

LẬP TRÌNH PYTHON

Phần cơ bản

(Fundamentals of Python)



LÊ VĂN HẠNH

levanhanhvn@gmail.com

Tháng 9-2019

MỤC LỤC

1	TỔNG QUAN	1
1.1.	Cài đặt Python trên Windows.....	1
1.2.	Công cụ phát triển	2
1.2.1.	PyCharm.....	2
1.2.2.	PyDev	3
1.2.3.	Atom IDE	3
1.2.4.	Wing Python.....	3
1.2.5.	PyScripter	3
1.3.	Cài đặt Pycharm	3
1.4.	Cài đặt VisualStudio code	7
1.5.	Tạo Project, python file, viết mã và thực thi chương trình Python trên PyCharm	9
2	NGÔN NGỮ LẬP TRÌNH PYTHON	12
2.1.	Định danh (Identifier).....	12
2.2.	Khởi lệnh	12
2.3.	Biến	13
2.4.	Hằng số (constant).....	13
2.5.	Các kiểu dữ liệu.....	14
2.5.1.	Phân loại.....	14
2.5.2.	Kiểu dữ liệu cơ bản	14
2.6.	Chuyển đổi kiểu dữ liệu	15
2.7.	Các toán tử (operators)	16
2.7.1.	Toán tử số học(Arithmetic operators)	16
2.7.2.	Toán tử gán (Assignment Operators)	16
2.7.3.	Toán tử so sánh (Comparison Operators).....	17
2.7.4.	Toán tử logic (Logical Operators).....	18
2.7.5.	Toán tử thành phần (Membership Operators)	18
2.7.6.	Toán tử bit (Operators).....	18
2.7.7.	Toán tử định danh (Identity Operators).....	19
2.7.8.	Độ ưu tiên của các toán tử (Operators Precedence)	20
2.8.	Console Input-Output.....	20
2.8.1.	Phương thức print()	20
2.8.2.	Xóa màn hình (console)	25
2.8.3.	Hàm input()	25
2.9.	Chú thích (comment) trong Python.....	26
2.10.	Cấu trúc điều kiện.....	26
2.10.1.	if	27
2.10.2.	switch...case.....	28
2.11.	Cấu trúc lặp	28
2.11.1.	while	28
2.11.2.	for	29
2.11.3.	Sử dụng else trong cấu trúc lặp	30
2.11.4.	Lệnh break, continue, pass	30
2.12.	Một số hàm tích hợp sẵn trong Python (builtin_function)	32
2.13.	Thao tác với đối tượng kiểu String.....	36
2.13.1.	Truy xuất chuỗi con bằng cách sử dụng index và toán tử cắt lát (slicing - [:]).....	36
2.13.2.	Duyệt chuỗi	37

2.13.3.	So sánh chuỗi.....	38
2.13.4.	Các phương thức dùng với string	38
2.14.	Debug.....	43
2.14.1.	Giới thiệu	43
2.14.2.	Các phương pháp debug	43
2.14.3.	Debug Tool trong Pycharm.....	44
3	USER DEFINE FUNCTION - MODULE - PACKAGE.....	46
3.1.	Hàm do người dùng tự tạo (UDF – User-Define Function)	46
3.1.1.	Định nghĩa	46
3.1.2.	Khai báo và xây dựng hàm	46
3.1.3.	Vị trí của hàm trong chương trình và lời gọi hàm	47
3.1.4.	Tầm vực của biến (Scope of variables)	47
3.1.5.	Tham số của hàm (parameters / arguments)	50
3.1.6.	Lệnh yield	53
3.1.7.	Hàm ẩn danh (Anonymous function - lambda)	54
3.2.	Module.....	57
3.2.1.	Giới thiệu	57
3.2.2.	Phân loại module/thư viện	57
3.2.3.	Cách khai báo và sử dụng file chứa các User Define Function	58
3.2.4.	Import module	58
3.2.5.	Xem thông tin về module	61
3.3.	Namespace.....	61
3.4.	Package.....	62
3.4.1.	Giới thiệu	62
3.4.2.	Package module	62
3.5.	Một số module sẵn có trong Python	63
3.5.1.	Module collections.....	63
3.5.2.	Module random.....	67
3.5.3.	Module math.....	69
3.5.4.	Các module liên quan tới thời gian.....	70
3.5.5.	Module sys.....	79
3.5.6.	Module struct.....	81
3.5.7.	Module platform	81
3.5.8.	Module textwrap.....	81
3.5.9.	Module itertools.....	83
4	CÁC ĐỐI TƯỢNG DẠNG DANH SÁCH TRONG PYTHON.....	88
4.1.	Iterator trong Python.....	88
4.1.1.	Giới thiệu	88
4.1.2.	Duyệt iterable.....	89
4.1.3.	So sánh các đối tượng dạng iterator trong Python.....	89
4.2.	List.....	89
4.2.1.	Giới thiệu	89
4.2.2.	Khai báo.....	90
4.2.3.	Đếm số lượng phần tử có trong List	90
4.2.4.	Xuất nội dung list ra màn hình	91
4.2.5.	Thêm phần tử vào list	91
4.2.6.	Cập nhật giá trị cho phần tử trong list	92

4.2.7.	Kiểm tra sự tồn tại của một phần tử trong list.....	92
4.2.8.	Copy list	93
4.2.9.	Xóa phần tử trong list.....	94
4.2.10.	Sắp xếp trong list.....	97
4.2.11.	Đảo ngược thứ tự trong list	97
4.2.12.	Trích xuất sublist.....	97
4.2.13.	Thao tác trên nhiều list	98
4.2.14.	List lồng nhau.....	99
4.2.15.	Chuyển đổi một iterable object (tuple, string, set, dictionary) thành list.....	101
4.3.	Tuple.....	102
4.3.1.	Giới thiệu.....	102
4.3.2.	Tạo tuple.....	102
4.3.3.	Truy cập các phần tử của tuple.....	103
4.3.4.	Cập nhật giá trị của phần tử trong tuple	104
4.3.5.	Kiểm tra một đối tượng có tồn tại trong tuple hay không?	104
4.3.6.	Xóa tuple	105
4.3.7.	Một số phương thức và hàm hỗ trợ việc xử lý trên tuple.....	105
4.4.	Dictionary.....	105
4.4.1.	Giới thiệu.....	105
4.4.2.	Tạo dictionary	106
4.4.3.	Thêm phần tử vào dictionary (hoặc cập nhật value thông qua key).....	106
4.4.4.	Truy xuất phần tử của dictionary	107
4.4.5.	Kiểm tra một key đã có trong dictionary hay chưa?	108
4.4.6.	Chuyển đổi 2 thành phần key và value cho nhau	108
4.4.7.	Chuyển đổi list sang dictionary	108
4.4.8.	Viết chương trình tạo dictionary từ string.....	109
4.4.9.	Xóa trên dictionary.....	110
4.4.10.	Gộp 2 dictionaries	111
4.4.11.	Sort trên dictionary	113
4.4.12.	Một số hàm hỗ trợ việc xử lý trên dictionary	114
4.4.13.	Một số phương thức của đối tượng dictionary	114
4.5.	Set.....	115
4.5.1.	Khai báo và gán giá trị cho set	115
4.5.2.	Xuất (in) nội dung set ra màn hình.....	115
4.5.3.	Duyệt qua các phần tử của set.....	115
4.5.4.	Sao chép set.....	116
4.5.5.	Thêm phần tử vào set	116
4.5.6.	Kiểm tra phần tử có tồn tại trong set hay không?	116
4.5.7.	Xóa phần tử khỏi set.....	116
4.5.8.	Các toán tử và phương thức trên set.....	117
4.5.9.	Một số hàm thường dùng trên set.....	118
4.5.10.	Frozenset	118
4.6.	Một số phương thức và hàm hỗ trợ việc xử lý trên đối tượng dạng danh sách	119
4.7.	Xây dựng hàm ẩn danh (Anonymous function - lambda) cho sequence data type	126
4.7.1.	Giới thiệu.....	126
4.7.2.	Một số ví dụ về dùng 2 hàm map và filter trên iterator object	127
4.7.3.	lambda function có thể gọi 1 lambda function khác	129
4.8.	Sử dụng kỹ thuật Comprehension cho sequence data type	130

5	XỬ LÝ NGOẠI LỆ (Exception Handling)	133
5.1.	Lỗi cú pháp (Syntax Error)	133
5.2.	Lỗi ngoại lệ (Exception Error)	133
5.3.	Assertions	133
5.4.	Standard Exceptions	134
5.4.1.	Các exception có sẵn trong Python	134
5.4.2.	Sử dụng try ... except trong việc xử lý ngoại lệ	135
5.5.	Exception do người dùng định nghĩa (User Defined Exception)	137
6	THAO TÁC VỚI TẬP TIN & THƯ MỤC	138
6.1.	Thao tác với các loại tập tin	138
6.1.1.	Mở file	138
6.1.2.	Đóng file	139
6.2.	Thao tác với tập tin văn bản (Text File)	140
6.2.1.	Đọc và ghi file	140
6.2.2.	Đổi tên file	143
6.2.3.	Xóa file	143
6.3.	Thao tác với tập tin CSV (CSV File)	144
6.3.1.	File CSV	144
6.3.2.	Module CSV	144
6.3.3.	Đọc file CSV	145
6.3.4.	Ghi file CSV	151
6.5.	Thao tác với thư mục (Directory) qua module OS	153
6.6.	Module os.path	154
	THAM KHẢO	155

TỔNG QUAN

Python là một ngôn ngữ lập trình bậc cao, một ngôn ngữ lập trình hướng đối tượng, do *Guido van Rossum* cho ra mắt vào năm 1991. Mục đích ra đời của *Python* là cung cấp một ngôn ngữ lập trình có cấu trúc rõ ràng, sáng sủa, thuận tiện cho người mới học lập trình. Một trong những đặc điểm độc nhất của *Python* là ngôn ngữ này không dùng đến dấu chấm phẩy, dấu mở-đóng ngoặc `{}` để kết thúc câu lệnh hay khối lệnh, mà cách duy nhất để nó nhận biết một lệnh là dấu thụt đầu dòng.

Python là một ngôn ngữ biên dịch (hay thông dịch - *Interpreter Language*), tức là không cần *build* thành file thực thi mà chạy trực tiếp như *PHP* và đa nền tảng (Mac OS, OS/2, Windows, Linux và các hệ điều hành khác thuộc họ Unix).

Hiện tại *Python* có 2 nhánh chính là 2.x và 3.x. Ở nhánh 2.x đã dừng phát triển và đang đứng ở phiên bản 2.7. Nhánh *Python* 3.x thì vẫn đang được tiếp tục phát triển.

Website chính thức của *Python*: www.python.org

1.1. Cài đặt Python trên Windows

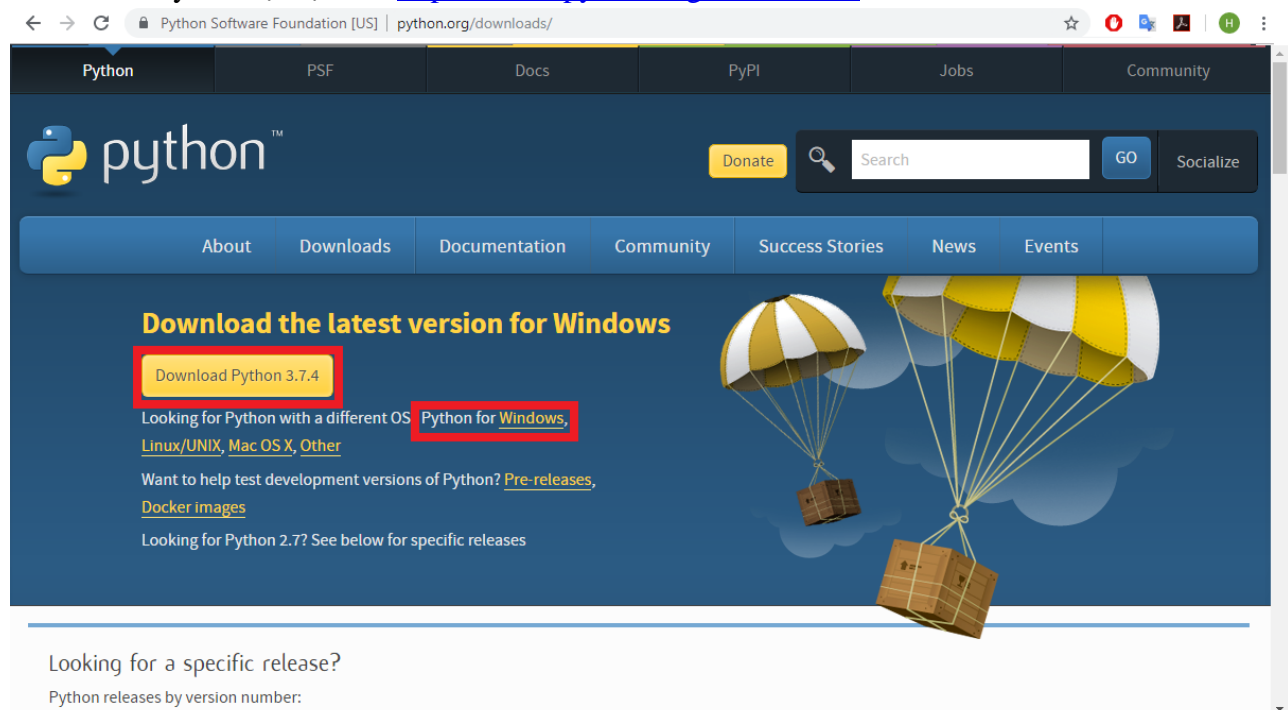
Python hỗ trợ hầu hết các nền tảng và rất dễ tìm thấy sẵn trên một số hệ điều hành như Mac OS...

Để biết là hệ thống của bạn đã cài *Python* chưa, có thể vào màn hình *command line* và gõ:

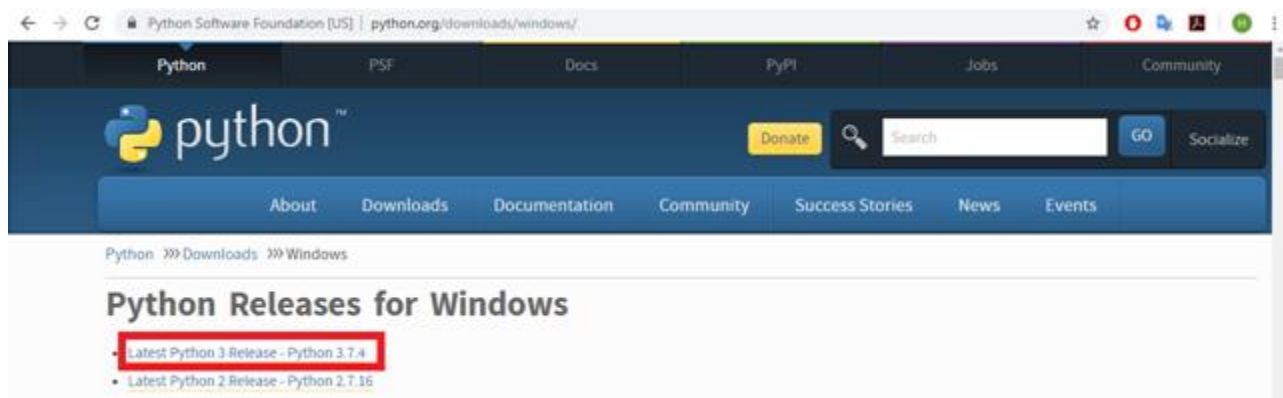
python --version

Nếu đã cài đặt *Python* thì sẽ hiển thị thông tin phiên bản *Python*. Ngược lại, nếu có báo lỗi đồng nghĩa với bạn chưa cài đặt *Python*.

Tải *Python* tại địa chỉ: <https://www.python.org/downloads/>



Click hyperlink Windows để mở trang sau, trong màn hình sau, chọn phiên bản *Python* 3.7.4.



Sau khi click vào hyperlink *Latest Python 3 Release-Python 3.7.4*, sẽ mở đến trang thứ 3. Cuộn đến cuối trang này để chọn bản cài đặt trên Windows phù hợp với hệ điều hành của máy

Files					
Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		68111671e5b2db4aef7b9ab01bf0f9be	23017663	SIG
XZ compressed source tarball	Source release		d33e4aae66097051c2eca45ee3604803	17131432	SIG
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	6428b4fa7583daff1a442c8a8cee08e6	34898416	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	5dd605c38217a45773bf5e4a936b241f	28082845	SIG
Windows help file	Windows		d63999573a2c06b2ac56cade6b4f7cd2	8131761	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	9b00c8cf6d9ec0b9abe83184a40729a2	7504391	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	a702b4b0ad76debd3043a583e563400	26680368	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	28cb1c608bbd73ae8e53a3bd351b4bd2	1362904	SIG
Windows x86 embeddable zip file	Windows		9fab3b81f8841879fda94133574139d8	6741626	SIG
Windows x86 executable installer	Windows		33cc602942a54446a3d6451476394789	25663848	SIG
Windows x86 web-based installer	Windows		1b670cfa5d317df82c30983ea371d87c	1324608	SIG

Khi click vào mục cần cài đặt, sẽ yêu cầu lưu file cài đặt vào máy.

Sau khi download hoàn tất, run file cài đặt để bắt đầu cài đặt

1.2. Công cụ phát triển

IDE (*Integrated Development Environment*) là môi trường tích hợp dùng để phát triển phần mềm giúp bạn viết code tốt hơn. Không chỉ vậy, các IDE còn kèm theo các công cụ hỗ trợ khác như trình biên dịch (*Compiler*), trình thông dịch (*Interpreter*), công cụ kiểm tra lỗi (*Debugger*), định dạng hoặc highlight mã nguồn, tổ chức thư mục chứa mã nguồn, tìm kiếm mã nguồn, ...

Vì vậy, mặc dù chỉ cần dùng một *text editor* là có thể viết được code *Python* nhưng người lập trình thường tìm cho mình một IDE thích hợp để phát triển là rất cần thiết.

Hiện tại các IDE hỗ trợ *Python* rất nhiều, trong tài liệu này chỉ xin giới thiệu một số IDE thường dùng:

1.2.1. PyCharm

PyCharm phát triển bởi JetBrains. Tuy phải trả phí sử dụng theo năm nhưng PyCharm được cho là IDE tốt nhất cho lập trình *Python* tất cả các cấp độ nhờ cung cấp nhiều tính năng như:

- Bộ code completion, dễ dàng điều hướng và kiểm tra lỗi.
- Có thể tự động thụt lề, phát hiện văn bản trùng lặp và kiểm tra lỗi.
- Tính năng tìm kiếm mã nguồn.



Trong phạm vi tài liệu này hướng đến cách thực thi Python bằng PyCharm IDE.

1.2.2. PyDev

PyDev là một Plugin cho phép bạn cài đặt vào Eclipse và lập trình Python ngay trên Eclipse IDE.



- Ưu điểm:
 - Miễn phí.
 - Có tính năng tự động hoàn thành code, thụt lề khối, highlight các dấu ngoặc, debugger, phân tích code, ...
- Nhược điểm: ít những tính năng và tiện lợi hơn PyCharm.

1.2.3. Atom IDE



- Miễn phí.
- Trình soạn thảo văn bản Python hiện đại, dễ sử dụng và có các tính năng vượt trội
- Cho phép lập trình viên tạo package mới cho mọi nhu cầu, từ làm theme, đến đồng bộ hóa...
- Cộng đồng sử dụng Atom đông, dễ dàng tìm thấy sự giúp đỡ và hợp tác.

1.2.4. Wing Python



Wing được đánh giá là một trình soạn thảo code cao cấp và không thiếu một tính năng nào (như auto-completion, highlight cú pháp, thụt lề, debugger...).

Wing được chia thành 2 phiên bản:

- Phiên bản personal của Wing được cung cấp miễn phí và đầy đủ các tính năng.
- Phiên bản professional (có trả phí), trong đó bổ sung thêm các tính năng nâng cao.

1.2.5. PyScripter



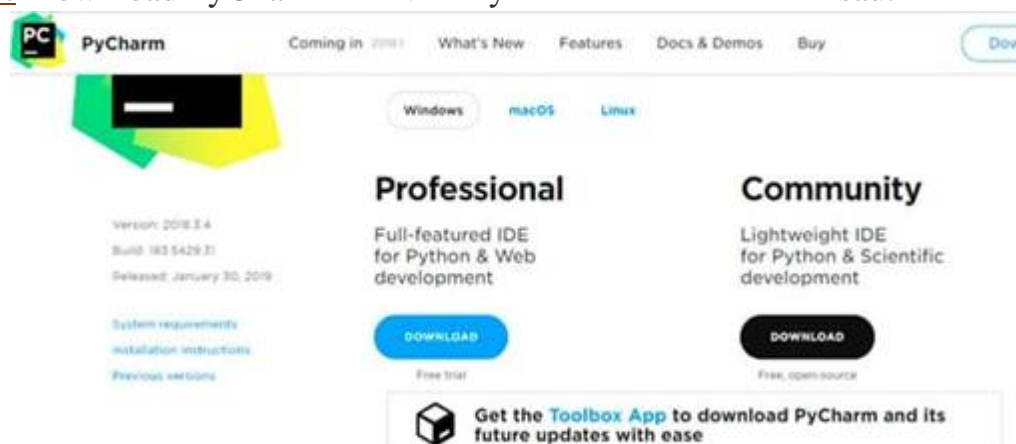
- Được cung cấp hoàn toàn miễn phí và là công cụ mã nguồn mở.
- Dung lượng dành cho PyScripter cực kỳ nhẹ.

1.3. Cài đặt Pycharm

Để cài đặt được Pycharm, trước đó, cần phải hoàn tất việc cài Python.

Bước 1. Vào website www.jetbrains.com

Bước 2. Download PyCharm IDE về máy tính cá nhân như hình sau:



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 phải mua bản quyền. Có thể download bản dùng thử.

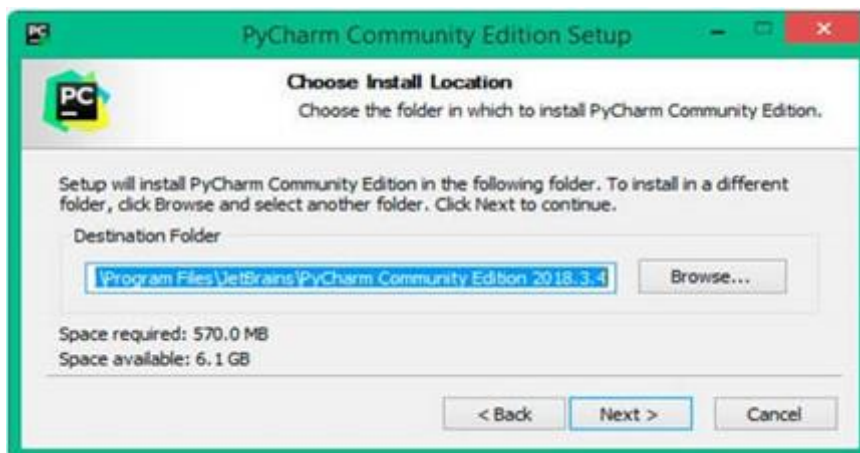
- Bản *Community*: Là bản chứa các tính năng cơ bản, để có thể phát triển Python. Bản này được tải miễn phí.

Bước 3. Cài đặt PyCharm IDE

- Sau khi download thành công, double click lên file vừa download để tiến hành cài đặt PyCharm. Xuất hiện màn hình chào mừng được hiển thị, chọn button Next để tiếp tục.



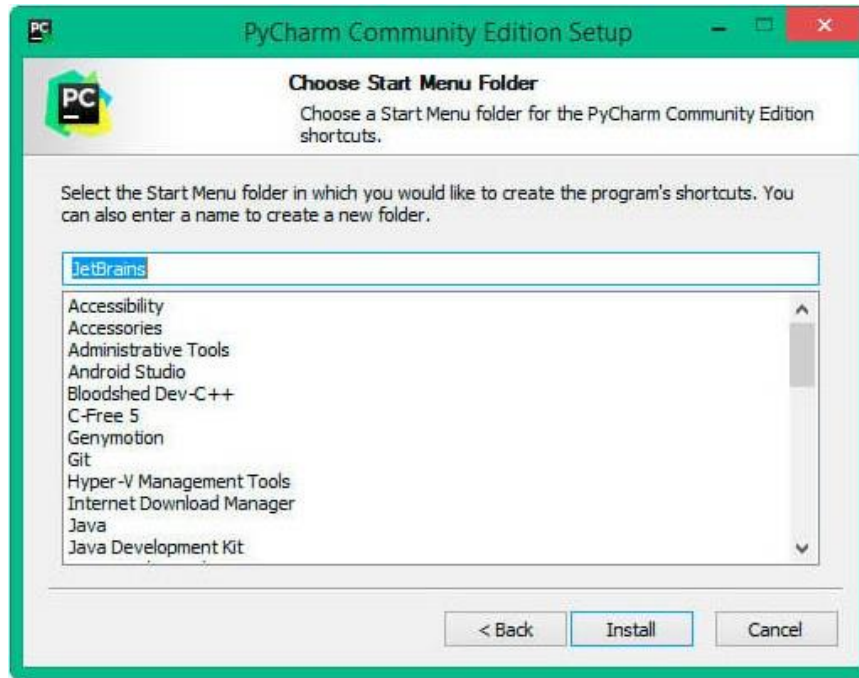
- Sau đó, ta chọn đường dẫn chứa bộ cài đặt. Tiếp tục chọn button Next



- Trong màn hình kế tiếp, chọn các tùy chọn cho việc cài đặt. Khi xong, chọn button Next



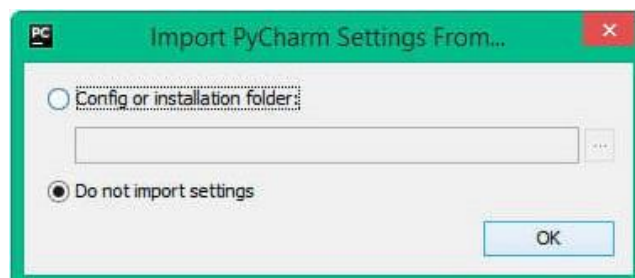
- Sau đó, ta chọn Install trong màn hình tiếp theo, để bắt đầu tiến hành cài đặt PyCharm.



- Sau khi cài đặt xong, PyCharm sẽ hỏi ta có muốn khởi động lại máy luôn hay không? Chọn RebootNow để khởi động lại máy tính nhằm hoàn tất quá trình cài đặt.



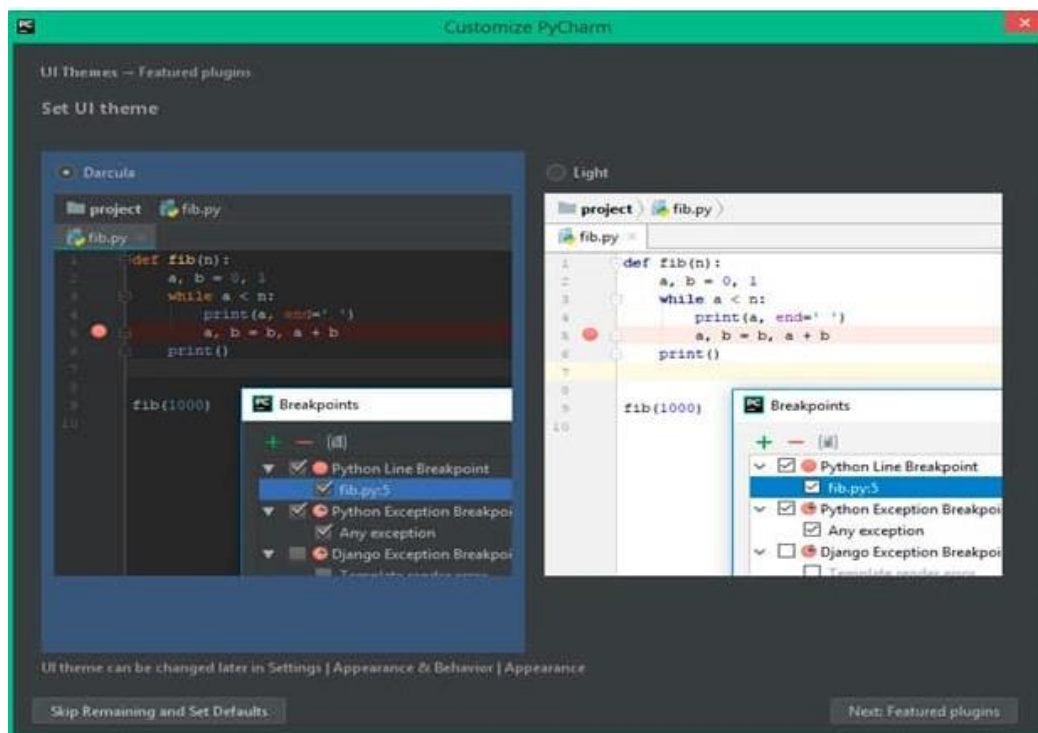
- Sau khi khởi động máy xong, mở PyCharm, Pycharm sẽ hỏi có muốn Import các thiết lập đã có từ trước hay không. Nếu cài mới hoàn toàn, ta chọn mục Do not import settings, rồi click button OK.



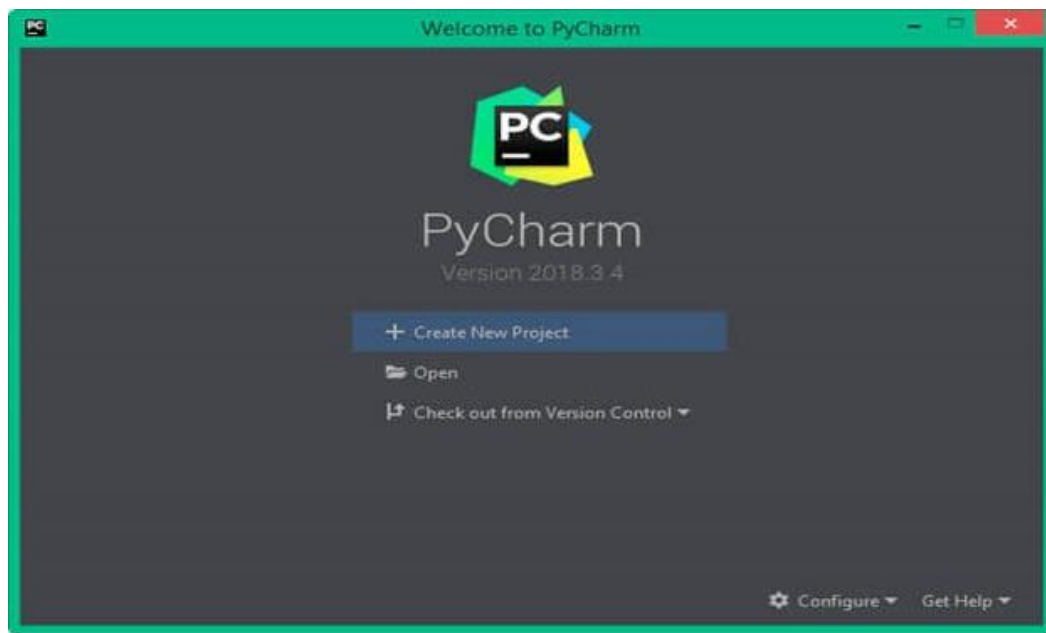
- Trong phần chính sách bảo mật, chọn “I confirm that ...”, sau đó click button Continue để tiếp tục.



- Trong màn hình Customize PyCharm, chọn 1 trong 2 đề xuất về màn hình làm việc sau này (nền đen hay trắng), xong click chọn button *Skip Remaining and Set Defaults* để xác nhận các lựa chọn làm mặc định.

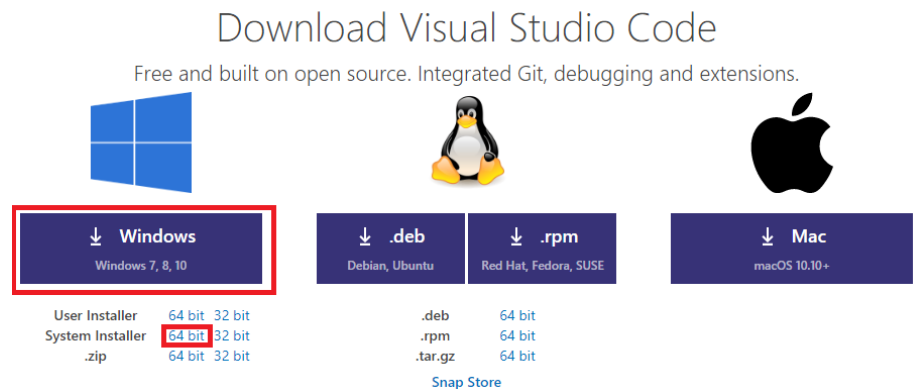


- Cuối cùng là màn hình *Welcome to PyCharm*, tại đây có thể chọn mục *Create New Project* để tạo một *Project* mới hoặc kết thúc việc cài đặt bằng cách click close button của cửa sổ.



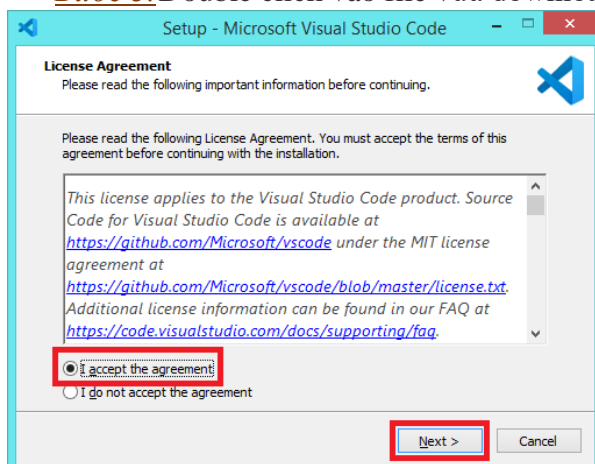
1.4. Cài đặt VisualStudio code

Bước 1. Vào trang <https://code.visualstudio.com/download>, chọn System Installer 64 bit của hệ điều hành Windows.

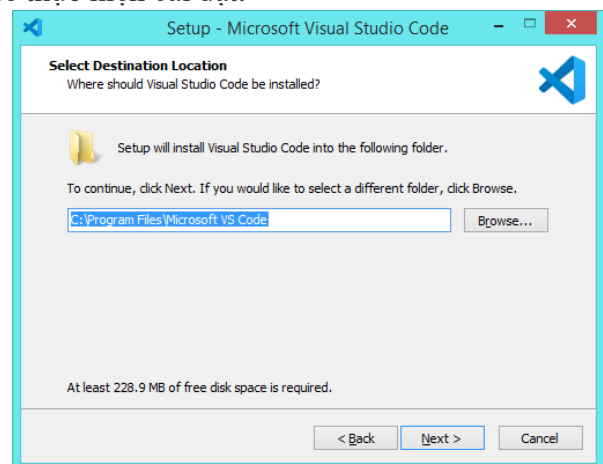


Bước 2. Lưu file download về máy tính

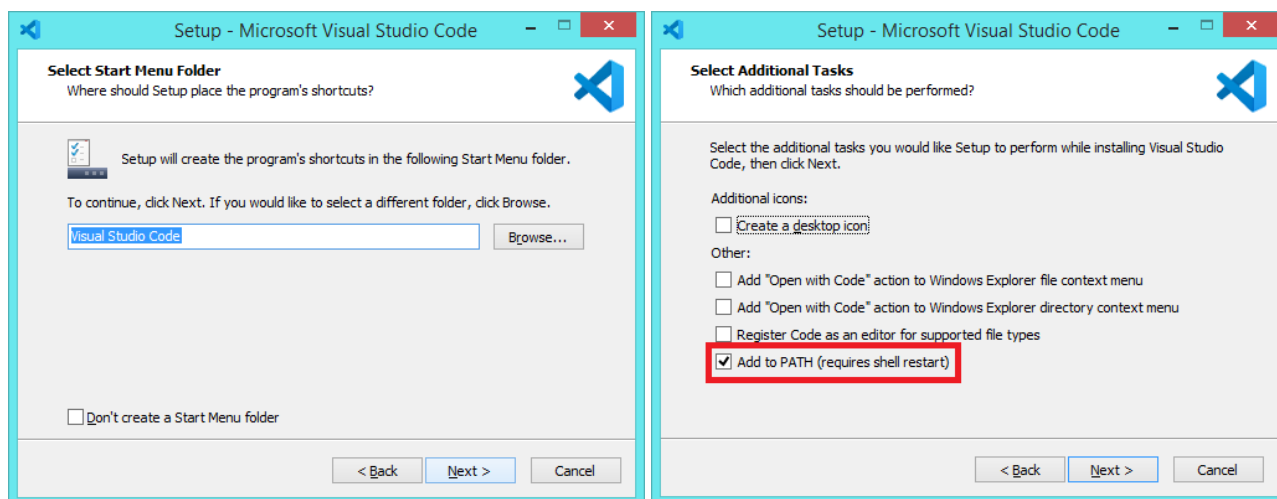
Bước 3. Double click vào file vừa download để thực hiện cài đặt.



Chọn I accept ... Xong chọn Next



Chọn đường dẫn lưu file. Xong chọn Next



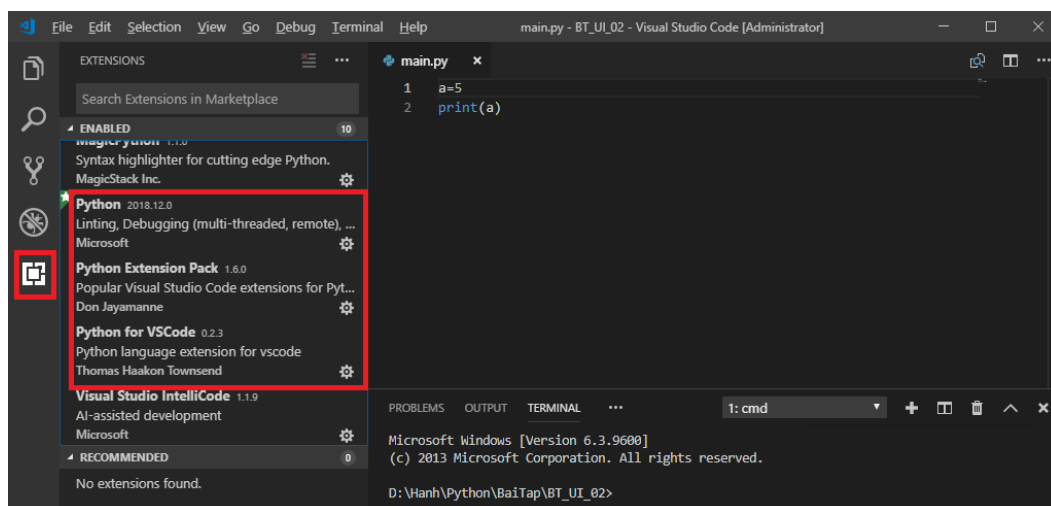
Xác nhận tên của shortcut sẽ được tạo

Chọn Add to PATH

Cuối cùng chọn Install để bắt đầu cài đặt.

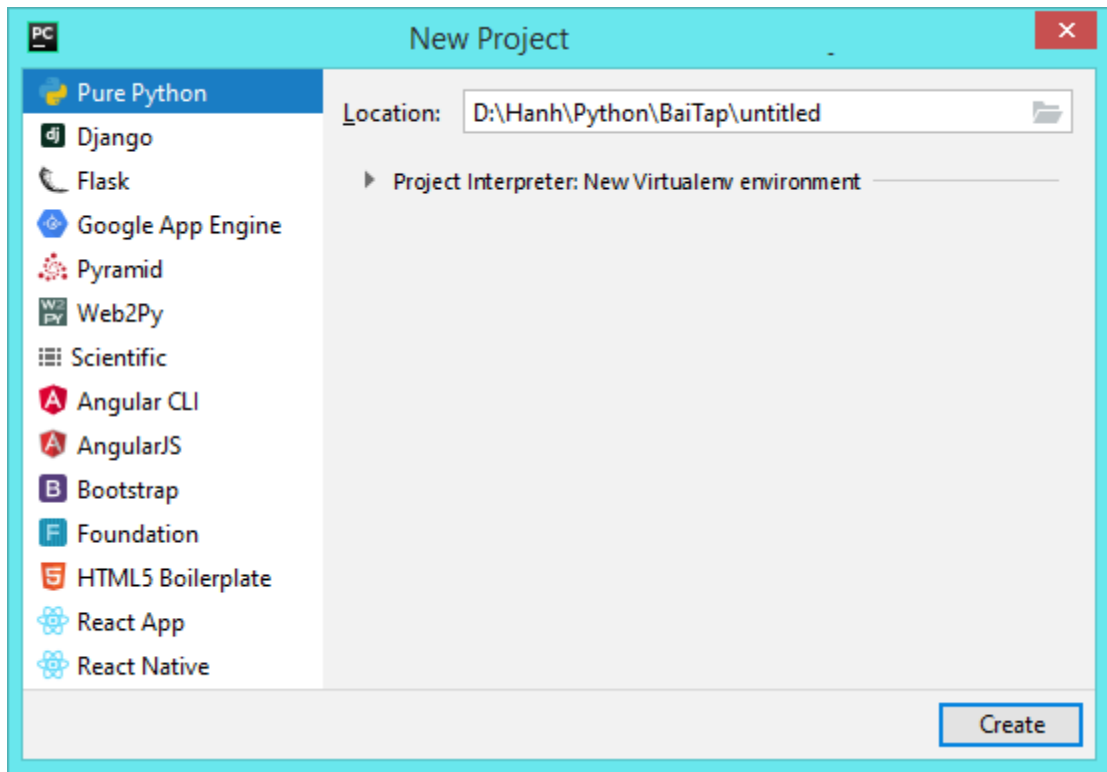
Bước 4. Khởi động Visual Code

- Click icon Extensions
- Chọn install các bổ sung cho việc viết code Python

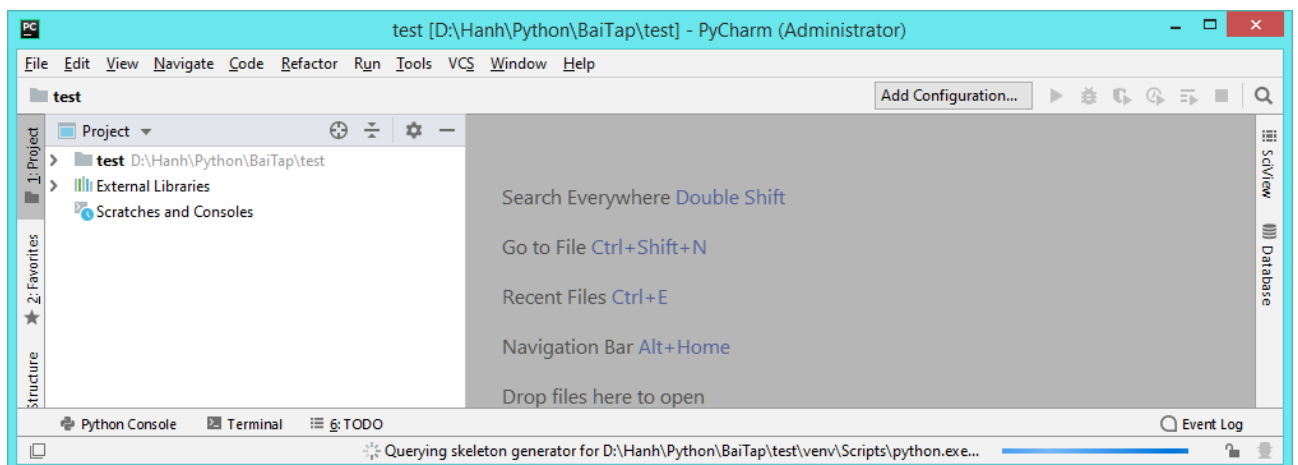


1.5. Tạo Project, python file, viết mã và thực thi chương trình Python trên PyCharm

Bước 1. Tạo project: chọn menu file/New Project... Trong màn hình vừa xuất hiện, chọn folder nơi lưu trữ project, xong click button Create.

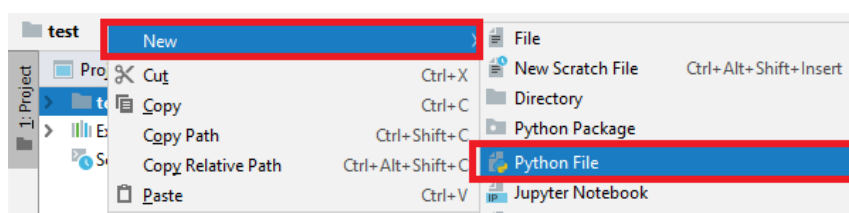


Sau khi quá trình trên được hoàn tất, Project mới sẽ được tạo ra và màn hình lúc này có dạng như hình sau:

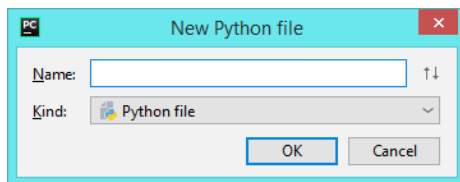


Bước 2. Tạo mới Python file

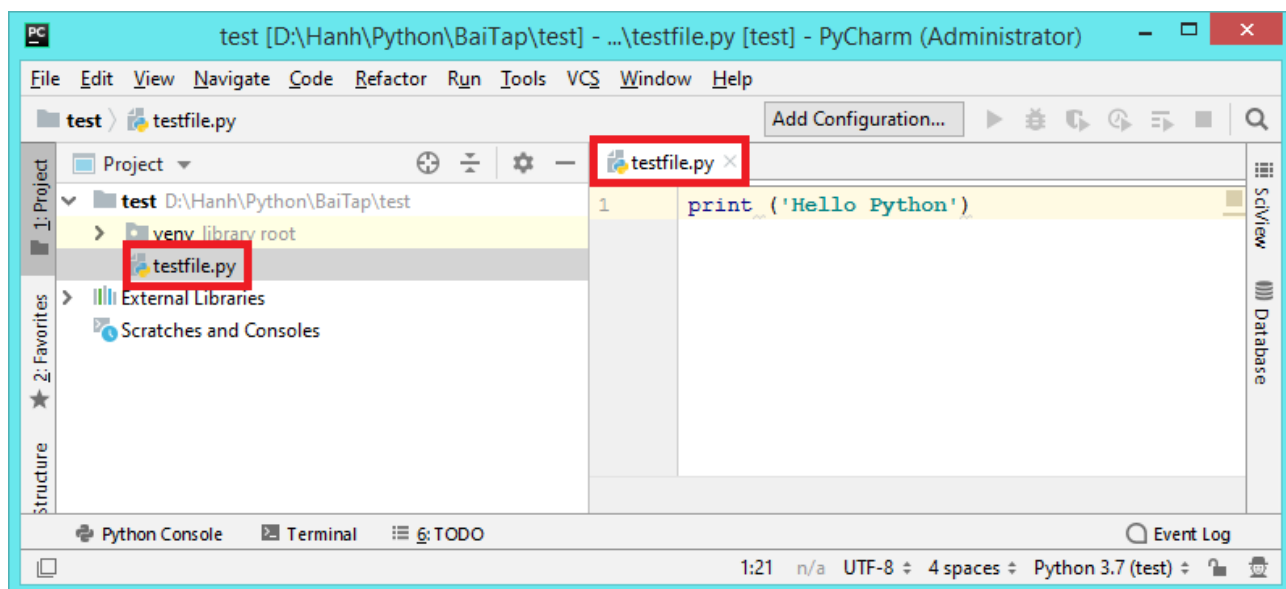
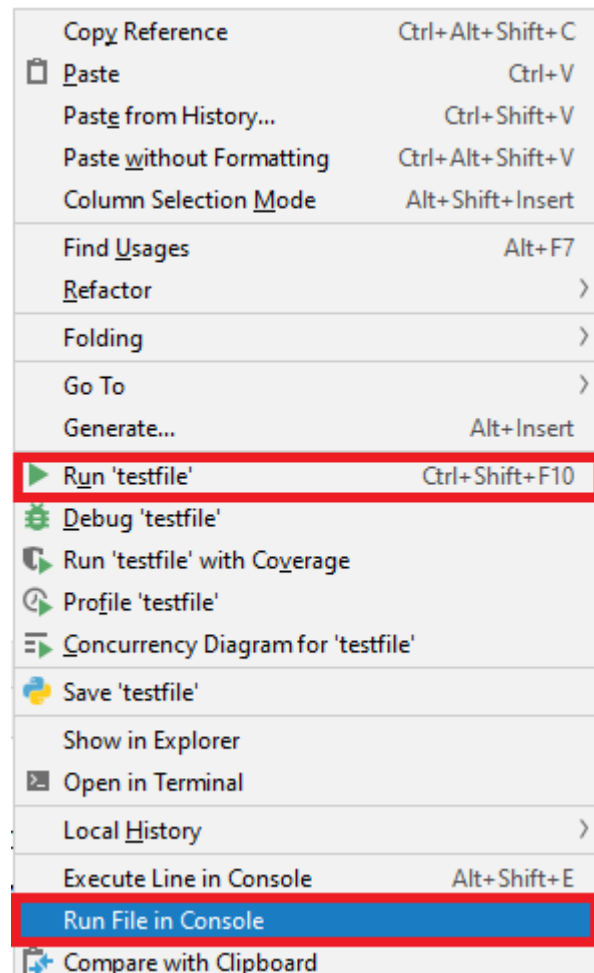
- Right Click vào tên project, chọn New/Python File



- Đặt tên cho file trong dialog sau:



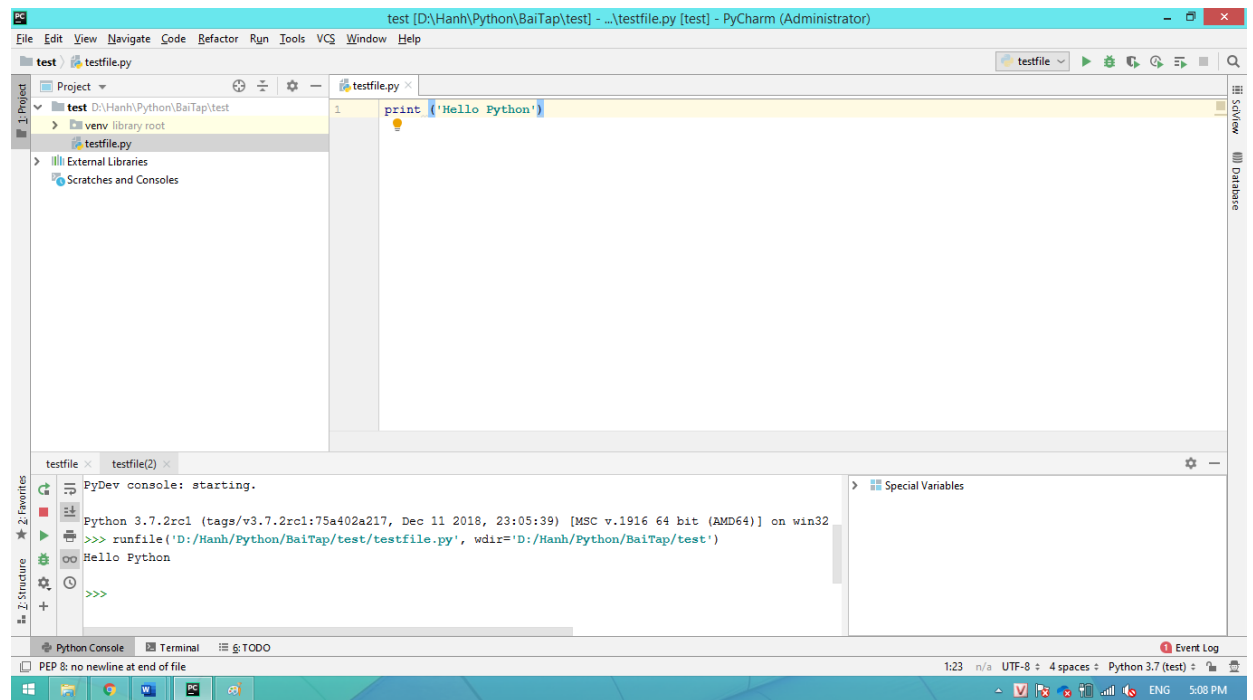
- Viết mã lệnh (ví dụ: `print("Hello Python")`)



Bước 3. Thực thi chương trình

Có thể thực thi chương trình bằng một trong các cách sau:

- Sử dụng tổ hợp phím tắt `Ctrl+Shift+F10`.
- Right click trên màn hình viết lệnh, chọn Run 'testfile' trong context menu
- Hoặc Run File in Console
- Kết quả thực thi sẽ xuất hiện trong cửa sổ cuối màn hình



NGÔN NGỮ LẬP TRÌNH PYTHON

2.1. Định danh (Identifier)

- Là tên được đặt cho biến(*variable*), phương thức/hàm (*function*), lớp (*class*), *module* và các đối tượng khác.
- *Identifier* phải bắt đầu bằng các ký tự A-Z, a-z hoặc `_`, tiếp đó là các ký tự, ký số 0-9. Không sử dụng các ký khác như `@`, `#`, `$`, `%`, ...
- *Identifier* có phân biệt chữ hoa, chữ thường.
- Một số quy tắc *identifier*:
 - Tên *class* bắt đầu bằng chữ hoa. Tất cả các *identifier* khác bắt đầu bằng chữ thường.
 - Tên *function* viết thường, các từ nối với nhau bằng dấu `_`
 - Không sử dụng các từ khóa (*keyword*) của *Python* khi đặt tên.

- Các từ khóa của *Python* (*Reserved Words*)

<code>and</code>	<code>continue</code>	<code>except</code>	<code>FALSE</code>	<code>is</code>	<code>or</code>	<code>try</code>
<code>as</code>	<code>def</code>	<code>exec</code>	<code>global</code>	<code>lambda</code>	<code>pass</code>	<code>while</code>
<code>assert</code>	<code>del</code>	<code>finally</code>	<code>if</code>	<code>None</code>	<code>raise</code>	<code>with</code>
<code>break</code>	<code>elif</code>	<code>for</code>	<code>import</code>	<code>nonlocal</code>	<code>return</code>	<code>yield</code>
<code>class</code>	<code>else</code>	<code>from</code>	<code>in</code>	<code>not</code>	<code>TRUE</code>	

2.2. Khối lệnh

- Dòng lệnh:
 - Mỗi lệnh nên được đặt trên 1 dòng riêng lẻ.
 - Khi có nhiều lệnh được đặt trên 1 dòng, các lệnh này phải cách nhau bởi dấu chấm phẩy (`;` - *Semicolon*).
 - Để viết 1 lệnh trên nhiều dòng, trừ dòng cuối của lệnh, ở cuối các dòng trước đó cần thêm dấu xuống phải (`\` - *Back slash*).
- Khối lệnh:
 - Các lệnh liên tiếp nhau được xem là cùng 1 khối lệnh khi được thụt lề (`()` như nhau, tức là có cùng số lượng khoảng trắng hoặc phím `tab`.
 - Khối lệnh đầu tiên của chương trình luôn phải được đặt sát lề trái.

```
1 x=5; print(x)
```

Khi có nhiều lệnh được đặt trên cùng 1 dòng, các lệnh này phải cách nhau bởi dấu chấm phẩy

```
1 x=5
2 print(x)
```

Khối lệnh đầu tiên được đặt sát lề trái

```
1 a=5
2 b=7
3 if a >= b:
4     print('a >= b')
5 else:
6     print('a < b')
```

Khối lệnh con được thụt lề đúng quy định

```
main.py x
1 x=5
2 print(x)
```

Lỗi do khối lệnh đầu tiên không được đặt sát lề trái

```
main.py x
1 a=5
2 b=7
3 if a >= b:
4     print('a >= b')
5 else:
6     print('a < b')
```

Lỗi do khối lệnh con của if else không được thụt lề

2.3. Biến

- Cần phải khai báo biến khi sử dụng.
- Khai báo biến bằng một câu lệnh gán. Ví dụ: `x=5`
- Có thể gán nhiều loại giá trị (số nguyên, số thập phân, chuỗi) cho một biến.

Ví dụ 2.1

Mã lệnh

Kết quả

<code>x=5</code> <code>print(x)</code>	5
<code>x="Hello"</code> <code>print(x)</code>	Hello

- Chú ý: có thể gán giá trị cho nhiều biến trên cùng 1 dòng lệnh.

Ví dụ 2.2 `a, b, c = 2, 3, 4`

#tương đương với gán từng dòng lệnh `a = 2; b = 3; c = 4`

- Xóa một biến đã có: sử dụng lệnh **del**

Ví dụ 2.3

Mã lệnh

Kết quả

<code>x=5</code> <code>print('x=', x)</code>	X= 5
del x <code>print('x=', x)</code>	<code>NameError: name 'x' is not defined</code>

2.4. Hằng số (constant)

- Hằng số là một loại biến có giá trị không thể thay đổi.
- Khai báo và gán giá trị cho hằng số trong Python
 - Tạo ra một module (là một file Python, giả sử đặt tên là `constants.py`), trong đó khai báo các hằng số với giá trị được gán sẵn.
 - Khi cần dùng, chương trình sẽ import file này ngay đầu chương trình.

- Ví dụ:

- Tạo file `constants.py` với nội dung khai báo các hằng số

```
PI = 3.14
GRAVITY = 9.8 #trọng lực
```

- Tạo file `main.py`, trong đó import module chứa các hằng số, sau đó sử dụng các hằng số này

```
import constants
print(constants.PI)
print(constants.GRAVITY)
```

Khi thực hiện chương trình, kết quả sẽ là:

```
3.14
9.8
```

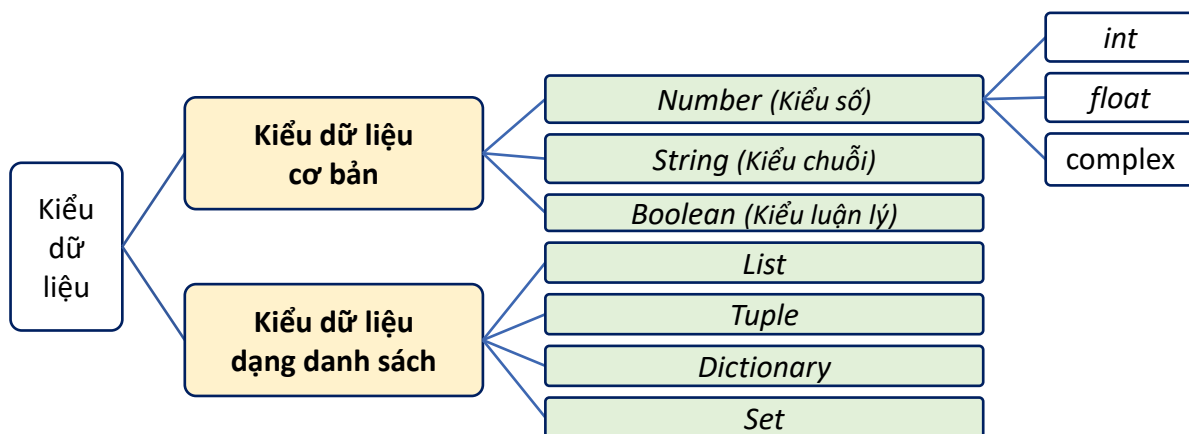
- Quy tắc và quy ước đặt tên cho các biến và hằng:

- Tạo một cái tên có ý nghĩa.
- Các hằng số được viết bằng tất cả các chữ in hoa và dấu gạch dưới phân tách các từ. Không sử dụng các ký hiệu đặc biệt như `!`, `@`, `#`, `$`, `%`, ... và không bắt đầu tên bằng một chữ số.

```
PI = 3.14
INCOME_TAX_RATE=0.1
MINIMUM_WAGE_IS_TAXABLE= 9000000
```

2.5. Các kiểu dữ liệu

2.5.1. Phân loại



2.5.2. Kiểu dữ liệu cơ bản

2.5.2.1. Number

- **int**:
 - Số lượng ký số của kiểu này là không giới hạn, chỉ bị phụ thuộc vào dung lượng của bộ nhớ.
 - Python hỗ trợ biểu diễn số nguyên dưới 3 dạng cơ số
 - *Decimal*: mặc định
 - *Octal*: thêm tiền tố **0o**
 - *Hexadecimal*: thêm tiền tố **0x**. Lưu ý: Python không phân biệt chữ hoa/chữ thường của tiền tố và ký tự thập lục phân, do đó **0X1A5F** và **0x1a5f** là như nhau.
 - Ví dụ 2.4: `5`, `-89`, `-0o490`, `0x5Cb7`, `-0X3A8f`
- **float**:
 - Có tối đa 15 số lẻ.
 - Ví dụ 2.5: `0.0`, `-12.38`, `7.52+e5`, `-98.71e100`
- **complex**:
 - Là 1 cặp số có thứ tự các số thực (*real floating point*) ký hiệu là `x + yj`, với `x` là *real* và `y` là *imag*
 - Ví dụ 2.6: `3+4j`, `3.14j`, `3e+26j`, `9.123456e-17j`

2.5.2.2. Boolean

- Chỉ nhận 1 trong 2 giá trị `True` hoặc `False`

Ví dụ 2.7

Mã lệnh	Kết quả
<code>result=True</code> <code>print(result)</code>	True
<code>not_result = not result</code> <code>print(not_result)</code>	False

2.5.2.3. String¹

- Là một chuỗi ký tự được đặt trong nháy kép (") hoặc nháy đơn (')
- Khai báo

(i)- Một chuỗi có thể khai báo bằng dấu nháy đôi (") hoặc nháy đơn ('). Ví dụ:

```
s1 = "Hello world"
name = 'SaiGon'
```

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

```
s3=""" Sai gon -
      thanh pho
      Ho Chi Minh"""
```

(iii)- Khai báo chuỗi UNICODE bằng cách thêm ký tự u trước chuỗi. Ví dụ:

```
s = u'Sài gòn'
hay s = u'\u0050\u0079\u0074\u0068\u006f\u0065\u0045\u0078\u0065\u0072
\u0063\u0069\u0073\u0065\u0073 \u002d \u0077\u0033\u0072\u0065\u0073
\u006f \u0075\u0072\u0063\u0065' # = w3resource
```

- Thao tác với string

- **Nối chuỗi: toán tử +**

Có thể tạo một chuỗi dài từ việc nối các chuỗi lại theo cú pháp:

skq = s1 + " " + s2

- **Lặp chuỗi: toán tử ***

Ví dụ 2.8

Mã lệnh	Kết quả
<code>one = "Hello"</code> <code>two = "Python"</code> <code>S = one + " " + two</code> <code>print(S)</code>	Hello Python
<code>print(S*3)</code>	Hello Python Hello Python Hello Python

2.6. Chuyển đổi kiểu dữ liệu

- Các phương thức

Phương thức	Mô tả
<code>int(x[,base])</code>	Chuyển chuỗi số thành <i>integer</i> theo cơ số chỉ định (nếu có)
<code>float(x[,base])</code>	Chuyển chuỗi số thành <i>float</i> theo cơ số chỉ định (nếu có)
<code>str(x)</code>	Chuyển đối tượng x thành chuỗi
<code>eval(String)</code>	Đánh giá kiểu dữ liệu của 1 chuỗi và trả về 1 <i>object</i> của kiểu được đánh giá
<code>complex(real[,imag])</code>	Chuyển chuỗi số thành <i>complex number</i>
<code>repr(x)</code>	Chuyển đối tượng x thành 1 chuỗi – <i>expression string</i>

¹ Xem thêm mục 2.12. Thao tác với đối tượng kiểu string trong chương này

chr(x)	Chuyển <i>integer</i> x thành ký tự tương ứng (mã ASCII)
ord(x)	Chuyển ký tự x thành giá trị mã ASCII của ký tự
hex(x)	Chuyển 1 <i>integer</i> x thành chuỗi <i>hexadecimal</i>
oct(x)	Chuyển 1 <i>integer</i> x thành chuỗi <i>octal</i>

– Ví dụ 2.9

Mã lệnh	Kết quả
strInt = "15" print(int(strInt)*2)	30
real=25 emag= 3.2 complexNumber=complex(real,emag) print(complexNumber)	(25+3.2j)
numEval=eval(strInt) print(numEval+numEval)	30
print(chr(98))	'b'
print(ord('c'))	99
print(oct(100))	0o144

2.7. Các toán tử (operators)


2.7.1. Toán tử số học (Arithmetic operators)

Toán tử	Mô tả	Ví dụ (cho a=5, b=2)	
		Biểu thức	Kết quả
+	phép cộng	a+b	7
-	phép trừ	a-b	3
*	phép nhân	a*b	10
**	Lũy thừa	a**b	25
/	phép chia	a/b	2.5
//	Chia làm tròn cận dưới (<i>Floor Division</i>)	a//b	2
%	phép chia lấy dư (modulo)	a%b	1

2.7.2. Toán tử gán (Assignment Operators)

Toán tử	Mô tả	Ví dụ 2.100 (cho a=5, b=2)	
		Biểu thức	Kết quả a
=	Gán giá trị của toán hạng thứ 2 (bên phải) cho toán hạng thứ 1 (bên trái)	a=b	2
+=	Cộng giá trị của toán hạng sau vào toán hạng đầu và gán kết quả cho toán hạng đầu	a+=b ↔ a=a+b	7
-=	Trừ giá trị của toán hạng sau khỏi toán hạng đầu và gán kết quả cho toán hạng đầu	a-=b ↔ a=a-b	3
=	Nhân giá trị của toán hạng sau với toán hạng đầu và gán kết quả cho toán hạng đầu	a=b ↔ a=a*b	10
=	Thực hiện phép tính số mũ và gán kết quả cho toán hạng đầu	a=b	25

		$\Leftrightarrow a = a ** b$	
/=	Chia giá trị của toán hạng đầu cho giá trị của toán hạng sau và gán kết quả cho toán hạng đầu	$a /= b$ $\Leftrightarrow a = a / b$	2.5
//=	Thực hiện phép chia lấy cận dưới giữa toán hạng đầu và toán hạng sau, và gán kết quả cho toán hạng đầu	$a //= b$ $\Leftrightarrow a = a // b$	2
%=	Thực hiện phép chia lấy dư giữa toán hạng đầu và toán hạng sau, và gán kết quả cho toán hạng đầu	$a \% = b$ $\Leftrightarrow a = a \% b$	1

 Một số lưu ý với phép gán bằng (=)

- Có thể cùng lúc gán nhiều giá trị cho nhiều biến trên cùng 1 phép gán bằng (=). Do đó, hai đoạn mã lệnh sau là tương đương nhau:

```
a = 5
b = 'Sai Gon'
c = 3.14
```

\Leftrightarrow a, b, c = 5, 'Sai Gon', 3.14

- Vận dụng: có thể hoán vị giá trị của 2 biến

Mã lệnh	Kết quả
<pre>a = 3 b = 2 print("Before swaping a=%d and b=%d" %(a,b)) a, b = b, a print("After swaping a=%d and b=%d" %(a,b))</pre>	<p>Before swaping a=3 and b=2</p> <p>After swaping a=2 and b=3</p>

2.7.3. Toán tử so sánh (Comparison Operators)

- So sánh giá trị của hai toán hạng (hoặc biểu thức), kết quả trả về là kiểu *boolean* (True hoặc False).
- Ứng dụng khi cần so sánh trên hai toán hạng (hoặc biểu thức) có kiểu số hoặc *boolean*.
- Được sử dụng trong các cấu trúc điều kiện và cấu trúc lặp (*if*, *while* và *for*)
- Các phép so sánh

Toán tử	Mô tả	Ví dụ 2.11 (cho a=5, b=2)	
		Biểu thức	Kết quả
<	Bé hơn	a < b	False
<=	Bé hơn hay bằng	a <= b	False
>	Lớn hơn	a > b	True
>=	Lớn hơn hay bằng	a >= b	True
==	Bằng	a == b	False
!=	Khác	a != b	True

- Hỗ trợ dạng so sánh 3 ngôi (so sánh kép). Ví dụ 2.12

Mã lệnh	Kết quả
x=5	
print (0 < x < 10)	True
print (5 > x >= 10)	False
print (5 >= x < 10)	True
print (5 != x < 10)	False
print (5 == x < 10)	True

2.7.4. Toán tử logic (Logical Operators)

- Giá trị đúng và sai tương ứng là *True* và *False*.
- Toán tử logic:

Toán tử	Mô tả	Ví dụ 2.13 (cho $a=5, b=8, c=7, d=True$)		
		Biểu thức	Trương đương với	Kết quả
and	phép tính logic và	$a \leq c$ and $c \leq b$	True and True	True
		$a \leq c$ and $c > b$	True and False	False
or	phép tính logic hoặc	$a \leq c$ or $c \leq b$	True or True	True
		$a > c$ or $c > b$	False or False	False
not	Phủ định	not d	not True	False

2.7.5. Toán tử thành phần (Membership Operators)

- Kết quả trả về là *True* hoặc *False*.

Toán tử	Mô tả	Ví dụ 2.14	
		Biểu thức	Kết quả
in	kiểm tra có tồn tại	<code>print("Sai gon" in "sai gon - TP.Ho Chi Minh")</code>	False
		<code>print("Sai gon" in "Sai gon - TP.Ho Chi Minh")</code>	True
		<code>a=10 myList=[1,2,3,4,5] print(a in myList)</code>	False
not in	kiểm tra không tồn tại	<code>print("Sai gon" not in "Sai gon-TP.Ho Chi Minh")</code>	False
		<code>b=6 myList=[1,2,3,4,5] print(b not in myList)</code>	True

2.7.6. Toán tử bit (Operators)

- Giá trị đúng và sai tương ứng là *True* và *False*.
- Nhắc lại toán tử *Bitwise* (Bitwise operator)

$op1$	$op2$	$op1 \& op2$ (AND)	$op1 op2$ (OR)	$op1 \wedge op2$ (XOR)
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

- Toán tử logic trên bit:

Toán tử	Mô tả	Ví dụ 2.15 (cho $a=13, b=10, c=2$)									
		Biểu thức	Kết quả								
&	AND	$a \& b$	Giá trị	128	64	32	16	8	4	2	1
			a	0	0	0	0	1	1	0	1
			b	0	0	0	0	1	0	1	0
			$a \& b$	0	0	0	0	1	0	0	0

	OR	a b	Giá trị	128	64	32	16	8	4	2	1
			a	0	0	0	0	1	1	0	1
			b	0	0	0	0	1	0	1	0
			a b	0	0	0	0	1	1	1	1
^	XOR	a^b	Giá trị	128	64	32	16	8	4	2	1
			a	0	0	0	0	1	1	0	1
			b	0	0	0	0	1	0	1	0
			a^b	0	0	0	0	0	1	1	1
~	Là đảo (hoặc lật) bit sao cho ~a + a = -1	~a	a 0b1101 ~a -0b1110								
<<	SHIFT LEFT	a<<=c	Giá trị	128	64	32	16	8	4	2	1
			a	0	0	0	0	1	1	0	1
			a<<1	0	0	0	1	1	0	1	0
			a<<2	0	0	1	1	0	1	0	0
>>	SHIFT RIGHT	a>>=c	Giá trị	128	64	32	16	8	4	2	1
			a	0	0	0	0	1	1	0	1
			a>>1	0	0	0	0	0	1	0	1
			a>>2	0	0	0	0	0	0	1	0

2.7.7. Toán tử định danh (Identity Operators)

Toán tử	Mô tả
is	Trả về <i>True</i> nếu các biến ở 2 bên cùng trỏ đến một đối tượng, ngược lại trả về <i>False</i>
is not	Trả về <i>True</i> nếu các biến ở 2 bên KHÔNG cùng trỏ đến một đối tượng, ngược lại trả về <i>False</i>

Ví dụ 2.16

Mã lệnh	Kết quả
<pre>a = 20 b = 20 if (a is b): print "Line 1 - a and b have same identity" else: print "Line 1 - a and b DO NOT have same identity"</pre>	Line 1 - a and b have same identity
<pre>if (id(a) == id(b)): print "Line 2 - a and b have same identity" else: print "Line 2 - a and b DO NOT have same identity"</pre>	Line 2 - a and b have same identity
<pre>b = 30 if (a is b): print "Line 3 - a and b have same identity"</pre>	Line 3 - a and b DO NOT have same identity

else:	
print "Line 3 - a and b DO NOT have same identity"	
if (a is not b):	
print "Line 4 - a and b DO NOT have same identity"	Line 4 - a and b DO NOT
else:	have same
print "Line 4 - a and b have same identity"	identity

2.7.8. Độ ưu tiên của các toán tử (Operators Precedence)

Độ ưu tiên được tính từ trên xuống dưới và từ trái qua phải

Operator	Mô tả
**	Lũy thừa
* / % //	Nhân, chia, chia lấy dư, chia lấy cận dưới
+ -	Cộng, trừ
>> <<	Shift right, shift left
&	AND trên bit
^ 	XOR, OR trên bit
<= < > >=	Nhỏ hơn hay bằng, nhỏ hơn, lớn hơn, lớn hơn hay bằng
<> == !=	Khác, gán, khác
= %= /= //=- += *= **=	Các toán tử gán
is is not	Các toán tử định danh
in not in	Các toán tử thành phần
not or and	Các toán tử logic

2.8. Console Input-Output

2.8.1. Phương thức print()

2.8.1.1. Cú pháp

```
print(objects[, sep=' ', end='\n', file=sys.stdout, flush=False])
```

Trong đó:

- **objects:**
 - đối tượng được in, có thể có nhiều đối tượng. Sử dụng dấu phẩy (,) hoặc dấu cộng (+) để nối các đối tượng. Tất cả sẽ được chuyển đổi thành chuỗi trước khi hiển thị ra màn hình.
 - Theo mặc định phương thức print sẽ in chuỗi theo bộ mã ASCII. Để xuất chuỗi theo UNICODE, sử dụng tiền tố **u** ngay phía trước chuỗi. Ví dụ

```
print (u'Hello Python').
```

- **sep** : ký tự dùng để ngăn cách các đối tượng có trong *objects*, giá trị mặc định là một khoảng trắng (' ').
- **end** : giá trị cuối cùng được in ra màn hình, giá trị mặc định là “\n” (xuống dòng).
- **file** : phải là 1 đối tượng với phương thức **write(string)**. Nếu bỏ qua, **sys.stdout** sẽ được dùng, nghĩa là in ra màn hình.
- **flush** : Giá trị mặc định là *False*

2.8.1.2. Các cách xuất bằng hàm print

- (i)- Sử dụng dấu phẩy (,) hoặc dấu cộng (+) để nối chuỗi trong hàm **print**.
- (ii)- Sử dụng định dạng % hoặc **f'{variableName}'** để xuất chuỗi kèm với biến.
- (iii)- Sử dụng cặp ngoặc nhọn ({} - curly brackets) và phương thức **format** của dữ liệu kiểu chuỗi theo dạng thức **"{@1:>@2} ".format(variables)**. Trong đó:
 - ① nếu có cho biết thứ tự các đối số được cung cấp trong danh sách *variables*, thứ tự này được đánh số từ 0.
 - ② có thể dùng 1 trong các dạng sau:
 - **{:>number}**: cho biết khoảng cách dành để in giá trị của biến là *number*.
 - Nếu giá trị của biến có chiều dài lớn hơn *number* thì lấy theo chiều dài thực tế của biến.
 - Ngược lại, khi giá trị của biến có chiều dài nhỏ hơn *number* thì thêm các khoảng trắng ngay trước giá trị biến để kết quả in ra có chiều dài bằng *number*.
 - **{:.Xf}** : cho biết lấy *X* số lẻ của số thực cần in.
 - **{:W.Xf}** : cho biết in số thực trong độ rộng =*W* với *X* số lẻ.
 - **{:+.Xf}** : cho biết lấy *X* số lẻ của số thực cần in và có in dấu (âm hay dương) trước giá trị của biến.
 - **{:<Xd|f}** : in canh trái. Ký hiệu **d|f** tùy thuộc kiểu dữ liệu là *int* hay *float*.
 - **{:Xd|f}** : in canh phải. Ký hiệu **d|f** tùy thuộc kiểu dữ liệu là *int* hay *float*.
 - **{:^Xd|f}** : in canh giữa. Ký hiệu **d|f** tùy thuộc kiểu dữ liệu là *int* hay *float*.
 - **{:0>Xd}** : in số nguyên sau khi chèn các số 0 bên trái sao cho đủ chiều dài *X*.
 - **{:C<Xd}** : in số nguyên sau đó in thêm các ký tự *C* bên phải của số nguyên cho đến khi đủ chiều dài *X*.
 - **{:,}**: in dấu phân cách phần ngàn. Chỉ chấp nhận dấu phẩy (,) làm dấu ngăn cách phần ngàn.
 - **{:.X%}**: định dạng in số dưới dạng phần trăm (%) với *X* số lẻ.
 - Cho phép các cặp ngoặc nhọn được lồng nhau.
 - **variables**: Danh sách các biến được sử dụng. Quy ước số thứ tự các biến được tính từ 0 và đánh số từ trái qua phải.
- (iv)- Canh lề cho chuỗi cần xuất:
 - **'{:<20s}'.format(strObj)** : canh trái đoạn văn bản *strObj*

- Hoặc `'%s' %strObj` : (mặc định) canh trái đoạn văn bản `strObj`
 □ `strObj.center(20, ' ')` : canh giữa đoạn văn bản `strObj`
 □ `'%20s' %(strObj)` : canh phải đoạn văn bản `strObj`

Ví dụ 2.17 (ký hiệu \cup đại diện cho khoảng trắng)

Mã lệnh

Kết quả

(i) Sử dụng nối chuỗi

```
'''Theo mặc định tham số sep của print là khoảng trắng'''
```

```
print(4, "world")
```

4 \cup world

```
'''Nối chuỗi'''
```

```
print("4" + 'world')
```

4world

```
print(str(4) + "world")
```

4world

(ii) Sử dụng định dạng % hoặc f'{variableName}'

```
'''Sử dụng định dạng % để xuất giá trị của biến name'''
```

```
name=input("What's your name?")
print("Hi %s" %name)
```

Hi Nam

```
'''Sử dụng định dạng % để xuất giá trị của biến hoặc biểu thức'''
```

```
a, b = 5, 3
print('%d + %d = %d' % (a,b,a+b))
```

5 + 3 = 8

```
'''Sử dụng định dạng f'{} để xuất giá trị của biến hoặc biểu thức'''
```

```
print(f'{a} + {b} = {a+b}')
```

5 + 3 = 8

(iii) Sử dụng phương thức format

```
''' xuất giá trị của biến hoặc biểu thức'''
```

```
a, b = 5, -3.333
print('{} + {} = {}'.format(a,b,a+b))
```

5 + -3.333 = 1.667

#hoặc có thể đảo thứ tự các biến/biểu thức, nhưng khi đó bắt buộc phải dùng số thứ tự của biến/biểu thức trong các cặp ngoặc nhọn

```
a, b = 5, -3.333
print('{2} + {0} = {1}'.format(b, a+b, a))
```

5 + -3.333 = 1.667

'''Xuất giá trị của biến hoặc biểu thức với tên trong cặp ngoặc nhọn (x,y,z) là do người lập trình tự đặt. a và b là các biến đã được gán giá trị trước đó'''

```
print('{x} + {y} = {z}'.format(x=a,y=b,z=a+b))
```

5 + -3.333 = 1.667

'''Xuất giá trị của biến kèm dấu (+ hoặc - tùy thuộc giá trị thực tế)'''

```
a, b = 5, -3.333
print('{} {:+.2f}={:+.0f}'.format(a,b,a+b))
```

5 -3.33 = 2

<code>a, b = 5, -3.333</code> <code>print('{ } {:+.2f} ={:+.0f}'.format(a,b,a+b))</code>	<code>5 -3.33 = +2</code>
'''Canh phải giá trị số'''	
<code>print('{0:>5} + {1:>8} =</code> <code>{2:>8}'.format(a,b,a+b))</code>	<code>~~~~5 + ~~-3.333 =</code> <code>1.6669999999999998</code>
'''Canh phải giá trị số với 2 số lẻ'''	
<code>n=2.345</code> <code>print('{:^10.2f}'.format(n))</code>	<code>~~~~~2.34</code>
'''Canh trái giá trị số'''	
<code>n=2.345</code> <code>print('{:<10.2f}'.format(n))</code>	<code>2.34~~~~~</code>
'''Sử dụng phương thức format để canh giữa giá trị số'''	
<code>print('{:10.2f}'.format(n))</code>	<code>~~~~2.34~~~</code>
'''Điền số 0 trước giá trị số cho đến khi tổng chiều dài là 4'''	
<code>print('{:0>4d}'.format(a))</code>	<code>0005</code>
'''Điền ký tự @ sau giá trị số cho đến khi tổng chiều dài là 4'''	
<code>print('{:@<4d}'.format(a))</code>	<code>5@@@@</code>
'''Định dạng hiển thị giá trị của biến với dấu ngăn cách phần ngàn và phần lẻ'''	
<code>n=1234567.892</code> <code>print('{:, .1f}'.format(n))</code>	<code>1,234,567.9</code>
'''Định dạng hiển thị giá trị của biến dưới dạng số phần trăm, lấy 2 số lẻ'''	
<code>n=12.345</code> <code>print('{:, .2%}'.format(n))</code>	<code>1,234.50%</code>
'''Định dạng hiển thị giá trị của biến với số lượng số lẻ cũng là 1 biến và in kèm ngay sau giá trị là cm²'''	
<code>dientich = 1256.45678</code> <code>decimals = 2</code> <code>print("Diện tích hình chữ nhật =</code> <code>{0:.{1}f}cm\u00b2".format(dientich, decimals))</code>	<code>Diện tích hình chữ</code> <code>nhật = 1256.46cm²</code>
'''Định dạng hiển thị giá trị của biến với số lượng số lẻ cũng là 1 biến và in kèm ngay sau giá trị là cm³'''	
<code>thetich = 1254.725</code> <code>decimals = 3</code> <code>print("Thể tích hình trụ=</code> <code>{0:.{1}f}cm\u00b3".format(thetich, decimals))</code>	<code>Thể tích hình trụ=</code> <code>1254.725cm³</code>

(iv) Canh lề cho chuỗi cần xuất

```
'''canh trái text, khi len(text)<20 thì thêm khoảng trắng vào cuối để tổng chiều dài đủ 20 '''
```

```
text = 'Hello Python'
print('{:~20s}'.format(text))
```

Hello Python~~~~~

```
'''canh giữa text, khi len(text)<20 thì thêm khoảng trắng vào 2 đầu để tổng chiều dài đủ 20 '''
```

```
print(text.center(20, '~'))
```

~~~~Hello Python~~~~

```
'''canh phải text, khi len(text)<20 thì thêm khoảng trắng vào đầu để tổng chiều dài đủ 20 '''
```

```
print('%20s' % (text))
```

~~~~~Hello Python

2.8.1.3. Các định dạng được dùng kèm trong hàm print

– Backslash sign(dấu \):

- Chuỗi “\n” sử dụng trong hàm print để xuống dòng văn bản.
- Sử dụng \ trước ký tự đặc biệt. Ví dụ: `print ("what\'s your age?")`.
- Các định dạng gồm:

| Định dạng | Ý nghĩa |
|-----------------|----------------------|
| <code>\\</code> | Backslash (\) |
| <code>\'</code> | Single quote (') |
| <code>\"</code> | Double quote (") |
| <code>\a</code> | ASCII Bell (BEL) |
| <code>\b</code> | ASCII Backspace (BS) |
| <code>\f</code> | ASCII Formfeed (FF) |

| Định dạng | Ý nghĩa |
|----------------------|--------------------------------------|
| <code>\n</code> | ASCII Linefeed (LF) |
| <code>\r</code> | ASCII Carriage Return (CR) |
| <code>\t</code> | ASCII Horizontal Tab (TAB) |
| <code>\v</code> | ASCII Vertical Tab (VT) |
| <code>\ooo</code> | ASCII character with octal value ooo |
| <code>\xhh...</code> | ASCII character with hex value hh... |

– Percent sign (%): gồm các định dạng sau:

| Ký tự | Ý nghĩa |
|---------------------|--|
| <code>%c</code> | Ký tự đơn |
| <code>%d, %i</code> | Số nguyên thập phân có dấu |
| <code>%e</code> | Số chấm động (ký hiệu có số mũ – with lowercase letters) |
| <code>%E</code> | Số chấm động (ký hiệu có số mũ – with UPPERCASE letters) |
| <code>%f</code> | Số chấm động (ký hiệu thập phân) |
| <code>%g</code> | Định dạng ngắn gọn của %f và %e |
| <code>%G</code> | Định dạng ngắn gọn của %f và %E |
| <code>%o</code> | Số nguyên hệ bát phân (Octal integer) |
| <code>%p</code> | Con trỏ (pointer) |

| | |
|-----------|---|
| %r | String (chuyển đổi bất kỳ đối tượng của Python bằng cách sử dụng hàm <code>repr()</code>). |
| %s | <ul style="list-style-type: none"> String (chuyển đổi bất kỳ đối tượng của Python bằng cách sử dụng <code>str()</code>). Nếu đối tượng hoặc định dạng được cung cấp là một chuỗi unicode, chuỗi kết quả cũng sẽ là unicode. Vì các chuỗi Python có độ dài rõ ràng, chuyển đổi <code>%s</code> không xem <code>'\0'</code> là kết thúc của chuỗi. |
| %u | Số nguyên không dấu |
| %x | Số nguyên hệ thập lục (Hexadecimal – with lowercase letters) |
| %X | Số nguyên hệ thập lục (Hexadecimal – with UPPERCASE letters) |
| %% | Xuất ra ký hiệu phần trăm (%) |

– Ví dụ 2.18 (ký hiệu `\n` đại diện cho khoảng trắng)

| Mã lệnh | Kết quả |
|---|---|
| <code>num = 1234.567</code> | |
| <code>print('num= %f' % num)</code> | num= 1234.567000 |
| <code>print('num= %.2f' % num)</code> | num= 1234.57 |
| <code>print('num= %.0f' % num)</code> | num= 1235 |
| <code>print('num= %, .1f' % num)</code> | num= 1,234.6 |
| <code>print('num= %,10.1f' % num)</code> | num= 1,234.6 |
| <code>str_num = "1234567890"</code> | |
| <code>print('%s' % str_num)</code> | 1234567890 |
| <code>print('%.6s' % str_num)</code> | 123456 |
| <code>print('%.11s' % str_num)</code> | 1234567890 |
| <code>name = 'Tý'</code> | |
| <code>age=19</code> | |
| <code>mark= 7.25</code> | |
| <code>print('Điểm thi %.1f là của học sinh %s, %d tuổi' % (mark, name, age))</code> | Điểm thi 7.2 là của học sinh Tý 19 tuổi |

2.8.2. Xóa màn hình (console)

Để xóa màn hình console, cần dùng lệnh sau:

```
import os          #import module os
os.system('cls')
```

2.8.3. Hàm `input()`

– Hàm `input` cho phép đợi người dùng nhập dữ liệu và kết thúc việc nhập khi phím **ENTER** được nhấn. Hàm trả về chuỗi do người dùng nhập vào (kể cả khi toàn bộ dữ liệu người dùng nhập vào đều là ký số).

– Cú pháp: `input(prompt)`

Trong đó: `prompt` là chuỗi thông báo sẽ được in ra màn hình.

– Ví dụ 2.19

```
name = input("Nhập tên: ")
print("Bạn tên là " + name)
```

- Sử dụng kết hợp hàm input với các hàm khác:
 - Hàm **eval**: Đánh giá kiểu dữ liệu của 1 chuỗi và trả về 1 object của kiểu được đánh giá. Thường dùng khi muốn chuyển dữ liệu nhập vào (đang thuộc kiểu String) sang kiểu dữ liệu khác (như int, float, ...).
 - Hàm chuyển kiểu dữ liệu như: **int**, **float**, ...
 - Ví dụ 2.20: kết quả thực hiện hai đoạn mã lệnh sau là tương đương.

| Mã lệnh | Mã lệnh tương đương |
|--|--|
| <pre>name = input("Nhập ten: ") age= eval(input("Tuoi: ")) mark= eval(input("Diem: ")) print('%s đang %d tuoi, có diem=%f' %(name, age, mark))</pre> | <pre>name = input("Nhập ten: ") age= int(input("Tuoi: ")) mark= float(input("Diem: ")) print('%s đang %d tuoi, có diem=%f' %(name, age, mark))</pre> |

2.9. Chú thích (comment) trong Python

- Chú thích (comment) là một hoặc nhiều dòng chứa một trong những nội dung sau:
 - Ghi chú riêng.
 - Phần chú thích có thể ghi thông tin tác giả, ngày viết, version.
 - Che đoạn mã lệnh tạm thời không sử dụng.
 - Giải thích cách xử lý của một đoạn chương trình, ...
- Sử dụng:
 - **#** (hash): các ký tự đi sau dấu này trên cùng dòng sẽ được xem là ghi chú.
 - **"""comment"""** hay **'''comment'''** cho phép ghi chú trên nhiều dòng.
 - Ví dụ 2.21

```
''' Module 1 of Python
Created on Feb 14, 2020
@author: Le Van Hanh
'''
mark = 7.25 # floating point
age = 19    #integer
city = "Sai Gon" #String
```

2.10. Cấu trúc điều kiện

Python hỗ trợ một số cấu trúc điều khiển thông dụng. Tất cả các cấu trúc điều khiển đều dựa vào thụt đầu dòng (indentation) để tạo thành một block xử lý, thay vì sử dụng {...} như các ngôn ngữ khác (C, C++, C#, PHP, Javascript).

Ví dụ 2.22 cho 2 biến a và b có kiểu là số nguyên. In ra màn hình so sánh giữa 2 biến này:

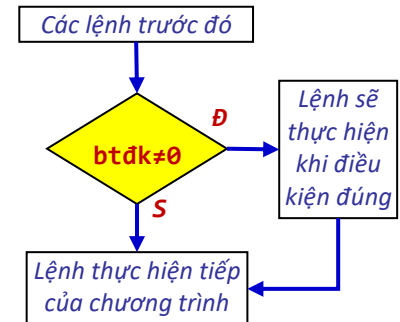
| C | Python |
|---|---|
| <pre>if (a>=b) { printf("a >= b"); } else { printf("a < b"); }</pre> | <pre>if a >= b: print('a >= b') else: print('a < b')</pre> |

2.10.1. if

- Cấu trúc điều kiện được sử dụng trong trường hợp việc tính toán trong chương trình có phụ thuộc vào giá trị của một điều kiện, khi điều kiện này đúng thì thực hiện một số câu lệnh nào đó, và nếu điều kiện sai thì lại thực hiện một số câu lệnh khác.

- Cú pháp:

- **Only if:** Kiểm tra điều kiện, nếu kết quả của điều kiện là đúng thì thực hiện thêm một số hành động trước khi tiếp tục thực thi chương trình



```

if condition:
    Statement(s) block for Condition is True
  
```

Ví dụ 2.23

```

so= eval(input("Nhập so: "))
if so < 0:
    so=abs(so)
print ("Trị tuyệt đối là %d" &so)
  
```

- **If ... else:** Được dùng để lựa chọn một trong hai nhánh thi hành của chương trình.

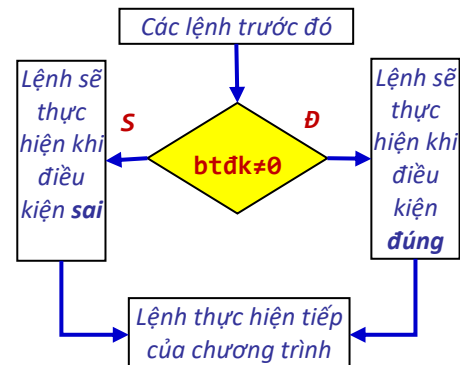
```

if condition1:
    Statement(s) block for Condition is True
else:
    Statement(s) block for Condition is False
  
```

Ví dụ 2.24

```

mark=eval(input('Nhập điểm: '))
if mark >= 5:
    print ("Đậu")
else:
    print ("Rớt")
  
```



- **Các lệnh if lồng trong lệnh if khác**

- **Dạng 1:**

```

if condition1:
    if condition2:
        Statement(s) Block For Condition1 is True
    else
        Statement(s) Block For Condition2 is True
else:
    if condition3:
        Statement(s) Block For Condition3 is True
    else:
        Statement(s) Block For Each Condition is False
  
```


▫ **Dạng 2:**

```

if condition1:
    Statement(s) Block For Condition1 is True
elif condition2:
    Statement(s) Block For Condition2 is True
elif condition3:
    Statement(s) Block For Condition3 is True
else:
    Statement(s) Block For Each Condition is False

```

Ví dụ 2.25

| Mã lệnh | Mã lệnh tương đương |
|--|--|
| <pre> mark=eval(input('Nhap diem: ')) if mark<0 or mark>10: print('Diem khong hop le') else: if mark >= 9: print ("Xuat sac") else: if mark >= 8: print ("Gioi") else: if mark >= 7: print ("Kha") else: if mark >= 6: print ("Trung binh - Kha") else: if mark >= 5: print ("Trung binh") else: print ("Yeu / Kem") </pre> | <pre> mark=eval(input('Nhap diem: ')) if mark<0 or mark>10: print('Diem khong hop le') elif mark >= 9: print ("Xuat sac") elif mark >= 8: print ("Gioi") elif mark >= 7: print ("Kha") elif mark >= 6: print ("Trung binh - Kha") elif mark >= 5: print ("Trung binh") else: print ("Yeu / Kem") </pre> |

2.10.2. switch...case

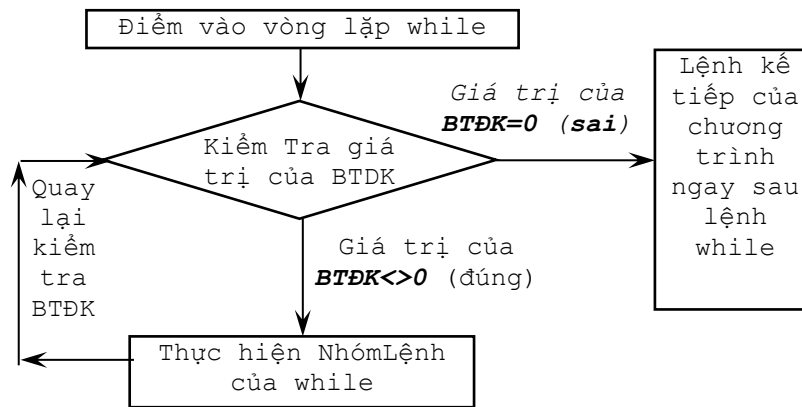
Python không có cấu trúc switch... case như các ngôn ngữ khác.

2.11. Cấu trúc lặp

- Một cấu trúc lặp gồm một câu lệnh hay một khối lệnh sẽ thi hành lặp lại cho tới khi biểu thức điều kiện sai
- Có hai loại cấu trúc lặp trong Python:
 - *while*
 - *for*

2.11.1. while

- Thực hiện việc lặp các lệnh trong thân của lệnh *while* khi điều kiện còn đúng (điều kiện được kiểm tra trước khi các lệnh được thi hành).



- **Cú pháp:** `while (expression):`
`statement(s)`

Ví dụ 2.26: in các số từ 5 đến 1.

| Mã lệnh | Kết quả |
|----------------------------------|-----------|
| <code>a = 5</code> | number: 5 |
| <code>while (a > 0):</code> | number: 4 |
| <code>print('number:', a)</code> | number: 3 |
| <code>a -= 1</code> | number: 2 |
| | number: 1 |

Ví dụ 2.27: in các số từ 1 đến 5

| | |
|----------------------------------|-----------|
| <code>a=1</code> | number: 1 |
| <code>while (a<=5):</code> | number: 2 |
| <code>print('number:', a)</code> | number: 3 |
| <code>a += 1</code> | number: 4 |
| | number: 5 |

- Ví dụ 2.28: yêu cầu người dùng nhập chuỗi “enter” (có phân biệt chữ hoa/chữ thường). Nếu nhập đúng, chương trình sẽ kết thúc, ngược lại, chương trình sẽ yêu cầu người dùng nhập lại:

```
quit = input('Type "enter" to quit:')
while quit != "enter":
    quit = input('Type "enter" to quit:')
```

- Ví dụ 2.29: (vòng lặp vô hạn - *Infinite loops*): là lệnh lặp chạy mãi mà không bao giờ dừng. Trường hợp này thường xảy ra khi người lập trình không để ý đến điều kiện thực hiện của lệnh lặp while:

```
i = 5
while i > 0:
    print("Inside the loop")
```

2.11.2. for

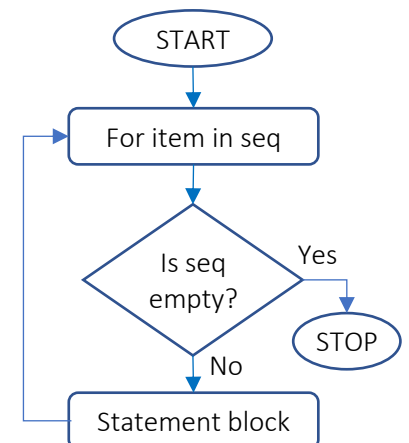
- Thực hiện việc lặp bằng cách duyệt qua các phần tử có trong tập hợp.

- **Cú pháp**

`for iterating_var in sequence:`
`statement(s)`

– Ví dụ 2.30

| Mã lệnh | Kết quả |
|------------------------------------|---------|
| <code>for i in range(1, 5):</code> | 1 |
| <code>print (i)</code> | 2 |
| | 3 |
| | 4 |



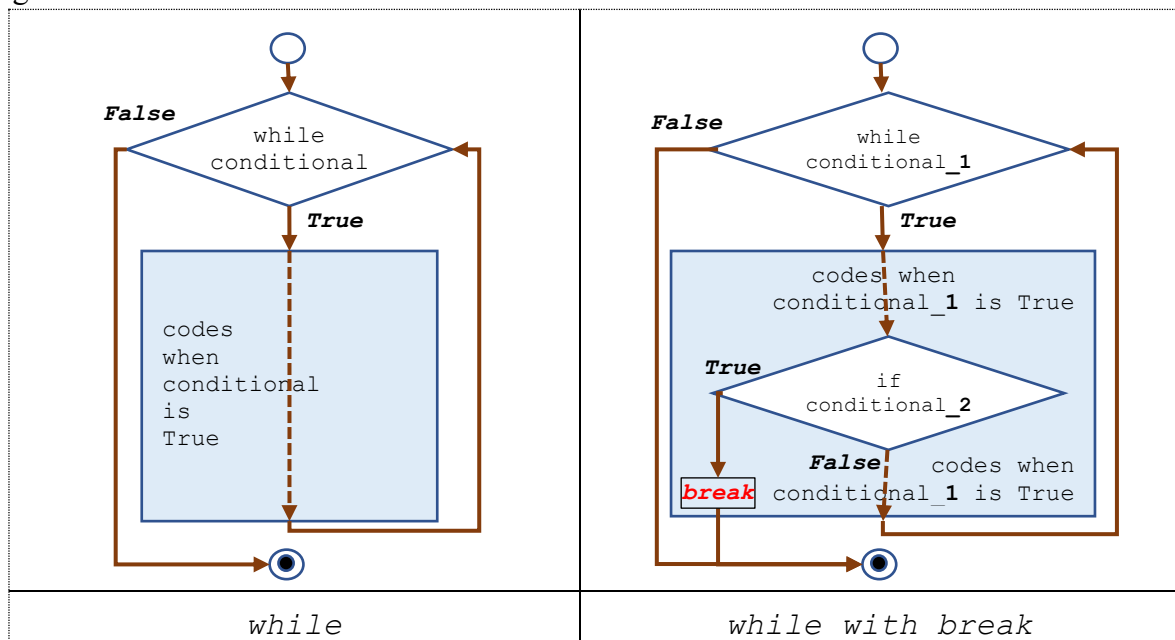
2.11.3. Sử dụng *else* trong cấu trúc lặp

- *for*: khi sử dụng *else*, khối lệnh trong *else* sẽ được thực hiện sau khi *for* đã duyệt xong danh sách.
- *while*: khi sử dụng *else*, khối lệnh trong *else* sẽ được thực hiện khi điều kiện trở thành *False*.

| Ví dụ 2.31 | Mã lệnh | Kết quả |
|------------|---|----------------------------------|
| | <pre>for item in range(1,5): print(item,end="") else: print("\nFinish")</pre> | 1234 Finish |
| | <pre>count = 0 while count < 5: print (count, end="") count = count + 1 else: print (count, "is not less than 5")</pre> | 01234
5 is not less than 5 |
| | <pre>count = 6 while count < 5: print (count, " is less than 5") count = count + 1 else: print ("while statement not executing")</pre> | while statement not
executing |

2.11.4. Lệnh *break*, *continue*, *pass*

Sử dụng 1 trong các lệnh này trong lệnh *for* hoặc *while* nhằm thay đổi thứ tự thi hành của chương trình.



- *break*: được dùng để kết thúc việc thực hiện của lệnh lặp. Khi lệnh này được thực hiện sẽ kết thúc việc thực hiện các lệnh có trong lệnh lặp nhưng đi sau lệnh này.

Ví dụ 2.32

```
while True:
    usr_command = input("Enter your command: ")
    if usr_command == "quit":
        break
    else:
        print("You typed " + usr_command)
```

Trong ví dụ trên, lệnh **break** được dùng để kết thúc các lệnh lặp vô hạn (*infinite while loop*) có điều kiện lặp là **True**. Khi người dùng gõ lệnh không phải là “quit”, chương trình sẽ in ra lệnh mà người dùng vừa gõ vào, ngược lại khi người dùng gõ lệnh “quit” chương trình sẽ kết thúc mà không in lại lệnh “quit”.

Ví dụ 2.33

| Mã lệnh | Kết quả |
|---|---------|
| <pre>for c in "saigon": if c == "i": break print(c)</pre> | s
a |

- **continue**: bỏ qua lần lặp hiện hành và quay về đầu vòng lặp kiểm tra lại điều kiện, nếu thỏa thì tiếp tục lặp, nếu không thỏa thì thoát.

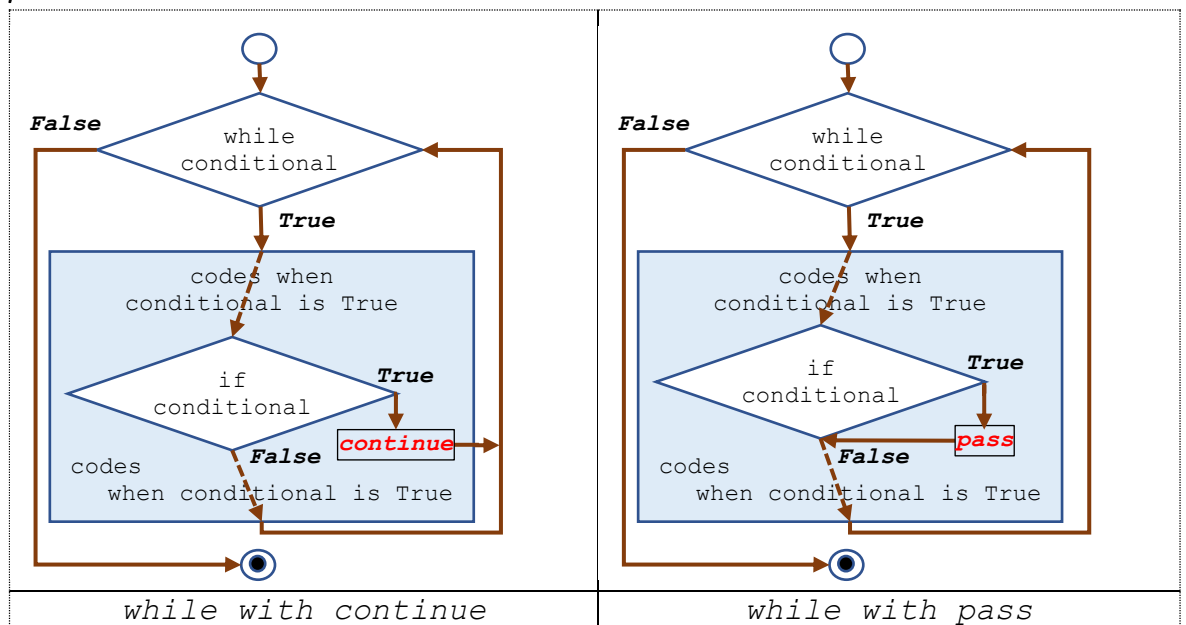
Ví dụ 2.34

```
while True:
    usr_command = input("Enter your command: ")
    if usr_command != "quit":
        print("You typed " + usr_command)
        continue
    else:
        break
```

Ví dụ 2.35

| Mã lệnh | Kết quả |
|--|-----------------------|
| <pre>for c in "saigon": if c == "i": continue print(c)</pre> | s
a
g
o
n |

- **pass**:



- Được sử dụng khi một câu lệnh được yêu cầu nhưng ta không muốn bất cứ lệnh hoặc code nào thực thi.
 - Được sử dụng khi cần thực hiện 1 lệnh là toán tử null (không làm gì cả).
 - Sử dụng hữu ích khi những đoạn code sẽ đi qua nhưng chưa được viết.
- Ví dụ 2.36: Chương trình sẽ cho nhập lệnh nhiều lần. Nếu lệnh nhập vào là “quit” thì kết thúc, ngược lại (else) sẽ không làm gì cả và tiếp tục yêu cầu nhập lệnh khác.

```
while True:
    usr_command = input("Enter your command: ")
    if usr_command == "quit":
        break
    else:
        pass
```

- Ví dụ 2.37:

| | |
|---|---|
| <pre>for number in range(5): if number == 3: pass #pass giúp bỏ qua đoạn code chưa được viết print('pass when number=3') print(str(number))</pre> | <pre>0 1 2 pass when number=3 3 4</pre> |
|---|---|

2.12. Một số hàm tích hợp sẵn trong Python (builtin_function)

(i).- *format()*²

- Trả về chuỗi đã được định dạng hiển thị (representation) của giá trị
- Ví dụ 2.38: chuyển đổi cơ số từ thập phân sang các cơ số khác. Có thể sử dụng hàm `bin()`, `oct()`, `hex()` để thực hiện với kết quả tương tự (khi đổi từ cơ số khác về cơ số thập phân thì phải dùng hàm `int()`)

| Mã lệnh | Kết quả |
|---|---|
| <pre>x = 30 print(x, 'in decimal base =', format(x, '7b'), 'in binary base') print(x, 'in decimal base =', format(x, '7o'), 'in octal base') print(x, 'in decimal base =', format(x, '7x'), 'in hexa decimal base')</pre> | <pre>30 in decimal base = 11110 in binary base 30 in decimal base = 36 in octal base 30 in decimal base = 1e in hexa decimal base</pre> |

(ii).- *float()*

- Trả về số thực từ tham số là chuỗi số.
- Ví dụ 2.39

| Mã lệnh | Kết quả |
|---|----------|
| <pre>n = 123.4567 print(float(n))</pre> | 123.4567 |

(iii).- *int()*

- Trả về số nguyên từ tham số của hàm. Với tham số có thể là số thực hoặc chuỗi số nguyên.
- Lưu ý:

² Xem thêm ví dụ tại mục 2.8.1.2. Các cách xuất bằng hàm print

- Khi chuyển từ số thực sang số nguyên sẽ bị mất đi phần số lẻ nếu có của số thực (giảm giá trị).
- Vì vậy, để chuyển từ chuỗi số thực sang số nguyên: thực hiện chuyển chuỗi sang số thực trước khi chuyển sang số nguyên.

- Ví dụ 2.40

| Mã lệnh | Kết quả |
|--|--|
| <code>n = 123.4567</code>
<code>print(int(n))</code> | 123 |
| <code>n = '123.4567'</code>
<code>print(int(n))</code> | ValueError: invalid literal for int() with base 10: '123.4567' |
| <code>n = '123.4567'</code>
<code>print('n=', int(float(n)))</code> | 123 |

- Ví dụ 2.41: chuyển đổi cơ số từ cơ số khác sang cơ số thập phân (khi đổi từ cơ số thập phân sang cơ số khác thì phải dùng hàm `format()`)

| Mã lệnh | Kết quả |
|---|---|
| <code>s = "10011"</code>
<code>print(s, 'in binary base =', int(s, 2), 'in decimal base')</code> | 10011 in binary base = 19 in decimal base |
| <code>s = "17"</code>
<code>print(s, 'in octal base =', int(s, 8), 'in decimal base')</code> | 17 in octal base = 15 in decimal base |
| <code>s = "6a"</code>
<code>print(s, 'in hexa decimal base =', int(s, 16), 'in decimal base')</code> | 6a in hexa decimal base = 106 in decimal base |

(iv).- ***isinstance()***

- Kiểm tra xem đối tượng có là 1 thể hiện (instance) của Class không?
- Ví dụ 2.42

| Mã lệnh | Kết quả |
|--|---------|
| <code>print(isinstance(25,int) or isinstance(25,str))</code> | True |
| <code>print(isinstance([25],int) or isinstance([25],str))</code> | False |
| <code>print(isinstance("25",int) or isinstance("25",str))</code> | True |

(v).- ***map()***

- Thường được dùng trong việc nhập dữ liệu khi muốn nhập nhiều giá trị trong cùng 1 lệnh input.
- Ví dụ 2.43: trong đoạn lệnh sau, giả sử người dùng nhập 3 5 \Rightarrow chương trình sẽ thực hiện gán a=3 và b=5. Do đó kết quả sẽ in ra: 3 + 5 = 8

```
a, b = map(int, input("Nhập 2 số nguyên cách nhau bởi khoảng trắng").split())
print(a, '+', b, '=', a+b)
```

(vi).- ***max & min(danh sách các giá trị)***

- Trả về phần tử có giá trị lớn(nhỏ) nhất có trong danh sách các tham số truyền cho hàm.
- Cú pháp: `max(value1, value2, ...)`
`min(value1, value2, ...)`

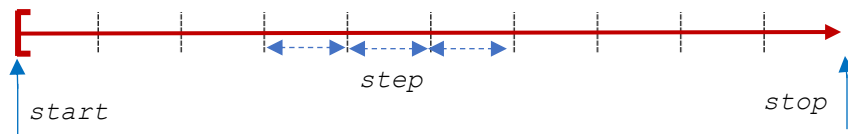
- Ví dụ 2.44

| Mã lệnh | Kết quả |
|---|---|
| <code>print(max(7, 4, 9, 12))</code> | 12 |
| <code>print(max("c", "A", "M"))</code> | c |
| <code>print(max(7, 4.12, 9, 9.26))</code> | 9.26 |
| <code>print(min(7, 4, 9, 12))</code> | 4 |
| <code>print(min("c", "A", "M"))</code> | A |
| <code>print(min(7, 4.12, 9, 9.26))</code> | 4.12 |
| <code>print(min(7, 4, "B"))</code> | <code>TypeError: '>' not supported between instances of 'str' and 'float'</code> |

(vii).- **range([start,]stop[,step])**

- Trả về dãy số nguyên từ số bắt đầu (*start*) đến trước số kết thúc (*stop*).

- Cú pháp: **range([start,] stop [, step])**



Trong đó:

- **start** (tùy chọn) Giá trị bắt đầu. Mặc định là 0.
- **stop** (bắt buộc) Giá trị kết thúc. Giá trị này sẽ không có trong danh sách các số trong kết quả.
- **step** (tùy chọn) Bước tăng (+) hoặc giảm (-) của các số trong kết quả. Mặc định là 1.

- Ví dụ 2.45

| Mã lệnh | Kết quả |
|--|--|
| <code>for k in range(1, 6, 1):</code>
<code>print(k, '\t', end='')</code> | 1 2 3 4 5 #không bao gồm số 6 |
| <code>for k in range(20, 5, -3):</code>
<code>print(k, '\t', end='')</code> | 20 17 14 11 8 #không bao gồm số 5 |

(viii).- **round(number[,n])**

- Làm tròn số.

- Cú pháp **round(number[,n])**

- Ví dụ 2.46

| Mã lệnh | Kết quả |
|---|-----------|
| <code>num=12645.6285</code> | 12645.629 |
| <code>print(round(num, 3))</code> | |
| <code>print(round(num, 2))</code> | 12645.63 |
| <code>print(round(num, 1))</code> | 12645.6 |
| <code>print(round(num, 0))</code> | 12646.0 |
| <code>print(round(num, -1))</code> | 12650.0 |
| <code>print(round(num, -2))</code> | 12600.0 |
| <code>print(int(round(num, -3)))</code> | 13000 |

(ix).- **str()**

- Nhận tham số là kiểu số (nguyên hoặc thực, trả về kiểu dữ liệu chuỗi của tham số.

- Ví dụ 2.47

| Mã lệnh | Kết quả |
|---|---|
| <code>X=27</code>
<code>Y=1234.567</code>
<code>a=str(X)</code> | <code>a= 27 . Type of a is <class 'str'></code> |

| | |
|---|--|
| b=str(Y) | |
| print('a=', a, '. Type of a is', type(a)) | |
| print('b=', b, '. Type of b is', type(b)) | b= 1234.567 . Type of b is <class 'str'> |

(x).- **type()**

- Trả về kiểu của đối tượng đang là tham số của hàm.
- Ví dụ 2.48: sử dụng hàm **type()** để xác định kiểu dữ liệu của biến

| Mã lệnh | Kết quả |
|---|------------------------|
| <pre>#var = {"x":100, "y":200, "z":300} #var = [1,3,5] #var= (5,7,8) var = 3.14 if type(var) is list: print('type of var is a list') elif type(var) is set: print('type of var is a set') elif type(var) is tuple: print('type of var is a tuple') elif type(var) is float: print('type of var is a float') else: print('Neither a list or a set or a tuple or a float.')</pre> | type of var is a float |

(xi).- **Một số hàm khác**

| Hàm | Mô tả |
|----------------------|--|
| abs() | Trả về giá trị tuyệt đối của một số |
| ascii() | Trả về một phiên bản có thể đọc được của bất kỳ đối tượng nào (String, Tuples, List, ...) |
| bin() | Chuyển đổi số nguyên sang chuỗi nhị phân |
| bool() | Chuyển một giá trị sang Boolean |
| bytearray() | Trả về mảng kích thước byte được cấp |
| bytes() | Trả về đối tượng byte không đổi |
| callable() | Kiểm tra xem đối tượng có thể gọi hay không |
| chr() | Trả về một ký tự (một chuỗi) từ Integer |
| classmethod() | Trả về một class method cho hàm |
| compile() | Trả về đối tượng code Python |
| complex() | Tạo một số phức |
| delattr() | Xóa thuộc tính khỏi đối tượng |
| dir() | Trả lại thuộc tính của đối tượng |
| divmod() | Trả về một Tuple của Quotient và Remainder |
| enumerate() | Thêm vào một bộ đếm vào trước mỗi iterable và trả về kết quả dưới dạng đối tượng liệt kê (enumerate object). |
| eval(String) | Đánh giá kiểu dữ liệu của 1 chuỗi và trả về 1 object của kiểu được đánh giá |
| exec() | Thực thi chương trình được tạo động |
| frozenset() | Trả về đối tượng frozenset không thay đổi |
| getattr() | Trả về giá trị thuộc tính được đặt tên của đối tượng |
| globals() | Trả về dictionary của bảng symbol toàn cục hiện tại |
| hasattr() | Trả về đối tượng dù có thuộc tính được đặt tên hay không? |

| | |
|-----------------------------|---|
| <code>hash()</code> | Trả về giá trị hash của đối tượng |
| <code>help()</code> | Gọi Help System được tích hợp sẵn |
| <code>hex()</code> | Chuyển Integer thành Hexadecimal |
| <code>id()</code> | Trả về định danh của đối tượng |
| <code>input()</code> | Đọc và trả về chuỗi trong một dòng |
| <code>issubclass()</code> | Kiểm tra xem đối tượng có là Subclass của Class không |
| <code>iter()</code> | Trả về iterator cho đối tượng |
| <code>locals()</code> | Trả về dictionary của bảng symbol cục bộ hiện tại |
| <code>memoryview()</code> | Trả về chế độ xem bộ nhớ của đối số |
| <code>next()</code> | Trích xuất phần tử tiếp theo từ Iterator |
| <code>object()</code> | Tạo một đối tượng không có tính năng (Featureless Object) |
| <code>oct()</code> | Chuyển số nguyên sang bát phân |
| <code>open()</code> | Trả về đối tượng File |
| <code>ord()</code> | Trả về mã Unicode code cho ký tự Unicode |
| <code>property()</code> | Trả về thuộc tính property |
| <code>repr()</code> | Trả về representation có thể in của đối tượng |
| <code>reversed()</code> | Trả về iterator đảo ngược của một dãy |
| <code>setattr()</code> | Đặt giá trị cho một thuộc tính của đối tượng |
| <code>slice()</code> | Cắt đối tượng được chỉ định bằng range() |
| <code>staticmethod()</code> | Tạo static method từ một hàm |
| <code>super()</code> | Cho phép tham chiếu đến Parent Class bằng super |
| <code>vars()</code> | Trả về thuộc tính <code>__dict__</code> của class |
| <code>__import__()</code> | Hàm nâng cao, được gọi bằng lệnh import |

- Để biết thêm về công dụng của hàm, các đối số, có thể sử dụng lệnh sau:

`print(ten_ham.__doc__)`

- Ví dụ 2.49: sử dụng 2 hàm `type()` và `isinstance()` để kiểm tra kiểu của dữ liệu

Mã lệnh

Kết quả

| | |
|---|--------------------------------|
| <code>a = 5</code> | |
| <code>print(a, "is of type", type(a))</code> | 5 is of type <class 'int'> |
| <code>b = 2.0</code> | |
| <code>print(a, "is of type", type(b))</code> | 2.0 is of type <class 'float'> |
| <code>c = 1+2j</code> | |
| <code>print(c, "is complex number?",
isinstance(1+2j,complex))</code> | (1+2j) is complex number? True |

2.13. Thao tác với đối tượng kiểu String

2.13.1. Truy xuất chuỗi con bằng cách sử dụng index và toán tử cắt lát (slicing - [:])

- **[index]**: cho phép truy xuất từng ký tự trong chuỗi thông qua chỉ số *index* với *index* được tính từ 0 đến chiều dài chuỗi -1.
- **[from:to]**: cho phép tạo các chuỗi con thông qua toán tử lấy khoảng (`range[from:to]`).
 - Mặc định **from** là từ vị trí đầu chuỗi (0)
 - Mặc định **to** là đến vị trí cuối chuỗi (`len(chuỗi) - 1`). Khi **to** là số âm (-t), sẽ được hiểu là lấy đến cuối chuỗi sau khi đã cắt bên phải chuỗi -t ký tự.

Ví dụ 2.50: với $S = \text{"Hello Python"}$

| | | | | | | | | | | | |
|-----|-----|-----|----|----|----|----|----|----|----|----|----|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| H | e | l | l | o | | P | y | t | h | o | n |
| -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

| Mã lệnh | Kết quả |
|-------------------------------|---------------------|
| <code>s="Hello Python"</code> | |
| <code>print (s)</code> | Hello Python |
| <code>print(s[6])</code> | P |
| <code>print (s[0:6])</code> | Hello |
| <code>print(s[4:7])</code> | oP |
| <code>print(s[3:])</code> | lo Python |
| <code>print (s[:6])</code> | Hello |
| <code>print (s[0:])</code> | Hello Python |
| <code>print (s[-2:6])</code> | #không xuất kết quả |
| <code>print (s[0:-5])</code> | Hello |

(ký hiệu `␣` dùng trong kết quả của ví dụ để minh họa cho khoảng trắng)

- **`[::-1]`**: dùng khi cần đảo chuỗi. Sử dụng kết hợp hàm `str` với `index`, với giá trị thứ 3 trong cặp ngoặc vuông là `-1`, đồng thời để trống về **from** và **to**

Ví dụ 2.51

| Mã lệnh | Kết quả |
|---|---------|
| <code>s='Sai Gon'</code> | |
| <code>sReverse= str(s) [::-1]</code> | |
| <code># hoặc dùng gọn hơn: sReverse= s[::-1]</code> | |
| <code>print(sReverse)</code> | noG iaS |

2.13.2. Duyệt chuỗi

- Python không hỗ trợ kiểu ký tự, vì vậy các ký tự cũng được xem như là 1 chuỗi với chiều dài là 1.
- String được lưu giữ dưới dạng các ký tự đơn trong các ô nhớ liên tiếp nhau. Nhờ vậy có thể truy cập từ cả hai hướng từ trái sang phải (*forward*) hoặc từ phải sang trái (*backward*).
- Ví dụ 2.52: duyệt chuỗi từ trái sang phải

| Sử dụng lệnh <code>for</code> | Sử dụng lệnh <code>while</code> | Kết quả |
|--|--|------------------|
| #Dùng index
<code>S="Sai gon"</code>
<code>print ('forward: ',end='')</code>
<code>for i in range(0,len(S)):</code>
<code> print (S[i], end='')</code> | #Dùng index
<code>S = "Sai Gon"</code>
<code>print ('forward:',end='')</code>
<code>i =0</code>
<code>while (i<len(S)):</code>
<code> print (S[i], end='')</code>
<code> i+=1</code> | forward: Sai Gon |
| #Hoặc không dùng index
<code>S="Sai gon"</code>
<code>print ('forward: ',end='')</code>
<code>for ch in S:</code>
<code> print (ch,end='')</code> | | forward: Sai Gon |

- Ví dụ 2.53: duyệt chuỗi từ phải sang trái

| Sử dụng lệnh <i>for</i> | Sử dụng lệnh <i>while</i> | Kết quả |
|--|---|-------------------|
| <pre>S="Sai gon" print('backward:',end='') k=len(S)-1 for i in range(k,-1,-1): print (S[i],end='')</pre> | <pre>print('backward:',end='') k =len(S)-1 while (k>=0): print (S[k], end='') k-=1</pre> | backward: noG iaS |

2.13.3. So sánh chuỗi

- Sử dụng 2 toán tử so sánh bằng (**==**), và so sánh khác (**!=**) để so sánh 2 chuỗi. Kết quả trả về là một giá trị boolean.

Ví dụ 2.54

| Mã lệnh | Kết quả |
|--|---------|
| <code>print("24" == "24")</code> | True |
| <code>print("15" == "8")</code> | False |
| <code>print("aa" == "ab")</code> | False |
| <code>print("Sai Gon" != "sai gon")</code> | True |

2.13.4. Các phương thức dùng với string

2.13.4.1.len()

- Công dụng: lấy chiều dài chuỗi
- Cú pháp: **len (hằng chuỗi hoặc biến kiểu chuỗi)**

Ví dụ 2.55

| Mã lệnh | Kết quả |
|---|---------|
| <pre>greeting = "Hello Python" print(len(greeting))</pre> | 12 |

2.13.4.2.center()

- Công dụng: Tạo bản sao của chuỗi bằng cách điền vào đầu và cuối chuỗi bằng ký tự *fillchar* để có chiều dài chuỗi = *width*.
- Cú pháp: **center(width[,fillchar])**

Ví dụ 2.56

| Mã lệnh | Kết quả |
|---|----------------------|
| <pre>s="hello python" print(s.center(20,"*"))</pre> | ****hello python**** |

2.13.4.3.count()

- Công dụng: Đếm số lần xuất hiện chuỗi con tính từ vị trí *start* đến vị trí *end*. Mặc định *start=0* và *end=len()-1*
- Cú pháp: **count(strSub[,start,end])**

Ví dụ 2.57

| Mã lệnh | Kết quả |
|--|---------|
| <pre>s="hello python" print(s.count("o",0,len(s)))</pre> | 2 |
| <pre>print(s.count("ll",0,len(s)))</pre> | 1 |

2.13.4.4.Đổi từ chữ hoa sang chữ thường hoặc ngược lại

Các phương thức trong nhóm này sẽ tạo ra bản sao của chuỗi gốc (không thay đổi chuỗi gốc).

- **capitalize()** : với ký tự đầu viết hoa.
- **lower()** : với tất cả các ký tự trong chuỗi gốc thành chữ thường.
- **upper()** : với tất cả các ký tự trong chuỗi gốc thành chữ hoa.

- **title()** : với tất cả các ký tự đầu tiên của mỗi từ trong chuỗi gốc thành chữ hoa.
- **swapcase()** : chuyển các ký tự đang in hoa sang thường và ký tự thường sang in hoa.

Ví dụ 2.58

| Mã lệnh | Kết quả |
|--|---------------------|
| <code>s=" SàI gÒN "</code> | |
| <code>print("Capitalize:",
s.capitalize())</code> | Capitalize: Sài gòn |
| <code>print("UPPER:", S.upper())</code> | UPPER: SÀI GÒN |
| <code>print("lower:", S.lower())</code> | lower: sài gòn |
| <code>print("Title:", S.title())</code> | title: Sài Gòn |
| <code>print("Swap case:", S.swapcase())</code> | Swap case: sÀI GÒn |

2.13.4.5. Kiểm tra chuỗi

Kết quả trả về của các phương thức trong nhóm này là *True* nếu đúng hoặc *False* nếu sai.

- **isnumeric()**: kiểm tra chuỗi chỉ chứa toàn ký số (0-9). Phương thức này được dùng trên các đối tượng unicode. Để xác định một chuỗi là unicode, chỉ cần thêm tiền tố 'u' vào trước chuỗi, ví dụ S = u'Sai Gon'.
- **isdecimal()**: kiểm tra chuỗi chỉ chứa toàn ký số (tương tự **isnumeric**, chỉ dùng trên các đối tượng unicode).
- **isdigit()**: kiểm tra chuỗi chỉ chứa toàn ký số (tương tự **isnumeric**).
- **isalnum()**: kiểm tra chuỗi chỉ chứa toàn ký tự trong bảng chữ cái hoặc ký số.
- **isalpha()**: kiểm tra chuỗi chỉ chứa toàn ký tự trong bảng chữ cái.
- **islower()**: kiểm tra chuỗi chỉ chứa toàn ký tự thường trong bảng chữ cái (a-z).
- **isupper()**: kiểm tra chuỗi chỉ chứa toàn ký tự in hoa trong bảng chữ cái (A-Z).
- **istitle()**: kiểm tra tất cả các từ trong chuỗi đều bắt đầu bằng chữ in hoa (các ký tự còn lại trong từ là ký số hay ký tự ngoài bảng chữ cái cũng được miễn sao không phải ký tự in hoa).
- **isspace()**: kiểm tra chuỗi chỉ chứa toàn ký tự khoảng trắng.
- **startswith(search_Char, start, end)**: kiểm tra xem chuỗi có bắt đầu bằng *search_Char* hay không? Kết quả trả về *True* nếu đúng hoặc *False* nếu sai.
- **endswith(search_Char, start, end)**: kiểm tra xem chuỗi hoặc khoảng chuỗi có được kết thúc bằng ký tự nào đó hay không? Kết quả trả về *True* nếu đúng hoặc *False* nếu sai.

Trong đó: • *start* là index bắt đầu chuỗi cần so sánh. Mặc định, *start* = 0.

• *end* là index kết thúc chuỗi cần so sánh. Mặc định, *end* = *len()*.

Ví dụ 2.59

| Mã lệnh | Kết quả |
|--|---------|
| <code>S1='123A'</code> | |
| <code>S2='1234'</code> | |
| <code>S3='AbcD'</code> | |
| <code>print(S1.isnumeric())</code> | False |
| <code>print(str.isnumeric(S2))</code> | True |
| <code>print(S1.isdigit())</code> | False |
| <code>print(str.isdigit(S2))</code> | True |
| <code>print(S1.isalpha())</code> | False |
| <code>print(str.isalpha(S3))</code> | True |

2.13.4.6. Thay thế/loại bỏ ký tự được chỉ định

Các phương thức trong nhóm này sẽ tạo ra bản sao của chuỗi gốc (không thay đổi chuỗi gốc).

- **`replace(sOld, sNew[, max])`**: Trả về chuỗi kết quả với `sOld` sẽ được thay thế bằng `sNew`. `max` nếu có dùng sẽ là số lần thay thế tối đa, mặc định của `max` là thay tất cả các lần tìm thấy `sOld` trong chuỗi gốc.
- **`expandtabs(num)`**: tìm kiếm thay thế `\t` bằng các ký tự khoảng trắng. Trong đó `num` là số lượng khoảng trắng sẽ thay thế cho mỗi `\t`.
- **`lstrip([chars])`**: Tương tự `strip` nhưng chỉ loại bỏ `chars` phía trước (đầu) chuỗi.
- **`rstrip([chars])`**: Tương tự `strip` nhưng chỉ loại bỏ `chars` phía sau (cuối) chuỗi. Ký tự mặc định bị loại bỏ là `'\n'`.
- **`strip([chars])`**: Tạo chuỗi kết quả sau khi đã loại bỏ `chars` ở đầu và cuối. Mặc định của `chars` là khoảng trắng. Khi sử dụng hàm này tương đương với việc sử dụng kết hợp 2 phương thức `lstrip` và `rstrip`.

Ví dụ 2.60

| Mã lệnh | Kết quả |
|--|--------------------------------------|
| <code>s="hello python"</code>
<code>print(s.replace("l", 'M'))</code> | heMMo python |
| <code>s="hello python"</code>
<code>print(s.replace("l", 'M', 1))</code> | heMlo python |
| <code>s="Hello world"</code>
<code>news=s.replace("world", "Sai gon")</code>
<code>print (news)</code> | Hello Sai gon |
| <code>s="###hello python#"</code>
<code>print(s.strip())</code> | ###hello python# |
| <code>print(s.strip('#'))</code> | hello python |
| <code>str1='Python Exercises\n'</code>
<code>print(str1)</code>
<code>print(str1.rstrip())</code> | Python Exercises
Python Exercises |
| <code>str1="###hello python#"</code>
<code>print(str1)</code>
<code>print(str1.rstrip('#'))</code> | ###hello python#
###hello python |

2.13.4.7. Tách chuỗi theo ký tự được chỉ định, kết quả đưa vào list

- **`split(s[, num])`**: Tạo ra list các phần tử được cắt theo `s`. Mặc định `s` là khoảng trắng. Nếu có `num` thì quy định số phần tử được tạo ra trong list là `num+1`.
- **`rsplit(s[, num])`**: Kết quả tương tự như phương thức `split`, nhưng khác là thực hiện từ phải sang trái
- **`splitlines(keeplinebreaks)`**: Tạo ra list các phần tử được cắt theo ký tự ngắt dòng `'\n'`. Nếu `keeplinebreaks` là `True`, dấu ngắt dòng được giữ lại cùng với chuỗi, ngược lại sẽ bỏ đi dấu ngắt dòng. Mặc định của `keeplinebreaks` là `False`.
- Ví dụ 2.61

| Mã lệnh | Kết quả |
|--|--|
| <code>S='https://www.w3resource.com/python-exercises/string'</code>
<code>lst1 = S.split('/', 2)</code>
<code>print(lst1)</code> | <code>['https:', '', 'www.w3resource.com/python-exercises/string']</code> |
| <code>lst1=S.split('/', 3)</code>
<code>print(lst1)</code> | <code>['https:', '', 'www.w3resource.com', 'python-exercises/string']</code> |

2.13.4.9. Tìm kiếm

- Gồm các phương thức:
 - **find**: tìm từ trái qua phải
 - **rfind**: tìm từ phải (cuối chuỗi) qua trái (đầu chuỗi).
 - **index**: tương tự như phương thức *find()* chỉ khác duy nhất là nếu như không tìm thấy thì hàm này sẽ gọi *exception*.
 - **rindex**: tương tự như phương thức *rfind()* chỉ khác duy nhất là nếu như không tìm thấy thì hàm này sẽ gọi *exception*.
- Cú pháp chung:


```
find/rfind/index/rindex (subStr, Start, End)
```
- Công dụng: Tìm chuỗi con tính từ vị trí *start* đến vị trí *end*. Khi tìm thấy, trả về vị trí tìm thấy trong chuỗi gốc (vị trí này không phụ thuộc tham số *start*); khi không tìm thấy trả về -1.
 - Giá trị mặc định của *Start* là 0
 - Giá trị mặc định của *End* là cuối chuỗi.

Ví dụ 2.63

| Mã lệnh | Kết quả |
|--|---------|
| <code>s="hello python"</code> | |
| <code>print(s.find("o",0,len(s)))</code> | 4 |
| <code>print(s.find("o",4,len(s)))</code> | 4 |
| <code>print(s.find("o",5,len(s)))</code> | 10 |
| <code>s="Hello world"</code> | |
| <code>print(s.find("l"))</code> | 2 |
| <code>print(s.find("l",5))</code> | 9 |
| <code>print(s.find("l",5,8))</code> | -1 |
| <code>print(s.rfind("l"))</code> | 9 |
| <code>print(s.rfind("l",5))</code> | 9 |
| <code>print(s.rfind("l",5,8))</code> | -1 |

2.13.4.10. maketrans()

- Phương thức *maketrans()* trả về một bảng tịnh tiến được sử dụng trong hàm *translate*.
- Thường được dùng phối hợp với hàm *translate()*.
- Cú pháp

str.maketrans(intab, outtab)

Trong đó:

- **intab**: là chuỗi có các ký tự thực sự.
- **outtab**: là chuỗi có các ký tự ánh xạ tương ứng.
- Cả hai tham số *intab* và *outtab* phải có cùng độ dài.

Ví dụ 2.64

| Mã lệnh | Kết quả |
|---|-----------|
| <code>S = "32.054,23"</code> | |
| <code>print(S)</code> | 32.054,23 |
| <code>S = S.translate(S.maketrans(','.',' ','.''))</code> | |
| <code>print(S)</code> | 32,054.23 |

2.13.4.11. startswith()

- Phương thức **startswith(strFind)** kiểm tra xem đối tượng chuỗi đang xét có được bắt đầu bằng chuỗi *sFind* hay không? Phương thức này sẽ trả về kết quả là **True** hoặc **False**.
- Ví dụ 2.65: kiểm tra xem chuỗi có bắt đầu bằng 'SA' hay không?

| Mã lệnh | Kết quả |
|--|---------|
| <code>S = "Sai Gon"</code> | |
| <code>print(S.startswith("SA"))</code> | False |
| <code>print(S.upper().startswith("SA"))</code> | True |

2.13.4.12. *translate()*

- Phương thức *translate()* trả về một bản sao của chuỗi ban đầu trong đó tất cả ký tự đã được thông dịch bởi sử dụng table (được xây dựng với hàm *maketrans()* trong *string module*), xóa tất cả ký tự một cách tùy ý trong chuỗi *deletechars*.
- Thường được dùng phối hợp với hàm *maketrans()*.
- Cú pháp

`str.translate(table[, deletechars])`

Trong đó:

- **table**: Có thể sử dụng hàm *maketrans()* để tạo một bảng thông dịch.
- **Deletechars**: danh sách các ký tự để được xóa từ chuỗi ban đầu.

2.14. Debug

2.14.1. Giới thiệu

2.14.1.1. Bug

- Trong lập trình, **bugs** ám chỉ các lỗi xảy ra trong logic, hay bất kỳ vấn đề gì làm cho ứng dụng không thực thi được hoặc thực thi sai.
- **Bugs** luôn tiềm ẩn ở mọi nơi, và người lập trình không thể lường trước được mọi tình huống có thể xảy ra mà chỉ có thể cố gắng làm giảm nó đến mức thấp nhất có thể tùy vào khả năng của người lập trình tại thời điểm phát triển và bảo trì ứng dụng.

2.14.1.2. Debug

- **Debug** là quá trình tìm kiếm ra lỗi hay nguyên nhân gây ra lỗi (**bug** ở đâu?) để có hướng sửa lỗi (**fix bug**).
- Người lập trình chính là người sinh ra **bugs** nhiều nhất. Vì vậy, cũng chính người lập trình phải là người phát hiện và gỡ bỏ bớt các **bugs**.
- Mục đích của **debug** không chỉ là để loại bỏ lỗi (**error**) khỏi chương trình mà quan trọng hơn còn để giúp lập trình viên hiểu rõ hơn sự thực thi của chương trình. Một lập trình viên không có khả năng **debug** hiệu quả thì không thể lập trình tốt được.

2.14.2. Các phương pháp debug

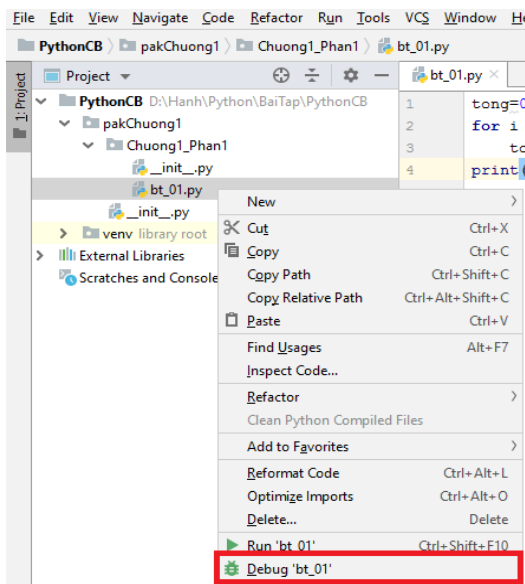
- (i)- **Debugging Tool** (công cụ để Debug - Debugger): là phương pháp Debug đi sâu vào *source code* nhất.
- **Debugger phần mềm**: như *Microsoft Visual Studio Debugger*, *GNU Debugger*, ...
 - **Debugger phần cứng**: được thiết kế kèm cho các hệ thống nhúng (*Embedded System*) bởi các thiết kế nhúng không phải là những thiết kế mang tính mục đích chung (*General-purpose*) mà thường được thiết kế trên các *platform* riêng biệt phục vụ các ứng dụng riêng biệt nên cũng cần những **Debugger** đặc thù đi kèm.

- (ii)- *Printlining*: người lập trình tự thêm vào *source code* những dòng lệnh để in ra những thông tin cần theo dõi trong quá trình thực thi. Sau khi hoàn tất việc debug, các dòng lệnh này sẽ được che lại hoặc xóa đi.
- (iii)- *Logging*: tạo ra một biểu mẫu để ghi (log) lại những thông tin sau khi chương trình thực thi. Phân tích nguyên nhân lỗi dựa trên những thông tin này.

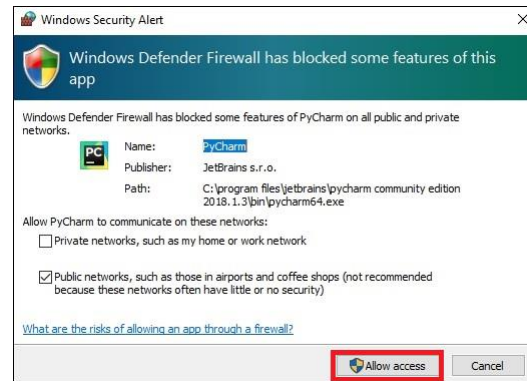
2.14.3. Debug Tool trong Pycharm

- Thực thi mã lệnh *python* bao gồm hai chế độ: chạy tập lệnh (*script*) và gỡ lỗi (*debugging*) tập lệnh.
- Các bước *debug* trên môi trường *Pycharm*: (giả sử file *bt_01.py* là file cần debug)

Bước 1. Right click vào tên file *bt_01.py* trong cửa sổ *Project*, chọn *Debug bt_01*

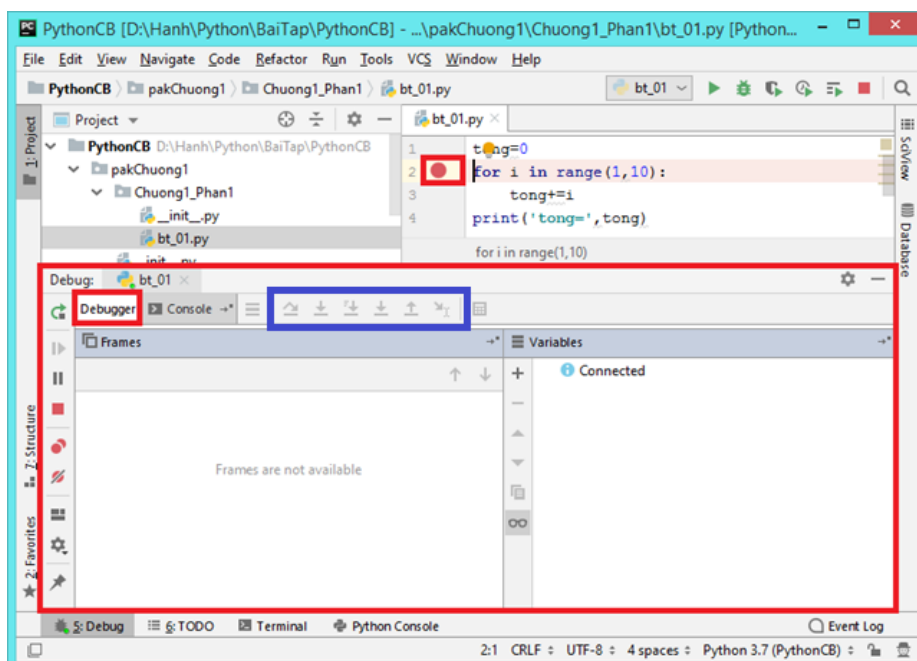


Minh họa Bước 1



Minh họa Bước 2

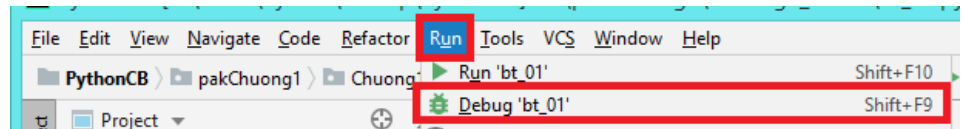
Bước 2. Đến đây, *firewall* của *Windows* có thể yêu cầu xác nhận việc cấp quyền *debugging* cho *project Python* vì debug liên quan đến việc biên dịch theo dòng. Chọn *Allow Access*.



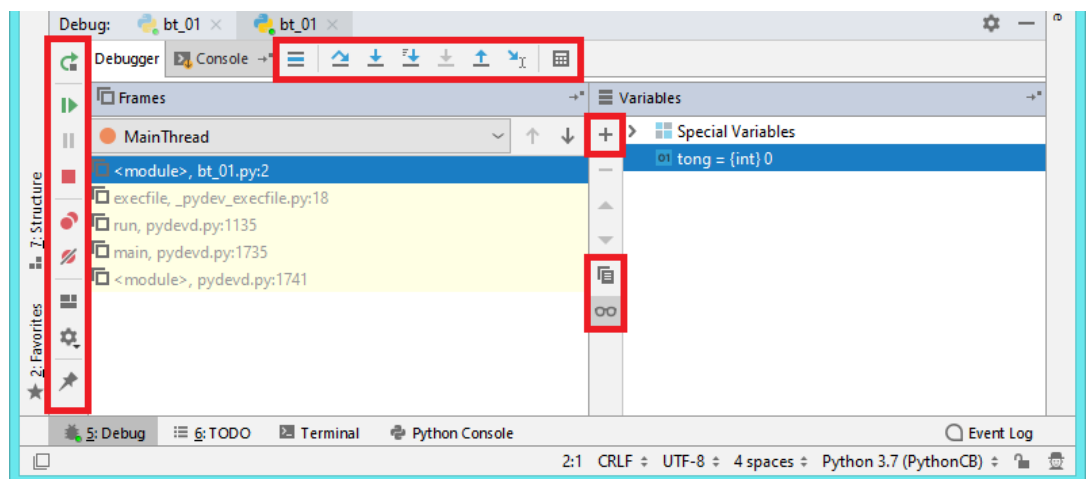
Lúc này cửa sổ *debug* xuất hiện như hình trên. Chọn *tab Debugger*.

Bước 3. Chọn điểm bắt đầu thực hiện *debug*: Trong cửa sổ của file *bt_01*, click vào vùng màu xám trước dòng lệnh mà ta muốn bắt đầu thực hiện *debug*.

Bước 4. Chọn menu *Run/Debug 'bt_01'*



- Lúc này thanh công cụ phục vụ cho việc *debug* mới được *enabled*. Tùy vào nhu cầu mà người lập trình sẽ sử dụng các *icon* này.



Chức năng của các icon:

- | | | | | | |
|--|----------------------------------|--|-----------------------|--|------------------------|
| | Show Execution point (Alt+F10) | | Step Over (F8) | | Step Into (F7) |
| | Step Into My Code (Alt+Shift+F7) | | Step Out (Shift + F8) | | Run To Cursor (Alt+F9) |
| | Evaluate Expression (Alt+F8) | | | | |

- Ngoài ra, trong cửa sổ *Variables* còn có thêm các icon (nhập thêm tên biến cần quan sát), (mở thêm cửa sổ *Watches*), (Cho hiện thông tin của biến trong cửa sổ *Variables*). Tùy nhu cầu sử dụng mà người lập trình sẽ click chọn các icon này.

Bước 5. Trong quá trình *debug*, người lập trình có thể sử dụng thêm các *button* trên thanh dọc bên trái của cửa sổ

- | | | | | | |
|--|----------------------------------|--|---------------------|--|----------------------|
| | Resume bt_01 (Ctrl+F5) | | Resume Program (F9) | | Stop bt_01 (Ctrl+F2) |
| | View Breakpoints (Ctrl+Shift+F8) | | Mute Breakpoint | | Restore Layout |
| | Settings | | Pin Tab | | |

USER DEFINE FUNCTION - MODULE - PACKAGE

Tất cả ví dụ cho đến thời điểm này đều được thực thi trong *command line* hoặc từ một file *python.py*. Tuy nhiên, đối với các ứng dụng lớn, có nhiều chức năng thì phân chia nhỏ dự án thành các file khác nhau sẽ giúp dễ bảo trì và tái sử dụng các thành phần đã thiết kế.

Chương này sẽ giúp bạn thiết kế các tính năng theo mô hình các module và khi cần thì sẽ gọi file tương ứng để sử dụng.

3.1. Hàm do người dùng tự tạo (UDF – User-Define Function)

3.1.1. Định nghĩa

- Hàm (*Function*) là tập hợp các dòng lệnh được viết để thực hiện một chức năng nào đó.
- Mặc dù *Python* cung cấp rất nhiều hàm được xây dựng sẵn (*built-in function*), nhưng *Python* vẫn cho phép người dùng có thể tự xây dựng các hàm (*user-define function*) cho riêng mình.
- Hàm giúp mức độ tái sử dụng mã lệnh tốt hơn.

3.1.2. Khai báo và xây dựng hàm

- Cú pháp:

```
def FunctionName(param1, param2,...):
    # mô tả về function nếu cần thiết
    statement(s)
    [return [expression]]
```

- Kết quả trả về dữ liệu (*return*) của function:
 - Hàm có thể trả về nhiều giá trị.
 - Nếu không trả về (không có lệnh *return*) thì mặc định sẽ trả về giá trị *None*.
- Ví dụ 3.1:

| Mã lệnh | Kết quả |
|--|---|
| <pre>#hàm không return def Show(s): print ('Hello',s) #----- name= 'Sai Gon' Show(name)</pre> | Hello Sai Gon |
| <pre>#hàm return 1 giá trị def Cong(a, b): return (a+b) #----- a = 3 b = 2 print("%d + %d = %d" %(a,b,Cong(a,b)))</pre> | 3 + 2 = 5 |
| <pre>#hàm return nhiều giá trị def Swap(a, b): a, b = b, a return a, b #----- a = 3 b = 2 print("Before swaping a = %d and b = %d" %(a, b)) a,b = Swap(a,b) print("After swaping a = %d and b = %d" %(a, b))</pre> | <p>Before swaping
a = 3 and b = 2
After swaping
a = 2 and b = 3</p> |

3.1.3. Vị trí của hàm trong chương trình và lời gọi hàm

- Quy ước: phần khai báo hàm phải nằm trước (bên trên) lời gọi hàm trong chương trình.
- Ví dụ 3.2

| Mã lệnh 1 | Kết quả | Diễn giải |
|--|--|---|
| <pre>def Cong(a,b): return (a+b) x=2 y=3 print (Cong(x,y))</pre> | 5 | Đúng và nên sử dụng theo cách này |
| Mã lệnh 2 | Kết quả | Diễn giải |
| <pre>x=2 y=3 def Cong(a,b): return (a+b) print (Cong(x,y))</pre> | 5 | <ul style="list-style-type: none"> - Không gây lỗi (mặc dù hàm nằm chen giữa nội dung chính của chương trình nhưng vẫn nằm trước lời gọi hàm). - Tuy không gây lỗi nhưng không nên sử dụng theo cách này. |
| Mã lệnh 3 | Kết quả | Diễn giải |
| <pre>x=2 y=3 print (Cong(x,y)) def Cong(a,b): return (a+b)</pre> | <pre>NameError: name 'Cong' is not defined</pre> | Gây lỗi vì lời gọi hàm xuất hiện trước phần khai báo hàm |

3.1.4. Tầm vực của biến (Scope of variables)

3.1.4.1. Phân loại biến

- Biến toàn cục (*global variables*) là một biến được tạo bên ngoài các function, được truy cập trên toàn chương trình kể cả trong các hàm của chương trình.
- Biến cục bộ (*local variables*): là một biến được tạo bên trong một hàm. Theo mặc định biến chỉ được truy cập trong hàm mà biến được khai báo. Khi biến toàn cục và biến cục bộ có trùng tên, biến cục bộ sẽ được ưu tiên sử dụng
- Biến *nonlocal*: là một biến có tầm vực cao hơn local và thấp hơn global. Thường dùng khai báo biến dạng này trong các hàm có chứa hàm con (sub Function). Khi đó, biến được khai báo nonlocal trong sub function, biến này được function ‘nhìn thấy’, nhưng chương trình chính thì không.

3.1.4.2. Từ khóa global trong Python

- Được sử dụng khi cần thay đổi giá trị của biến toàn cục trong thân các hàm.
- Việc sử dụng từ khóa *global* bên ngoài các hàm sẽ không có ý nghĩa.
- Ví dụ 3.3

| Mã lệnh | Kết quả |
|---|---------|
| <pre>#truy cập biến toàn cục bên trong function def add(): print(c) c = 5 # global variable add()</pre> | 5 |

| | |
|--|---|
| <pre>'''thay đổi giá trị biến toàn cục bên trong function và không sử dụng từ khóa global ⇒ gây lỗi''' def add(): c = c + 2 # increment c by 2 print(c) c = 5 # global variable add()</pre> | <pre>UnboundLocalError: local variable 'c' referenced before assignment</pre> |
| <pre>'''để không gây lỗi, cần sử dụng từ khóa global khi thay đổi giá trị của biến toàn cục bên trong function''' def add(): global c c = c + 2 # increment c by 2 print('inside function add, c=',c) c = 5 # global variable add() print('outside function add, c=',c)</pre> | <pre>inside function add, c= 7 outside function add, c= 7</pre> |

3.1.4.3. Sử dụng biến toàn cục thông qua module

– Các bước thực hiện:

- B1: Tạo file python thứ 1 (giả sử đặt tên là globalVariable.py) trong đó chứa các biến toàn cục.
- B2: Tạo file python thứ 2 (giả sử đặt tên là updateVariable.py) trong đó chứa các hàm có sử dụng biến toàn cục. Lưu ý, đầu file này phải sử dụng lệnh import file python thứ nhất
- B3: Tạo file python thứ 3 làm chương trình chính (giả sử đặt tên là main.py), trong đó sử dụng các biến toàn cục

– Ví dụ 3.4

| Mã lệnh | Kết quả |
|--|---------|
| <pre>#Nội dung file globalVariable.py a=10 name='noName'</pre> | |
| <pre>#Nội dung file updateVariable.py from globalVariable import * def addA(x): return a+x def changeName(S): name=S return name</pre> | |
| <pre>#Nội dung file chương trình chính main.py from updateVariable import * #sử dụng biến toàn cục a a=5 k=2 print(addA(k)) S='Jacky'</pre> | |

| | |
|---|---|
| <pre>'''sử dụng hàm changeName trong file updateVariable.py trong hàm này có thay đổi biến toàn cục''' print('Hello '+changeName(S)) #sử dụng biến toàn cục a a=5 k=2 print('Before add, a=', a) print('After add, a=', addA(k)) #sử dụng biến toàn cục a print('Before change, name= '+name) '''sử dụng hàm changeName trong file updateVariable.py trong hàm này có thay đổi biến toàn cục''' S='Jacky' print('After change, name= '+changeName(S))</pre> | <p>Before add, a= 5
After add, a= 12</p> |
| | <p>Before change,
name= noName</p> <p>After change,
name= Jacky</p> |

3.1.4.4. Sử dụng biến global trong các hàm lồng nhau (nested function)

– Ví dụ 3.5

| Mã lệnh | Kết quả |
|---|---|
| <pre>def foo(): x = 20 # biến x là local def bar(): global x # biến x global x = 25 #sử dụng x là local của foo function print("Before calling bar: ", x) bar() '''sử dụng x là global đã được tạo trong hàm bar ''' print("After calling bar: ", x) foo() #sử dụng x là global đã được tạo trong hàm bar print("x in main: ", x)</pre> | <p>Before calling bar: 20</p> <p>After calling bar: 20</p> <p>x in main: 25</p> |

3.1.4.5. Sử dụng biến nonlocal trong các hàm lồng nhau (nested function)

– Ví dụ 3.6: Lưu ý: giá trị của biến nonlocal chỉ thay đổi được ở **ham_trong**.

| Mã lệnh | Kết quả |
|---|---|
| <pre>def ham_ngoai(): x = 'biến local' print('Trước khi gọi hàm trong, x=', x) def ham_trong(): nonlocal x ''' giá trị của biến nonlocal chỉ thay đổi được ở ham_trong''' x = 'biến nonlocal' print('Bên trong, x=', x) ham_trong() print('Sau khi gọi ham_trong, x=', x) ham_ngoai()</pre> | <p>Trước khi gọi hàm trong, x= biến local</p> <p>Bên trong, x= biến nonlocal</p> <p>Sau khi gọi ham_trong, x= biến nonlocal</p> |

3.1.5. Tham số của hàm (parameters / arguments)

3.1.5.1. Tham chiếu (pass by reference) và tham trị (pass by value)

- Tham số có kiểu tham trị (pass by value):
 - Khi giá trị của tham số có bị thay đổi giá trị trong hàm thì sau khi hàm kết thúc, tham số vẫn nhận lại giá trị ban đầu vì khi hàm nhận tham số, bên trong hàm sẽ tạo một biến khác và copy giá trị của tham số truyền vào vào biến mới này, và khi kết thúc hàm thì biến này sẽ bị xóa, do đó không ảnh hưởng đến tham số để truyền cho hàm.
 - Trong Python, mọi tham số luôn truyền theo kiểu tham trị.
- Tham số có kiểu tham chiếu (pass by reference):
 - Khi giá trị của tham số có bị thay đổi giá trị trong hàm thì sau khi hàm kết thúc, tham số sẽ nhận giá trị vừa được thay đổi đó.
 - Trong Python, tham số có kiểu dữ liệu dạng danh sách như *list*, *tuple*, *dictionary* luôn là tham chiếu khi trong thân hàm có thực hiện các thao tác thêm, xóa phần tử của danh sách.
- Ví dụ 3.7

| Mã lệnh | Kết quả |
|---|---|
| <pre>def square(lst): for item in lst: item=item*item print('Inside square function:',lst)</pre> | <pre>'''giá trị của các phần tử trong lst không bị thay đổi do lệnh trong for không tác động đến lst''' Inside square function: [1, 2, 3]</pre> |
| <pre>def additem(lst): for item in range(4,6): lst.append(item) print('Inside additem function:', lst)</pre> | <pre>'''số lượng phần tử trong lst bị thay đổi do phương thức append đã tác động đến lst''' Inside additem function: [1, 2, 3, 4, 5]</pre> |
| <pre>lst=[1,2,3] print('Before square function:',lst) square(lst) print('After square function:', lst) additem(lst) print('After additem function:', lst)</pre> | <pre>Before square function: [1, 2, 3] After square function: [1, 2, 3] After additem function: [1, 2, 3, 4, 5]</pre> |

3.1.5.2. Phân biệt tham số (parameter) và đối số(argument)

- Tham số (parameter) là một biến được xác định bởi một hàm, sẽ nhận một giá trị khi một hàm được gọi.
- Đối Số (argument) là giá trị truyền vào khi gọi hàm.
- Ví dụ 3.8

| Mã lệnh | Diễn giải |
|---|--|
| <pre>def Greetings(sGreeting, sName): print('Good', sGreeting, sName)</pre> | <pre>''' sGreeting và sName được gọi là tham số'''</pre> |
| <pre>Greetings('morning', 'Sửu') S='Dần' Greetings('afternoon', S)</pre> | <pre>""" 'morning', 'Sửu', 'afternoon' và S được gọi là đối số """</pre> |

3.1.5.3. Phân loại đối số

Trong Python, tham số được phân thành 4 loại:

- Required argument (đối số bắt buộc).
- Keyword argument (đối số từ khóa).

- *Default argument* (đối số với giá trị mặc định).
- *Variable-length argument* (đối số có chiều dài thay đổi không xác định).

3.1.5.3.1. Required argument (đối số bắt buộc)

- Là đối số khi truyền vào cho hàm phải **đúng thứ tự** và **đúng số lượng** tham số như khi hàm được xây dựng. Thứ tự các đối số truyền cho tham số được tính từ trái sang phải.
- Ví dụ 3.9

| Mã lệnh | Kết quả |
|---|---|
| <code>def LuyThua(a, b):</code>
<code>print(a, '^', b, '=', a**b)</code> | <code>#Hàm nhận tham số là 2 giá trị kiểu số</code> |
| <code>x=2</code>
<code>y=3</code>
<code>LuyThua(x, y)</code> | <code>2 ^ 3 = 8</code> |
| <code>LuyThua(y, x)</code> | <code>3 ^ 2 = 9</code> |
| <code>LuyThua(y)</code> | <code>'''Bảo lỗi vì truyền thiếu tham số b cho hàm (vì lúc này hàm sẽ xem như a=y)'''</code>
<code>TypeError: LuyThua() missing 1 required positional argument: 'b'</code> |

3.1.5.3.2. Keyword argument (đối số từ khóa)

- Là đối số khi truyền vào hàm **không cần theo đúng thứ tự** của các tham số khi hàm được xây dựng. Tuy nhiên lại cần chỉ rõ tham số nào sẽ nhận giá trị gì hay nói cách khác là **đặt tên cho tham số** khi gọi hàm thông qua các đối số.
- Số lượng đối số khi gọi hàm cũng phải **đúng** với **số lượng** tham số khi hàm được xây dựng.
- Ví dụ 3.10

| Mã lệnh | Kết quả |
|---|------------------------|
| <code>def LuyThua(a, b):</code>
<code>print(a, '^', b, '=', a**b)</code>
<code>x=2</code>
<code>y=3</code>
<code>LuyThua(x, y)</code> | <code>2 ^ 3 = 8</code> |
| <code>LuyThua(b=y, a=x)</code> | <code>2 ^ 3 = 8</code> |

3.1.5.3.3. Default argument (đối số với giá trị mặc định)

- Là trường hợp khi gọi hàm:
 - *Có truyền đối số*: tham số sẽ nhận giá trị được truyền qua đối số như các cách gọi hàm bình thường
 - *Bỏ qua đối số*: hàm sẽ lấy giá trị mặc định đã được khai báo khi xây dựng hàm.
- Giá trị mặc định của tham số được cung cấp ngay khi khai báo tham số của hàm.
- Quy ước về *Default argument*:
 - Chỉ có 1 tham số loại này trong danh sách các tham số của hàm.
 - Nếu có sử dụng, tham số này phải đứng sau cùng.

– Ví dụ 3.11

| Mã lệnh | Kết quả | Diễn giải |
|--|------------|--------------------------------|
| def Cong(a, b=7):
print(a, '+', b, '=', a+b) | | |
| x=2
y=3
Cong(b=y, a=x) | 2 + 3 = 5 | #không sử dụng đối số mặc định |
| Cong(4, 9) | 4 + 9 = 13 | #không sử dụng đối số mặc định |
| Cong(x) | 2 + 7 = 9 | # sử dụng đối số mặc định |

– Ví dụ 3.12

| Mã lệnh | Kết quả |
|---|--|
| def Cong(a, b=None):
if b==None:
return "Truyền thiếu đối số b khi gọi hàm"
else:
return (a+b) | |
| x=2
y=3
print('Lần 1: ', x, '+', y, '=', Cong(b=x, a=y)) | Lần 1: 2 + 3 = 5 |
| print('Lần 2: ', x, '+', y, '=', Cong(a=y)) | Lần 2: 2 + 3 = Truyền thiếu đối số b khi gọi hàm |

3.1.5.3.4. Variable-length argument (tham số có số lượng thay đổi không xác định được trước)

- Nhắc lại về toán tử Splat (dấu hoa thị - *): thường được dùng để đại diện cho tất cả các phần tử trong 1 iterable object.
- Ví dụ 3.13

| Mã lệnh | Kết quả |
|---|--------------------|
| A = [1, 2, 3]
B = [*A, 4, 5, 6] #tương đương với listB = A + [4,5,6]
print(B) | [1, 2, 3, 4, 5, 6] |

- Là tham số được sử dụng khi chưa xác định được số lượng các giá trị truyền vào cho hàm, tức là một tham số loại này có thể bao gồm rất nhiều giá trị đi kèm.
- Để cho biết số lượng tham số truyền cho hàm là không biết trước, Python cho phép dùng dấu * trước tên tham số.
- *args và **kwargs:
 - args (viết tắt của arguments - đối số): trong Python, khi truyền tham số cho hàm ký hiệu **"*args"** ám chỉ danh sách các đối số truyền cho hàm. Các đối số này sẽ được Python đưa vào 1 list để xử lý. Do đó nếu đối số là 1 list thì cũng chỉ được xem là 1 thành phần trong list vừa nêu (sub List).

Ví dụ 3.14: sử dụng lệnh print để lần lượt in các phần tử trong list *danh_sach* ra màn hình

| Mã lệnh | Kết quả |
|--|--|
| def hello(Loi_Chao, *danh_sach):
for item in danh_sach:
print(Loi_Chao, item) | #Không thực hiện lệnh lặp for vì danh_sach là rỗng |

| | |
|--|---|
| <code>#!/usr/bin/env python3</code> | |
| <code>hello('Good morning')</code> | |
| <code>lst=['Ty', 'Suu', 'Dan']</code>
<code>hello('Good afternoon', lst)</code> | '''lst trở thành 1 phần tử duy nhất trong list <code>danhsach</code> (sublist của <code>danhsach</code>) nên cả lst chỉ được in trong 1 lần'''
Good afternoon ['Ty', 'Suu', 'Dan'] |
| <code>hello('Good night', 'Meo', 'Thin', 'Ty')</code> | #mỗi giá trị 'Meo', 'Thin', 'Ty' trở thành 1 phần tử trong list <code>danhsach</code> nên mỗi phần tử được in riêng theo từng lệnh print trong vòng lặp for
Good night Meo
Good night Thin
Good night Ty |

- **kwargs** là viết tắt của *keyword arguments* (đối số từ khóa): ****kwargs** tạo ra một *dictionary* chứa toàn bộ các đối số khi gọi hàm (mỗi cặp trong các đối số sẽ trở thành 1 thành phần của *dictionary*).

Ví dụ 3.15: tham số đầu tiên truyền cho hàm có kiểu là chuỗi, 3 đối số đi sau sẽ được đưa vào 1 *dictionary*

| Mã lệnh | Kết quả |
|--|---|
| <code>def printPetNames(owner, **pets):</code>
<code>print(f"Owner Name: {owner}")</code>
<code>for pet,name in pets.items():</code>
<code>print(f"{pet}: {name}")</code> | Owner Name: Jonathan
mouse: Mickey
fish: ['Larry', 'Curly', 'Moe']
cat: Tom |
| <code>print("Dictionary pets</code>
<code>is:",pets)</code> | Dictionary pets is:
{ 'mouse': 'Mickey',
'fish': ['Larry', 'Curly', 'Moe'],
'cat': 'Tom' } |

#CHƯƠNG TRÌNH CHÍNH

```
printPetNames("Jonathan", mouse="Mickey", fish=["Larry",
                                                "Curly", "Moe"], cat="Tom")
```

3.1.6. Lệnh yield

- Lệnh `yield` sẽ đình chỉ việc thực thi của hàm và trả về một giá trị cho nơi gọi hàm, nhưng vẫn giữ lại đầy đủ trạng thái của hàm để ngay sau khi thực thi xong câu lệnh `yield`, hàm vẫn có thể quay lại thực thi tiếp được tại câu lệnh tiếp theo nằm ngay sau câu lệnh `yield` vừa chạy xong. Điều này cho phép một hàm có thể trả về một loạt các giá trị theo thời gian, thay vì phải tính toán cùng một lúc ra nhiều outputs rồi đưa tất cả chúng vào trong một list và trả về duy nhất list đó.
- Ví dụ 3.16:

```
def myFunc():
    yield 72
    yield 'Sài gòn'
    yield 123.45
for value in myFunc():
    print(value)
```

Kết quả in ra:

```
72
Sài gòn
123.45
```

- Nhận xét:

- So sánh yield và return

| Lệnh | Kết thúc việc thực hiện của hàm | Giá trị trả về |
|---------------|------------------------------------|------------------------------------|
| return | Có | Tùy tham số truyền vào cho hàm |
| yield | Không, chỉ đình chỉ việc thực hiện | Tùy vào ngữ cảnh thực hiện của hàm |

- Lệnh yield thường được sử dụng:

- Khi muốn lặp qua một kiểu dữ liệu dạng iterable (string, list, ...) để xử lý và tạo ra các kết quả, mà không muốn phải lưu trữ toàn bộ các giá trị kết quả trong bộ nhớ (tức là tính ra được kết quả nào thì trả về luôn chứ không cần phải lưu lại kết quả tại mỗi lần xử lý vào một iterable khác – gây tốn bộ nhớ).
- Các generator functions được khai báo giống như hàm bình thường, nhưng khi cần tạo ra (hay phát sinh hoặc trả về) một giá trị, hàm sẽ thực hiện điều đó bằng từ khóa yield thay vì return.

- Nếu bên trong phần thân hàm của một hàm có chứa câu lệnh *yield*, hàm này sẽ tự động trở thành một hàm tạo – *generator function*.

- Ví dụ 3.17:

```
def nextSquareNumber():
    i = 1;
    while True:
        yield i * i
        i += 1 # Lệnh sẽ được thực thi khi hàm được gọi lần kế tiếp
n=100
print("Các số chính phương <= %d: " %n, end=' ')
for num in nextSquareNumber():
    if num > n:
        break
    print(num, end=' ')
```

3.1.7. Hàm ẩn danh (Anonymous function - lambda)

3.1.7.1. Giới thiệu

- Gọi là *Anonymous function* vì function không được khai báo theo cách tiêu chuẩn bằng cách dùng từ khóa *def* mà được dùng 1 cách ngắn gọn bằng từ khóa *lambda*. Thường các *function* dạng này được viết chỉ trên 1 dòng lệnh.

3.1.7.2. Cú pháp

lambda [argument1 [, argument2, ...]]: expression

Giải thích:

- *lambda function*:

- *Lambda function* có *local namespace* riêng và không thể truy cập các biến khác và các biến trong phạm vi *global namespace*.
- Kết quả trả về có thể gồm 1 hoặc nhiều giá trị như các hàm bình thường.

- *argument*: Tương tự như các hàm thông thường, số lượng đối số truyền vào của *lambda* là không giới hạn.

- *expression*
 - Các hàm *lambda* chỉ chấp nhận một và chỉ một biểu thức (một lệnh đơn).
 - Có thể sử dụng tối đa 1 lệnh *if* trong *expression* (xem ví dụ 3.19).
 - Có thể gọi hàm khác trong *expression* (xem ví dụ 3.20).

3.1.7.3. Các cách xây dựng *lambda function*

- **Trường hợp 1:** sử dụng thay cho cách viết hàm thông thường
 - Ví dụ 3.18: *lambda* không có đối số

| Sử dụng hàm thông thường | Sử dụng Anonymous function | Kết quả |
|--|--|---|
| <pre>def Dung(): return True def Sai(): return False #- CHƯƠNG TRÌNH CHÍNH - n=int(input('Nhập điểm: ')) print('Đã hoàn tất môn học: ',end=' ') if n>=5: print(Dung()) else: print(Sai())</pre> | <pre>Dung = lambda : True Sai = lambda : False #--- CHƯƠNG TRÌNH CHÍNH --- n=int(input('Nhập điểm: ')) print('Đã hoàn tất môn học: ',end=' ') if n>=5: print(Dung()) else: print(Sai())</pre> | <pre>'''Tùy thuộc vào giá trị của n mà kết quả sẽ in ra 1 trong 2 dạng sau: Đã hoàn tất môn học: True Đã hoàn tất môn học: False'''</pre> |

- Ví dụ 3.19: cho 2 biến *x* và *y*. Tính tổng của *x* và *y*

| Sử dụng hàm thông thường | Sử dụng Anonymous function | Kết quả |
|---|--|---------|
| <pre>def Cong(a, b): return a + b #--- CHƯƠNG TRÌNH CHÍNH --- x=2; y=3 print('%d+%d=%d'%(x, y, Cong(x, y)))</pre> | <pre>Cong = lambda a, b: a+b #--- CHƯƠNG TRÌNH CHÍNH --- x=2; y=3 print('%d+%d=%d'%(x, y, Cong(x, y)))</pre> | 2+3=5 |

- Ví dụ 3.20: tính giá trị của biểu thức $(3x+2)/(2x-1)$ với *x* là tham số của hàm

| Sử dụng hàm thông thường | Sử dụng Anonymous function | Kết quả |
|---|--|---------|
| <pre>def Tinh(x): return (3*x+2)/(2*x-1) #--- CHƯƠNG TRÌNH CHÍNH --- print('%.2f' %Tinh(2))</pre> | <pre>Tinh = lambda x : (3*x+2)/(2*x-1) #--- CHƯƠNG TRÌNH CHÍNH --- print('%.2f' % Tinh(2))</pre> | 2.67 |

- Ví dụ 3.21: sử dụng *if* trong biểu thức của *lambda*: hàm nhận tham số là số nguyên *x*. Cho biết *x* là số lẻ hay số chẵn.

| Sử dụng hàm thông thường | Sử dụng Anonymous function | Kết quả |
|---|--|------------|
| <pre>def ChanLe(x): if x%2==0: return 'chẵn' else: return 'lẻ' #- CHƯƠNG TRÌNH CHÍNH - n=5 print(n, 'là số', ChanLe(n))</pre> | <pre>ChanLe = lambda x : 'chẵn' if n%2==0 else 'lẻ' #--- CHƯƠNG TRÌNH CHÍNH --- n=5 print(n, 'là số', ChanLe(n))</pre> | 5 là số lẻ |

- Ví dụ 3.22: sử dụng if lồng nhau trong biểu thức của lambda: hàm nhận tham số là số nguyên x. Cho biết x là số dương, bằng không hay là số âm.

| Sử dụng hàm thông thường | Sử dụng Anonymous function | Kết quả |
|--|---|-------------|
| <pre>def KiemTra (x): if x>0: return "dương" elif x==0: return 'không' else: return 'âm' #--- CHƯƠNG TRÌNH CHÍNH --- n=-5 print(n, 'là số', KiemTra(n))</pre> | <pre>KiemTra = lambda x : 'dương' if x>0 else 'không' if x==0 else 'âm' #--- CHƯƠNG TRÌNH CHÍNH --- n=-5 print(n, 'là số', KiemTra(n))</pre> | -5 là số âm |

- Ví dụ 3.23: Hàm lambda có thể trả về nhiều hơn 1 giá trị: Hàm nhận 2 đối số là số thực d, r là chiều dài và chiều rộng của hình chữ nhật. Cho biết chu vi và diện tích hình chữ nhật.

| Sử dụng Anonymous function | Kết quả |
|---|-------------------------|
| <pre>#d,r = map(int,input('Nhập vào 2 số d,r:').split(',')) d=3 r=2 DienTich_Chuvi = lambda d,r : (d*r, 2*(d+r)) #các giá trị trả về cách nhau bởi dấu phẩy print('Diện tích: %d, chu vi:%d' % DienTich_Chuvi(d,r))</pre> | Diện tích: 6, chu vi:10 |

- Ví dụ 3.24: Gọi hàm khác trong expression của lambda. Tìm số nguyên tố lớn hơn và gần với giá trị của x (cho trước). Với x=-5, đoạn chương trình sau sẽ in ra số nguyên tố kế tiếp là 2.

| |
|--|
| <pre>from math import * def LaSNT (x): if (x<=1): return False for i in range(2,int(sqrt(x))+1): if x%i==0: return False return True def NextPrime (x): x=x+1 while LaSNT (x)==False: x+=1 return x #--- CHƯƠNG TRÌNH CHÍNH --- x=-5 Tim = lambda x : NextPrime (x) print('Số nguyên tố lớn hơn và gần với %d nhất là %d' % (x, Tim (x)))</pre> |
|--|

- Ví dụ 3.25: kết hợp sử dụng if và gọi hàm trong lambda: hàm lambda nhận tham số là 1 số nguyên (n), hàm trả về ‘là số nguyên tố’ nếu số n đúng là số nguyên tố, ngược lại sẽ trả về ‘KHÔNG là số nguyên tố’. Với x=5, đoạn chương trình sau sẽ in ra ‘5 LA số nguyên tố’.

```

from math import *
def LaSNT(x):
    if (x<=1):
        return False
    for i in range(2,int(sqrt(x))+1):
        if x%i==0:
            return False
    return True
#--- CHƯƠNG TRÌNH CHÍNH ---
n=5
anoFunc= lambda n : str(n)+' LA so nguyen to' if LaSNT(n)==True else
                                                str(n)+' KHONG la so nguyen to'
print(anoFunc(n))

```

- **Trường hợp 2:** Sức mạnh của *Lambda* được thể hiện rõ hơn khi sử dụng chúng như một hàm ẩn danh bên trong một hàm khác.

- Ví dụ 3.26: tính giá trị của biểu thức $(ax+2)/(bx-1)$ với x là tham số của hàm myfunc:

| Sử dụng Anonymous function | Kết quả |
|--|---|
| <pre> def anoFunc(x): return lambda a,b : (a*x+2)/(b*x-1) Tinh = anoFunc(4) print('Kết quả= %.5f' % Tinh(2,3)) </pre> | <pre> # => (2*4+2)/(3*4-1) Kết quả= 0.90909 </pre> |

- Ví dụ 3.27: Với trường hợp 2 này, ta có thể sử dụng cùng một định nghĩa hàm myfunc cho 2 hàm ẩn danh khác nhau để thay đổi giá trị của x :

| Sử dụng Anonymous function | Kết quả |
|--|--|
| <pre> def myFunc(x): return lambda a,b : (a*x+2)/(b*x-1) #----- anoF4 = myFunc(4) print('Với x=4, kết quả= %.5f' % anoF4(2,3)) anoF5 = myFunc(5) print('Với x=5, kết quả= %.5f' % anoF5(2,3)) </pre> | <pre> # => Với x=4, kết quả= 0.90909 # => Với x=5, kết quả= 0.85714 </pre> |

3.2. Module

3.2.1. Giới thiệu

- Module là một tập tin chứa code Python. Trong module ta có thể định nghĩa *function*, *class*, *variable*, thậm chí thể chèn mã lệnh của chương trình chính.
- Việc nhóm các code có liên quan vào chung 1 module giúp cho code dễ hiểu và dễ sử dụng.
- Người lập trình có thể tham chiếu tới các module đã được xây dựng trước đó.

3.2.2. Phân loại module/thư viện

Có 3 loại module thường thấy là:

- Viết bằng Python: có phần mở rộng là *.py*
- Các thư viện liên kết động: có phần mở rộng là *.dll*, *.pyd*, *.so*, *.sl*, ...
- C-Module liên kết với trình biên dịch.

3.2.3. Cách khai báo và sử dụng file chứa các User Define Function

- Giả sử đã có file Python *MyModule.py* với nội dung như sau:

| Nội dung file <i>MyModule.py</i> |
|--|
| <pre>lst=['Python_program.py', 'www.cisco.com', '192.168.10.1'] def Cong(a, b): return a + b def Tru(a, b): return a - b def Nhan(a, b): return a * b def Chia(a, b): if (b != 0): return 0 else: return a/b</pre> |

- Sau đó, tạo một file có tên *main.py*, trong cùng thư mục với file *MyFunction.py* vừa tạo ở trên, có nội dung như sau:

| Nội dung file <i>main.py</i> | Kết quả khi thực hiện chương trình |
|---|---|
| <pre>import MyModule x = 1 y = 2 print('Sử dụng hàm "Cong" đã được khai báo trong file MyModule.py: ') print (x, ' + ', y, ' = ', MyModule.Cong(x, y)) print('Truy xuất các thành phần trong list có tên là lst được khai báo trong file MyModule.py: ') print(MyModule.lst[0]) print(MyModule.lst[1]) print(MyModule.lst[2])</pre> | <p>Sử dụng hàm "Cong" đã được khai báo trong file <i>MyModule.py</i>:</p> <p>1 + 2 = 3</p> <p>Truy xuất các thành phần trong list có tên là <i>lst</i> được khai báo trong file <i>MyModule.py</i>:</p> <p>Python_program.py</p> <p>www.cisco.com</p> <p>192.168.10.1</p> |

3.2.4. Import module

3.2.4.1. Cú pháp

Để tải một *module* vào *script*, sử dụng 1 trong 2 dạng cú pháp sau:

- Dạng 1:

import module_name

- Dạng 2:

from module_name import functions_name/properties_name

Giải thích:

- Dạng 1:

- Sẽ *import* tất cả các *functions*, *properties* có trong *module_name* vào *script*.
- Khi sử dụng *functions/properties* phải kèm tên *module* đi liền trước, ví dụ

```
print(math.abs(-3)).
```


- Dạng 2:
 - Chỉ *import functions, properties* cần dùng có trong *module_name* vào *script*. Do đó nếu muốn sử dụng dạng 2 để *import* tất cả các *functions, properties* có trong *module_name* vào *script*, ta sử dụng cú pháp sau:

from module_name import *
 - Khi sử dụng *functions/properties* không cần kèm tên *module* đi liền trước, ví dụ `print(abs(-3))`.

3.2.4.2. Lưu ý

- Lệnh *import ...* hoặc *from ...* phải được đặt ở đầu *script*. Lưu ý tên file trong lệnh *import* không có phần mở rộng (*MyFunction*).
- Module chỉ được load 1 lần và không phụ thuộc vào số lần được *import*.

3.2.4.3. Thứ tự tìm kiếm module của trình thông dịch

Khi chương trình có *import* một *module* nào đó, trình thông dịch sẽ tiến hành tìm kiếm file module tương ứng theo thứ tự thư mục sau:

- Thư mục hiện hành mà script đang gọi.
- Các thư mục trong *PYTHONPATH* (nếu có set)
- Các thư mục cài đặt mặc định (*/usr/local/lib/python* trên *Linux/Unix*).
- Module tìm kiếm đường dẫn được lưu trữ trong module hệ thống *sys* có tên *sys.path* variable. Trong *sys.path* variable chứa thư mục hiện hành, *PYTHONPATH* và những cài đặt phụ thuộc mặc định (*installation-dependent default*)
- Nếu không tìm thấy module thì báo lỗi *ImportError*

Ví dụ 3.28: tìm đường dẫn của một module đã được load:

| Mã lệnh | Kết quả minh họa |
|--|---|
| <code>import math</code>
<code>math.__file__</code> | <code>'/usr/lib/python2.5/lib-dynload/math.so'</code> |
| <code>import random</code>
<code>random.__file__</code> | <code>'/usr/lib/python2.5/random.pyc'</code> |

3.2.4.4. Biến `__name__`

- Trong Python, một chương trình hoặc một *module* nhỏ trong một chương trình lớn thì mã nguồn sẽ được lưu dưới dạng là một file có đuôi mở rộng là *.py*.
- Khi cần sử dụng file mã nguồn này, ta có 2 cách:
 - Cách 1: Thực thi mã nguồn Python trực tiếp bằng câu lệnh console của hệ điều hành (*Command Line*) hoặc trong màn hình của ứng dụng *Pycharm*, right click vào chương trình đang có rồi chọn *Run File in Python Console*.
 - Cách 2: *import* mã nguồn Python vào trong một file mã nguồn Python khác.
- Biến `__name__`: khi đoạn mã thực thi, có thể người lập trình cần kiểm soát đoạn mã lệnh đang thực hiện được chứa trong file Python nào? Vì vậy, Python cung cấp biến `__name__` với 2 dạng giá trị như sau:
 - Dạng 1: nếu file Python được thực thi trực tiếp bằng *Command Line* thì biến `__name__` sẽ có giá trị là `"__main__"`.

- **Dạng 2:** nếu file Python được import thành module của chương trình Python khác thì giá trị của biến `__name__` sẽ là tên của file Python đang chứa module đó.

Ví dụ 3.29: Cho 2 file Python *mot.py* và *hai.py* với nội dung như sau:

| | |
|---------------|--|
| <i>mot.py</i> | <pre>from hai import * print ('Run file mot.py, bien __name__ =', __name__) print ('Ket qua khi goi ham trong file hai.py:', NhanDoi(7))</pre> |
| <i>hai.py</i> | <pre>def NhanDoi(n): print('Thuc hien ham trong file hai.py, __name__ =', __name__) return n*2</pre> |

Kết quả khi cho thực thi file *mot.py*

```
Run file mot.py, bien __name__ = __main__
Thuc hien ham trong file hai.py, __name__ = hai
Ket qua khi goi ham trong file hai.py: 14
```

- Câu lệnh `if __name__ == "__main__"`

Thường được dùng khi người lập trình muốn một số đoạn code chỉ được thực thi khi bạn chạy trực tiếp bằng *Command Line* (cách 1) mà không được thực thi khi được import thành module (cách 2).

Ví dụ 3.30: lấy lại ví dụ trên nhưng thêm lệnh `if __name__ == "__main__"` trong cả 2 file *mot.py* và *hai.py*:

| | |
|---------------|---|
| <i>mot.py</i> | <pre>from hai import * if __name__ == "__main__": print ('Run file mot.py, bien __name__ =', __name__) print ('Ket qua khi goi ham trong file hai.py:', NhanDoi(7))</pre> |
| <i>hai.py</i> | <pre>def NhanDoi(n): print('Thuc hien ham trong file hai.py, __name__ =', __name__) if __name__ == "__main__": return n*2 else: return n</pre> |

Kết quả khi cho thực thi file *mot.py*

```
Run file mot.py, bien __name__ = __main__
Thuc hien ham trong file hai.py, __name__ = hai
Ket qua khi goi ham trong file hai.py: 7
```

Ví dụ 3.31: lấy lại ví dụ trên nhưng cả 2 file *mot.py* và *hai.py* đều có hàm *NhanDoi* (2 hàm trùng tên và ở 2 file .py khác nhau). Theo thứ tự ưu tiên thì hàm *NhanDoi* trong file *mot.py* được thực hiện.

| | |
|---------------|--|
| <i>mot.py</i> | <pre>from hai import * def NhanDoi(n): print('Thuc hien ham trong file co __name__ =', __name__) if __name__ == "__main__": return n*3 else: return n if __name__ == "__main__":</pre> |
|---------------|--|

| | |
|--------|--|
| | <pre>print ('Run file mot.py, bien __name__ =', __name__) print ('Ket qua khi goi ham :', NhanDoi(7))</pre> |
| hai.py | <pre>def NhanDoi(n): print('Thuc hien ham trong file hai.py, __name__ =', __name__) if __name__ == "__main__": return n*2 else: return n</pre> |

Kết quả khi cho thực thi file *mot.py*

```
Run file mot.py, bien __name__ = __main__
Thuc hien ham trong file co __name__ = __main__
Ket qua khi goi ham : 21
```

3.2.5. Xem thông tin về module

3.2.5.1. Xem thông tin về các hàm, phương thức, ... có trong module

– Cú pháp: `print (dir(module_name))`

– Ví dụ 3.32:

```
import time
print(dir(time))
```

hay

```
import math
print(dir(math))
```

3.2.5.2. Xem hàm/phương thức thuộc module nào

– Sử dụng phương thức `getmodule` trong module `inspect`

– Ví dụ 3.33:

```
from inspect import getmodule
from math import sqrt
print(getmodule(sqrt))
```

3.2.5.3. Xem đường dẫn chứa file module trên máy tính

Ví dụ 3.34

| Mã lệnh | Kết quả |
|---|--|
| <pre>import sys print("\nList of directories in sys module:") print(sys.path)</pre> | <pre>List of directories in sys module: ['C:\\Program Files\\JetBrains\\PyCharm 2019.1.1\\helpers\\pycharm_matplot lib_backend', 'D:/Hanh/Python/BaiTap/BT04']</pre> |
| <pre>import os print("\nList of directories in os module:") print(os.path)</pre> | <pre>List of directories in os module: <module 'ntpath' from 'C:\\Program Files\\Python37\\lib\\ntpath.py'></pre> |

3.3. Namespace

- Variable là tên/định danh (*identifier*) ánh xạ đến object. *namespace* là một thư mục của các *variable name* (*key*) và các đối tượng tương ứng (*value*).
- Các lệnh (*statement*) có thể truy xuất variable trong *local namespace* và trong *global namespace*. Nếu *local* và *global variable* có cùng tên với nhau, sẽ ưu tiên cho *local variable*.
- Mỗi *function* có *local namespace* riêng. Phương thức của *class* có các quy tắc xác định phạm vi tương tự như *function*.
- Để xem các tên được định nghĩa trong *module*, ta dùng `dir()`.

- Để lấy được danh sách các thuộc tính và phương thức mà *module* hỗ trợ, sử dụng hàm *dir(modulename)*. Ví dụ 3.35

| Mã lệnh | Kết quả |
|---|---|
| <pre>import math print(dir(math))</pre> | <pre>['__doc__', '__file__', '__name__', '__package__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'hypot', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'loglp', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']</pre> |

- Có thể gọi hàm *dir()* không truyền tham số để lấy các thuộc tính và phương thức của *scope* hiện tại đang thực thi. Ví dụ 3.36

| Mã lệnh | Kết quả |
|-------------------------|--|
| <pre>print(dir())</pre> | <pre>['__annotations__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'math']</pre> |

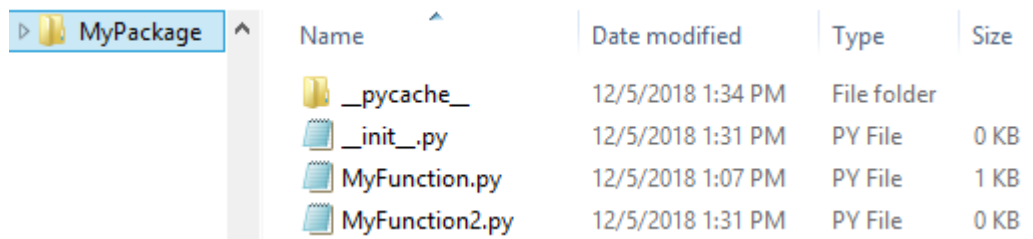
3.4. Package

3.4.1. Giới thiệu

Là cấu trúc dạng thư mục, nơi chứa các *module*, *subpackage*, ... trong ứng dụng *Python*.

3.4.2. Package module

- Có thể gom nhiều *module.py* vào một thư mục và tên thư mục là tên của *package* và tạo một file *__init__.py* trong thư mục này.
- Như vậy, cấu trúc thư của một *package* sẽ như sau:



| Name | Date modified | Type | Size |
|----------------|-------------------|-------------|------|
| __pycache__ | 12/5/2018 1:34 PM | File folder | |
| __init__.py | 12/5/2018 1:31 PM | PY File | 0 KB |
| MyFunction.py | 12/5/2018 1:07 PM | PY File | 1 KB |
| MyFunction2.py | 12/5/2018 1:31 PM | PY File | 0 KB |

- Có thể sử dụng *MyFunction* theo cú pháp import sau:
 - *import MyPackage.MyFunction*
 - hoặc *import MyPackage.MyFunction as MyFunc1*
- Khi sử dụng một module thuộc một package thì các lệnh trong file *__init__.py* sẽ được thực hiện trước. Thông thường thì file *__init__.py* sẽ rỗng.
- Có thể tạo các subpackage bên trong một package theo đúng cấu trúc thư mục, có file *__init__.py*. Ví dụ: *import mypack.mysubpack.mysubsubpack.module*

3.5. Một số module sẵn có trong Python³

3.5.1. Module collections

3.5.1.1. Counter

3.5.1.1.1. Giới thiệu

- Counter là một class trong standard library "collections" với chức năng là đếm trên một argument được truyền cho class này. Argument có thể là 1 iterable object bất kỳ (list, string, dict, ...).
- Kết quả của việc đếm là thu được 1 dict object chứa các object và số lần các phần tử xuất hiện (tần suất xuất hiện) và dict object này được sắp xếp giảm dần theo value.
- Do kết quả là 1 dict, nên có thể thực hiện việc cập nhật, xóa như mọi dict khác.

3.5.1.1.2. Tạo mới Counter Object

- **Cách 1:** tạo Counter object trống (empty) bằng cách bỏ qua đối số trong phương thức khởi tạo của Counter.

Ví dụ 3.37

| Mã lệnh | Kết quả |
|---|-----------|
| <pre>from collections import Counter counterObj = Counter() print(counterObj)</pre> | Counter() |

- **Cách 2:** tạo Counter object bằng cách cung cấp đối số là 1 dictionary trong phương thức khởi tạo của Counter

Ví dụ 3.38

| Mã lệnh | Kết quả |
|--|-----------------------------------|
| <pre>from collections import Counter counterObj = Counter({'a', 'a', 'b'}) print(counterObj)</pre> | Counter({'a': 2, 'b': 1}) |
| <pre>#hoặc counterObj = Counter(a=2, b=3, c=1) print(counterObj)</pre> | Counter({'b': 3, 'a': 2, 'c': 1}) |

- **Cách 3:** sử dụng bất kỳ Iterable nào làm đối số để tạo Counter object, kể cả chuỗi ký tự (string) và List.

Ví dụ 3.39

| Mã lệnh | Kết quả |
|--|---|
| <pre>from collections import Counter # string là đối số của Counter counterObj = Counter('madam') print(counterObj)</pre> | Counter({'m': 2, 'a': 2, 'd': 1}) |
| <pre># List là đối số của Counter words_list = ['Cat', 'Dog', 'Horse', 'Dog'] counterObj = Counter(words_list) print(counterObj)</pre> | Counter({'Dog': 2, 'Cat': 1, 'Horse': 1}) |

³ Xem module OS trong chương 7 của tài liệu này

| | |
|--|--|
| <pre># Dictionary là đối số của Counter word_count_dict = {'Dog': 2, 'Cat': 1, 'Horse': 1} counterObj = Counter(word_count_dict) print(counterObj)</pre> | <pre>Counter({'Dog': 2, 'Cat': 1, 'Horse': 1})</pre> |
|--|--|

- **Cách 4:** có thể sử dụng dữ liệu không phải là số cho các giá trị đếm, nhưng điều đó sẽ làm mất mục đích của lớp *Counter*.

Ví dụ 3.40

| Mã lệnh | Kết quả |
|--|---|
| <pre>from collections import Counter dem = Counter(name='Pankaj', age=20) print(dem)</pre> | <pre>Counter({'name': 'Pankaj', 'age': 20})</pre> |

3.5.1.1.3. Một số phương thức và thao tác trên Counter object

a. Truy xuất value thông qua key của phần tử trong Counter object

- Nếu sử dụng key không tồn tại, kết quả sẽ trả về zero (0) mà không phát sinh lỗi *KeyError*.

Ví dụ 3.41

| Mã lệnh | Kết quả |
|--|---------|
| <pre>from collections import Counter counterObj = Counter({'Dog': 2, 'Cat': 1, 'Horse': 1}) valueDog = counterObj['Dog'] print(valueDog)</pre> | 2 |
| <pre>valueElephant = counterObj['Elephant'] print(valueElephant)</pre> | 0 |

b. Thay đổi giá trị count của phần tử trong Counter object

- Có thể thay đổi giá trị hiện có trong *Counter object*.
- Nếu phần tử không tồn tại trong *Counter object*, thì phần tử đó sẽ được thêm vào *Counter object*.
- Ví dụ 3.42

| Mã lệnh | Kết quả |
|---|---|
| <pre>from collections import Counter counterObj = Counter({'Dog': 2, 'Cat': 1, 'Horse': 1}) '''thay đổi value của key tồn tại trong Counter object''' counterObj['Horse']=4 print(counterObj)</pre> | <pre>Counter({'Horse': 4, 'Dog': 2, 'Cat': 1})</pre> |
| <pre>'''thay đổi value của key không tồn tại=> thêm mới key:value vào Counter object''' counterObj['Elephant']=3 print(counterObj)</pre> | <pre>Counter({'Horse': 4, 'Elephant': 3, 'Dog': 2, 'Cat': 1})</pre> |

c. Xóa phần tử trong Counter object

- Sử dụng lệnh *del* để xóa phần tử trong *Counter object*.

- Ví dụ 3.43

| Mã lệnh | Kết quả |
|---|--|
| <pre>from collections import Counter counterObj = Counter({'Dog': 2, 'Cat': 1, 'Horse': 1}) del counterObj['Dog'] print(counterObj)</pre> | <pre>Counter({'Cat': 1, 'Horse': 1})</pre> |

d. Phương thức *elements()*

- Phương thức này trả về danh sách các phần tử có trong *Counter object* với giá trị *key* sẽ được xuất hiện *value* lần. Do đó những *key* có *value* ≤ 0 sẽ không có trong kết quả.
- Ví dụ 3.44

| Mã lệnh | Kết quả |
|---|---------------------------|
| <pre>from collections import Counter counterObj = Counter({'Dog': 2, 'Cat': 1, 'Horse': 0, 'Elephant': -1}) elements = counterObj.elements() # doesn't return elements with count 0 or less for value in elements: print(value, end=', ')</pre> | <pre>Dog, Dog, Cat,</pre> |

e. Phương thức *most_common(n)*

- Phương thức này trả về *n* phần tử đầu tiên trong *Counter object*. Do đó nếu *n* ≤ 0 thì danh sách kết quả là rỗng.
- Nếu bỏ qua tham số *n*:
 - Trả về tất cả các phần tử đầu tiên trong *Counter object*.
 - Lúc này có thể sử dụng cặp ngoặc vuông [*index_of_elements*] [0/1]: nếu sử dụng số 0, sẽ truy xuất *key* (của phần tử thứ *index_of_elements*; ngược lại nếu sử dụng số 1 sẽ truy xuất *value*).
 - Hoặc có thể sử dụng toán tử *slicing* (cắt lát) để truy xuất phần tử cuối cùng trong *Counter object*.

Ví dụ 3.45: đếm tần suất xuất hiện của 1 giá trị có trong list. trong ví dụ này sử dụng các ký tự viết tắt với ý nghĩa như sau *R=red, G=green, B=black, P=pink, W=white, O=orange, E=eyes*

| Mã lệnh | Kết quả |
|---|---|
| <pre>from collections import Counter colorList = ['R', 'G', 'B', 'P', 'B', 'W', 'B', 'E', 'W', 'B', 'O', 'P', 'P', 'R', 'R', 'W', 'O', 'W', "B", 'P', 'G', 'G', 'P', 'G', 'P', 'W', 'O', "O", 'R'] counterObj = Counter(colorList) print('counterObj=', counterObj) # most_common() mostCommon = counterObj.most_common(1) print('n=1:', mostCommon) mostCommon = counterObj.most_common(2) print('n=2:', mostCommon) mostCommon = counterObj.most_common(3) print('n=3:', mostCommon)</pre> | <pre>counterObj= Counter({'P': 6, 'B': 5, 'W': 5, 'R': 4, 'G': 4, 'O': 4, 'E': 1}) n=1: [('P', 6)] n=2: [('P', 6), ('B', 5)] n=3: [('P', 6), ('B', 5), ('W', 5)]</pre> |

| | |
|---|---|
| <pre>mostCommon = counterObj.most_common() print('ignore n:', mostCommon)</pre> | <pre>ignore n: [('P', 6), ('B', 5), ('W', 5), ('R', 4), ('G', 4), ('O', 4), ('E', 1)]</pre> |
| <pre>maxCount = counterObj.most_common()[0][1] maxKey = counterObj.most_common()[0][0] print(maxKey, 'dat tan suat lon nhat la', maxCount)</pre> | <pre>P dat tan suat lon nhat la 6</pre> |
| <pre># least common minCount = counterObj.most_common()[len(counterObj)-1][1] minKey = counterObj.most_common()[len(counterObj)-1][0] print(minKey, 'dat tan suat nho nhat la', minCount)</pre> | <pre>E dat tan suat nho nhat la 1</pre> |
| <pre>#hoặc least_common_element = counterObj.most_common()[::-2:-1] print(least_common_element)</pre> | <pre>[('E', 1)]</pre> |

f. Phương thức *subtract()*

- Được sử dụng để trừ tổng số phần tử của *Counter object* này cho *Counter object* khác.
- Ví dụ 3.46

| Mã lệnh | Kết quả |
|---|---|
| <pre>from collections import Counter counterObj1 = Counter('ababab') print(counterObj1)</pre> | <pre>Counter({'a': 3, 'b': 3})</pre> |
| <pre>counterObj2 = Counter('abc') print(counterObj2)</pre> | <pre>Counter({'a': 1, 'b': 1, 'c': 1})</pre> |
| <pre>counterObj1.subtract(counterObj2) print(counterObj1)</pre> | <pre>Counter({'a': 2, 'b': 2, 'c': -1})</pre> |

g. Phương thức *update()*

- Được sử dụng để thêm số đếm cho *Counter object* này từ một *Counter object* khác.
- Ví dụ 3.47

| Mã lệnh | Kết quả |
|---|--|
| <pre>from collections import Counter counterObj1 = Counter('ababab') print(counterObj1)</pre> | <pre>Counter({'a': 3, 'b': 3})</pre> |
| <pre>counterObj2 = Counter('abc') print(counterObj2)</pre> | <pre>Counter({'a': 1, 'b': 1, 'c': 1})</pre> |
| <pre>counterObj1.update(counterObj2) print(counterObj1)</pre> | <pre>Counter({'a': 4, 'b': 4, 'c': 1})</pre> |

3.5.1.1.4. Các phép toán được dùng trên 2 *Counter object*

- Có thể thực hiện một số thao tác số học trên *Counter object*. Tuy nhiên, *Counter object* kết quả chỉ chứa các phần tử có giá trị là số dương (>0).

Ví dụ 3.48

| Mã lệnh | Kết quả |
|---|--|
| <pre>from collections import Counter c1 = Counter(a=2, b=0, c=-1) c2 = Counter(a=1, b=-1, c=2) c = c1 + c2 print(c)</pre> | <pre>#Counter({'a': 2, 'b': 0, 'c': -1}) #Counter({'c': 2, 'a': 1, 'b': -1}) Counter({'a': 3, 'c': 1})</pre> |
| <pre>c = c1 - c2 print(c)</pre> | <pre>Counter({'a': 1, 'b': 1})</pre> |
| <pre>c = c1 & c2 print(c)</pre> | <pre>Counter({'a': 1})</pre> |
| <pre>c = c1 c2 print(c)</pre> | <pre>Counter({'a': 2, 'c': 2})</pre> |

3.5.1.1.5. Một số thao tác khác trên Counter object

Ví dụ 3.49

| Mã lệnh | Kết quả |
|---|---|
| <pre>from collections import Counter counter = Counter({'a': 3, 'b': 3, 'c': 0}) # Cộng giá trị của tất cả các phần tử print(sum(counter.values()))</pre> | <pre>6</pre> |
| <pre># Chuyển Counter object sang list print(list(counter))</pre> | <pre>['a', 'b', 'c']</pre> |
| <pre># Chuyển Counter object sang set print(set(counter))</pre> | <pre>{'c', 'a', 'b'}</pre> |
| <pre># Chuyển Counter object sang dict print(dict(counter))</pre> | <pre>{'a': 3, 'b': 3, 'c': 0}</pre> |
| <pre># Chuyển Counter object sang dict_items print(counter.items())</pre> | <pre>dict_items([('a', 3), ('b', 3), ('c', 0)])</pre> |
| <pre>''' Xóa những phần tử có giá trị <=0 ra khỏi Counter object ''' counter = Counter(a=2, b=3, c=-1, d=0) counter = +counter print(counter)</pre> | <pre>Counter({'b': 3, 'a': 2})</pre> |
| <pre># xóa trống Counter object counter.clear() print(counter)</pre> | <pre>Counter()</pre> |

3.5.2. Module random

- **Công dụng:** Giúp tạo ra số ngẫu nhiên
- **Sử dụng:** import đầu chương trình `from random import *`

3.5.2.1. Hàm dùng cho số nguyên (integers)

Gồm 2 hàm: `randrange([start,] stop [, step])`
 `randint (start, end)`

- Trong đó:
 - **start:** số nhỏ nhất có thể phát sinh ngẫu nhiên. Mặc định là 0
 - **stop:** với `stop-1` là số lớn nhất có thể phát sinh ngẫu nhiên.

- **end** \approx **stop+1**. Do đó:
 - Hàm `randint(start, end)` trả về số ngẫu nhiên n , với $start \leq n \leq end$.
 - Việc sử dụng 2 hàm sau là tương đương nhau:

$$\text{random.randint}(start, end) \approx \text{randrange}(start, stop+1)$$
- **step**: lựa chọn các số có dạng $start + (step * k)$ và nằm trong khoảng từ $start$ đến $stop-1$. Mặc định $step=1$.

– Minh họa tham số step

| Mã lệnh | Các số được chọn để phát sinh ngẫu nhiên |
|----------------------------------|--|
| <code>randrange(2, 10, 1)</code> | 2, 3, 4, 5, 6, 7, 8, 9 |
| <code>randrange(2, 10, 2)</code> | 2, 4, 6, 8 |
| <code>randrange(2, 10, 3)</code> | 2, 5, 8 |

– Ví dụ 3.50

| Mã lệnh | Kết quả |
|--|--------------------------------|
| <pre>from random import * i=0 mylist=[] while i<10: mylist.append(randrange(2, 10, 3)) i+=1 print(mylist)</pre> | [8, 8, 2, 2, 5, 2, 8, 5, 2, 8] |
| <pre>for i in range(1, 10): print(randint(2, 6), end=' ')</pre> | 5 4 4 2 3 6 4 3 6 |

3.5.2.2. Hàm dùng cho số thực (float)

- **random()**
 - Trả về giá trị ngẫu nhiên kiểu float nằm trong khoảng [0.0, 1.0).
- **uniform(a, b)**
 - Trả về giá trị ngẫu nhiên n có kiểu là float nằm trong khoảng
 - $a \leq n \leq b$ khi $a \leq b$
 - và $b \leq n \leq a$ khi $a > b$
 - Giá trị điểm cuối b có thể có hoặc không có trong kết quả phát sinh số ngẫu nhiên tùy thuộc vào việc làm tròn trong phương trình $a + (b-a) * \text{random}()$.

3.5.2.3. Hàm dùng cho tập hợp có thứ tự

- **choice(seq)**
 - Trả về phần tử ngẫu nhiên trong một tập hợp `seq` không rỗng.
 - Nếu tập hợp `seq` rỗng sẽ phát sinh lỗi `IndexError`.
 - Ví dụ 3.51

| Mã lệnh | Kết quả |
|--|--------------------------------|
| <pre>from random import * i=0 mylist=[] while i<10:</pre> | [5, 5, 2, 2, 6, 8, 0, 6, 6, 6] |

```
mylist.append(randint(0, 10))
i+=1
print(mylist)
print(choice(mylist))
```

6

– *sample(range(a, b), quantity)*

- Trả về *quantity* số ngẫu nhiên *n*, với $a \leq n < b$ và *n* không trùng nhau.
- Sẽ phát sinh lỗi *ValueError* khi $b - a > quantity$
- Ví dụ 3.52: tạo ngẫu nhiên 2 list số nguyên. Tạo ra list thứ 3 chứa các số có trong cả 2 list trước đó

| Mã lệnh | Kết quả |
|--|---|
| <pre>from random import * listA = sample(range(1,15), 10) print ("listA= ",listA) listB = sample(range(1,15), 5) print ("listB= ",listB) listResult = [i for i in listA if i in listB] print ("listResult= ",listResult) listD = sample(range(1,5), 10) print ("listA= ",listA)</pre> | <pre>listA= [13, 3, 12, 8, 5, 7, 11, 6, 4, 1] listB= [4, 7, 13, 1, 14] listResult= [13, 7, 4, 1] ValueError: Sample larger than population or is negative</pre> |

3.5.3. Module math

| Hàm ⁴ | Kết quả trả về | Ví dụ | Kết quả |
|---------------------------------|--|---|---------------|
| <code>abs(x)</code> | Trị tuyệt đối của số nguyên x | <code>print(abs(-15))</code> | 15 |
| <code>ceil(x)</code> | Giá trị nguyên nhỏ nhất lớn hơn x. | <code>print(ceil(-15.27))</code> | 16 |
| <code>copysign(x, y)</code> | Trả về kết quả: = x, khi $y \geq 0$
= -x, khi $y < 0$ | <pre>x=15.2 y=-2 z=copysign(x, y) print('z=',z)</pre> | z = -15.2 |
| <code>fabs(x)</code> | Trị tuyệt đối của số thực x | <code>print(abs(-5.17))</code> | 5.17 |
| <code>factorial(x)</code> | Trả về giai thừa (factorial) của x | | |
| <code>floor(x)</code> | Giá trị nguyên lớn nhất nhỏ hơn x. | <code>print(floor(-15.27))</code> | -15 |
| <code>fmod(x, y)</code> | Trả về phần dư (remainder) của phép chia x cho y | | |
| <code>isfinite(x)</code> | Trả về True nếu x không phải là vô cực cũng không phải là số (NaN- Not a Number) | | |
| <code>isinf(x)</code> | Trả về True nếu x là vô cực dương hoặc âm (positive or negative infinity) | | |
| <code>isnan(x)</code> | Trả về True nếu x không phải là số (NaN- Not a Number) | | |
| <code>ldexp(x, i)</code> | Trả về kết quả của $x * (2^i)$ | | |
| <code>modf(x)</code> | Tách số thực x thành 2 phần: phần lẻ và phần nguyên | <code>print(modf(-15.27))</code> | (0.27, -15.0) |
| <code>max(x1, x2, ...xn)</code> | Giá trị lớn nhất trong các tham số x_i | | |
| <code>min(x1, x2, ...xn)</code> | Giá trị nhỏ nhất trong các tham số x_i | | |

⁴ Ngoài các hàm trong bảng này thuộc module *math*, còn có hàm *round* đã được giới thiệu trong mục 2.11.9

| | | | |
|------------------------|--|-----------------------------|---|
| <code>trunc(x)</code> | Trả về giá trị nguyên bị cắt (truncated) của số thực x | | |
| <code>pow(x, y)</code> | x^y | | |
| <code>sqrt(x)</code> | Trả về căn bậc 2 của x | <code>print(sqrt(9))</code> | 3 |
| <code>pi</code> | Hằng số PI (3.14159...) | | |
| <code>e</code> | Hằng số e (2.71828...) | | |

3.5.4. Các module liên quan tới thời gian

3.5.4.1. Các định dạng hiển thị về thời gian

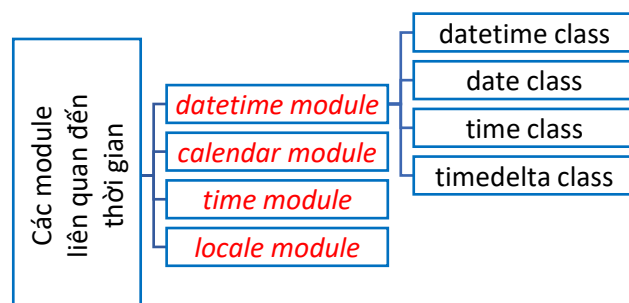
Format Ý nghĩa (giả sử ngày đang xét là 10/23/2019)

| | |
|-----------------|---|
| <code>%b</code> | 3 ký tự đầu tiên của tên tháng. VD: Oct |
| <code>%d</code> | Số nguyên chỉ ngày (1-31). VD: 23 |
| <code>%Y</code> | Số nguyên chỉ năm với 4 ký số. VD: 2019 |
| <code>%H</code> | Trả về số giờ. |
| <code>%M</code> | Trả về số phút (từ 0 đến 59). |
| <code>%S</code> | Trả về số giây (từ 0 đến 59). |

Ví dụ 3.53

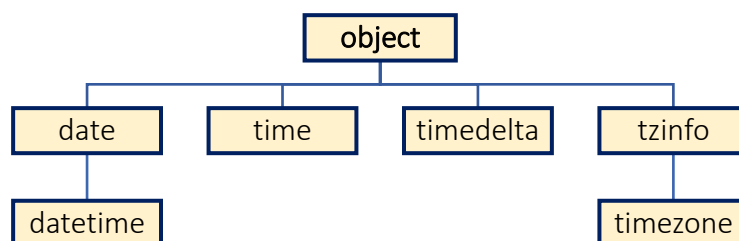
| Mã lệnh | Kết quả |
|---|----------------------|
| <pre>import datetime x = datetime.datetime(2018, 10, 23) print(x.strftime("%b %d %Y %H:%M:%S"))</pre> | Oct 23 2018 00:00:00 |
| <pre>x = datetime.datetime(2018, 10, 23, 15, 59, 24) print(x.strftime("%b %d %Y %H:%M:%S"))</pre> | Oct 23 2018 15:59:24 |

Các module liên quan đến thời gian được giới thiệu trong tài liệu này:



3.5.4.2. datetime module

- Module *datetime* cung cấp cách thức xử lý ngày giờ cùng các mốc thời gian.
- Module *datetime* định nghĩa một số lớp con (*subClass*) đại diện cho ngày tháng và thời gian.



| Class | Mô tả |
|----------------------------|---|
| <code>datetime.date</code> | Một đối tượng <i>date</i> đại diện cho ngày tháng (<i>date</i>), không bao gồm thời gian, theo Dương lịch. Ngày tháng được truyền ở dạng ' <i>year, month, day</i> '. |

| | |
|---------------------------------|---|
| <code>datetime.datetime</code> | Một đối tượng <i>datetime</i> đại diện cho ngày tháng (<i>date</i>) và thời gian, theo Dương lịch. |
| <code>datetime.time</code> | Một đối tượng <i>time</i> đại diện cho thời gian (<i>time</i>), không bao gồm ngày tháng (<i>date</i>). |
| <code>datetime.tzinfo</code> | Là một lớp cơ sở trừu tượng (<i>base abstract class</i>) cho các đối tượng thông tin múi giờ (<i>timezone</i>). |
| <code>datetime.timezone</code> | Là một lớp con trực tiếp của lớp <i>tzinfo</i> , theo chuẩn UTC (<i>Coordinated Universal Time</i>) (Giờ hợp nhất quốc tế). |
| <code>datetime.timedelta</code> | Đối tượng <i>timedelta</i> mô tả một khoảng thời gian (<i>duration</i>) giữa 2 thời điểm. Thứ tự các tham số lần lượt là: <i>days</i> , <i>seconds</i> , <i>microseconds</i> , <i>milliseconds</i> , <i>minutes</i> , <i>hours</i> , <i>weeks</i> , <i>fold</i> |

- Ví dụ 3.54: in ngày tháng năm hiện tại theo định dạng: năm-tháng-ngày giờ : phút : giây và cho biết 5 ngày và 2 giờ trước là ngày và giờ nào? 5 ngày và 3 giờ nữa là ngày giờ nào?

Mã lệnh

Kết quả

| | |
|---|---|
| <pre>from datetime import * Now = datetime.now() print('Current date and time:', Now.strftime("%Y-%m-%d %H:%M:%S")) Pre_5day = Now - timedelta(days=5, hours=2) print('5 days and 2 hours BEFORE current date :', Pre_5day)</pre> | <pre>Current date and time: 2019-05-24 16:04:11 5 days and 2 hours BEFORE current date : 2019-05-19 14:04:11.171217</pre> |
| <pre>Next_5day = Now - timedelta(days=-5, hours=-3) print('5 days and 3 hours AFTER current date :', Next_5day)</pre> | <pre>5 days and 3 hours AFTER current date : 2019-05- 29 19:04:11.171217</pre> |

3.5.4.2.1. Một số phương thức trong class *datetime.datetime*

(i)- *Datetime_variable.strftime*

- Công dụng: định dạng thời gian thành một String.
- Cú pháp: `Datetime_variable.strftime(chuỗi định dạng mẫu)`
- Phương thức *strftime* có một số chỉ thị để định dạng giá trị ngày:

Format

Ý nghĩa

| | |
|-----------------|--|
| <code>%a</code> | Trả về 3 ký tự đầu tiên của thứ. VD: Wed. |
| <code>%A</code> | Trả về tên đầy đủ của thứ. VD: Wednesday. |
| <code>%B</code> | Trả về tên đầy đủ của thứ tháng. VD: September. |
| <code>%w</code> | Trả về giá trị số từ 0 đến 9 với Sunday được tính là 0. |
| <code>%m</code> | Trả về giá trị số của tháng (từ 01 đến 12). |
| <code>%p</code> | Trả về AM/PM dùng cho định dạng thời gian |
| <code>%y</code> | Trả về 2 ký số cuối của năm. VD "19" thay cho "2019". |
| <code>%Y</code> | Trả về năm với đủ 4 ký số . VD "2019". |
| <code>%f</code> | Trả về microsecond từ 000000 đến 999999. |
| <code>%Z</code> | Trả về timezone. |
| <code>%z</code> | Trả về UTC offset. |
| <code>%j</code> | Trả về số ngày trong năm (từ 001 đến 366) |
| <code>%W</code> | Trả về số thứ tự của tuần trong năm (từ 00 đến 53), với Monday được xem là ngày đầu tiên của tuần. |

| | |
|-----------------|--|
| <code>%U</code> | Trả về số thứ tự của tuần trong năm (từ 00 đến 53), với Sunday được xem là ngày đầu tiên của tuần. |
| <code>%C</code> | Trả về ngày giờ địa phương (local). |
| <code>%x</code> | Trả về ngày địa phương (local). |
| <code>%X</code> | Trả về giờ địa phương (local). |

- Ví dụ 3.55

| Mã lệnh | Kết quả |
|---|--|
| <pre>from datetime import * homnay = datetime.now() print("Hom nay la ngay ", homnay.strftime("%d-%b-%Y")) print(Nam hien tai la:', homnay.strftime("%Y")) '''lệnh trên tương đương với lệnh sau''' print(Nam hien tai la:', date.today().year)</pre> | <pre>Hom nay la ngay 23-Oct-2019 Nam hien tai la 2019 Nam hien tai la 2019</pre> |

(ii)- **`Datetime_variable.strptime`**

- Công dụng: phân tích một String có định dạng thời gian là '%d %B, %Y' (ngày, 3 ký tự đầu của tháng, dấu phẩy, 4 ký số của năm) thành 1 object date (ngày, tháng, năm).

- Ví dụ 3.56

| Mã lệnh | Kết quả |
|--|---|
| <pre>from datetime import datetime dateString = "7 March, 2019" print("date string=", dateString) dateObject = datetime.strptime(dateString, "%d %B, %Y") print("date object=", dateObject.strftime("%d/%m/%Y"))</pre> | <pre>date string= 7 March, 2019 date object= 07/03/2019</pre> |

(iii)- **`datetime.datetime.now()`**

- Trả về ngày giờ của hệ thống
- Ví dụ 3.57

| Mã lệnh | Kết quả |
|--|--|
| <pre>from datetime import datetime hientai = datetime.now() print("Bay gio la", hientai)</pre> | <pre>Bay gio la 2019-07-09 09:52:10.001547</pre> |

(iv)- **`datetime.datetime (year, month, day)`**

- Trả về 1 đối tượng date dựa trên 3 tham số year, month, day nếu hợp lệ; ngược lại phát sinh lỗi *ValueError*.

- Ví dụ 3.58

| Mã lệnh | Kết quả |
|--|--|
| <pre>import datetime d=23 m=10 y=2019 hom_nay=datetime.datetime(y,m,d) print("Hom nay la ngay ",hom_nay) hom_nay=datetime.datetime(2019,10,23) print("Hom nay la ngay ",hom_nay)</pre> | <pre>Hom nay la ngay 2019-10-23 00:00:00 Traceback (most recent call last): ValueError: month must be in 1..12</pre> |

(v)- **datetime.combine** (dateObj, time Obj)

- Trả về 1 đối tượng *datetime* mới có các thành phần ngày bằng với ngày của tham số *dateObj*, và thành phần thời gian bằng với thời gian của tham số *timeObj*. Nếu tham số *dateObj* là một đối tượng *datetime*, các thành phần thời gian của nó bị bỏ qua.
- Ví dụ 3.59

| Mã lệnh | Kết quả |
|--|---|
| <pre>from datetime import date, datetime, time dObj = date.today() tObj = time(18, 27, 59) print('dateObject=', dObj) print('timeObject=', tObj) print('combine=', datetime.combine(dObj, tObj))</pre> | <pre>dateObject= 2020-04-02 timeObject= 18:27:59 combine= 2020-04-02 18:27:59</pre> |
| <pre>dObj=datetime.now() print('timeObject=', tObj) print('datetimeObject=', dObj) #time của datetime object là dObj bị bỏ qua print('combine=', datetime.combine(dObj, tObj))</pre> | <pre>timeObject= 18:27:59 datetimeObject= 2020-04-02 16:34:29.278271 combine= 2020-04-02 18:27:59</pre> |

3.5.4.2.2. Một số phương thức trong class *datetime.date*(i)- **date** (year, month, day)

- Trả về 1 đối tượng *date* dựa trên 3 tham số là 3 số nguyên *year*, *month*, *day* nếu hợp lệ; ngược lại phát sinh lỗi *ValueError*.
- Ví dụ 3.60

| Mã lệnh | Kết quả |
|--|-----------------------|
| <pre>from datetime import date d = date(2019, 10, 23) print(d)</pre> | <pre>2019-10-23</pre> |

(ii)- **today** ()

- Trả về 1 đối tượng *date* với giá trị là ngày hiện tại.
- Ví dụ 3.61

| Mã lệnh | Kết quả |
|--|---|
| <pre>from datetime import date hom_nay = date.today() print("Nam - thang - ngay hien tai la:", hom_nay) print("Ngay hien tai la:", hom_nay.day) print("Thang hien tai la:", hom_nay.month) print("Nam hien tai la:", hom_nay.year)</pre> | <pre>Nam - thang - ngay hien tai la: 2020-02-29 Ngay hien tai la: 29 Thang hien tai la: 2 Nam hien tai la: 2020</pre> |

3.5.4.2.3. Một số phương thức trong class *datetime.time*(i)- **time** ()

- Trả về 1 đối tượng thuộc class *time* chứa thông tin về thời gian, không bao gồm thông tin về ngày tháng.
- Ví dụ 3.62

| Mã lệnh | Kết quả |
|---|-----------------------------|
| <pre>from datetime import time # time(hour = 0, minute = 0, second = 0)</pre> | <pre>time1 = 00:00:00</pre> |

| | |
|--|--------------------------------------|
| <code>time1 = time()</code>
<code>print("time1 =", time1)</code> | |
| <code># time(hour, minute and second)</code>
<code>time2 = time(10, 23, 56)</code>
<code>print("time2 =", time2)</code> | <code>time2 = 10:23:56</code> |
| <code># time(hour, minute and second)</code>
<code>time3 = time(hour = 10, minute = 23, second = 56)</code>
<code>print("time3 =", time3)</code> | <code>time3 = 10:23:56</code> |
| <code># time(hour, minute, second, microsecond)</code>
<code>time4 = time(10, 23, 56, 123456)</code>
<code>print("time4 =", time4)</code> | <code>time4 = 10:23:56.123456</code> |
| <code>print("gio =", time4.hour)</code> | <code>gio = 10</code> |
| <code>print("phut =", time4.minute)</code> | <code>phut = 23</code> |
| <code>print("giay =", time4.second)</code> | <code>giay = 56</code> |
| <code>print("micro giay =", time4.microsecond)</code> | <code>micro giay = 123456</code> |

3.5.4.2.4. class datetime.timedelta

- Đối tượng thuộc class *timedelta* lưu giữ khoảng thời gian khác biệt giữa 2 mốc thời gian.
- Các toán tử/hàm hỗ trợ *timedelta*:

| Toán tử/hàm | Ý nghĩa | Ví dụ | |
|-----------------------|---|---|------------------------------------|
| | | Mã lệnh | Kết quả |
| = | Gán <i>timedelta object</i> này cho <i>timedelta object</i> khác | <code>from datetime import timedelta</code>
<code>t1=timedelta(hours=2, minutes=3)</code>
<code>t2=timedelta(hours=8, seconds=12)</code>
<code>t3 = t2</code>
<code>print('Phep gan: t3=', t3)</code> | Phep gan: t3= 8:00:12 |
| + | Cộng 2 <i>timedelta object</i> | <code>t4 = t1 + t2</code>
<code>print('Tong: t4=', t4)</code> | Tong: t4= 10:03:12 |
| - | Trừ 2 <i>timedelta object</i> | <code>t5 = t1 - t2</code>
<code>print('Hieu: t5=', t5)</code> | Hieu: t5= -1 day, 18:02:48 |
| * | Nhân <i>timedelta object</i> với số nguyên | <code>k=3</code>
<code>t6 = t2 * k</code>
<code>print('Nhan so nguyen: t6=', t6)</code> | Nhan so nguyen: t6= 1 day, 0:00:36 |
| +timedelta obj | Trả về <i>timedelta object</i> (<i>tdObj</i>) | <code>print('Phep +t3=', +t3)</code> | Phep +t3= 8:00:12 |
| -timedelta obj | Trả về <i>timedelta object</i> với giá trị sẽ là: <i>tdO + (-tdObj) = 0</i> | <code>print('Phep -t3=', -t3)</code> | Phep -t3= -1 day, 15:59:48 |
| abs(tdObj) | Giá trị tuyệt đối, tương đương với <i>+tdObj</i> khi <i>tdObj.days >= 0</i> , và là <i>-tdObj</i> khi <i>tdO.days < 0</i> | <code>print('abs(t3)=', abs(t3))</code> | abs(t3)= 8:00:12 |
| str(tdObj) | Trả về string theo mẫu <i>[D day[s,][H]H:MM:SS[.UUUUUU];</i> với <i>D</i> có thể nhận giá trị âm. | <code>print('str(t3)=', str(t3))</code> | str(t3)= 8:00:12 |

| | | |
|---------------------------------|---|--|
| <i>repr</i>
(<i>tdObj</i>) | Trả về string theo mẫu <code>datetime.timedelta(D[, S[, U]])</code> ; với <i>D</i> có thể nhận giá trị âm | <code>repr(t3)=
datetime.timed
elta(seconds=2
8812)</code> |
|---------------------------------|---|--|

- Sử dụng *timedelta*

- Công dụng 1: sử dụng hàm *timedelta* để cộng/trừ một số nguyên vào ngày của đối tượng thuộc *date class* hoặc *datetime class*

| Mã lệnh | Kết quả |
|---|---|
| <pre>from datetime import * homNay = datetime.now() homQua=homNay + timedelta(-1) ngayMai=homNay + timedelta(1) print('Hom qua:', homQua) print('Hom nay:', homNay) print('Ngày mai:', ngayMai)</pre> | <pre>Hom qua: 2020-02-28 14:42:06.700622 Hom nay: 2020-02-29 14:42:06.700622 Ngày mai: 2020-03-01 14:42:06.700622</pre> |

- Công dụng 2: tính khoảng thời gian chênh lệch giữa 2 đối tượng thuộc *date class* hoặc *datetime class*

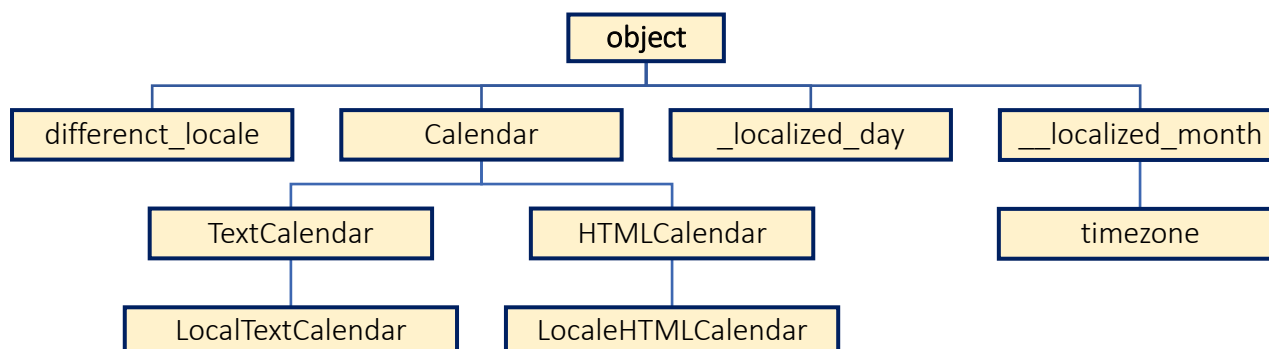
| Mã lệnh | Kết quả |
|--|------------------------|
| <pre>from datetime import datetime, date t1 = date(year = 2019, month = 10, day = 23) t2 = date(year = 2018, month = 10, day = 23) t3 = t1 - t2 print(t3)</pre> | 365 days,
0:00:00 |
| <pre>t4 = datetime(year=2018, month=10, day=23, hour=7, minute=10, second=46) t5 = datetime(year=2019, month=10, day=23, hour=9, minute=8, second=34) t6 = t4 - t5 print(t6)</pre> | -366 days,
22:02:12 |

- Công dụng 3: tính khoảng thời gian chênh lệch giữa 2 đối tượng thuộc *timedelta class*

| Mã lệnh | Kết quả |
|--|----------------------|
| <pre>from datetime import timedelta t1 = timedelta(weeks = 2, days = 5, hours = 1, seconds = 23) t2 = timedelta(days=4, hours=12, minutes=34, seconds=56) t3 = t1 - t2 print(t3)</pre> | 14 days,
12:25:27 |

3.5.4.3. calendar module

- *Module* này cung cấp các hàm, và một vài class liên quan tới lịch, hỗ trợ tạo ra hình ảnh của bộ lịch dưới dạng *text*, *html*, và *exception* có liên quan.



- Một số phương thức trong *module calendar*:

(i)- `calendar.isleap (year)`

- Kết quả trả về *True* nếu là năm nhuận, ngược lại trả về *False*

Ví du 3.63

| Mã lệnh | Kết quả |
|--|---------------|
| import calendar | 2019 is leap: |
| print('2019 is leap: ', calendar.isleap(2019)) | False |
| print('2020 is leap: ', calendar.isleap(2020)) | 2020 is leap: |
| | True |

(ii)- `calendar.monthcalendar(year, month)`

- Trả về danh sách các ngày trong tháng, năm được chọn, chia theo từng tuần, với:
 - Giá trị 0 trong kết quả là ngày không nằm trong tháng.
 - Ngày đầu tiên trong mỗi cặp ngoặc vuông là ngày thứ Hai.
- Ví dụ 3.64

| Mã lệnh | Kết quả |
|---|--|
| <pre>import calendar print(calendar.monthcalendar(2019, 7))</pre> | <pre>[[1, 2, 3, 4, 5, 6, 7], [8, 9, 10, 11, 12, 13, 14], [15, 16, 17, 18, 19, 20, 21], [22, 23, 24, 25, 26, 27, 28], [29, 30, 31, 0, 0, 0, 0]]</pre> |

(iii)- `calendar.weekday(year, month, day)`

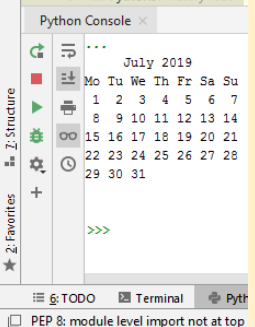
- Trả về số thứ tự của ngày/tháng/năm; với giá trị 0 là Monday, 1 là Tuesday, ...
- Ví dụ 3.65

| Mã lệnh | Kết quả |
|---|-------------------------------------|
| <pre>import calendar Thu=['Thu Hai','Thu Ba','Thu Tu','Thu Nam','Thu Sau', 'Thu Bay','Chu nhat'] print('Ngày 23 Oct 2019 là ngày', Thu[calendar.weekday(2019,10,23)])</pre> | <p>Ngày 23 Oct 2019 là ngày Thu</p> |

(iv)- **calendar.month(year, month)**

- Kết quả là lịch của tháng/năm tương ứng có thể in ra được bằng lệnh print

Ví dụ 3.66

| Mã lệnh | Kết quả |
|---|---|
| <pre>import calendar print (calendar.month(2019,7))</pre> |  |

(v)- **calendar.monthrange(year, month)**

- Trả về 2 phần tử số:
 - Phần tử đầu*: cho biết là thứ của ngày đầu tiên trong tháng (0 tương ứng với thứ Hai, 1 tương ứng với thứ Ba, ...)
 - Phần tử sau*: cho biết số ngày có trong tháng.

Ví dụ 3.67

| Mã lệnh | Kết quả |
|--|---------|
| <pre>import calendar print (calendar.monthrange(2019,7))</pre> | (0, 31) |

Giải thích: ngày đầu tiên của tháng 7/2019 là ngày Thứ Hai
(tương ứng với giá trị 0) và tháng 7 có 31 ngày

3.5.4.4. *time module*

- Ticks**: Trong Python, khoảng thời gian giữa thời điểm 12h sáng ngày 1 tháng 1 năm 1970 với thời điểm hiện tại được tính bằng số giây (*seconds*). Khoảng thời gian đó được gọi là *Ticks*.
- time module*: chỉ bao gồm các hàm, và các hằng số liên quan tới ngày tháng và thời gian, không có class nào được định nghĩa trong module này.
- Struct *time***: (trong module *time*) gồm các thành phần sau:

| Thành phần | Miền giá trị | Ví dụ | Kết quả có thể thu được |
|------------|-----------------|--------------|--|
| tm_year | | tm_year=2019 | January 1, 1970 to Mon
Jul 8 16:03:49 2019

Hoặc

2019-Jul 8 16:03:49 |
| tm_mon | [1,12] | tm_mon=7 | |
| tm_mday | [1,31] | tm_mday=8 | |
| tm_hour | [0,23] | tm_hour=15 | |
| tm_min | [0,59] | tm_min=54 | |
| tm_sec | [0,61] (*) | tm_sec=43 | |
| tm_wday | [0, 6] (**) | tm_wday=0 | |
| tm_yday | [1,366] | tm_yday=189 | |
| tm_isdst | 0 hoặc 1 hay -1 | tm_isdst=0 | |

(*): là giây nhuận (*double leap seconds*). Rất hiếm gặp.

(**): Thứ hai được tính là 0 và tính tiếp tục cho các thứ khác trong tuần.

- Các hàm trong *time module*:

- (i)- **time.time()**: trả về số giây (*seconds*) tính từ thời điểm 12h sáng ngày 1 tháng 1 năm 1970 tới hiện tại. Kết quả trả về có kiểu là số chấm động (*floating point*)

(ii)- **time.localtime(time.time())**: trả về thời gian hiện tại của hệ thống dưới dạng các thành phần của *datetime*.

(iii)- **time.sleep(secs_num)**: dừng chương trình thực thi trong số giây đã cho là *secs_num*.

(iv)- **time.asctime(time.localtime(time.time()))**: trả về chuỗi thời gian theo định dạng:

Thứ tháng ngày giờ:phút:giây năm

- Ví dụ 3.68

| Mã lệnh | Kết quả |
|--|--|
| <pre>import time print("Ham time: ", time.time())</pre> | Ham time: 1562576083.0464394 |
| <pre>print("Ham localtime: ", time.localtime(time.time()))</pre> | Ham localtime: time.struct_time(tm_year=2019, tm_mon=7, tm_mday=8, tm_hour=15, tm_min=54, tm_sec=43, tm_wday=0, tm_yday=189, tm_isdst=0) |
| <pre>print("Ham asctime: ", time.asctime(time.localtime(time.time())))</pre> | Ham asctime: Mon Jul 8 15:54:43 2019 |
| <pre>print("Number of ticks since 12:00am, January 1, 1970 to ", time.asctime(time.localtime (time.time()))), "= ", time.time())</pre> | Number of ticks since 12:00am, January 1, 1970 to Mon Jul 8 16:03:49 2019 = 1562576629.0216188 |

- Ví dụ 3.69: tính thời gian thực hiện của chương trình tính tổng các số từ 1 đến 5 triệu

| Mã lệnh | Kết quả |
|--|--|
| <pre>import time def sum_of_n_numbers(n): start = time.time() s = 0 for i in range(1,n+1): s = s + i finish = time.time() return s,finish-start n = 5000000 tong, thoigian = sum_of_n_numbers(n) print("\nTổng các số từ 1 đến",n,"là",tong, "và thời gian thực hiện tính toán là:" , thoigian, "giây")</pre> | Tổng các số từ 1 đến 5000000 là 12500002500000 và thời gian thực hiện tính toán là: 0.530475378036499 giây |

- Ví dụ 3.70: tương tự ví dụ 3.40 nhưng sử dụng hàm *default_timer* trong module *time*.

| Mã lệnh | Kết quả |
|--|---------|
| <pre>from timeit import default_timer def sum_of_n_numbers(n): start = default_timer() s = 0 for i in range(1,n+1): s = s + i finish = default_timer() return s,finish-start</pre> | |

| | |
|---|---|
| <pre>n = 5000000 tong, thoigian=sum_of_n_numbers(n) print("\nTổng các số từ 1 đến",n,"là",tong, "và thời gian thực hiện tính toán là:", thoigian, "giây")</pre> | Tổng các số từ 1 đến 5000000 là 12500002500000 và thời gian thực hiện tính toán là: 0.4951079079999998 giây |
|---|---|

3.5.4.5. locale module

Module này chứa các hàm sử dụng để định dạng (*format*), hoặc phân tích (*parse*) ngày tháng và thời gian dựa trên *locale* (vùng miền, địa lý).

3.5.5. Module sys

- Module *sys* cung cấp các hàm và các biến được sử dụng để thao tác các phần khác nhau của môi trường chạy *Python*. Module này cho phép chúng ta truy cập các tham số và chức năng cụ thể của hệ thống.
- Một số hàm phổ biến được dùng trong *module sys*:

| Hàm | Công dụng |
|-----------------------------------|---|
| <code>sys.argv</code> | Hàm trả về một danh sách các đối số dòng lệnh được truyền cho tập lệnh <i>Python</i> . Tên của tập lệnh luôn là mục ở chỉ số 0 và phần còn lại của các đối số được lưu trữ tại các chỉ mục tiếp theo. |
| <code>sys.base_exec_prefix</code> | Hàm cung cấp một cách hiệu quả cho cùng giá trị như <i>exec_prefix</i> . Nếu không chạy môi trường ảo, giá trị sẽ giữ nguyên. |
| <code>sys.base_prefix</code> | Hàm được thiết lập trong quá trình khởi động <i>Python</i> , trước khi <i>site.py</i> chạy, có cùng giá trị với tiền tố. |
| <code>sys.byteorder</code> | Hàm là một dấu hiệu của <i>byteorder</i> gốc cung cấp một cách hiệu quả để làm một cái gì đó. |
| <code>sys.exit([prompt])</code> | Hàm được sử dụng để thoát khỏi <i>console</i> trong <i>Python</i> hoặc dấu nhắc lệnh và cũng được sử dụng để thoát khỏi chương trình trong trường hợp ngoại lệ.
Lưu ý rằng khi <i>exit</i> được gọi, sẽ phát sinh ra ngoại lệ <i>SystemExit</i> , cho phép các chức năng dọn dẹp hoạt động của các khối lệnh trong <i>except</i> và <i>finally</i> . |
| <code>sys.executable</code> | Giá trị trả về của hàm là đường dẫn tuyệt đối đến trình thông dịch <i>Python</i> . Hàm rất hữu ích để biết nơi <i>Python</i> được cài đặt trên máy của người khác. |
| <code>sys.float_info</code> | Lấy thông tin về kiểu dữ liệu <i>float</i> |
| <code>sys.int_info</code> | Lấy thông tin về kiểu dữ liệu <i>int</i> |
| <code>sys.getrefcount</code> | Hàm trả về số tham chiếu của một đối tượng. |
| <code>sys.getsizeof</code> | Trả về kích thước của 1 đối tượng |
| <code>sys.maxsize</code> | Hàm trả về số nguyên lớn nhất của một biến. |
| <code>sys.modules</code> | Hàm cung cấp tên của các <i>module Python</i> hiện có đã <i>import</i> . |
| <code>sys.path</code> | Hàm hiển thị bộ <i>PYTHONPATH</i> trong hệ thống hiện tại. Đây là một biến môi trường là một đường dẫn tìm kiếm cho tất cả các <i>module Python</i> . |
| <code>sys.platform</code> | Giá trị trả về của hàm được sử dụng để xác định nền tảng mà chúng ta đang làm việc. |
| <code>sys.stdin</code> | Hàm là một đối tượng chứa các giá trị gốc của <i>stdin</i> khi bắt đầu chương trình và được sử dụng trong quá trình hoàn thiện. Hàm có thể khôi phục các tập tin. |

- Ví dụ 3.71: kiểm tra version và thông tin về version của Python đang dùng

| Mã lệnh | Kết quả |
|--|--|
| <pre>import sys print('Python version: ', sys.version) print('Version info: ', sys.version_info)</pre> | <pre>Python version: 3.7.2rc1 (tags/v3.7.2rc1:75a402a217, Dec 11 2018, 23:05:39) [MSC v.1916 64 bit (AMD64)] Version info: sys.version_info(major=3, minor=7, micro=2, releaselevel='candidate', serial=1)</pre> |

- Ví dụ 3.72: lấy kích thước tính bằng bytes của 1 đối tượng

| Mã lệnh | Kết quả |
|---|--|
| <pre>import sys str1 = "one" str2 = "four" str3 = "three" num=1 flag=True print("Memory size of '"+str1+"' = " +str(sys.getsizeof(str1))+ " bytes") print("Memory size of '"+str2+"' = " +str(sys.getsizeof(str2))+ " bytes") print("Memory size of '"+str3+"' = " +str(sys.getsizeof(str3))+ " bytes") print("Memory size of '"+str(num)+"' = " +str(sys.getsizeof(num))+ " bytes") if flag: var='True' else: var='False' print("Memory size of '"+var+"' = " +str(sys.getsizeof(flag))+ " bytes")</pre> | <pre>Memory size of 'one' = 52 bytes Memory size of 'four' = 53 bytes Memory size of 'three' = 54 bytes Memory size of '1' = 28 bytes Memory size of 'True' = 28 bytes</pre> |

- Ví dụ 3.73: lấy thông tin về kiểu dữ liệu float, int và lấy kích thước tối đa của kiểu integer

| Mã lệnh | Kết quả |
|--|---|
| <pre>import sys print("Float value information: ", sys.float_info)</pre> | <pre>Float value information: sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308, min=2.2250738585072014e-308, min_exp=-1021, min_10_exp=-307, dig=15, mant_dig=53, epsilon=2.220446049250313e-16, radix=2, rounds=1)</pre> |
| <pre>print("\nInteger value information: ", sys.int_info)</pre> | <pre>Integer value information: sys.int_info(bits_per_digit=30, sizeof_digit=4)</pre> |
| <pre>print("\nMaximum size of an integer: ", sys.maxsize)</pre> | <pre>Maximum size of an integer: 9223372036854775807</pre> |

3.5.6. Module struct

- Module này thực hiện chuyển đổi giữa các giá trị Python và các cấu trúc C được biểu diễn dưới dạng các đối tượng byte Python. Chuỗi định dạng là cơ chế được sử dụng để chỉ định bố cục dữ liệu khi đóng gói và giải nén dữ liệu. Module struct chỉ được đưa vào từ Python 3.x, do đó không thể dùng trên 2.x.
- Ví dụ 3.74: sử dụng phương thức `calcsizes` trong đoạn chương trình sau để biết máy tính đang chạy hệ điều hành 32 hay 64 bit.

```
import struct
print(struct.calcsize("P") * 8)
```

3.5.7. Module platform

Hỗ trợ truy cập thông tin về hệ thống đang sử dụng.

- `platform.system()` : trả về tên hệ điều hành đang dùng
- `platform.release()` : version của hệ điều hành đang dùng
- `platform.node()` : trả về hostname của server
- `platform.version()` : trả về thông tin chi tiết của version đang dùng
- `platform.machine()` : trả về tên phần cứng dùng cho máy tính như i86, AMD64,...
- `platform.processor()` : trả về định danh của CPU. Ví dụ: Intel64 Family 6 Model 37 Stepping 5, GenuineIntel

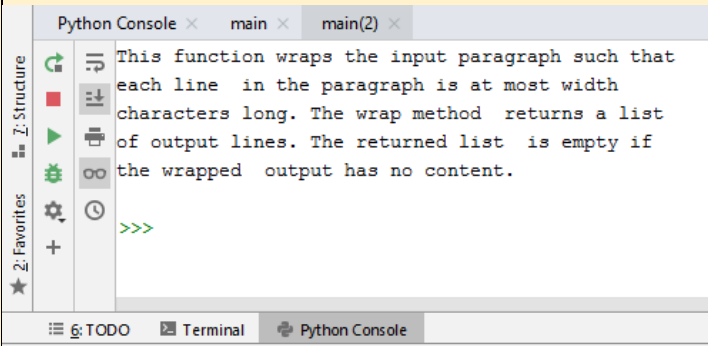
3.5.8. Module textwrap

- Module `textwrap` được sử dụng để gói và định dạng văn bản thuần túy. Module này cung cấp định dạng của văn bản bằng cách điều chỉnh ngắt dòng trong đoạn đầu vào.
- Các thuộc tính phiên bản `TextWrapper` (và các đối số từ khóa cho hàm tạo) như sau:
 - width**: quy định chiều ngang tối đa của đoạn văn bản được hiển thị. Giá trị mặc định là 70.
 - expand_tabs**: Giá trị mặc định là `True`. Khi giá trị bằng `True`, tất cả các ký tự tab trong đoạn văn bản được mở rộng thành khoảng trắng.
 - tabsize**: Giá trị mặc định là 8. Nếu giá trị của `expand_tabs` là `True`, phương thức này mở rộng tất cả các ký tự tab trong văn bản thành 0 hoặc nhiều khoảng trắng, tùy thuộc vào cột văn bản hiện tại và kích thước tab đã cho.
 - replace_whitespace**: Giá trị mặc định là `True`. Nếu giá trị là `True`, sau khi mở rộng tab nhưng trước khi cuộn xuống dòng (wrapping), phương thức `wrap()` sẽ thay thế các ký tự khoảng trắng liên tiếp nhau bằng duy nhất một khoảng trắng. Các ký tự khoảng trắng này được dùng để thay thế cho: tab, dòng mới (newline), tab dọc (vertical tab), formfeed và trả về `('\t\n\v\f\r')`.
 - drop_whitespace**: Giá trị mặc định là `True`. Khi giá trị là `True`, các khoảng trắng ở đầu và cuối của mỗi dòng (sau khi wrap nhưng trước khi thụt lề) sẽ bị hủy.
 - initial_indent**: Giá trị mặc định là chuỗi rỗng (`""`). Phương thức này chuẩn bị chuỗi đã cho vào đầu dòng văn bản đã được wrap trước khi xuất.
 - subsequent_indent**: Giá trị mặc định là chuỗi rỗng (`""`). Phương thức này thực hiện tương tự như phương thức `initial_indent` cho tất cả các dòng ngoại trừ dòng đầu tiên.

- (viii)- *placeholder*: Giá trị mặc định là '`...`'. Phương thức này nối thêm chuỗi ở cuối văn bản đầu ra nếu chuỗi bị cắt ngắn.
- (ix)- *max_lines*: Giá trị mặc định là **None**. Khi giá trị không phải là None, văn bản đầu ra chứa tối đa các dòng *max_lines* với *placeholder* ở cuối đầu ra.
- (x)- *break_long_words*: Giá trị mặc định là True. Nếu True, các từ làm cho văn bản dài hơn chiều rộng quy định sẽ được tách để phù hợp với mọi dòng khác. Ngược lại, khi là False, các từ dài sẽ được đặt trên một dòng, để giảm thiểu số lượng vượt quá chiều rộng.
- (xi)- *break_on_hyphens*: Giá trị mặc định là True. Khi là True, wrap xảy ra trên khoảng trắng và ngay sau dấu gạch nối trong từ ghép. Nếu giá trị bằng False, ngắt dòng chỉ xảy ra trên các khoảng trắng, nhưng khi đó cần đặt *break_long_words* thành False.

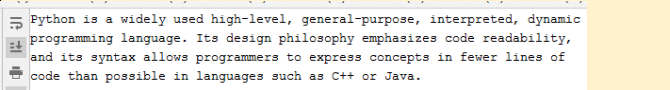
3.5.8.1. *Wrap()*

- Hàm này giúp hiển thị đoạn text sao cho mỗi dòng trong đoạn có độ dài tối đa width ký tự.
- Cú pháp:
`textwrap.wrap (text[, attributes])`
- Phương thức wrap trả về một danh sách các dòng đầu ra sao cho kích thước theo chiều ngang do thuộc tính width quy định (mặc định = 70).
- Ví dụ 3.75:

| Mã lệnh | Kết quả |
|--|--|
| <pre>import textwrap value = """This function wraps the input paragraph such that each line in the paragraph is at most width characters long. The wrap method returns a list of output lines. The returned list is empty if the wrapped output has no content.""" # Wrap this text. wrapper = textwrap.TextWrapper(width=50) word_list = wrapper.wrap(text=value) # Print each line. for element in word_list: print(element)</pre> |  |

3.5.8.2. *fill()*

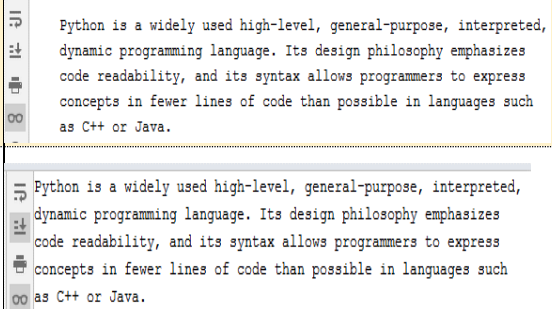
- Hàm *fill()* hoạt động tương tự như *textwrap.wrap*, hàm *fill* bao bọc đoạn văn bản đầu vào trong văn bản và trả về một chuỗi đơn chứa đoạn được bao bọc sao cho vừa với thuộc tính *width*.
- Ví dụ 3.76: dòng đầu không thụt đầu dòng, các dòng còn lại thụt vào 4 khoảng trắng.

| Mã lệnh | Kết quả |
|---|--|
| <pre>import textwrap sampleText = ''' Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax</pre> |  |

| | |
|--|---|
| allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java.
'''
print(sampleText)
text1 =
textwrap.dedent(sampleText).strip()
print(textwrap.fill(text1,
initial_indent='',
subsequent_indent=' ' * 4,
width=80,)) | |
| | Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. |

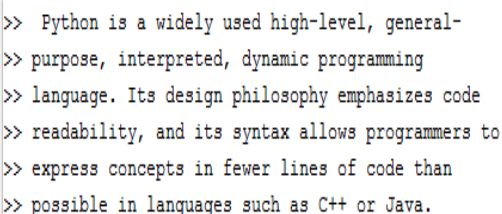
3.5.8.3. dedent()

- Xóa thụt lề hiện có khỏi tất cả các dòng trong một văn bản nhất định.
- Ví dụ 3.77:

| Mã lệnh | Kết quả |
|--|---|
| import textwrap
sampleText = '''
Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java.
'''
print(sampleText)
textWithoutIndentation =
textwrap.dedent(sampleText)
print(textWithoutIndentation) |  |

3.5.8.4. indent()

- Hàm thực hiện thêm chuỗi subS (do người dùng chỉ định) vào đầu mỗi dòng trong đoạn text đã được wrap..
- Ví dụ 3.78:

| Mã lệnh | Kết quả |
|--|--|
| import textwrap
sampleText = '''
Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. '''
textWithoutIndentation =
textwrap.dedent(sampleText)
wrapped =
textwrap.fill(textWithoutIndentation, width=50)
final_result = textwrap.indent(wrapped, '>>>')
print(final_result) |  |

3.5.9. Module itertools

- *itertools module* rất hữu ích trong việc tạo các trình vòng lặp hiệu quả và ngăn chặn các rủi ro khi dùng mã lệnh của người lập trình.

- Đây là đối tượng biểu diễn dòng dữ liệu và trả về từng phần tử một. *Iterator* rất hữu dụng khi thao tác với *lists*, *tuples*, *strings*, ...

3.5.9.1. *chain()*

- Giúp truy xuất từng phần tử riêng lẻ trong 1 hoặc nhiều đối tượng dạng danh sách mà không quan tâm đến kiểu dữ liệu của các thành phần
- Ví dụ 3.79:

| Mã lệnh | Kết quả |
|---|--|
| <pre>from itertools import * list1=[12.3, 25, True] list2=['C#', 'Python', 'Java'] for value in chain(list1, list2): print(value,end=' ')</pre> | <pre>12.3 25 True C# Python Java</pre> |

- Ví dụ 3.80: phối hợp hàm *sum()* và *chain()*

| Mã lệnh | Kết quả |
|--|----------------------|
| <pre>from itertools import * list1=[1,3,5] list2=[2, 4, 6, 8.9] print('SUM=',sum(chain(list1, list2)))</pre> | <pre>SUM= 29.9</pre> |

- Ví dụ 3.81: xây dựng hàm *mychain* với chức năng tương tự như hàm *chain()*

| Mã lệnh | Kết quả |
|---|--|
| <pre>from itertools import * list1=[1,3,5] list2=[2, 4, 6, 8.9] print('SUM=',sum(chain(list1, list2)))</pre> | <pre>SUM= 29.9</pre> |
| <pre>from itertools import * def mychain(*iterables): list3=[] for it in iterables: for element in it: list3.append(element) return list3 list1=[0,1,2] str1='ABCD' print(mychain(list1,str1)) print(list(chain(list1,str1)))</pre> | <pre>[0, 1, 2, 'A', 'B', 'C', 'D'] [0, 1, 2, 'A', 'B', 'C', 'D']</pre> |

3.5.9.2. *islice()*

- Hàm này trả về một *iterator* theo cách thức tương tự như *slice*
- Cú pháp `islice(iterable, start[, stop[, step]]`.
Lưu ý: *start*, *stop*, *step* không được nhận giá trị âm.
- Ví dụ 3.82:

| Mã lệnh | Kết quả |
|---|--|
| <pre>from itertools import * list1=[0,1,2,3,4,5,6,7,8,9] print('list:',list1)</pre> | <pre>list:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]</pre> |

| | |
|--|---|
| <pre>print('islice(list1, 3)=> chỉ có 1 tham số (tsố) về số lượng',end=' ') for num in islice(list1, 3): print(num,end=' ')</pre> | <pre>islice(list1, 3)=> chỉ có 1 tham số (tsố) về số lượng 0 1 2</pre> |
| <pre>print('\nislice(list1, 3, 7)=> tsố 1 là START, tsố 2 là END:',end=' ') for num in islice(list1, 3, 7): print(num,end=' ')</pre> | <pre>islice(list1, 3, 7)=> tsố 1 là START, tsố 2 là END: 3 4 5 6</pre> |
| <pre>print('\nislice(list1, 3, 7, 2)=> tsố 1 là START, tsố 2 là END, tsố 3 là STEP:',end=' ') for num in islice(list1, 3, 7, 2): print(num,end=' ')</pre> | <pre>islice(list1, 3, 7, 2)=> tsố 1 là START, tsố 2 là END, tsố 3 là STEP: 3 5</pre> |

3.5.9.3. *itertools.product()*

- Hàm này thực hiện việc tìm tất cả các tổ hợp có được từ tham số đầu vào iterables. Hàm này thực hiện tương đương với các vòng lặp lồng nhau.
- Cú pháp:

`iterator = itertools.product(*iterables, repeat=1)`

Trong đó:

- ***Iterables***: các tập hợp dữ liệu input;
- ***repeat***: được sử dụng để ghép các iterables với chính nó *repeat* lần, mặc định *repeat* = 1.

- Ví dụ 3.83

| Mã lệnh | Kết quả |
|--|---|
| <pre>from itertools import * list1=[0,1] str1='ABC' # Tạo tổ hợp từ 1 danh sách print (list(product(list1,repeat=len(list1)))) # Tạo tổ hợp từ nhiều danh sách resultlist=list(product(list1, str1)) print(resultlist)</pre> | <pre>[(0,0), (0,1), (1,0), (1,1)]</pre> |
| <pre>'''đoạn lệnh sau tương đương với dùng product để tạo tổ hợp từ nhiều danh sách''' resultlist =list(((x,y) for x in list1 for y in str1)) print(resultlist)</pre> | <pre>[(0, 'A'), (0, 'B'), (0, 'C'), (1, 'A'), (1, 'B'), (1, 'C')]</pre> |
| <pre>'''Tạo tổ hợp từ các danh sách con trong 1 danh sách lớn''' resultlist = [[1,2], 'AB', [3,4,5]] print (list(product(*resultlist)))</pre> | <pre>[(1,'A',3), (1,'A',4), (1,'A',5), (1,'B',3), (1,'B',4), (1,'B',5), (2,'A',3), (2,'A',4), (2,'A',5), (2,'B',3), (2,'B',4), (2,'B',5)]</pre> |

3.5.9.4. *itertools.imap* & *itertools.starmap*

- Các hàm như *map*, *filter* và *zip* được coi là lỗi thời trong *Python*, do đó hàm *map* đã được thay thế bằng *itertools.imap*.
- Sự khác biệt chính giữa *imap* và *starmap* là
 - *imap*: tham số đầu vào là một danh sách các mục như [1, 2, 3, 4, 5].

- *starmap*: tham số đầu vào là một danh sách các *iterables* (có thể là một *list* hoặc một *tuple*, hoặc bất cứ đối tượng dạng danh sách khác) như [(1, 2, 3, 4, 5), (6, 7, 8, 9, 10)]

3.5.9.5. *itertools.combinations*

- Tạo các tổ hợp từ một tập hợp *iterable*
- Cú pháp:

`iterator = itertools.combinations(iterables, r)`

Trong đó:

- ***Iterables***: các tập hợp dữ liệu input;
- ***r***: là tham số bắt buộc, cho biết số lượng phần tử có trong tập hợp.

- Ví dụ 3.84

| Mã lệnh | Kết quả |
|---|---|
| <pre>from itertools import combinations print(list(combinations("ABC", 2)))</pre> | <pre>[('A', 'B'), ('A', 'C'), ('B', 'C')]</pre> |

3.5.9.6. *itertools.combinations_with_replacement()*

- Tạo các tổ hợp trùng lặp từ một tập hợp *iterable*
- Cú pháp:

`iterator = itertools.combinations_with_replacement(iterables, r)`

Trong đó:

- ***Iterables***: các tập hợp dữ liệu input;
- ***r***: là tham số bắt buộc, cho biết số lượng phần tử có trong tập hợp.

- Ví dụ 3.85

| Mã lệnh | Kết quả |
|---|---|
| <pre>from itertools import * print(list(combinations_with_replacement("ABC", 2)))</pre> | <pre>[('A', 'A'), ('A', 'B'), ('A', 'C'), ('B', 'B'), ('B', 'C'), ('C', 'C')]</pre> |

3.5.9.7. *itertools.compress()*

- Chọn ra các phần tử của tập hợp được đánh dấu trước.
- Cú pháp: **`compressObject = itertools.compress(data, selector)`**

Trong đó:

- ***Iterables***: các tập hợp dữ liệu input;
- ***selector***: là 1 list đánh dấu các phần tử dữ liệu được chọn. Quy ước 1 là chọn, 0 là không chọn.

Ví dụ 3.86

| Mã lệnh | Kết quả |
|---|----------------------------|
| <pre>from itertools import compress print(list(compress("ABCD", [1, 0, 1, 1])))</pre> | <pre>['A', 'C', 'D']</pre> |
| <pre>print(list(compress("ABCD", [1, 0, 1])))</pre> | <pre>['A', 'C']</pre> |

3.5.9.8. *itertools.groupby()*

- Được sử dụng để gom nhóm dữ liệu theo keyword.
- Cú pháp: **`iterator = itertools.groupby(iterable[, key])`**

Trong đó:

- **Iterables**: các tập hợp dữ liệu input;
- **key**: là phần tử được chọn để gom nhóm.

- Ví dụ 3.87

| Mã lệnh | Kết quả |
|--|--|
| <pre>from itertools import groupby instruments = [("gun", "police"), ("pen", "student"), ("pencil", "student"), ("ruler", "student"), ("screwdrivers", "engineer"), ("pliers", "engineer")] for key, group in groupby(instruments, lambda x:x[1]): print(key) print(list(group))</pre> | <pre>police [('gun', 'police')] student [('pen', 'student'), ('pencil', 'student'), ('ruler', 'student')] engineer [('screwdrivers', 'engineer'), ('pliers', 'engineer')]</pre> |

3.5.9.9. *itertools.permutations()*

- Được dùng để thực hiện phép chỉnh hợp từ một tập hợp.
- Cú pháp:

iterator = itertools.permutations(iterable, r=None)

Trong đó:

- **Iterables**: các tập hợp dữ liệu input;
- **r**: là số phần tử được chọn ra từ tập **Iterables**. Mặc định **r=None=len(Iterables)**.

- Ví dụ 3.88

| Mã lệnh | Kết quả |
|---|--|
| <pre>from itertools import permutations print(list(permutations("ABC",2))) print(list(permutations("ABC")))</pre> | <pre>[('A', 'B'), ('A', 'C'), ('B', 'A'), ('B', 'C'), ('C', 'A'), ('C', 'B')] [('A', 'B', 'C'), ('A', 'C', 'B'), ('B', 'A', 'C'), ('B', 'C', 'A'), ('C', 'A', 'B'), ('C', 'B', 'A')]</pre> |

CÁC ĐỐI TƯỢNG DẠNG DANH SÁCH TRONG PYTHON

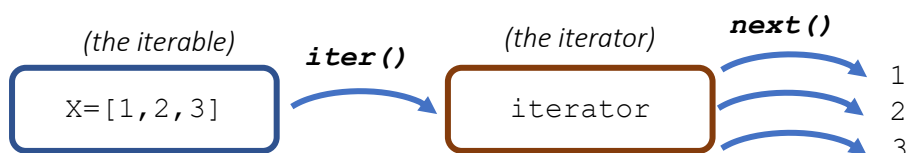
4.1. Iterator trong Python

4.1.1. Giới thiệu

- *Iterator* (hay *sequences types*) là các đối tượng gồm nhiều phần tử, *Iterator* cho phép ta truy cập đến từng phần tử và hành động này có thể được lặp đi lặp lại.

Về mặt kỹ thuật, *Iterator* trong *Python* phải thực hiện hai phương thức đặc biệt là `__iter__()` và `__next__()`, gọi chung là giao thức *iterator* (*Iterator Protocol*).

- Phương thức `__iter__` trả về chính đối tượng *iterator*. Phương thức này được yêu cầu cài đặt cho cả đối tượng "iterable" và *iterator* để có thể sử dụng các câu lệnh `for` và `in`.
- Phương thức `__next__` trả về phần tử tiếp theo. Nếu không còn phần tử nào nữa thì sẽ có lỗi ngoại lệ *StopIteration* xảy ra.



- Trong *Python*, có 6 loại *Iterator* là *string*, *Unicode string*, *lists*, *tuple*, *buffers* và *xrange*.
- `iter()` là một hàm dựng sẵn trong *Python* nhận đầu vào là một đối tượng *iterable* và trả về kết quả là một *iterator*.

Ví dụ 4.1: sử dụng *iterable object* là 1 list để thu được 1 *iterator*. Duyệt và in các phần tử trong *iterator* ra màn hình.

| Mã lệnh | Kết quả |
|--|--|
| <pre>my_list = ['Sài gòn', 3.14, 18, (1,3,5), [7, 'Hà Nội']] # tạo ra một iterator bằng cách sử dụng iter() my_iter = iter(my_list) i=0 while i<len(my_list): print(next(my_iter)) i+=1 #nếu sử dụng tiếp lệnh này sẽ gây ra StopIteration exception print(next(my_iter))</pre> | <pre>Sài gòn 3.14 18 (1, 3, 5) [7, 'Hà Nội'] StopIteration</pre> |

Ví dụ 4.2: tương tự như ví dụ trên nhưng dùng vòng lặp vô hạn, `try ... except` và *StopIteration exception* để duyệt *iterator*

| Mã lệnh | Kết quả |
|--|--------------------------------------|
| <pre># Khai báo một list my_list = ['Sài gòn', 3.14, 18, (1,3,5)] # lấy một iterator bằng cách sử dụng iter() my_iter = iter(my_list) i=0 while True: try: print(next(my_iter)) except StopIteration: # nếu xảy ra lỗi StopIteration => break ra khỏi while break</pre> | <pre>Sài gòn 3.14 18 (1, 3, 5)</pre> |

Lưu ý: khi tạo và duyệt *iterator* có thể dẫn đến một *Iterable* lặp vô hạn

- **Ví dụ 4.3:** tương tự như ví dụ 4.1 nhưng dùng vòng lặp vô hạn, `try ... except` và `StopIteration` exception để duyệt iterator

| Mã lệnh | Kết quả |
|--|---|
| <pre>my_iterator = iter(int,1) while True: try: print(next(my_iterator)) except StopIteration: break</pre> | Liên tục in ra các số 0 và chương trình chạy không dừng |

- Giải thích: Do hàm `int()` luôn trả về 0. Vì vậy, khi sử dụng hàm `iter()` để tạo iterator bằng dòng lệnh `my_iterator = iter(int,1)` sẽ trả về một iterator lặp cho đến khi giá trị trong đó bằng 1. Điều này không bao giờ xảy ra và đây chính là iterator lặp vô hạn.

4.1.2. Duyệt iterable

4.1.2.1. Dùng `enumerate` duyệt qua cả index và giá trị

- **Ví dụ 4.4**

| Mã lệnh | Kết quả |
|---|---|
| <pre>s= 'Hello world !' L=s.split() for i, x in enumerate(L): print('{}: {}'.format(i, x))</pre> | <pre>0: world 1: ! 2: Hello</pre> |
| <pre>T=tuple(s.split()) for i, x in enumerate(T): print('{}: {}'.format(i, x))</pre> | <pre>0: world 1: ! 2: Hello</pre> |
| <pre>S=set(s.split()) for i, x in enumerate(S): print('{}: {}'.format(i, x))</pre> | <pre>0: world 1: ! 2: Hello</pre> |
| <pre>d = a = {'one': 1, 'two': 2, 'three': 3, 'four': 4} for i, x in enumerate(a): print('{}: {}'.format(i, x))</pre> | <pre>0: one 1: two 2: three 3: four</pre> |

4.1.3. So sánh các đối tượng dạng iterator trong Python

| Iterator | Có thứ tự | Sử dụng chỉ mục | Có thể thay đổi | Cho phép dữ liệu trùng |
|------------|-----------|-----------------|-----------------|------------------------|
| List | Có | Có | Có | Có |
| Tuple | Có | Có | Không | Có |
| Dictionary | Không | Có | Có | Không |
| Set | Không | Không | Có | Không |
| String | Có | Có | Có | Có |

4.2. List

4.2.1. Giới thiệu

- Lists là tập hợp có thứ tự các phần tử (*elements*) thuộc nhiều kiểu dữ liệu khác nhau (như *strings*, *integers*, *objects*, *other lists*, ...). Do đó trong Python, key của một mảng có thể vừa là số, vừa là chuỗi (*associated array*). Tuy nhiên, nếu các phần tử đều thuộc cùng 1 kiểu dữ liệu sẽ giúp dễ xử lý hơn.
- List trong Python là cấu trúc mảng và các phần tử có index thứ tự tính từ 0 và index của phần tử cuối là số lượng phần tử trong list-1.

- Trong Python, khi muốn tạo một danh sách có key là chuỗi thường thì người lập trình sẽ sử dụng cấu trúc Dictionary (sẽ được đề cập trong các phần sau).
- Một List được khai báo như mảng trong JSON.

4.2.2. Khai báo

- Sử dụng cặp ngoặc vuông `[]` để khai báo một list.
- Khai báo list:
 - Khai báo list rỗng: `list1 = []`
 - Khai báo và gán giá trị


```
list2 = [9, 2.17, "Sai gon", "X"]
list3 = ["Ha noi", "Hue", "Sai gon"]
list4 = [7, 2, 5, 8, 1, 9]
```
 - Khai báo và gán giá trị dựa vào hàm `range(Start, Stop[, Step])`

Ví dụ 4.5: `list5=range(2,11)`

 - Kết quả sẽ tạo ra 1 list với các giá trị như sau: `[2, 3, 4, 5, 6, 7, 8, 9, 10]`
 - Lưu ý:
 - Số lượng phần tử có trong `list = Stop - Start` (trong ví dụ đang xét là `11-2=9`).
 - Giá trị lớn nhất trong list sẽ là `max-1`.
 - Khai báo và gán giá trị dựa vào dấu hoa thị (*)
 - Toán tử * đại diện cho "phần còn lại"
 - Ví dụ 4.6

| Mã lệnh | Kết quả |
|--|--|
| <code>*a, b, c = [1, 2, 3, 4, 5]</code>
<code>print ('a=',a, '; b=',b, '; c=',c)</code> | <code>a=[1, 2, 3]; b=4; c=5</code> |
| <code>a, *b, c = (1, 2, 3, 4, 5)</code>
<code>print ('a=',a, '; b=',b, '; c=',c)</code> | <code>a=1; b=[2, 3, 4]; c=5</code> |
| <code>a, b, *c = {1, 2, 3, 4, 5}</code>
<code>print ('a=',a, '; b=',b, '; c=',c)</code> | <code>a=1; b=2 ; c=[3, 4, 5]</code> |
| <code>a, b, c = {1, 2, 3, 4, 5}</code>
<code>print ('a=',a, '; b=',b, '; c=',c)</code> | <code>ValueError: too many values to unpack (expected 3)</code>
<code>'''Lỗi vì số lượng biến bên vế trái và số lượng giá trị bên vế phải không bằng nhau'''</code> |

- Lưu ý: chỉ được dùng tối đa 1 toán tử * trong vế trái

4.2.3. Đếm số lượng phần tử có trong List

- Để biết được số lượng phần tử của 1 list, có thể sử dụng hàm `len(listName)`.
- Ví dụ 4.7

| Mã lệnh | Kết quả |
|---|---------|
| <code>lst=['Sai Gon','Hue','Ha Noi']</code>
<code>print(len(lst))</code> | 3 |

- Truy xuất từng phần tử của list bằng chỉ mục (index), với `0 <= index <= len(listName)`.

4.2.4. Xuất nội dung list ra màn hình

Ví dụ 4.8: cho `L = [5, 10, 15, 20, 25, 30, 35, 40]`. Có 2 cách chính để in List:

4.2.4.1. In toàn bộ list trong một lần

Gồm 3 cách như sau:

| Cách | Mã lệnh | Kết quả |
|------|--------------------------|--|
| 1a | <code>print(L)</code> | <code>[5, 10, 15, 20, 25, 30, 35, 40]</code> |
| 1b | <code>print(*L)</code> | <code>5 10 15 20 25 30 35 40</code> |
| 1c | <code>print(L[:])</code> | <code>[5, 10, 15, 20, 25, 30, 35, 40]</code> |

4.2.4.2. Duyệt lần lượt từng phần tử trong list để in ra màn hình

Sử dụng lệnh lặp `for` hoặc `while`.

| Cách | Mã lệnh | Kết quả |
|------|---|---|
| 2a | <code>#sử dụng for duyệt từng giá trị có trong list</code>
<code>for item in L:</code>
<code> print(item, end=',')</code> | <code>5, 10, 15, 20, 25,</code>
<code>30, 35, 40,</code> |
| 2b | <code>#sử dụng for và index để duyệt từng giá trị có</code>
<code>trong list</code>
<code>for index in range(len(L)):</code>
<code> print(L[index], end=' ')</code> | <code>5 10 15 20 25 30 35</code>
<code>40</code> |
| 2c | <code>#sử dụng while và index để duyệt từng giá trị</code>
<code>có trong list</code>
<code>index=0</code>
<code>while (index < len(L)):</code>
<code> print (L[index], end=',')</code>
<code> index = index+1</code> | <code>5, 10, 15, 20, 25,</code>
<code>30, 35, 40,</code> |

4.2.5. Thêm phần tử vào list

4.2.5.1. Thêm phần tử vào cuối list

- Phương thức `listName.append(element)` để thêm phần tử `element` vào cuối `list`.

Ví dụ 4.9

| Mã lệnh | Kết quả |
|--|-----------------------------|
| <code>numList = [5, 9, 6]</code>
<code>print (numList)</code> | <code>[5, 9, 6]</code> |
| <code>numList.append(100)</code>
<code>print (numList)</code> | <code>[5, 9, 6, 100]</code> |

4.2.5.2. Chèn phần tử vào vị trí được chỉ định trong list

- Phương thức `list.insert(index, element)` để chèn phần tử `element` vào vị trí `index` của `list`.

Ví dụ 4.10

| Mã lệnh | Kết quả |
|---|-----------------------------|
| <code>numList = [5, 9, 6]</code>
<code>print (numList)</code> | <code>[5, 9, 6]</code> |
| <code>numList.insert(1, 100)</code>
<code>print (numList)</code> | <code>[5, 100, 9, 6]</code> |

4.2.5.3. Nối 1 iterable object vào list hiện tại

4.2.5.3.1. Sử dụng phép cộng (+) hoặc nhân (*)

- Ví dụ 4.11

| Mã lệnh | Kết quả | Diễn giải |
|---|---------|-----------|
| <code>Lst1 = [5, 9, 7]</code>
<code>Lst2 = [6, 2]</code> | | |

| | | |
|--|------------------------------|------------------------------------|
| Lst3 = Lst1 + Lst2
print ('Lst3 =',Lst3) | Lst3 = [5, 9, 7, 6, 2] | Nối Lst1 trước
Lst2 |
| Lst4 = Lst2 + Lst1
print ('Lst4 =',Lst4) | Lst4 = [6, 2, 5, 9, 7] | Nối Lst2 trước
Lst1 |
| Lst5 = Lst1 + (Lst2 *2)
print ('Lst5 =',Lst5) | Lst5 = [5, 9, 7, 6, 2, 6, 2] | Nối 2 lần Lst2 vào
list kết quả |

4.2.5.3.2. Sử dụng phương thức extend

- Phương thức `listName.extend(iterableObject)` để nối các thành phần của 1 iterable khác vào list hiện tại.
- Ví dụ 4.12

| Mã lệnh | Kết quả |
|--|---|
| fruits = ['apple', 'banana', 'cherry']
cars = ['Ford', 'BMW', 'Volvo']
points = (1, 4, 5, 9) | |
| fruits.extend(cars)
print(fruits) | ['apple', 'banana', 'cherry', 'Ford', 'BMW', 'Volvo'] |
| cars.extend(points)
print(cars) | ['Ford', 'BMW', 'Volvo', 1, 4, 5, 9] |

4.2.6. Cập nhật giá trị cho phần tử trong list

- Cú pháp: `listName[index] = giá_trị`
- Ví dụ 4.13

| Mã lệnh | Kết quả | Diễn giải |
|--------------------------------|--|---|
| Lst = [5, 9, 6]
print (Lst) | [5, 9, 6] | |
| Lst[2]=100
print(Lst) | [5, 9, 100] | |
| Lst[3]=50
print(Lst) | IndexError: list assignment index out of range | Lst chỉ có index từ 0 đến 2, không có index=3 |

4.2.7. Kiểm tra sự tồn tại của một phần tử trong list

4.2.7.1. Kiểm tra dựa trên toán tử thành phần (Membership Operators)

- Sử dụng toán tử `in/not in`

Ví dụ 4.14

| Mã lệnh | Kết quả |
|--|---------|
| DanhSach = [5, 10, 15, 20]
print(10 in DanhSach) | True |
| x=3
print(x not in DanhSach) | True |
| DanhSach = ["Ty", "Suu", "Dan"]
name = input("Nhập tên cần tìm: ")
if name in DanhSach:
print(name,"có trong danh sách.")
else:
print(name,"KHÔNG có trong danh sách.") | |

4.2.7.2. Tìm vị trí của một giá trị trong list dựa trên phương thức index của list

Phương thức `list.index(element)` giúp tìm vị trí (*index*) của *element* trong một *list*. Nếu tìm thấy sẽ trả về *index* của phần tử đầu tiên tìm thấy. Nếu không tìm thấy sẽ phát sinh `ValueError Exception`.

Ví dụ 4.15

| Mã lệnh | Kết quả |
|---|----------------------|
| <pre>myList = [123, 'Sai Gon', 45.78, 'Sai Gon', 'Viet Nam'] try: print ("Index for Sai Gon:", myList.index('Sai Gon')) except ValueError: print ("Sai Gon is not in myList")</pre> | Index for Sai Gon: 1 |
| <pre>try: print ("Index for 122 : ", myList.index(122)) except ValueError: print ("122 is not in myList")</pre> | 122 is not in myList |

4.2.7.3. Đếm số lần xuất hiện của một giá trị trong list

- Phương thức `list.count(element)` giúp đếm số lần xuất hiện của `element` trong `list`.
- Thường được dùng để kiểm sự tồn tại.
- Ví dụ 4.16: xóa tất cả các số 1 đang có trong list

| Mã lệnh | Kết quả |
|---|----------------------------------|
| <pre>myList = [1,2,1,3,1] print ("Before delete 1:", myList)</pre> | Before delete 1: [1, 2, 1, 3, 1] |
| <pre>while myList.count(1)>0: myList.remove(1) print ("After delete 1:", myList)</pre> | After delete 1: [2, 3] |

4.2.7.4. Kiểm tra dựa trên IndexError Exception

- Python cho phép truy xuất một phần tử bất kỳ (dựa vào `index`) của mảng. Tuy nhiên, nếu truy xuất đến một phần tử không tồn tại thì ứng dụng sẽ báo lỗi. Do đó, trước khi truy xuất một phần tử, cần kiểm tra xem phần tử này đã tồn tại hay chưa bằng cách sử dụng cấu trúc `try ... except. IndexError Exception`
- Ví dụ 4.17

| Mã lệnh | Kết quả |
|--|------------------------------|
| <pre>ConGiap=["Ty/", "Suu", "Dan", "Meo", "Thin", "Ty.", "Ngo", "Mui", "Than", "Dau", "Tuat", "Hoi"] index=12 try: x = ConGiap[index] #hay if ConGiap[index]>0: print ('Phan tu thu',index,'co gia tri la:',x) except IndexError: # xử lý khi trong List không có index đó print('List khong co phan tu thu',index)</pre> | List khong co phan tu thu 12 |

4.2.8. Copy list

- Phương thức `list.copy()` trả về bản sao của list hiện tại.
- Thường được dùng khi không muốn làm ảnh hưởng đến bản gốc của list.

Ví dụ 4.18

| Mã lệnh | Kết quả |
|--|------------------------|
| <pre>myList = [1, 2, 3] copyList = myList.copy() copyList.append(4) print ('myList=',myList)</pre> | myList= [1, 2, 3] |
| <pre>print ('copyList=',copyList)</pre> | copyList= [1, 2, 3, 4] |

4.2.9. Xóa phần tử trong list

Có 5 cách xóa phần tử vào *list*:

- Xóa phần tử cuối *list*
- Xóa phần tử đầu tiên trong *list* có giá trị khớp với giá trị chỉ định
- Xóa phần tử bằng hàm *del* và dựa trên *index*
- Xóa toàn bộ *list* bằng hàm *del*
- Xóa rỗng (xóa toàn bộ các phần tử đang có) *list*

4.2.9.1. Xóa phần tử cuối trong list

- Phương thức **list.pop()**: giúp lấy (xóa) phần tử cuối cùng ra khỏi *list*.
- Ví dụ 4.19

| Mã lệnh | Kết quả |
|---------------------------|---------|
| myList = [1, 2, 3] | |
| lastnumber = myList.pop() | |
| print (lastnumber) | 3 |
| print (myList) | [1, 2] |

4.2.9.2. Xóa phần tử đầu tiên trong List có giá trị khớp với giá trị chỉ định

- Phương thức **list.remove(element)**: thực hiện xóa phần tử *element* ra khỏi *list* bằng cách lặp từ đầu đến cuối và so sánh giá trị cần xóa, nếu phần tử nào khớp với giá trị được chỉ định thì xóa.
- Lưu ý

(i)- Phương thức chỉ xóa lần so khớp đầu tiên, tức nếu trong *list* có từ 2 phần tử giống nhau trở lên thì chỉ phần tử đầu tiên bị xóa, các phần tử còn lại sẽ không bị ảnh hưởng.

- Ví dụ 4.20

| Mã lệnh | Kết quả |
|---------------------------------|--------------------------------|
| aList = [1,2,1,3,1] | |
| print ("Before remove:", aList) | Before remove: [1, 2, 1, 3, 1] |
| aList.remove(1) | |
| print ("After remove:", aList) | After remove: [2, 1, 3, 1] |

(ii)- Nếu *element* không tồn tại trong *list* thì chương trình sẽ phát sinh *ValueError exception*.

- Ví dụ 4.21

| Mã lệnh | Kết quả |
|-----------------|---|
| aList.remove(5) | ValueError: list.remove(x): x not in list |

(iii)- Khi sử dụng các lệnh lặp (*for/while*) để duyệt *list* từ đầu đến cuối và xóa bằng phương thức *remove*, các lệnh lặp này dựa trên *index*, vì vậy khi 1 phần tử tại *index = i* bị xóa, các số đi sau sẽ tự động dồn lên trước, do đó phần tử trước khi xóa có *index = i+1* thì sau khi xóa phần tử này sẽ có *index = i* sẽ bị các lệnh lặp bỏ qua vì *index=i* vừa được xét xong (ví dụ cách 1 và cách 2).

Để tránh trường hợp trên, nên thực hiện duyệt *list* từ cuối về đầu khi thực hiện *remove* (ví dụ cách 3).

Vì vậy khi không bị bắt buộc dùng phương thức *remove* trên *list*, người ta thường sử dụng hàm ẩn danh trên *list* ban đầu để tạo ra 1 *list* mới chứa các giá trị không cần xóa (ví dụ cách 4).

- Ví dụ 4.22: xóa các số chẵn trong list

| Mã lệnh | Kết quả |
|--|--|
| ''' Cách 1 chương trình không gây lỗi nhưng khi các số cần xóa nằm liền nhau thì số đi sau sẽ bị bỏ qua ''' | |
| <pre>print('CÁCH 1: ') lst=[2,2,3,6,8] for so in lst: print('So đang xét:',i) if so %2==0: lst.remove(so) print('Xóa %d,list con lại' %so, lst) print('List sau xóa:',lst)</pre> | <pre>CÁCH 1: So đang xét: 2 Xóa 2,list con lại [2, 3, 6, 8] So đang xét: 3 #bỏ qua số 2 lúc này có index=0 So đang xét: 6 Xóa 6,list con lại [2, 3, 8] List sau xóa: [2, 3, 8] #bỏ qua số 8 lúc này có index=3</pre> |
| ''' Cách 2 các số cần xóa nằm liền nhau vẫn bị bỏ qua & chương trình phát sinh lỗi IndexError''' | |
| <pre>print('CÁCH 2') lst=[2,2,3,6,8] dai=len(lst) for i in range (dai): try: so=lst[i] print('Đang xét index=%d, voi gia tri %d' %(i,lst[i])) if lst[i] % 2 == 0: lst.remove(lst[i]) print('Xóa %d,list con %d phan tu:' %(so,len(lst)), lst) except IndexError: print('IndexError: index=%d, khi len(lst)=%d' %(i,len(lst))) print('List sau xóa:',lst)</pre> | <pre>CÁCH 2 Đang xét index=0, voi gia tri 2 Xóa 2,list con 4 phan tu: [2, 3, 6, 8] Đang xét index=1, voi gia tri 3 #bỏ qua số 2 lúc này có index=0 Đang xét index=2, voi gia tri 6 Xóa 6,list con 3 phan tu: [2, 3, 8] #đoạn cuối của list bị phát sinh lỗi IndexError IndexError: index=3, khi len(lst)=3 IndexError: index=4, khi len(lst)=3 List sau xóa: [2, 3, 8]</pre> |
| ''' Cách 3 duyệt list từ cuối về đầu sẽ không gây lỗi và không bị bỏ qua các phần tử nằm kề nhau và cùng thỏa điều kiện ''' | |
| <pre>print('CÁCH 3') lst=[2,2,3,6,8] for i in range (len(lst)-1,-1,-1): try: so=lst[i] print('Đang xét index=%d, voi gia tri %d' %(i,lst[i])) if lst[i] % 2 == 0: lst.remove(lst[i]) print('Xóa %d,list con %d phan tu la:' %(so,len(lst)), lst) except IndexError: print('IndexError: index=%d, khi len(lst)=%d' %(i,len(lst)))</pre> | <pre>CÁCH 3 Đang xét index=4, voi gia tri 8 Xóa 8,list con 4 phan tu la: [2, 2, 3, 6] Đang xét index=3, voi gia tri 6 Xóa 6,list con 3 phan tu la: [2, 2, 3] Đang xét index=2, voi gia tri 3 Đang xét index=1, voi gia tri 2 Xóa 2,list con 2 phan tu la: [2, 3]</pre> |

| | |
|---|--|
| <code>print('List sau xoa:',lst)</code> | Đang xét index=0, với giá trị 2
Xóa 2, list còn 1 phần tử là:
[3]
List sau xoa: [3] |
|---|--|

''' **Cách 4** không sử dụng remove, tạo list phụ chứa các phần tử KHÔNG thỏa điều kiện '''

| | |
|--|-------------------|
| <pre>print('CÁCH 4:') lst=[2,2,3,6,8] lstB=list(filter(lambda x: x%2!=0, lst)) print('List sau xoa:',lstB)</pre> | List sau xoa: [3] |
|--|-------------------|

4.2.9.3. Xóa phần tử bằng hàm del và dựa trên index

- Nếu index không hợp lệ sẽ phát sinh lỗi.
- Thứ tự của các phần tử sẽ dịch chuyển tùy vào vị trí của phần tử bị xóa.
- Thực hiện:

- Cách 1: sử dụng **index**

| | | |
|------------|---|------------------------------|
| Ví dụ 4.23 | Mã lệnh | Kết quả |
| | <pre>chars=['a','b','c','d','e'] del chars[1] print (chars)</pre> | <pre>['a','c','d','e']</pre> |

- Cách 2: sử dụng toán tử lấy khoảng **[start:end]**.

| | | |
|------------|---|----------------------|
| Ví dụ 4.24 | Mã lệnh | Kết quả |
| | <pre>chars=['a','b','c','d','e'] del chars[1:4] print (chars)</pre> | <pre>['a','e']</pre> |

4.2.9.4. Xóa toàn bộ list ra khỏi bộ nhớ bằng hàm del

- Sau khi xóa list bằng hàm del, list sẽ bị xóa khỏi bộ nhớ. Vì vậy, nếu sau đó chương trình có sử dụng list sẽ phát sinh lỗi *NameError*.
- Ví dụ 4.25

| | |
|---|---|
| Mã lệnh | Kết quả |
| <pre>aList = [1,2,1,3,1] print ("Before delete list:", aList) del (aList) print ("After delete list:", aList)</pre> | <pre>Before delete list: [1, 2, 1, 3, 1] NameError: name 'aList' is not defined</pre> |

4.2.9.5. Xóa rỗng (xóa toàn bộ các phần tử đang có) list

- Phương thức **list.clear()**: chỉ xóa toàn bộ các phần tử đang có list. Sau khi thực hiện, list vẫn còn tồn tại trong bộ nhớ nhưng không chứa phần tử nào (rỗng).
- Ví dụ 4.26

| | |
|---|--|
| Mã lệnh | Kết quả |
| <pre>aList = [1,2,3] print ("Before clear list:", aList) aList.clear() print ("After clear list:", aList)</pre> | <pre>Before clear list: [1, 2, 1, 3, 1] After clear list: []</pre> |

4.2.10. Sắp xếp trong list

- Phương thức `list.sort([reverse=True/False], key=FunctionName)`: sắp xếp thứ tự của giá trị trong `list`.

Trong đó:

- `reverse=True/False`: mặc định là sắp tăng dần (`False`).
- `key=FunctionName`: thường được dùng khi người lập trình muốn việc sắp xếp theo cách riêng nào đó.
- Lưu ý: Chỉ sử dụng phương thức này khi các phần tử có cùng kiểu dữ liệu. Nếu không sẽ phát sinh lỗi `TypeError`.

Ví dụ 4.27

| Mã lệnh | Kết quả |
|---|---|
| <pre>numList = [123, 45.67, 79, 15.13] numList.sort() print ("List : ", numList)</pre> | List : [15.13, 45.67, 79, 123] |
| <pre>strList = ['Ty', 'Suu', 'Dan'] strList.sort() print ("List : ", strList)</pre> | List : ['Dan', 'Suu', 'Ty'] |
| <pre>mixList = [123, 'Suu', 45.67] mixList.sort() print ("List : ", mixList)</pre> | TypeError: '<' not supported between instances of 'str' and 'int' |
| <pre># lấy phần tử thứ 2 làm căn cứ để sort def takeSecond(elem): return elem[1] # khai báo list lst = [(2, 2), (3, 4), (4, 1), (1, 3)] print('Before list:', lst) # sắp xếp list dựa trên hàm takeSecond lst.sort(key=takeSecond) print('Sorted list:', lst)</pre> | <p>Before</p> <p>list: [(2, 2), (3, 4), (4, 1), (1, 3)]</p> <p>Sorted</p> <p>list: [(4, 1), (2, 2), (1, 3), (3, 4)]</p> |
| <pre>def takeSecond(elem): return elem[1] lst = [(2, 2), (3, 4), (4, 1), (1, 3)] print('Before list:', lst) lst.sort(reverse=True, key=takeSecond) print('Sorted list:', lst)</pre> | <p>Before</p> <p>list: [(2, 2), (3, 4), (4, 1), (1, 3)]</p> <p>Sorted</p> <p>list: [(3, 4), (1, 3), (2, 2), (4, 1)]</p> |

4.2.11. Đảo ngược thứ tự trong list

- Phương thức `list.reverse()` giúp đảo ngược thứ tự các giá trị của một `list`.
- Phương thức này không trả về kết quả mà thay đổi trực tiếp `list`.

Ví dụ 4.28

| Mã lệnh | Kết quả |
|---|--------------|
| <pre>numList = [1, 2, 3, 4] numList.reverse() print (numList)</pre> | [4, 3, 2, 1] |

4.2.12. Trích xuất sublist

- Có thể tạo các `list` mới từ `sublist` thông qua toán tử lấy khoảng (`range`).
- Cú pháp `listname [indexStart : indexEnd: step]`

Trong đó:

- **indexStart** : vị trí bắt đầu, mặc định là 0
- **indexEnd** : vị trí kết thúc. Mặc định là vị trí cuối của *list*.
- **Step** : đây là tham số tùy chọn, thể hiện bước tăng. Khi *indexStart* > *indexEnd*, để lấy được các phần tử thì *step* < 0.

- Ví dụ 4.29

| Mã lệnh | Minh họa List | | | | | | | | | | | | Kết quả subL |
|---|---------------|----|----|----|----|----|----|----|----|----|----|----|---|
| lst = [5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60] | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | |
| | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 | |
| subL=lst[1:10] | | x | x | x | x | x | x | x | x | x | | | [10, 15, 20, 25, 30, 35, 40, 45, 50] |
| subL=lst[1:10:-1] | | | | | | | | | | | | | [] |
| subL=lst[1:10:0] | | | | | | | | | | | | | Lỗi "ValueError: slice step cannot be zero" |
| subL=lst[1:10:1] | | x | x | x | x | x | x | x | x | x | | | [10, 15, 20, 25, 30, 35, 40, 45, 50] |
| subL=lst[1:10:2] | | x | | x | | x | | x | | x | | | [10, 20, 30, 40, 50] |
| subL=lst[1:10:3] | | x | | | x | | | x | | | | | [10, 25, 40] |
| subL=lst[1:10:4] | | x | | | | x | | | | x | | | [10, 30, 50] |
| subL=lst[10:2:1] | | | | | | | | | | | | | [] |
| subL=lst[10:2:-1] | | | | x | x | x | x | x | x | x | x | | [55, 50, 45, 40, 35, 30, 25, 20] |
| subL=lst[10:2:-2] | | | | | x | | x | | x | | x | | [55, 45, 35, 25] |
| subL=lst[10:2:-3] | | | | | x | | | x | | | x | | [55, 40, 25] |
| subL=lst[10:2:-4] | | | | | | | x | | | | x | | [55, 35] |
| subL=lst[10:2:-5] | | | | | | x | | | | | x | | [55, 30] |
| subL=lst[::-1] | | x | x | x | x | x | x | x | x | x | x | x | [60, 55, 50, 45, 40, 35, 30, 25, 20, 15, 10, 5] |
| subL=lst[::-2] | | x | | x | | x | | x | | x | | x | [60, 50, 40, 30, 20, 10] |

4.2.13. Thao tác trên nhiều list

4.2.13.1. Mở rộng list bằng cách nối list khác vào list muốn mở rộng

- Cú pháp: **Tên_List_muốn_mở_rộng.extend(tên_list_khác)**

Ví dụ 4.30

| Mã lệnh | Kết quả |
|---|-----------------|
| one =[1,2]
two=[3,4,5]
one.extend(two)
print (one) | [1, 2, 3, 4, 5] |

4.2.13.2. Tạo list mới bằng cách nối nhiều list

- Sử dụng toán tử **+** để nối giá trị của 2 (hay nhiều) list và tạo ra một list lớn có số lượng phần tử là tổng số lượng phần tử của các list ban đầu.
- Sử dụng toán tử ***n** để tạo ra một list mới từ việc lặp lại list ban đầu **n** lần

Ví dụ 4.31

| Mã lệnh | Kết quả |
|---|--------------------|
| one = [1, 2]
two = [3, 4]
print (one+two) | [1, 2, 3, 4] |
| three = one+two
print (three) | [1, 2, 3, 4] |
| four = one*3
print (four) | [1, 2, 1, 2, 1, 2] |

4.2.13.3. Tìm những phần tử có trong list1 nhưng không có trong list2

- Cú pháp
 - Cách 1: **list(set(list1) - set(list2))**
 - Cách 2: **set(list1).difference(set(list2))**
- Ví dụ 4.32

| Mã lệnh | Kết quả |
|--|---------|
| list1 = [1, 2, 3, 4]
list2 = [1, 2]
list3 = list(set(list1) -
set(list2))
print(list3) | [3, 4] |
| list3 = list(set(list2) -
set(list1))
print(list3) | [] |
| set1 = set(list1)
set2 = set(list2)
list3 =
list(set1.difference(set2))
print(list3) | [3, 4] |

4.2.14. List lồng nhau

Là 1 List gồm các phần tử là 1 hay nhiều List con (SubList) . Ví dụ:

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

4.2.14.1. Làm phẳng list (flatten list)

- Là “rã” các phần tử có trong subList thành những phần tử đơn thuộc List chính.
- Yêu cầu các thành phần trong List đều phải là sublist (chứ không phải là các đối tượng đơn như int, string, float, ...)

4.2.14.1.1. Cách 1: Dùng phương thức chain.from_iterable trpng module itertools

- Ví dụ 4.33

| Mã lệnh | Kết quả |
|--|--|
| <pre># 1. trong List là các subList import itertools L = [[1, 2, 3], ['m', 'Kg']] print('Before',L) L = list(itertools.chain.from_iterable(L)) print('After',L)</pre> | <p>Before [[1, 2, 3], ['m', 'Kg']]</p> <p>After [1, 2, 3, 'm', 'Kg']</p> |
| <pre>L = [[1, 2], 3, ['m', 'Kg']] #Lỗi do số nguyên (int) 3 không thuộc 1 sublist L = list(itertools.chain.from_iterable(L)) print('After',L)</pre> | <p>TypeError: 'int' object is not iterable</p> <p>#Lệnh không thực hiện do có lỗi trước đó</p> |
| <pre>''' 2. trong mỗi subList lại gồm nhiều subsubList ''' import itertools L = [[1,2,3], [4,5,6,7]],[['m','Kg'], [8]] print('Before',L) L = list(itertools.chain.from_iterable(L)) print('After',L)</pre> | <p>Before [[[1, 2, 3], [4, 5, 6, 7]], [['m', 'Kg'], [8]]]</p> <p>After [[1, 2, 3], [4, 5, 6, 7], ['m', 'Kg'], [8]]</p> |

4.2.14.1.2. Cách 2:Dùng hàm sum

- Ví dụ 4.34

| Mã lệnh | Kết quả |
|---|--|
| <pre># 1. trong List là các subList L = [[1, 2, 3], ['m', 'Kg']] print('Before',L) L = sum(L, []) print('After',L)</pre> | <p>Before [[1, 2, 3], ['m', 'Kg']]</p> <p>After [1, 2, 3, 'm', 'Kg']</p> |
| <pre>L = [[1, 2, 3], ['m', 'Kg'],4] print('Before',L) #Lỗi do số nguyên (int) 4 không thuộc 1 sublist L = sum(L, [])</pre> | <p>Before [[1, 2, 3], ['m', 'Kg'], 4]</p> <p>TypeError: can only concatenate list (not "int") to list</p> |
| <pre>print('After',L)</pre> | <p>#Lệnh không thực hiện do có lỗi trước đó</p> |
| <pre># 2. trong mỗi subList lại gồm nhiều subsubList L=[[1,2], [3,4,5]], [['m','Kg'], [6],['A']] print('Before',L) L = sum(L, []) print('After',L)</pre> | <p>Before [[1, 2], [3, 4, 5], ['m', 'Kg'], [6], ['A']]</p> <p>After [1, 2, 3, 4, 5, 'm', 'Kg', 6, 'A']</p> |

4.2.14.1.3. Cách 3:Dùng list comprehension

Ví dụ 4.35

| Mã lệnh | Kết quả |
|--|--|
| <pre># 1. trong List là các subList L = [[1, 2, 3], ['m', 'Kg']] print('Before',L)</pre> | <p>Before [[1, 2, 3], ['m', 'Kg']]</p> |
| <pre>L=[x for subL in L for x in subL]</pre> | <p>After [1, 2, 3, 'm', 'Kg']</p> |

| | |
|---|--|
| <code>print('After',L)</code> | |
| <code>L = [[1, 2, 3], ['m', 'Kg'], 4]</code> | |
| <code>#Lỗi do số nguyên (int) 4 không thuộc 1 sublist</code>
<code>L=[x for subL in L for x in subL]</code> | <code>TypeError: 'int' object is not iterable</code> |
| <code>print('After',L)</code> | <code>#Lệnh không thực hiện do có lỗi trước đó</code> |
| <code># 2. trong mỗi subList lại gồm nhiều subsubList</code>
<code>L= [[1,2], [3,4,5], [['m','Kg'], [6],['A']]]</code>
<code>print('Before',L)</code> | Before <code>[[1, 2], [3, 4, 5], [['m', 'Kg'], [6], ['A']]]</code> |
| <code>L=[x for sL in L for ssL in sL for x in ssL]</code>
<code>print('After',L)</code> | After <code>[[1, 2], [3, 4, 5], ['m', 'Kg'], [6], ['A']]</code> |

4.2.14.2. Gộp từng cặp sublist của 2 list

Yêu cầu số lượng sublist trong 2 list là **bằng nhau**;

4.2.14.2.1. Cách 1: sử dụng map và lambda

Ví dụ 4.36

| Mã lệnh | Kết quả |
|--|--|
| <code>list1 = [[1, 2], [3, 4, 5], [6]]</code>
<code>list2 = [[7], [8, 9], [10, 11, 12]]</code>
<code># Using map + lambda to merge lists</code>
<code>resultList = list(map(lambda x, y: x + y, list1, list2))</code>
<code>print(resultList)</code> | <code>[[1, 2, 7], [3, 4, 5, 8, 9], [6, 10, 11, 12]]</code> |

4.2.14.2.2. Cách 2: sử dụng hàm Zip

Ví dụ 4.37

| Mã lệnh | Kết quả |
|---|--|
| <code>list1 = [[1, 2], [3, 4, 5], [6]]</code>
<code>list2 = [[7], [8, 9], [10, 11, 12]]</code>
<code># Using zip to merge lists</code>
<code>resultList = [x + y for x, y in zip(list1, list2)]</code>
<code>print(resultList)</code> | <code>[[1, 2, 7], [3, 4, 5, 8, 9], [6, 10, 11, 12]]</code> |

4.2.14.2.3. Cách 3: sử dụng các hàm starmap(), concat() và zip()

Ví dụ 4.38

| Mã lệnh | Kết quả |
|--|--|
| <code>from operator import concat</code>
<code>from itertools import starmap</code>
<code>list1 = [[1, 2], [3, 4, 5], [6]]</code>
<code>list2 = [[7], [8, 9], [10, 11, 12]]</code>
<code># Using zip to merge lists</code>
<code>resultList = list(starmap(concat, zip(list1, list2)))</code>
<code>print(resultList)</code> | <code>[[1, 2, 7, 8, 9], [3, 4, 10, 11, 12], [5, 6, 13, 14, 15]]</code> |

4.2.15. Chuyển đổi một iterable object (tuple, string, set, dictionary) thành list.

- Sử dụng hàm `list()` để chuyển đổi một đối tượng có thể lặp (tuple, string, set, dictionary) thành list.

- Ví dụ 4.39

| Mã lệnh | Kết quả |
|---|-----------------|
| <pre>mytuple = ('x', 'y', 'z') lst=list(mytuple) print (lst)</pre> | ['x', 'y', 'z'] |
| <pre>mydict = dict({1:'one', 2:'two'}) lst=list(mydict) print (lst)</pre> | [1, 2] |
| <pre>myset={1,5,3,4,2} lst=list(myset) print (lst)</pre> | [1, 2, 3, 4, 5] |

4.3. Tuple

4.3.1. Giới thiệu

- Tuple cũng là một cấu trúc dạng *sequence* tương tự như cấu trúc *list*, có thể chứa dữ liệu thuộc nhiều kiểu dữ liệu khác nhau.
- So sánh giữa *List* và *Tuple*:
 - **Giống nhau:**
 - Index của các phần tử được tính từ 0 và index của phần tử cuối là số lượng phần tử của tuple-1.
 - Cùng hỗ trợ các cách truy xuất phần tử như *index*, *range*, tìm kiếm, ...
 - **Khác nhau:** Một *Tuple* đã được khai báo rồi thì:
 - Không thay đổi được giá trị (*immutable*)
 - Số lượng phần tử là cố định. Do đó, *tuple* không hỗ trợ các phương thức như *append()*, *pop()*.
 - *Tuple* thường được sử dụng cho các kiểu dữ liệu không đồng nhất (khác nhau) và *list* thường sử dụng cho các kiểu dữ liệu (đồng nhất) giống nhau.
 - **Ưu điểm** của *tuple* so với *list*:
 - Vì *tuple* không thể thay đổi, việc lặp qua các phần tử của *tuple* nhanh hơn so với *list*.
 - *Tuple* chứa những phần tử không thay đổi, có thể được sử dụng như *key* cho *dictionary*. Với *list*, điều này không thể làm được.
 - Nếu trong ứng dụng có dữ liệu không cho phép thay đổi, việc triển khai dữ liệu dưới dạng *tuple* sẽ đảm bảo rằng dữ liệu đó được bảo vệ chống ghi (*write-protected*).

4.3.2. Tạo tuple

4.3.2.1. Khai báo và gán giá trị cho tuple

- Các phần tử trong *tuple* được đặt trong cặp ngoặc đơn (*parentheses*) và cách nhau bằng dấu phẩy (,).
- Tuy *Python* cho phép bỏ qua cặp dấu ngoặc đơn khi khai báo, nhưng thông thường người ta vẫn dùng vì giúp nhìn *tuple* dưới dạng 1 danh sách.
- Ví dụ 4.40

| Mã lệnh | Kết quả |
|--|-----------------|
| <pre>mytuple = ('x', 'y', 'z') print (mytuple)</pre> | ('x', 'y', 'z') |
| <pre>mytuple2 = 'x', 'y', 'z' print (mytuple2)</pre> | ('x', 'y', 'z') |

4.3.2.2. Tạo tuple từ đối tượng dạng iterator (list, string, set, dictionary) đã có

- Tạo ra 1 tuple với giá trị được sao chép từ 1 đối tượng dạng iterator (*list, string, set, dictionary*).
- Cú pháp: **tuple_Name = tuple (Iterator_Name)**
- Ví dụ 4.41

| Mã lệnh | Kết quả |
|--|-------------------------------------|
| mylist = ['x', 12, 3.14]
mytuple = tuple(mylist)
print (mytuple) | ('x', 12, 3.14) |
| S='Sai Gon'
mytuple = tuple(S)
print (mytuple) | ('S', 'a', 'i', ' ', 'G', 'o', 'n') |

4.3.2.3. Tạo mới tuple bằng toán tử + hoặc * trên tuple đã có

- **Tuple1 + Tuple2**: tạo mới 1 tuple từ 2 Tuple1 và Tuple2 đã có.
- **Tuple1 * n**: (n là số nguyên) tạo mới 1 tuple mới bằng cách thực hiện n lần việc copy và nối Tuple1.
- Ví dụ 4.42

| Mã lệnh | Kết quả |
|--|-------------------------|
| tuple1 = (2, 4, 6)
tuple2 = (1, 3)
tuple3 = tuple1 + tuple2
print(tuple3) | (2, 4, 6, 1, 3) |
| tuple4 = "Sai gon "
tuple5 = tuple4 * 3
print(tuple5) | Sai gon Sai gon Sai gon |

4.3.3. Truy cập các phần tử của tuple

4.3.3.1. Truy cập các phần tử của tuple bằng toán tử index

- Sử dụng toán tử **index []** để truy cập vào phần tử trong tuple với index bắt đầu bằng 0. Nghĩa là nếu tuple có n phần tử thì index của tuple sẽ bắt đầu từ 0 đến n-1.
- Lỗi khi sử dụng sai index:
 - Lỗi *IndexError* sẽ xảy ra khi $index \geq n$.
 - Lỗi *TypeError* sẽ xảy ra khi index không phải là số nguyên (như số thập phân hay bất kỳ kiểu dữ liệu nào khác).
- **Index âm**: Python cho phép lập chỉ mục âm cho các đối tượng dạng chuỗi. Index -1 tham chiếu đến phần tử cuối cùng, -2 là thứ 2 tính từ cuối tính lên.

| | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|--------|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | ←index |
| "S" | "a" | "i" | " " | "G" | "o" | "n" | |
| -7 | -6 | -5 | -4 | -3 | -2 | -1 | ←index |

- **Toán tử slicing (cắt lát)**: Có thể truy cập đến một loạt phần tử trong tuple bằng cách sử dụng toán tử slicing : (dấu 2 chấm).

4.3.3.2. Truy cập thành phần con của các phần tử trong tuple

- Những thành phần con của các phần tử trong tuple được truy cập bằng cách sử dụng index lồng nhau.

- Sử dụng lệnh *for* để duyệt qua từng phần tử trong *tuple*
- Ví dụ 4.43

| Mã lệnh | Kết quả |
|---|--|
| <code>mytuple = ([2, 4, 6, 8], "Sai gon", (1.23, 4.2))</code> | |
| <code>print(mytuple [0])</code> | [2, 4, 6, 8] |
| <code>print(mytuple [1])</code> | Sai gon |
| <code>print(mytuple [0] [2])</code> | 6 |
| <code>print(mytuple [1] [2])</code> | i |
| <code>print(mytuple [0] [-1])</code> | 8 |
| <code>print(mytuple [0] [-2])</code> | 6 |
| <code>for x in mytuple:</code>
<code> print(x)</code> | [2, 4, 6, 8]
Sai gon
(1.23, 4.2) |
| <code>mytuple = ([2, 4, 6, 8], "Sai gon", (1.23, 4.2))</code>
<code>for x in mytuple:</code>
<code> dem=len(x)-1</code>
<code> while dem>=0:</code>
<code> print (x[dem])</code>
<code> dem-=1</code> | 2
4
6
8
S
A
i |
| <code>#hoặc</code>
<code>for item in mytuple:</code>
<code> for element in item:</code>
<code> print (element)</code> | g
o
n
1.23
4.2 |

4.3.4. Cập nhật giá trị của phần tử trong tuple

- Không giống như *list*, *tuple* không thể thay đổi sau khi đã được gán giá trị.
- Nhưng, nếu bản thân phần tử đó là một kiểu dữ liệu có thể thay đổi (như *list* chẳng hạn) thì các phần tử lồng nhau có thể được thay đổi. Chúng ta cũng có thể gán giá trị khác cho *tuple* (gọi là gán lại - *reassignment*).
- Ví dụ 4.44

| Mã lệnh | Kết quả |
|--|---|
| <code>mytuple = ([2,4,6,8], "Sai gon", (1.23, 4.2))</code> | |
| <code>#lệnh gán thành công vì mytuple[0] là 1 list</code>
<code>mytuple [0] [-1]=10</code>
<code>print(mytuple)</code> | ([2, 4, 6, 10], "Sai gon", (1.23, 4.275, 15)) |
| <code>#lệnh gán không thành công vì mytuple[2] là 1 tuple</code>
<code>mytuple [2] [-1]=6.78</code> | <code>TypeError: 'tuple' object does not support item assignment</code> |

4.3.5. Kiểm tra một đối tượng có tồn tại trong tuple hay không?

- Sử dụng toán tử *in* hoặc *not in* để kiểm tra xem một đối tượng có tồn tại trong *tuple* hay chưa. Kết quả trả về *True* hoặc *False*.
- Ví dụ 4.45

| Mã lệnh | Kết quả |
|--|---------|
| <code>mytuple = ([2,4,6,8], "Sai gon", (1.23, 4.2))</code>
<code>print(5 in mytuple)</code> | False |
| <code>print(5 not in mytuple)</code> | True |
| <code>print([2,4,6,8] in mytuple)</code> | True |

4.3.6. Xóa tuple

- Các phần tử trong *tuple* không thể thay đổi nên chúng ta cũng không thể xóa, loại bỏ phần tử khỏi *tuple*.
- **del**: giúp xóa một *tuple* khỏi bộ nhớ
- Ví dụ 4.46

| Mã lệnh | Kết quả |
|---------------------------------------|--|
| mytuple = (2, 4, 6)
print(mytuple) | (2, 4, 6) |
| del mytuple
print(mytuple) | NameError: name 'mytuple' is not defined |

4.3.7. Một số phương thức và hàm hỗ trợ việc xử lý trên tuple

4.3.7.1. Một số phương thức của tuple

Do *tuple* không cho thêm, sửa, xóa trên các phần tử nên *tuple* không có các phương thức: *append*, *extend*, *insert*, *pop*, *remove*, *reverse*, *sort* mà *tuple* chỉ có các phương thức sau:

- **count(x)**: Đếm số phần tử *x* trong *tuple*.
- **index(x)**: Trả về giá trị *index* của phần tử *x* đầu tiên tìm thấy trong *tuple*.

4.3.7.2. Một số hàm hỗ trợ việc xử lý tuple (giống với list)

- **all()** : Trả về giá trị *True* nếu tất cả các phần tử của *tuple* là *true* hoặc *tuple* rỗng.
- **any()** : Trả về *True* nếu bất kỳ phần tử nào của *tuple* là *True*, nếu *tuple* rỗng trả về *False*.
- **enumerate()**: Trả về đối tượng *enumerate* (liệt kê), chứa cặp *index* và giá trị của tất cả phần tử của *tuple*.
- **len()** : Trả về độ dài (số lượng phần tử) của *tuple*.
- **max()** : Trả về phần tử lớn nhất của *tuple*.
- **min()** : Trả về phần tử nhỏ nhất của *tuple*.
- **sorted()** : Lấy phần tử trong *tuple* và trả về *list* mới được sắp xếp (*tuple* không sắp xếp được).
- **sum()** : Trả về tổng tất cả các phần tử trong *tuple*.

4.4. Dictionary

4.4.1. Giới thiệu

- *Dictionary* cũng là một cấu trúc dạng *sequence*. *Dictionary* gồm nhiều phần tử không có thứ tự, mỗi phần tử sẽ là 1 bộ **key:value** (cấu trúc *Dictionary* tương tự như một *object json*). Trong đó:
 - **key**: giá trị của *key* là duy nhất (không trùng) và sau này không thể chỉnh sửa. Kiểu dữ liệu có thể là *number*, *string*, *tuple*.
 - **value**: có giá trị tùy ý. Do đó có thể chỉnh sửa/cập nhật sau đó.
- *Dictionary* thường được sử dụng khi chúng ta có một số lượng lớn dữ liệu. Các *dictionary* được tối ưu hóa để trích xuất dữ liệu với điều kiện phải biết được *key* để lấy giá trị.
- Khác biệt giữa *Dictionary* và *List*, *Tuple*: *List*, *Tuple* sử dụng **index** để phân biệt các phần tử, còn *Dictionary* thì dùng các **key** để phân biệt.

4.4.2. Tạo dictionary

4.4.2.1. Tạo dictionary bằng cách khai báo và gán trực tiếp giá trị

Tạo dictionary bằng cách khởi tạo giá trị: một *Dictionary* được khởi tạo bằng cặp dấu ngoặc nhọn **{}** (*curly braces*) với mỗi phần tử là một cặp theo dạng **key : value**. Với *Key* và *value* này có thể thuộc bất kỳ kiểu dữ liệu nào.

Ví dụ 4.47

| Ý nghĩa
tạo
dictionary | Mã lệnh | Kết quả |
|------------------------------|---|--|
| Rỗng | <pre>dict0 = {}
print (dict0)</pre> | { } |
| key
là chuỗi | <pre>dict1 = {'x': 1, 'y': 0}
print (dict1)</pre> | {'x':1, 'y': 0} |
| key
là số | <pre>dict2 = {7:'apple', 2:'banana'}
print (dict2)</pre> | {7: 'apple', 2:
'banana'} |
| key
là hỗn hợp | <pre>dict3 = {'Tinh_TP': 'Sai gon', 1: [1, 3, 5]}
print (dict3)</pre> | {'Tinh_TP':
'Sai gon', 1:
[1, 3, 5]} |

4.4.2.2. Tạo dictionary bằng hàm **dict()**

Ví dụ 4.48

| Mã lệnh | Kết quả |
|--|-----------------------|
| <pre>dict4 = dict({1:'one', 0:'zero'})
'''2 cách sau đây cho kết quả tương tự:
dict4=dict([(1, 'one'), (0: 'zero')])
dict4=dict(zip([1, 0], ['one', 'zero']))'''
print (dict4)</pre> | {1: 'one', 0: 'zero'} |

4.4.2.3. Tạo dictionary bằng phương thức **fromkeys()**

Ví dụ 4.49

| Mã lệnh | Kết quả |
|---|---|
| <pre>mylist=['one','two','three']
mydict=dict.fromkeys(mylist)
print("my dict=",mydict)</pre> | my dict= {'one': None, 'two': None, 'three':
None} |

4.4.3. Thêm phần tử vào dictionary (hoặc cập nhật value thông qua key)

- Cú pháp: **dict_name[key] = value**
- Giải thích:
 - Nếu *key* đã có, giá trị sẽ được cập nhật là *value*.
 - Ngược lại, khi chưa có, một cặp **key: value** mới sẽ được thêm vào *dictionary*.

- Ví dụ 4.50

| Ý nghĩa | Mã lệnh | Kết quả |
|-----------------------------------|--|---|
| Khai báo và gán
giá trị | <pre>mydict = {'name': 'Jone', 'age': 30}
print (mydict)</pre> | {'name': 'Jone',
'age': 30} |
| Thêm phần tử
vào <i>mydict</i> | <pre>mydict['country'] = 'Vietnam'
print (mydict)</pre> | {'name': 'Jone',
'age': 30,
'country': 'Vietnam'} |

| | | |
|------------------------------|---|--|
| Thay đổi value thông qua key | <pre>mydict['name'] = 'Hong' print (mydict)</pre> | <pre>{'name': 'Hong', 'age': 30, 'country': 'Vietnam'}</pre> |
|------------------------------|---|--|

4.4.4. Truy xuất phần tử của dictionary

4.4.4.1. Truy xuất các items của dictionary

- Phương thức *items* trả về 1 đối tượng thuộc class '*dict_items*'.
- Ví dụ 4.51

| Mã lệnh | Kết quả |
|--|--|
| <pre>dict_A = {1: "MySQL", 2: "SQLServer", 3:"SQLite"} print(dict_A.items())</pre> | <pre>dict_items([(1, 'MySQL'), (2, 'SQLServer'), (3, 'SQLite')])</pre> |

4.4.4.2. Truy xuất cả 2 thành phần key và value của Dictionary

- Sử dụng phương thức *items* của đối tượng *dictionary*.
- Ví dụ 4.52: có thể dùng 1 trong 2 cách sau:

| Mã lệnh | Kết quả |
|--|---|
| <pre>D = {'three':3, 'one':1, 'four':4, 'two':2} for item in D.items(): k, v = item print('key=', k, ':value=', v)</pre> | <pre>key=three:value=3 key=one:value=1 key=four:value=4 key=two:value=2</pre> |
| <pre>for k, v in D.items(): print(' key={} :value={} '.format(k, v))</pre> | <pre>key=three:value=3 key=one:value=1 key=four:value=4 key=two:value=2</pre> |

4.4.4.3. Lấy danh sách các key trong dictionary

- Phương thức *keys* trả về 1 đối tượng thuộc class '*dict_keys*'.
- Ví dụ 4.53

| Mã lệnh | Kết quả |
|---|---------------------------------|
| <pre>dict_A = {1: "MySQL", 2: "SQLServer", 3:"SQLite"} print(dict_A.keys())</pre> | <pre>dict_keys([1, 2, 3])</pre> |
| <pre>for item in dict_A.keys(): print(item,end='; ')</pre> | <pre>1; 2; 3;</pre> |

4.4.4.4. Lấy danh sách các value trong dictionary

- Phương thức *values* trả về 1 đối tượng thuộc class '*dict_values*'.
- Ví dụ 4.54

| Mã lệnh | Kết quả |
|---|--|
| <pre>dict_A = {1: "MySQL", 2: "SQLServer", 3:"SQLite"} print(dict_A.values())</pre> | <pre>dict_values(['MySQL', 'SQLServer', 'SQLite'])</pre> |
| <pre>for item in dict_A.values(): print(item,end='; ')</pre> | <pre>MySQL; SQLServer; SQLite;</pre> |

4.4.4.5. Truy xuất value thông qua key

- Các kiểu dữ liệu lưu trữ khác sử dụng *index* để truy cập vào các giá trị thì *dictionary* sử dụng các *key*.
- Cách sử dụng: có 2 cách
 - Cách 1: đặt *Key* trong cặp dấu ngoặc vuông (dùng *Key* làm *index*).
 - Cách 2: đặt *Key* làm tham số trong phương thức *get()* của đối tượng *dictionary*.

- Ví dụ 4.55

| Ý nghĩa | Mã lệnh | Kết quả |
|--|--|------------------------------|
| Truy cập qua key | <pre>mydict = {0:'zero', 9:'nine', 6:'six', 2:'two'} print ("mydict[2]= ", mydict[2])</pre> | <pre>mydict[2]= two</pre> |
| Duyệt mydict dựa trên key để lấy value | <pre>for key in mydict: print(mydict[key]) #hoặc for key in mydict: print(mydict.get(key))</pre> | <pre>zero nine six two</pre> |

4.4.5. Kiểm tra một key đã có trong dictionary hay chưa?

- Sử dụng toán tử *in* hoặc *not in* để kiểm tra một key đã có trong *dictionary* hay chưa? Kết quả trả về True hoặc False.
- Lưu ý: không thể thực hiện với *value*.
- Ví dụ 4.56

| Mã lệnh | Kết quả |
|---|--|
| <pre>mydict = {0: 'khong', 1: 'mot', 2: 'hai', 3: 'ba'} print ("\ 'hai\ ' KHONG co trong mydict:", 'hai' not in mydict)</pre> | <pre>'hai' KHONG co trong mydict: True</pre> |
| <pre>print ("2 CO trong mydict:", 'một' in mydict)</pre> | <pre>2 CO trong mydict: True</pre> |

4.4.6. Chuyển đổi 2 thành phần key và value cho nhau

- Sử dụng hàm *zip* để thực hiện.
- Ví dụ 4.57

| Mã lệnh | Kết quả |
|---|--|
| <pre>D = {'one': 1, 'two': 2, 'three': 3, 'four': 4} print(D.items())</pre> | <pre>dict_items([('one',1), ('two',2), ('three',3), ('four',4)])</pre> |
| <pre>D2 = dict(zip(D.values(), D.keys())) print(D2.items())</pre> | <pre>dict_items([(1,'one'), (2,'two'), (3,'three'), (4,'four')])</pre> |

4.4.7. Chuyển đổi list sang dictionary**4.4.7.1. Sử dụng hàm dict**

- Do cấu trúc của *dictionary* gồm 2 phần *Key : Value* nên khi chuyển đổi sang *dictionary*, mỗi phần tử của *list* phải là một cặp.
- Ví dụ 4.58

| Mã lệnh | Kết quả |
|---|-------------------------|
| <pre>mylist = [[2,4],[1,3]] mydict = dict(mylist) print(mydict)</pre> | <pre>{2: 4, 1: 3}</pre> |

4.4.7.2. Viết chương trình tạo dictionary từ list

- **Ví dụ 4.59:** Cho nhập 1 chuỗi (S). Sử dụng dictionary để đếm số lần xuất hiện của từng từ trong (S).
- Minh họa:

```
S=''' ... Ai ngồi, ai câu, ai sâu, ai thăm
Ai thương, ai cảm, ai nhớ, ai trông ... '''
```

sẽ xuất ra: {'...': 2, 'Ai': 2, 'ngồi,': 1, 'ai': 6, 'câu,': 1, 'sâu,': 1, 'thâm': 1, 'thương,': 1, 'cảm,': 1, 'nhớ,': 1, 'trông': 1}

- Thực hiện:

```
def word_count(S):
    countDict = dict() #tạo dictionary rỗng
    wordList = (S).split() #tạo list chứa các từ
    for word in wordList: #duyệt wordList để đưa vào countDict
        if word in countDict:
            countDict[word] += 1
        else:
            countDict[word] = 1
    return countDict
```

```
S=''' ... Ai ngồi, ai câu, ai sâu, ai thâm
Ai thương, ai cảm, ai nhớ, ai trông ...'''
print(word_count(S))
```

4.4.8. Viết chương trình tạo dictionary từ string

Ví dụ 4.60 Cho nhập 1 chuỗi (S). Sử dụng dictionary để đếm số lần xuất hiện của từng ký tự trong (S).

- Minh họa: S='good morning' sẽ xuất ra:

| | | | |
|---|---|---|----------------------------|
| g | = | 2 | lần |
| o | = | 3 | lần |
| d | = | 1 | lần |
| = | = | 1 | lần #số lượng khoảng trắng |
| m | = | 1 | lần |
| r | = | 1 | lần |
| n | = | 2 | lần |
| i | = | 1 | lần |

- Thực hiện:

- **CÁCH 1:** kết quả được sắp xếp theo thứ tự xuất hiện của ký tự trong chuỗi ban đầu

```
def char_frequency(Str):
    dict = {}
    for C in Str:
        Keys = dict.keys()
        if C in Keys: #có trong dict => tăng value
            dict[C] += 1
        else: #thêm key mới là C với value=1
            dict[C] = 1
    return dict
```

```
Str='good morning'
mydict=char_frequency(Str)
print('Các ký tự xuất hiện trong chuỗi:')
for k,v in mydict.items():
    print(k,'=',v,'lần')
```

- **CÁCH 2:** kết quả được sắp xếp giảm dần theo value

```
import collections
Str='good morning'
#tạo dictionary rỗng
mydict = collections.defaultdict(int)
'''xét từng ký tự trong Str, nếu chưa có thì tạo key mới=c và value=1; nếu có rồi thì tăng value lên 1'''
for c in Str:
```

```

mydict[c] += 1
print('Các ký tự xuất hiện trong chuỗi:')
'''sắp xếp mydict giảm dần (reverse=True) dựa trên value
(key=mydict.get). Duyệt từng ký tự trong mydict đã được sắp
xếp để in key-value'''
for c in sorted(mydict, key= mydict.get, reverse=True):
    print('%s = %d lần' % (c, mydict[c]))

```

4.4.9. Xóa trên dictionary

4.4.9.1. Xóa phần tử được chỉ định trong dictionary

- Cú pháp

- Cách 1: `Tên_dictionary.pop(KEY [, ValueIfNotFound])`

Giải thích:

- Xóa phần tử có `key=KEY` trong `dictionary`.
- Nếu tìm thấy `key`, trả về giá trị (`value`) của phần tử bị xóa. Ngược lại khi không tìm thấy sẽ trả về `ValueIfNotFound`. Nếu `ValueIfNotFound` không được cấp và `KEY` không tồn tại sẽ tạo lỗi `KeyError`.

Ví dụ 4.61

| Mã lệnh | Kết quả |
|--|---|
| <pre> mydict = {1: 'mot', 2: 'hai', 3: 'ba'} #xóa key=2 có trong mydict x=mydict.pop(2) print("x=", x) </pre> | <pre> x= hai </pre> |
| <pre> #xóa key=5 KHÔNG có trong mydict x=mydict.pop(5,-1) print("x=", x) </pre> | <pre> x= -1 </pre> |
| <pre> '''#xóa key=5 KHÔNG có trong mydict và không có tham số ValueIfNotFound''' x=mydict.pop(5) print("x=", x) </pre> | <pre> KeyError: 5 '''Không thực hiện do lỗi KeyError trước đó''' </pre> |

- Cách 2: `del Tên_dictionary[KEY]`

Giải thích:

- Xóa phần tử có `key=KEY` trong `dictionary`. Nếu `KEY` không tồn tại sẽ tạo lỗi `KeyError`
- Không trả về giá trị (`value`) của phần tử bị xóa.

Ví dụ 4.62

| Mã lệnh | Kết quả |
|--|---|
| <pre> mydict = {1: 'mot', 2: 'hai', 3: 'ba'} print("my dict = ", mydict) </pre> | <pre> my dict = {1: 'mot', 2: 'hai', 3: 'ba'} </pre> |
| <pre> #xóa key=2 CÓ trong mydict del mydict[2] print("my dict = ", mydict) </pre> | <pre> my dict = {1: 'mot', 3: 'ba'} </pre> |
| <pre> mydict = {1: 'mot', 2: 'hai', 3: 'ba'} print("my dict = ", mydict) #xóa key=5 KHÔNG có trong mydict del mydict[5] </pre> | <pre> KeyError: 5 </pre> |
| <pre> print("my dict = ", mydict) </pre> | <pre> '''Không thực hiện do lỗi KeyError trước đó''' </pre> |

4.4.9.2. Xóa phần tử cuối cùng trong dictionary

- **Cú pháp:** `Tên_dictionary.popitem()`
- **Giải thích:**
 - Sinh lỗi `KeyError` nếu *dictionary* rỗng.
 - Kết quả trả về ở dạng *(key, value)* của phần tử bị xóa.
- Ví dụ 4.63

| Mã lệnh | Kết quả |
|---|---|
| <code>mydict = {1: 'mot', 2: 'hai', 3: 'ba'}
print("my dict=", mydict)</code> | <code>my dict= {1: 'mot', 2: 'hai', 3: 'ba'}</code> |
| <code>dict_item = mydict.popitem()
print("dic_item=", dict_item)</code> | <code>dic_item= (3, 'ba')</code> |
| <code>print("my dict=", mydict)</code> | <code>my dict= {1: 'mot', 2: 'hai'}</code> |

4.4.9.3. Xóa tất cả các phần tử trong dictionary

- **Cú pháp:** `Tên_dictionary.clear()`
- **Giải thích:** xóa tất cả các phần tử trong *dictionary*. Sau xóa, *dictionary* vẫn còn tồn tại nhưng rỗng.
- Ví dụ 4.64

| Mã lệnh | Kết quả |
|---|---|
| <code>mydict = {1: 'mot', 2: 'hai', 3: 'ba'}
print("my dict=", mydict)</code> | <code>my dict= {1: 'mot', 2: 'hai', 3: 'ba'}</code> |
| <code>mydict.clear()
print("my dict=", mydict)</code> | <code>my dict= {}</code> |

4.4.9.4. Xóa bỏ hẳn dictionary

- **Cú pháp:** `del Tên_dictionary`
- **Giải thích:** xóa bỏ hẳn *dictionary* trong bộ nhớ.
- Ví dụ 4.65

| Mã lệnh | Kết quả |
|---|--|
| <code>mydict = {1: 'mot', 2: 'hai', 3: 'ba'}
print("my dict=", mydict)</code> | <code>my dict= {1: 'mot', 2: 'hai', 3: 'ba'}</code> |
| <code>del mydict
print(mydict)</code> | <code>NameError: name 'mydict' is not defined</code> |

4.4.10. Gộp 2 dictionaries

4.4.10.1. Sử dụng phương thức update()

- **Trường hợp 1:** Phương thức *update* không trả về 1 *dictionary* mới, *dictionary* bên vế trái của biểu thức sẽ bị xóa toàn bộ nội dung, *dictionary* bên vế phải không thay đổi. Vì vậy, không nên dùng trường hợp này mặc dù chỉ để xóa nội dung trong *dictionary* bên trái của biểu thức.

Ví dụ 4.66

| Mã lệnh | Kết quả |
|---|--|
| <code>dict1 = {'A': 1, 'B': 2, 'C': 3}
dict2 = {'C': 4, 'D': 5, 'E': 6}
dict1=dict2.update()
print('dict1=', dict1)
print('dict2=', dict2)</code> | <code>dict1= None
dict2= {'C': 4, 'D': 5, 'E': 6}</code> |

- Trường hợp 2: `update(other)`: *dictionary* đóng vai trò *other* sẽ giữ nguyên, *dictionary* được *update* sẽ nhận thêm toàn bộ giá trị của *dictionary other*. Do đó những *item* có *key* trùng nhau trong *dictionary* được *update* sẽ được thay thế bằng *item* của *dictionary other*.

Ví dụ 4.67

| Mã lệnh | Kết quả |
|---|---|
| <pre>dict1 = {'A': 1, 'B': 2, 'C': 3} dict2 = {'C': 4, 'D': 5, 'E': 6} dict1.update(dict2) print('dict1=', dict1)</pre> | <pre>dict1= {'A':1, 'B':2, 'C':4, 'D':5, 'E':6}</pre> |
| <pre>print('dict2=', dict2)</pre> | <pre>dict2= {'C': 4, 'D': 5, 'E': 6}</pre> |

4.4.10.2. Single Expression

- Cú pháp: `resultDict = dict(dict1, **dict2)`
- Giải thích: tạo ra 1 *dictionary* mới với nội dung có trong các *dictionary* thành phần. Tuy nhiên, nếu có những *items* với *key* trùng nhau, thì kết quả sẽ nhận *item* của *dictionary* có 2 dấu hoa thị (`**`) đi trước.
- Ví dụ 4.68

| Mã lệnh | Kết quả |
|---|---|
| <pre>dict1 = {'A': 1, 'B': 2, 'C': 3} dict2 = {'C': 4, 'D': 5, 'E': 6} result = dict(dict1, **dict2) print('dict1=', dict1) print('dict2=', dict2) print('result=', result)</pre> | <pre>dict1={'A': 1, 'B': 2, 'C': 3} dict2={'C': 4, 'D': 5, 'E': 6} result={'A':1, 'B':2, 'C':4, 'D':5, 'E':6}</pre> |

4.4.10.3. Retaining Key Values

- Ý tưởng: sử dụng
 - Hàm `defaultdict` (trong module `collections`) để tạo 1 *dictionary* mới (rỗng) với *value* không phải là 1 giá trị mà là 1 *list* các giá trị.
 - Hàm `chain` (trong module `itertools`) để kết nối các *dictionary*.
- Kết quả thực hiện:
 - Đối với những *items* với *key* trùng nhau, các *value* đều được giữ lại và đưa vào chung *list* kết quả.
 - Các *dictionary* tham gia không bị ảnh hưởng.
- Ví dụ 4.69

| Mã lệnh | Kết quả |
|---|--|
| <pre>from itertools import chain from collections import defaultdict dict1 = {'A': 1, 'B': 2, 'C': 3} dict2 = {'C': 4, 'D': 5, 'E': 6} result = defaultdict(list) # kiểm tra nội dung result mới tạo print('result_defaultdict=', dict(result)) for k, v in chain(dict1.items(), dict2.items()): result[k].append(v) # kiểm tra nội dung result sau mỗi lần lặp print('result_in_for=', dict(result))</pre> | <pre>result_defaultdict= {} # kết quả sau mỗi lần lặp result= {'A':[1]} result= {'A':[1], 'B':[2]} result= {'A':[1], 'B':[2], 'C':[3]} result= {'A':[1], 'B':[2], 'C':[3], 'D':[5], 'E':[6]}</pre> |

| | |
|--|--|
| | 'C':[3,4]}
result= {'A':[1], 'B':[2],
'C':[3,4], 'D': [5]}
result= {'A':[1], 'B':[2],
'C':[3,4], 'D':[5], 'E':[6]} |
| # kết quả thực hiện
print('dict1=',dict1) | dict1= {'A': 1, 'B': 2,
'C': 3} |
| print('dict2=',dict2) | dict2= {'C': 4, 'D': 5,
'E': 6} |
| print('result_dict=',result) | result= defaultdict(<class
'list'>, {'A':[1],
'B':[2], 'C':[3, 4],
'D':[5], 'E':[6]}) |

- Ghi chú: có thể không dùng *chain* như ở trên và thay bằng 2 lệnh *for* như sau:

| Mã lệnh cũ | Mã lệnh mới |
|---|--|
| for k, v in chain(dict1.items(), dict2.items()) :

result[k].append(v) | for k, v in dict1.items():
result[k].append(v)
for k, v in dict2.items():
result[k].append(v) |

4.4.11. Sort trên dictionary

Sử dụng hàm *sorted(iterableName [, key=])*

Ví dụ 4.70

| Mã lệnh | Kết quả |
|---|---|
| data = {3:'b', 2:'a', 1: 'c' }
print('Dictionary ban đầu:', data) | Dictionary ban đầu: {3: 'b', 2: 'a', 1: 'c'} |
| lst1=sorted(data.values())
print('List chứa key đã sort:', lst1) | List chứa key đã sort: ['a', 'b', 'c'] |
| lst2=sorted(data, key=data.get)
print('Sau khi sort theo key, list chứa value:',lst2) | Sau khi sort theo key, list chứa value: [2, 3, 1] |
| '''Sắp xếp dict theo key, chuyển kết quả thành 1 list chứa các tuple với thành phần mỗi tuple là 1 value:key'''
#sort by key
lst3 = sorted(data.items(), key=lambda x:x[1])
print('lst3 (sort by key):', lst3)
print('or lst3:')
for item in lst3:
print (item[1],':',item[0]) | lst3 (sort by key): [(2, 'a'), (3, 'b'), (1, 'c')]
or lst3:
a : 2
b : 3
c : 1 |
| '''Sắp xếp dict theo value, chuyển kết quả thành 1 list chứa các tuple với thành phần mỗi tuple là 1 value:key'''
#sort by name
lst4=sorted(data.items(), key=lambda x:x[0])
print('lst4 (sort by value):', lst4)
print('or lst4:')
for item in lst4:
print (item[1],':',item[0]) | lst4 (sort by value): [(1, 'c'), (2, 'a'), (3, 'b')]
or lst4:
c : 1
a : 2
b : 3 |

4.4.12. Một số hàm hỗ trợ việc xử lý trên dictionary

- (i)- **`len(Tên_dictionary)`** : Trả về kích thước (số lượng phần tử) của *dictionary*.
- (ii)- **`str(Tên_dictionary)`** : Trả về chuỗi chứa nội dung của *dictionary*.

Ví dụ 4.71

| Mã lệnh | Kết quả |
|--|---|
| <code>mydict = {3: 'three', 1: 'one', 2: 'two'}</code> | <code>my dict= {3: 'three', 1: 'one', 2: 'two'}</code> |
| <code>print ("my dict= ", mydict)</code> | |
| <code>print ("length= ", len(mydict))</code> | <code>length= 3</code> |
| <code>mylist = str(mydict).split(',')
for item in mylist:
 print (item)</code> | <code>{3: 'three'
1: 'one'
2: 'two'}</code> |
| <code>for k,v in mydict.items():
 print('Key='+ str(k) + ' and value= ' + v)</code> | <code>Key=3 and value= three
Key=1 and value= one
Key=2 and value= two</code> |
| <code>#phải dùng str(k) vì k có kiểu dữ liệu là số</code> | |

4.4.13. Một số phương thức của đối tượng dictionary

- (i)- **`Tên_dictionary.copy()`** : Trả về một *dictionary* mới với nội dung được sao chép từ *dictionary* hiện tại.
- (ii)- **`Tên_dictionary.values()`** : Trả về một *list* chứa các *value* đang có trong *dictionary*. Thường được dùng thêm 1 trong các hàm *list* / *tuple* / *set* để chuyển kết quả thành 1 đối tượng *list/tuple/set* tương ứng.
- (iii)- **`Tên_dictionary.fromkeys(seq[,v])`** : Trả về *dictionary* mới với danh sách các *key* lấy từ *seq* và nếu có truyền *value* thì lấy đó làm giá trị cho các phần tử, ngược lại (mặc định) là *None*.
- (iv)- **`Tên_dictionary.items()`** : Trả về danh sách các bộ *tuple (key, value)* của *dictionary*. Thường được dùng thêm 1 trong các hàm *list* / *tuple* / *set* để chuyển kết quả thành 1 đối tượng *list/tuple/set* tương ứng.
- (v)- **`Tên_dictionary.get(key[,d])`** Trả về giá trị của *key*, nếu *key* không tồn tại, trả về *d*. (default là *None*).
- (vi)- **`Tên_dictionary.has_key(key)`** kiểm tra một *key* có tồn tại trong đối tượng hay không?
- (vii)- **`Tên_dictionary.keys()`** Trả về một *list* chứa các *key* trong *dictionary*.
- (viii)- **`Tên_dictionary.setdefault(key[,d])`** Nếu *key* tồn tại trả về *value* tương ứng. Ngược lại, nếu không tồn tại sẽ thêm *key* với *value* là *d* và trả về *d* (default là *None*).
- (ix)- **`Tên_dictionary.update([other])`** Cập nhật *dictionary* với cặp *key : value* từ *other*, ghi đè lên các *key* đã có.
- (x)- **`Tên_dictionary.values()`** Trả về một *list* chứa các *value* đang có trong *dictionary*.

Ví dụ 4.72

| Mã lệnh | Kết quả |
|--|---|
| <code>mydict = {1: 'mot', 2: 'hai', 3: 'ba',
4: 'bon'}</code> | |
| <code>dict_copy = mydict.copy()
print ("Dict copy= ", dict_copy)</code> | <code>Dict copy= {1: 'mot', 2: 'hai', 3: 'ba', 4: 'bon'}</code> |
| <code>mylist = list(mydict.values())
print ("mylist= ", mylist)</code> | <code>mylist= ['mot', 'hai', 'ba', 'bon']</code> |
| <code>mylist = ['one', 'two', 'three']
mydict = dict.fromkeys(mylist)
print("my dict= ",mydict)</code> | <code>my dict= {'one': None, 'two': None, 'three': None}</code> |


```
mytuple = list(mydict.items())
print(mytuple)
```

```
[(1, 'mot'), (2, 'hai'), (3, 'ba'), (4, 'bon')]
```

4.5. Set

- Trong *Python*, *set* là class đại diện cho khái niệm toán học của 1 tập hợp, nghĩa là *set* gồm tập hợp các phần tử duy nhất (không trùng lặp) và có thể thực hiện các phép toán về tập hợp như: hợp, giao,
- Đặc điểm của *set*:
 - Không giới hạn số lượng phần tử.
 - Có thể thêm hoặc xóa phần tử.
 - Có thể chứa biến thuộc nhiều kiểu dữ liệu khác nhau, nhưng không thể chứa phần tử có thể thay đổi được như *list*, *set* hay *dictionary*.
 - Được tối ưu hóa trong việc kiểm tra xem một phần tử cụ thể có được chứa trong tập hợp hay không nhờ dựa trên cấu trúc dữ liệu bảng băm (*hash table*).

4.5.1. Khai báo và gán giá trị cho set

- Tạo bằng phép gán: *set* được tạo bằng cách đặt tất cả các phần tử trong dấu ngoặc nhọn { } và phân tách nhau bằng dấu phẩy (,).
- Tạo set bằng cách sử dụng hàm *set()*.

4.5.2. Xuất (in) nội dung set ra màn hình

- Có thể sử dụng dấu hoa thị (*) trước tên *set* để bỏ qua cặp ngoặc nhọn ({}) khi in.
- Ví dụ 4.73

| | Mã lệnh | Kết quả |
|--------|---|-----------------|
| | <pre>myset1 = set() # myset1 là tập hợp rỗng myset2="Sai gon" myset3 = set({1,5,3,4,2})</pre> | |
| Cách 1 | print(myset1) | set() |
| | print(myset2) | Sai gon |
| | print(myset3) | {1, 2, 3, 4, 5} |
| Cách 2 | print(myset1) | #rỗng |
| | print(*myset2) | {1, 2, 3, 4, 5} |
| | print(*myset3) | 1 2 3 4 5 |

4.5.3. Duyệt qua các phần tử của set

- Vì *set* là tập hợp các phần tử không có thứ tự nên:
 - *index* không có ý nghĩa với *set*.
 - Toán tử cắt đoạn `:` sẽ không làm việc trên *set*.
- Ví dụ 4.74

| Mã lệnh | Kết quả |
|--|---|
| myset={1, 5, 3, 4, 2}
print(myset[1]) | TypeError: 'set' object does not support indexing |
| print(myset[1:3]) | TypeError: 'set' object is not subscriptable |

- Sử dụng vòng lặp *for* để lặp qua các phần tử của *set*.

| Mã lệnh | Kết quả |
|---|---|
| <pre>myset="Sai gon" for x in myset: print(x)</pre> | <p>S</p> <p>a</p> <p>i</p> <p>g</p> <p>o</p> <p>n</p> |

| | |
|--|---|
| | i |
| | g |
| | o |
| | n |

4.5.4. Sao chép set

- Cách 1: gán trực tiếp qua toán tử bằng (=).
- Cách 2: sử dụng phương thức copy của set nguồn.

Ví dụ 4.75

| Mã lệnh | Kết quả |
|--|-----------|
| myset = {1, 6, 4}
set2 = myset
print(set2) | {1, 4, 6} |
| set3 = myset.copy()
print(set3) | {1, 4, 6} |

4.5.5. Thêm phần tử vào set

- **add()**: thêm một phần tử vào set.
- **update()**: thêm nhiều phần tử vào set trong cùng 1 lần. **update()** có thể nhận *tuple*, *list*, *string* và **set** làm đối số. Trong mọi trường hợp, set chỉ chứa giá trị duy nhất (không trùng), do đó các giá trị được add hoặc update nhưng bị trùng sẽ tự động bị loại bỏ.

Ví dụ 4.76

| Mã lệnh | Kết quả |
|---|---|
| myset={5}
myset.add(8)
print(myset) | {8, 5} |
| mystr="Sai gon"
mytuple=(4.12, 3, 'a')
mylist=["HCM", 5, 3]
myset.update(mystr, mytuple, mylist)
print(myset) | {'S', ' ', 'a', 5, 3, 8, 'n', 'HCM', 4.12, 'g', 'i', 'o'} |

4.5.6. Kiểm tra phần tử có tồn tại trong set hay không?

- Sử dụng toán tử thành phần **in** hoặc **not in**.

| Mã lệnh | Kết quả |
|---|---------|
| myset = {1, 6, 4, 3, 5}
print(5 in myset) | True |
| print(5 not in myset) | False |

4.5.7. Xóa phần tử khỏi set

- **discard()** và **remove()**: xóa phần tử cụ thể khỏi set. Khi phần tử cần xóa không tồn tại trong set thì **discard()** không làm gì cả, còn **remove()** sẽ báo lỗi.

| Mã lệnh | Kết quả |
|---|-----------------|
| myset = {1, 6, 4, 3, 5}
print(myset) | {1, 3, 4, 5, 6} |
| myset.discard(4)
print(myset) | {1, 3, 5, 6} |
| myset.remove(6)
print(myset) | {1, 3, 5} |

| | |
|------------------------------|--------------------------|
| <code>myset.remove(2)</code> | <code>KeyError: 2</code> |
|------------------------------|--------------------------|

- **pop()**: do bản chất của *set* là không có thứ tự, vì vậy phương thức này sẽ xóa ngẫu nhiên 1 phần tử trong *set*. Phương thức này sẽ gây lỗi khi *set* rỗng.

| Mã lệnh | Kết quả |
|---|---------|
| <code>myset = {1, 6, 4, 3, 5}</code>
<code>print(myset.pop())</code> | 1 |

- **clear()**: xóa rỗng các thành phần trong *set*.

| Mã lệnh | Kết quả |
|---|-----------------|
| <code>myset = {1, 6, 4, 3, 5}</code>
<code>print(myset)</code> | {1, 3, 4, 5, 6} |
| <code>myset.clear()</code>
<code>print(myset)</code> | set() |

- **del(tên_set)**: xóa *set* ra khỏi bộ nhớ.

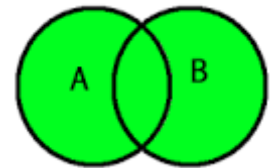
| Mã lệnh | Kết quả |
|--|---|
| <code>myset = {1, 6, 4, 3, 5}</code>
<code>del(myset)</code>
<code>print(myset)</code> | <code>NameError: name 'myset' is not defined</code> |

4.5.8. Các toán tử và phương thức trên set

- Các phép toán trên set sau đây sẽ cho kết quả là 1 *set* mới
- Cho 2 *set* sau: `setA = {1, 2, 3, 4, 5}`
`setB = {4, 5, 6, 7, 8}`

4.5.8.1. Phép hợp

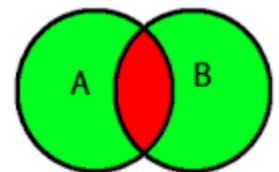
- Trả về *set* mới là hợp của 2 *set*.
- Toán tử `|`
- Phương thức **`union()`**.
- Ví dụ 4.77



| Mã lệnh | Kết quả |
|--------------------------------------|--------------------------|
| <code>print(setA setB)</code> | {1, 2, 3, 4, 5, 6, 7, 8} |
| <code>print(setA.union(setB))</code> | {1, 2, 3, 4, 5, 6, 7, 8} |
| <code>print(setA)</code> | {1, 2, 3, 4, 5} |

4.5.8.2. Phép giao

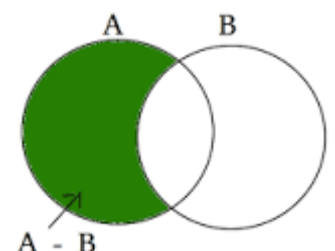
- Trả về *set* mới chứa phần tử chung (phần giao) của 2 *set*.
- Toán tử `&`
- Phương thức **`intersection()`**.
- Ví dụ 4.78



| Mã lệnh | Kết quả |
|---|---------|
| <code>print(setA & setB)</code> | {4, 5} |
| <code>print(setB.intersection(setA))</code> | {4, 5} |

4.5.8.3. Phép hiệu (trừ)

- **Hiệu của A và B** ($= A - B$) sẽ trả về *set* mới là tập hợp phần tử chỉ có trong A và không có trong B.
- Toán tử trừ `-`
- Phương thức **`difference()`**.

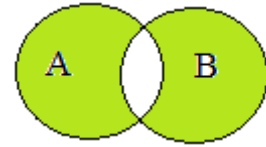


- Ví dụ 4.79

| Mã lệnh | Kết quả |
|---|-----------|
| <code>print(setA - setB)</code> | {1, 2, 3} |
| <code>print(setB.difference(setA))</code> | {8, 6, 7} |

4.5.8.4. Phép bù

- Bù của A và B sẽ trả về *set* mới là tập hợp những phần tử có trong A và B nhưng không phải phần tử chung của hai tập hợp này.
- Toán tử \wedge
- Phương thức *symmetric_difference()*.
- Ví dụ 4.80



| Mã lệnh | Kết quả |
|---|--------------------|
| <code>print(setA ^ setB)</code> | {1, 2, 3, 6, 7, 8} |
| <code>print(setB.symmetric_difference(setA))</code> | {1, 2, 3, 6, 7, 8} |

4.5.8.5. Một số phương thức khác

| Phương thức | Mô tả |
|--------------------------------------|---|
| <i>intersection_update()</i> | Cập nhật <i>set</i> với phần tử chung của chính <i>set</i> đó và <i>set</i> khác. |
| <i>isdisjoint()</i> | Trả về <i>True</i> nếu 2 <i>set</i> không có phần tử chung. |
| <i>issubset()</i> | Trả về <i>True</i> nếu <i>set</i> khác chứa <i>set</i> này. |
| <i>issuperset()</i> | Trả về <i>True</i> nếu <i>set</i> này chứa <i>set</i> khác. |
| <i>symmetric_difference_update()</i> | Cập nhật <i>set</i> với những phần tử khác nhau của chính <i>set</i> đó và <i>set</i> khác. |

4.5.9. Một số hàm thường dùng trên set

- *len(tên_set)* : đếm số lượng phần tử có trong *set*.
- *max(tên_set)*: trả về phần tử có giá trị lớn nhất trong *set*. Các phần tử trong *set* phải cùng kiểu dữ liệu.
- *min(tên_set)* : trả về phần tử có giá trị nhỏ nhất trong *set*. Các phần tử trong *set* phải cùng kiểu dữ liệu.
- *sum(tên_set)*: tính tổng các phần tử có trong *set*. Chỉ sử dụng được trên chỉ chứa kiểu dữ liệu là số.
- *sorted(tên_set[,reverse=False/True])* : sắp xếp thứ tự trong *set*. Mặc định là sắp xếp tăng dần (*reverse=False*).

4.5.10. Frozenset

- *Frozenset* là một class, có đặc điểm của một *set*, nhưng phần tử trong đó không thể thay đổi được sau khi gán. Để dễ hình dung thì *tuple* là *list* bất biến còn *frozenset* là *set* bất biến.
- Các *set* có thể thay đổi được nhưng không thể băm (*hash*) được, do đó không thể sử dụng *set* để làm *key* cho *dictionary*. Nhưng *frozenset* có thể băm được nên có thể dùng như các *key* cho *dictionary*.
- *Frozenset* có thể tạo bằng hàm *frozenset()*.
- *Frozenset* hỗ trợ các phương thức như *copy()*, *difference()*, *intersection()*, *isdisjoint()*, *issubset()*, *issuperset()*, *symmetric_difference()* và *union()*.
- Vì không thể thay đổi nên phương thức *add()* hay *remove()* không sử dụng được trên *frozenset*.

4.6. Một số phương thức và hàm hỗ trợ việc xử lý trên đối tượng dạng danh sách

(i).- *accumulate()*

- Hàm *accumulate* cũng được dùng để tính tổng của các phần tử trong *sequence()*.
- Sự khác biệt giữa 2 hàm *accumulate()* và *reduce()*.

| | <i>reduce()</i> | <i>accumulate()</i> |
|---|---|--|
| Được định nghĩa trong module | functools | itertools |
| Kết quả trả về | giá trị tổng cuối cùng | trả về một danh sách chứa kết quả trung gian. Số cuối cùng của danh sách được trả về là giá trị tổng của danh sách |
| Thứ tự các đối số: + đối số 1
+ đối số | function_name
sequence | sequence
function_name |

- Ví dụ 4.81: phối hợp giữa hàm *reduce* và *anonymous function* để tính tổng và tìm số lớn nhất có trong *list lst*.

| Mã lệnh | Kết quả |
|--|--|
| <pre>import itertools import functools lst = [1, 3, 4, 10, 4] print('List ban dau:', lst) print("Kết quả tính tổng lst bằng hàm accumulate: ", end="") print(list(itertools.accumulate(lst, lambda x, y: x + y))) print("Kết quả tính tổng lst bằng hàm reduce: ", end="") print(functools.reduce(lambda x, y: x + y, lst))</pre> | <p>List ban dau: [1, 3, 4, 10, 4]</p> <p>Kết quả tính tổng lst bằng hàm accumulate: [1, 4, 8, 18, 22]</p> <p>Kết quả tính tổng lst bằng hàm reduce: 22</p> |

- Ví dụ 4.82: Sử dụng phương thức *Counter* trong module *collections*.

| Mã lệnh | Kết quả |
|--|---------|
| <pre>import collections num = [2,2,4,6,6,8,6,10,4] print(sum(collections.Counter(num).values()))</pre> | 9 |

(ii).- *all()*

- Trả về *True* khi tất cả các phần tử trong *iterable* đều có kết quả là *True*.
- Ví dụ 4.83: xét xem chuỗi *s = 'Sai Gon'* có chứa toàn ký tự in thường hay không?

| Mã lệnh | Kết quả |
|---|---|
| <pre>s = 'Sai Gon' print(all(c.islower() for c in s))</pre> | False |
| <pre>s = 'sai gon' print(all(c.islower() for c in s))</pre> | False '''vì ký tự khoảng trắng ở giữa không có trong bộ alphabet''' |
| <pre>s = 'saigon' print(all(c.islower() for c in s))</pre> | True |

- Ví dụ 4.84: xét xem các giá trị trong *list* có cùng thỏa 1 điều kiện nào đó (dùng *comprehension*)

| Mã lệnh | Kết quả |
|---|--------------------------|
| <pre>num = [2, 3, 4] print(all(x > 1 for x in num)) print(all(x >= 4 for x in num))</pre> | <p>True</p> <p>False</p> |

(iii).-**any()**

- Trả về *True* khi bất kỳ phần tử nào của *iterable* là *True*
- Ví dụ 4.85: xét xem chuỗi `s = 'Sai Gon'` có chứa ít nhất 1 ký tự in thường hay không?

| Mã lệnh | Kết quả |
|---|---------|
| <pre>s = 'Sai Gon' print(any(c.islower() for c in s))</pre> | True |

(iv).-**count()**

- Đếm số lần xuất hiện của 1 giá trị trong list
- Cú pháp: **listName.count(value/variableName)**
- Ví dụ 4.86

| Mã lệnh | Kết quả |
|--|----------------------|
| <pre>lst=[2,3,4,2,2] x=2 print('Số %d xuất hiện %d lần' %(x,lst.count(2)))</pre> | Số 2 xuất hiện 3 lần |

(v).- **enumerate()**

- Thêm vào một bộ đếm vào trước mỗi *iterable* và trả về kết quả dưới dạng đối tượng liệt kê (*enumerate object*). Các đối tượng *enumerate* này sau đó có thể được sử dụng trực tiếp trong các vòng lặp hoặc được chuyển đổi thành một danh sách, một *tuple* bằng phương thức *list()* và *tuple()*.
- Cú pháp

enumerate(iterable, start=0)

Trong đó::

- **iterable**: chuỗi, list, tuple, iterator hoặc bất cứ đối tượng hỗ trợ iteration nào.
- **start**: **enumerate()** sẽ bắt đầu bộ đếm từ số này. Mặc định, **start=0**.
- Ví dụ 4.87

| Mã lệnh | Kết quả |
|---|---|
| <pre>TinhTP = ['Hà Nội', 'Huế', 'Sài Gòn'] '''Chuyển list thành enum, phần tử đầu tiên được đánh số từ 0''' enum_TinhTP = enumerate(TinhTP) print('Type of enum_TinhTP:', type(enum_TinhTP)) # chuyển đổi enum thành list print(list(enum_TinhTP))</pre> | Type of
enum_TinhTP:
<class
'enumerate'>

[(0, 'Hà Nội'),
(1, 'Huế'), (2,
'Sài Gòn')] |
| <pre>'''Chuyển list thành enum, phần tử đầu tiên được đánh số từ 5''' enum_TinhTP = enumerate(TinhTP, 5) print(list(enum_TinhTP)) #Các cách duyệt trên enumerate for item in enumerate(TinhTP): print(item) for count, item in enumerate(TinhTP): print(count, item) for count, item in enumerate(TinhTP, 10): print(count, item)</pre> | [(5, 'Hà Nội'),
(6, 'Huế'), (7,
'Sài Gòn')]

(0, 'Hà Nội')
(1, 'Huế')
(2, 'Sài Gòn')

0 Hà Nội
1 Huế
2 Sài Gòn

10 Hà Nội
11 Huế
12 Sài Gòn |

(vi).-**filter()**

- Dùng để lọc các *item* trong 1 *iterable_object*, hàm tạo ra 1 *filter object* chứa các *item* thỏa điều kiện của *function_name*.
- **Cú pháp:** **filter(function_name, sequence)**
- Ví dụ 4.88: phối hợp giữa *filter* và *anonymous function* để tạo ra 1 *list* mới từ *list* cũ bằng cách lọc ra những số chia chắn cho 13.

| Mã lệnh | Kết quả |
|--|---------------|
| <pre>lst = [12, 65, 54, 39, 102, 339, 221, 50, 70,] result = list(filter(lambda x: (x % 13 == 0), lst)) print(result)</pre> | [65, 39, 221] |

(vii).- **iter()**

- Hàm *iter()* được sử dụng để chuyển đổi 1 danh sách hoặc 1 đối tượng dạng danh sách (*iterable Object*) thành 1 đối tượng lặp (*iterator Object*), *iterator Object* cho phép duyệt qua các phần tử có trong danh sách ban đầu như khi sử dụng các lệnh lặp *for/while*.
- Hàm *iter()* sử dụng *next()* để truy cập và trả về các giá trị.
- **Cú pháp:** `iter(obj [, sentinel])`

Giải thích:

- *obj*: Đối tượng sẽ được chuyển đổi thành *iterator object* (thường là một *iterable object*).
- *sentinel*: (tùy chọn). Nếu đối tượng là một đối tượng có thể gọi thì quá trình lặp sẽ dừng khi giá trị trả về giống với *sentinel*

- Ví dụ:

| Mã lệnh | Kết quả |
|--|--|
| <pre>lst=list(range(1,6)) obj=iter(lst) print("Duyệt lst cách 1:", end=' ') print(next(obj), end=' ') print(next(obj), end=' ') print(next(obj), end=' ') print(next(obj), end=' ') print(next(obj), end=' ') print(next(obj), end=' ') print("\nDuyệt lst cách 2:", end=' ') #phải tạo lại iterator object khác trước khi duyệt obj=iter(lst) while True: k=next(obj) print(k, end=' ') if k==5: # nếu thay giá trị khác>5 => lặp vô hạn break</pre> | Duyệt lst cách 1: 1
2 3 4 5

Duyệt lst cách 2: 1
2 3 4 5 |
| <pre>print("\nDuyệt lst cách 3:", end=' ') #phải tạo lại iterator object khác trước khi duyệt obj=iter(lst) for i in range(1,6): print(next(obj), end=' ') </pre> | Duyệt lst cách 3: 1
2 3 4 5 |

(viii).- **map()**⁵

- Hàm **map(functionName, iterableObject)**: duyệt qua tất cả các phần tử của *iterableObject* và áp dụng hàm *functionName* lên các phần tử này. Từ python 3 trở đi, kết quả trả về của hàm *map* là một *map object* (trong python 2, hàm *map* trả về kết quả là 1 *list*).
- Nếu có nhiều hơn một *sequence* được cung cấp thì phương thức sẽ được gọi kết hợp cho từng phần tử của các *sequence*.
- Nếu 1 *sequence* ngắn hơn *sequence* khác thì *sequence* kết quả sẽ có số phần tử bằng với *sequence* ngắn.
- **Cú pháp: `map(function_name, sequence1[, sequence2, ...])`**
- Ví dụ 4.89: Tạo 1 *list* mới từ 3 *list* đã có sao cho giá trị các phần tử trong *list* mới là tổng từng cặp số trong mỗi *list*. Chú ý các *list* tham gia có thể có chiều dài khác nhau.

| Mã lệnh | Kết quả |
|---|---------|
| <pre>lst1 = [1, 2, 3] lst2 = [4, 5, 6, 7] lst3 = [0, 1] result = list(map(lambda x, y, z: x + y+z, lst1, lst2, lst3)) print(result)</pre> | [5, 8] |

- Ví dụ 4.90: **map()** có thể chia 1 chuỗi thành 1 *list* mới gồm nhiều phần tử và mỗi phần tử là 1 ký tự trong chuỗi ban đầu.

| Mã lệnh | Kết quả |
|---|--------------------------------------|
| <pre>lst = ['sai', 'gon'] resultList = list(map(list, lst)) print(resultList)</pre> | [['s', 'a', 'i'], ['g', 'o', 'n']] |

- Ví dụ 4.91: trong đoạn lệnh sau, giả sử người dùng nhập 3 5 \Rightarrow chương trình sẽ thực hiện gán *a*=3 và *b*=5. Do đó kết quả sẽ in ra: 3 + 5 = 8

```
a, b = map(int, input("Nhập 2 số nguyên cách nhau bởi
                        khoảng trắng").split())
print(a, '+', b, '=', a+b)
```

- Ví dụ 4.92: Tạo một *list* từ 1 *tuple* đang có sao cho giá trị các phần tử trong *list* là bình phương của các giá trị đang có trong *tuple*

| Mã lệnh | Kết quả |
|--|---------------|
| <pre>def square(i): return i*i myTuble = (1, 2, 3, 4) ResultMap = map(square, myTuble) # chuyển map object ResultMap thành list print(list(ResultMap))</pre> | [1, 4, 9, 16] |

- Ví dụ 4.93: Tương tự như ví dụ trên, nhưng sử dụng *lambda* để tạo một *list* từ 1 *tuple* đang có sao cho giá trị các phần tử trong *list* là bình phương của các giá trị đang có trong *tuple*

| Mã lệnh | Kết quả |
|--|---------------|
| <pre>myTuble = (1, 2, 3, 4) result = list(map(lambda x: x * x, myTuble)) print(result)</pre> | [1, 4, 9, 16] |

⁵ Xem thêm về ứng dụng hàm *map* trên *list object* ở phần 4.2.16.5

(ix).- **max & min(iterable, *, key, default)**

- Trả về phần tử có giá trị lớn(nhỏ) nhất có trong iterable hoặc trong các đối số truyền cho hàm. Nếu có nhiều phần tử cùng có giá trị lớn (hoặc nhỏ) nhất, hàm trả về mục đầu tiên trong số đó.

- **Cú pháp:**

```
max(iterable, *, key, default))
max(arg1, arg2, *args[, key])
min(iterable, *, key, default))
min(arg1, arg2, *args[, key])
```

Trong đó:

- **key:** nếu có, là hàm được áp dụng lên tất cả các thành phần có trong danh sách tham số, như các hàm như `str.lower`, `strupper`, `len`, ...
- **default:** (tùy chọn) Giá trị mặc định sẽ trả về nếu iterable trống.

- Ví dụ 4.94

| Mã lệnh | Kết quả |
|---|---|
| <pre># find largest item in the string print(max("abcDEF"))</pre> | c |
| <pre># find largest item with multiple arguments print(max(2, 1, 4, 3))</pre> | 4 |
| <pre># find largest item in the list print(max([2, 1, 4, 3]))</pre> | 4 |
| <pre># find largest item in the tuple print(max(("one", "two", "three")))</pre> | two |
| <pre>lst=['Sai Gon','Hue','Ha Noi'] print(max(lst))</pre> | Sai Gon |
| <pre>lst=['Sai Gon','Hue','Ha Noi'] print(min(lst))</pre> | Ha Noi |
| <pre># find largest item in the dict print(max({1: "one", 2: "two", 3: "three"}))</pre> | 3 |
| <pre># supressing the error with default value print(max([], default=0))</pre> | 0 |
| <pre># empty iterable causes ValueError print(max([]))</pre> | Traceback (most recent call last): File "<stdin>", line 1, in <module> ValueError: max() arg is an empty sequence |
| <pre># find largest item with multiple arguments print(max('a', 'b', 'c', 'D', 'E', 'F'))</pre> | c |
| <pre># find largest item with multiple arguments print(max('a', 'b', 'c', 'D', 'E', 'F', key=str.lower))</pre> | F |
| <pre># find largest item with multiple arguments print(max('Python','Java','Ruby','Java script'))</pre> | Ruby |
| <pre># find largest item with multiple arguments print(max('Python', 'Java', 'Ruby', 'Java script', key=str.len))</pre> | Java script |

(x).- **reduce()**

- Hàm `reduce (function_name, sequence)` được sử dụng để áp dụng hàm `function_name` cho tất cả các thành phần trong danh sách `sequence`. Kết quả trả về của hàm là 1 giá trị đơn.
- **Cú pháp:** `reduce(function_name, sequence) → value`

- Hàm này được định nghĩa trong module *functools*. Vì vậy cần *import* module này trước khi sử dụng.
- Ví dụ 4.95: phối hợp giữa hàm *reduce* và *anonymous function* để tính tổng và tìm số lớn nhất có trong *list* *lst*.

| Mã lệnh | Kết quả |
|---|---|
| <pre>import functools lst = [1, 3, 5, 6, 2] print('Cac so co trong lst:',lst)</pre> | Cac so co trong lst:
[1, 3, 5, 6, 2] |
| <pre># sử dụng hàm reduce để tính tổng của lst sum=functools.reduce(lambda a, b: a + b, lst) print("Tong cac so trong lst=",sum)</pre> | Tong cac so trong lst=
17 |
| <pre># sử dụng hàm reduce để tìm số lớn nhất trong lst print("So lon nhat trong lst la:", end="") print(functools.reduce(lambda a, b: a if a > b else b, lst))</pre> | So lon nhat trong lst
la:6 |

- Ví dụ 4.96: phối hợp giữa hàm *reduce* và các hàm trong *operator* để đạt được chức năng tương tự như với các hàm *lambda* và giúp cho mã dễ đọc hơn.

| Mã lệnh | Kết quả |
|---|--|
| <pre>import functools import operator lst1 = [1, 3, 5, 6, 2] lst2=["geeks", "for", "geeks"]</pre> | |
| <pre>''' phối hợp reduce với hàm add trong operator để tính tổng lst ''' print("Tong cac so trong lst1= ", end="") print(functools.reduce(operator.add, lst1))</pre> | Tong cac so
trong lst1= 17 |
| <pre>'''phối hợp reduce với hàm mul trong operator để tính tích các số trong lst''' print("Tich cac so trong lst1= ", end="") print(functools.reduce(operator.mul, lst1))</pre> | Tich cac so
trong lst1= 180 |
| <pre># using reduce to concatenate string print("Ket qua noi cac phan tu trong lst2: ", end="") print(functools.reduce(operator.add, lst2))</pre> | Ket qua noi cac
phan tu trong
lst2:
geeksforgeeks |

(xi).-sorted (Iterator[,key])

- Nhận tham số là *Iterator*, trả về *list* đã được sắp xếp tăng dần
- Cú pháp: **sorted (iterator[,key])**
- Ví dụ 4.97

| Mã lệnh | Kết quả |
|--|---|
| <pre>S='Sai Gon' sortList=sorted(S) print(sortList)</pre> | [' ', 'G', 'S', 'a', 'i',
'n', 'o'] |
| <pre>sortList=sorted(S, key=str.upper) print(sortList)</pre> | [' ', 'a', 'G', 'i', 'n',
'o', 'S'] |
| <pre>myList=[9, 3.14, 7] sortList=sorted(myList) print(sortList)</pre> | [3.14, 7, 9] |
| <pre>myList=['Ha noi', 9, 3.14, '"Hue', 'Sai gon'] sortList=sorted(myList) print(sortList)</pre> | TypeError: '<' not
supported between |

instances of 'int' and 'str'

(xii).- **sum()**

- Trả về tổng của tất cả các phần tử trong *list*.
- Ví dụ 4.98

| Mã lệnh | Kết quả |
|---|---------|
| <pre>s = sum([10, 20, 30]) print("\nSum of the container: ", s)</pre> | 60 |

(xiii).- **zip()**

- Công dụng
 - (i)- Hàm **zip()** trong *Python* trả về một đối tượng *zip*, là một *iterator* dạng danh sách các *tuple* kết hợp các phần tử từ các *iterator* (được tạo thành từ các *iterable*) khác.

Các *tuple* trong đối tượng *zip* cần được chuyển đổi thành các dạng danh sách khác như *list*, *set*, ... trước khi sử dụng.

- (ii)- Giải nén danh sách bằng cách sử dụng toán tử *** cùng với *zip()*.
- Cú pháp

zip([*]*iterable*)

Trong đó:

- *iterable*: các *iterable* được tích hợp sẵn (như *list*, *string*, *dict*) hoặc *iterable* do người dùng khai báo (được tạo thành từ phương thức `__iter__`)
- Giá trị trả về từ *zip()*:
 - Nếu không có tham số nào được truyền, *zip()* trả về một *iterator* rỗng.
 - Nếu tham số được truyền chỉ có duy nhất một *iterable*, *zip()* trả về *tuple* có 1 phần tử.
 - Nếu tham số được truyền có nhiều *iterable* và độ dài của các *iterable* không bằng nhau, *zip* sẽ tạo các *tuple* có độ dài bằng với số *iterable* nhỏ nhất.
- Một số ví dụ
 - Ví dụ 4.99 Cách hoạt động của *zip()*

| Mã lệnh | Kết quả |
|---|---|
| <pre>numberList = [1, 2, 3] strList = ['one', 'two', 'three'] # Không truyền iterable result1 = zip() print('1.- Không truyền tham số, zip object kết quả là:', result1)</pre> | <p>1.- Không truyền tham số, zip object kết quả là:
<zip object at 0x0000002C2A3DEE88></p> |
| <pre># Truyền 1 iterator result2 = zip(strList) print('2.- Truyền 1 list làm tham số, zip object kết quả là:', result2)</pre> | <p>2.- Truyền 1 list làm tham số, zip object kết quả là: <zip object at 0x0000002C2A3DED48></p> |
| <pre># Truyền 1 iterator, chuyển đổi iterator thành list resultList = list(zip(strList)) print('3.- Truyền 1 list làm tham số, rồi chuyển zip kết quả thành list:', resultList)</pre> | <p>3.- Truyền 1 list làm tham số, rồi chuyển zip kết quả thành list:
[('one',), ('two',), ('three',)]</p> |
| <pre># Truyền 2 iterator result3 = zip(numberList, strList) print('4.- Truyền 2 list làm tham số, zip</pre> | <p>4.- Truyền 2 list làm tham số, zip object kết</p> |

| | |
|---|---|
| object kết quả là:', result3) | quả là: <zip object at 0x0000002C2A3DE788> |
| # Truyền 2 iterator, chuyển đổi iterator thành set
print('5.- Truyền 2 list làm tham số, rồi chuyển zip kết quả thành set:', set(result3)) | 5.- Truyền 2 list làm tham số, rồi chuyển zip kết quả thành set: {(1, 'one'), (2, 'two'), (3, 'three')} |

- Ví dụ 4.100 Các iterator có số phần tử khác nhau

| Mã lệnh | Kết quả |
|---|---|
| <pre> numbersList = [1, 2, 3] strList = ['one', 'two'] numbersTuple = ('mot', 'hai', 'ba', 'bon') ''' len(numbersList)=3, len(numbersTuple)=4 => len(zip object)=3 ''' result = zip(numbersList, numbersTuple) # Chuyển đổi thành set resultSet = set(result) print('Chuyển zip object thành set', resultSet) </pre> | Chuyển zip object thành set {(1, 'mot'), (2, 'hai'), (3, 'ba')} |
| <pre> ''' len(numbersList)=3, len(strList)=2, len(numbersTuple)=4 => len(zip object)=2 ''' result = zip(numbersList, strList, numbersTuple) # Chuyển đổi thành list resultList = list(result) print('Chuyển zip object thành list', resultList) </pre> | Chuyển zip object thành list [(1, 'one', 'mot'), (2, 'two', 'hai')] |

- Ví dụ 4.101: Giải nén danh sách bằng cách sử dụng toán tử * cùng với zip()

| Mã lệnh | Kết quả |
|--|---|
| <pre> Clist = ['x', 'y', 'z'] Vlist = [3, 4, 5, 0, 9] resultList = list(zip(Clist, Vlist)) print('Zip object sau khi chuyển thành list:', resultList) c, v = zip(*resultList) print('Sử dụng Zip(*) để tách các thành phần trong list:', resultList) print('Coordinate =', c) print('Value =', v) </pre> | <p>Zip object sau khi chuyển thành list: [('x', 3), ('y', 4), ('z', 5)]</p> <p>Sử dụng Zip(*) để tách các thành phần trong list: Coordinate = ('x', 'y', 'z')
Value = (3, 4, 5)</p> |

4.7. Xây dựng hàm ẩn danh (Anonymous function - lambda) cho sequence data type

4.7.1. Giới thiệu

- Sử dụng *lambda function* trên các đối tượng thuộc kiểu dữ liệu đơn chỉ cho ra 1 kết quả đơn.
- Trong thực tế, thao tác phổ biến của người lập trình thường thực hiện 1 tác vụ nào đó cho từng mục trong các đối tượng dạng *iterable* (string, list, ...)

- Trong python, để sử dụng *lambda* trên *iterable object*, người lập trình thường sử dụng 1 trong 2 hàm là *map* hoặc *filter*.
- Nhắc lại:
 - Hàm **map**(*functionName*, *iterableObject*): duyệt qua tất cả các phần tử của *iterableObject* và áp dụng hàm *functionName* lên các phần tử này. Từ python 3 trở đi, kết quả trả về của hàm *map* là một *map object* (trong python 2, hàm *map* trả về kết quả là 1 *list*).
 - Hàm **filter**(*function_name*, *sequence*): dùng để lọc các *item* trong 1 *iterable_object* thỏa điều kiện là *True*, hàm tạo ra 1 *filter object* chứa các *item* thỏa điều kiện của *function_name*.
- Số lượng phần tử trong tập kết quả:
 - Gọi *p* là số lượng phần tử có trong tập nguồn, và *q* là số lượng phần tử có trong tập kết quả.
 - Hàm *map* thường được dùng dưới dạng ánh xạ 1-1, nghĩa là $p = q$.
 - Hàm *filter* luôn có kết quả $p \geq q$.

4.7.2. Một số ví dụ về dùng 2 hàm *map* và *filter* trên *iterator object*

4.7.2.1. Ví dụ so sánh cách dùng 2 hàm *map* và *filter*

- **Ví dụ 4.102** minh họa số lượng phần tử của 2 tập kết quả bằng nhau khi sử dụng hàm *map* và *filter*: Cho list *L*=[1,2,3,4], sử dụng hàm *square* để tạo ra list mới (*lst*) chứa bình phương các phần tử có trong *L*.

| Sử dụng hàm thông thường | Sử dụng Anonymous function | Kết quả |
|---|---|----------------------|
| <pre>def square(x): return x**2</pre> | | |
| <pre>L=[1,2,3,4] lst=list(map(square, L)) print('map:',lst)</pre> | <pre>L=[1,2,3,4] lst=list(map(lambda x: x**2, L)) print('map:',lst)</pre> | map: [1, 4, 9, 16] |
| <pre>lst=list(filter(square, L)) print('filter:',lst)</pre> | <pre>lst=list(filter(lambda x: x**2, L)) print('filter:',lst)</pre> | filter: [1, 2, 3, 4] |

- **Ví dụ 4.103** minh họa số lượng phần tử của tập kết quả khi sử dụng hàm *map* nhiều hơn so với khi sử dụng hàm *filter*: Cho list *L*=[0, 1, 2, 3], tạo ra list mới (*lst*) chứa bình phương các phần tử có trong *L*.

| Sử dụng hàm thông thường | Sử dụng Anonymous function | Kết quả |
|--|---|---------------------|
| <pre>def square(x): return x**2</pre> | <pre>def square(x): return x**2</pre> | |
| <pre>L=[0, 1, 2, 3] lst=list(map(square, L)) print('map:',lst)</pre> | <pre>L=[0, 1, 2, 3] lst=list(map(lambda x: square(x), L)) print('map:',lst)</pre> | map: [0,1,4,9] |
| <pre>lst=list(filter(square,L)) print('filter:',lst)</pre> | <pre>lst=list(filter(lambda x: square(x),L)) print('filter:',lst)</pre> | filter: [1,2,3] (i) |

(i): giá trị 0 bị loại khỏi tập kết quả do 0 có giá trị tương đương với False.

- **Ví dụ 4.104** minh họa việc dựa vào mục đích mà người lập trình mong muốn về tập kết quả để chọn sử dụng hàm map hay filter: Cho list $L = [1, 3, 2, 5, 20, 21]$, tạo ra list mới (lst) với biểu thức trong lambda là $x \% 2 == 0$ (phần tử trong L là số chẵn).

| Sử dụng Anonymous function | Kết quả |
|--|---|
| <pre>L=[1, 3, 2, 5, 20, 21] lst=list(map(lambda x: x % 2 == 0, L)) print('map:',lst)</pre> | map: [False, False, True, False, True, False] |
| <pre>L=[1, 3, 2, 5, 20, 21] lst=list(filter(lambda x: x % 2 == 0, L)) print('filter:',lst)</pre> | filter: [2, 20] |

4.7.2.2. Sử dụng hàm map

- **Ví dụ 4.105:** cho 1 list chứa các số nguyên từ 2 đến 5. Tăng giá trị các phần tử thêm n ($n=3$) đơn vị.

| Sử dụng hàm thông thường | Sử dụng Anonymous function | Kết quả |
|--|--|-----------------------------------|
| <pre>def Cong(lst,n): for i in range(len(lst)): lst[i]+=n return lst lst1 = [2,3,4,5] print("Trước tăng: ",lst1)</pre> | <pre>lst1 = [2,3,4,5] print("Trước tăng: ",lst1)</pre> | Trước
tăng:
[2, 3,
4, 5] |
| <pre>n=3 lst2 = Cong(lst1,n) print("Sau tăng:",lst2)</pre> | <pre>n=3 lst2 = list(map(lambda x: x+n ,lst1)) print("Sau tăng:",lst2)</pre> | Sau
tăng:
[5, 6,
7, 8] |

4.7.2.3. Sử dụng hàm filter

- **Ví dụ 4.106:** tạo mới list tên *resultst* từ list *lst* cũ bằng cách lọc ra những số chia chẵn cho 13.

| Mã lệnh | Kết quả |
|--|---------------|
| <pre>lst = [12, 65, 54, 39, 102, 339, 221, 50, 70,] resultList = list(filter(lambda x: (x % 13 == 0), lst)) print(resultList)</pre> | [65, 39, 221] |

- **Ví dụ 4.107:** cho 1 list chứa các số nguyên từ 1 đến 19. Yêu cầu lọc ra các số đang có trong list sao cho các số này là số chẵn và có giá trị nằm trong khoảng từ 5 đến 15.

| Mã lệnh | Kết quả |
|--|--------------------|
| <pre>lst = [] for i in range(1,20): lst.append(i) newList = list(filter(lambda x: (x % 2 == 0 and 5<=x<=15), lst)) print (newList)</pre> | [6, 8, 10, 12, 14] |

- **Ví dụ 4.108:** cho 2 list *lst1* và *lst2* chứa các số nguyên. Cho biết các số có trong cả 2 list.

| Sử dụng hàm thông thường | Sử dụng Anonymous function | Kết quả |
|--|--|--|
| <pre>def intersect(L1,L2): L3=[] for i in L1: if i in L2: L3.append(i) return L3</pre> | <pre>def intersect(L1,L2): return list(filter(lambda x: x in lst2 , lst1))</pre> | num in
lst1,
lst2:
[2, 3,
5] |

| | | |
|--|--|--|
| <pre>lst1 = [1,2,3,4,5,6,7,8] lst2 = [2,3,5,9,12,14] lst3 = intersect(lst1 , lst2) print("num in lst1, lst2:",lst3)</pre> | <pre>lst1 = [1,2,3,4,5,6,7,8] lst2 = [2,3,5,9,12,14] print("num in lst1 and lst2:",intersect(lst1, lst2))</pre> | |
|--|--|--|

- **Ví dụ 4.109:** tạo mới list tên *resultst* từ list *lst* cũ bằng cách lọc ra những từ đối xứng (*palindromes*).

| Mã lệnh | Kết quả |
|---|-------------------------------------|
| <pre>lst = ["abba", "MADAM", "maDaM", "NaN", "Sai gon"] resultList = list(filter(lambda x: (x == "".join(reversed(x))), lst)) print(resultList)</pre> | <pre>['abba', 'MADAM', 'NaN']</pre> |

- **Ví dụ 4.110:** tạo mới list tên *resultst* từ list *lst* cũ bằng cách lọc ra những từ là đảo chữ của từ làm mẫu *strPattern*.

| Mã lệnh | Kết quả |
|--|-------------------------------|
| <pre>from collections import Counter lst = ["geeks", "geeg", "keegs", "geeps", "gee ks"] strPattern = "eegsk" result =list(filter(lambda x: (Counter(strPattern)==Counter(x)), lst)) print(result)</pre> | <pre>['geeks', 'keegs']</pre> |

4.7.2.4. Có thể sử dụng 1 trong 2 hàm *map* và *filter* để giải quyết cùng 1 vấn đề

- **Ví dụ 4.111:** cho 1 list chứa số nguyên. Yêu cầu đếm số lượng số chẵn, số lẻ trong list

| Mã lệnh | Kết quả |
|---|--|
| <pre>lst=[4,9,7,5,6] print('Dùng hàm map:') slChan=list(map(lambda x: x%2==0, lst)).count(True) slLe=list(map(lambda x: x%2==0, lst)).count(False) print('Trong list có %d số chẵn và %d số lẻ'%(slChan, slLe))</pre> | <pre>Dùng hàm map: Trong list có 2 số chẵn và 3 số lẻ</pre> |
| <pre>print('Dùng hàm filter:') slChan=len(list(filter(lambda x: x%2 == 0, lst))) slLe=len(list(filter(lambda x: x%2 != 0, lst))) print('Trong list có %d số chẵn và %d số lẻ' %(slChan, slLe))</pre> | <pre>Dùng hàm filter: Trong list có 2 số chẵn và 3 số lẻ</pre> |

4.7.3. *lambda* function có thể gọi 1 *lambda* function khác

- **Ví dụ 4.112:** Cho 1 list chứa các chuỗi như sau: *lst=['26587', '4.2365', '00', '-12547', 'A001', '-16.4', '-24587.11']*. Xác định xem phần tử nào trong list là số hợp lệ, phần tử nào là không hợp lệ.

| Mã lệnh | Kết quả |
|--|---|
| <pre>lst=['26587','4.2365','00','-12547','A001', '-16.4','-24587.11'] is_num1 = lambda strObj: strObj.replace('.', '', 1).isdigit() print("\nKiểm tra số lần 1:") for item in lst: if is_num1(item): print('%s is a digit' %item)</pre> | <pre>Kiểm tra số lần 1: 26587 is a digit 4.2365 is a digit 00 is a digit 001 is a digit -12547 is NOT a digit A001 is NOT a digit -16.4 is NOT a digit</pre> |

| | |
|--|--|
| <pre> else: print('%s is NOT a digit' %item) print("\nKiểm tra số lần 2:") is_num2 = lambda r: is_num1(r[1:]) if r[0]=='-' else is_num1(r) for item in lst: if is_num2(item): print('%s is a digit'%item) else: print('%s is NOT a digit'%item) </pre> | <pre> -24587.11 is NOT a digit Kiểm tra số lần 2: 26587 is a digit 4.2365 is a digit 00 is a digit 001 is a digit -12547 is a digit A001 is NOT a digit -16.4 is a digit -24587.11 is a digit </pre> |
|--|--|

4.8. Sử dụng kỹ thuật Comprehension cho sequence data type

- *Comprehension* là một biểu thức đi kèm với lệnh *for* được đặt trong cặp dấu ngoặc vuông ([]) – khi cần tạo *list* hoặc cặp dấu ngoặc nhọn ({}) – khi cần tạo *set* hoặc *dictionary*.
- Nhược điểm của *comprehensions*: tất cả các phần tử sẽ được sinh ra và lưu vào bộ nhớ. Vì vậy, chỉ nên sử dụng *comprehensions* đối với những đối tượng có số lượng phần tử không quá lớn.
- **Cú pháp** sử dụng *Comprehension*:

- Đối với *list*:

```
[f(x) for x in iterable [if condition] ]
```

Hoặc

```
[f(x) for x in iterable1 [f(y) for y in iterable2] [if condition]]
```

- Đối với *set*:

```
{f(x) for x in iterable [if condition] }
```

Hoặc

```
{f(x) for x in iterable1 [f(y) for y in iterable2] [if condition]}
```

- Đối với *dictionary*:

```
{x : f(x) for x in iterable [if condition] }
```

Hoặc

```
{x:f(x) for x in iterable1 [f(y) for y in iterable2]
[if condition]}
```

Trong đó:

- $f(x)$ là hàm hoặc biểu thức bất kỳ,
- `[if condition]` là tùy chọn.
- `[f(y) for y in iterable2]` là biểu thức tùy chọn.
- **Lưu ý:** cho phép nhiều vòng *for* lồng nhau trong cùng 1 lần sử dụng *comprehension* và chỉ có các lệnh *for* từ thứ 2 trở đi mới được dùng *if*.
- **Một số ví dụ minh họa cách sử dụng kỹ thuật *comprehension***

- Ví dụ 4.113: Tạo một *list* gồm các số nguyên từ 1 đến 10

| Cách thông thường | Cách sử dụng kỹ thuật <i>Comprehension</i> | Kết quả |
|---|--|--|
| <pre> lst = [] for i in range (1,11): lst.append(i) print(lst) </pre> | <pre> lst = [i for i in range(1,11)] print(lst) </pre> | <pre> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] </pre> |

- Ví dụ 4.114: Tạo một list gồm các số nguyên là 3^k ($0 \leq k \leq 9$)

| Cách thông thường | Cách sử dụng kỹ thuật Comprehension | Kết quả |
|--|---|--|
| <pre>lst = [] for x in range (10): lst.append(3**x) print(lst)</pre> | <pre>lst = [3 ** x for x in range(10)] print(lst)</pre> | [1, 3, 9, 27, 81, 243, 729, 2187, 6561, 19683] |

- Ví dụ 4.115: Giả sử năm hiện tại là 2019. Cho listYears chứa năm sinh của 1 số người. Tạo ra một list mới tên là listAges chứa tuổi của những năm sinh có trong listYears.

| Cách thông thường | Cách sử dụng kỹ thuật Comprehension | Kết quả |
|---|---|--------------------------|
| <pre>listYears = [1990, 1991, 1990, 1990, 1992, 1991] listAges = [] for year in listYears: listAges.append(2019 - year) print(listAges)</pre> | <pre>listYears = [1990, 1991, 1990, 1990, 1992, 1991] listAges = [2019 - year for year in listYears] print(listAges)</pre> | [29, 28, 29, 29, 27, 28] |

- Ví dụ 4.116: Tạo một list gồm các số lẻ trong khoảng từ 0 đến 18

| Cách thông thường | Cách sử dụng kỹ thuật Comprehension | Kết quả |
|---|---|---------------------------------|
| <pre>lst = [] for x in range (19): if x % 2 == 1 lst.append(x) print(lst)</pre> | <pre>lst=[x for x in range (19) if x % 2 == 1] print(lst)</pre> | [1, 3, 5, 7, 9, 11, 13, 15, 17] |

- Ví dụ 4.117: Tạo một list bằng cách nối chuỗi từ 2 list

| Cách thông thường | Cách sử dụng kỹ thuật Comprehension | Kết quả |
|--|--|--|
| <pre>lst= [] lstX=['Ngôn ngữ', 'Lập trình'] lstY=['Python', 'C++'] for x in lstX: for y in lstY: kq=x+y lst.append(kq) print (lst)</pre> | <pre>lstX=['Ngôn ngữ', 'Lập trình'] lstY=['Python', 'C++'] lst= [x+y for x in lstX for y in lstY] print (lst)</pre> | ['Ngôn ngữ Python', 'Ngôn ngữ C++', 'Lập trình Python', 'Lập trình C++'] |

- Ví dụ 4.118: sử dụng 3 for lồng nhau với 2 if. Cho 3 list, thực hiện tính toán các giá trị trong 3 list này với những điều kiện cho trước

| Cách sử dụng kỹ thuật Comprehension | Kết quả |
|--|--|
| <pre>lst1 = [i for i in range(1,3)] print('lst1=', lst1)</pre> | lst1= [1, 2] |
| <pre>lst2=[i for i in range(11,21)] print('lst2=', lst2)</pre> | lst2= [11, 12, 13, 14, 15, 16, 17, 18, 19, 20] |
| <pre>lst3=[i for i in range(21,31)] print('lst3=', lst3)</pre> | lst3= [21, 22, 23, 24, 25, 26, 27, 28, 29, 30] |
| <pre>lst=[i*(k-j) for i in lst1 for j in lst2 if j%4==0 for k in lst3 if k% 5==0] print(lst)</pre> | [13, 18, 9, 14, 5, 10, 26, 36, 18, 28, 10, 20] |

- Ví dụ 4.119: Tạo một **set** gồm các số nguyên trong đoạn [1..20] và chia hết cho 3

| Cách thông thường | Cách sử dụng kỹ thuật Comprehension | Kết quả |
|--|---|----------------------------------|
| <pre>s = set() for i in range(1, 21): if i % 3 == 0: s.add(i) print(s)</pre> | <pre>s = {i for i in range(1, 21) if i % 3 == 0} print(s)</pre> | <pre>{3, 6, 9, 12, 15, 18}</pre> |

- Ví dụ 4.120: Tạo một **dictionary** gồm các số nguyên trong đoạn [1..20] và chia hết cho 3

| Cách thông thường | Cách sử dụng kỹ thuật Comprehension | Kết quả |
|--|---|--|
| <pre>d = {} for k in range(1, 21): if k % 3 == 0: d[k] = k**2 print(d)</pre> | <pre>d = {k: k**2 for k in range(1, 21) if k % 3 == 0} print(d)</pre> | <pre>{3: 9, 6: 36, 9: 81, 12: 144, 15: 225, 18: 324}</pre> |

XỬ LÝ NGOẠI LỆ (Exception Handling)

5.1. Lỗi cú pháp (Syntax Error)

- Là các lỗi lập trình khi chương trình bị viết sai cú pháp quy định. Lỗi này còn biết đến như lỗi phân tích (*parsing error*).
- Ví dụ 5.1:

| Mã lệnh | Kết quả |
|--|--|
| <pre>i=1 while True print(i) i=i+1 if i>=5: break</pre> | <pre>File "D:/MyProject/venv/exercise.py", line 2 while True ^ SyntaxError: invalid syntax</pre> |

- Bộ phân tích lặp lại dòng gây lỗi và hiển thị một mũi tên nhỏ (^) trở vào điểm đầu tiên lỗi được phát hiện (lỗi nằm ở *phía sau* dấu hiệu mũi tên).
- Trong ví dụ trên, lỗi được phát hiện tại dòng lệnh *while*, ngay sau từ khóa *True*, vì thiếu một dấu hai chấm (":"). *Python* hiển thị *Tên tập tin* và *số dòng* để hỗ trợ trong việc tìm lỗi.

5.2. Lỗi ngoại lệ (Exception Error)

- Một câu lệnh hoặc biểu thức không có lỗi cú pháp vẫn có thể tạo lỗi khi thực thi.
- Lỗi (error) được phát hiện trong quá trình thực thi chương trình được gọi là *Exception* (ngoại lệ).
- Trong *Python*, mỗi *exception* là một đối tượng của *Python*, đại diện cho một sự kiện lỗi xuất hiện khi thực thi chương trình, lỗi đó đã phá vỡ cấu trúc bình thường của chương trình.
- Khi có phát sinh 1 *exception* và *exception* này không (hoặc chưa) được xử lý, chương trình sẽ bị ngắt tại nơi gây ra *exception*. Ngược lại nếu đã được người lập trình xử lý thì khi *exception* phát sinh, chương trình sẽ xử lý theo hướng của người lập trình đã cài đặt sẵn.
- Ngoài tính năng *debug*, *Python* cung cấp hai tính năng rất quan trọng để xử lý lỗi trong chương trình là:
 - *Assertions*.
 - *Exception Handling*.

5.3. Assertions

- Cú pháp: **`assert(Expression), [, Arguments]` thông_báo_lỗi**
- Giải thích:
 - Khi thấy một *assertion statement*, *Python* đánh giá các *expression* kèm theo. Nếu *expression* có kết quả là:
 - *True*: lệnh đi ngay sau lệnh *assert* sẽ tiếp tục được thực hiện.
 - *False*: *Python* sẽ phát ra một *AssertionError Exception* bằng cách dùng *ArgumentExpression* làm đối số cho *AssertsError*.
 - Các ngoại lệ của *AssertsError* có thể được bắt và xử lý như bất kỳ ngoại lệ nào khác bằng cách sử dụng câu lệnh *try-except*, nhưng nếu không được xử lý, chúng sẽ chấm dứt chương trình và tạo ra một *traceback* (*module traceback* là cách khác để xử lý *exception* trong *Python*).
 - Về cơ bản, *traceback* được sử dụng để xuất dấu vết của một chương trình sau khi một *exception* xảy ra. *traceback* bao gồm thông báo lỗi, số dòng gây ra lỗi và *call stack* của *function* gây ra lỗi.

- Ví dụ 5.2: tìm ước số chung lớn nhất của 2 số. Hàm USCLN sau đây chỉ tính được USCLN của 2 số >0

| Mã lệnh | Kết quả |
|--|--|
| <pre>def USCLN(a,b): assert (a>0 and b>0), "a va b phai>0" while a!=b: if a>b: a-=b else: b-=a return a x=6 y=9 print (USCLN(x,y)) z=-2 print (USCLN(x,z))</pre> | <p>3</p> <p>AssertionError: a va b phai la so >=0</p> |

5.4. Standard Exceptions

5.4.1. Các exception có sẵn trong Python

| TT | Exception Name | Mô tả |
|----|---------------------------|--|
| 1 | Exception | Đây là lớp cơ sở (base class) cho tất cả các exception, exception này sẽ xuất hiện khi có bất cứ một lỗi nào xảy ra. |
| 2 | StopIteration | Xuất hiện khi phương thức <i>next()</i> của <i>iterator</i> không trả về một đối tượng nào. |
| 3 | SystemExit | Xuất hiện khi dùng phương thức <i>sys.exit()</i> |
| 4 | StandardError | Lớp cơ sở cho tất cả các <i>exception</i> , ngoại trừ <i>StopIteration</i> và <i>SystemExit</i> . |
| 5 | ArithmeticError | Lớp cơ sở cho tất cả các lỗi xảy ra khi tính toán các số. |
| 6 | OverflowError | Xuất hiện khi thực hiện tính toán và giá trị tính toán vượt quá ngưỡng giới hạn cho phép của kiểu dữ liệu. |
| 7 | FloatingPointError | Xuất hiện khi tính toán các số kiểu float thất bại. |
| 8 | ZeroDivisionError | Xuất hiện khi thực hiện phép chia (<i>division</i>) hoặc chia lấy dư (<i>modulo</i>) một số cho 0 (<i>zero</i>). |
| 9 | AssertionError | Xuất hiện trong trường hợp lệnh <i>assert</i> thất bại. |
| 10 | AttributeError | Xuất hiện khi không tồn tại thuộc tính này, hoặc thiếu tham số truyền vào cho thuộc tính. |
| 11 | EOFError | Xuất hiện khi không có dữ liệu từ hàm <i>input()</i> hoặc <i>raw_input()</i> hay lỗi do thao tác trên file khi con trỏ đang ở cuối file. |
| 12 | ImportError | Xuất hiện khi lệnh <i>import</i> thất bại (lỗi). |
| 13 | KeyboardInterrupt | Xuất hiện khi người dùng gián đoạn việc thực hiện chương trình, thường bằng cách nhấn Ctrl + C |
| 14 | LookupError | Lớp cơ sở cho tất cả các lỗi về <i>lookup</i> |

| | | |
|----|----------------------------|---|
| 15 | IndexError | Xuất hiện khi index không tồn tại trong list, string, ... |
| 16 | KeyError | Xuất hiện khi <i>key</i> không tồn tại trong <i>dictionary</i> . |
| 17 | NameError | Xuất hiện khi một biến không tồn tại trong phạm vi chương trình gọi biến đó. |
| 18 | UnboundLocalError | Raised when trying to access a local variable in a function or method but no value has been assigned to it. |
| 19 | EnvironmentError | Là lớp cơ sở cho tất cả các exception về lỗi khi có bất kỳ một lỗi nào ngoài phạm vi của Python. |
| 20 | IOError | Xuất hiện khi xử dụng <i>input/ output</i> thất bại, hoặc mở <i>file</i> không thành công (không tồn tại). |
| 21 | OSError | Xuất hiện khi có lỗi từ hệ điều hành. |
| 22 | SyntaxError | Xuất hiện khi chương trình có lỗi cú pháp. |
| 23 | IndentationError | Xuất hiện khi có lệnh thụt đầu dòng không đúng. |
| 24 | SystemError | Xuất hiện khi trình thông dịch phát hiện có vấn đề, nhưng lúc này trình thông dịch <i>Python</i> không tự thoát (kết thúc) được |
| 25 | SystemExit | Xuất hiện khi trong code không sử dụng hàm <code>sys.exit()</code> nhưng trình thông dịch <i>Python</i> vẫn được thoát bằng hàm <code>sys.exit()</code> . |
| 26 | TypeError | Xuất hiện khi thực thi toán tử hoặc hàm mà kiểu dữ liệu bị sai so với kiểu dữ liệu đã định nghĩa ban đầu. |
| 27 | ValueError | Xuất hiện khi chúng ta build 1 function mà kiểu dữ liệu đúng nhưng khi chúng ta thiết lập ở tham số là khác so với khi truyền vào. |
| 28 | RuntimeError | Xuất hiện khi lỗi được sinh ra không thuộc một danh mục nào. |
| 29 | NotImplementedError | Xuất hiện khi một phương thức trừu tượng cần được thực hiện trong lớp kế thừa chứ không phải là lớp thực thi |
| 30 | UnboundLocalError | Xuất hiện khi chúng ta cố tình truy cập vào một biến trong hàm hoặc phương thức, nhưng không thiết lập giá trị cho biến. |

5.4.2. Sử dụng try ... except trong việc xử lý ngoại lệ

5.4.2.1. Cú pháp

```

try
    # khối lệnh có khả năng xảy ra lỗi
except loại_lỗi_1 as tên_biến_báo_lỗi_1
    # in thông báo lỗi
except loại_lỗi_1 as tên_biến_báo_lỗi_1
    # in thông báo lỗi
...
else
    # khối lệnh khi không có exception nào xảy ra

```

Lưu ý: có thể sử dụng lệnh `raise` để định nghĩa bổ sung cho *Exception*

5.4.2.2. Một số ví dụ

- Ví dụ 5.3: sử dụng *Exception* với *try ... except* để thực hiện yêu cầu chỉ cho nhập số nguyên dương. Kết quả được in ra màn hình tùy thuộc dữ liệu nhập vào của người dùng.

```
def NhapSo():
    while True:
        try:
            n=eval(input('Nhap so nguyen >0: '))
        except Exception:
            print('Gia tri nhap khong phai kieu so')
        else:
            if type(n) is not int:
                print('Chi nhan so nguyen >0')
            elif n<=0:
                print('La so nguyen nhung phai >0')
            else:
                return n
    print ('\nn=',NhapSo())
```

- Ví dụ 5.4: sử dụng *ZeroDivisionError* với *try ... except* để kiểm tra mẫu số trong phép chia phải khác zero (0). Nếu mẫu số (y) có giá trị khác 0, chương trình sẽ in ra kết quả của phép chia x cho y, ngược lại sẽ xuất hiện lỗi *division by zero*

```
x, y = 5, 0
try:
    print(x, '/', y, '=', x/y)
except ZeroDivisionError as err:
    print('Error: ',err)
```

- Ví dụ 5.5: tính tổng, hiệu, tích, thương của 2 số x và y. Sử dụng *ZeroDivisionError* với *try ... except ... finally* để kiểm tra mẫu số trong phép chia x cho y phải khác zero (0)

| Mã lệnh | Kết quả |
|--|---|
| <pre>x, y = 5, 0 try: print(x, '/', y, '=', x/y) except ZeroDivisionError as err: print('Error: ',err) finally: print(x, '+', y, '=', x + y) print(x, '-', y, '=', x - y) print(x, '*', y, '=', x * y)</pre> | <div>Error: division by zero</div> <div>5 + 0 = 5</div> <div>5 - 0 = 5</div> <div>5 * 0 = 0</div> |

- Ví dụ 5.6: sử dụng kết hợp *NameError* và *ZeroDivisionError*:

| Mã lệnh | Kết quả |
|---|---|
| <pre>try: x = eval(input('input x: ')) y = eval(input('input y: ')) z = x/y except (NameError, ZeroDivisionError) as err: print('Error: ',err) else: print(x, '/', y, '=', z)</pre> | <div>#Khi KHÔNG CÓ lỗi</div> <div>input x: >? 2</div> <div>input y: >? 3</div> <div>2 / 3 = 0.6666666666666666</div> <div>#Khi CÓ lỗi</div> <div>input x: >? 4</div> <div>input y: >? 0</div> <div>Error: division by zero</div> <div>Hoặc:</div> <div>input x: >? 2</div> |

```
input y: >? a
Error: name 'a' is not
defined
```

- Ví dụ 5.7: sử dụng kết hợp `raise` với `ZeroDivisionError` và `NameError`:

| Mã lệnh | Kết quả |
|---|--|
| <pre>try: x = eval(input('input x: ')) y = eval(input('input y: ')) if (y==0): raise ZeroDivisionError ('y phải khác 0 (zero)') z = x/y except (NameError) as err: print('Error: ',err) except (ZeroDivisionError) as err: print('Error: ', err) else: print(x, '/', y, '=', z)</pre> | <pre>#Khi KHÔNG CÓ lỗi input x: >? 2 input y: >? 3 2 / 3 = 0.6666666666666666 #Khi CÓ lỗi input x: >? 4 input y: >? 0 Error: y phải khác 0 (zero) Hoặc: input x: >? 2 input y: >? a Error: name 'a' is not defined</pre> |

5.5. Exception do người dùng định nghĩa (User Defined Exception)

- Một *exception* trong *Python* do người dùng định nghĩa luôn bắt buộc *exception* này phải kế thừa các lớp thuộc *Standard built-in Exception* trong *Python*.
- Để gọi *exception* do người dùng định nghĩa, cần sử dụng *keyword raise* theo cú pháp sau:
`raise exception_Name`

- Ví dụ 5.8
 - B1: định nghĩa ra *exception* của người dùng (*my_Exception*)
 - B2: viết chương trình (hoặc hàm), trong đó gọi *my_Exception* bằng lệnh *raise*
 - B3: từ chương trình chính hoặc hàm khác, gọi hàm *X* và gọi tham số sẽ gây lỗi để biết kết quả

| Mã lệnh | Kết quả |
|--|-------------------------------|
| <pre>#B1: class my_Exception(Exception): def __init__(self, value): print("Loi: " + value) #B2: def division(a, b): if (b == 0): raise my_Exception('b phai khac 0') return a / b #B3: division (6, 0)</pre> | <pre>Loi: b phai khac 0</pre> |

THAO TÁC VỚI TẬP TIN & THƯ MỤC

6.1. Thao tác với các loại tập tin

File là tập hợp của các thông tin được đặt tên và được lưu trữ trên bộ nhớ máy tính như đĩa cứng, đĩa mềm, CD, DVD,... Hiểu theo một cách khác thì File chính là một dãy bit có tên và được lưu trữ trên các thiết bị bộ nhớ của máy tính.

Có 3 loại file thông dụng: văn bản, hình ảnh và âm thanh. Trong phần này chủ yếu chỉ hướng đến file dạng văn bản.

Python cung cấp các phương thức cơ bản và cần thiết để thao tác với tập tin theo mặc định. Ta có thể thực hiện các thao tác với tập tin bằng cách sử dụng *file object*.

6.1.1. Mở file

- Trước khi muốn đọc hoặc ghi file, cần có thao tác mở file theo cú pháp:

```
fileObject = open(fileName [, mode = accessMode] [,newline='']  
[ , buffering] [, encoding = 'utf-8'])
```

trong đó:

- **fileName**: tên file sẽ truy cập (kèm đường dẫn nếu file không có trong thư mục hiện hành). Nếu file không tồn tại hoặc đường dẫn đến file sai sẽ phát sinh lỗi *FileNotFoundError*.
- **accessMode**: chế độ mở tập tin: read, write, append, ... Có một số chế độ mở file là:
 - **r** : mở để đọc nội dung (mặc định)
 - **w** : mở để ghi nội dung.
 - **a** : mở để ghi thêm nội dung vào cuối file.
 - **r+** : mở để đọc và ghi. Con trỏ nằm ở đầu file.
 - **w+** : mở để đọc và ghi. Ghi đè nếu file đã tồn tại, nếu file chưa tồn tại thì tạo file mới để ghi.
 - **a+** : mở để đọc và thêm vào cuối file. Con trỏ nằm ở cuối file. Nếu file chưa tồn tại thì tạo file mới để ghi.

Mặc định là mở file *text*, để mở file dạng nhị phân (*binary*) cần thêm **b**, như: **rb**, **wb**, **ab**, **rb+**, **wb+**, **ab+**.

- **buffering**: gồm các giá trị:
 - **1** : có sử dụng buffer,
 - **0** : không sử dụng buffer,
 - **>1** : buffer size,
 - **<0** : default size. Đây là giá trị mặc định.
- **newline=""**: (tùy chọn) cho biết chế độ khi ghi dòng mới. Các giá trị hợp lệ là: **None**, cặp dấu nháy đơn (**"**), **'\n'**, **'r'**, **'\r\n'**. Khi ghi file, nếu các dòng bị cách nhau bởi 1 dòng trống, khi đó sử dụng **newline=""**.
- **encoding='utf-8'**: mặc định, python sử dụng mã ASCII. Vì vậy, khi cần sử dụng mã UNICODE, ta phải sử dụng thêm tùy chọn này.

- Ví dụ 6.1: `f = open('D:/test1.txt', mode = 'r')`
hoặc `f = open('test2.txt', mode = 'r', encoding = 'utf-8')`
- Sau khi gọi hàm **open()** thành công, kết quả sẽ trả về một *file object* hay còn gọi là **"handle"** có các thuộc tính:

- **closed** : True nếu file đã đóng.
- **mode** : chế độ khi mở.
- **file name** : tên của file.
- **softspace** : cờ đánh dấu softspace khi dùng với hàm *print*.

6.1.2. Đóng file

- Sau khi hoàn tất các thao tác đọc ghi trên file, cần gọi phương thức **close()** để đóng file. Việc đóng file để đảm bảo quy chế đóng mở và giải phóng bộ nhớ cho chương trình.
- Thực hiện: có thể sử dụng 1 trong 3 cách sau, trong đó, cách 3 là tốt và gọn nhất:

6.1.2.1. Cách 1

- **Cú pháp**

```
fileObject = open(fileName [,accessMode] [, buffering] [, encoding])
# thực hiện các thao tác với file
fObject.close()
```

- **Ví dụ 6.2**

| Mã lệnh | Kết quả |
|--|---|
| <pre>f = open('d:\\test.txt', 'r', encoding = 'utf-8') result = f.read() print(result) print('\nDa doc xong file') f.close()</pre> | <pre>Thuyền và biển Chỉ có thuyền mới hiểu Biển mênh mông nhường nào Chỉ có biển mới biết Thuyền đi đâu về đâu ... Da doc xong file</pre> |

- **Nhận xét:** Cách này chưa thực sự đảm bảo vì vẫn có trường hợp một số ngoại lệ xảy ra khi chúng ta thực hiện các thao tác với file khiến chương trình tự động thoát ra mà không đóng file.

6.1.2.2. Cách 2

- **Sử dụng khối try...finally:** do *finally* sẽ luôn luôn được thực thi bất chấp có hay không ngoại lệ.

- **Cú pháp**

try:

```
fileObject = open(fileName [,accessMode] [, buffering] [, encoding])
# thực hiện các thao tác với file
```

finally:

```
f.close()
```

- **Ví dụ 6.3**

| Mã lệnh | Kết quả |
|--|---|
| <pre>try: f = open('d:\\test.txt', 'r', encoding = 'utf-8') result = f.read() print(result) print('\nDa doc xong file') finally: f.close()</pre> | <pre>Thuyền và biển Chỉ có thuyền mới hiểu Biển mênh mông nhường nào Chỉ có biển mới biết Thuyền đi đâu về đâu ... Da doc xong file</pre> |

- **Nhận xét:** Bằng cách này, ta có thể yên tâm file được đóng đúng ngay cả khi phát sinh ngoại lệ khiến chương trình dừng đột ngột.

6.1.2.3. Cách 3

- Sử dụng câu lệnh `with`: Lệnh `with` đảm bảo rằng file luôn luôn được đóng mà không cần biết những logic xử lý bên trong.

- Cú pháp

```
with open("test.txt", encoding = "utf-8") as f:
```

thực hiện các thao tác với file

- Ví dụ 6.4

| Mã lệnh | Kết quả |
|---|---|
| <pre>with open('d:\\test.txt', 'r', encoding = 'utf-8') as f: result = f.read() print(result) print('\nDa doc xong file')</pre> | <pre>Thuyền và biển Chỉ có thuyền mới hiểu Biển mênh mông nhường nào Chỉ có biển mới biết Thuyền đi đâu về đâu ... Da doc xong file</pre> |

6.1.2.4. Cách 4

- Phối hợp cách 2 (`try`) và 3(`with`):

- Ví dụ 6.5

| Mã lệnh | Kết quả |
|---|---|
| <pre>try: with open('d:\\test.txt', 'r', encoding='utf-8') as f: result = f.read() print(result) print('\nDa doc xong file') except FileNotFoundError or IOError: print('Không tìm thấy file hoặc file bị lỗi')</pre> | <pre>Thuyền và biển Chỉ có thuyền mới hiểu Biển mênh mông nhường nào Chỉ có biển mới biết Thuyền đi đâu về đâu ... Da doc xong file</pre> |

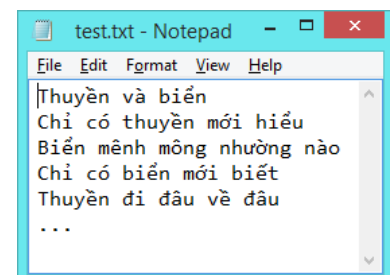
6.2. Thao tác với tập tin văn bản (Text File)

6.2.1. Đọc và ghi file

6.2.1.1. Đọc file

6.2.1.1.1. Phương thức `read`

- Cú pháp: `string_variable = fileObject.read([size])`
 - Trong đó: `size` là số lượng byte muốn đọc, nếu bỏ qua (không truyền) tham số này thì đọc từ đầu đến cuối file.
 - Lưu ý:
 - File đã được mở ở chế độ đọc.
 - Phương thức này đọc được tất cả các ký tự xuống dòng (`'\n'`) có trong file.
- Ví dụ 6.6: Giả sử đã có file `D:\test.txt` với nội dung như hình bên. Thực hiện đọc 1 ký tự đầu, sau đó đọc tiếp 5 ký tự kế tiếp



| Mã lệnh | Kết quả |
|---|-----------------------|
| <pre>try: with open('d:\\test.txt', 'r', encoding='utf-8') as f: ''' Khi file được lưu với mã UNICODE, để đọc ký tự đầu tiên của file phải tự cộng thêm 1 để bỏ qua ký hiệu U + FEFF ''' result = f.read(2) print(result) result = f.read(5) print(result) print('\nDa doc xong file') except FileNotFoundError or IOError: print('Không tìm thấy file hoặc file bị lỗi')</pre> | <p>T</p> <p>huyền</p> |

- Ví dụ 6.7: Cũng với file D:\test.txt. Thực hiện đọc 1 lần toàn bộ nội dung có trong file.

| Mã lệnh | Kết quả |
|---|--|
| <pre>try: with open('d:\\test.txt', 'r', encoding='utf-8') as f: result = f.read() print(result) print('\nDa doc xong file') except FileNotFoundError or IOError: print('Không tìm thấy file hoặc file bị lỗi')</pre> | <p>Thuyền và biển</p> <p>Chỉ có thuyền mới hiểu</p> <p>Biển mênh mông nhường</p> <p>nào</p> <p>Chỉ có biển mới biết</p> <p>Thuyền đi đâu về đâu</p> <p>...</p> <p>Da doc xong file</p> |

- Ví dụ 6.8: viết hàm nhận 1 tham số là tên file cần đọc (filename). Biết rằng các chuỗi số trong file được cách nhau bởi 1 khoảng trắng. Hàm thực hiện đọc file này và tính tổng các số đã đọc được:

| Mã lệnh | Kết quả |
|--|--|
| <pre>def DocFile (filename): try: with open(filename, 'r', encoding='utf-8') as f: ''' xóa các ký tự khoảng ở đầu và cuối kết quả đọc được nếu có''' result = f.read().strip() print('Ket qua doc=', result) lst = result.split(' ') print('Chuyển sang List:', lst) tong = 0 for i in lst: num=0 if i.isdigit(): num=int(i) elif i[1:].isdigit(): #số âm num=int(i[1:])*(-1) print(num, end=' ') tong += num print('\nTong= %d' %tong) except FileNotFoundError or IOError: print('Không tìm thấy file hoặc file bị lỗi')</pre> | <p>Ket qua doc= -5 12 7 -9 4 8 2</p> <p>Chuyển sang List: ['-5', '12', '7', '-9', '4', '8', '2']</p> <p>-5 12 7 -9 4 8 2</p> <p>Tong= 19</p> |
| <pre>DocFile("d:\\test1.txt")</pre> | |

6.2.1.1.2. Đọc file với phương thức readline

- Cú pháp **`string_variable = fileObject.readline()`**
- Lưu ý:
 - File đã được mở ở chế độ đọc.
 - Phương thức này cho phép đọc mỗi lần một dòng có trong file, trong đó ký tự xuống dòng (**`\n`**) sẽ được đọc và ở cuối của chuỗi kết quả.
- Ví dụ 6.9: cũng với file D:\test.txt. Sử dụng phương thức readline để đọc mỗi lần 1 dòng có trong file.

| Mã lệnh | Kết quả |
|--|---|
| <pre>try: with open("D:\\test.txt", 'r', encoding='utf-8') as f: count=1 while True: result = f.readline() if result!='': print('Dong %d: %s'%(count,result),end='') count+=1 else: break except FileNotFoundError or IOError: print('Không tìm thấy file hoặc file bị lỗi')</pre> | <p>Dong 1: Thuyền và biển</p> <p>Dong 2: Chỉ có thuyền mới hiểu</p> <p>Dong 3: Biển mênh mông nhường nào</p> <p>Dong 4: Chỉ có biển mới biết</p> <p>Dong 5: Thuyền đi đâu về đâu</p> <p>Dong 6: ...</p> |

- Ví dụ 6.10: cũng với file D:\test.txt. Sử dụng phương thức readline để đọc 3 lần, lần đầu đọc 1 dòng, lần thứ hai đọc 1 ký tự trên dòng hiện hành, lần thứ 3 đọc 5 ký tự trên dòng hiện hành.

| Mã lệnh | Kết quả |
|--|---|
| <pre>try: with open("D:\\test.txt", 'r', encoding='utf-8') as f: result = f.readline() print(result,end='') result = f.readline(1) print(result) result = f.readline(5) print(result) except FileNotFoundError or IOError: print('Không tìm thấy file hoặc file bị lỗi')</pre> | <p>Thuyền và biển</p> <p>c</p> <p>hi có</p> |

6.2.1.2. Ghi nội dung vào file

- Cú pháp: **`fileObject.write(content)`**
 - Trong đó: **`content`** là nội dung cần ghi
 - Lưu ý: file đã được mở ở chế độ ghi.
- Ví dụ 6.11


```
try:
    with open("D:\\test.txt", 'a+', encoding='utf-8') as f:
        f.write('\nXuân Quỳnh')
        print('Ghi thêm nội dung vào file thành công')
except FileNotFoundError or IOError:
    print('Không ghi được vào file')
```
- Ví dụ 6.12: viết hàm nhận 2 tham số là tên file cần tạo (filename) và số lượng số nguyên (n) được phát sinh ngẫu nhiên cần ghi vào file. Các số ghi vào file được cách nhau bởi 1 khoảng trắng:

```

from random import *
def GhiFile(filename, n):
    try:
        with open(filename, 'w', encoding='utf8') as f:
            for i in range (n):
                f.write('%s' %str(randint(-10,100)))
                print('Tạo và ghi file thành công')
    except FileNotFoundError or IOError:
        print('Không ghi được vào file')

GhiFile('D:\\test02.txt',50)

```

6.2.1.3. Hàm hỗ trợ cho đọc ghi file

- **tell()**: cho biết vị trí hiện tại trong file
- **seek(offset[, from])**: thay đổi vị trí hiện tại của tập tin.
trong đó:
 - **offset**: khoảng cách di chuyển (tính bằng byte)
 - **from**: vị trí làm mốc cho việc di chuyển, gồm các giá trị
 - 0: đầu file
 - 1: vị trí hiện thời
 - 2: cuối file
- Ví dụ 6.13

| Mã lệnh | Kết quả |
|--|---|
| <pre> try: with open('d:\\test.txt', 'r', encoding='utf8') as f: str = f.read(7); print("Content is : ", str) position = f.tell(); print("Current file position: ", position) str = f.read(7); print("Content is : ", str) position = f.seek(0, 0); print("After seek, position=", position) str = f.read(15); print("Read after seek:", str) except FileNotFoundError or IOError: print('Không ghi được vào file') </pre> | <p>Content is : Thuyền</p> <p>Current file position: 11</p> <p>Content is : và biể</p> <p>After seek, position= 0</p> <p>Read after seek: Thuyền và biể</p> |

6.2.2. Đổi tên file

- Sử dụng phương thức **os.rename(oldName, newName)** để đổi tên một file.
- Ví dụ 6.14:


```

import os
os.rename('oldName.txt', 'newName.txt')

```

6.2.3. Xóa file

- Sử dụng phương thức **os.remove(file)** để xóa một file khỏi hệ thống.
- Ví dụ 6.15:


```

import os
os.remove('test.txt')

```

6.3. Thao tác với tập tin CSV (CSV File)

6.3.1. File CSV

CSV được viết tắt từ *Comma Separated Values* ("giá trị được phân tách bằng dấu phẩy"). Đây là một định dạng văn bản dành cho việc trình bày dữ liệu dạng bảng. Trừ dòng đầu tiên trong file là tiêu đề các cột của bảng, tất cả các dòng còn lại mỗi dòng là một dòng dữ liệu của bảng. Các giá trị của các cột riêng lẻ được phân tách bằng ký hiệu dấu phân cách - dấu phẩy (,), dấu chấm phẩy (;) hoặc ký hiệu khác.

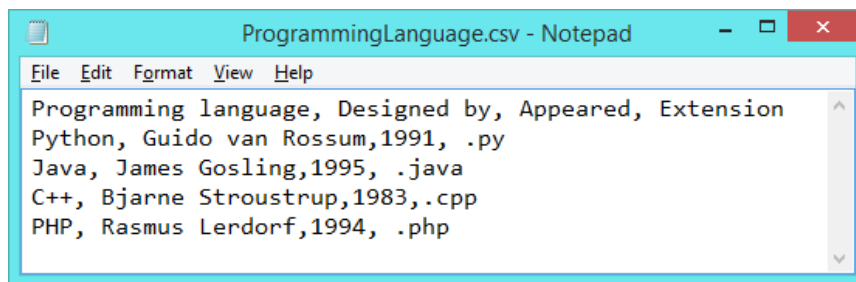
File CSV thường được sử dụng để trao đổi dữ liệu giữa các ứng dụng khác nhau bằng cách xuất dữ liệu phức tạp từ một ứng dụng sang file CSV, sau đó nhập dữ liệu trong file CSV đó vào một ứng dụng khác.

Nội dung file CSV thường được tạo lập/hiệu chỉnh bằng ứng dụng *MSEXcel* hoặc *Notepad*.

Ví dụ để có bảng dữ liệu sau

| <i>Programming language</i> | <i>Designed by</i> | <i>Appeared</i> | <i>Extension</i> |
|-----------------------------|--------------------|-----------------|------------------|
| Python | Guido van Rossum | 1991 | .py |
| Java; | James Gosling | 1995 | .java |
| C++ | Bjarne Stroustrup | 1983 | .cpp |
| PHP | Rasmus Lerdorf | 1994 | .php |

Sử dụng ứng dụng *Notepad* để tạo 1 file *ProgrammingLanguage.csv* với nội dung:



Hoặc có thể sử dụng ứng dụng *MSEXcel* để tạo 1 file *ProgrammingLanguage.csv* với nội dung:

| | A | B | C | D | E |
|---|----------------------|-------------------|----------|-----------|---|
| 1 | Programming language | Designed by | Appeared | Extension | |
| 2 | Python | Guido van Rossum | 1991 | .py | |
| 3 | Java | James Gosling | 1995 | .java | |
| 4 | C++ | Bjarne Stroustrup | 1983 | .cpp | |
| 5 | PHP | Rasmus Lerdorf | 1994 | .php | |
| 6 | | | | | |

6.3.2. Module CSV

6.3.2.1. Một số phương thức trong module CSV

| Hàm | Công dụng |
|---|--|
| <code>csv.field_size_limit([<i>new_limit</i>])</code> | Trả về kích thước max của trường hiện tại được cho phép bởi parser. Nếu có tham số <i>new_limit</i> , giá trị này sẽ trở thành giới hạn mới. |

| | |
|--|--|
| <code>csv.get_dialect (name)</code> | Trả về dialect liên kết đến <i>name</i> |
| <code>csv.list_dialects</code> | Trả về tên của tất cả các dialect đã đăng ký. |
| <code>csv.reader</code>
(<i>csvfile</i> , <i>dialect</i> ='excel', <i>**fmtparams</i>) | Đọc dữ liệu từ file csv. |
| <code>csv.register_dialect</code>
(<i>name</i> , [<i>dialect</i> ,] <i>**fmtparams</i>) | Liên kết dialect với một <i>name</i> . Với <i>name</i> phải là một chuỗi hoặc một đối tượng Unicode. |
| <code>csv.writer</code> | Ghi dữ liệu vào file csv. |
| <code>csv.unregister_dialect</code> | Xóa dialect được liên kết với tên từ sổ đăng ký dialect. Nếu tên không phải là tên dialect đã đăng ký, sẽ phát sinh lỗi. |

6.3.2.1. Một số hằng số trong module CSV

| Hằng số | Ý nghĩa |
|-----------------------------------|---|
| <code>csv.QUOTE_ALL</code> | Trích dẫn tất cả các field, không phân biệt kiểu |
| <code>csv.QUOTE_MINIMAL</code> | Chỉ trích dẫn những field có chứa các ký tự đặc biệt như dấu ngoặc kép, dấu phân cách, v.v. |
| <code>csv.QUOTE_NONNUMERIC</code> | Trích dẫn tất cả các field không phải là số. |
| <code>csv.QUOTE_NONE</code> | Không trích dẫn bất cứ điều gì ở đầu ra. |

6.3.3. Đọc file CSV

6.3.3.1. Đọc file CSV với phương thức `csv.reader`

- Cú pháp:

```
try:
    with open(fileName [, skipinitialspace=False|True] [,**fmtparams]) as csvfile:
        data = csv.reader(csvfile, [, delimiter=','])
        #các xử lý trên dữ liệu vừa đọc được trong data
except FileNotFoundError or IOError:
    #Xử lý lỗi
```

- Giải thích:

- Hàm `csv.reader(csvfile)` được sử dụng để đọc nội dung file CSV theo từng dòng của file mỗi dòng gồm nội dung trên tất cả các cột theo thứ tự từ trái qua phải.
- `csvfile` có thể là file CSV hoặc các đối tượng dạng danh sách (*string*, *list*, ...).
- *delimiter*: là 1 ký tự được sử dụng sử dụng làm dấu phân cách giữa các cột. Mặc định là dấu phẩy (,).
- *skipinitialspace=False|True*:
 - Khi *=False* (giá trị mặc định), sẽ giữ nguyên các khoảng trắng sau dấu phân cách trong file.
 - Khi *=True*, xóa tất cả các khoảng trắng (nếu có) sau dấu phân cách trong file và luôn thêm khoảng trắng vào ngay sau dấu phân cách trong kết quả.
- ***fmtparams*: hằng số cho biết loại dữ liệu cần trích dẫn. Mỗi dòng đọc từ `csvfile` được trả về dưới dạng danh sách các chuỗi. Không có chuyển đổi loại dữ liệu tự động được

thực hiện trừ khi tùy chọn định dạng `QUOTE_NONNUMERIC` được chỉ định (trong trường hợp đó, các trường không được trích dẫn được chuyển thành float).

- Kết quả trả về của hàm `csv.reader()` là một `csvreader` object chứa các dòng có trong `csvfile` đã cho, kiểu dữ liệu của các dòng là 1 `list`, nhờ vậy ta có thể truy cập các phần tử của `list` qua `index` (ví dụ `row[index]`).
- Thuộc tính `line_num` xác định số dòng đọc được trong `csvreader` object (bao gồm cả dòng đầu tiên chứa tiêu đề các field)
- Phương thức `csvReaderName.__next__()`:
 - Trả về 1 `list` chứa nội dung của dòng hiện tại trong tập kết quả của `csvreader` object.
 - Có thể sử dụng phương thức này bằng 1 trong 2 cách:
 - `csvReaderName.__next__()`
 - `listName = csvReaderName.__next__()`
 - Kết quả sẽ tương tự như phương thức `csvReaderName.__next__()` khi sử dụng hàm `next(csvReaderName)`.
- **Ví dụ 6.16:** viết hàm sử dụng `csv.reader` đọc file `ProgrammingLanguage.csv`, in tất cả kết quả đọc được theo từng dòng ra màn hình:

```
import csv
def DocNoiDungFile_CSV_1(filename):
    try:
        with open(filename, 'rt') as f:
            csvReaderObj = csv.reader(f, delimiter=',')
            for row in csvReaderObj:
                print(row)
            print("Tổng cộng gồm %d dòng" % csvReaderObj.line_num)
    except FileNotFoundError:
        print("Không tìm thấy tập tin %s" % filename)
#-----
DocNoiDungFile_CSV_1('D:\\ProgrammingLanguage.csv')
```

Kết quả khi thực hiện chương trình trên:

```
['Programming language', 'Designed by', 'Appeared', 'Extension']
['Python', 'Guido van Rossum', '1991', '.py']
['Java', 'James Gosling', '1995', '.java']
['C++', 'Bjarne Stroustrup', '1983', '.cpp']
['PHP', 'Rasmus Lerdorf', '1994', '.php']
Tổng cộng gồm 5 dòng
```

| | | | | |
|--------------|------------------------|--|--|--|
| csvReaderObj | 'Programming language' | | | |
| | 'Designed by' | | | |
| | 'Appeared' | | | |
| | 'Extension' | | | |
| | 'Python' | | | |
| | 'Guido van Rossum' | | | |
| | '1991' | | | |
| | '.py' | | | |
| | 'Java' | | | |
| | 'James Gosling' | | | |
| | '1995' | | | |
| | '.java' | | | |
| | 'C++' | | | |
| | 'Bjarne Stroustrup' | | | |
| | '1983' | | | |
| | '.cpp' | | | |
| | 'PHP' | | | |
| | 'Rasmus Lerdorf' | | | |
| | '1994' | | | |
| | '.php' | | | |

Mỗi dòng trong bảng này là 1 `list` (đối tượng `row`) trong mã lệnh của ví dụ 6.16

Hình 6.1. Minh họa ví dụ 6.16

- **Ví dụ 6.17:** tương tự như ví dụ 6.16, chỉ khác là dòng tiêu đề của bảng được in sau khi đã in các dòng chứa dữ liệu:

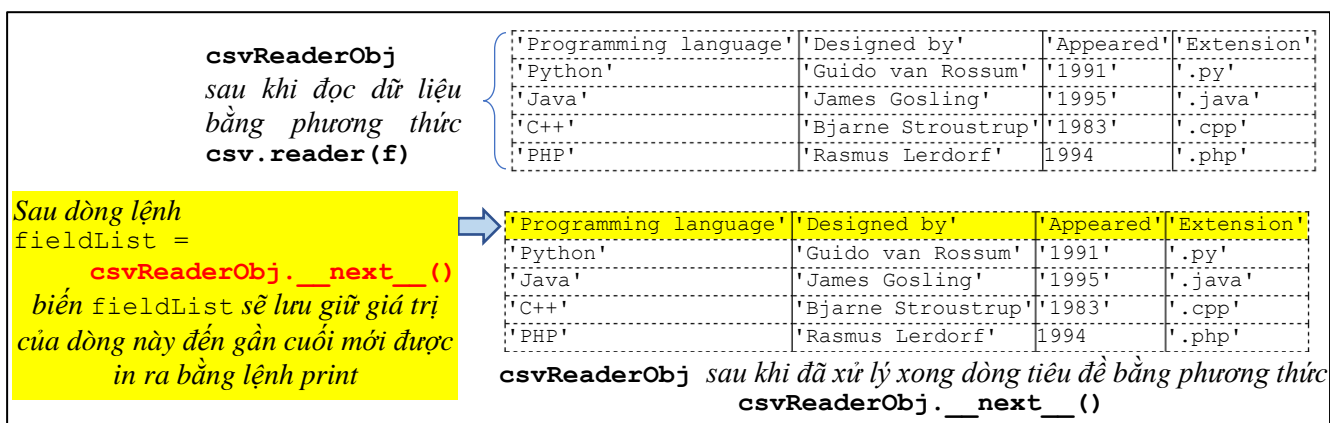
```
import csv
def DocNoiDungFile_CSV_2(filename):
    try:
        with open(filename, 'rt', encoding='utf8') as f:
            csvReaderObj = csv.reader(f)
            # Đưa dòng tiêu đề của bảng dữ liệu vào 1 list
            fieldList = csvReaderObj.__next__()
            for row in csvReaderObj:
                print(row)          # xuất ra dưới dạng list
                # print(', '.join(row)) # xuất ra dưới dạng chuỗi
            print("Tổng cộng gồm %d dòng dữ liệu" % (csvReaderObj.line_num-1))

            print("Tên các cột lần lượt là: ", end='')
            print(', '.join(field for field in fieldList))
    except FileNotFoundError:
        print("Không tìm thấy tập tin %s" % filename)

#-----
DocNoiDungFile_CSV_2('D:\\ProgrammingLanguage.csv')
```

Kết quả khi thực hiện chương trình trên:

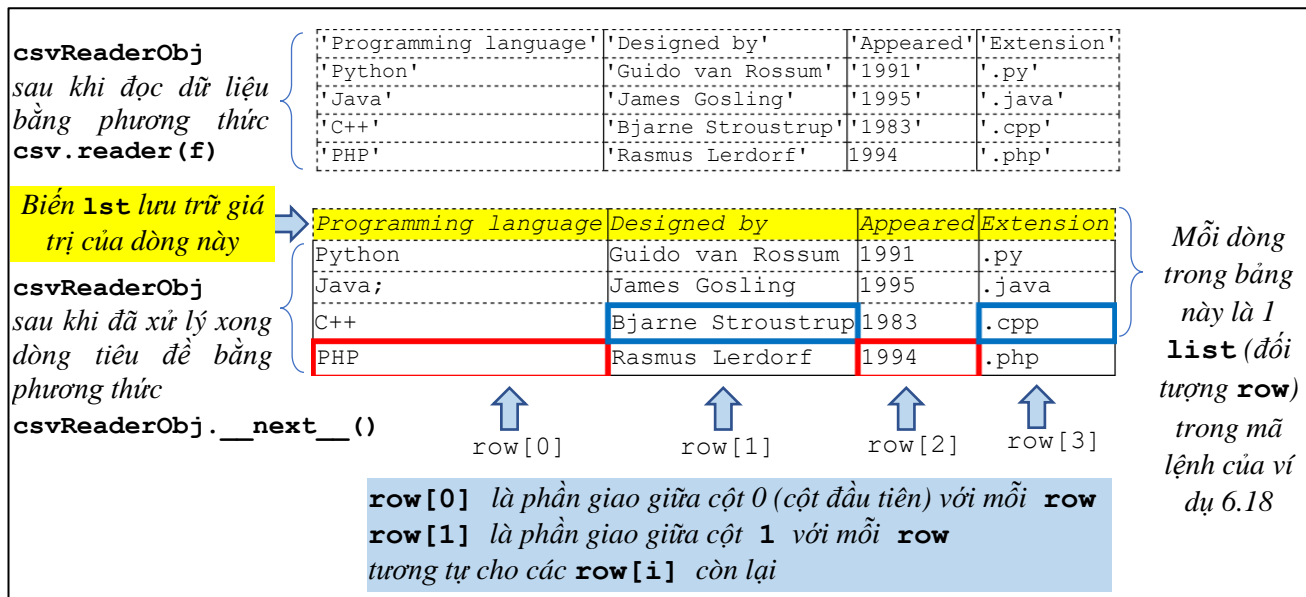
```
['Python', ' Guido van Rossum', ' 1991', ' .py']
['Java', ' James Gosling', ' 1995', ' .java']
['C++', ' Bjarne Stroustrup', '1983', '.cpp']
['PHP', ' Rasmus Lerdorf', ' 1994', ' .php']
Tổng cộng gồm 4 dòng dữ liệu
Tên các cột lần lượt là: Programming language, Designed by, Appeared, Extension
```



Hình 6.2. Minh họa ví dụ 6.17

- **Ví dụ 6.18:** tương tự ví dụ 6.16 nhưng bổ sung lệnh cho hàm `DocNoiDungFile_CSV` để định dạng dữ liệu xuất ra màn hình.

```
import csv
def DocNoiDungFile_CSV_4(filename):
    try:
        with open(filename, 'rt', encoding='utf8') as f:
            csvReaderObj = csv.reader(f)
            # Lấy dòng đầu tiên chứa các tên field ra đưa vào 1 list
            lst=csvReaderObj.__next__()
            #Sau khi lệnh trên thực hiện, biến lst có dạng
            #['Programming language', 'Designed by', 'Appeared', 'Extension']
            #In tiêu đề cột
            print('|',lst[0].center(20),'|', lst[1].center(20), '|',
                  lst[2].center(10), '|', lst[3].center(13),'|')
            print('_' * 75)
            for row in csvReaderObj:
                print('|{: <21s}|'.format(row[0]),'|',
                      '{: <20s}'.format(row[1]),
                      '| %10s | %13s |' % (row[2], row[3]))
            print(f'Dữ liệu gồm {csvReaderObj.line_num-1} dòng')
    except FileNotFoundError:
        print("Không tìm thấy tập tin %s" % filename)
#-----
DocNoiDungFile_CSV_4('D:\\ProgrammingLanguage.csv')
```



Hình 6.3. Minh họa ví dụ 6.19

Kết quả khi thực hiện chương trình trên:

| Programming language | Designed by | Appeared | Extension |
|----------------------|-------------------|----------|-----------|
| Python | Guido van Rossum | 1991 | .py |
| Java | James Gosling | 1995 | .java |
| C++ | Bjarne Stroustrup | 1983 | .cpp |
| PHP | Rasmus Lerdorf | 1994 | .php |

Dữ liệu gồm 4 dòng

- **Ví dụ 6.19:** sử dụng `csv.reader` đọc từ 1 *list*:

| Mã lệnh | Kết quả |
|---|------------------------------------|
| <pre>import csv for row in csv.reader(['one', 'two', 'three']): print (row)</pre> | <pre>['one', 'two', 'three']</pre> |

6.3.3.2. Đọc file CSV với `csv.DictReader`

- **Cú pháp:** `csv.DictReader (csvfile [, delimiter=';', **fmtparams])`
- **Giải thích:**
 - *csvfile* có thể là file CSV hoặc các đối tượng dạng danh sách (*string*, *list*, ...).
 - *delimiter*: là 1 ký tự được sử dụng làm dấu phân cách giữa các cột. Mặc định là dấu phẩy (.). Tham số này quan trọng khi dấu phân cách trong file không phải là dấu phẩy(,) hoặc chấm phẩy(;).
 - `csv.DictReader()` được sử dụng để đọc nội dung file CSV theo từng dòng dữ liệu của file mỗi dòng là 1 *list* và mỗi thành phần của *list* là 1 *dictionary object*, với *key* là tên field của bảng dữ liệu và *value* là giá trị của dòng đó tại field tương ứng.
- **Ví dụ 6.20:** viết hàm sử dụng `csv.DictReader` đọc file `ProgrammingLanguage.csv` và in nội dung các dòng ra màn hình:

```
import csv
try:
    with open(filename, 'rt', encoding='utf8') as f:
        csvreader = csv.DictReader(f)
        for row in csvreader:
            print(row) # xuất ra dưới dạng list
            # print(', '.join(row)) # xuất ra dưới dạng chuỗi
except FileNotFoundError:
    print("Không tìm thấy tập tin %s" % filename)
DocNoiDungFile_CSV('D:\\ProgrammingLanguage.csv')
```

Kết quả khi thực hiện chương trình trên:

```
OrderedDict([('Programming language', 'Python'), (' Designed by', ' Guido van Rossum'),
            (' Appeared', ' 1991'), (' Extension', ' .py')])
OrderedDict([('Programming language', 'Java'), (' Designed by', ' James Gosling'),
            (' Appeared', ' 1995'), (' Extension', ' .java')])
OrderedDict([('Programming language', 'C++'), (' Designed by', ' Bjarne Stroustrup'),
            (' Appeared', '1983'), (' Extension', ' .cpp')])
OrderedDict([('Programming language', 'PHP'), (' Designed by', ' Rasmus Lerdorf'),
            (' Appeared', ' 1994'), (' Extension', ' .php')])
```

- **Ví dụ 6.21:** tương tự ví dụ 6.17, nhưng trình bày lại kết quả khi in ra màn hình:

```
import csv
def DocNoiDungFile_CSV(filename):
    try:
        with open(filename, 'rt', encoding='utf8') as f:
            csvreader = csv.DictReader(f)
            stt = 1
            for row in csvreader:
                print('%d.-' % stt, end='')
                for k, v in row.items():
                    print('\t%20s : %s' % (k, v))
                stt += 1
    except FileNotFoundError:
        print("Không tìm thấy tập tin %s" % filename)

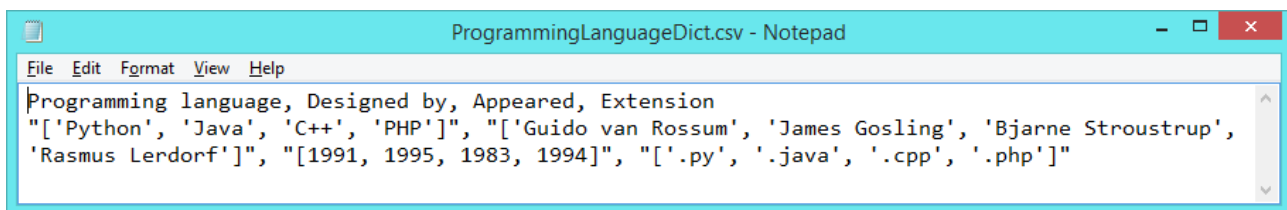
DocNoiDungFile_CSV('D:\\ProgrammingLanguage.csv')
```

Kết quả khi thực hiện chương trình trên:

- 1.- Programming language : Python
 Designed by : Guido van Rossum
 Appeared : 1991
 Extension : .py
- 2.- Programming language : Java
 Designed by : James Gosling
 Appeared : 1995
 Extension : .java
- 3.- Programming language : C++
 Designed by : Bjarne Stroustrup
 Appeared : 1983
 Extension : .cpp
- 4.- Programming language : PHP
 Designed by : Rasmus Lerdorf
 Appeared : 1994
 Extension : .php

- **Ví dụ 6.22:** viết hàm sử dụng `csv.DictReader` đọc file `ProgrammingLanguageDict.csv` và in nội dung các dòng ra màn hình.

Biết file `ProgrammingLanguageDict.csv` có nội dung như hình trên, trong đó dữ liệu gồm 2 dòng: dòng đầu chứa tiêu đề các cột của bảng dữ liệu, dòng 2: gồm n chuỗi, với n là số cột của bảng dữ liệu. Mỗi chuỗi trong n chuỗi đó là giá trị trên từng cột của bảng dữ liệu, các giá trị này lại được đặt trong 1 list trước khi được bao bởi cặp dấu nháy đôi:



```
def DocFile(fileName):
    try:
        with open(fileName, 'r') as csvfile:
            csvReaderObj = csv.DictReader(csvfile)
            for row in csvReaderObj:
                print(row)
    except FileNotFoundError:
        print("Không tìm thấy tập tin %s" % fileName)
```

Khi thực hiện chương trình trên, kết quả sẽ hiện ra trên duy nhất 1 dòng, với nội dung xuất hiện như sau :

```
OrderedDict([('Programming language', "['Python', 'Java', 'C++', 'PHP']"), ('Designed by', "['Guido van Rossum', 'James Gosling', 'Bjarne Stroustrup', 'Rasmus Lerdorf']"), ('Appeared', '[1991, 1995, 1983, 1994]'), ('Extension', "['.py', '.java', '.cpp', '.php']")])
```

6.3.4. Ghi file CSV

- Cú pháp:
 - Tạo *csvwriter object*:

```
with open(fileName, mode='w', encoding='utf8', newline='') as csvfile:
    csvWriterObject = csv.writer(csvfile, [delimiter=','])
    '''sử dụng các lệnh csvWriterObject.writerow hoặc csvWriterObject.writerows
    để ghi nội dung vào file'''
```

Giải thích tham số hàm open

- *newline=""*: nếu được dùng, các dòng trong file kết quả sẽ không bị cách nhau bởi 1 dòng trống.
- *encoding='utf8'*: được dùng khi dữ liệu cần ghi là UNICODE.
- Ghi dữ liệu vào file
 - Ghi 1 dòng dữ liệu vào *csvwriter object*:

```
csvWriterObject.writerow(listObject, [delimiter=','])
```

- Ghi tất cả các dòng vào *csvwriter object* trong 1 lần ghi. Phương thức này chỉ sử dụng trên 1 List mà trong đó các phần tử là 1 subList:

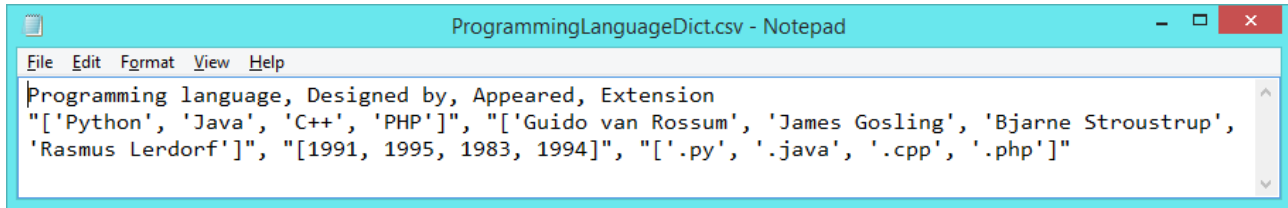
```
csvWriterObject.writerows(listObject)
```

Giải thích:

- *delimiter*: là 1 ký tự được sử dụng làm dấu phân cách giữa các cột. Mặc định là dấu phẩy (,).
 - *listObject*: tham số của phương thức phải là 1 list object.
- **Ví dụ 6.23:** Viết hàm tạo file *ProgrammingLanguage.csv* với nội dung đã biết và dấu phân cách (*delimiter*) là dấu chấm phẩy (;). Lưu ý, trong phương thức *writerow* vẫn dùng dấu phẩy (,) để phân cách.

```
import csv
def GhiNoiDungFile_CSV(filename):
    try:
        with open(filename, mode='w', encoding='utf8', newline='') as f:
            csvWriterObj = csv.writer(f, delimiter=',')
            # way to write to csv file
            csvWriterObj.writerow(['Programming language',
                                   'Designed by', 'Appeared', 'Extension'])
            csvWriterObj.writerow(['Python', 'Guido van Rossum',
                                   '1991', '.py'])
            csvWriterObj.writerow(['Java', 'James Gosling',
                                   '1995', '.java'])
            csvWriterObj.writerow(['C++', 'Bjarne Stroustrup',
                                   '1985', '.cpp'])
            csvWriterObj.writerow(['PHP', 'Rasmus Lerdorf',
                                   '1994', '.php'])
    except FileNotFoundError:
        print("Không tìm thấy tập tin %s" % filename)
GhiNoiDungFile_CSV('ProgrammingLanguage.csv')
```

- **Ví dụ 6.24:** Viết hàm tạo file *ProgrammingLanguageDict.csv* với nội dung như hình sau:



```
def TaoFile(filename):
    #Khai báo 1 list chứa các tên field
    csv_columns = ['Programming language', 'Designed by', 'Appeared',
                  'Extension']
    '''Khai báo 1 dictionary với số thành phần = số lượng field
        Mỗi thành phần sẽ gồm key chính là tên field: value là 1 list chứa
        các giá trị có trên field của tất cả các dòng dữ liệu'''
    dict_data = {'Programming language': ['Python', 'Java', 'C++', 'PHP'],
                 'Designed by': ['Guido van Rossum', 'James Gosling',
                                'Bjarne Stroustrup', 'Rasmus Lerdorf'],
                 'Appeared': [1991, 1995, 1983, 1994],
                 'Extension': ['.py', '.java', '.cpp', '.php'] }

    try:
        with open(filename, 'w', newline='') as csvfile:
            writer = csv.DictWriter(csvfile, fieldnames=csv_columns)
            writer.writeheader()
            writer.writerow(dict_data)
    except IOError:
        print("I/O error")

TaoFile("D:\\ProgrammingLanguageDict.csv")
```

6.5. Thao tác với thư mục (Directory) qua module OS⁶

- Module **os** là một module có nhiều phương thức hữu ích trong việc thao tác với các file và directory như:

| Chức năng | Cú pháp | Ví dụ |
|--|---|--|
| Tạo thư mục | os.mkdir(path) | <pre>import os os.mkdir('D:\\NewFolder')</pre> |
| Tạo thư mục theo đường dẫn. Tương tự như mkdir nhưng nếu các thư mục con trên path chưa tồn tại thì phương thức này cũng tạo ra luôn | os.makedirs(path) | |
| Xóa thư mục (hoặc tập tin) | os.rmdir(path) | <pre>import os os.rmdir('D:/MyFolder/baitap.py') os.rmdir('D:/MyFolder')</pre> |
| Xoá một đường dẫn. Tương tự rmdir , nhưng nếu xoá thành công thư mục lá, removedirs sẽ cố gắng xoá liên tiếp mọi thư mục cha được hiển thị trong path . | os.removedirs(path) | |
| Thay đổi thư mục hiện hành | os.chdir(path) | <pre>import os os.chdir('D:/NewFolder')</pre> |
| Lấy danh sách tập tin, thư mục | os.listdir(path) | <pre>from os import listdir files_list = [f for f in listdir('d:/')] print(files_list);</pre> |
| Lấy đường dẫn và tên thư mục hiện hành | os.getcwd() | <pre>import os print(os.getcwd())</pre> |
| Trả về tên file sau khi loại bỏ phần mở rộng | os.basename(path) | |
| Kiểm tra một file (hoặc thư mục) có tồn tại trong thư mục pathname hay không? Nếu không có đường dẫn đi kèm, sẽ tìm trong thư mục hiện hành? Kết quả trả về True hoặc False | os.path.exists(pathname) | <pre>import os print(os.path.exists('d:\\ baitap.py')) hoặc print(os.path.exists('baitap.py'))</pre> |
| Copy file | os.popen('copy SourceFile DestFile') | <pre>import os os.popen('copy D:\\OldFile.txt D:\\Newfile.txt')</pre> |

⁶ Xem thêm tại <https://docs.python.org/2/library/os.html>

| | | |
|------------------------|--|--|
| Đổi tên/di chuyển file | <code>os.rename(OldName, NewName)</code> | <pre>import os import os os.rename('D:\\cu', 'D:\\moi') hoặc os.rename('Old.txt', 'New.txt')</pre> |
|------------------------|--|--|

- Một số phương thức khác trong module `os`:
 - `os.system('cls')` : xóa màn hình (console)
 - `os.name` : cho tên hệ điều hành đang dùng

6.6. Module `os.path`⁷

Do các hệ điều hành khác nhau có các quy ước tên đường dẫn khác nhau, vì vậy module `os.path` hỗ trợ các phương thức giúp thao tác nhanh chóng và thuận tiện hơn trên đường dẫn.

| Phương thức/Cú pháp | Ý nghĩa |
|-------------------------------------|--|
| <code>os.path.exists(path)</code> | Kiểm tra đường dẫn của <i>path</i> có tồn tại hay không? |
| <code>os.path.isfile(path)</code> | Kiểm tra xem <i>path</i> có phải là file thông thường không? |
| <code>os.path.isdir(path)</code> | Kiểm tra xem <i>path</i> có phải là một thư mục hay không? |
| <code>os.path.dirname(path)</code> | Trả về tên thư mục của <i>path</i> |
| <code>os.path.getctime(path)</code> | Trả về thời gian chỉnh sửa cuối của metadata trên hệ thống. Hoặc trả về thời gian tạo trên Windows |
| <code>os.path.getfile(path)</code> | Lấy file size (byte) của <i>path</i> |
| <code>os.path.getsize(path)</code> | lấy file size (byte). |
| <code>os.path.getatime(path)</code> | Trả về thời gian truy cập mới nhất |
| <code>os.path.getmtime(path)</code> | Trả về thời gian chỉnh sửa cuối cùng |
| <code>os.path.basename(path)</code> | Tách lấy tên file (kể cả phần mở rộng) ra khỏi chuỗi <i>path</i> . Nếu <i>path</i> chỉ gồm đường dẫn và không chứa tên file, phương thức sẽ trả về chuỗi rỗng. |
| <code>os.path.splitext(path)</code> | Trả về 1 tuple gồm 2 thành phần chứa kết quả của phép tách <i>path</i> thành 2 phần dựa trên dấu chấm phân cách giữa phần tên và phần mở rộng của tên file. |

Ví dụ 6.25:

```
print(os.path.basename('D:/python/baitap01.py'))    ⇒ baitap01.py
print(os.path.basename('D:/python/ '))              ⇒ kết quả là chuỗi rỗng
mytuple= os.path.splitext('D:/python/baitap01.py')
                                                    ⇒ ('D:/python/baitap01', '.py')
print(os.path.splitext(os.path.basename('D:/python/baitap01.py')))[0]
                                                    ⇒ baitap01
```

⁷ Xem thêm tại <https://docs.python.org/2/library/os.path.html>

THAM KHẢO

Ebook:

- [1] Dave Kuhlman – A Python Book: Beginning Python, Advanced Python, and Python Exercises

Nguồn Internet:

- [1] <https://docs.python.org/>
- [2] <https://www.programiz.com/python-programming>
- [3] <https://www.tutorialspoint.com/python3>
- [4] <https://www.tutorialspoint.com/pycharm>
- [5] <https://thepythonguru.com>
- [6] https://www.w3schools.com/python/python_exercises.asp
- [7] <https://www.practicepython.org/>
- [8] <https://www.pythonforbeginners.com/games/>
- [9] <https://quantrimang.com/python>