm có tác dụng thực thi các câu lệnh trên các phần tử của list và trả về một list mới. Hai hàm này có tác dụng tương tự như cú pháp comprehension. Hiện tại thì người ta có xu hướng dùng cú pháp comprehension nhiều hơn do tính hữu dụng của nó.

```
1 def to_upper(s):
2     return s.upper()
3
4 words = ["stone", "cloud", "dream", "sky"]
5
6 words2 = map(to_upper, words)
7 print (words2)
```

Hàm map() có tác dụng thực thi một hàm khác lên từng phần tử của một list.

```
1 def to_upper(s):
2     return s.upper()
```

Trong ví dụ trên thì đây là hàm mà chúng ta sẽ dùng để thực thi cho các phần tử của list. Ở đây chúng ta viết hoa một string cho trước.

```
1 words2 = map(to_upper, words)
2 print (words2)
```

Tham số đầu tiên của hàm map() là tên của hàm sẽ được thực thi, tham số thứ hai là list nguồn.

```
1     ['STONE', 'CLOUD', 'DREAM', 'SKY']
```

Hàm filter() cũng thực thi một hàm nhưng không làm gì với list gốc cả mà chỉ đơn giản là lấy các phần tử của list gốc nếu chúng thỏa điều kiện nào đó.

```
1     def positive(x):
2         return x > 0
3
4     n = [-2, 0, 1, 2, -3, 4, 4, -1]
5
6     print (filter(positive, n))
```

Trong ví dụ trên, chúng ta tạo một list có các số nguyên. Hàm filter() sẽ lọc ra các phần tử > 0 bằng hàm positive().

```
1     def positive(x):
2         return x > 0
```

Đoạn code trên là hàm do chúng ta định nghĩa, hàm này sẽ được sử dụng bởi hàm filter(). Giá trị trả về của hàm này là một kiểu bool.

```
1     [1, 2, 4, 4]
```

5.2 Kiểu Tuple

5.2.1 Giới thiệu và khai báo

Tuple trong Python là một kiểu dữ liệu dùng để lưu trữ các đối tượng không thay đổi (không thêm, sửa, xóa được) về sau (giống như hằng số). Còn lại thì cách lưu trữ của nó cũng khá giống như kiểu dữ liệu list mà bài trước chúng ta đã được tìm hiểu. Các phần tử của dữ liệu kiểu nhóm được đặt trong cặp dấu () và cách nhau bởi dấu phẩy. Tuple giống như kiểu dữ liệu enum ở một số ngôn ngữ.

Khai báo: (val1, val2, val3, ...)

```
Dayso = (1, 3, 5, 7, 9)
danh sach_hocsinh = ("Nguyễn Văn An", "Nguyễn Chí
Cường", "Nguyễn Mạnh Tuấn")
```

Các phần tử trong Tuple được đánh dấu từ 0 theo chiều từ trái qua phải.

Và ngược lại từ -1 theo chiều từ phải qua trái.

5.2.2 Truy cập các phần tử

Sử dụng vòng lặp for để truy cập vào các phần tử của Tuple:

Ví dụ:

```
ds_trai_cay = ("Cam", "Đào", "Mận", "Ổi", "Dừa", "Bơ", "Lê")
```

```
for trai_cay in ds_trai_cay:  
    print("Trái: ", trai_cay)
```

Truy cập thông qua chỉ số (index):

Bạn cũng có thể truy cập vào các phần tử của Tuple thông qua chỉ số. Các phần tử của Tuple được đánh chỉ chỉ từ trái sang phải, bắt đầu từ 0.

Ví dụ:

```
for i in range(0, len(ds_trai_cay)):  
    print("Phần tử thứ", i, " = ", ds_trai_cay[i])
```

5.2.3 Các phép tính trên Tuple

Các phép tính trên Tuple gần giống với phép tính trên List, tuy nhiên Tuple chỉ hỗ trợ các phép tính đọc thông tin của Tuple, không hỗ trợ các phép tính làm thay đổi nội dung của Tuple

- **Hàm len:** trả về số phần tử của một Tuple
- **Phép cộng (+) 2 Tuple:** trả về một Tuple mới bằng cách ghép 2 Tuple với nhau
- **Phép nhân (*):** trả về một Tuple mới bằng cách lặp lại n lần. Ví dụ: tuple * 2
- **Phép toán in:** Kiểm tra một phần tử có nằm trong Tuple
Ví dụ: "Abc" in ("One", "Abc") → True
- **Phép toán not in:** Kiểm tra một phần tử có không nằm trong Tuple
- **Hàm index:** Tìm vị trí của một phần tử trong Tuple.
Cú pháp: tuple.index(obj, [start, [stop]])
 - Trả về chỉ số nhỏ nhất trong danh sách tìm thấy phần tử có giá trị là obj. Ném ra ValueError nếu không tìm thấy.
 - Nếu có tham số start, stop, chỉ tìm từ chỉ số start đến chỉ số stop (Và không bao gồm stop).

- Hàm **min**: Trả về giá trị nhỏ nhất của một Tuple
- Hàm **max**: Trả về giá trị lớn nhất của một Tuple
- Truy nhập phần tử qua chỉ số: tương tự như List
- Hàm **count**: trả về số lần xuất hiện của obj trong tuple. Công thức: tuple.count(obj)
- Các hàm convert:
 - Từ tuple sang list: **list**(tuple)
 - Từ tuple sang set: **set**(tuple)
 - Từ tuple sang string: **str**(tuple)

5.3 Kiểu Dictionary

5.3.1 Giới thiệu

Kiểu dictionary là kiểu dữ liệu danh sách, trong đó các phần tử được lưu trữ theo các cặp khóa-giá trị (key-value). Các phần tử trong dictionary không có thứ tự, tức là bạn không thể truy xuất chúng qua chỉ số mà chỉ dùng khóa, ngoài ra vì không có thứ tự nên Python cũng không có sẵn các hàm sắp xếp như hàm sort(), tuy nhiên nếu muốn bạn vẫn có thể tự code lấy hàm sort cho riêng mình. Trong các ngôn ngữ khác thì kiểu dictionary hay được gọi là bảng băm. Trong dictionary không có 2 khóa trùng nhau.

5.3.2 Khởi tạo dictionary

Có nhiều cách tạo dictionay. Đầu tiên chúng ta sẽ tìm hiểu cách khởi tạo một dictionary dùng hàm hàm tạo dict.

```

1     weekend = { "Sun": "Sunday", "Mon": "Monday" }
2     vals = dict(one=1, two=2)
3
4     capitals = {}
5     capitals["svk"] = "Bratislava"
6     capitals["deu"] = "Berlin"
7     capitals["dnk"] = "Copenhagen"
8
9     d = { i: object() for i in range(4) }
10
11    print (weekend)
12    print (vals)
13    print (capitals)
14    print (d)

```

Đoạn code trên ví dụ về 4 cách để khởi tạo 1 dictionary trong Python.

```
1     weekend = { "Sun": "Sunday", "Mon": "Monday" }
```

Cách đầu tiên và cũng là cách đơn giản nhất. Chúng ta tạo dictionary bằng cách gán dữ liệu trực tiếp. Dictionary được bao bọc bởi cặp dấu ngoặc nhọn {}. Trong đó mỗi phần tử được gán giá trị theo cú pháp key1:value1, key2:value2..., các phần tử phân cách nhau bởi dấu phẩy.

```
1     vals = dict(one=1, two=2)
```

Cách thứ 2 là chúng ta dùng hàm dict().

```

1     capitals = {}
2     capitals["svk"] = "Bratislava"
3     capitals["deu"] = "Berlin"
4     capitals["dnk"] = "Copenhagen"

```

Trên đây là cách thứ 3, đầu tiên chúng ta khởi tạo dict rỗng bằng cặp dấu {}. Sau đó khởi tạo các khóa và gán giá trị, các khóa được tạo trong cặp dấu ngoặc vuông [].

```
1     d = { i: object() for i in range(4) }
```

Cũng giống như kiểu list, dictionary cũng có thể được khởi tạo theo cú pháp comprehension. Cú pháp này có 2 phần, phần đầu tiên là phần biểu thức i: object(), phần thứ 2 là vòng lặp for i in range(4). Tức là cứ mỗi lần lặp, giá trị từ biểu thức sẽ được thêm vào dictionary. Hàm object() khởi tạo một đối tượng kiểu object. Đối tượng này không có giá trị, do đó khi in ra màn hình python sẽ in thông tin về địa chỉ bộ nhớ.

```

1     {'Sun': 'Sunday', 'Mon': 'Monday'}
2     {'two': 2, 'one': 1}
3     {'svk': 'Bratislava', 'dnk': 'Copenhagen', 'deu':
4     'Berlin'}

```

```
5 {0: <object object at 0xb76cb4a8>, 1: <object object at  
0xb76cb4b0>,  
2: <object object at 0xb76cb4b8>, 3: <object object at  
0xb76cb4c0>}
```

Chuyển đổi JSON sang dictionary:

```
import json  
jsonstring = '{"name": "erik", "age": 38, "married": true}'  
my_dict = json.loads(jsonstring)
```

5.3.3 Các phép toán cơ bản

Tiếp theo chúng ta sẽ tìm hiểu về các phép toán cơ bản với dictionary.

```
1 basket = { 'oranges': 12, 'pears': 5, 'apples': 4 }  
2  
3 basket['bananas'] = 5  
4  
5 print (basket)  
6 print ("There are %d various items in the basket" %  
7 len(basket))  
8  
9 print (basket['apples'])  
10 basket['apples'] = 8  
11 print (basket['apples'])  
12  
13 print (basket.get('oranges', 'undefined'))  
    print (basket.get('cherries', 'undefined'))
```

Trong ví dụ trên chúng ta có một dictionary và chúng ta sẽ thực hiện một số phép toán với dict này.

```
1 basket = { 'oranges': 12, 'pears': 5, 'apples': 4 }
```

Đầu tiên chúng ta tạo dict với 3 cặp khóa-giá trị.

```
1 basket['bananas'] = 5
```

Tiếp theo chúng ta tạo thêm 1 cặp khóa-giá trị nữa. Ở đây khóa là bananas còn giá trị là 5.

```
1 print ("There are %d various items in the basket" %  
len(basket))
```

Chúng ta dùng hàm len() để đếm số lượng các cặp khóa-giá trị.

```
1 print (basket['apples'])
```

Tiếp theo chúng ta in ra màn hình giá trị của khóa apples.

```
1 basket['apples'] = 8
```

Dòng trên thay đổi giá trị của khóa apples thành 8.

```
1 print (basket.get('oranges', 'undefined'))
```

Phương thức get() trả về giá trị của khóa oranges, nếu không có khóa nào có tên như thế thì trả về dòng chữ undefined.

```
1 print (basket.get('cherries', 'undefined'))
```

Dòng trên sẽ trả về undefined vì không có khóa nào tên là cherries.

```
1 {'bananas': 5, 'pears': 5, 'oranges': 12, 'apples': 4}
2 There are 4 various items in the basket
3 4
4 8
5 12
6 undefined
```

Tiếp theo chúng ta tìm hiểu về 2 phương thức fromkeys() và setdefault().

```
1 basket = ('oranges', 'pears', 'apples', 'bananas')
2
3 fruits = {}.fromkeys(basket, 0)
4 print (fruits)
5
6 fruits['oranges'] = 12
7 fruits['pears'] = 8
8 fruits['apples'] = 4
9
10 print (fruits.setdefault('oranges', 11))
11 print (fruits.setdefault('kiwis', 11))
12
13 print (fruits)
```

Phương thức fromkeys() tạo một dictionary từ một list, còn phương thức setdefault() trả về giá trị của một khóa, nếu khóa đó không tồn tại thì nó tự động thêm một khóa mới với giá trị mặc định do chúng ta chỉ định.

```
1 basket = ('oranges', 'pears', 'apples', 'bananas')
```

Đầu tiên chúng ta tạo một list các string.

```
1 fruits = {}.fromkeys(basket, 0)
```

Tiếp theo chúng ta dùng phương thức fromkeys() để tạo dictionary từ list trên, trong đó các khóa sẽ là các phần tử của list, còn các giá trị sẽ được gán mặc định là 0. Lưu ý phương thức fromkeys là phương thức của một lớp nên cần được gọi từ tên lớp, trong trường hợp này là {}.

```
1 fruits['oranges'] = 12
2 fruits['pears'] = 8
3 fruits['apples'] = 4
```

Ba dòng trên thay đổi giá trị của các khóa trong dict.

```
1 print (fruits.setdefault('oranges', 11))
2 print (fruits.setdefault('kiwis', 11))
```

Ở 2 dòng trên, dòng đầu tiên sẽ in số 12 ra màn hình vì oranges là khóa đã tồn tại trong dict, dòng thứ 2 sẽ in số 11 ra màn hình vì khóa kiwis chưa tồn tại nên sẽ được tự động thêm vào dict với giá trị mặc định là 11.

```
1 {'bananas': 0, 'pears': 0, 'oranges': 0, 'apples': 0}
2 12
3 11
4 {'kiwis': 11, 'bananas': 0, 'pears': 8, 'oranges': 12,
  'apples': 4}
```

Trong ví dụ tiếp theo, chúng ta sẽ tìm hiểu cách nối 2 dictionary với nhau.

```
1 domains = { "de": "Germany", "sk": "Slovakia", "hu": "Hungary"}
2 domains2 = { "us": "United States", "no": "Norway" }
3
4 domains.update(domains2)
5
6 print (domains)
```

Để nối 2 dictionary thì chúng ta dùng phương thức update().

```
1 domains.update(domains2)
```

Chúng ta nối domains2 vào domains.

```
1 {'sk': 'Slovakia', 'de': 'Germany', 'no': 'Norway',
  2 'us': 'United States', 'hu': 'Hungary'}
```

Tiếp theo chúng ta học cách xóa một phần tử ra khỏi dictionary.

```
1     items = { "coins": 7, "pens": 3, "cups": 2,
2             "bags": 1, "bottles": 4, "books": 5 }
3
4     print (items)
5
6     items.pop("coins")
7     print (items)
8
9     del items["bottles"]
10    print (items)
11
12    items.clear()
13    print (items)
```

Trong ví dụ trên, chúng ta có 6 cặp khóa-giá trị, chúng ta sẽ xóa chúng ra khỏi items.

```
1     items.pop("coins")
```

Đầu tiên là phương thức pop(), phương thức này xóa một phần tử trong dictionary theo khóa.

```
1     del items["bottles"]
```

Cách thứ 2 là dùng từ khóa del, dòng code trên sẽ xóa phần tử có khóa bottles ra khỏi dict bằng từ khóa del.

```
1     items.clear()
```

Tiếp theo phương thức clear() có tác dụng xóa toàn bộ phần tử ra khỏi dictionary.

```
1     {'bags': 1, 'pens': 3, 'coins': 7, 'books': 5,
2      'bottles': 4, 'cups': 2}
3     {'bags': 1, 'pens': 3, 'books': 5, 'bottles': 4, 'cups': 2}
4     {'bags': 1, 'pens': 3, 'books': 5, 'cups': 2}
{ }
```

5.3.4 Làm việc với khóa và giá trị

Trong phần này chúng ta sẽ tìm hiểu phương thức keys(), values() và items(), phương thức keys() trả về list các khóa có trong dict, phương thức values() trả về list các giá trị, còn phương thức items() trả về list các tuple, trong đó mỗi tuple có 2 phần tử, phần tử đầu tiên là khóa, phần tử thứ 2 là giá trị.

```
1 domains = { "de": "Germany", "sk": "Slovakia", "hu":  
2 "Hungary",  
3 "us": "United States", "no": "Norway" }  
4  
5 print (domains.keys())  
6 print (domains.values())  
7 print (domains.items())  
8  
9 print ("de" in domains)  
print ("cz" in domains)
```

Đoạn code trên ví dụ về cách sử dụng các phương thức đã trình bày ở trên, ngoài ra chúng ta còn dùng thêm từ khóa in.

```
1 print (domains.keys())
```

Dòng trên trả về một list các khóa trong dict bằng phương thức keys().

```
1 print (domains.values())
```

Dòng tiếp theo trả về một list các giá trị trong dict bằng phương thức values().

```
1 print (domains.items())
```

Cuối cùng phương thức items() trả về list các tuple, mỗi tuple chứa 2 phần tử là khóa và giá trị trong dict.

```
1 print ("de" in domains)  
2 print ("cz" in domains)
```

Từ khóa in có tác dụng kiểm tra xem một giá trị nào đó có tồn tại trong dict hay không.

Giá trị trả về True hoặc False.

```
1 ['sk', 'de', 'no', 'us', 'hu']  
2 ['Slovakia', 'Germany', 'Norway', 'United States',  
3 'Hungary']  
4 [('sk', 'Slovakia'), ('de', 'Germany'), ('no',  
5 'Norway'),  
6 ('us', 'United States'), ('hu', 'Hungary')]  
True  
False
```

5.3.5 Duyệt dictionary

Trong phần này chúng ta sẽ tìm hiểu cách duyệt một dictionary bằng vòng lặp for.

```
1     domains = { "de": "Germany", "sk": "Slovakia", "hu":  
2         "Hungary",  
3             "us": "United States", "no": "Norway" }  
4  
5     for key in domains:  
6         print (key)  
7  
8     for k in domains:  
9         print (domains[k])  
10  
11    for k, v in domains.items():  
12        print ("{}: {}".format(k, v))
```

Trong ví dụ trên, chúng ta duyệt 3 lần, mỗi lần duyệt chúng ta lấy khóa, giá trị và cả khóa lẫn giá trị.

```
1     for key in domains:  
2         print (key)
```

Vòng lặp trên sẽ in ra danh sách các khóa trong dict.

```
1     for k in domains:  
2         print (domains[k])
```

Vòng lặp trên in ra danh sách các giá trị trong list.

```
1     for k, v in domains.items():  
2         print ("{}: {}".format(k, v))
```

Vòng lặp cuối cùng in ra cả khóa và giá trị.

```
1     sk  
2     de  
3     no  
4     us  
5     hu  
6     Slovakia  
7     Germany  
8     Norway  
9     United States  
10    Hungary  
11    sk: Slovakia  
12    de: Germany  
13    no: Norway  
14    us: United States  
15    hu: Hungary
```

Xóa phần tử:

```
phone_numbers = { 'Hien': '0989666999', 'NhatNghe': '39322735' }
print(phone_numbers)
del(phone_numbers['Hien'])
print(phone_numbers)
```

kết quả:

```
{'Hien': '0989666999', 'NhatNghe': '39322735'}
{'NhatNghe': '39322735'}
```

Lấy danh sách các key:

```
list(phone_numbers)
```

Kiểm tra key có tồn tại không?

```
print('Hien' in phone_numbers)
print('NhatNghe' not in phone_numbers)
```

Duyệt danh sách:

```
for name, phone_number in phone_numbers.items():
    print(name, ":", phone_number)
```

Các hàm xây dựng sẵn:

Method	What is does	Example
clear()	Remove all key/value pairs (empty the dictionary)	phone_numbers.clear()
get(key)	Get a single item with given key, with optional default value	phone_numbers.get('Martha', 'Unknown person')
items()	Returns a view object containing key-value pairs from the dictionary	phone_numbers.items()

Method	What is does	Example
keys()	Returns a view object with a list of all keys from the dictionary	phone_numbers.keys()
values()	Returns a view_object with a list of all values from the dictionary	phone_numbers.values()
pop(key, default_value)	Returns and removes the element with the specified key	phone_numbers.pop('Martha')
popitem()	Returns and removes the last inserted item (Python 3.7+) or a random item	phone_numbers.popitem()
setdefault(key, value)	Returns the value of specified key. If the key does not exist, it's inserted with the given value	phone_numbers.setdefault('John Doe', 1234)
update(iterable)	Add all pairs from given iterable, e.g. a dictionary	phone_numbers.update({"Alina": 1234, "Alice", 2345})

5.4 Kiểu Set

5.4.1 Giới thiệu

Kiểu dữ liệu Set dùng để chứa các phần tử của một tập hợp. So với kiểu List, kiểu Set có điểm khác là:

- Trong một Set không có 2 phần tử cùng giá trị.
- Không thể dùng index để truy nhập phần tử của một Set

5.4.2 Các phép tính trên Set

5.4.2.1 Hàm add:

Thêm giá trị mới vào Set

Ví dụ:

```
>>> s = set([1, 2]); s.add(3); print(s)
{1, 2, 3}
```

5.4.2.2 Hàm remove:

Xóa phần tử ra khỏi Set

Ví dụ:

```
>>> s = set([1, 2, 3]); s.remove(2); print(s)
{1, 3}
```

5.4.2.3 Phép in:

Kiểm tra giá trị có nằm trong Set

Ví dụ:

```
>>> s = set([1, 2, 3]); print(1 in s)
True
```

5.4.2.4 Phép not in:

Kiểm tra giá trị có không nằm trong tập hợp

Ví dụ:

```
>>> s = set([1, 2, 3]); print(3 not in s)
False
```

5.4.2.5 Hàm union:

Trả về hợp của 2 Set

Ví dụ:

```
>>> s1 = set([1, 2, 3]); s2 = set([3, 4, 5]);
print(s1.union(s2))
{1, 2, 3, 4, 5}
```

5.4.2.6 Hàm intersection:

Giao của 2 Set

Ví dụ:

```
>>> s1 = set([1, 2, 3]); s2 = set([3, 4, 5]);
print(s1.intersection(s2))
{3}
```

5.4.2.7 Hàm difference:

Hiệu của 2 Set

Ví dụ:

```
>>> s1 = set([1, 2, 3]); s2 = set([3, 4, 5]);
print(s1.difference(s2))
{1, 2}
```

5.4.3 Một số ví dụ minh họa

Ví dụ 1:

```
s1 = set([1, 2])
s1.add(3)
s1.remove(1)

if 1 not in s1:
    print("1 không nằm trong tập hợp ", s1)

if 2 in s1:
    print("2 nằm trong tập hợp ", s1)

s2 = set([3, 4])
s3 = set.union(s1, s2)
print(s3)

s4 = set.intersection(s1, s2)
print(s4)
```

Ví dụ 2:

```
weekdays = set(['Thứ hai', 'Thứ ba', 'Thứ tư', 'Thứ năm', 'Thứ sáu'])
weekends = set(['Thứ bảy', 'Chủ nhật'])

day = "Thứ ba"

if day in weekdays:
    print(day, ' là ngày làm việc')

if day in weekends:
    print(day, ' là ngày nghỉ cuối tuần')
```

Bài 6: Hàm (function)

6.1 Xây dựng, gọi sử dụng phương thức

6.1.1 Giới thiệu

Hàm là một phần của chương trình nhằm thực hiện một tác vụ cụ thể, có thể trả về một giá trị hay tùy chọn là không trả về. Hàm có thể sử dụng nhiều lần hoặc đôi khi chỉ một lần. Nhờ có hàm mà code của bạn dễ đọc, ngắn gọn, xúc tích.

6.1.2 Khai báo hàm

```
def function_name([parameters]):
```

```
    return value_return
```

- Hàm nếu không trả dữ liệu thì mặc định sẽ trả về giá trị **None**.
- Sử dụng TAB để phân biệt các khối mã thuộc về.

Ví dụ khai báo hàm tính và trả về giá trị tổng của 2 tham số đầu vào

```
def sum(a, b):
```

```
    return (a + b)
```

Cách gọi hàm:

```
sum(1, 2)
```

Hàm có hỗ trợ giá trị mặc định cho tham số khi không truyền vào. Ví dụ hàm sau:

```
def plus(c, d = 10):
```

```
    return (c + d)
```

Nếu gọi hàm trên như sau:

```
plus(15)
```

6.1.3 Tham số của hàm

Hàm có thể có các tham số (parameter). Các giá trị mà chúng ta truyền qua các tham số này được gọi là đối số (argument).

Ví dụ:

```
def loi_chao(ten='HIENLTH', khoa_hoc='Python cơ bản'):
    print("Chào bạn", ten, "Mời bạn tham gia khóa học", khoa_hoc)

loi_chao()
loi_chao(ten='Nhất Nghệ')
loi_chao(khoa_hoc='Python FastAPI')
loi_chao(ten='Nhất Nghệ', khoa_hoc='Data Science')
loi_chao(khoa_hoc='Machine Learning', ten='Nhất Nghệ')
```

Ta thấy, với hàm có tham số mặc định, tùy vào lời gọi hàm sẽ sử dụng tham số tương ứng.

Hàm trả về nhiều giá trị:

Sử dụng return value1, value2, ...

6.2 Anonymous Function (lambda)

6.2.1 Cú pháp

Ta có thể định nghĩa hàm mà không cần tên, đó là các viết hàm dạng anonymous hay lambda.

Cú pháp:

lambda arguments: expression

Ví dụ: Định nghĩa hàm gấp đôi

```
double = lambda x: x * 2
và gọi hàm: print(double(5))
```

Hàm trên tương đương với:

```
def double(x):
    return x * 2
```

Một ví dụ khác:

```
full_name = lambda first, last: f'Full name: {first.title()}\n{last.title()}'\nprint(full_name('Hien', 'Luong'))
```

Hàm này có 2 tham số first và last.

Hàm Lambda được sử dụng khi cần một hàm vô danh trong thời gian ngắn, ví dụ như dùng làm đối số cho một hàm bậc cao hơn. Hàm Lambda thường được sử dụng cùng với các hàm Python tích hợp sẵn như filter() hay map(),..

6.2.2 Hàm Lambda với filter()

Hàm filter() sẽ lấy các tham số là một hàm và một list. Hàm được gọi với tất cả các mục trong list và list mới sẽ được trả về, chứa các mục mà hàm đánh giá là True. Ví dụ sau đây dùng hàm filter() để lọc ra các số chẵn trong list.

```
list_goc = [10, 9, 8, 7, 6, 1, 2, 3, 4, 5]\n\nlist_moi = list(filter(lambda a: (a%2 == 0) , list_goc))\n\n# Kết quả: [10, 8, 6, 2, 4]
```

6.2.3 Hàm Lambda với map()

Hàm map() cũng lấy các tham số là một hàm và một list. Hàm được gọi với tất cả các mục trong list và list mới được trả về chứa các mục được hàm trả về tương ứng cho mỗi mục.

```
list_goc = [10, 9, 8, 7, 6, 1, 2, 3, 4, 5]\n\nlist_moi = list(map(lambda a: a*2 , list_goc))\n\n# Kết quả: [20, 18, 16, 14, 12, 2, 4, 6, 8, 10]
```

Bài 7: Module và làm việc với thư viện

7.1 Giới thiệu module

Khi xây dựng một chương trình lớn, thông thường các chức năng sẽ được chia nhỏ thành các module để việc phát triển và bảo trì được dễ dàng.

Các module tham chiếu đến một tệp chứa các câu lệnh và định nghĩa trong Python. Một tệp chứa mã Python, ví dụ: example.py được gọi là module và tên module của nó sẽ là example. Bạn có thể sử dụng các mô-đun để chia nhỏ các chương trình lớn thành các tệp nhỏ để có thể quản lý và sắp xếp một cách. Hơn nữa, các mô-đun cung cấp khả năng tái sử dụng của mã. Chúng ta có thể xác định các chức năng được sử dụng nhiều nhất trong một mô-đun và thực thi chúng, thay vì phải sao chép các definitions của chúng vào các chương trình khác nhau.

7.2 Ví dụ cách tạo và sử dụng module

Ví dụ, chương trình main.py như sau:

```
# main.py

def add2num(a,b):
    return a + b

print(add2num(1,2))
```

Khi chương trình lớn, thay vì việc để tất cả các hàm trong 1 file, chúng ta tạo các file nhỏ để chứa các hàm xử lý, ví dụ file add.py với nội dung sau:

```
# add.py

def add2num(a,b):
    return a + b
```

Trong file main.py, chúng ta sử dụng lệnh import đến file add.py để gọi các hàm:

```
# main.py
import add

print(add.add2num(1,2))
```

Ngoài việc import toàn bộ module như trên, có thể import từng hàm trong module với lệnh :

```
from <module> import <functions>
```

Ví dụ:

```
# main.py
from add import add2num

print(add2num(1,2))
```

Ngoài ra, các biến khai báo trong module cũng có thể được import tương tự như các hàm.

Ví dụ :

```
# const.py
PI = 3.141592654

# main.py
from const import PI
print(PI/2)
```

Bạn có thể đổi tên module bạn mới import vào bằng lệnh as.

```
import mymodule as mx

from package import mymodule as mx
```

Bạn cũng có thể liệt kê các hàm trong module bạn mới import vào bằng lệnh dir().

```
import platform

x = dir(platform)
print(x)
```

Liệt kê các hàm có trong module: dir(module_name)

```
dir(example)
['__builtins__',
 '__cached__',
 '__doc__']
```

```
'__file__',  
'__initializing__',  
'__loader__',  
'__name__',  
'__package__',  
'add']
```

7.3 Package

Cài đặt các package: pip install <package_name>

Gỡ cài đặt package: pip uninstall <package_name>

Liệt kê các pakage đã cài: pip list

Bài 8: Xử lý ngoại lệ

8.1 Giới thiệu

Ngoại lệ (Exception) là lỗi xảy ra trong quá trình thực thi một chương trình. Khi nó xảy ra, Python tạo ra một exception để xử lý vẫn đề đó tránh cho ứng dụng của bạn bị crash.

Ngoại lệ có thể là bất kỳ điều kiện bất thường nào trong chương trình phá vỡ luồng thực thi chương trình đó. Bất cứ khi nào một ngoại lệ xuất hiện, chương trình sẽ ngừng thực thi, chuyển qua quá trình gọi và in ra lỗi đến khi nó được xử lý.

Ngoại lệ là các lỗi có thể phát sinh trong quá trình chạy chương trình. Ví dụ:

- Truy nhập biến không tồn tại
- Mở file không tồn tại
- Chuyển đổi một xâu không hợp lệ sang dạng số
- Truy nhập vượt quá chỉ số của List
- Truy nhập key không tồn tại trong Dictionary
- ...

Khi một lỗi xảy ra, nếu không được xử lý, Python sẽ đưa ra thông báo lỗi và dừng chương trình.

Ví dụ:

```
print(int('a'))
```

Chương trình trên khi chạy sẽ đưa ra thông báo:

```
ValueError: invalid literal for int() with base 10: 'a'
```

Cách xử lý ngoại lệ trong Python :

```
try:  
    # do something  
except:  
    print("Exception occurred")
```

Khi ngoại lệ xảy ra, chương trình sẽ chạy vào khối lệnh bên dưới dòng except:, sau đó đoạn chương trình tiếp theo khối try/except vẫn tiếp tục được thực hiện.

Để xử lý riêng với từng loại ngoại lệ, có thể chỉ định rõ loại của ngoại lệ trong khai báo except:

```
try:  
    # do something  
except ErrorType1:  
    print("Error type 1")  
except ErrorType2:  
    print("Error type 2")  
except:  
    print("Other exception")
```

Ví dụ:

```
try:  
    x = int('a')  
except ValueError:  
    print("Invalid input number")  
except:  
    print("Other exception")
```

Trong trường hợp muốn thực hiện một hành động sau khi kết thúc một công việc bất chấp ngoại lệ có xảy ra trong quá trình thực hiện công việc đó hay không, có thể dùng cấu trúc:

```
try:  
    # do something  
except:  
    print("Exception occurred")  
finally:  
    print('After try/except finished')
```

8.2 Các exception có sẵn trong Python.

Dưới đây là danh sách các exception mặc định trong Python.

Exception Name	Chú Thích
Exception	Đây là lớp cơ sở cho tất cả các exception, nó sẽ xuất hiện khi có bất cứ một lỗi nào xảy ra.
StopIteration	Xuất hiện khi phương thức next() của iterator không trả đến một đối tượng nào.
SystemExit	Xuất hiện khi dùng phương thức sys.exit()
StandardError	Lớp cơ sở cho tất cả các exception.
ArithmetricError	Xuất hiện khi có lỗi tính toán giữa các số với nhau
OverflowError	Xuất hiện khi thực hiện tính toán và giá trị của nó vượt quá ngưỡng giới hạn cho phép của kiểu dữ liệu.
FloatingPointError	Xuất hiện khi tính toán float thất bại.
ZeroDivisionError	Xuất hiện khi thực hiện phép chia cho 0.
AssertionError	Xuất hiện trong trường hợp lệnh assert thất bại.
AttributeError	Xuất hiện khi không tồn tại thuộc tính này, hoặc thiếu tham số truyền vào nó.
EOFError	Xuất hiện khi không có dữ liệu từ hàm input() hoặc cuối file.
ImportError	Xuất hiện khi lệnh import thất bại.
KeyboardInterrupt	Xuất hiện khi ngắt trình biên dịch.
LookupError	Lớp cơ sở cho tất cả các lỗi về lookup.
IndexError	Xuất hiện khi index không tồn tại trong list, string,...
KeyError	Xuất hiện khi key không tồn tại trong dictionary.
NameError	Xuất hiện khi một biến không tồn tại trong phạm vi bạn gọi nó.
EnvironmentError	Xuất hiện khi có bất kỳ một lỗi nào ngoài phạm vi của Python.
IOError	Xuất hiện khi xử dụng input/ output thất bại, hoặc mở file không thành công.

OSError	Xuất hiện khi có lỗi từ hệ điều hành.
SyntaxError	Xuất hiện khi chương trình có lỗi cú pháp.
IndentationError	Xuất hiện khi bạn thụt dòng không đúng.
SystemError	Xuất hiện khi trình biên dịch có vấn đề nhưng mà nó lại không tự động exit.
SystemExit	Xuất hiện khi trình biên dịch được thoát bởi <code>sys.exit()</code> .
TypeError	Xuất hiện khi thực thi toán tử hoặc hàm mà kiểu dữ liệu bị sai so với kiểu dữ liệu đã định nghĩa ban đầu.
ValueError	Xuất hiện khi chúng ta build 1 function mà kiểu dữ liệu đúng nhưng khi chúng ta thiết lập ở tham số là khác so với khi truyền vào.
RuntimeError	Xuất hiện khi lỗi được sinh ra không thuộc một danh mục nào.
NotImplementedError	Xuất hiện khi một phương thức trừu tượng cần được thực hiện trong lớp kế thừa chứ không phải là lớp thực thi
UnboundLocalError	Xuất hiện khi chúng ta cố tình truy cập vào một biến trong hàm hoặc phương thức, nhưng không thiết lập giá trị cho nó.

8.3 Custom Exception trong Python

Python cho phép bạn sử dụng `raise` 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
try:
```

```
raise Networkerror("Bad hostname")
except Networkerror,e:
    print e.args
```

Bài 9: File I/O

9.1 Làm việc với tập tin (text/nhị phân)

9.1.1 Mở file

Lệnh mở file trong Python:

```
f = open(<file_name>, <mode>, encoding=<encoding>)
```

Trong đó :

<file_name> : tên file cần mở
<mode> : Chế độ mở file, các giá trị thường dùng:
 'r' : Mở file để đọc, đây là giá trị mặc định
 'rb' : Mở file để đọc dưới dạng nhị phân (binary)
 >w' : Tạo file mới để ghi
 'wb' : Tạo file mới để ghi dưới dạng nhị phân (binary)
 'a' : Mở file để ghi thêm vào cuối (append)
 'ab' : Mở file để ghi thêm vào cuối dưới dạng nhị phân
 'x' : Tạo mới file, trả về lỗi nếu file đã có
<encoding>: Chế độ encoding với file văn bản
 None : giá trị mặc định, tùy thuộc từng hệ thống
 'utf-8' : mã hóa Unicode dạng UTF-8

9.1.2 Đọc file

Đọc toàn bộ nội dung file:

```
data = f.read()
```

Nếu file được mở theo chế độ văn bản ('r'), kết quả đọc file là một biến string. Nếu file được mở theo chế độ nhị phân ('rb'), kết quả đọc file là một biến có kiểu dữ liệu bytes.

Đọc từng dòng của file: line = f.readline().

Đọc toàn bộ các dòng của file (với file văn bản):

Nếu mở file theo chế độ văn bản ('r') có thể đọc nội dung file theo từng dòng với lệnh :

```
lines = f.readlines()
```

Kết quả trả về là một List, mỗi phần tử là một string chứa nội dung các dòng của file

Đọc từng dòng của file:

Nếu mở file theo chế độ văn bản ('r') có thể đọc từng dòng của file với cấu trúc:

```
for line in f:  
    # process line
```

9.1.3 Ghi file:

Với file văn bản, dữ liệu ghi vào file cần có kiểu string

```
f.write(<string>)
```

Ví dụ:

```
f = open('test.txt', 'w')  
f.write('hello')  
f.close()
```

Với file nhị phân, dữ liệu ghi vào file cần có kiểu bytes

```
f.write(<bytes>)
```

Ví dụ:

```
f = open('test.dat', 'wb')  
f.write('hello'.encode())  
f.close()
```

9.1.4 Đóng file

```
f.close()
```

9.1.5 Truy xuất file với cấu trúc 'with'

Sau khi mở file với lệnh open, phải đóng file với lệnh close để tránh bị thiếu nội dung file khi kết thúc chương trình. Để tránh việc quên đóng file sau khi mở, có thể dùng cấu trúc **with**:

```
with open(<file_name>, <mode>, <encoding>) as f:  
    # Process file
```

Với cấu trúc **with**, file sẽ tự động được đóng khi chương trình chạy xong khỏi lệnh bên trong **with**.

Ví dụ:

```
with open('test.txt', 'w') as f:  
    f.write('Line1\n')  
    f.write('Line2\n')
```

9.1.6 Một số ví dụ minh họa

Ví dụ 1:

Đọc vào một file văn bản, tạo ra một file văn bản mới chứa các dòng của file nguồn, bỏ đi các dòng trống.

File input.txt:

```
Một năm có 365 hoặc 366 ngày  
Năm thường có 365 ngày
```

```
Năm nhuận có 366 ngày
```

File output.txt:

```
Một năm có 365 hoặc 366 ngày  
Năm thường có 365 ngày  
Năm nhuận có 366 ngày
```

```
infile = open('input.txt', encoding='utf-8')  
outfile = open('output.txt', 'w', encoding='utf-8')
```

```
for line in infile:  
    if line.strip() != "":  
        outfile.write(line)  
  
infile.close()  
outfile.close()
```

Ví dụ 2:

Một file csv chứa bảng điểm một môn học của các học sinh một lớp học. Mỗi dòng của file là thông tin điểm của một học sinh bao gồm : họ tên, điểm hệ số 1, điểm hệ số 2, điểm hệ số 3, các giá trị này ngăn cách nhau bởi dấu phẩy. Viết chương trình để đọc file csv đầu vào và tạo ra một file csv mới có thêm cột điểm trung bình.

File input.csv

```
Nguyễn Văn An, 8, 7, 8  
Nguyễn Văn Bình, 6, 6, 8
```

Nguyễn Thị Chi, 8, 8, 9
Lê Văn Cường, 8, 7, 9
Phạm Thu Trang, 7, 8, 8

File output.csv

Nguyễn Văn An, 8, 7, 8, 7.7
Nguyễn Văn Bình, 6, 6, 8, 7.0
Nguyễn Thị Chi, 8, 8, 9, 8.5
Lê Văn Cường, 8, 7, 9, 8.2
Phạm Thu Trang, 7, 8, 8, 7.8

```
fi = open('input.csv', encoding='utf-8')
fo = open('output.csv', 'w', encoding='utf-8')
```

```
for line in fi:
    hoten, diemhs1, diemhs2, diemhs3 = line.split(',')
    diemhs1 = int(diemhs1)
    diemhs2 = int(diemhs2)
    diemhs3 = int(diemhs3)
    diem_tb = (diemhs1 + 2*diemhs2 + 3*diemhs3) / 6
    fo.write(f'{hoten}, {diemhs1}, {diemhs2}, {diemhs3}, {round(diem_tb, 1)}\n')

fi.close()
fo.close()
```

9.1.7 Một số hàm xử lý khác

Xóa một file:

```
import os
os.remove("demofile.txt")
```

Kiểm tra file tồn tại hay không:

```
os.path.exists("demofile.txt")
```

Xóa thư mục:

```
os.rmdir("myfolder")
```

9.2 Làm việc với thư mục

Bài 10: Lập trình hướng đối tượng

- Lớp và đối tượng (class & object)
- Ké thừa (Inheritance): Overriding method, Overloading method, Data Hiding
- Lớp trừu tượng (Abstract base class – ABC)

Classes, Attributes, and Inheritance

Function	Description
classmethod()	Returns a class method for a function
delattr()	Deletes an attribute from an object
getattr()	Returns the value of a named attribute of an object
hasattr()	Returns True if an object has a given attribute
isinstance()	Determines whether an object is an instance of a given class
issubclass()	Determines whether a class is a subclass of a given class
property()	Returns a property value of a class
setattr()	Sets the value of a named attribute of an object
super()	Returns a proxy object that delegates method calls to a parent or sibling class

Bài 11: Làm việc với dữ liệu JSON/CSV

11.1 Làm việc với JSON

11.1.1 Giới thiệu

JSON (JavaScript Object Notation): là một định dạng dữ liệu rất phổ biến, được dùng để lưu trữ và thể hiện các dữ liệu có cấu trúc, dạng các cặp key - value. JSON là định dạng dữ liệu phổ biến được sử dụng để truyền và nhận dữ liệu giữa ứng dụng web và web server.

Ví dụ dữ liệu Python:

```
{  
    "name": "HIEN Luong",  
    "age": 39,  
    "degree": [  
        "mathematics",  
        "computer science"  
    ],  
    "retired": false,  
    "carrer": {  
        "dek": {  
            "from": 2019,  
            "role": "software engineer"  
        },  
        "nhatnghe": {  
            "from": 2011,  
            "role": "it trainer"  
        },  
        "hcmue": {  
            "from": 2007,  
            "role": "lecturer"  
        }  
    }  
}
```

11.1.2 Làm việc với JSON bằng Python

Python cho phép thao tác với JSON dưới dạng chuỗi hoặc lưu đối tượng JSON vào trong file.

Để có thể làm việc với JSON, ta cần [import module json](#). Ta cần import module trước khi gọi các hàm để thao tác với json.

Để có thể parse một JSON string, ta gọi method `json.loads()`. Phương thức này sẽ trả về một đối tượng dictionary chứa dữ liệu được chứa trong JSON string.

Ví dụ:

```
# Import json module
import json

# Khai báo một JSON string
listfruits = '{"orange":"Qua cam", "strawberry":"Day tay", \
    "grape":"Nho", "durian":"Sau rieng"}'

# Đọc JSON String, method này trả về một Dictionary
mylist = json.loads(listfruits)

# In ra thông tin của Dictionary
print(mylist)

# In ra một giá trị trong Dictionary
print(mylist['durian'])
```

Để đọc một file có chứa JSON object, ta gọi method `json.load()`.

Ví dụ:

```
with open('path_to_file/person.json') as f:
    data = json.load(f)
# Output: {'name': 'Bob', 'languages': ['English', 'French']}
```

Để chuyển đổi từ một dictionary thành một JSON string, ta gọi method `json.dumps()`.

Ví dụ:

```
person_dict = {'name': 'Bob',
               'age': 12,
               'children': None
}
person_json = json.dumps(person_dict)

# Output: {"name": "Bob", "age": 12, "children": null}
```

Bạn có thể chuyển đổi các kiểu dữ liệu sau ra kiểu JSON string:

Python	JSON
<code>dict</code>	<code>object</code>
<code>list</code> , <code>tuple</code>	<code>array</code>
<code>str</code>	<code>string</code>
<code>int</code> , <code>float</code> , <code>int</code>	<code>number</code>
<code>True</code>	<code>true</code>
<code>False</code>	<code>false</code>
<code>None</code>	<code>null</code>

Để ghi dữ liệu JSON ra file trong Python, ta sử dụng method `json.dump()`.

Ví dụ:

```
import json

person_dict = {"name": "Bob",
"languages": ["English", "French"],
"married": True,
"age": 32
}

with open('person.txt', 'w') as json_file:
    json.dump(person_dict, json_file)
```

kết quả file person.txt được tạo ra với nội dung như sau:

```
{"name": "Bob", "languages": ["English", "French"], "married": true, "age": 32}
```

Phương thức `json.dumps()` cung cấp các tham số để cho phép định dạng kết quả (thụt lè) hoặc sắp xếp kết quả xử lý.

```
import json

person_string = '{"name": "Bob", "languages": "English", "numbers": [2, 1.6, null]}'

# Getting dictionary
person_dict = json.loads(person_string)

# Pretty Printing JSON string back
print(json.dumps(person_dict, indent = 4, sort_keys=True))
```

kết quả chạy:

```
{
    "languages": "English",
    "name": "Bob",
    "numbers": [
        2,
        1.6,
        null
    ]
}
```

- Cấu trúc JSON
- Đọc, xử lý dữ liệu JSON từ Internet
- Mở, đọc, ghi dữ liệu JSON

11.2 Làm việc với file CSV

11.2.1 Module csv trong python

CSV (Comma Separated Values) hay giá trị được phân tách bằng dấu phẩy là một **định dạng tệp** đơn giản được sử dụng để lưu trữ dữ liệu dạng bảng, chẳng hạn như bảng tính hoặc cơ sở dữ liệu. Tệp CSV lưu trữ dữ liệu dạng bảng (số và văn bản) ở dạng văn bản thuần túy. Mỗi dòng của tệp là một bản ghi dữ liệu. Mỗi bản ghi bao gồm một hoặc nhiều trường, được phân tách bằng dấu phẩy. Việc sử dụng dấu phẩy làm dấu phân tách trường là nguồn gốc của tên cho định dạng tệp này.

Đối với các tệp CSV hoạt động trong python, có một mô-đun có sẵn gọi là **csv**.

11.2.2 Cách đọc csv

```
# importing csv module
import csv
# csv file name
filename = "aapl.csv"
# initializing the titles and rows list
fields = []
rows = []
# reading csv file
with open(filename, 'r') as csvfile:
    # creating a csv reader object
    csvreader = csv.reader(csvfile)
    # extracting field names through first row
    fields = next(csvreader)
    # extracting each data row one by one
    for row in csvreader:
        rows.append(row)
    # get total number of rows
    print("Total no. of rows: %d" % (csvreader.line_num))
# printing the field names
print('Field names are:' + ', '.join(field for field in fields))
# printing first 5 rows
print('\nFirst 5 rows are:\n')
for row in rows[:5]:
    # parsing each column of a row
    for col in row:
        print("%10s" % col),
    print('\n')
```

Output

```
Total no. of rows: 252
Field names are:Date, Open, High, Low, Close, Volume

First 5 rows are:

 7-Dec-16      109.26      111.19      109.16      111.03      29998719
 6-Dec-16      109.50      110.36      109.19      109.95      26195462
 5-Dec-16      110.00      110.03      108.25      109.11      34324540
 2-Dec-16      109.17      110.09      108.85      109.90      26527997
 1-Dec-16      110.36      110.94      109.03      109.49      37086862
```

Ví dụ về File CSV trong Python

11.2.3 Cấu tạo 1 file csv

Một file CSV gồm 3 phần:

- Phần đầu tiên: tương ứng với cột đầu tiên trong bảng tính, biểu thị tên của các cột, mỗi cột được ngăn cách với nhau bởi dấu phẩy.
- Phần thứ 2: tương ứng với cột cuối cùng trong bảng tính
- Phần thứ 3: bao gồm các dòng có cấu trúc tương đương nhau, tương ứng với nội dung của các cột giá trị trong bảng tính. Lưu ý, mỗi dòng của văn bản là một dòng giá trị khác nhau trên bảng tính.

11.2.4 Sử dụng csv trong python

Có nhiều cách khác nhau để đọc tệp CSV sử dụng mô-đun csv hoặc thư viện [pandas trong Python](#).

- **Mô-đun csv:** Mô-đun CSV là một trong những mô-đun bằng Python cung cấp các lớp để đọc và ghi thông tin dạng bảng ở định dạng tệp CSV.
- **Thư viện pandas:** Thư viện pandas là một trong những thư viện Python mã nguồn mở cung cấp các cấu trúc dữ liệu thuận tiện, hiệu suất cao và các công cụ và kỹ thuật phân tích dữ liệu cho lập trình Python.

Tệp CSV dưới đây có tên 'Giants.csv':

Organization	CEO	Established
Alphabet	Sundar Pichai	02-Oct-15
Microsoft	Satya Nadella	04-Apr-75
Aamzon	Jeff Bezos	05-Jul-94

Ví dụ về đọc file CSV trong Python
>>> Tham khảo: [Khóa học lập trình Python](#)

11.2.5 SỬ DỤNG CSV.READER ()

Lúc đầu, tệp CSV được mở bằng phương thức `open()` ở chế độ 'r' (chỉ định chế độ đọc trong khi mở tệp) trả về đối tượng tệp sau đó nó được đọc bằng cách sử dụng phương thức `reader()` của mô-đun CSV trả về đối tượng trình đọc lặp lại trong suốt các dòng trong tài liệu CSV được chỉ định.

Lưu ý: Các từ khóa 'with' được sử dụng cùng với các phương pháp `open()` vì nó đơn giản hóa việc xử lý ngoại lệ và tự động đóng các tập tin CSV.

```
import csv
# opening the CSV file
with open('Giants.csv', mode ='r')as file:
    # reading the CSV file
    csvFile = csv.reader(file)
    # displaying the contents of the CSV file
    for lines in csvFile:
        print(lines)
Output nhận được:
['Organization', 'CEO', 'Established']
```

```
['Alphabet', 'Sundar Pichai', '02-Oct-15']
['Microsoft', 'Satya Nadella', '04-Apr-75']
['Aamzon', 'Jeff Bezos', '05-Jul-94']
```

Trong chương trình trên, phương thức reader () được sử dụng để đọc tệp Giants.csv ánh xạ dữ liệu thành danh sách.

11.2.6 SỬ DỤNG LỚP CSV.DICTREADER ()

Tương tự như phương pháp trước, tệp CSV lần đầu tiên được mở bằng phương thức open() này sau đó được đọc bằng cách sử dụng DictReader, lớp mô-đun csv hoạt động giống như một trình đọc thông thường nhưng ánh xạ thông tin trong tệp CSV vào từ điển. Dòng đầu tiên của tệp bao gồm các khóa từ điển.

```
import csv
# opening the CSV file
with open('Giants.csv', mode ='r') as file:
    # reading the CSV file
    csvFile = csv.DictReader(file)
    # displaying the contents of the CSV file
    for lines in csvFile:
        print(lines)
```

Output

```
OrderedDict([('Organiztion', 'Alphabet'), ('CEO', 'Sundar Pichai'),
('Established', '02-Oct-15')])
OrderedDict([('Organiztion', 'Microsoft'), ('CEO', 'Satya Nadella'),
('Established', '04-Apr-75')])
OrderedDict([('Organiztion', 'Aamzon'), ('CEO', 'Jeff Bezos'),
('Established', '05-Jul-94')])
```

Sử dụng phương thức pandas.read_csv (): Rất dễ dàng và đơn giản để đọc tệp CSV bằng các hàm thư viện pandas. Ở đây, phương thức read_csv() trong thư viện Pandas được sử dụng để đọc dữ liệu từ tệp CSV.

```
import pandas
# reading the CSV file
csvFile = pandas.read_csv('Giants.csv')
# displaying the contents of the CSV file
print(csvFile)
Output
```

	Organiztion	CEO	Established
0	Alphabet	Sundar Pichai	02-Oct-15
1	Microsoft	Satya Nadella	04-Apr-75
2	Aamzon	Jeff Bezos	05-Jul-94

11.3 Thư viện request, urllib3

11.3.1 Thư viện request

Thư viện requests là một thư viện HTTP đơn giản dành cho python. Về cơ bản nó sử dụng để gửi yêu cầu HTTP qua các dịch vụ web API. Thư viện requests không đi kèm với thư viện tiêu chuẩn của python, để cài đặt nó bằng lệnh: pip install requests

Cú pháp: `requests .methodname(params)`

Các loại yêu cầu thường dùng: POST, GET, PUT, DELETE.

Method	Description
<code>delete(url, args)</code>	Sends a DELETE request to the specified url
<code>get(url, params, args)</code>	Sends a GET request to the specified url
<code>head(url, args)</code>	Sends a HEAD request to the specified url
<code>patch(url, data, args)</code>	Sends a PATCH request to the specified url
<code>post(url, data, json, args)</code>	Sends a POST request to the specified url
<code>put(url, data, args)</code>	Sends a PUT request to the specified url
<code>request(method, url, args)</code>	Sends a request of the specified method to the specified url

Ví dụ gửi request GET đến trang nhatnghe.com

```
import requests  
  
res = requests.get('https://nhatnghe.com')
```

Xử lý kết quả trả về: res.text, res.json(), status_code

Ví dụ gửi request POST:

```
data = {'username': 'test_user', 'password': 'test_pass'}  
  
res = requests.post('http://httpbin.org/post', json = data)
```

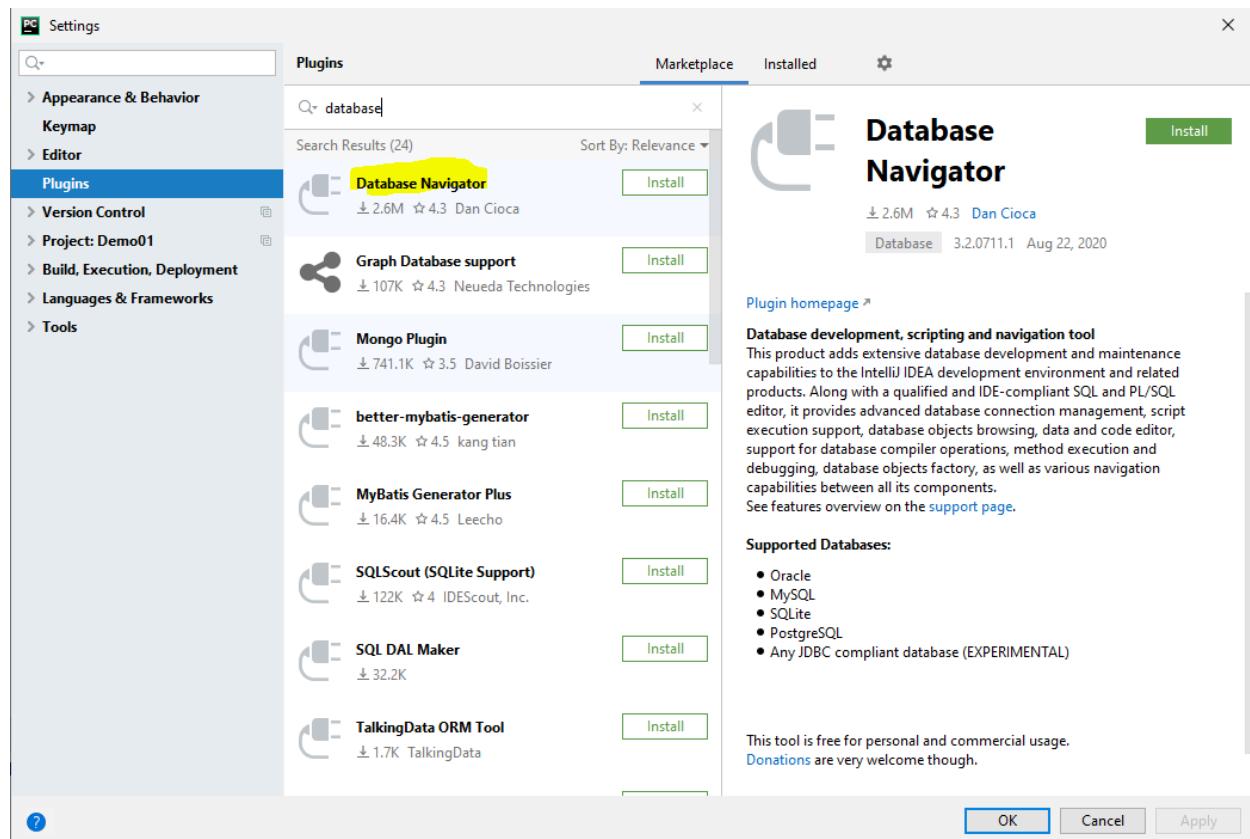
11.3.2 Thư viện urllib3

Bài 12: Làm việc với cơ sở dữ liệu

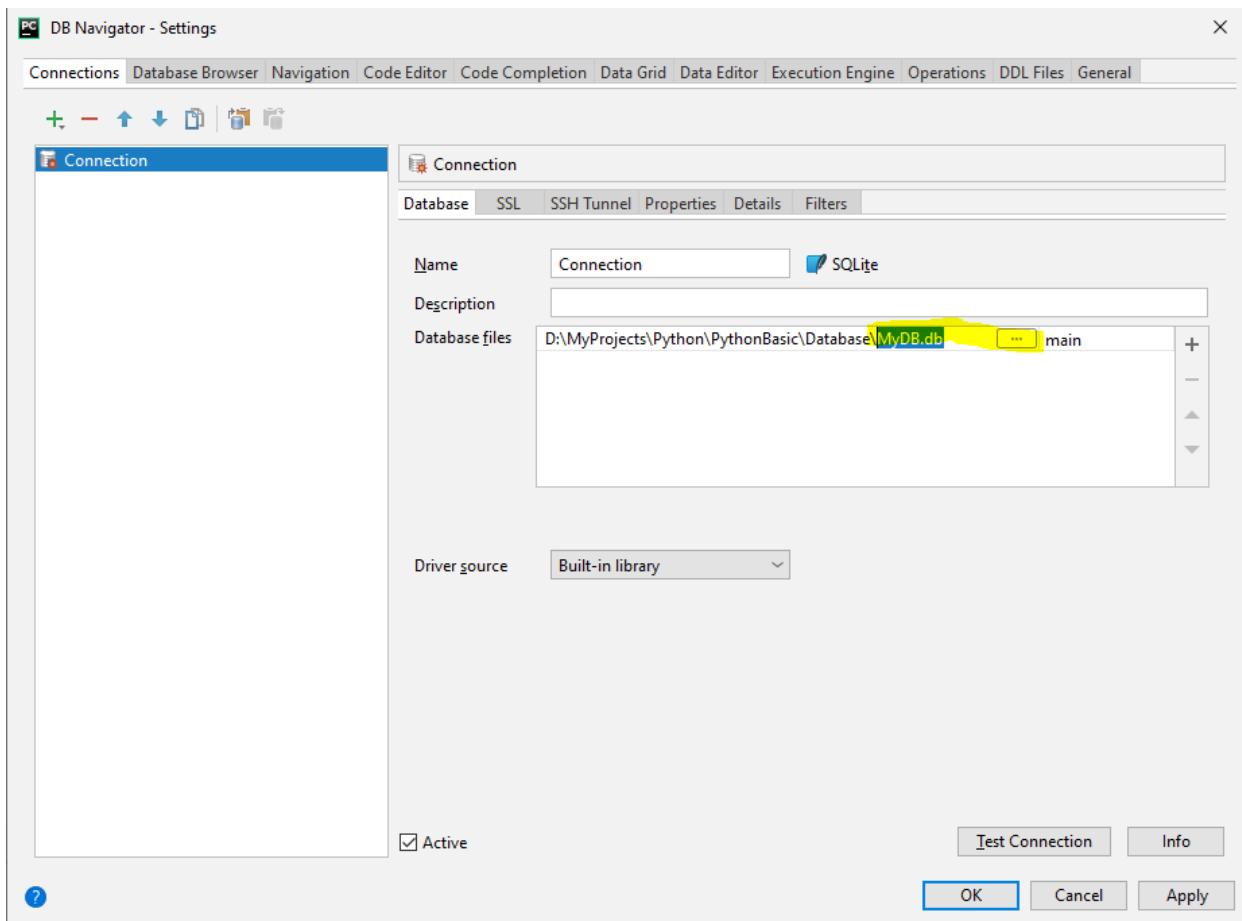
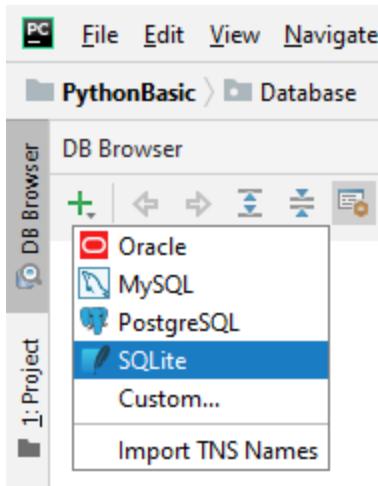
12.1 Giới thiệu hệ quản trị CSDL SQLite

Cài plugin cho PyCharm

Vào menu **File → Settings**, chọn Tab **Plugins**, tìm, chọn và cài đặt **Database Navigator**.

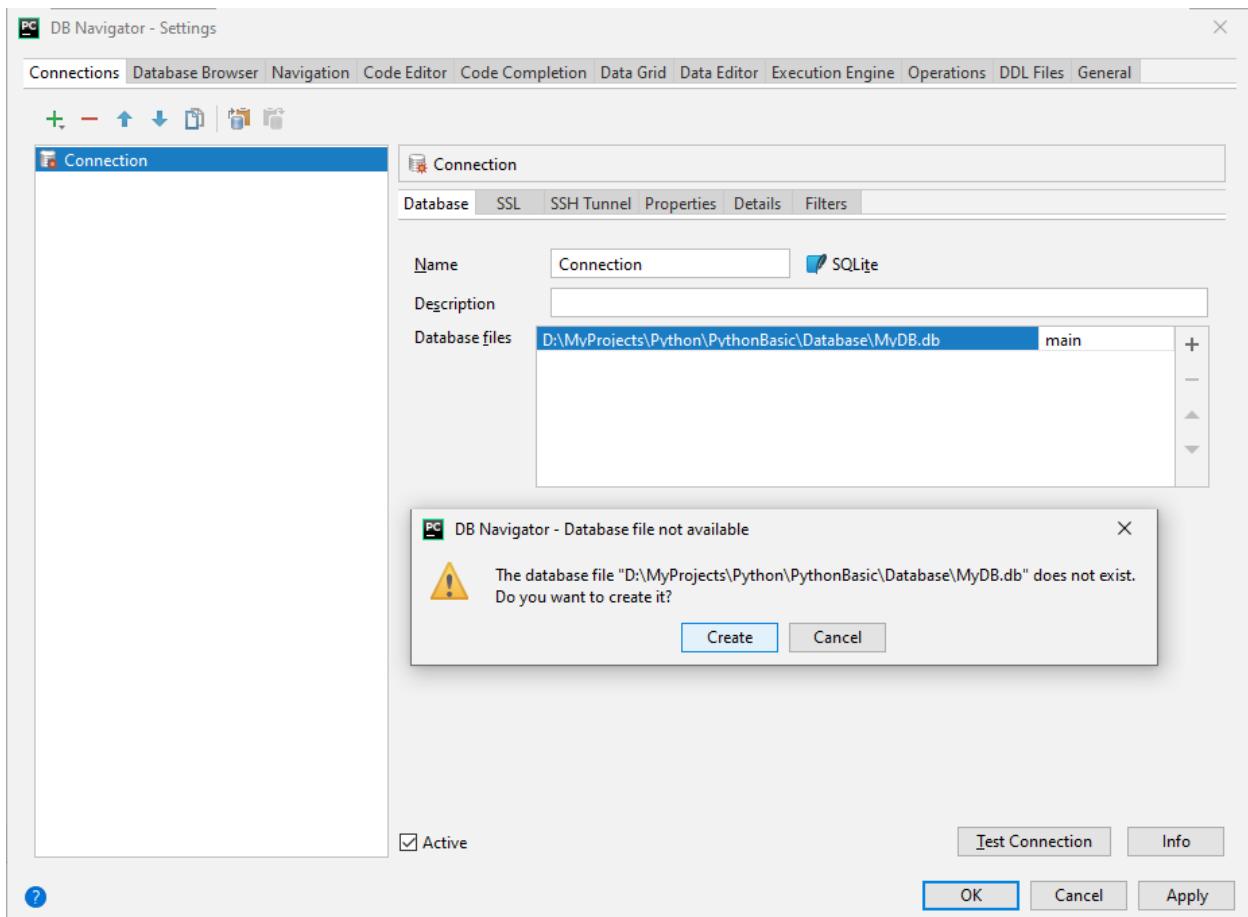


Vào menu DB Browser, chọn nút (+) để thêm kết nối, chọn Sqlite

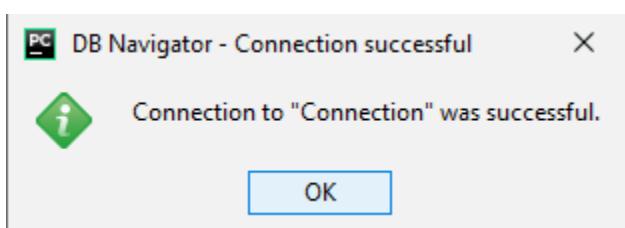


Đặt tên database file và chọn nơi lưu trữ.

Thử bấm Test Connection, màn hình sau xuất hiện:



Bấm Create để tạo mới.



Bấm OK 2 lần để trở về ứng dụng

Tạo cấu trúc CSDL bằng câu lệnh SQL

CREATE TABLE [Departments] (

[DepartmentId] **INTEGER NOT NULL PRIMARY KEY**,

[DepartmentName] **NVARCHAR(50) NOT NULL**

);

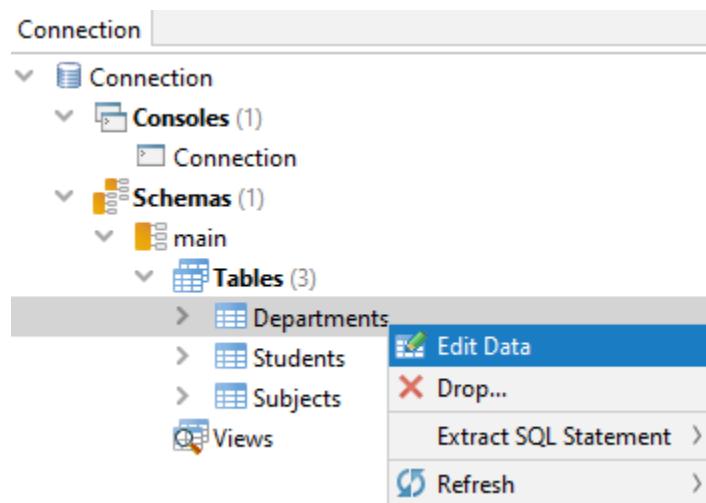
```

CREATE TABLE [Students] (
    [StudentId] INTEGER PRIMARY KEY NOT NULL,
    [StudentName] NVARCHAR(50) NOT NULL,
    [DepartmentId] INTEGER NULL,
    [DateOfBirth] DATE NULL
);

```

Khi chạy SQL Script. Chú ý commit

Để nhập liệu trực tiếp, chuột phải trên table, chọn **Edit Data**

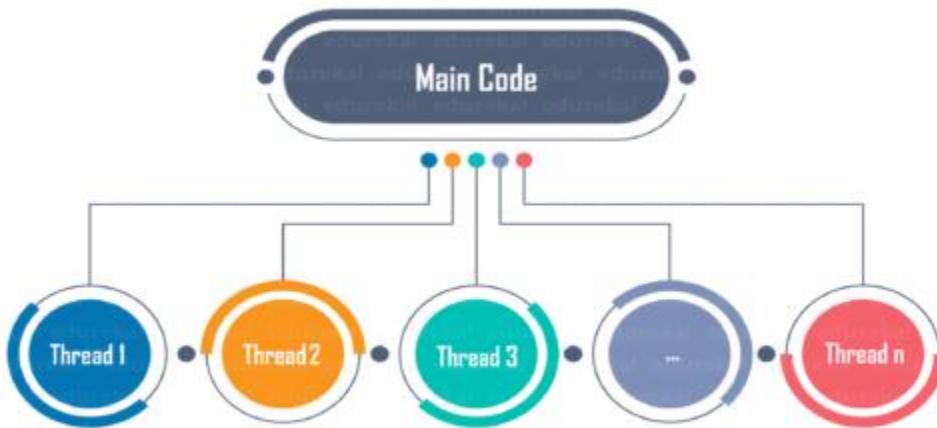


- Python sqlite3 module APIs
- Làm việc với CSDL SQLite: xây dựng ứng dụng CRUD

12.2 Làm việc với MySQL

Bài 13: Thread - multi thread

- Khái niệm chương trình, tiến trình, chương trình đơn luồng (single thread) và chương trình đa luồng (multi thread)
- Thread
- Multithreaded Priority Queue



13.1 Cách hoạt động của đa luồng trong python

Chạy một số luồng tương tự như chạy một số chương trình khác nhau một cách đồng thời, nhưng với những lợi ích sau:

- Nhiều luồng trong một quy trình chia sẻ cùng một không gian dữ liệu với luồng chính và do đó có thể chia sẻ thông tin hoặc giao tiếp với nhau dễ dàng hơn nếu chúng là các quy trình riêng biệt.

- Các luồng đôi khi được gọi là các quy trình nhẹ và chúng không yêu cầu nhiều bộ nhớ; chúng rẻ hơn các quy trình.

Một chuỗi có một phần mở đầu, một chuỗi thực thi và một phần kết luận. Nó có một con trỏ hướng dẫn theo dõi vị trí mà nó hiện đang chạy trong ngữ cảnh của nó.

- Nó có thể được làm trống trước

- Nó có thể tạm thời được đặt ở trạng thái chờ (còn được gọi là ngủ) trong khi các luồng khác đang chạy - điều này được gọi là năng suất.

13.2 Cách bắt đầu luồng mới trong Python

Để tạo ra một luồng khác, bạn cần gọi phương thức có sẵn trong mô-đun luồng như dưới đây:

`thread.start_new_thread (function, args[, kwargs])`

Lệnh gọi phương thức cho phép tạo các luồng mới trong ở Linux và Window một cách nhanh chóng và hiệu quả

Trong phương thức trên, args là một loạt các đối số, sử dụng một bộ giá trị trống để gọi hàm mà không chuyển bất kỳ đối số nào. kwargs là một từ điển tùy chọn cả các đối số từ khóa.

Ví dụ:

```
#!/usr/bin/python
import thread
import time
# Define a function for the thread
def print_time( threadName, delay):
    count = 0
    while count < 5:
        time.sleep(delay)
        count += 1
        print "%s: %s" % ( threadName, time.ctime(time.time()) )
# Create two threads as follows
try:
    thread.start_new_thread( print_time, ("Thread-1", 2, ) )
    thread.start_new_thread( print_time, ("Thread-2", 4, ) )
except:
    print "Error: unable to start thread"
while 1:
    pass
```

Khi đoạn mã trên được thực thi, kết quả ta nhận được như sau:

```
Thread-1: Thu Jan 22 15:42:17 2009
Thread-1: Thu Jan 22 15:42:19 2009
Thread-2: Thu Jan 22 15:42:19 2009
Thread-1: Thu Jan 22 15:42:21 2009
Thread-2: Thu Jan 22 15:42:23 2009
Thread-1: Thu Jan 22 15:42:23 2009
Thread-1: Thu Jan 22 15:42:25 2009
Thread-2: Thu Jan 22 15:42:27 2009
Thread-2: Thu Jan 22 15:42:31 2009
Thread-2: Thu Jan 22 15:42:35 2009
```

13.3 Các luồng mô đun trong Python

Mô đun luồng mới được dùng trong Python 2.4 cung cấp hỗ trợ cấp cao, mạnh mẽ hơn nhiều cho các luồng so với mô đun luồng được thảo luận trong phần trước

Mô đun luồng cho thấy tất cả các phương pháp của mô-đun thread và cung cấp một số phương thức như sau:

- `threading.activeCount()` – Trả về số đối tượng luồng đang hoạt động.
- `currentThread()` – Trả về số đối tượng luồng trong điều khiển luồng của người gọi.
- `threading.enumerate()` – Trả về danh sách tất cả các đối tượng luồng hiện đang hoạt động.

Ngoài các phương pháp, mô-đun threading có lớp Thread thực hiện luồng. Các phương thức được cung cấp bởi lớp *Thread* như sau:

- run () - Phương thức run () là điểm vào của một luồng.
- start () - Phương thức start () bắt đầu một luồng bằng cách gọi phương thức run.
- join ([time]) - đợi luồng kết thúc.
- Alive () - Phương thức is Alive () kiểm tra xem một luồng có còn đang thực thi hay không.
- getName () - Phương thức getName () trả về tên của một luồng.
- setName () - Phương thức setName () đặt tên của một luồng.

13.4 Tạo luồng sử dụng threading module

Để thực hiện một luồng mới sử dụng mô đun threading, bạn cần phải làm theo những điều sau:

- Xác định một lớp phụ của lớp Thread
- Ghi đè phương thức `__init__ (self [, args])` để thêm các đối số bổ sung.
- Sau đó, ghi đè phương thức run (self [, args]) để triển khai những gì luồng sẽ làm khi bắt đầu.

Khi bạn đã tạo lớp con Thread mới, bạn có thể tạo một thể hiện của nó và sau đó bắt đầu một luồng mới bằng cách gọi `start ()`, đến lượt nó sẽ gọi `run ()`

13.5 Queue đa luồng ưu tiên trong python

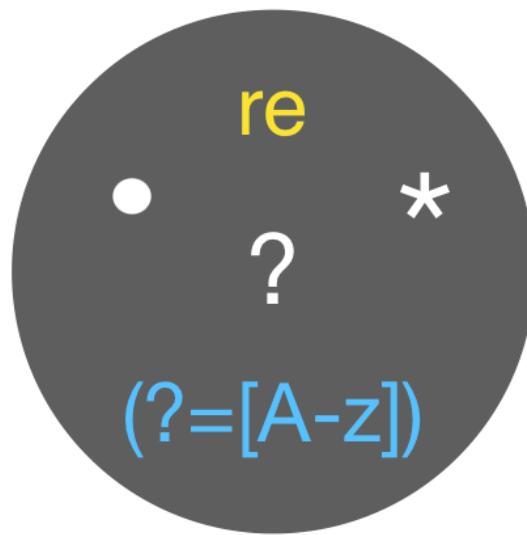
Các mô-đun *Queue* cho phép bạn tạo một đối tượng hàng đợi mới có thể tổ chức một số cụ thể của các mặt hàng. Có các phương pháp sau để kiểm soát Hàng đợi:

- get () - Hàm get () xóa và trả về một mục từ hàng đợi.
- put () - Đặt thêm mục vào hàng đợi.
- qsize () - qsize () trả về số lượng các mục hiện đang có trong hàng đợi.
- rỗng () - Giá trị rỗng () trả về giá trị True nếu hàng đợi trống; ngược lại, Sai.
- full () - full () trả về True nếu queue đầy; ngược lại, Sai

Bài 14: Regular Expression

- Regular Expression Pattern
- match/ search
- So sánh Matching và Searching
- Tìm kiếm và thay thế

Python
regex
Regular Expresiones



Regex - Regular expression hay còn gọi là biểu thức chính quy trong Python. Regex là một chuỗi miêu tả một bộ các chuỗi khác, theo những quy tắc cú pháp nhất định. Bạn cũng có thể gọi Regex là một ngôn ngữ. Và hầu như ngôn ngữ lập trình nào cũng hỗ trợ Regular expression.

Regular Expression trong Python được thể hiện qua module re, re Module cung cấp sự hỗ trợ đầy đủ các Regular Expression trong Python. Module này tạo Exception là re.error nếu xảy ra một lỗi trong khi biên dịch hoặc khi sử dụng một Regular Expression. Để sử dụng re việc đầu tiên bạn cần phải import module re vào chương trình, sử dụng cú pháp như sau:

```
import re
```

14.1 Các hàm regex trong python

Các hàm regex sau được sử dụng trong Python.

STT	Hàm	Mô tả
1	match	Hàm này khớp với mẫu regex trong chuỗi với cờ tùy chọn. Nó trả về true nếu một kết quả khớp được tìm thấy trong chuỗi nếu nó trả về false.

2	search	Hàm này trả về đối tượng khớp nếu có một kết quả khớp được tìm thấy trong chuỗi.
3	findall	Nó trả về một danh sách chứa tất cả các kết quả khớp của một mẫu trong chuỗi.
4	split	Trả về một danh sách trong đó chuỗi đã được phân chia theo mỗi kết quả khớp.
5	sub	Thay thế một hoặc nhiều kết quả khớp trong chuỗi.

14.2 Xây dựng regex trong python

Một **regex** trong Python có thể được xây dựng bởi các MetaCharacters, set hoặc ký tự đặc biệt.

Xây dựng bằng Metacharacter

Metacharacter	Mô tả	Ví dụ
[]	Nó đại diện cho một tập các ký tự.	"[a-z]"
\	Nó đại diện cho ký tự đặc biệt.	"\r"
.	Nó đại diện cho bất kỳ ký tự nào xuất hiện ở một số nơi cụ thể.	"Ja.v."
^	Nó đại diện cho mẫu có mặt ở đầu chuỗi.	"^Java"
\$	Nó đại diện cho mẫu có mặt ở cuối chuỗi.	"viettuts"
*	Nó đại diện cho không hoặc nhiều lần xuất hiện của một mẫu trong chuỗi.	"hello*"
+	Nó đại diện cho một hoặc nhiều lần xuất hiện của một mẫu trong chuỗi.	"hello+"
{}	Số lần xuất hiện đã chỉ định của một mẫu trong chuỗi.	"java{2}"
	Nó biểu diễn cho cái này hoặc cái kia (điều kiện or).	"python2 python3"
()	Nhóm các thành phần.	

14.3 Regex trong python - xây dựng bằng set

Một set là một nhóm các ký tự được đưa ra bên trong một cặp dấu ngoặc vuông. Nó đại diện cho ý nghĩa đặc biệt.

STT	Set	Mô tả

1	[arn]	Trả về một kết quả khớp nếu chuỗi chứa bất kỳ ký tự nào được chỉ định trong tập hợp.
2	[a-n]	Trả về một kết quả khớp nếu chuỗi chứa bất kỳ ký tự từ a đến n.
3	[^arn]	Trả về một kết quả khớp nếu chuỗi chứa các ký tự ngoại trừ a, r và n.
4	[0123]	Trả về một kết quả khớp nếu chuỗi chứa bất kỳ chữ số nào được chỉ định.
5	[0-9]	Trả về một kết quả khớp nếu chuỗi chứa bất kỳ chữ số nào trong khoảng từ 0 đến 9.
6	[0-5][0-9]	Trả về một kết quả khớp nếu chuỗi chứa bất kỳ chữ số nào trong khoảng từ 00 đến 59.
10	[a-zA-Z]	Trả về một kết quả khớp nếu chuỗi chứa bất kỳ bảng chữ cái nào (chữ thường hoặc chữ hoa).

14.4 Regex trong python - xây dựng bằng ký tự đặc biệt

Ký tự đặc biệt là các chuỗi có chứa \ theo sau là một trong các ký tự.

Ký tự	Mô tả
\A	Nó trả về một kết quả khớp nếu các ký tự được chỉ định có mặt ở đầu chuỗi.
\b	Nó trả về một kết quả khớp nếu các ký tự được chỉ định có mặt ở đầu hoặc cuối chuỗi.
\B	Nó trả về một kết quả khớp nếu các ký tự được chỉ định có mặt ở đầu chuỗi nhưng không ở cuối chuỗi.
\d	Nó trả về một kết quả khớp nếu chuỗi chứa các chữ số [0-9].
\D	Nó trả về một kết quả khớp nếu chuỗi không chứa các chữ số [0-9].
\s	Nó trả về một kết quả khớp nếu chuỗi chứa bất kỳ ký tự khoảng trắng nào.
\S	Nó trả về một kết quả khớp nếu chuỗi không chứa bất kỳ ký tự khoảng trắng nào.
\w	Nó trả về một kết quả khớp nếu chuỗi chứa bất kỳ ký tự từ nào.

\W	Nó trả về một kết quả khớp nếu chuỗi không chứa bất kỳ từ nào.
\Z	Trả về một kết quả khớp nếu các ký tự được chỉ định ở cuối chuỗi.

Bài 15: Xây dựng ứng dụng GUI với Tkinter

- Giới thiệu GUI
- Các thuộc tính cơ bản của Widgets
- Làm việc với Tkinter Widgets
- Geometry Management

15.1 Tkinter

Tkinter là một gói trong Python có chứa module Tk hỗ trợ cho việc lập trình GUI. Tk ban đầu được viết cho ngôn ngữ **Tcl**. Sau đó Tkinter được viết ra để sử dụng Tk bằng trình thông dịch Tcl trên nền Python.

Trong Tkinter có hai loại widget, loại thứ nhất là các widget thường như nút bấm, textbox... loại thứ hai là container, đây là những widget chứa các widget khác, chúng còn có một cái tên nữa là layout.

Trong Tkinter có 3 loại layout là pack, grid và place. Trong đó place là kiểu layout tự do, thuật ngữ gọi là **Absolute Positioning**, tức là bạn sẽ phải tự quy định vị trí cũng như kích thước của các widget. Layout pack sắp xếp các widget của bạn theo chiều ngang và chiều dọc. Còn layout grid sắp xếp widget theo dạng bảng.

15.2 Tạo ứng dụng đầu tiên

Việc tạo một ứng dụng sử dụng Tkinter là một công việc vô cùng đơn giản. Tất cả những gì bạn cần làm là làm theo các bước sau:

- Nhập mô đun Tkinter
- Tạo cửa sổ ứng dụng chính của GUI
- Thêm một vài widgets vào ứng dụng GUI
- Nhập vòng lặp event chính để thực hiện hành động với từng sự kiện do người dùng kích hoạt

Ví dụ:

```
import Tkinter  
top = Tkinter.Tk()  
# Code to add widgets will go here...  
top.mainloop()
```

15.3 Các widgets của Tkinter python

Tkinter cung cấp nhiều bảng điều khiển khác nhau được sử dụng trong một ứng dụng GUI như các nút, nhãn và hộp kiểm,... Những bảng điều khiển này thường được gọi là widget. Hiện tại có 15 kiểu widget trong Tkinter. Các tiện ích này được liệt kê trong bảng dưới đây:

STT	Mô tả
1	Button: Tiện ích Button được sử dụng để hiển thị các nút trong ứng dụng
2	Canvas: Sử dụng để vẽ các hình dạng, chẳng hạn như đường thẳng, hình bầu dục, đa giác và hình chữ nhật, trong ứng dụng của bạn.
3	Checkbutton: sử dụng để hiển thị một số tùy chọn dưới dạng hộp kiểm. Người dùng có thể chọn nhiều tùy chọn cùng một lúc.
4	Entry: được sử dụng để hiển thị trường văn bản một dòng để chấp nhận các giá trị từ người dùng.
5	Frame: được sử dụng như một widget vùng chứa để sắp xếp các widget khác.
6	Label: được sử dụng để cung cấp chú thích một dòng cho các tiện ích con khác. Nó cũng có thể chứa hình ảnh.
7	Listbox: được sử dụng để cung cấp danh sách các tùy chọn cho người dùng.
8	Menubutton: được sử dụng để hiển thị các menu trong ứng dụng của bạn.
9	Menu: được sử dụng để cung cấp các lệnh khác nhau cho người dùng. Các lệnh này được chứa bên trong Menubutton.
10	Message: được sử dụng để hiển thị các trường văn bản nhiều dòng để chấp nhận các giá trị từ người dùng.
11	Radiobutton: được sử dụng để hiển thị một số tùy chọn dưới dạng các nút radio. Người dùng chỉ có thể chọn một tùy chọn tại một thời điểm.
12	Scale: được sử dụng để cung cấp tiện ích con trượt.
13	Scrollbar: được sử dụng để thêm khả năng cuộn vào các tiện ích con khác nhau, chẳng hạn như hộp danh sách.
14	Text: được sử dụng để hiển thị văn bản trong nhiều dòng.
15	Toplevel: được sử dụng để cung cấp một vùng chứa cửa sổ riêng biệt.

16	Spinbox: Tiện ích Spinbox là một biến thể của tiện ích Tkinter Entry tiêu chuẩn, có thể được sử dụng để chọn từ một số giá trị cố định.
17	PanedWindow: PanedWindow là một widget vùng chứa có thể chứa bất kỳ số lượng ngăn nào, được sắp xếp theo chiều ngang hoặc chiều dọc.
18	LabelFrame: Labelframe là một tiện ích chứa đơn giản. Mục đích chính của nó là hoạt động như một bộ đệm hoặc vùng chứa cho các bố cục cửa sổ phức tạp
19	tkMessageBox: Mô-đun này được sử dụng để hiển thị các hộp thông báo trong các ứng dụng của bạn.

15.4 Quản lý hình học trong Tkinter python

Để lập trình GUI bằng tkinter thành thạo các khối hình cũng là yếu tố cần thiết và quan trọng. Các widget Tkinter đều có quyền truy cập vào các phương pháp quản lý hình học cụ thể, có mục đích tổ chức các widget trong toàn bộ khu vực widget chính. Tkinter cũng đưa ra các lớp với trình quản lý hình học như: pack, grid và place:

- Phương thức **pack()**: sắp xếp các tiện ích trong khối trước khi đặt chúng vào các parent widgets.
- Phương thức **grid()**: Sắp xếp các tiện ích trong một cấu trúc bảng tương tự như trong tiện ích parents.
- Phương thức **place()**: Sắp xếp các tiện ích bằng cách đặt chúng vào các vị trí cụ thể trong tiện ích parents.

Bài 16: Một số ứng dụng Python

16.1 Thư viện NumPy

16.1.1 Giới thiệu

Numpy là thư viện cho phép xử lý các mảng số (mảng một chiều & mảng nhiều chiều). So với kiểu List có sẵn của Python, các mảng của NumPy khác ở điểm là các phần tử của mảng có cùng kiểu dữ liệu (int, float, string, ...), do đó tốc độ xử lý trên các mảng này nhanh hơn so với xử lý trên kiểu List.

16.1.2 Một số hàm thường sử dụng của numpy

Hàm **numpy.array(lst, dtype)** : chuyển đổi đối tượng kiểu List sang một mảng có kiểu dữ liệu *dtype*. Các giá trị *dtype* thường dùng:

- numpy.float
- numpy.float32
- numpy.float64
- numpy.int
- numpy.uint
- numpy.uint8
- numpy.int8
- numpy.uint16
- numpy.int16
- numpy.int32
- numpy.uint32
- numpy.int64
- numpy.uint64

Ví dụ:

```
>>> import numpy  
>>> X = numpy.array([1,2,3,4])  
>>> print(X)  
[1 2 3 4]  
>>> print(X.dtype)  
int32  
>>> X = numpy.array([1.0,2,3,4])  
>>> print(X)  
[1. 2. 3. 4.]  
>>> print(X.dtype)  
float64  
>>> X = numpy.array([1,2,3,4], dtype=numpy.float64)
```

```
>>> print(X)
[1. 2. 3. 4.]
>>> print(X.dtype)
float64
```

Nếu kiểu dữ liệu không được chỉ định, numpy sẽ dựa vào giá trị các phần tử của List để chọn kiểu dữ liệu phù hợp.

Hàm **numpy.zeros(shape, dtype)**: tạo một mảng có kích thước bằng với *shape* và kiểu dữ liệu *dtype*, các phần tử của mảng được khởi tạo bằng 0. Giá trị mặc định của *dtype* là *numpy.float64*

Nếu *shape* là một số , mảng tạo ra là mảng một chiều :

```
>>> X = numpy.zeros(4)
>>> print(X)
[0. 0. 0. 0.]
```

Nếu *shape* là một bộ tuple/list , mảng tạo ra là mảng nhiều chiều:

```
>>> X = numpy.zeros((2,2))
>>> print(X)
[[0. 0.]
 [0. 0.]]
```

Hàm **numpy.ones(shape)** : tương tự hàm zeros chỉ khác là các phần tử của mảng được khởi tạo bằng 1

```
>>> X = numpy.ones(4)
>>> print(X)
[1. 1. 1. 1.]
>>> X = numpy.ones((2,2))
>>> print(X)
[[1. 1.]
 [1. 1.]]
```

Hàm **numpy.concatenate((arr1, arr2, ...), axis)** : Ghép các mảng lại thành một theo chiều dữ liệu thứ axis. Giá trị mặc định của axis là 0.

Ví dụ:

```
>>> X1 = numpy.array([1,2,3,4])
>>> print(X1)
[1 2 3 4]
>>> X2 = numpy.array([5,6,7,8])
>>> print(X2)
[5 6 7 8]
>>> X3 = numpy.concatenate((X1,X2))
>>> print(X3)
```

```

[1 2 3 4 5 6 7 8]
>>> X1 = numpy.zeros((2,2))
>>> print(X1)
[[0. 0.]
 [0. 0.]]
>>> X2 = numpy.ones((2,2))
>>> print(X2)
[[1. 1.]
 [1. 1.]]
>>> X3 = numpy.concatenate((X1,X2))
>>> print(X3)
[[0. 0.]
 [0. 0.]
 [1. 1.]
 [1. 1.]]
>>> X4 = numpy.concatenate((X1,X2),axis=0)
>>> print(X4)
[[0. 0.]
 [0. 0.]
 [1. 1.]
 [1. 1.]]
>>> X5 = numpy.concatenate((X1,X2),axis=1)
>>> print(X5)
[[0. 0. 1. 1.]
 [0. 0. 1. 1.]]

```

Hàm **arr.shape**, **arr.dtype** : trả về kích thước và kiểu dữ liệu của mảng arr

```

>>> X = numpy.array([[1,2],[3,4]])
>>> print(X.dtype)
Int32
>>> print(X.shape)
(2, 2)

```

Hàm **arr.reshape(new_shape)**: tạo ra một mảng mới bằng cách xếp lại dữ liệu của mảng arr đã có theo kích thước mới new_shape

```

>>> X1 = numpy.array([1,2,3,4])
>>> print(X1)
[1 2 3 4]
>>> X2 = X1.reshape((2,2))
>>> print(X2)

```

```
[[1 2]
 [3 4]]
```

16.1.3 Truy nhập phần tử của mảng trong numpy

Với mảng một chiều, cách truy nhập phần tử của mảng tương tự cách truy nhập phần tử của List

```
>>> X = numpy.array([1,2,3,4])
>>> print(X[0])
1
>>> print(X[:2])
[1 2]
>>> print(X[-2:])
[3 4]
```

Với mảng nhiều chiều, một phần tử được truy nhập bởi danh sách các chỉ số nằm trong một cặp dấu ngoặc vuông duy nhất : arr[i1, i2, ..., in]

```
>>> print(X)
[[1 2]
 [3 4]]
>>> print(X[0,1])
2
>>> print(X[1,0])
3
```

Tương tự mảng một chiều hay List, có thể cách dùng các chỉ số dạng hai chấm (start:end, start:, :end, ...) để truy nhập tới từng đoạn nhỏ trong mảng nhiều chiều:

```
>>> X = numpy.array([[1,2],[3,4]])
>>> print(X)
[[1 2]
 [3 4]]
>>> print(X[:, 0])
[1 3]
>>> print(X[1, :])
[3 4]
```

16.1.4 Các phép tính trên mảng dữ liệu numpy

16.1.4.1 Phép tính element-wise

Nếu hai mảng có cùng kích thước thì các phép tính `+`, `-`, `*`, `/`, `**`, `%`, `//`, ... có thể được thực hiện trên hai mảng đó. Kết quả là một mảng có cùng kích thước mà mỗi phần tử là kết quả của phép toán được thực hiện trên từng cặp phần tử của hai mảng đầu vào

Ví dụ:

```
>>> X1 = numpy.array([[1,2],[3,4]])
>>> print(X1)
[[1 2]
 [3 4]]
>>> X2 = numpy.array([[5,6],[7,8]])
>>> print(X2)
[[5 6]
 [7 8]]
>>> X3 = X1 + X2
>>> print(X3)
[[ 6  8]
 [10 12]]
>>> X4 = X1 ** 2
>>> print(X4)
[[ 1  4]
 [ 9 16]]
```

16.1.4.2 So sánh một mảng với một số

Kết quả phép so sánh một mảng với một số là một mảng có kiểu dữ liệu `dtype=boolean`, cùng kích thước với mảng ban đầu, tương tự như phép tính *element-wise*

```
>>> X1 = numpy.array([[1,2],[3,4]])
>>> print(X1)
[[1 2]
 [3 4]]
>>> X2 = X1 > 2
>>> print(X2)
[[False False]
 [ True  True]]
```

Giá trị của mảng boolean kết quả có thể dùng để làm chỉ số để truy nhập mảng gốc (hoặc các mảng cùng kích thước) :

```
>>> X1 = numpy.array([1,2,3,4])
>>> print(X1)
```

```

[1 2 3 4]
>>> X2 = numpy.array([5,6,7,8])
>>> print(X2)
[5 6 7 8]
>>> X3 = X1[X1>2]
>>> print(X3)
[3 4]
>>> X4 = X2[X1>2]
>>> print(X4)
[7 8]

```

16.1.4.3 Các hàm toán học

Numpy cung cấp các hàm toán học cùng tên với các hàm của thư viện math : **sin, cos, tan, asin, acos, sqrt, floor, ceil, log, log10, exp**,... Các hàm toán học cũng được thực hiện theo cách xử lý element-wise

Ví dụ:

```

>>> X1 = numpy.array([[1,10],[100,1000]])
>>> print(X1)
[[ 1 10]
 [100 1000]]
>>> X2 = numpy.log10(X1)
>>> print(X2)
[[0. 1.]
 [2. 3.]]

```

16.1.4.4 Các hàm thống kê

numpy.sum(arr, axis=None) : Tính tổng một mảng dọc theo chiều dữ liệu axis.

Nếu axis=None, hàm trả về tổng của toàn mảng.

```

>>> X = numpy.array([[1,2],[3,4]])
>>> print(X)
[[1 2]
 [3 4]]
>>> print(numpy.sum(X))
10
>>> print(numpy.sum(X, axis=0))
[4 6]
>>> print(numpy.sum(X, axis=1))
[3 7]

```

numpy.mean(arr, axis=0) : Tính trung bình của mảng dọc

```

>>> X = numpy.array([[1,2],[3,4]])
>>> print(numpy.mean(X))

```

```

2.5
>>> print(numpy.mean(X, axis=0))
[2. 3.]
>>> print(numpy.mean(X, axis=1))
[1.5 3.5]

numpy.min(arr, axis=None) : Tính giá trị nhỏ nhất của mảng
>>> X = numpy.array([[1,2],[3,4]])
>>> print(numpy.min(X))
1
>>> print(numpy.min(X, axis=0))
[1 2]
>>> print(numpy.min(X, axis=1))
[1 3]

numpy.max(arr, axis=None) : Tính giá trị lớn nhất của mảng
>>> X = numpy.array([[1,2],[3,4]])
>>> print(numpy.max(X))
4
>>> print(numpy.max(X, axis=0))
[3 4]
>>> print(numpy.max(X, axis=1))
[2 4]

```

16.2 Làm việc với Matplotlib để vẽ đồ thị

Cài thư viện nếu chưa có: Mở CMD, gõ lệnh `pip install matplotlib`

Để thực hiện các suy luận thống kê cần thiết, cần phải trực quan hóa dữ liệu của bạn và Matplotlib là một trong những giải pháp như vậy cho người dùng Python. Nó là một thư viện vẽ đồ thị rất mạnh mẽ hữu ích cho những người làm việc với Python và NumPy. Module được sử dụng nhiều nhất của Matplotlib là Pyplot cung cấp giao diện như MATLAB nhưng thay vào đó, nó sử dụng Python và nó là nguồn mở.

Để sử dụng thư viện Matplotlib, bước đầu tiên sau khi thiết lập môi trường Python là chúng ta phải cài đặt package phải không nào. Sử dụng `pip install matplotlib` để cài đặt version mới nhất.

Sau đó ta import thư viện bằng dòng code :

```
import matplotlib.pyplot as plt
```

16.2.1 Khái niệm chung

Một Matplotlib figure có thể được phân loại thành nhiều phần như dưới đây:

- **Figure:** Như một cái cửa sổ chứa tất cả những gì bạn sẽ vẽ trên đó.
- **Axes:** Thành phần chính của một figure là các axes (những khung nhỏ hơn để vẽ hình lên đó). Một figure có thể chứa một hoặc nhiều axes. Nói cách khác, figure chỉ là khung chứa, chính các axes mới thật sự là nơi các hình vẽ được vẽ lên.
- **Axis:** Chúng là dòng số giống như các đối tượng và đảm nhiệm việc tạo các giới hạn biểu đồ.
- **Artist:** Mọi thứ mà bạn có thể nhìn thấy trên figure là một artist như `Text` objects, `Line2D` objects, `collection` objects. Hầu hết các Artists được gắn với Axes.

16.2.2 Bắt đầu với Pyplot

Pyplot là một module của Matplotlib cung cấp các hàm đơn giản để thêm các thành phần plot như lines, images, text, v.v. vào các axes trong figure.

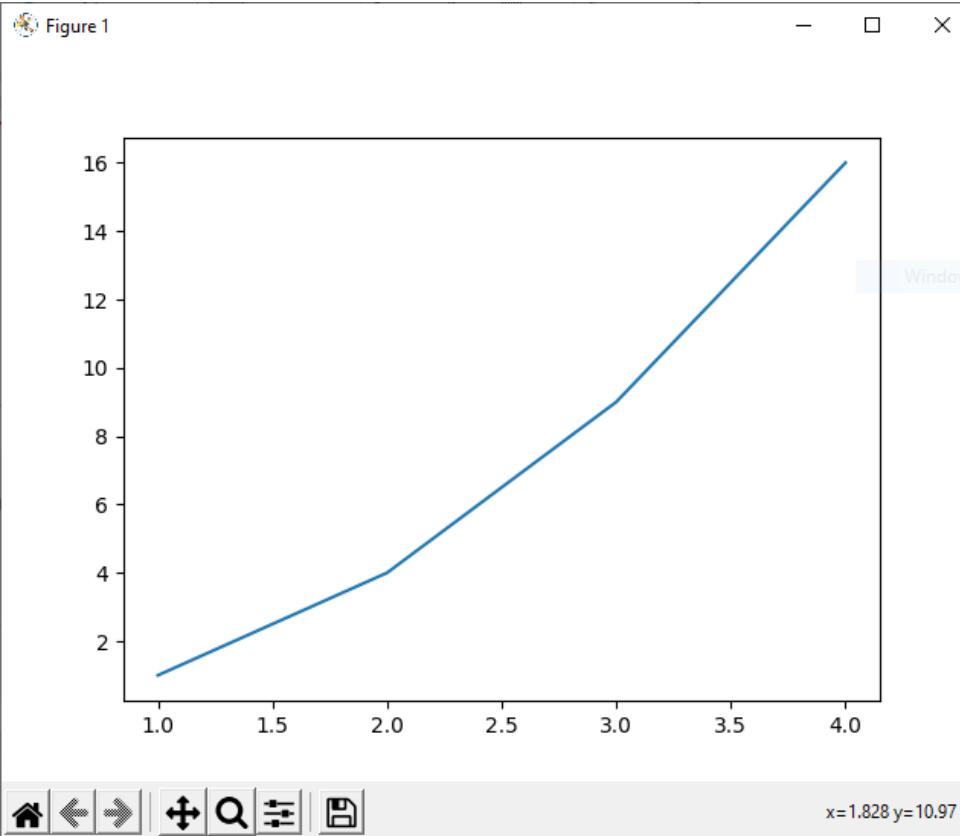
16.2.2.1 Tạo một biểu đồ đơn giản

```
import matplotlib.pyplot as plt
```

Ở đây chúng ta import Matplotlib's Pyplot module và thư viện Numpy vì hầu hết các dữ liệu mà ta sẽ làm việc sẽ chỉ ở dạng mảng.

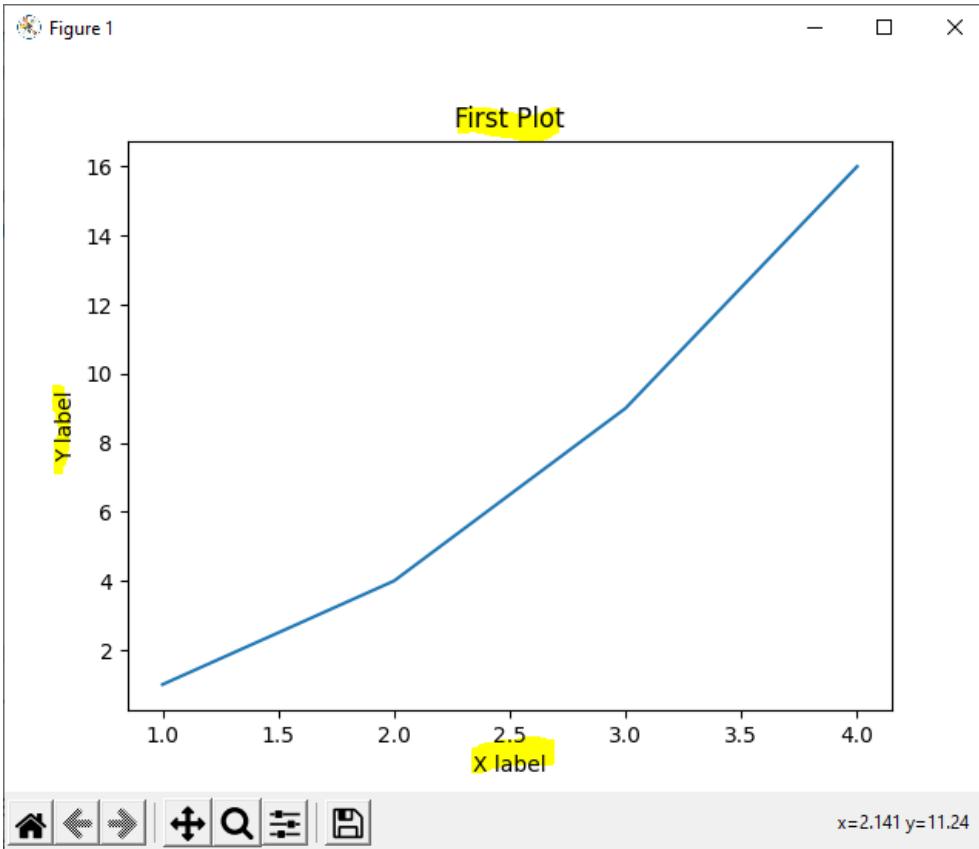
```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
plt.show()
```



Chúng ta chuyển hai mảng làm đối số đầu vào cho phương thức `plot()` và sử dụng phương thức `show()` để gọi biểu đồ được yêu cầu. Ở đây lưu ý rằng mảng đầu tiên xuất hiện trên trục x và mảng thứ hai xuất hiện trên trục y của biểu đồ. Bây giờ, biểu đồ đầu tiên của chúng ta đã sẵn sàng, chúng ta hãy thêm tiêu đề và đặt tên trục x và trục y bằng cách sử dụng các phương thức `title()`, `xlabel()` và `ylabel()`.

```
plt.plot([1,2,3,4],[1,4,9,16])
plt.title("First Plot")
plt.xlabel("X label")
plt.ylabel("Y label")
plt.show()
```

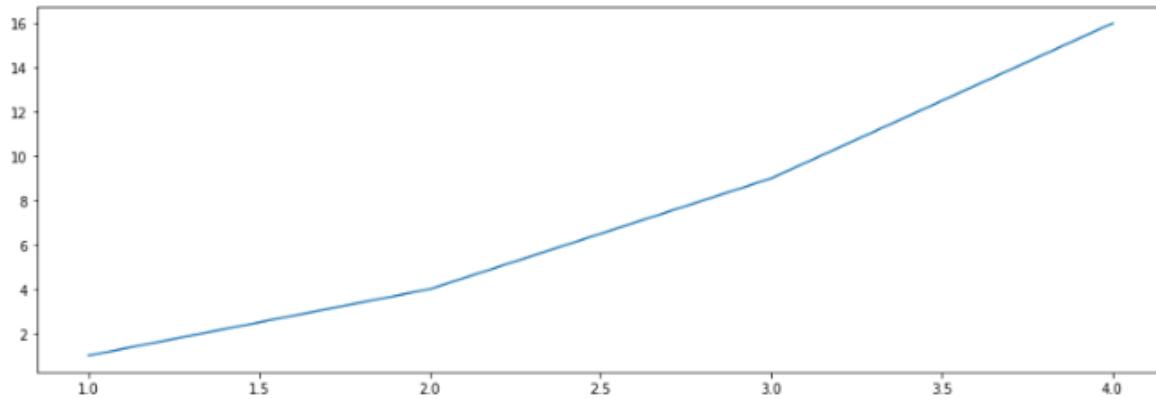


Chúng ta cũng có thể chỉ định kích thước của hình bằng cách sử dụng phương thức `figure()` và truyền các giá trị dưới dạng một tuple về độ dài của các hàng và cột cho đối số `figsize`

```
plt.figure(figsize=(15, 5))
plt.plot([1,2,3,4], [1,4,9,16])
plt.title("First Plot")
plt.xlabel("X label")
plt.ylabel("Y label")
plt.show()
```

```
import matplotlib.pyplot as plt
import numpy as np

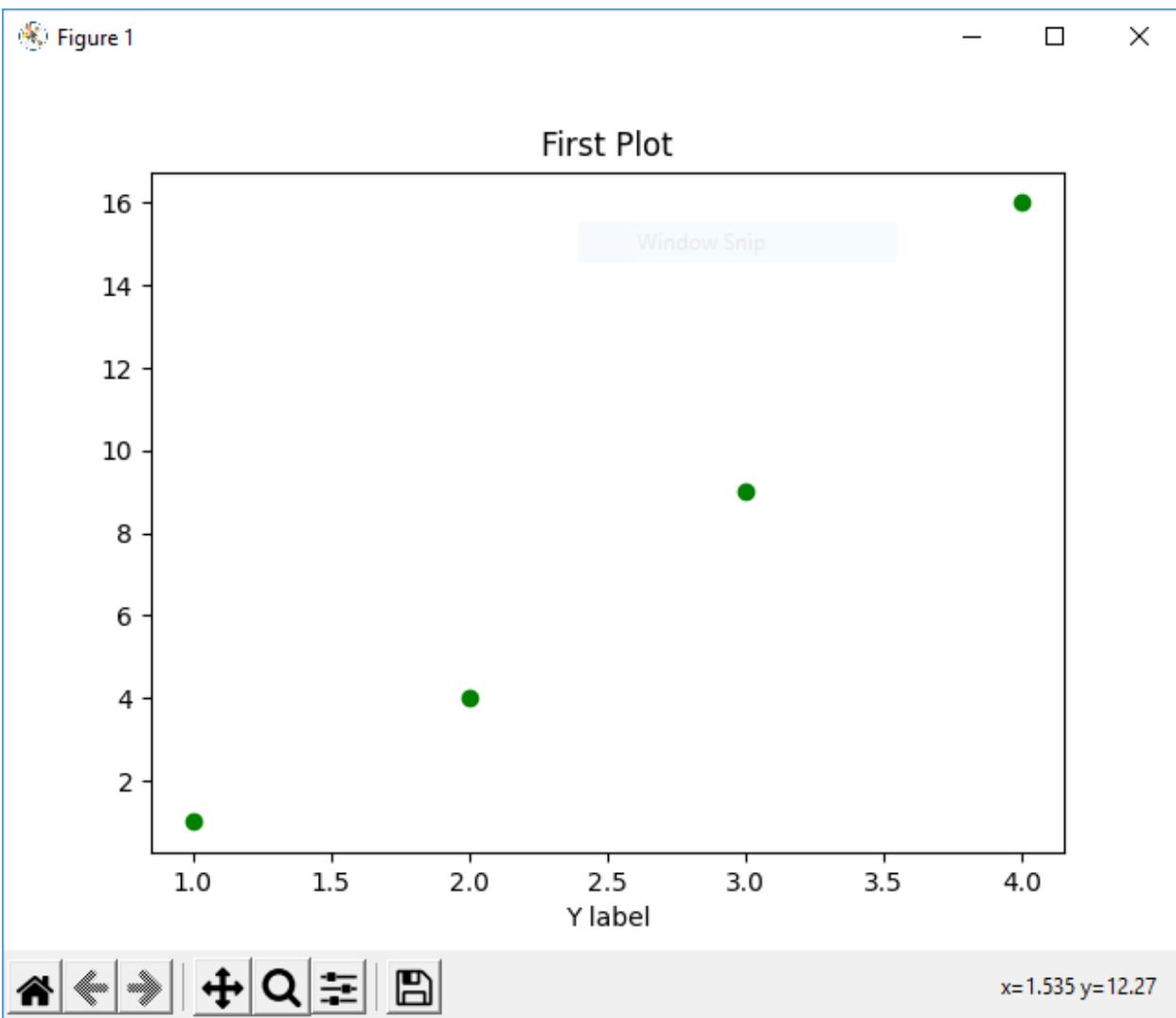
plt.figure(figsize=(15,5))
plt.plot([1,2,3,4],[1,4,9,16])
plt.show()
```



Với mỗi đối số X và Y, bạn cũng có thể chuyển một đối số thứ ba tùy chọn dưới dạng một chuỗi cho biết màu sắc và loại đường của biểu đồ. Định dạng mặc định là **b**- có nghĩa là một đường màu xanh lam đặc. Trong hình dưới đây, mình sử dụng **go** có nghĩa là vòng tròn màu xanh lá cây. Tương tự như vậy, chúng ta có thể thực hiện nhiều kết hợp như vậy để định dạng biểu đồ của mình.

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], "go")
plt.title("First Plot")
plt.xlabel("X label")
plt.ylabel("Y label")
plt.show()
```

Figure 1

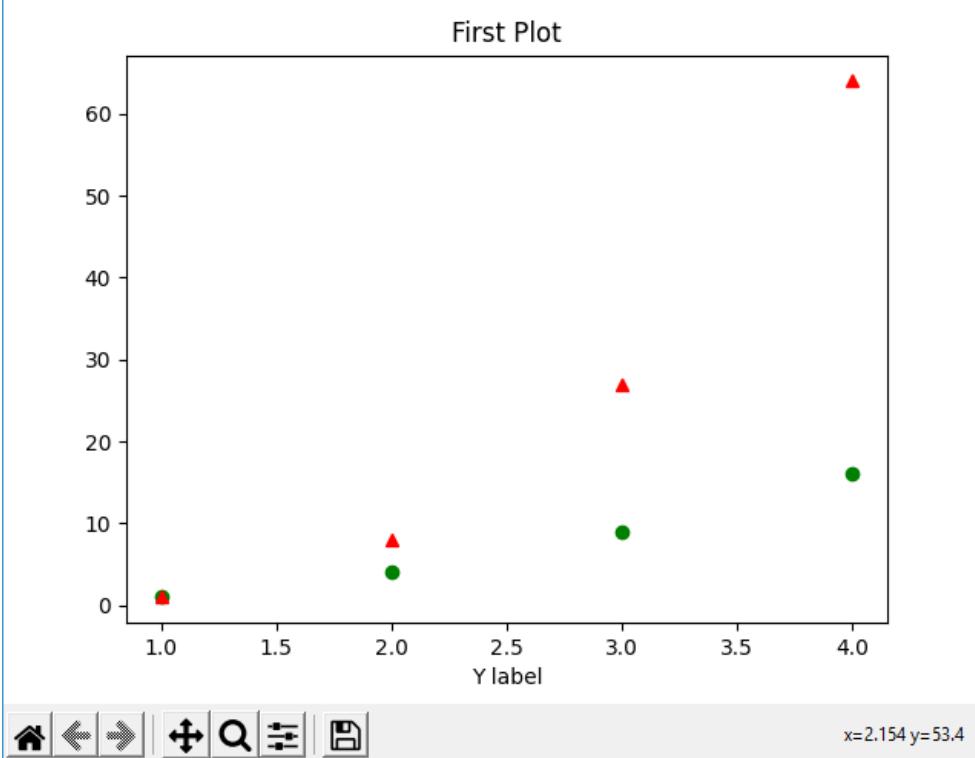


Chúng ta cũng có thể vẽ nhiều bộ dữ liệu bằng cách chuyển vào nhiều bộ đối số của trục X và Y trong phương thức `plot()` như bên dưới.

```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(1, 5)
y = x ** 3
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], "go", x, y, "r^")
plt.title("First Plot")
plt.xlabel("X label")
plt.ylabel("Y label")
plt.show()
```

Figure 1



16.2.2.2 Nhiều biểu đồ trong 1 figure

Chúng ta có thể sử dụng phương thức `subplot()` để thêm nhiều plots trong một hình. Trong hình ảnh bên dưới, mình đã sử dụng phương pháp này để phân tách hai biểu đồ mà đã vẽ trên cùng một trục trong ví dụ trước. Phương thức `subplot()` có ba đối số: `nrows`, `ncols` và `index`. Chúng chỉ ra số lượng hàng, số cột và số index của sub-plot. Ví dụ, mình muốn tạo hai sub-plot trong một hình sao cho nó nằm trên một hàng và trên hai cột và do đó ta chuyển các đối số (1,2,1) và (1,2,2) trong phương thức `subplot()`. Lưu ý rằng ta đã sử dụng riêng phương thức `title()` cho cả các subplots. Ta sử dụng phương thức `suptitle()` để tạo một tiêu đề tập trung cho hình.

```
import matplotlib.pyplot as plt
import numpy as np

plt.subplot(1, 2, 1)
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], "go")
plt.title("First subplot")

x = np.arange(1, 5)
```

```

y = x ** 3
plt.subplot(1, 2, 2)
plt.plot(x, y, "r^")
plt.title("Second subplot")

plt.suptitle("My sub-plots")
plt.show()

```



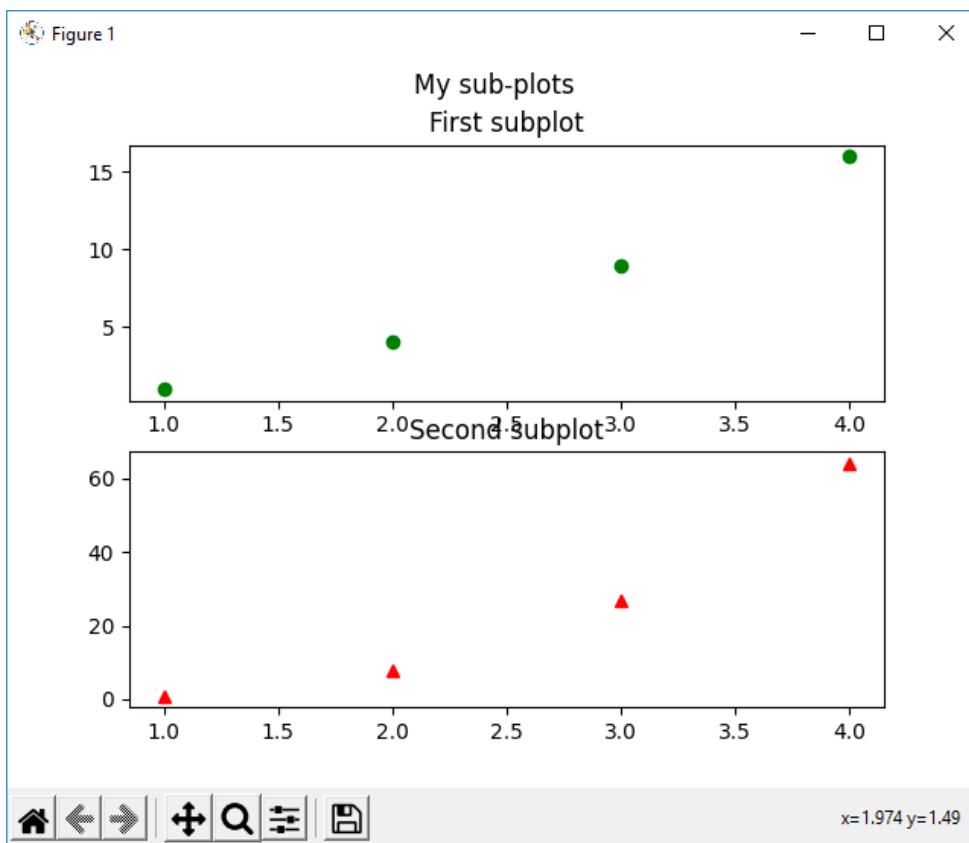
Nếu ta muốn các sub-plots thành hai hàng và một cột, chúng ta có thể truyền các đối số (2,1,1) và (2,1,2)

```

plt.subplot(2, 1, 1)
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], "go")
plt.title("First subplot")

```

```
plt.subplot(2, 1, 2)
plt.plot(x, y, "r^")
plt.title("Second subplot")
```



Cách tạo ra subplots trên đây trở nên hơi tẻ nhạt khi chúng ta muốn có nhiều subplots trong hình. Một cách thuận tiện hơn là sử dụng phương thức `subplots()`. Bạn có thể tham khảo thêm các hình thức trình bày subplot tại https://matplotlib.org/stable/gallery/subplots_axes_and_figures/subplots_demo.html.

16.2.2.3 Tạo các loại biểu đồ khác nhau với Pyplot

1. Biểu đồ thanh

Biểu đồ thanh là một trong những loại biểu đồ phổ biến nhất và được sử dụng để hiển thị dữ liệu được liên kết với các biến phân loại. Pyplot cung cấp một phương thức `bar()` để tạo các biểu đồ thanh có các đối số: biến phân loại, giá trị và màu sắc của chúng (nếu bạn muốn chỉ định bất kỳ)

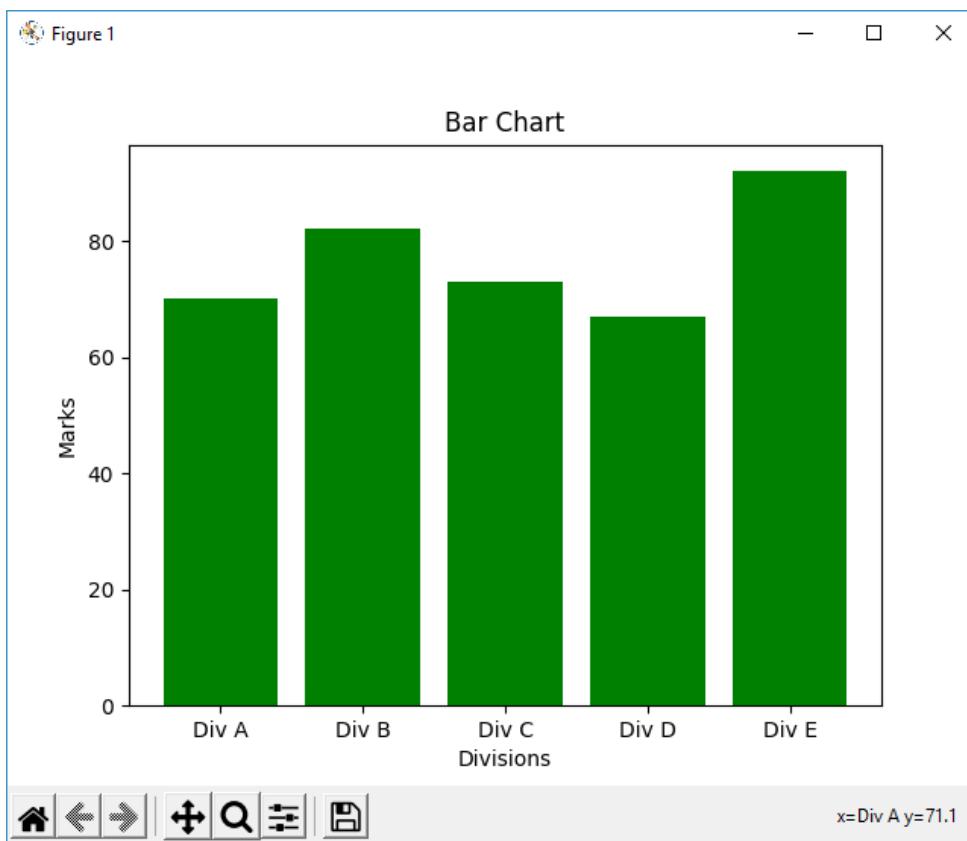
```

import matplotlib.pyplot as plt

divisions = ["Div A", "Div B", "Div C", "Div D", "Div E"]
division_average_marks = [70, 82, 73, 67, 92]

plt.bar(divisions, division_average_marks, color="green")
plt.title("Bar Chart")
plt.xlabel("Divisions")
plt.ylabel("Marks")
plt.show()

```



Để tạo biểu đồ thanh ngang sử dụng phương thức `barh()` Ngoài ra, chúng ta có thể truyền đối số (với giá trị của nó) `xerr` or `yerr` (trong trường hợp biểu đồ thanh dọc ở trên) để mô tả phương sai trong dữ liệu của chúng ta như sau:

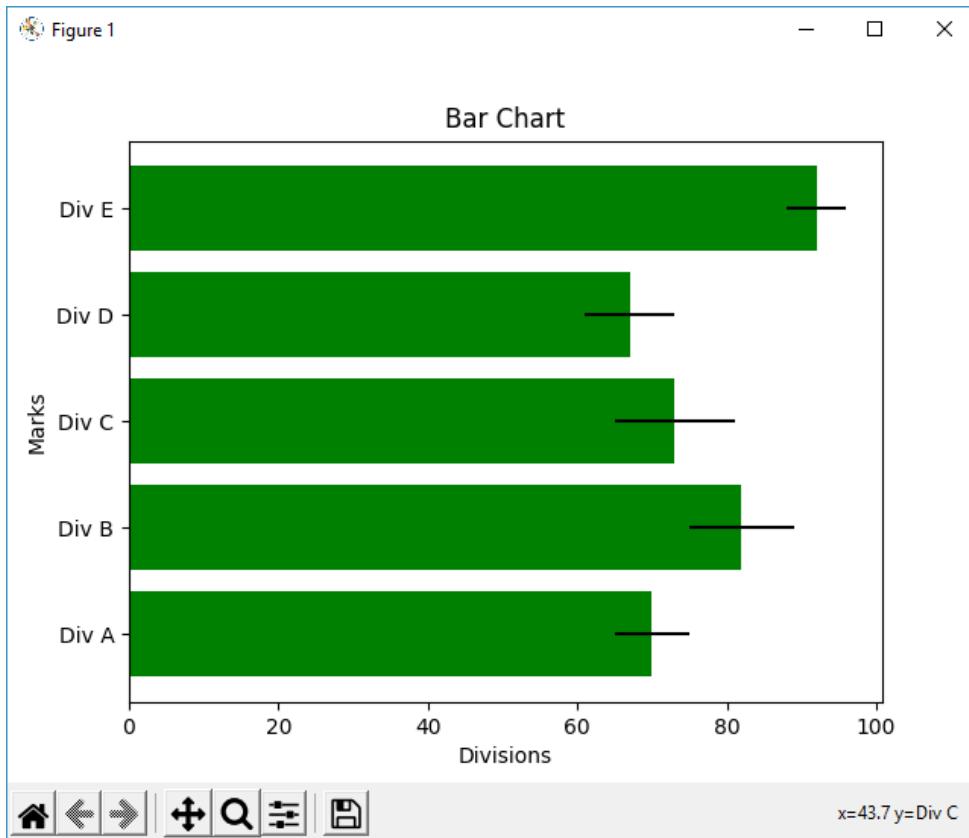
```

divisions = ["Div A", "Div B", "Div C", "Div D", "Div E"]
division_average_marks = [70, 82, 73, 67, 92]
variance = [5, 7, 8, 6, 4]

```

```
plt.barh(divisions, division_average_marks,
```

```
xerr=variance, color="green")
plt.title("Bar Chart")
plt.xlabel("Divisions")
plt.ylabel("Marks")
plt.show()
```



Để tạo các biểu đồ thanh xếp chồng theo chiều ngang, ta sử dụng phương thức `bar()` hai lần và chuyển các đối số trong đó ta đề cập đến index và width của biểu đồ thanh để xếp chúng theo chiều ngang. Ngoài ra, chú ý việc sử dụng hai phương thức `legend()` được sử dụng để hiển thị chú giải của biểu đồ và `xticks()` để gắn nhãn trực x dựa trên vị trí của các thanh.

```

divisions = ["Div-A", "Div-B", "Div-C", "Div-D", "Div-E"]
division_average_marks = [70, 82, 73, 65, 68]
boys_average_marks = [68, 67, 77, 61, 70]

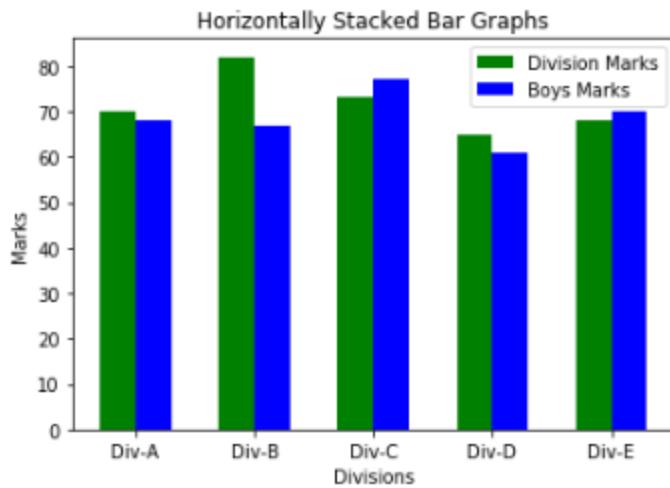
index = np.arange(5)
width = 0.30

plt.bar(index, division_average_marks, width, color='green',label='Division Marks')
plt.bar(index+width, boys_average_marks, width, color='blue',label='Boys Marks')
plt.title("Horizontally Stacked Bar Graphs")

plt.ylabel("Marks")
plt.xlabel("Divisions")
plt.xticks(index+ width/2, divisions)

plt.legend(loc='best')
plt.show()

```



Tương tự, để xếp theo chiều dọc các biểu đồ thanh với nhau, chúng ta có thể sử dụng đối số `bottom` và đề cập đến biểu đồ thanh mà chúng ta muốn xếp chồng bên dưới làm giá trị của nó.

```

divisions = ["Div-A", "Div-B", "Div-C", "Div-D", "Div-E"]
boys_average_marks = [68, 67, 77, 61, 70]
girls_average_marks = [72, 97, 69, 69, 66]

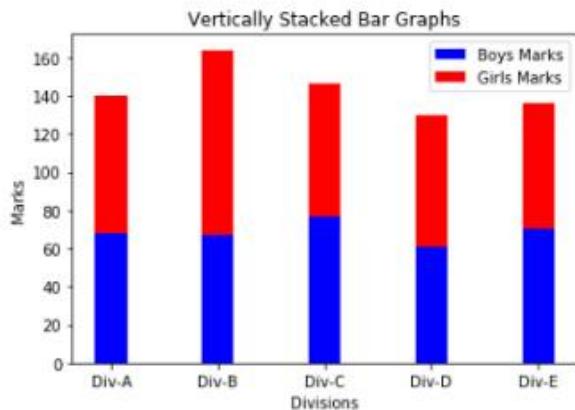
index = np.arange(5)
width = 0.30

plt.bar(index, boys_average_marks, width, color="blue", label="Boys Marks")
plt.bar(index, girls_average_marks, width, color="red", label="Girls Marks", bottom=boys_average_marks)

plt.title("Vertically Stacked Bar Graphs")
plt.xlabel("Divisions")
plt.ylabel("Marks")
plt.xticks(index, divisions)

plt.legend(loc='best')
plt.show()

```



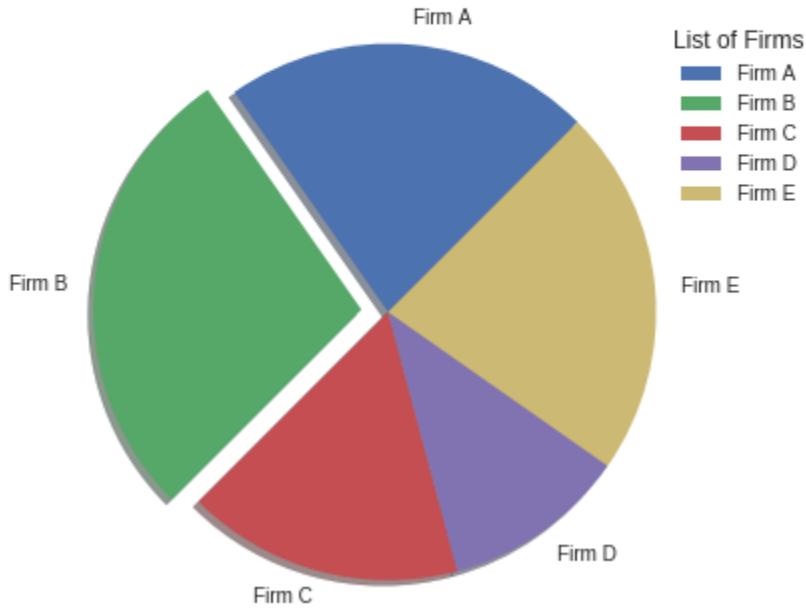
2. Biểu đồ tròn

Một loại biểu đồ cơ bản nữa là biểu đồ Pie có thể được tạo bằng phương thức `pie()`. Chúng ta cũng có thể chuyển các đối số để tùy chỉnh biểu đồ Pie của mình để hiển thị shadow, explode một phần của nó, nghiêng nó theo một góc như sau:

```

firms = ["Firm A", "Firm B", "Firm C", "Firm D", "Firm E"]
market_share = [20, 25, 15, 10, 20]
Explode = [0, 0.1, 0, 0, 0]
plt.pie(market_share, explode=Explode, labels=firms, shadow=True, startangle=45)
plt.axis('equal')
plt.legend(title="List of Firms")
plt.show()

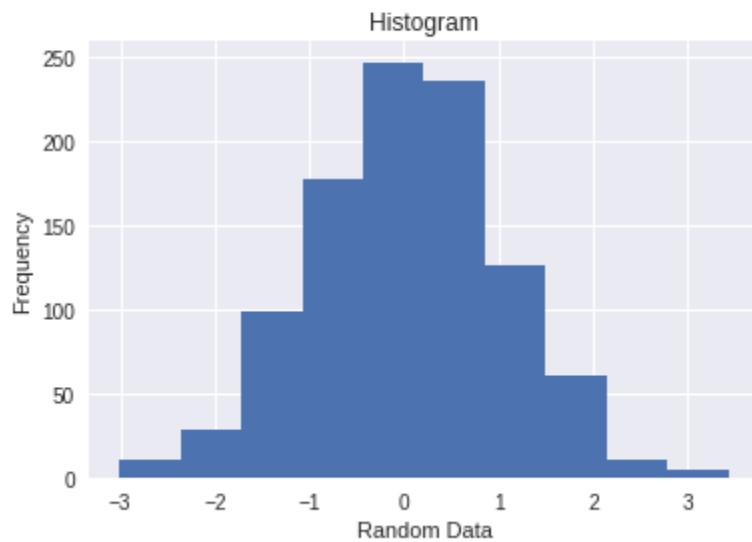
```



3. Histogram Histogram là một loại biểu đồ rất phổ biến khi chúng ta xem xét dữ liệu như chiều cao và cân nặng, giá cổ phiếu, thời gian chờ đợi của một khách hàng, v.v ... liên tục trong tự nhiên. Histogram's data được vẽ trong một phạm vi so với tần số của nó. Histograms là các biểu đồ xuất hiện rất phổ biến trong xác suất và thống kê và tạo cơ sở cho các distributions khác nhau như normal - distribution, t-distribution, v.v. Trong ví dụ sau, chúng ta tạo dữ liệu liên tục ngẫu nhiên gồm 1000 entries và vẽ biểu đồ theo tần số của nó với dữ liệu chia thành 10 tầng bằng nhau. Mình đã sử dụng phương thức `random.randn()` của NumPy's để tạo dữ liệu với các thuộc tính của standard normal distribution, nghĩa là $\mu = 0$ và độ lệch chuẩn $\sigma = 1$ và do đó biểu đồ trông giống như một đường cong normal distribution.

```
x = np.random.randn(1000)

plt.title("Histogram")
plt.xlabel("Random Data")
plt.ylabel("Frequency")
plt.hist(x,10)
plt.show()
```



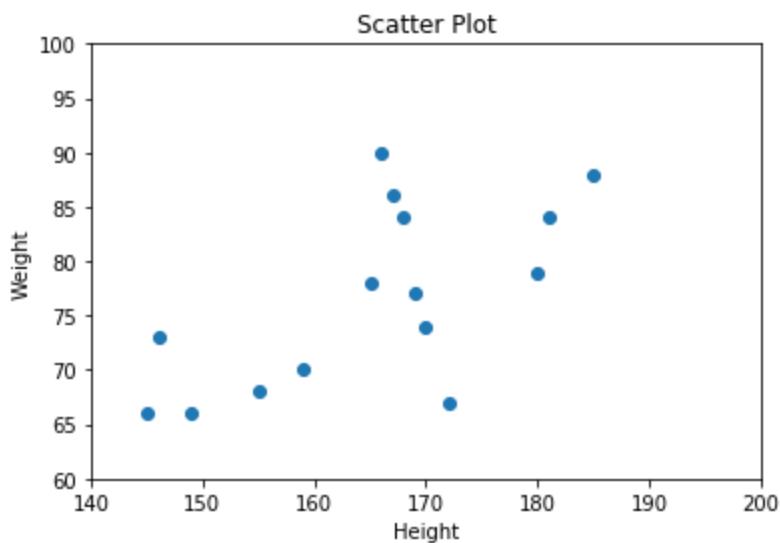
4. Sơ đồ phân tán và 3 chiều

Các biểu đồ phân tán là các biểu đồ được sử dụng rộng rãi, đặc biệt là chúng có ích trong việc hình dung một vấn đề về hồi quy. Trong ví dụ sau, cung cấp dữ liệu được tạo tùy ý về chiều cao và cân nặng và vẽ chúng với nhau. Mình đã sử dụng các phương thức `xlim()` và `ylim()` để đặt giới hạn của trục X và trục Y tương ứng.

```

height = np.array([167,170,149,165,155,180,166,146,
                  159,185,145,168,172,181,169])
weight = np.array([86,74,66,78,68,79,90,73,
                  70,88,66,84,67,84,77])
plt.xlim(140,200)
plt.ylim(60,100)
plt.scatter(height,weight)
plt.title("Scatter Plot")
plt.xlabel("Height")
plt.ylabel("Weight")
plt.show()

```

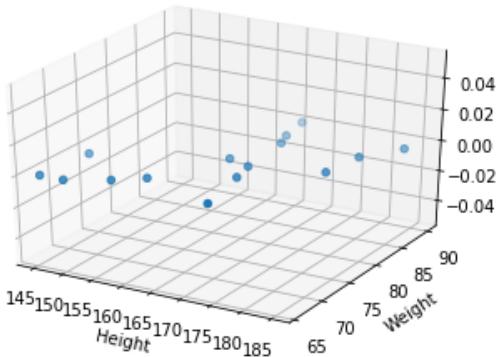


Sự phân tán ở trên cũng có thể được hình dung trong ba chiều. Để sử dụng chức năng này, trước tiên ta cần import module `mplplot3d` như sau:

```
from mpl_toolkits import mplot3d
```

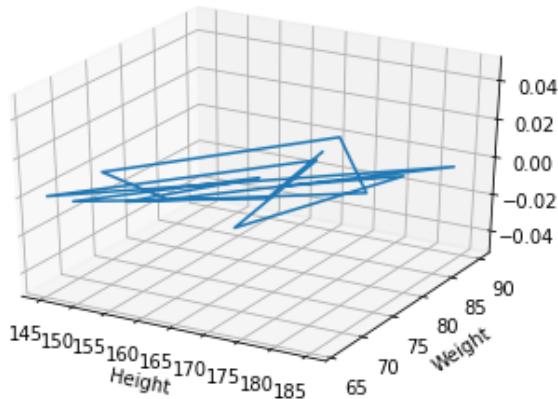
Khi module được nhập, một trục ba chiều được tạo bằng cách chuyển từ khóa `projection='3d'` sang phương thức `axes()` của module Pyplot. Khi đối tượng được tạo, chúng ta chuyển chiều cao và trọng số của đối số cho phương thức `scatter3D()`.

```
ax = plt.axes(projection='3d')
ax.scatter3D(height,weight)
ax.set_xlabel("Height")
ax.set_ylabel("Weight")
plt.show()
```



Chúng ta cũng có thể tạo các biểu đồ 3 chiều của các loại khác như biểu đồ đường, bề mặt, khung lưới, đường viền, v.v. Ví dụ trên ở dạng biểu đồ đường đơn giản như sau: Ở đây thay vì `scatter3D()` chúng ta sử dụng phương thức `plot3D()`

```
ax = plt.axes(projection='3d')
ax.plot3D(height,weight)
ax.set_xlabel("Height")
ax.set_ylabel("Weight")
plt.show()
```



16.3 Scraping data

TÀI LIỆU THAM KHẢO

1. <https://docs.python.org/3/reference/index.html>
2. <https://www.python-course.eu/course.php>
3. <https://www.learnpython.org/>
4. <https://pandas.pydata.org/pandas-docs/stable/index.html>
5. <https://phocode.com/python>
6. <https://static.realpython.com/python-basics-sample-chapters.pdf>
7. <https://pymbook.readthedocs.io/en/latest/index.html>
8. <https://www.codecademy.com/learn/learn-python>
9. <https://automatetheboringstuff.com>
10. Các bài giảng, bài viết từ internet.

■ ■ ■