

Trung tâm Đào tạo CNTT Nhất Nghệ

GV: Lương Trần Hy Hiền

Tài liệu

Lập trình Python

TP.HCM

09, 2020





LỜI MỞ ĐẦU

—

MỤC LỤC

LỜI MỞ ĐẦU	1
MỤC LỤC.....	3
Bài 1: Tổng quan Lập trình Python.....	9
1.1 Lập trình là gì?.....	9
1.2 Python là gì.....	10
1.2.1 Giới thiệu.....	10
1.2.2 Lịch sử của Python	10
1.2.3 Các phiên bản Python đã phát hành.....	11
1.2.4 Python được dùng ở đâu?	13
1.2.5 Đặc điểm của Python.....	14
1.3 Cài đặt môi trường.....	14
1.3.1 Cài đặt Python	14
1.3.2 Giới thiệu Pycharm IDE	20
1.4 Phân khối và chú thích	33
1.4.1 Phân khối	33
1.4.2 Chú thích.....	34
1.5 Python keyword.....	34
1.6 Biến.....	44
1.6.1 Biến là gì	44
1.6.2 Gán giá trị cho biến	45
1.6.3 Xóa biến khỏi bộ nhớ.....	45
1.6.4 Quy tắc đặt tên biến.....	45

1.7	Nhập xuất dữ liệu trên shell	45
1.7.1	Nhập với input	46
1.7.2	Xuất với print	47
Bài 2:	Kiểu dữ liệu, Toán tử.....	48
2.1	Các kiểu dữ liệu	48
2.1.1	Xác định id đối tượng	49
2.1.2	Xác định kiểu dữ liệu	49
2.1.3	Kiểu số.....	50
2.1.4	Kiểu chuỗi.....	51
2.1.5	Kiểu Boolean.....	54
2.1.6	Kiểu None.....	56
2.1.7	Kiểu Tuple	56
2.1.8	Kiểu List	60
2.1.9	Kiểu Set.....	65
2.1.10	Kiểu từ điển	67
2.2	Toán tử số học	68
2.3	Toán tử logic	69
2.4	Toán tử so sánh.....	69
2.5	Gán	70
2.6	Logic.....	72
2.7	Định danh	73
2.8	Độ ưu tiên toán tử	73
2.9	Các hàm toán học:.....	73
2.9.1	Type Conversion	74

2.9.2	Iterables and Iterators	75
2.9.3	Composite Data Type	75
Bài 3:	Cấu trúc Điều khiển, Lặp	77
3.1	Lệnh rẽ nhánh	77
3.1.1	Câu lệnh If.....	77
3.1.2	Câu lệnh If .. Else.....	77
3.1.3	Câu lệnh If .. ElIf .. Else.....	77
3.2	Lệnh lặp while	78
3.3	Lệnh lặp for.....	80
3.4	Sử dụng break,continue, pass statement	82
3.4.1	Câu lệnh Break.....	82
3.4.2	Câu lệnh Continue	83
Bài 4:	Number, String, Date & Time.....	86
4.1	Kiểu số	86
4.2	Kiểu chuỗi (string)	86
4.2.1	Xử lý chuỗi	86
4.2.2	Nối chuỗi	86
4.2.3	Dùng dấu nháy bên trong string	87
4.2.4	Độ dài của string	88
4.2.5	Loại bỏ khoảng trắng thừa trong string	88
4.2.6	Các ký tự thoát	90
4.2.7	So sánh string	91
4.2.8	Truy xuất phần tử trong string	92
4.2.9	Tìm string con.....	94

4.2.10	Toán tử trên string	97
4.2.11	Thay thế string.....	98
4.2.12	Tách, nối string	99
4.2.13	Chữ HOA và chữ thường	102
4.2.14	Một số thao tác khác trên string	103
4.2.15	Định dạng string	105
4.3	Kiểu Date, Time.....	109
Bài 5:	Kiểu cấu trúc	111
5.1	Kiểu List.....	111
5.1.1	Khởi tạo list	111
5.1.2	Hàm list()	113
5.1.3	Các phép tính trên list.....	114
5.1.4	Một số hàm dùng với list.....	115
5.1.5	Thêm phần tử mới vào list	116
5.1.6	Lỗi IndexError, TypeError.....	117
5.1.7	Xóa phần tử trong list.....	119
5.1.8	Thay đổi giá trị phần tử trong list	121
5.1.9	Sao chép một list.....	122
5.1.10	Chỉ số trong list.....	124
5.1.11	Duyệt chuỗi	126
5.1.12	Đếm số lượng phần tử trong list.....	128
5.1.13	Lồng các list lại với nhau.....	129
5.1.14	Sắp xếp list.....	131
5.1.15	Đảo ngược list.....	133

5.1.16	Khởi tạo list bằng comprehension	133
5.1.17	Hàm map() và hàm filter()	134
5.2	Kiểu Tuple.....	136
5.2.1	Giới thiệu và khai báo	136
5.2.2	Truy cập các phần tử.....	136
5.2.3	Các phép tính trên Tuple.....	137
5.3	Kiểu Dictionary	137
5.3.1	Khởi tạo dictionary	138
5.3.2	Các phép toán cơ bản.....	140
5.3.3	Làm việc với khóa và giá trị.....	145
5.3.4	Duyệt dictionary	146
5.4	Kiểu Set	150
5.4.1	Các phép tính trên Set	150
5.4.2	Một số ví dụ minh họa	151
Bài 6:	Hàm (function)	153
6.1	Xây dựng, gọi sử dụng phương thức.....	153
6.1.1	Giới thiệu.....	153
6.1.2	Khai báo hàm.....	153
6.1.3	Tham số của hàm.....	154
6.2	Anonymous Function (lambda).....	154
Bài 7:	Module và làm việc với thư viện	Error! Bookmark not defined.
Bài 8:	Xử lý ngoại lệ	Error! Bookmark not defined.
Bài 9:	File I/O	Error! Bookmark not defined.
Bài 10:	Lập trình hướng đối tượng.....	Error! Bookmark not defined.

Bài 11:	Làm việc với dữ liệu JSON	Error! Bookmark not defined.
Bài 12:	Làm việc với cơ sở dữ liệu	Error! Bookmark not defined.
12.1	Giới thiệu hệ quản trị CSDL SQLite	Error! Bookmark not defined.
12.2	Làm việc với MySQL	Error! Bookmark not defined.
Bài 13:	Thread - multi thread.....	Error! Bookmark not defined.
Bài 15:	Regular Expression	Error! Bookmark not defined.
Bài 16:	Xây dựng ứng dụng GUI.....	Error! Bookmark not defined.
16.1	Tkinter.....	Error! Bookmark not defined.
Bài 17:	Một số ứng dụng Python	Error! Bookmark not defined.
17.1	Thư viện NumPy	Error! Bookmark not defined.
17.1.1	Giới thiệu.....	Error! Bookmark not defined.
17.1.2	Một số hàm thường sử dụng của numpy	Error! Bookmark not defined.
17.1.3	Truy nhập phần tử của mảng trong numpy.....	Error! Bookmark not defined.
17.1.4	Các phép tính trên mảng dữ liệu numpy.....	Error! Bookmark not defined.
17.2	Làm việc với Matplotlib để vẽ đồ thị.....	Error! Bookmark not defined.
TÀI LIỆU THAM KHẢO		158

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 luôn nằm trong top 10 ngôn ngữ lập trình phổ biến nhất ở tất cả các bảng xếp hạng lớn (TIOBE, RedMonk, PYPL), đó là một minh chứng mạnh mẽ để tuyên bố rằng Python có tốc độ rất nhanh, mạnh mẽ và có mặt ở khắp mọi nơi.

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.

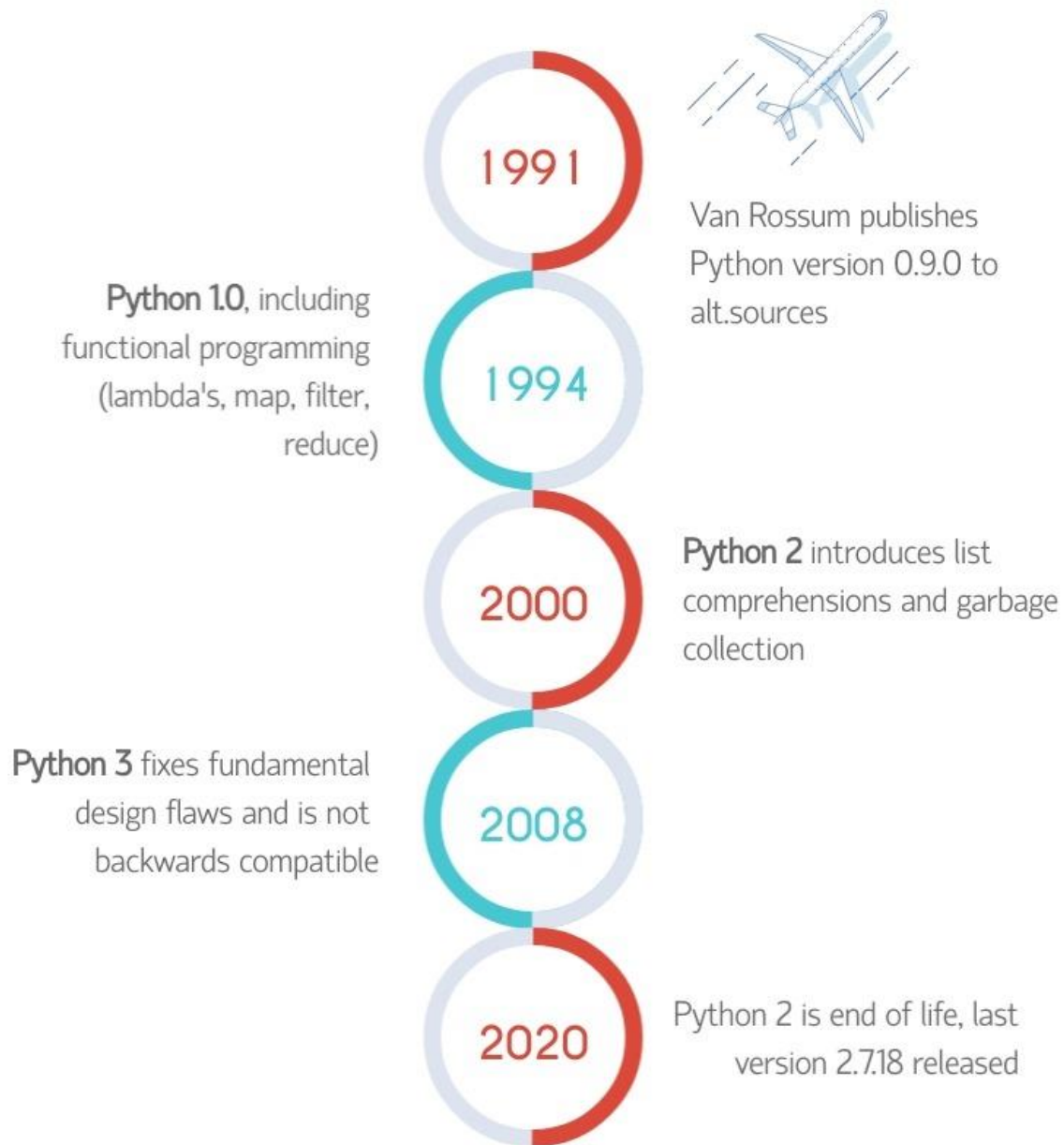
Python 2.1 ra đời kế thừa từ Python 1.6.1 và Python 2.0. Bản quyền của phiên bản này được đổi thành Python Software Foundation License. Từ thời điểm này trở đi, Python thuộc sở hữu của Python Software Foundation (PSF), một tổ chức phi lợi nhuận được thành lập theo mẫu Apache Software Foundation.

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

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 và khả năng tuyệt vời thì cú pháp đơn giản và dễ sử dụng của nó là lý do chính cho việc 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.

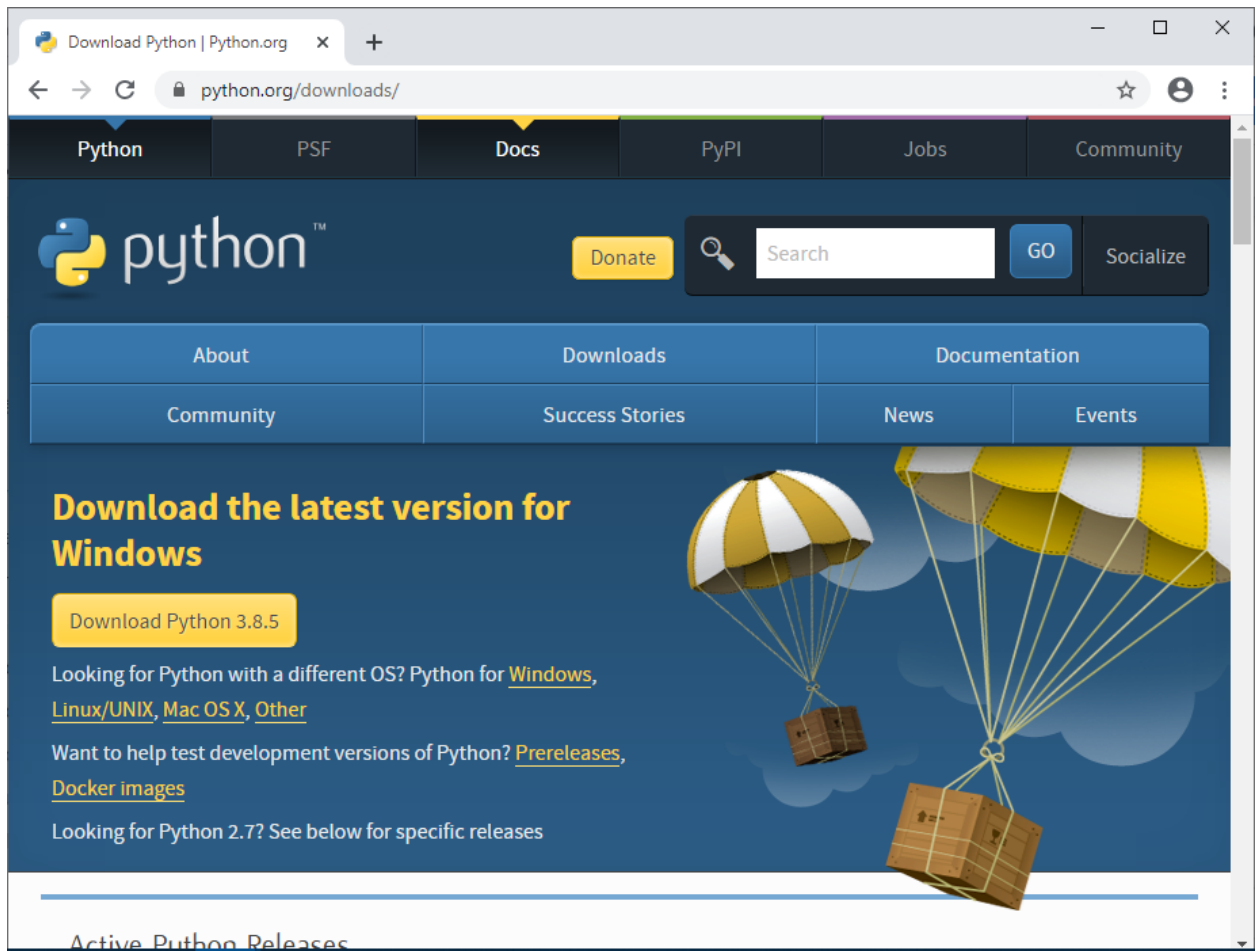
Bạn có thể tải và cài đặt **python-xyz.exe** tại website

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

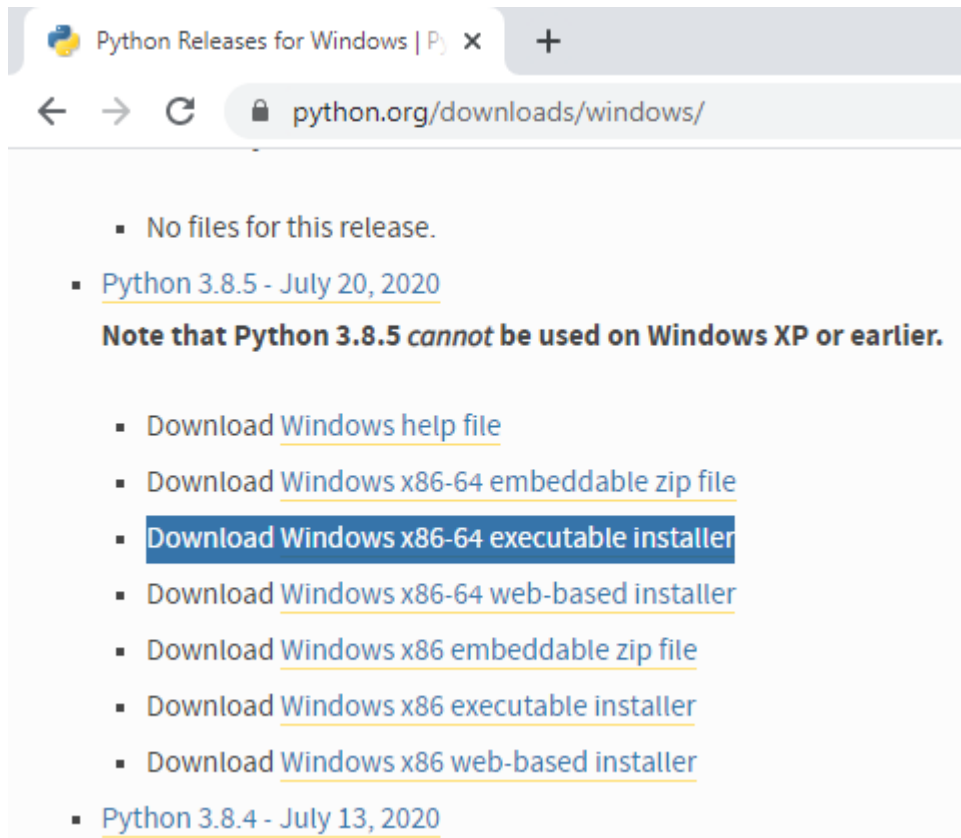
1.3.1.1 [Bước 1: Downloading Python](#)

1. 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.




2. Chọn mục **Download** → **Windows**.



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

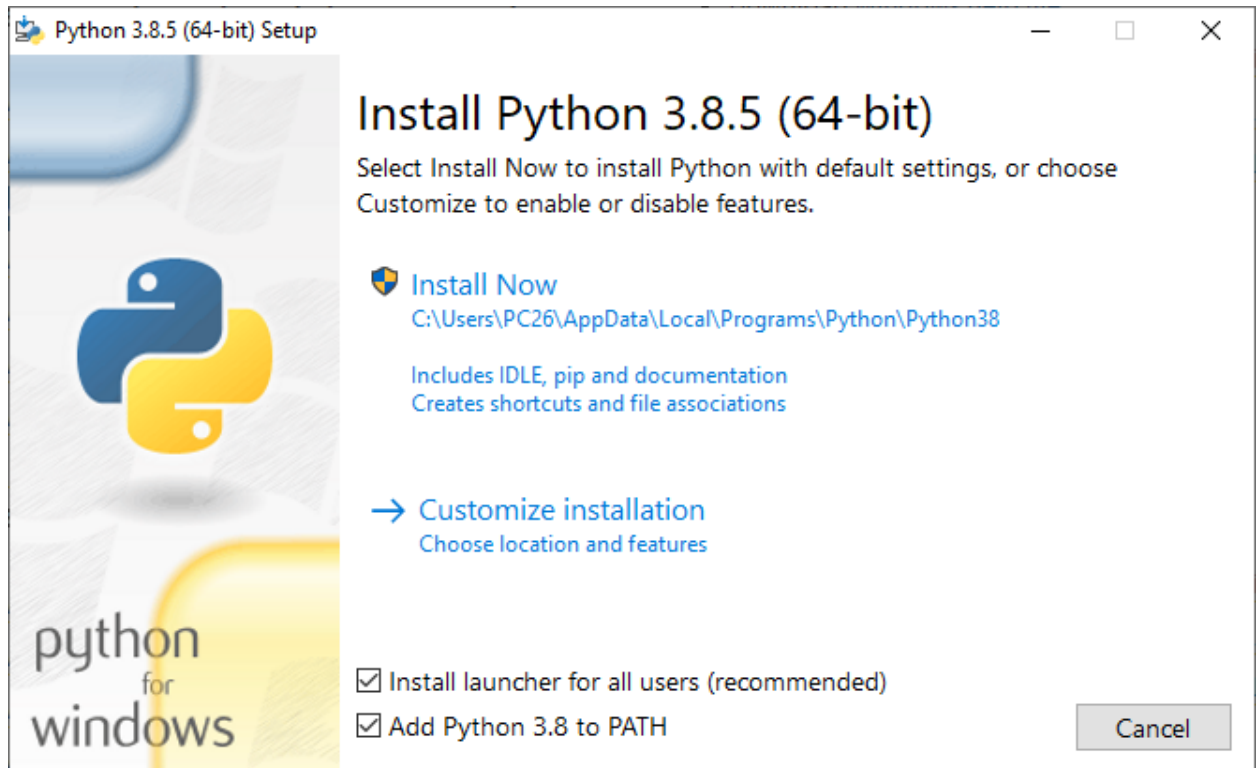
3. Click **Download python 3.8.5.amd64.exe** .

 python-3.8.5-amd64.exe ^

4. Sau đó mở file **python-3.8.5.amd64.exe** để bắt đầu **Installing**.

1.3.1.2 Bước 2: Cài đặt Python

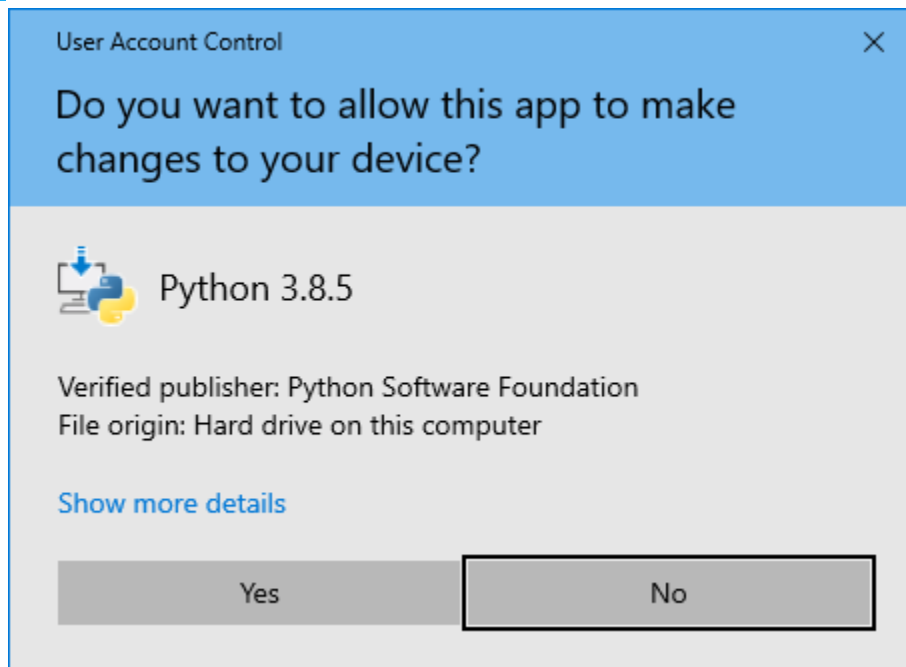
1. Double-click file cài đặt **python-3.8.5.exe**. Một màn hình xuất hiện.



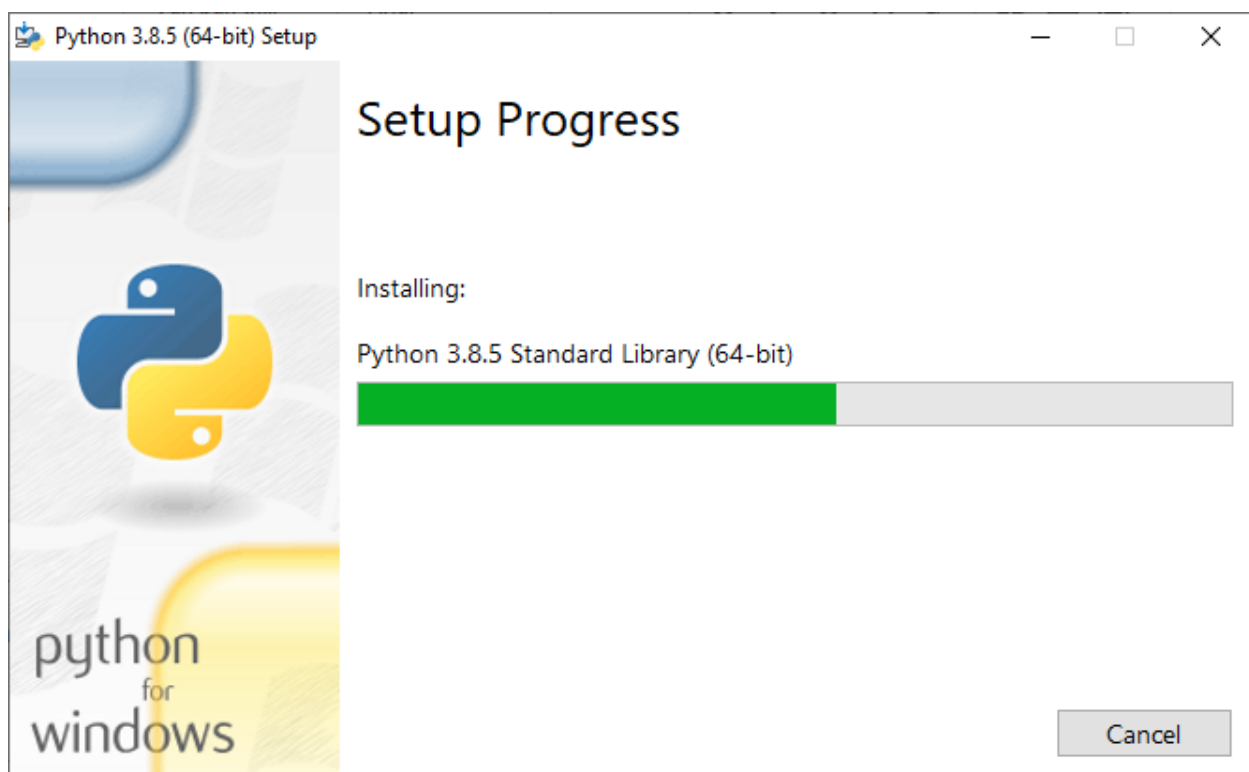
Nên chọn cài Python cho toàn bộ user **Install launcher for all users (recommended)** và chọn **Add Python 3.8 to PATH** để tự setup đường dẫn PATH.

2. Click chọn **Install Now**.

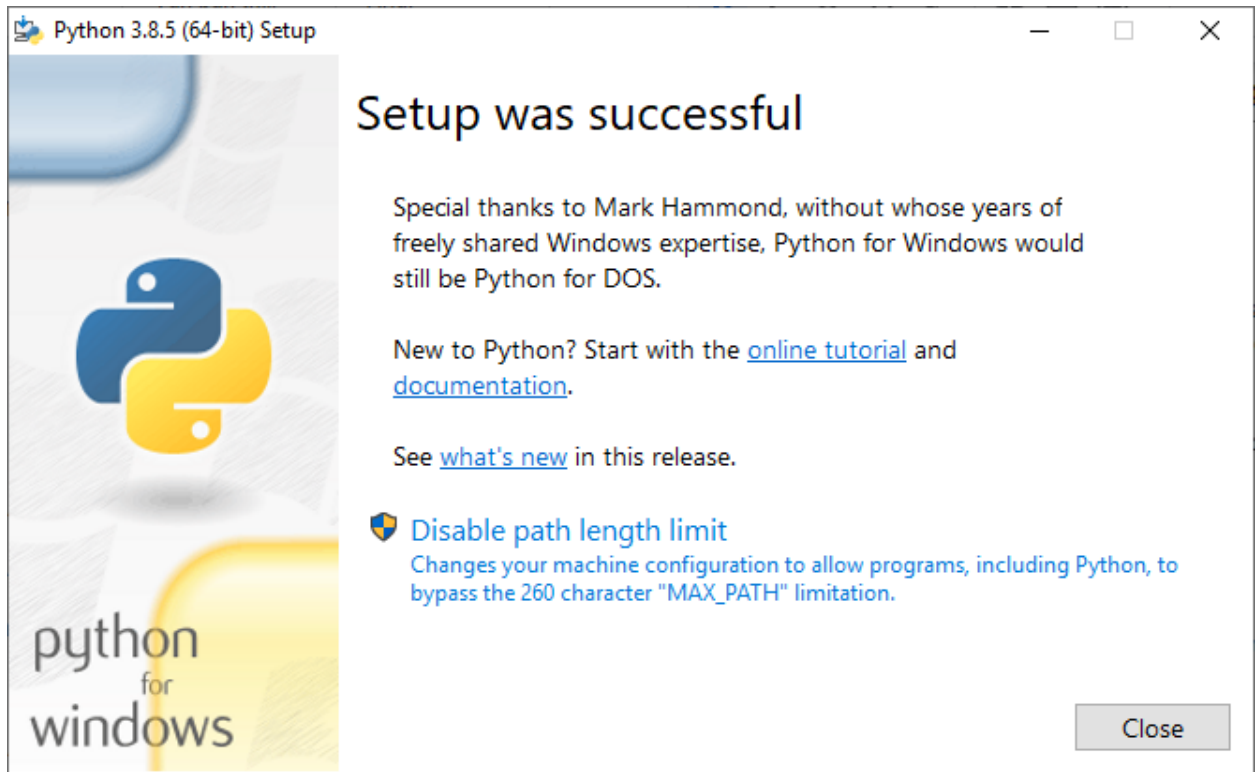
Khi chạy, cửa sổ **User Account Control** sẽ pop-up để xác nhận **Do you want to allow this app to make changes to your device**.



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



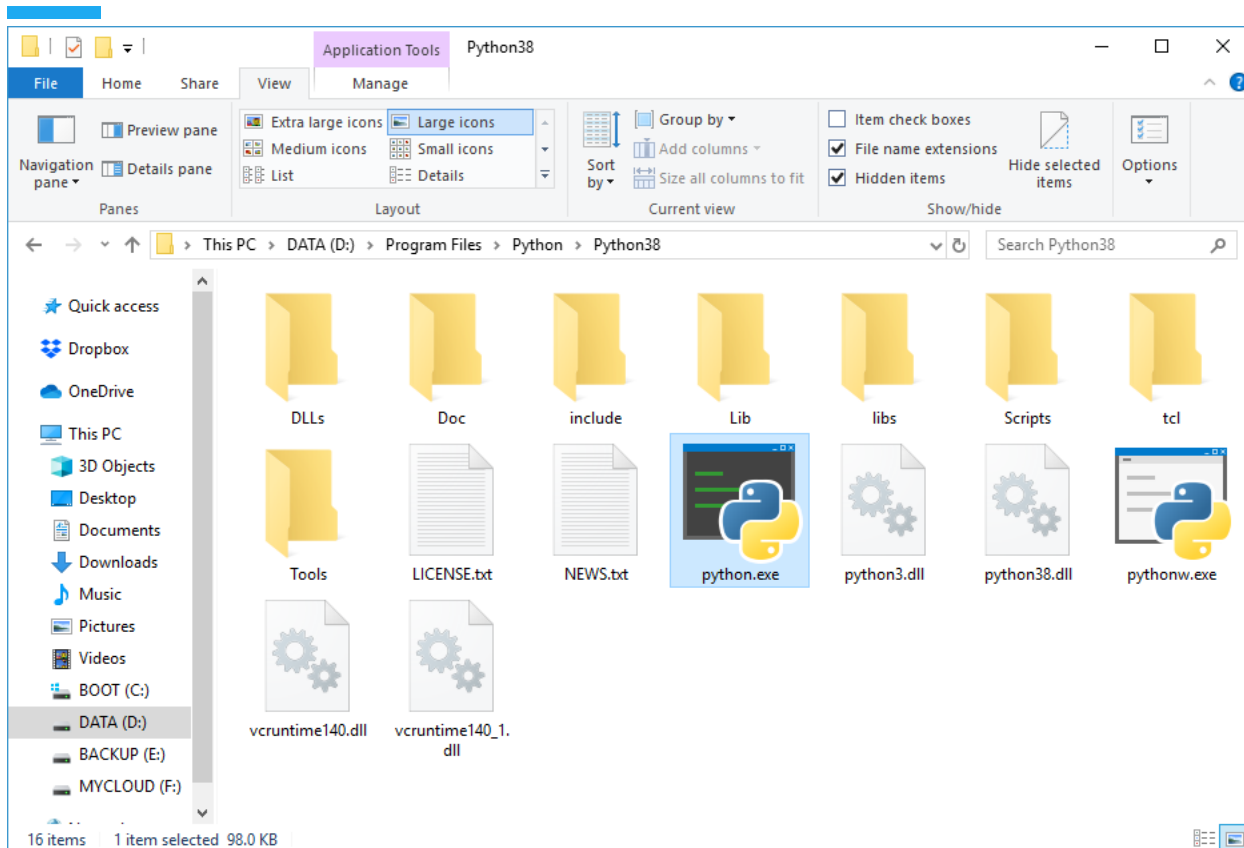
Cuối cùng thông báo cài thành công **Setup was successful..**



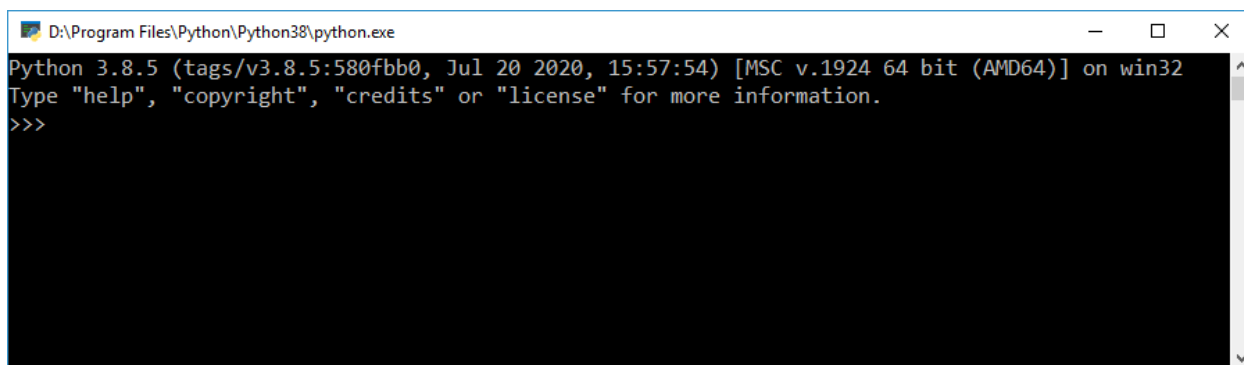
4. Click **Close** để kết thúc.

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.

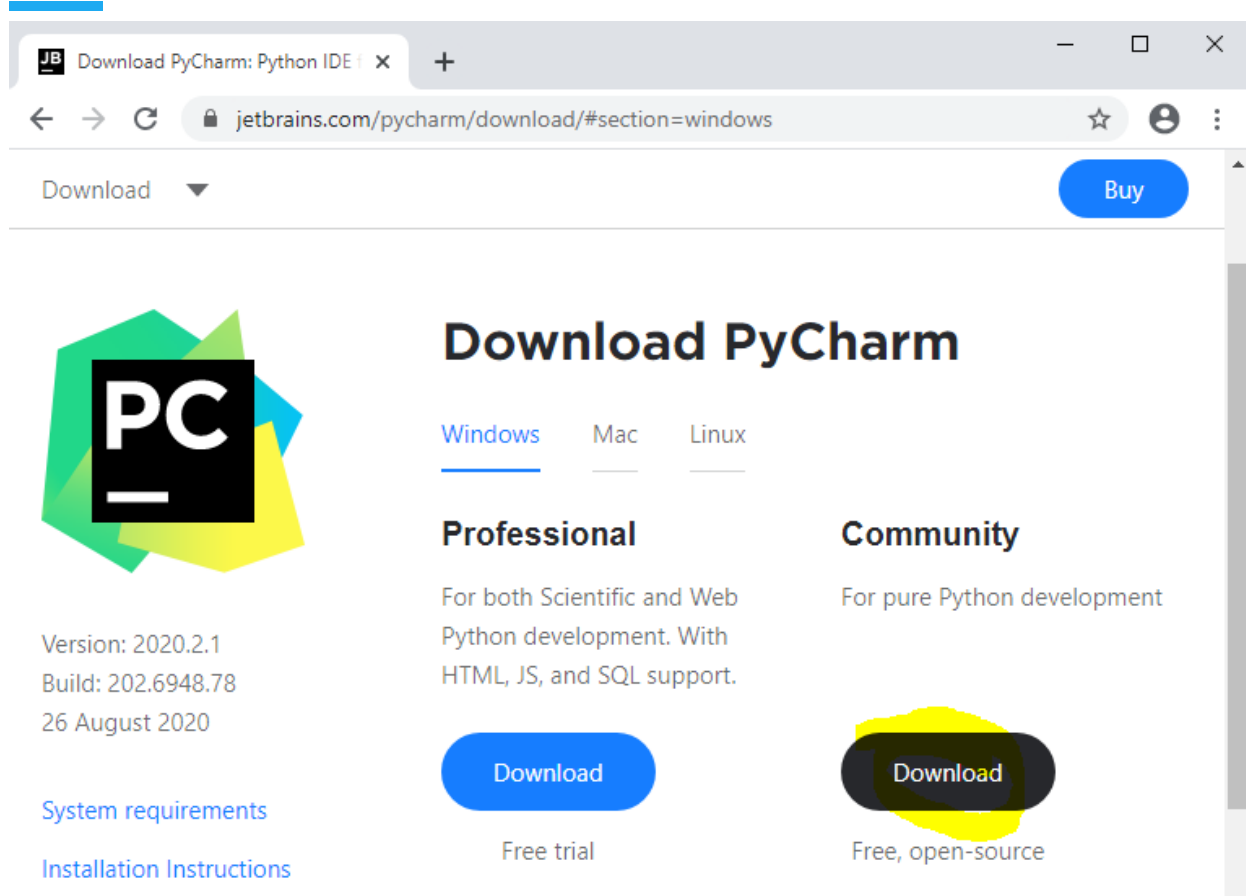
1.3.2.1 Bước 1: Download và cài đặt PyCharm IDE

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

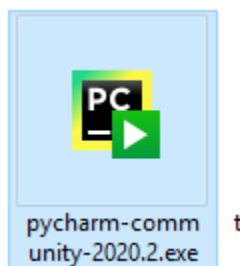
- Ta vào website www.jetbrains.com/pycharm
- Download để tải PyCharm IDE về máy tính cá nhân như hình bên dưới.

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.

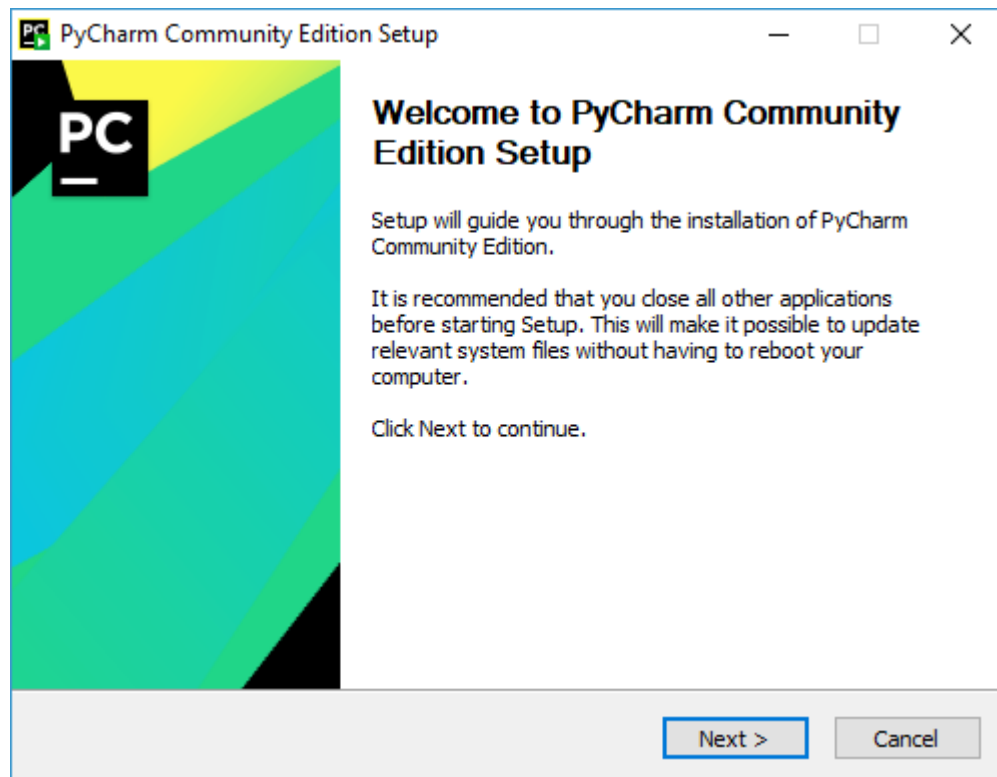


Sau khi download thành công, PyCharm sẽ được lưu tại một thư mục Download của máy tính.



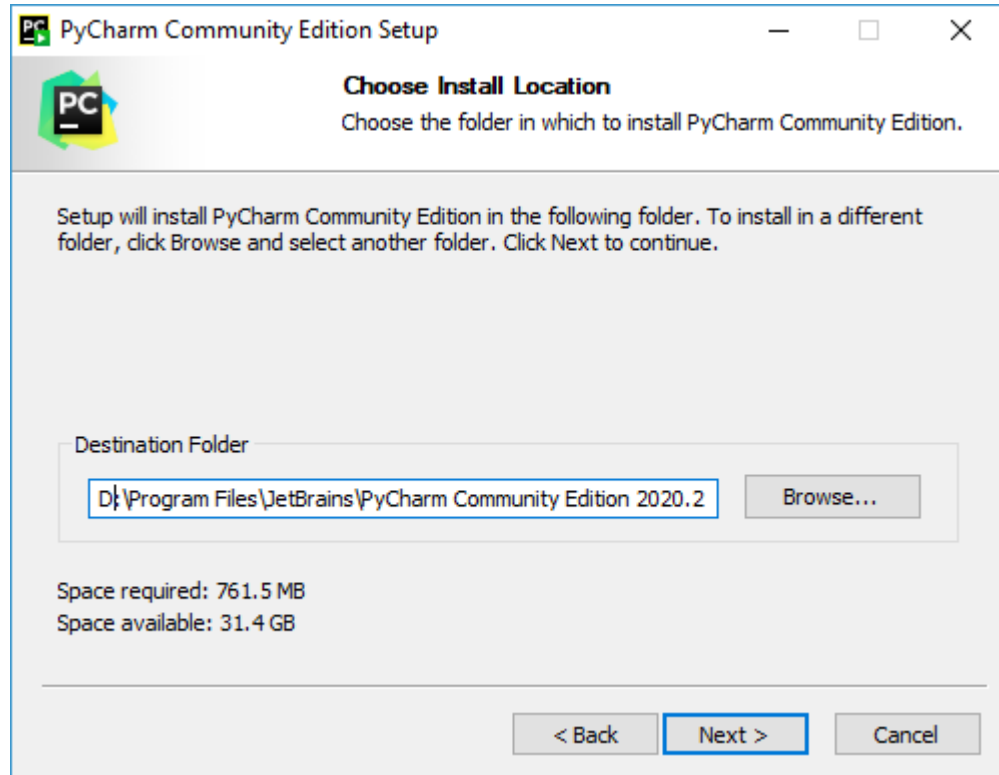
Ta click đúp lên file cài, để tiến hành cài đặt PyCharm.

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



Giao diện cài Đặt PyCharm đầu tiên - Chọn Next

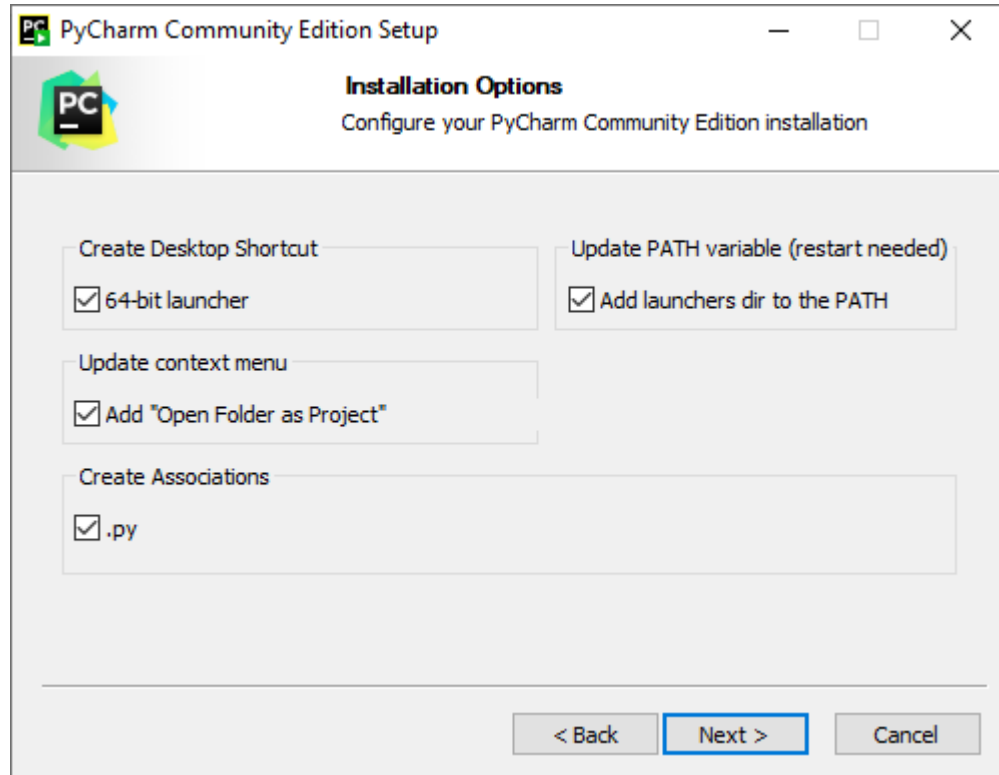
Sau đó, ta chọn đường dẫn thư mục chứa bộ cài nói trên.



Chọn đường dẫn lưu PyCharm

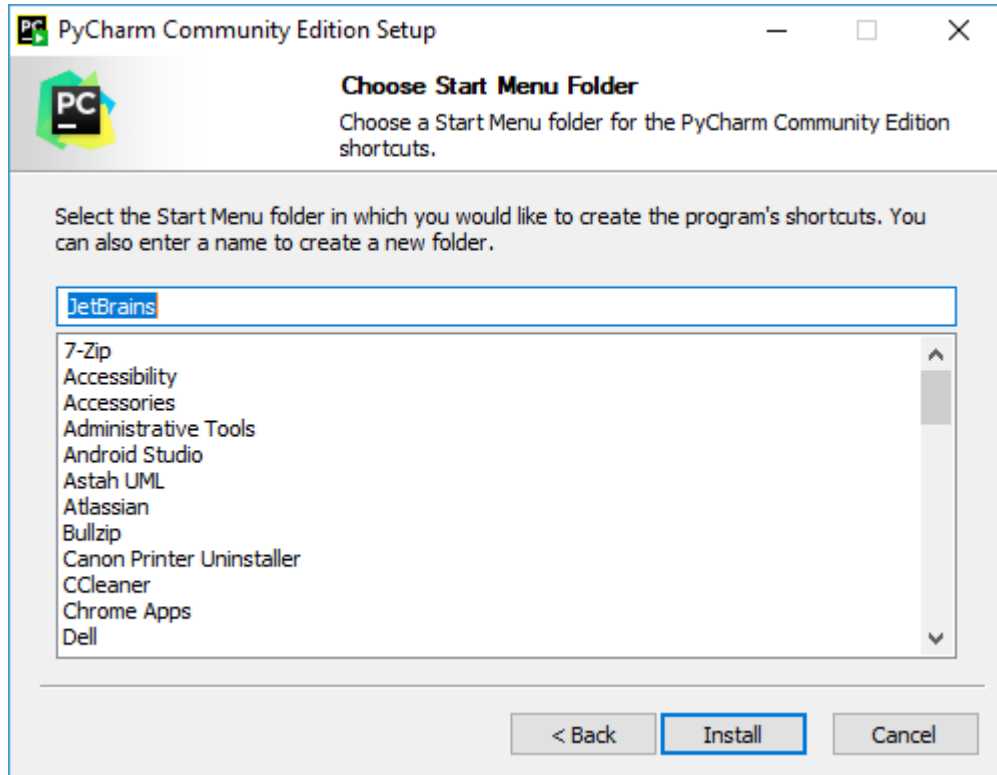
Tiếp theo, ta chọn các tùy chọn cho việc cài đặt.

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

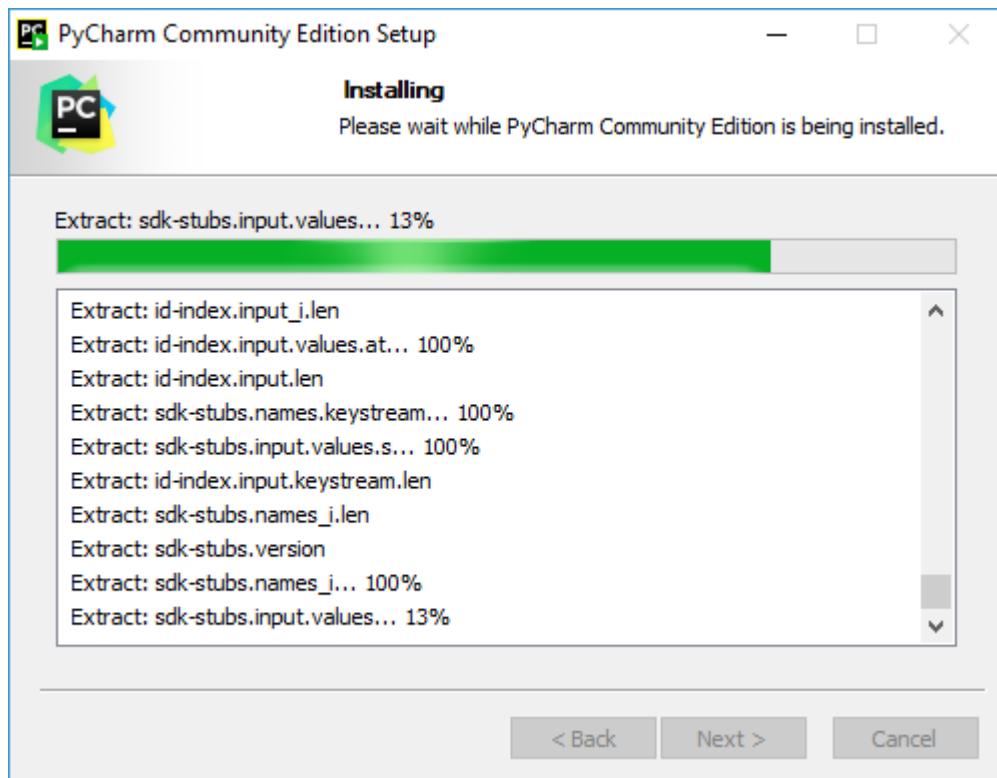


Chọn cài Tùy chọn Cài Đặt - Lựa chọn Như Hình nếu chưa cài Java, OS 64 Bit

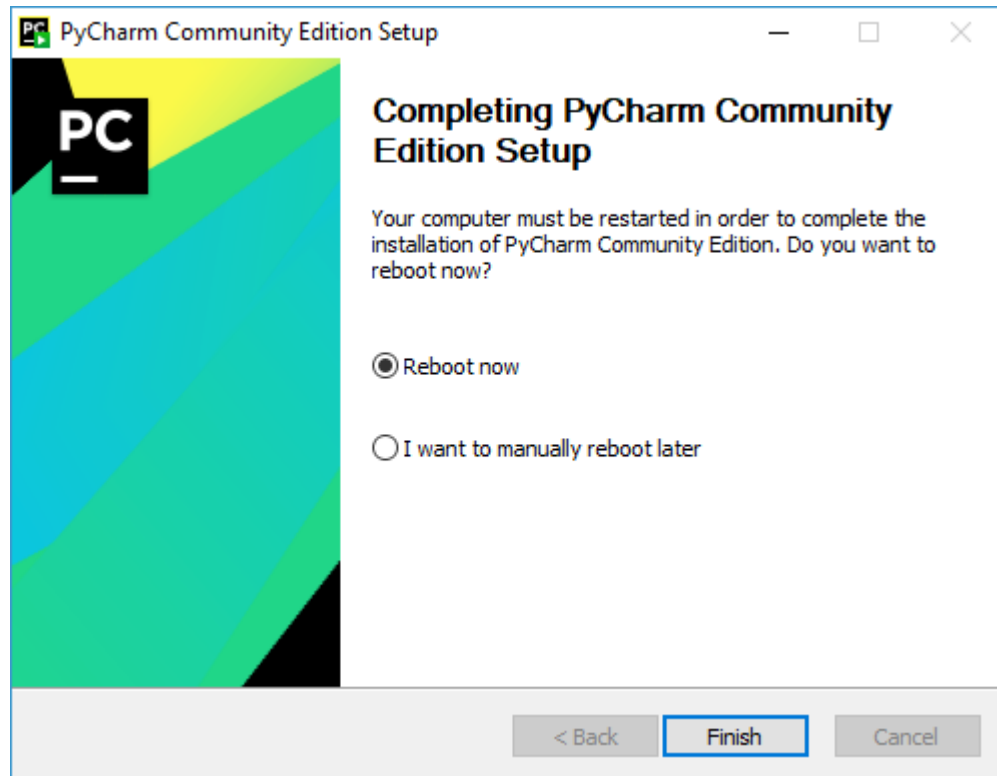
Sau đó ta chọn Install trong màn hình tiếp theo, để bắt đầu tiến hành cài đặt PyCharm.



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.

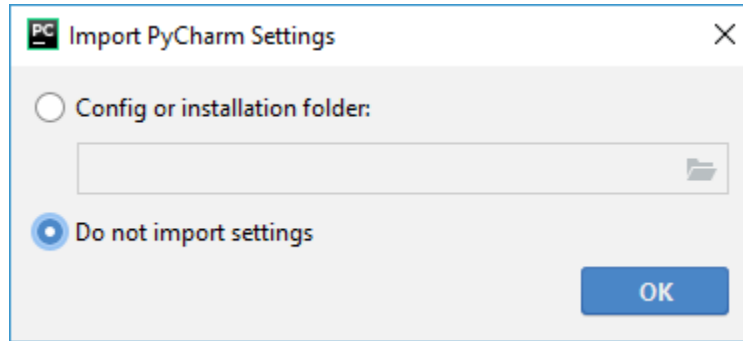


Chọn Reboot now để khởi động lại và hoàn tất cài đặt PyCharm

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?".

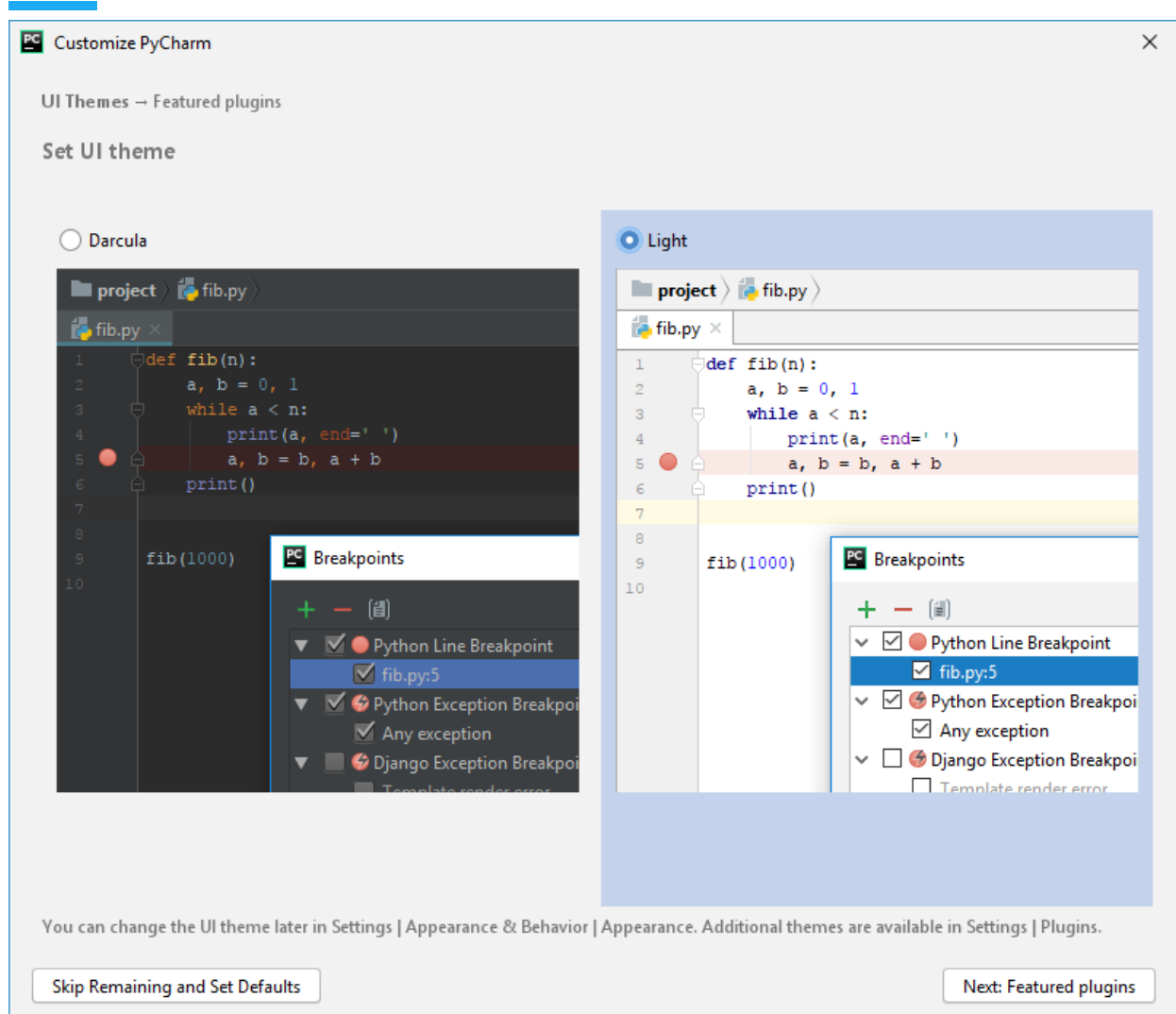
Nếu cài mới hoàn toàn, ta chọn mục **Do not import settings**, rồi nhấn OK.



Chọn Có hay Không Import các cài đặt trước

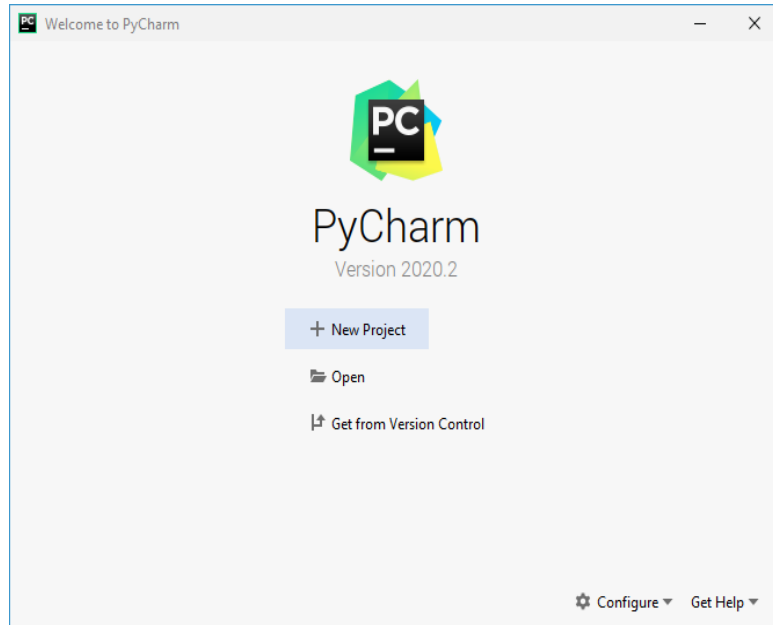
Trong phần chính sách bảo mật, ta nhấn xác nhận và nhấn Continue để tiếp tục.

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



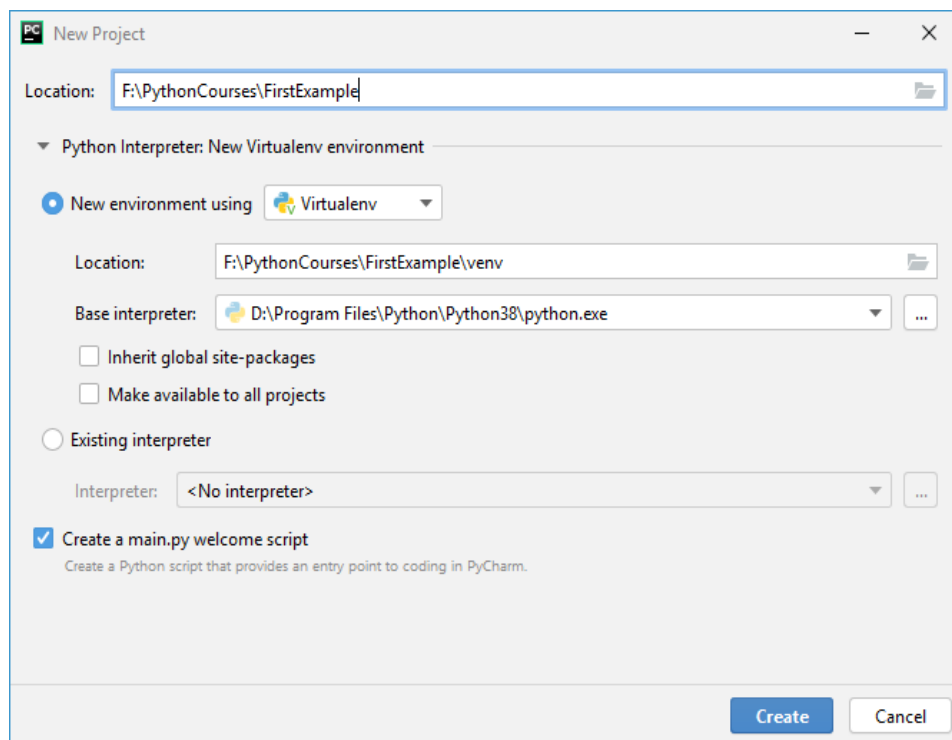
Sử dụng các thiết lập mặc định của PyCharm

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



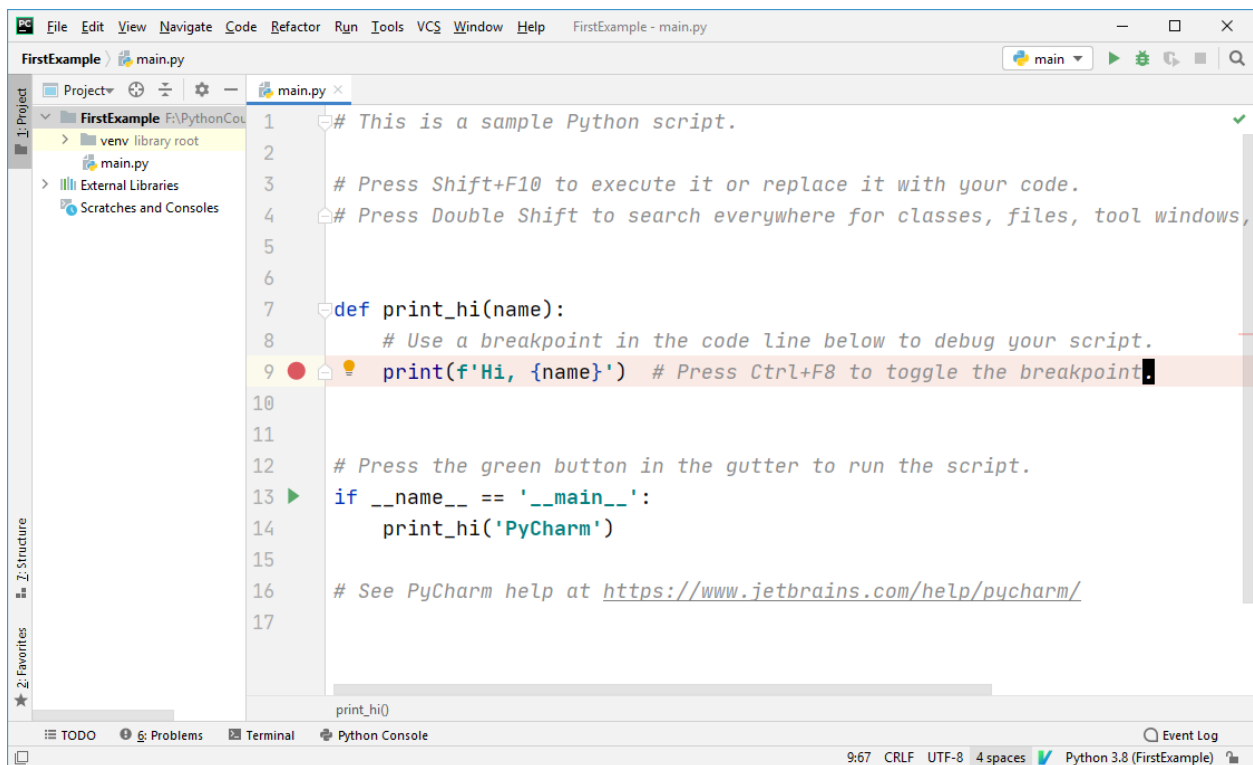
Tạo 1 Project mới trong PyCharm

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 đã 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 ten từ bàn phím
name = input("Nhập họ ten")
```

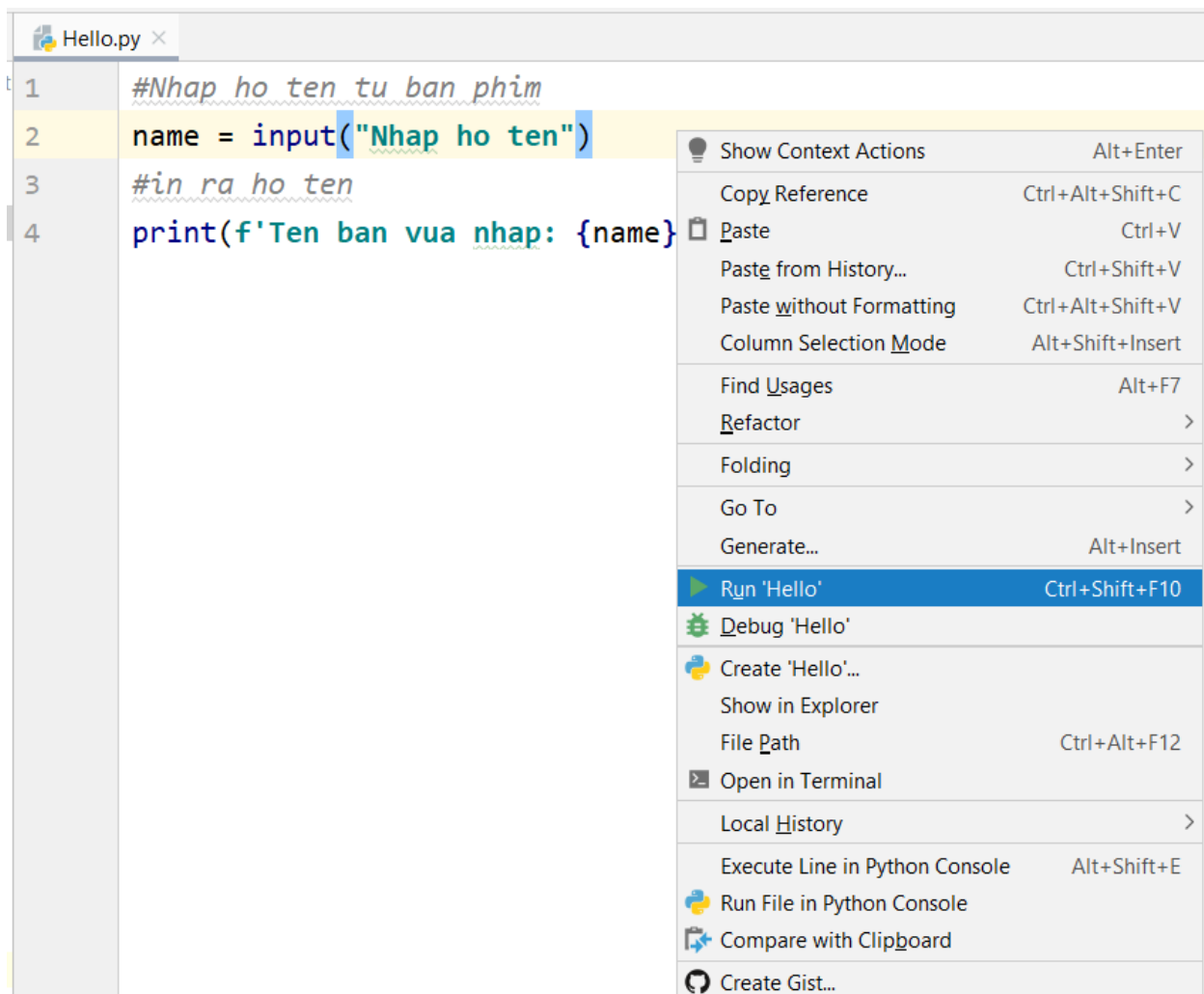


```
#in ra ho ten
print(f'Ten ban vua nhap: {name}')
```

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

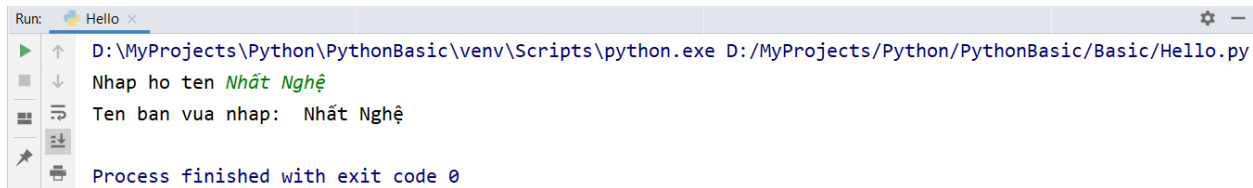
Sau khi đã viết mã xong, ta click phải chuột lên cửa sổ soạn thảo, rồi chọn mục **Run** để thực thi file.

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 x
D:\MyProjects\Python\PythonBasic\venv\Scripts\python.exe D:/MyProjects/Python/PythonBasic/Basic/Hello.py
Nhap ho ten Nhất Nghệ
Ten ban vua nhap: Nhất Nghệ
Process finished with exit code 0
```

Kết quả chạy thử nghiệm File Python bằng PyCharm

1.4 Phân khối và chú thích

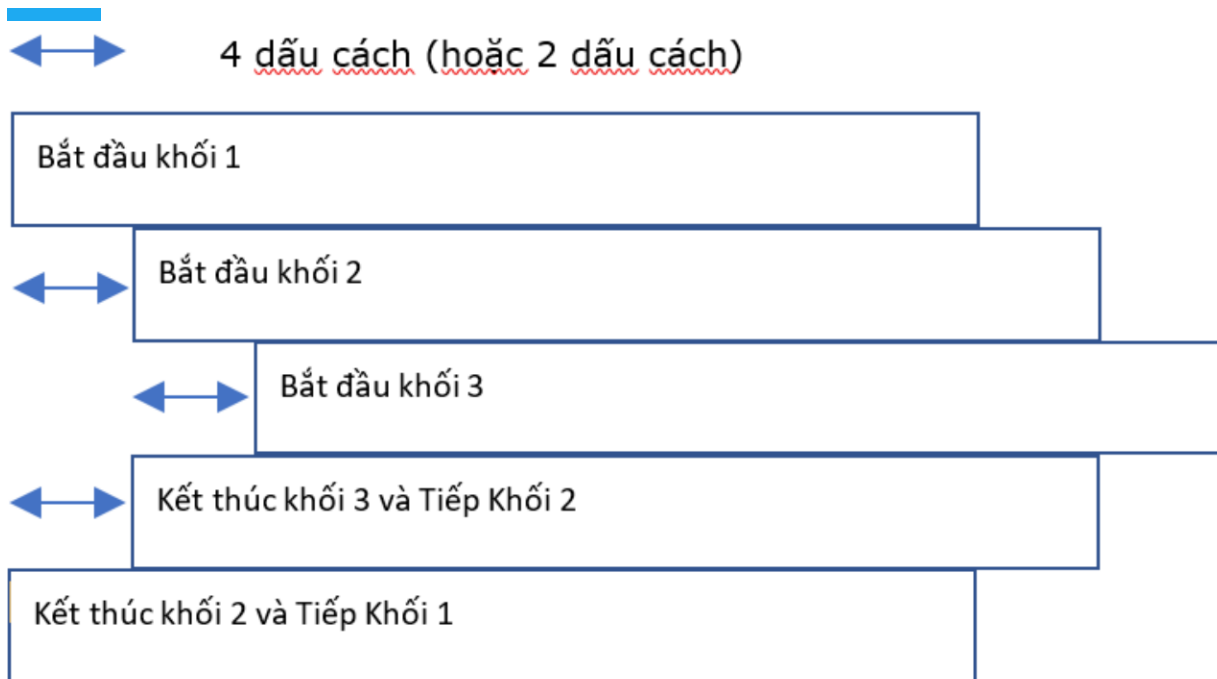
1.4.1 Phân khối

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, 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.



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	numbers = range(1,11) for number in numbers: if number == 7: continue

		the next cycle of the nearest enclosing loop.	
7	global	<p>global statement allows us to modify the variables outside the current scope.</p> <p>This keyword is used to define a variable inside the function to be of a global scope.</p>	<pre>x = 0 def add(): global x x = x + 10 add() print(x) # 10</pre>
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.	<pre>def foo(): pass</pre>
9			

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") # prints "x is even"
12	raise	The raise statement is used to throw exceptions in the program.	def square(x): if type(x) is not int: raise TypeError("Require int argument") print(x * x)
13	and	Boolean operator for and operation.	x = True y = False print(x and y) # False
14	del	The del keyword is used to delete objects such as variables, list, objects, etc.	s1 = "Hello" print(s1) # Hello del s1 print(s1) # NameError: name 's1' is not defined
15	import	The import statement is used to import modules and	# importing class from a module from collections import OrderedDict # import module import math

		classes into our program.	
16	return	The return statement is used in the function to return a value.	def add(x,y): return x+y
17	as	Python as keyword is used to provide name for import, except, and with statement.	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
18	elif	The elif statement is always used with if statement for “else if” operation.	x = 10 if x > 10: print('x is greater than 10') elif x > 100: print('x is greater than 100') elif x == 10: print('x is equal to 10') else: print('x is less than 10')
19	in	Python in keyword is used to test membership.	l1 = [1, 2, 3, 4, 5] if 2 in l1: print('list contains 2') s = 'abcd' if 'a' in s: print('string contains a')

20	try	Python try statement is used to write exception handling code.	<code>x = " try: i = int(x) except ValueError as ae: print(ae) # invalid literal for int() with base 10: "</code>
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.	<code>def divide(a, b): assert b != 0 return a / b</code>
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.	<code>if False: pass else: print('this will always print')</code>

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) i+=1 # Output # 0 # 1 # 2 </pre>
25	async	New keyword introduced in Python 3.5. This keyword is always used in couroutine 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('askpython.com'), ping('python.org'),) if __name__ == '__main__': then = time.time() loop = asyncio.get_event_loop() loop.run_until_complete(main()) now = time.time() print(f'Execution Time = {now - then}') # Output Ping Started for askpython.com Ping Started for python.org Ping Finished for askpython.com Ping Finished for python.org Execution Time = 1.004091739654541 </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.	<code>multiply = lambda a, b: a * b</code> <code>print(multiply(8, 6)) # 48</code>
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>def division(x, y):</code> <code>try:</code> <code>return x / y</code> <code>except</code> <code>ZeroDivisionError as e:</code> <code>print(e)</code> <code>return -1</code> <code>finally:</code> <code>print('this will always execute')</code> <code>print(division(10, 2))</code>

		code in finally block is always executed. It's mainly used to close resources.	<code>print(division(10, 0))</code> # Output this will always execute 5.0 division by zero this will always execute -1
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>
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 multiplyByTen(*kwargs): for i in kwargs: yield i * 10 a = multiplyByTen(4, 5,) # a is a generator object, an iterator # showing the values for i in a: print(i) # Output 40 50</pre>
33	break	The break statement is used with nested "for" and "while" loops. It stops the	<pre>number = 1 while True: print(number) number += 2 if number > 5: break print(number) # never executed # Output 1 3 5</pre>

		current loop execution and passes the control to the start of the loop.	
34	for	Python for keyword is used to iterate over the elements of a sequence or iterable object.	s1 = 'Hello' for c in s1: print(c) # Output H e l l o
35	not	The not keyword is used for boolean not operation.	x = 20 if x is not 10: print('x is not equal to 10') x = True print(not x) # False

Python luôn được cập nhật, 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 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

Input/Output

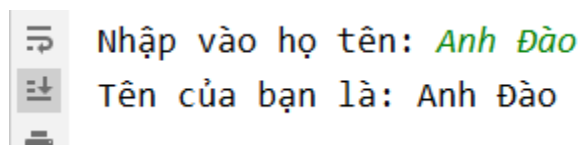
Function	Description
<code>format()</code>	Converts a value to a formatted representation
<code>input()</code>	Reads input from the console
<code>open()</code>	Opens a file and returns a file object
<code>print()</code>	Prints to a text stream or the console

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)
```



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

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:

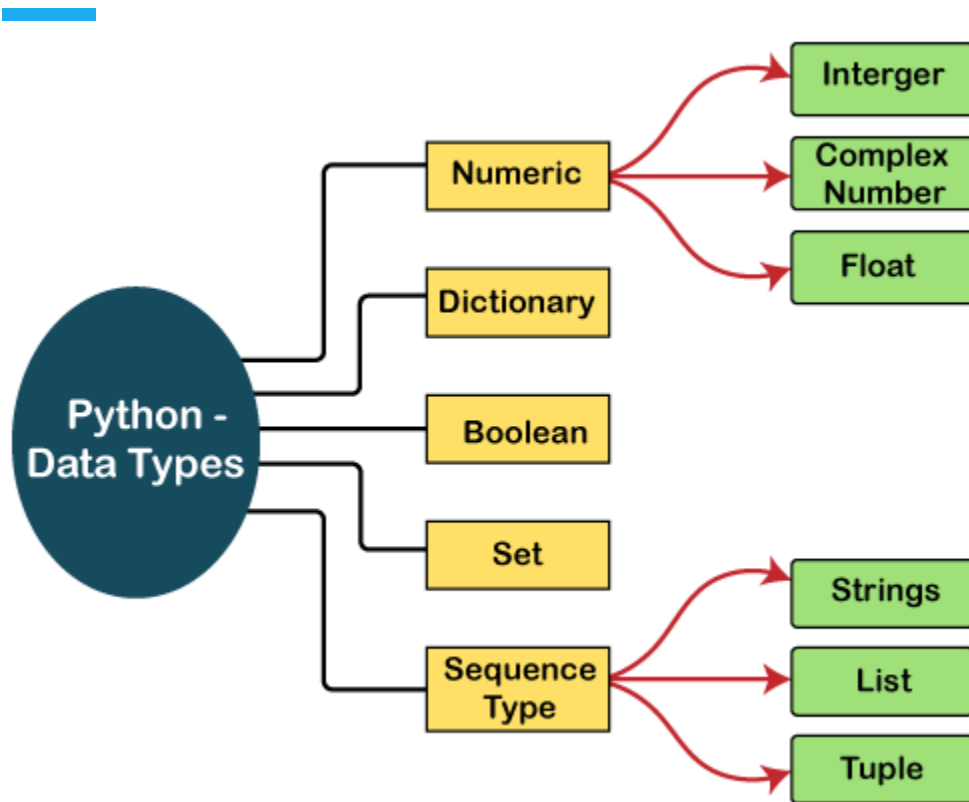
```
print(f'{x} + {y} = {z}')
print(f'{x} + {y} = {x+y}')
```


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:

Text Type:	<code>str</code>
Numeric Types:	<code>int</code> , <code>float</code> , <code>complex</code>
Sequence Types:	<code>list</code> , <code>tuple</code> , <code>range</code>
Mapping Type:	<code>dict</code>
Set Types:	<code>set</code> , <code>frozenset</code>
Boolean Type:	<code>bool</code>
Binary Types:	<code>bytes</code> , <code>bytearray</code> , <code>memoryview</code>



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()`

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

Ví dụ:

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

2.1.3 Kiểu số

Đây là kiểu dữ liệu sử dụng nhiều nhất. Python hỗ trợ số nguyên, số thập phân và số phức, chúng lần lượt được định nghĩa là các lớp `int`, `float`, `complex` trong Python.

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.

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

2.1.4 Kiểu chuỗi

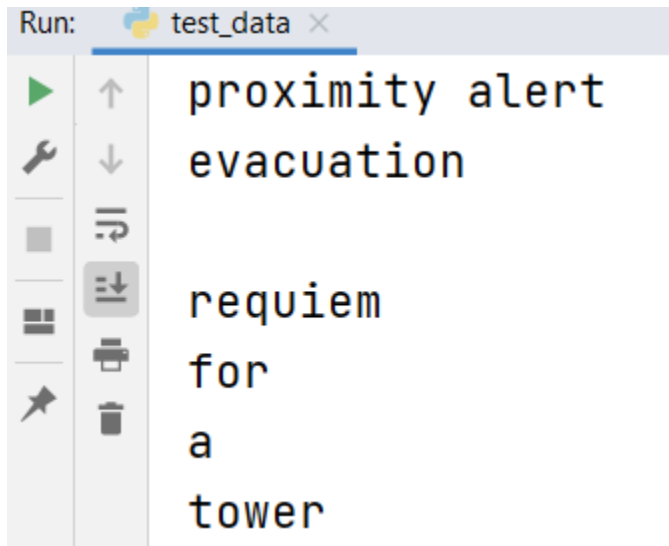
String là kiểu dữ liệu lưu trữ văn bản.

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.

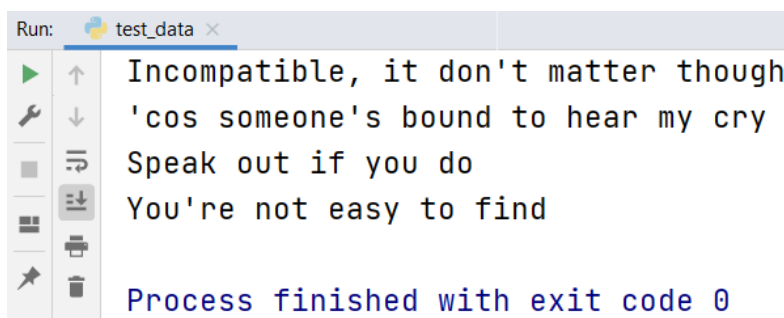


```
Run: test_data x
proximity alert
evacuation
requiem
for
a
tower
```

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.



```
Run: test_data x
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 Pytooo
```

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 để 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 x
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

Lâu nay người ta thường dịch boolean là kiểu “luận lý”, đối với dân kỹ thuật thì cái từ này nghe hơi khó hiểu, nên mình xin mạn phép dịch là kiểu “đúng sai” cho đơn giản 😊 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.

```
1 import random
```

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

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

Và ở đây chúng ta sẽ sử dụng phương thức `randint(x, y)`, phương thức này sẽ trả về giá trị ngẫu nhiên từ $x \rightarrow y$, ở đây là $0 \rightarrow 1$. 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.

```
1 print (bool(True))
2 print (bool(False))
3 print (bool("text"))
4 print (bool(""))
5 print (bool(' '))
6 print (bool(0))
7 print (bool())
8 print (bool(3))
9 print (bool(None))
```



```
1     True
2     False
3     True
4     False
5     True
6     False
7     False
8     True
9     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

```
1     def function():
2         pass
3
4     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`.

```
1     None
```

2.1.7 Kiểu Tuple

Kiểu Tuple là kiểu tập hợp nhiều phần tử, kiểu này lưu trữ các phần tử một cách có thứ tự và có thể lưu nhiều kiểu giá trị khác nhau trong một tuple. Giá trị trong tuple không thể thay đổi được.

```
1 fruits = ("oranges", "apples", "bananas")
2 fruits = "oranges", "apples", "bananas"
3 print (fruits)
```

Ở trên là hai cách tạo tuple.

```
1 first = (1, 2, 3)
2 second = (4, 5, 6)
3
4 print ("len(first) : ", len(first))
5 print ("max(first) : ", max(first))
6 print ("min(first) : ", min(first))
7 print ("first + second :", first + second)
8 print ("first * 3 : ", first * 3)
9 print ("1 in first : ", 1 in first)
10 print ("5 not in second : ", 5 not in second)
```

Đoạn code trên ví dụ về các thao tác với một tuple. Hàm `len()` có tác dụng lấy số lượng phần tử trong một tuple. Hàm `max()` và `min()` lấy giá trị lớn nhất và nhỏ nhất của tuple. Toán tử `+` gộp 2 tuple với nhau. Toán tử `*` tạo ra thêm n tuple. Toán tử `in` tìm xem một giá trị có nằm trong một tuple hay không.

```
1 len(first) : 3
2 max(first) : 3
3 min(first) : 1
4 first + second : (1, 2, 3, 4, 5, 6)
5 first * 3 : (1, 2, 3, 1, 2, 3, 1, 2, 3)
6 1 in first : True
```

```
7     5 not in second : False
```

Tiếp theo chúng ta tìm hiểu về cách truy xuất phần tử.

```
1     five = (1, 2, 3, 4, 5)
2
3     print ("five[0] : ", five[0])
4     print ("five[-1] : ", five[-1])
5     print ("five[-2] : ", five[-2])
6     print ("five[:] : ", five[:])
7     print ("five[0:4] : ", five[0:4])
8     print ("five[1:2] : ", five[1:2])
9     print ("five[:2] : ", five[:2])
10    print ("five[:-1] : ", five[:-1])
11    print ("five[:9] : ", five[:9])
```

Để lấy một phần tử nào đó trong tuple, chúng ta sử dụng cặp dấu [], và cũng giống như các ngôn ngữ lập trình khác, chỉ số trong một tuple bắt đầu từ 0. Tức là nếu trong tuple có 5 phần tử, thì các phần tử được đánh số từ 0..4. Bạn cũng có thể dùng chỉ số là số nguyên âm, và Python sẽ tự động lấy lùi, chẳng hạn nếu bạn đưa chỉ số là -1, bạn sẽ được phần tử thứ 4. Bạn cũng có thể lấy các đoạn giá trị cố định dùng dấu hai chấm, ví dụ lấy các phần tử là 1, 2, 3, 4 là [1:4], ngoài ra nếu bạn không đưa chỉ số bắt đầu hay chỉ số kết thúc, bạn sẽ lấy được tất cả các phần tử về trước hay về sau đó, ví dụ [:3] sẽ cho ra các phần tử 0, 1, 2, 3.

```
1 five[0] : 1
2 five[-1] : 5
3 five[-2] : 4
4 five[:] : (1, 2, 3, 4, 5)
5 five[0:4] : (1, 2, 3, 4)
6 five[1:2] : (2,)
7 five[:2] : (1, 2)
8 five[:-1] : (1, 2, 3, 4)
9 five[:9] : (1, 2, 3, 4, 5)
```

Tuple có thể lưu trữ các kiểu dữ liệu khác nhau một cách linh hoạt.

```
1 mix = (1, 2, "solaris", (1, 2, 3))
2
3 print ("mix[1] :", mix[1])
4 print ("mix[2] :", mix[2])
5 print ("mix[3] :", mix[3])
6 print ("mix[3][0] :", mix[3][0])
7 print ("mix[3][1] :", mix[3][1])
8 print ("mix[3][2] :", mix[3][2])
```

Trong ví dụ trên, tuple của chúng ta có số nguyên, string và cả một tuple khác.

```
1 mix[1] : 2
2 mix[2] : solaris
3 mix[3] : (1, 2, 3)
4 mix[3][0] : 1
```

```
5     mix[3][1] : 2
```

```
6     mix[3][2] : 3
```

Để truy xuất phần tử của tuple bên trong một tuple, bạn dùng hai cặp dấu `[]`.

2.1.8 Kiểu List

Kiểu list cũng là một kiểu lưu các giá trị tuần tự, một list cũng có thể lưu nhiều giá trị khác nhau. Do đó list và tuple có nhiều điểm tương đồng. Điểm khác biệt giữa list và tuple là các phần tử trong list có thể thay đổi giá trị, ngoài ra list có một số phương thức mà tuple không có. Chúng ta sẽ có nguyên một bài để nói riêng về kiểu list.

```
1     actors = ["Jack Nicholson", "Antony Hopkins", "Adrien Brody"]
```

Để tạo một list thì ta dùng cặp ký tự `[]`.

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

```
2
```

```
3     print (num[0])
```

```
4     print (num[2:])
```

```
5     print (len(num))
```

```
6     print (num + [8, 9])
```

Như đã nói ở trên, chúng ta có thể dùng các thao tác cộng trừ, lấy chỉ số... như của tuple.

```
1     0
```

```
2     [5, 4, 6, 7]
```

```
3     6
```

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

Tiếp theo chúng ta tìm hiểu về một số thao tác của riêng list. Bắt đầu là thao tác sắp xếp một list.

```

1  numbers = [4, 3, 6, 1, 2, 0, 5 ]
2
3  print (numbers)
4  numbers.sort()
5  print (numbers)

```

Để sắp xếp một list thì chúng ta dùng phương thức `sort()`. 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.

```

1  [4, 3, 6, 1, 2, 0, 5]
2  [0, 1, 2, 3, 4, 5, 6]

```

Phương thức `reverse()` cũng sắp xếp các phần tử nhưng theo thứ tự ngược lại.

```

1  numbers.reverse()  # [5, 4, 3, 2, 1, 0]

```

Chúng ta có thể đếm một phần tử xuất hiện bao nhiêu lần trong một list bằng phương thức `count()`.

```

1  numbers = [0, 0, 2, 3, 3, 3, 3]
2
3  print ("zero is here ", numbers.count(0), "times")
4  print ("one is here ", numbers.count(1), "times")
5  print ("two is here ", numbers.count(2), "time")
6  print ("three is here ", numbers.count(3), "times")

```

Trong ví dụ trên chúng ta đếm các số 0, 1, 2, 3 xuất hiện bao nhiêu lần.

```

1  zero is here  2 times
2

```

```
3     one is here  0 times
4     two is here  1 time

    three is here  4 times
```

Tiếp theo chúng ta tìm hiểu cách thêm, xóa phần tử.

```
1     names = []
2
3     names.append("Frank")
4     names.append("Alexis")
5     names.append("Erika")
6     names.append("Ludmila")
7
8     print (names)
9     names.insert(0, "Adriana")
10    print (names)
11    names.remove("Frank")
12    names.remove("Alexis")
13    del names[1]
14    print (names)
15    del names[0]
16    print (names)
```

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.

Để xóa một phần tử chúng dùng phương thức `remove()` hoặc dùng từ khóa `del`. Phương thức `remove()` xóa một phần tử có giá trị nhất định. Còn từ khóa `del` sẽ xóa phần tử ở một vị trí cụ thể.

```
1      ['Frank', 'Alexis', 'Erika', 'Ludmila']
2      ['Adriana', 'Frank', 'Alexis', 'Erika', 'Ludmila']
3      ['Adriana', 'Ludmila']
4      ['Ludmila']
```

Tiếp theo chúng ta tìm hiểu cách thay đổi một list.

```
1      first = [1, 2, 3]
2      second = [4, 5, 6]
3
4      first.extend(second)
5      print (first)
6
7      first[0] = 11
8      first[1] = 22
9      first[2] = 33
10     print (first)
11
12     print (first.pop(5))
13     print (first)
```

Chúng ta có dùng phương thức `extend()` để nhập một list vào một list khác, dùng phương thức `pop()` để lấy phần tử thứ n .


```
1      [1, 2, 3, 4, 5, 6]
2      [11, 22, 33, 4, 5, 6]
3      6
4      [11, 22, 33, 4, 5]
```

Tiếp theo là cách tìm vị trí xuất hiện của một giá trị nào đó trong list.

```
1      numbers = [0, 1, 2, 3, 3, 4, 5]
2
3      print (numbers.index(1))
4      print (numbers.index(3))
```

Chúng ta dùng phương thức `index()`, giá trị trả về là vị trí xuất hiện của phần tử mà chúng ta tìm, nếu có nhiều phần tử có giá trị giống nhau thì phương thức này in ra vị trí đầu tiên kể từ vị trí số 0.

```
1      1
2      3
```

Tiếp theo là cách chuyển đổi kiểu..

```
1      first = [1, 2, 3]
2      second = (4, 5, 6)
3
4      print (tuple(first))
5      print (list(second))
6
7      print (first)
8      print (second)
```

Chúng ta có thể dùng hàm `tuple()` để lấy một tuple từ một list, hoặc dùng hàm `list()` để lấy một list từ một tuple. Ở đây các list hay tuple nguồn không bị thay đổi mà Python sẽ lấy các giá trị gốc để tạo ra một list hay tuple mới rồi mới trả về cho chúng ta.

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

2.1.9 Kiểu Set

Set là kiểu tập hợp các phần tử không có thứ tự, không có nhiều hơn 2 phần tử có cùng một giá trị. Các phép toán có thể dùng trên set là phép hợp, giao, hiệu... giống như trong toán học.

```
1      set1 = set(['a', 'b', 'c', 'c', 'd'])
2      set2 = set(['a', 'b', 'x', 'y', 'z'])
3
4      print ("set1: " , set1)
5      print ("set2: " , set2)
6      print ("intersection: ", set1 & set2)
7      print ("union: ", set1 | set2)
8      print ("difference: ", set1 - set2)
9      print ("symmetric difference: ", set1 ^ set2)
```

Để tạo một set thì chúng ta dùng hàm `set()`. Trong ví dụ trên có 2 set. Chúng ta thực hiện phép giao 2 set bằng toán tử `&`, kết quả được một set mới có chứa các phần tử của chung cả 2 set, nếu chỉ tồn tại trong 1 set thì không lấy. Dùng toán tử `|` để thực hiện phép hợp, phép hợp lấy tất cả các phần tử của cả 2 set. Phép hiệu có toán tử là dấu trừ `-`, `set1 - set2` sẽ trả về các phần tử của `set1` và các phần tử vừa tồn tại trong `set1`, vừa tồn tại trong `set2`. Cuối cùng phép hiệu đối xứng là toán

từ ^, phép toán này lấy các phần tử trong set1 và set2, nhưng nếu tồn tại trong cả 2 set thì không lấy.

```
1  set1:  set(['a', 'c', 'b', 'd'])
2  set2:  set(['a', 'x', 'b', 'y', 'z'])
3  intersection:  set(['a', 'b'])
4  union:  set(['a', 'c', 'b', 'd', 'y', 'x', 'z'])
5  difference:  set(['c', 'd'])
6  symmetric difference:  set(['c', 'd', 'y', 'x', 'z'])
```

Tiếp theo là một số thao tác trên set.

```
1  set1 = set([1, 2])
2  set1.add(3)
3  set1.add(4)
4
5  set2 = set([1, 2, 3, 4, 6, 7, 8])
6  set2.remove(8)
7
8  print (set1)
9  print (set2)
10
11 print ("Is set1 subset of set2 ? : ", set1.issubset(set2))
12 print ("Is set1 superset of set2 ? : ", set1.issuperset(set2))
13
14 set1.clear()
15
```

```
print (set1)
```

Phương thức `add()` sẽ chèn một phần tử vào set. Phương thức `remove()` xóa một phần tử tại vị trí bất kỳ. Phương thức `clear()` xóa toàn bộ phần tử trong set. Phương thức `issubset()` kiểm tra xem một set có phải là set con của một set khác không. Phương thức `issuperset()` kiểm tra xem một set có phải là set cha của set khác không.

```
1  set([1, 2, 3, 4])
2  set([1, 2, 3, 4, 6, 7])
3  Is set1 subset of set2 ? : True
4  Is set1 superset of set2 ? : False set([])
```

2.1.10 Kiểu từ điển

Kiểu này lưu trữ các phần tử theo dạng các cặp khóa-giá trị (**key-value**). Các chỉ số trong từ điển là các khóa. Do đó các khóa phải khác nhau, chúng ta sẽ có một bài riêng để nói về kiểu này.

```
1  words = { 'girl': 'Maedchen', 'house': 'Haus', 'death': 'Tod' }
2
3  print (words['house'])
4
5  print (words.keys())
6  print (words.values())
```

```

7     print (words.items())
8
9     print (words.pop('girl'))
10    print (words)
11    words.clear()
12    print (words)

```

Ví dụ trên mô tả sơ lược về cách sử dụng kiểu từ điển.

```

1     Haus
2     ['house', 'girl', 'death']
3     ['Haus', 'Maedchen', 'Tod']
4     [('house', 'Haus'), ('girl', 'Maedchen'), ('death', 'Tod')]
5     Maedchen
6     {'house': 'Haus', 'death': 'Tod'}
7     {}

```

- Chuyển đổi kiểu

2.2 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 ($a^{**}b = a^b$)	$2 ** 3 = 8$ $5 ** 7 = 78125$

2.3 Toán tử logic

2.4 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 tử 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)
>=	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	7 >= 6 (True)
<=	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	5 <= 6 (true)

2.5 Gán

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

Loại toán tử	Mục đích	Cách dùng
=	Gán giá trị của vế phải cho vế trái	x = 5
+=	Tăng giá trị của vế phải sau đó gán cho vế trái	x += 5 (x = x + 5)

-=	Giảm giá trị của vế phải sau đó gán cho vế trái	$x -= 5$ ($x = x - 5$)
*=	Nhân giá trị của vế phải trước sau đó gán cho vế trái	$x *= 5$ ($x = x * 5$)
/=	Chia giá trị của vế phải sau đó gán cho vế trái (chia nguyên)	$x /= 5$ ($x = x / 5$)
%=	Chi giá trị của vế phải sau đó gán cho vế trái (chia lấy dư)	$x \% = 5$ ($x = x \% 5$)
//=	Phép chia lấy phần nguyên.	$x //= 5$ ($x = x // 5$)
**=	Tính số mũ của vế phải sau đó gán giá trị của vế trái	$x ** = 5$ ($x = x ** 5$)
&=	Thực hiện phép toán của toán tử AND của vế phải sau đó gán cho vế trái	$x \& = 5$ ($x = x \& 5$)
=	Thực hiện phép toán của toán tử OR của vế phải sau đó gán cho vế trái	$x = 5$ ($x = x 5$)

$\wedge=$	Thực hiện phép toán của toán tử XOR của vế phải sau đó gán cho vế trái	$x \wedge= 5$ ($x = x \wedge 5$)
$\gg=$	Thực hiện phép toán dịch phải của vế phải sau đó gán cho vế trái	$x \gg= 5$ ($x = x \gg 5$)
$\ll=$	Thực hiện phép toán dịch trái của vế phải sau đó gán cho vế trái	$x \ll= 5$ ($x = x \ll 5$)

2.6 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

2.7 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.

Trong Python, **is** và **is not** chính là 2 toán tử định danh. Ví dụ:

```
a = 5
```

```
b = 7
```

```
print(a is b)
```

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

Kết quả khi chạy chương trình:

False

True

2.8 Độ ưu tiên toán tử

2.9 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

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

2.9.1 Type Conversion

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

2.9.2 Iterables and Iterators

Function	Description
<code>all()</code>	Returns <code>True</code> if all elements of an iterable are true
<code>any()</code>	Returns <code>True</code> 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
<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

2.9.3 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



Function	Description
<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:

Khối 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.

```
1     for letter in 'Python':
```

```
2         print('Current Letter :', letter)
```

```
3
```

```
4     fruits = ['banana', 'apple', 'mango']
```

```
5     for fruit in fruits:
```

```
6         print ('Current fruit :', fruit)
```

```
1     Current Letter : P
```

```
2     Current Letter : y
```

```
3     Current Letter : t
```

```
4     Current Letter : h
```

```
5     Current Letter : o
```

```
6     Current Letter : n
```

```
7     Current fruit : banana
```

```
8     Current fruit : apple
```

```
9     Current fruit : mango
```

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

```

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

```

```

1  Current fruit : banana
2  Current fruit : apple
3  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>, (tức <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`

- For, while, nest loop

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)

```

Trong ví dụ trên, ở vòng lặp for, chúng ta cho lặp tiếp khi tìm thấy kí tự 'h' mà không in ký 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

Bài 4: Number, String, Date & Time

4.1 Kiểu số

Python cung cấp kiểu dữ liệu số nguyên, số thực và số phức.

Các hàm chuyển đổi kiểu:

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

Thư viện sinh số ngẫu nhiên: `random`

```
import random
print(random.randint(1, 55))
```

4.2 Kiểu chuỗi (string)

4.2.1 Xử lý 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"""
```

4.2.2 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

4.2.3 Dùng dấu nhảy bên trong string

Để tạo string thì chúng ta dùng các dấu nhảy, nhưng nếu chúng ta muốn dùng chính các dấu nhảy đó bên trong string thì sao? Có hai cách, một là dùng ký tự thoát, hai là trộn lẫn giữa dấu nhảy cần dùng với dấu nhảy khởi tạo string.

```
1 print ("There are many stars.")
2 print ("He said, \"Which one is your favourite?\"")
3
4 print ('There are many stars.')
5 print ('He said, "Which one is your favourite?"')
```

Cách thứ nhất là dùng ký tự thoát \ các ký tự điều khiển sẽ bị vô hiệu hóa và sẽ được in ra như các ký tự bình thường. Cách thứ hai là chúng ta trộn, nếu muốn sử dụng dấu nhảy kép trong string thì chúng ta dùng dấu nhảy đơn để bao bọc lấy string và ngược lại.

```
1 There are many stars.
2 He said, "Which one is your favourite?"
```



```
3     There are many stars.
4     He said, "Which one is your favourite?"
```

4.2.4 Độ dài của string

Phương thức `len()` trả về số lượng chữ cái trong string, kể cả dấu cách.

```
1     s1 = "Eagle"
2     s2 = "Eagle\n"
3     s3 = "Eagle  "
4
5     print (len(s1))
6     print (len(s2))
7     print (len(s3))
```

Ví dụ trên in ra độ dài của 3 string.

```
1     s1 = "Eagle"
2     s2 = "Eagle\n"
3     s3 = "Eagle  "
```

Ở đây, string thứ 2 có kí tự xuống dòng, string thứ 3 có 2 dấu cách và chúng đều được tính.

```
1     5
2     6
3     7
```

4.2.5 Loại bỏ khoảng trắng thừa trong string

Đôi khi các khoảng trắng thừa ở đầu string hay cuối string nhìn khá là khó chịu và bạn muốn loại bỏ chúng đi. Python cung cấp các phương thức `strip()`, `rstrip()` và `lstrip()` để làm công

việc này. Bản thân các phương thức này không loại bỏ các kí tự trong string gốc mà chúng sẽ copy string gốc sang một nơi khác, sau đó loại bỏ khoảng trắng ở các string này rồi trả string mới này cho chúng ta.

```
1     s = " Eagle  "
2
3     s2 = s.rstrip()
4     s3 = s.lstrip()
5     s4 = s.strip()
6
7     print (s, len(s))
8     print (s2, len(s2))
9     print (s3, len(s3))
10    print (s4, len(s4))
```

Ở ví dụ trên, chúng ta áp dụng cả 3 phương thức trên vào string `s`. String `s` có 1 khoảng trắng thừa ở đầu câu và 2 khoảng trắng thừa ở cuối câu.

```
1     s2 = s.rstrip()
```

Phương thức `rstrip()` loại bỏ khoảng trắng thừa ở cuối string.

```
1     s3 = s.lstrip()
```

Phương thức `lstrip()` loại bỏ khoảng trắng thừa ở đầu string.

```
1     s4 = s.strip()
```

Phương thức `strip()` loại bỏ khoảng trắng thừa ở cả đầu và cuối string.

```

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

```

Chúng ta in kết quả ra màn hình. Lưu ý là python tự động cách dòng các thành phần trong hàm `print`.

4.2.6 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 bound
2       to hear my cry")

       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 Pytooo

```

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 ký tự bình thường.

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

```
1 Another world\n
```

Ngoài ra Python còn nhiều kiểu ký tự thoát khác nữa, bạn có thể tham khảo tại [đây](#).

4.2.7 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 ký tự đầu tiên bằng nhau nhưng ký tự thứ hai khác nhau, trong đó ký tự `a` lớn hơn ký 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
```

4.2.8 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 ký 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 = "Pho Code"
2
3 for i in s:
4     print (i,)
```

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

```
1 P h o C o d e
```

4.2.9 Tìm string 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"))
```

Trong 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
```

4.2.10 Toán tử trên string

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

```
1 print ("eagle " * 5)
2
3 print ("eagle " "falcon")
4
5 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,

4.2.11 Thay thế string

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.
```

4.2.12 Tách, nối string

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 tách, phần tử thứ 2 là chính bản thân kí tự phân tách, phần tử thứ 3 là đoạn string nằm phía sau kí tự phân tách.

```

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')
```

4.2.13 Chữ HOA và chữ thường

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

```
1 a = "PhoCode"
2
3 print (a.upper())
4 print (a.lower())
5 print (a.swapcase())
6 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 PHOCODE
2 phocode
3 pHOcODE
4 Phocode
```

4.2.14 Một số thao tác khác trên string

```
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 i.isspace():
13        spaces += 1
14
15
16 print ("There are", len(sentence), "characters")
17 print ("There are", alphas, "alphabetic characters")
```



```
18     print ("There are", digits, "digits")
        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

```

4.2.15 Định dạng string

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 theo 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 (thật ra mình cũng chẳng biết kiểu định dạng này 😊)

```

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
```

4.3 Kiểu Date, Time

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(2019,1,1)          # 01/01/2019
dt = datetime(2019,1,1,23,59,59) # 23:59:59 01/01/2019
```

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)
```

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, 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)
```

- Mathematical function, String method, Unicode String

- Type() và id()

Bài 5: Kiểu cấu trúc

5.1 Kiểu List

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.

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.

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

Trong ví dụ trên, chúng ta có một list có 5 phần tử. List được khởi tạo trong cặp dấu []. Các phần tử của list được phân cách bởi dấu phẩy.

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

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", (2, 3),
5            Being(), {}]
6
7  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 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, 0, 0]
2     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
3     [10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 0, 0, 0, 0]
```

5.1.2 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("PhoCode"))
8     print (list(['Ruby', 'Python', 'Perl']))
```

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("PhoCode"))
```

Dòng code trên tạo một list từ một string.

```
1 print (list(['Ruby', 'Python', 'Perl']))
```

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 ['Ruby', 'Python', 'Perl']
```

5.1.3 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
```

5.1.4 Một số hàm dùng với list

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

```
2
```

```
3 print ("There are %d items" % len(n))
```

```
4 print ("Maximum is %d" % max(n))
```

```
5 print ("Minimum is %d" % min(n))
```

```
6 print ("The sum of values is %d" % sum(n))
```

Đoạn code trên ví dụ về tác dụng của bốn hàm `len()`, `max()`, `min()`, và `sum()`.

```
1 print ("There are %d items" % len(n))
```

Hàm `len()` lấy độ dài của một list.

```
1 print ("Maximum is %d" % max(n))
```

```
2 print ("Minimum is %d" % min(n))
```

Hàm `max()` và `min()` tìm giá trị lớn nhất và nhỏ nhất có trong list, tất nhiên là trong trường hợp các phần tử trong list phải có cùng kiểu dữ liệu, nếu không sẽ báo lỗi.

```
1 print ("The sum of values is %d" % sum(n))
```

Hàm `sum()` trả về tổng các giá trị của các phần tử trong list, chỉ có thể dùng khi list chứa các phần tử kiểu số.

```
1 There are 8 items
```

```
2 Maximum is 8
```

```
3 Minimum is 1
```

```
4 The sum of values is 36
```

5.1.5 Thêm phần tử mới vào list

```
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 kì 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 thể 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 []
```

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()`, ví dụ `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 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
```

```
3     for e in w:
```

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

```
3     ['Python', 'Ruby', 'Perl'] ['Python', 'Ruby', 'Perl'] ['Python',  
    'Ruby', 'Perl']
```

```
    ['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]
```

5.1.15 Đảo ngược list

```
1     a1 = ["bear", "lion", "tiger", "eagle"]
2
3     a1.reverse()
4     print (a1)
```

Để đảo ngược một list thì chúng ta dùng phương thức `reverse()`.

```
1     ['eagle', 'tiger', 'lion', 'bear']
```

5.1.16 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 tường minh lại 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]
```

5.1.17 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_hoc sinh = ("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

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.

5.3.1 Khởi tạo dictionary

Có nhiều cách tạo dictionary. Đầu tiên chúng ta sẽ tìm hiểu cách khởi tạo một dictionary dùng 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': 'Berlin'}
4     {0: <object object at 0xb76cb4a8>, 1: <object object at
5     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)
```

5.3.2 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" % len(basket))
7
8     print (basket['apples'])
9     basket['apples'] = 8
10    print (basket['apples'])
11
12    print (basket.get('oranges', 'undefined'))
13    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
```

```

7     fruits['oranges'] = 12
8     fruits['pears'] = 8
9     fruits['apples'] = 4
10
11    print (fruits.setdefault('oranges', 11))
12    print (fruits.setdefault('kiwis', 11))
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, 'bottles': 4,
2      'cups': 2}
3      {'bags': 1, 'pens': 3, 'books': 5, 'bottles': 4, 'cups': 2}
4      {'bags': 1, 'pens': 3, 'books': 5, 'cups': 2}
      {}

```

5.3.3 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": "Hungary",
2                  "us": "United States", "no": "Norway"  }
3
4      print (domains.keys())
5      print (domains.values())
6      print (domains.items())
7
8      print ("de" in domains)
9      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', 'Hungary']
```

```
3 [('sk', 'Slovakia'), ('de', 'Germany'), ('no', 'Norway'),
```

```
4 ('us', 'United States'), ('hu', 'Hungary')]
```

```
5 True
```

```
6 False
```

5.3.4 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": "Hungary",
```

```
2            "us": "United States", "no": "Norway" }
```

```
3
```

```
4 for key in domains:
```

```
5
```

```
6         print (key)
7
8     for k in domains:
9         print (domains[k])
10
11    for k, v in domains.items():
12
13        print (": ".join((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
3         print (key)
```

Vòng lặp trên sẽ in ra danh sách các khóa trong dict.

```
1     for k in domains:
2
3         print (domains[k])
```

Vòng lặp trên in ra danh sách các giá trị trong list.

```
1     for k, v in domains.items():
2
3         print (": ".join((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()
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	phone_numbers.popitem()

Method	What is does	Example
	(Python 3.7+) or a random item	
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

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

5.4.1 Các phép tính trên Set

Hàm **add** : thêm giá trị mới vào Set

Ví dụ:

```
>>> s = set([1, 2]); s.add(3); print(s)
{1, 2, 3}
```

Hàm **remove** : xóa phần tử ra khỏi Set

Ví dụ:

```
>>> s = set([1, 2, 3]); s.remove(2); print(s)
{1, 3}
```

Phép **in** : kiểm tra giá trị có nằm trong Set

Ví dụ:

```
>>> s = set([1, 2, 3]); print(1 in s)
True
```

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
```

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}
```

Hàm **intersection** : giao của 2 Set

Ví dụ:

```
>>> s1 = set([1, 2, 3]); s2 = set([3, 4, 5]); print(s1.intersection(s2))
{3}
```

Hàm **difference** : hiệu của 2 Set

Ví dụ:

```
>>> s1 = set([1, 2, 3]); s2 = set([3, 4, 5]); print(s1.difference(s2))
{1, 2}
```

5.4.2 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)

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()}  
{last.title()}'  
print(full_name('Hien', 'Luong'))  
Hàm này có 2 tham số first và last.
```

—

—

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://automatetheboringstuff.com/>
9. Các bài giảng, bài viết từ internet.

