(Ghi chú)

Ngôn ngữ lập trình Python

**Thực hiện: Nguyễn Văn Hào**

**Email:** [hao2205tb@gmail.com](mailto:hao2205tb@gmail.com)

(Tài liệu được biên xoạn lại từ nhiều nguồn và không dùng trong mục đích kinh doanh kiếm tiền)

# Mục lục

[Mục lục 1](#_Toc128349135)

[Tổng quan 10](#_Toc128349136)

[Đặc điểm của python 11](#_Toc128349137)

[Học Python để làm gì và khi nào thì nên học Python 11](#_Toc128349138)

[Phát triển phần mềm chạy trên máy tính 11](#_Toc128349139)

[Phát triển trí tuệ nhân tạo 11](#_Toc128349140)

[Phân tích dữ liệu 11](#_Toc128349141)

[Người không nên học Python 12](#_Toc128349142)

[Tạo ứng dụng điện thoại thông minh (iPhone, Android) 12](#_Toc128349143)

[Phát triển ứng dụng và dịch vụ web 12](#_Toc128349144)

[Phát triển hệ thống/ Ứng dụng nhúng. 12](#_Toc128349145)

[Hello world 13](#_Toc128349146)

[Chương trình đầu tiên 13](#_Toc128349147)

[Comment 13](#_Toc128349148)

[Đặt mã ký tự: 13](#_Toc128349149)

[print() 14](#_Toc128349150)

[Cú pháp: 14](#_Toc128349151)

[Căn trái, phải, giữa: 14](#_Toc128349152)

[Chèn biến vào chuỗi sử dụng toán tử % 15](#_Toc128349153)

[Định dạng số khi in ra màn hình trong Python. 15](#_Toc128349154)

[In list, tuple và dictionary 15](#_Toc128349155)

[Pprint(): prety-print 16](#_Toc128349156)

[So sánh với print: 16](#_Toc128349157)

[Cú pháp 17](#_Toc128349158)

[Chỉ định số ký tự tối đa trên mỗi dòng in | đối số width 17](#_Toc128349159)

[Chỉ định số phần tử con tối đa được in trong dữ liệu lồng nhau | đối số depth 18](#_Toc128349160)

[Chỉ định độ thụt lề trên mỗi dòng in | đối số indent 18](#_Toc128349161)

[Giảm thiểu tối đa số lần ngắt dòng khi in các list có nhiều phần tử | Đối số compact 19](#_Toc128349162)

[input() 20](#_Toc128349163)

[Nhập dữ liệu 20](#_Toc128349164)

[Nhập cùng lúc nhiều dữ liệu trên một dòng 20](#_Toc128349165)

[Nhập nhiều dữ liệu trên nhiều dòng 20](#_Toc128349166)

[Nhập số 20](#_Toc128349167)

[Số trong python 21](#_Toc128349168)

[Số nguyên int 21](#_Toc128349169)

[Biểu diễn số nguyên dưới dạng nhị phân (binary) , 8 (octal) và 16 (Hex – hexadecimal): 22](#_Toc128349170)

[Biến đổi, ép kiểu dữ liệu về số nguyên 22](#_Toc128349171)

[Kiểu dữ liệu số thực float trong python | float python là gì 22](#_Toc128349172)

[Sử dụng số e 23](#_Toc128349173)

[Ép kiểu dữ liệu về kiểu float 23](#_Toc128349174)

[Kiểu dữ liệu số phức complex trong python 23](#_Toc128349175)

[Toán tử số học 24](#_Toc128349176)

[Toán tử bit 25](#_Toc128349177)

[Phép dịch bit 25](#_Toc128349178)

[Thứ tự ưu tiên toán tử 26](#_Toc128349179)

[Bảng ưu tiên 26](#_Toc128349180)

[Làm tròn số 27](#_Toc128349181)

[Round() 27](#_Toc128349182)

[Số pi 28](#_Toc128349183)

[Kiểm tra số chẵn lẻ 29](#_Toc128349184)

[Kiểm tra số nguyên, số thực 30](#_Toc128349185)

[Kiểm tra số nguyên số thực trong python | Hàm isinstance() 30](#_Toc128349186)

[Kiểm tra số thực có phải là số nguyên trong python | Phương thức is\_integer() 31](#_Toc128349187)

[Kiểm tra số nguyên tố 31](#_Toc128349188)

[Kiểm tra số chính phương 32](#_Toc128349189)

[Số hoàn hảo 33](#_Toc128349190)

[Số đối xứng 33](#_Toc128349191)

[Biến 34](#_Toc128349192)

[Khái niệm 34](#_Toc128349193)

[Khởi tạo 34](#_Toc128349194)

[Xoá biến 35](#_Toc128349195)

[Lệnh điều kiện 35](#_Toc128349196)

[If 35](#_Toc128349197)

[Toán tử so sánh 36](#_Toc128349198)

[Toán tử logic 37](#_Toc128349199)

[Toán tử 3 ngôi 37](#_Toc128349200)

[Vòng lặp 37](#_Toc128349201)

[For 37](#_Toc128349202)

[While 38](#_Toc128349203)

[While else break 39](#_Toc128349204)

[Break và continue 39](#_Toc128349205)

[Pass 39](#_Toc128349206)

[Range() 40](#_Toc128349207)

[True, False trong python 40](#_Toc128349208)

[Chuỗi 41](#_Toc128349209)

[Khai báo: 41](#_Toc128349210)

[Khai báo sử dụng dấu nháy kép “” hoặc ‘’ 41](#_Toc128349211)

[Khai báo chuỗi nhiều dòng bằng 3 nháy đơn hoặc 3 nháy kép 41](#_Toc128349212)

[Khai báo chuỗi nhiều dòng chứa dấu nháy đơn ' hoặc dấu nháy kép " 41](#_Toc128349213)

[Ứng dụng của chuỗi nhiều dòng trong python 42](#_Toc128349214)

[Độ dài chuỗi 42](#_Toc128349215)

[Tính độ dài chuỗi bằng len: 42](#_Toc128349216)

[Nối chuỗi 42](#_Toc128349217)

[Nối chuỗi bằng phương thức join() 43](#_Toc128349218)

[Lặp chuỗi 43](#_Toc128349219)

[Cắt chuỗi (Slice string) 43](#_Toc128349220)

[Tách chuỗi 44](#_Toc128349221)

[Tách chuỗi bằng split() 44](#_Toc128349222)

[Tách chuỗi bằng splitlines() 45](#_Toc128349223)

[Regular Expression 46](#_Toc128349224)

[Khái niệm 46](#_Toc128349225)

[Regular expression module 47](#_Toc128349226)

[Hàm RegEx 47](#_Toc128349227)

[Siêu ký tự (Metacharacters) 47](#_Toc128349228)

[Chuỗi đặc biệt(Special Sequences) 48](#_Toc128349229)

[Bộ (sets) 49](#_Toc128349230)

[Hàm findall() 49](#_Toc128349231)

[Hàm search() 50](#_Toc128349232)

[Hàm split() 50](#_Toc128349233)

[Hàm sub() 51](#_Toc128349234)

[Phân biệt các ký tự/chuỗi bắt đầu/kết thúc với ^, $, /A, /b. 51](#_Toc128349235)

[Tham khảo: 53](#_Toc128349236)

[Tách số trong chuỗi 53](#_Toc128349237)

[Tách dãy số trong chuỗi python dưới dạng một list | hàm re.findall() 53](#_Toc128349238)

[Tách số trong chuỗi python dưới dạng một chuỗi | hàm re.sub() 54](#_Toc128349239)

[Tách số đầu tiên trong chuỗi python | hàm re.search() 54](#_Toc128349240)

[Tách dãy số đầu tiên xuất hiện trong chuỗi python | hàm re.search() python 55](#_Toc128349241)

[Xoá ký tự trong chuỗi 55](#_Toc128349242)

[strip() xoá các ký tự đầu và cuối trong chuỗi 55](#_Toc128349243)

[Rstrip() và lstrip() 56](#_Toc128349244)

[Sử dụng replace() để xoá các ký tự trong cả chuỗi 57](#_Toc128349245)

[Sử dụng slice để xoá chuỗi 57](#_Toc128349246)

[Tìm chuỗi con trong chuỗi 58](#_Toc128349247)

[Find() 58](#_Toc128349248)

[Rfind() 58](#_Toc128349249)

[Index() 59](#_Toc128349250)

[Rindex() 60](#_Toc128349251)

[Index dương, index âm 60](#_Toc128349252)

[index dương 60](#_Toc128349253)

[index âm 61](#_Toc128349254)

[Thay thế ký tự trong chuỗi 61](#_Toc128349255)

[Replace() 61](#_Toc128349256)

[Đếm số lần xuất hiện chuỗi con trong chuỗi. 61](#_Toc128349257)

[Count() 61](#_Toc128349258)

[Kiểm tra chuỗi có phải là chữ hay không trong python 62](#_Toc128349259)

[Isascii() 62](#_Toc128349260)

[Isalpha() 62](#_Toc128349261)

[Isalnum() 64](#_Toc128349262)

[Kiểm tra chuỗi có phải là số 65](#_Toc128349263)

[Isdecimal() 65](#_Toc128349264)

[So sánh isdecimal, isdigit, isnumeric 66](#_Toc128349265)

[Chuyển số thành chuỗi: 66](#_Toc128349266)

[Chuyển list thành chuỗi: 67](#_Toc128349267)

[Sử dụng join() 67](#_Toc128349268)

[Sử dụng RegEx 67](#_Toc128349269)

[Sử dụng hàm map() 67](#_Toc128349270)

[Kiểm tra chữ hoa, chữ thường 68](#_Toc128349271)

[Kiểm tra chữ hoa 68](#_Toc128349272)

[Kiểm tra chữ thường 68](#_Toc128349273)

[Kiểm tra tiêu đề 68](#_Toc128349274)

[Chuyển chữ hoa thành chữ thường và ngược lại trong python 69](#_Toc128349275)

[Chuyển chữ hoa thành chữ thường trong python 69](#_Toc128349276)

[Chuyển chữ thường thành chữ hoa trong python 69](#_Toc128349277)

[Phương thức capitalize trong python 70](#_Toc128349278)

[Phương thức title trong python 70](#_Toc128349279)

[Phương thức swapcase trong python 70](#_Toc128349280)

[Căn lề 71](#_Toc128349281)

[Căn lề trái một chuỗi trong Python | phương thức ljust() 71](#_Toc128349282)

[Căn lề giữa một chuỗi trong Python |phương thức center() 71](#_Toc128349283)

[Căn lề phải một chuỗi trong Python |phương thức rjust() 72](#_Toc128349284)

[Chỉ định width nhỏ hơn hoặc bằng độ dài chuỗi ban đầu khi sử dụng các phương thức ljust, center, rjust 72](#_Toc128349285)

[Căn lề phải một chuỗi và lấp khoảng trắng trong chuỗi bằng số 0 | phương thức zfill() 72](#_Toc128349286)

[Đảo ngược chuỗi 73](#_Toc128349287)

[Đảo ngược chuỗi trong python| Phương thức reverse() 73](#_Toc128349288)

[Đảo ngược chuỗi trong python | Hàm reversed() 74](#_Toc128349289)

[Đảo ngược chuỗi trong python | Cắt (slice) chuỗi 75](#_Toc128349290)

[So sánh chuỗi 75](#_Toc128349291)

[Kiểm tra mã unicode của một ký tự 75](#_Toc128349292)

[So sánh chuỗi bằng toán tử 75](#_Toc128349293)

[Kiểm tra chuỗi con với toán tử in 76](#_Toc128349294)

[startswith(), endswith() Kiểm tra đầu chuỗi và cuối chuỗi có bắt đầu bằng một chuỗi con khác 76](#_Toc128349295)

[So sánh không phân biệt hoa thường. 77](#_Toc128349296)

[Lấy ra các phần tử giống nhau trong 2 chuỗi| sử dụng set và toán tử and 77](#_Toc128349297)

[Lấy ra các phần tử khác nhau trong 2 chuỗi| sử dụng set và toán tử ^ (XOR) (tìm phần bù) 78](#_Toc128349298)

[Ký tự đặc biệt (escape sequence) 78](#_Toc128349299)

[Bảng ký tự đặc biệt 78](#_Toc128349300)

[Loại bỏ ký tự đặc biệt 79](#_Toc128349301)

[Chuỗi thô (raw string) 80](#_Toc128349302)

[Định dạng chuỗi 81](#_Toc128349303)

[Định dạnh bằng toán tử % 81](#_Toc128349304)

[Định dạnh chuỗi bằng f 84](#_Toc128349305)

[Định dạng chuỗi bằng format 86](#_Toc128349306)

[Các phương thức xử lý chuỗi 90](#_Toc128349307)

[Ljust() 90](#_Toc128349308)

[Rjust() 90](#_Toc128349309)

[Zfill() 91](#_Toc128349310)

[Lower() 91](#_Toc128349311)

[Upper() 91](#_Toc128349312)

[Swapcase() 92](#_Toc128349313)

[Capitalize() 92](#_Toc128349314)

[Title() 92](#_Toc128349315)

[Join() 92](#_Toc128349316)

[List 93](#_Toc128349317)

[Khởi tạo 93](#_Toc128349318)

[Len() 94](#_Toc128349319)

[Index 94](#_Toc128349320)

[Truy cập bằng index 95](#_Toc128349321)

[Thay đổi giá trị bằng index 95](#_Toc128349322)

[List comprehension: phương pháp viết nội hàm 96](#_Toc128349323)

[Xoá phần tử 97](#_Toc128349324)

[Pop() 97](#_Toc128349325)

[Del 97](#_Toc128349326)

[Remove() 98](#_Toc128349327)

[Clear() 98](#_Toc128349328)

[Xoá các phần tử trùng lặp | sử dụng set 99](#_Toc128349329)

[List.sort() Xắp xếp phần tử 99](#_Toc128349330)

[So sánh phần tử 100](#_Toc128349331)

[So sánh 2 list giống nhau 100](#_Toc128349332)

[So sánh 2 list và lấy ra các phần tử giống nhau | sử dụng set và toán tử & 100](#_Toc128349333)

[So sánh 2 list và lấy ra các phần tử giống nhau | sử dụng set và toán tử ^ 100](#_Toc128349334)

[Thêm phần tử 101](#_Toc128349335)

[Append() 101](#_Toc128349336)

[Extend() 101](#_Toc128349337)

[Hàm khác 102](#_Toc128349338)

[Min() 102](#_Toc128349339)

[Max() 102](#_Toc128349340)

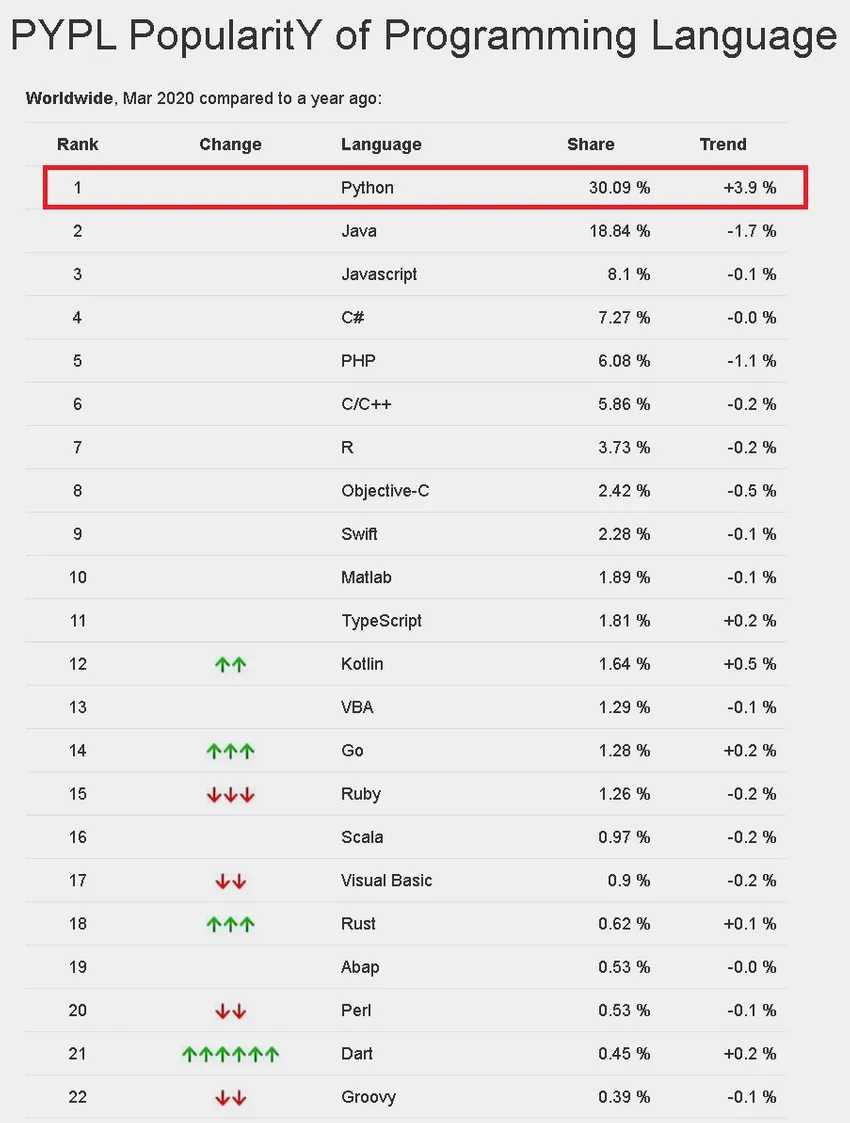
[Sorted() 103](#_Toc128349341)

[Tham khảo 104](#_Toc128349342)

# Tổng quan

Python ban đầu được phát triển bởi Guido van Rossum vào năm 1991, nhằm quản lý một hệ điều hành có tên Amoeba.

Python đang là ngôn ngữ lập trình được ưu chuộng nhất thế giới.



## Đặc điểm của python

Tham khảo: <http://laptrinhcanban.com/python/nhap-mon-lap-trinh-python/gioi-thieu-python/python-la-gi/>

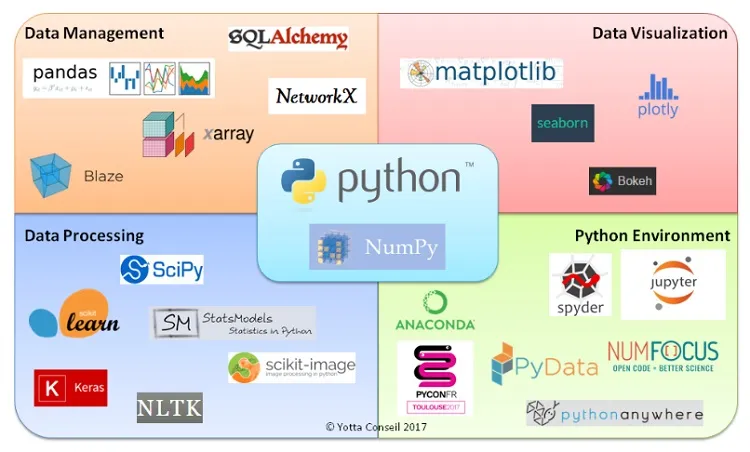
## Học Python để làm gì và khi nào thì nên học Python

### Phát triển phần mềm chạy trên máy tính

Tạo phần mềm tìm kiếm một file trong ổ cứng, chỉnh sửa ảnh, tạo video mp4 từ ảnh chụp sẵn. Xử lý nhanh chóng từ các file dữ liệu như excel, json đến các file âm thanh, hình ảnh thông dụng …

### Phát triển trí tuệ nhân tạo

Python được tích hợp rất nhiều thư viện được sử dụng cho nghiên cứu, phát triển deep learning.



### Phân tích dữ liệu

Làm việc với Big DATA

Python, với library điển hình là Pandas giúp bạn thực hiện các lệnh như đọc thông tin file CSV, sau dó tiến hành thêm, sửa, xóa và phân tích chúng một cách dễ dàng và hiệu quả.

## Người không nên học Python

### Tạo ứng dụng điện thoại thông minh (iPhone, Android)

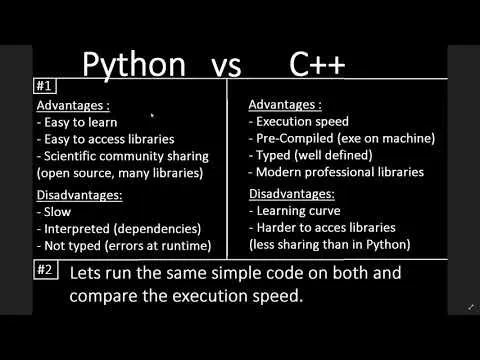
Nếu số sánh với Swift của iPhone hay Java của Android, và gần đây là Kotlin chẳng hạn, thì Python không có lợi thế trong lĩnh vực này.

### Phát triển ứng dụng và dịch vụ web

Tại Nhật Bản, thì hầu như các dự án tạo web được viết bởi Ruby và PHP. Do đó python tuy có tiềm năng nhưng thực tế chưa được ứng dụng nhiều như một ngôn ngữ chính để tạo web, mà được sử dụng chủ yếu để quản lý dữ liệu của trang web.

### Phát triển hệ thống/ Ứng dụng nhúng.

Do là ngôn ngữ tự dịch ( interpreter) - việc dịch python ra ngôn ngữ mà máy tính hiểu và việc thực hiện tác vụ được thực hiện song song, nên tốc độ của python trong hệ thống sẽ không thể so sánh được với các ngôn ngữ thông dụng khác đang được sử dụng như C/C++.



# Hello world

### Chương trình đầu tiên

Chương trình python được viết trong file .py

Nội dung chương trình in ra dòng hello world:

File: main.py

print("Hello world!")

Chạy chương trình trên cmd: python main.py

### Comment

Comment 1 dòng:

# comment

Comment nhiều dòng: dùng nháy kép hoặc nháy đơn đều được.

“““

Comment line 1

Comment line 2

”””

Hoặc

‘‘‘

Comment line 1

Comment line 2

’’’

## Đặt mã ký tự:

Tham khảo

<http://laptrinhcanban.com/python/nhap-mon-lap-trinh-python/kien-thuc-can-ban-ve-chuong-trinh-python/dat-ma-ky-trong-file-python/>

## print()

### Cú pháp:

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

Trong đó:

* \*objects : đối tượng (dữ liệu) cần in ra màn hình. Dấu \* có ý nghĩa là số nhiều và chúng ta cũng có thể chỉ định nhiều đối tượng khác nhau và in chúng cùng lúc ra màn hình.
* sep: đối tượng chỉ định sẽ được phân tách thành các phần nhỏ bằng ký tự phân tách sep trước khi được in, và mặc định giá trị này là một khoảng trắng ' '.
* end: giá trị cuối cùng được in ra màn hình, và mặc định giá trị này là ký tự xuống dòng \n. Đối số này sẽ quyết định việc in xuống dòng hay in không xuống dòng trong Python.
* file=sys.stdout: chỉ định lưu kết quả đầu ra vào bộ nhớ đệm sys.stdout.
* flush=False: chỉ định cưỡng chế lưu giữ kết quả vào bộ nhớ đệm, và giá trị mặc định là false, có nghĩa là KHÔNG lưu giữ kết quả vào bộ nhớ.

### Căn trái, phải, giữa:

Sử dụng ljust(), rjust(), center().

VD:

str1="học lập trình"

print("[" + str1.ljust(20) + "]")

#>> [học lập trình       ]

VD:

str1="học lập trình"

print("[" + str1.center(20) + "]")

#>> [   học lập trình    ]

### Chèn biến vào chuỗi sử dụng toán tử %

Cú pháp:

string % variable

VD:

s = 'Shin chan'

i = 2

print('Shin chan is %d years old' % i)

#>> Shin chan is 2 years old

print('%s is %d years old' % (s, i))

#>> Shin chan is 2 years old

### Định dạng số khi in ra màn hình trong Python.

VD:

print("{:,d}".format(1234567))

#>> 1,234,567

print("{:\_d}".format(1234567))

#>> 1\_234\_567

Chỉ định số chữ số đằng sau dấu phẩy và làm tròn số.

print("{:f}".format(1.2345))

#>>1.234500

print("{:.1f}".format(1.2345))

#>> 1.2

### In list, tuple và dictionary

VD:

l = [0, 1, 2]

print(l)

#>> [0, 1, 2]

t = (0, 1, 2)

print(t)

#>> (0, 1, 2)

d = { 1: 'red', 2: 'green' , 3: 'black'}

print(d)

#>> {1: 'red', 2: 'green', 3: 'black'}

Phân tách phần tử sử dụng ký hiệu \* vào trước list, tuple hay dictionary.

VD:

l = [0, 1, 2]

print(\*l)

#>> 0 1 2

t = (0, 1, 2)

print(\*t)

#>> 0 1 2

d = { 1: 'red', 2: 'green' , 3: 'black'}

print(\*d)

#>> 0 1 2

## Pprint(): prety-print

<http://laptrinhcanban.com/python/nhap-mon-lap-trinh-python/nhap-xuat-trong-python/ham-pprint-trong-python-va-cach-in-list-va-tu-dien-thong-minh/>

### So sánh với print:

Ví dụ, chúng ta có một list có cấu trúc phức tạp như sau:

l = [{'Name': 'Yamada', 'Age': 20, 'Points': [80, 20]},

{'Name': 'Kiyoshi', 'Age': 30, 'Points': [90, 10]},

{'Name': 'Honda', 'Age': 40, 'Points': [70, 30]}]

Kết quả khi sử dụng print

print(l)

#>> [{'Name': 'Yamada', 'Age': 20, 'Points': [80, 20]}, {'Name': 'Kiyoshi', 'Age': 30, 'Points': [90, 10]}, {'Name': 'Honda', 'Age': 40, 'Points': [70, 30]}]

Kết quả khi sử dụng pprint

import pprint

pprint.pprint(l)

#>> [{'Age': 20, 'Name': 'Yamada', 'Points': [80, 20]},

#>>  {'Age': 30, 'Name': 'Kiyoshi', 'Points': [90, 10]},

#>>  {'Age': 40, 'Name': 'Honda', 'Points': [70, 30]}]

### Cú pháp

pprint.pprint ( object, stream=None, indent=1, width=80, depth=None,compact=False, sort\_dicts=True )

Trong đó các đối số có tác dụng như sau:

* width : chỉ định chiều rộng đầu ra (số ký tự), mặc định bằng 80
* indent: chỉ định độ rộng thụt lề, mặc định bằng 1
* depth: chỉ định độ sâu của phần tử đầu ra
* compact: giảm thiểu ngắt dòng

### Chỉ định số ký tự tối đa trên mỗi dòng in | đối số width

VD:

pprint.pprint(l)

#>> [{'Age': 20, 'Name': 'Yamada', 'Points': [80, 20]},

#>>  {'Age': 30, 'Name': 'Kiyoshi', 'Points': [90, 10]},

#>>  {'Age': 40, 'Name': 'Honda', 'Points': [70, 30]}]

pprint.pprint(l, width=40)

#>> [{'Age': 20,

#>>  'Name': 'Yamada',

#>>  'Points': [80, 20]},

#>> {'Age': 30,

#>>  'Name': 'Kiyoshi',

#>>  'Points': [90, 10]},

#>> {'Age': 40,

#>>  'Name': 'Honda',

#>>  'Points': [70, 30]}]

pprint.pprint(l, width=10)

#>> [{'Age': 20,

#>>   'Name': 'Yamada',

#>>   'Points': [80,

#>>              20]},

#>>  {'Age': 30,

#>>   'Name': 'Kiyoshi',

#>>   'Points': [90,

#>>              10]},

#>>  {'Age': 40,

#>>   'Name': 'Honda',

#>>   'Points': [70,

#>>              30]}]

### Chỉ định số phần tử con tối đa được in trong dữ liệu lồng nhau | đối số depth

Độ sâu là số phần tử con tối đa được in của mỗi phần tử.

Nếu dữ liệu lồng nhau chứa số phần tử con vượt quá depth thì các phần tử con này sẽ được biểu thị bởi dấu ba chấm ... khi in ra màn hình như sau:

VD:

pprint.pprint(l, depth=1)

#>> [{...}, {...}, {...}]

pprint.pprint(l, depth=2)

#>> [{'Age': 20, 'Name': 'Yamada', 'Points': [...]},

#>>  {'Age': 30, 'Name': 'Kiyoshi', 'Points': [...]},

#>>  {'Age': 40, 'Name': 'Honda', 'Points': [...]}]

### Chỉ định độ thụt lề trên mỗi dòng in | đối số indent

VD:

pprint.pprint(l)

##> [{'Age': 20, 'Name': 'Yamada', 'Points': [80, 20]},

##>  {'Age': 30, 'Name': 'Kiyoshi', 'Points': [90, 10]},

##>  {'Age': 40, 'Name': 'Honda', 'Points': [70, 30]}]

pprint.pprint(l, indent=4)

##> [   {'Age': 20, 'Name': 'Yamada', 'Points': [80, 20]},

##>     {'Age': 30, 'Name': 'Kiyoshi', 'Points': [90, 10]},

##>     {'Age': 40, 'Name': 'Honda', 'Points': [70, 30]}]

### Giảm thiểu tối đa số lần ngắt dòng khi in các list có nhiều phần tử | Đối số compact

Với các list có quá nhiều phần tử, nếu tổng ký tự của các phần tử không vừa với độ dài của một dòng in thì python sẽ tự động ngắt dòng và in từng phần tử ra màn hình:

import pprint

l\_long = [list(range(10)), list(range(1000, 1010))]

print(l\_long)

#>> [[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], [1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009]]

pprint.pprint(l\_long, width=40)

#>>[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9],

#>> [1000,

#>>  1001,

#>>  1002,

#>>  1003,

#>>  1004,

#>>  1005,

#>>  1006,

#>>  1007,

#>>  1008,

#>>  1009]]

Để có thể giảm thiểu tối đa số ngắt chuỗi khi in như trên, chúng ta có thể chỉ định đối số compact = True khi sử dụng với các list có quá nhiều phần tử trong nó.

pprint.pprint(l\_long, width=40, compact=True)

#>> [[0, 1, 2, 3, 4, 5, 6, 7, 8, 9],

#>>  [1000, 1001, 1002, 1003, 1004, 1005,

#>>   1006, 1007, 1008, 1009]]

## input()

input() gửi về kiểu dữ liệu string.

### Nhập dữ liệu

dulieu = input("Hãy nhập dữ liệu:")

#>> Hãy nhập dữ liệu:123abc

print(dulieu)

#>> 123abc

### Nhập cùng lúc nhiều dữ liệu trên một dòng

s = input(">>").split()

print(s)

>> 1 23 ab

['1', '23','ab']

### Nhập nhiều dữ liệu trên nhiều dòng

s = [input(">>") for i in range(3)]

print(s)

>> 1

>> 23

>> ab

['1', '23','ab']

### Nhập số

Input gửi về kiểu dữ liệu chuỗi. Để nhận số cần ép kiểu.

VD:

val = int(input('Enter number: '))

print(val)

print(type(val))

# Enter number: 50

# 50

# <class 'int'>

# Số trong python

Kiểu dữ liệu số trong python cũng thuộc dạng Literal - một giá trị mà nó thể hiện chính nó.

## Số nguyên int

Kiểu dữ liệu số nguyên trong python bao gồm số nguyên dương (1,2,3…), số nguyên âm (-1,-2,-3…) và số 0.

Khi viết số nguyên trong một chương trình, chúng ta viết giá trị của chúng như ví dụ dưới đây:

# Số nguyên dương

1

23

# Số nguyên âm

-1

-123

# Số 0

0

Python cho phép chúng ta thêm dấu gạch chân \_ vào vị trí bất kỳ để phân chia chữ số cho dễ nhìn khi viết kiểu dữ liệu số nguyên int như sau:

10000

10\_000

1000\_0

1\_0000

### Biểu diễn số nguyên dưới dạng nhị phân (binary) , 8 (octal) và 16 (Hex – hexadecimal):

0b0110 # nhị phân

0o1234 # bát phân

0xFFAA # Thập lục phân

### Biến đổi, ép kiểu dữ liệu về số nguyên

# Gán chuỗi "15" vào biến a

a="15"

# Chuyển a từ kiểu dữ liệu chuỗi thành kiểu dữ liệu số nguyên

# Sau đó gán giá trị này vào b

b = int(a)

print(b)

#>>15

# Chuyển b từ số thập phân sang số nhị phân

print(bin(b))

#>> 0b1111

# Chuyển b từ số thập phân sang số bát phân

print(oct(b))

#>> 0o17

# Chuyển b từ số thập phân sang số thập lục phân

print(hex(b))

#>> 0xf

## Kiểu dữ liệu số thực float trong python | float python là gì

Kiểu dữ liệu số thực trong python hay còn gọi là float python là kiểu số được biểu diễn dưới dạng số thực dấu phẩy động( tiếng anh : Floating point number; tiếng Nhật:浮動小数点 ).

VD

12.49

-19.5

Chúng ta cũng có thể dùng dấu gạch chân \_ vào vị trí bất kỳ để phân chia chữ số cho dễ nhìn khi viết kiểu dữ liệu số thực float như sau:

1\_223.496\_6

-1\_143\_249.5

### Sử dụng số e

VD:

8.5e+7

2.34e-12

Trong đó, 8.5e+7 = và 2.34e-12 =.

### Ép kiểu dữ liệu về kiểu float

# Gán chuỗi "15" vào biến a

a="15"

# Chuyển a từ kiểu dữ liệu chuỗi thành kiểu dữ liệu số thực

# Sau đó gán giá trị này vào b

b = float(a)

print(b)

#>>15.0

## Kiểu dữ liệu số phức complex trong python

Kiểu dữ liệu số phức trong python là các số được biểu thị bằng phần thực và phần ảo như dưới đây:

a + bj

Nếu bạn muốn viết kiểu dữ liệu số phức trong Python, hãy tham khảo như ví dụ dưới đây:

4.2 + 5j

3 + 4j

4j

(3.2 + 4j)

VD

print("Số nguyên:",12345)

#>> Số nguyên: 12345

print("Số thực  :",2.34e-12)

#>> Số thực  : 2.34e-12

print("Số phức  :", 4.2 + 5j)

#>> Số phức  : (4.2+5j)

## Toán tử số học

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Toán tử | Ý nghĩa |  |
| 1 | (expressions...), | Hiển thị tuple, list, danh sách, từ điển |  |
|  | [expressions...], |  |  |
|  | {key: value...}, |  |  |
|  | {expressions...} |  |  |
| 2 | x[index], x[index:index], | Đăng ký, cắt, gọi, tham chiếu thuộc tính |  |
|  | x(arguments...), x.attribute |  |  |
| 3 | await x | Biểu thức await |  |
| 4 | \*\* | Lũy thừa |  |
| 5 | +x, -x, ~x | Giữ dấu, Đổi dấu,bit NOT |  |

Sử dụng toán tử số học trong python khi tính toán giữa các loại số khác nhau.

Kết quả trả về sẽ là kiểu dữ liệu có ít hạn mức nhất trong các kiểu dữ liệu ban đầu.

Thứ tự kiểu dữ liệu số có hạn mức từ thấp tới cao trong python là như sau:

Số phức < Số thực < Số nguyên

Ví dụ như khi sử dụng toán tử số học để tính toán giữa số nguyên và số thực thì kết quả trả về sẽ là số thực. Hay giữa số thực và số phức thì kết quả trả về sẽ là số phức.

VD

print("Tinh toan giua cac loai so khac nhau")

print("2 + 1.485 = " + str(2 + 1.485))

print("9 \* 5.23 = " + str(9 \* 5.23))

print("16 / 4.23 = " + str(16 / 4.23))

print("16 \*\* 4.23 = " + str(16 \*\* 4.23))

Tinh toan giua cac loai so khac nhau

2 + 1.485 = 3.4850000000000003

9 \* 5.23 = 47.07000000000001

16 / 4.23 = 3.7825059101654843

16 \*\* 4.23 = 124001.66787161745

## Toán tử bit

x & y AND

x | y OR

~x NOT

x ^ y XOR

x << n Phép toán dịch bit trái

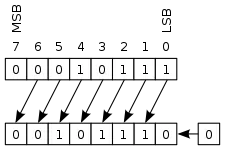
x >> n Phép toán dịch bit phải

### Phép dịch bit

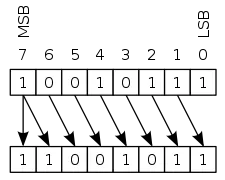
Phép toán dịch bit trong python

Khi dịch bit, các số được dịch chuyển qua đầu hoặc đuôi sẽ bị loại bỏ.

Khi dịch chuyển bit qua trái, số 0 sẽ được thêm vào bên phải để lấp chỗ trống để lại.



Khi dịch chuyển bit qua phải, bit thể hiện dấu sẽ được thêm vào bên trái để lấp chỗ trống để lại, và dấu của số sẽ được giữ nguyên.



Tham khảo: <http://laptrinhcanban.com/python/nhap-mon-lap-trinh-python/so-trong-python/toan-tu-thao-tac-bit-trong-python/>

## Thứ tự ưu tiên toán tử

### Bảng ưu tiên

|  |  |  |
| --- | --- | --- |
| STT | Toán tử | Ý nghĩa |
| 1 | (expressions...), | Hiển thị tuple, list, danh sách, từ điển |
|  | [expressions...], |  |
|  | {key: value...}, |  |
|  | {expressions...} |  |
| 2 | x[index], x[index:index], | Đăng ký, cắt, gọi, tham chiếu thuộc tính |
|  | x(arguments...), x.attribute |  |
| 3 | await x | Biểu thức await |
| 4 | \*\* | Lũy thừa |
| 5 | +x, -x, ~x | Giữ dấu, Đổi dấu,bit NOT |
| 6 | \*, @, /, //, % | Nhân, matrix, chia, chia làm tròn và chia lấy dư |
| 7 | +, - | Cộng trừ |
| 8 | <<, >> | Phép toán dịch bit |
| 9 | & | bit AND |
| 10 | ^ | bit XOR |
| 11 |  | bit OR |
| 12 | in, not in, is, is not, | So sánh |
|  | <, <=, >, >=, !=, == |  |
| 13 | not x | Boolean NOT |
| 14 | and | Boolean AND |
| 15 | or | Boolean OR |
| 16 | if -- else | Biểu thức điều kiện |
| 17 | lambda | Biểu thức Lambda |

## Làm tròn số

### Round()

round(number , ndigits)

Trong đó

* number là số cần làm tròn
* ndigits là vị trí thập phân muốn làm tròn. Nếu không chỉ định ndigits thì mặc định ndigits bằng 0.

Nếu không chỉ định ndigits, kết quả trả về kiểu int, ngược lại, kết quả trả về kiểu float.

VD:

a = 123.456

b= round(a, 2)

print(type(b), b)

b= round(a, 0)

print(type(b), b)

b= round(a)

print(type(b), b)

b= round(a, -2)

print(type(b), b)

<class 'float'> 123.46

<class 'float'> 123.0

<class 'int'> 123

<class 'float'> 100.0

Tham khảo: <http://laptrinhcanban.com/python/nhap-mon-lap-trinh-python/so-trong-python/lam-tron-so-trong-python-bang-ham-round/>

## Số pi

VD:

import math

math.pi

Lấy số pi trong python

import math

math.piimport math

print(math.pi)

##> 3.141592653589793

Tính sin bằng số pi trong python

import math

print("sin(π/4) = ",math.sin(math.pi/4))

##> sin(π/4) =  0.7071067811865475

Tính cos bằng số pi trong python

import math

print("cos(π/4) = ",math.cos(math.pi/4))

##> cos(π/4) =  0.7071067811865476

Tính tan bằng số pi trong python

import math

print("tan(π/4) = ",math.tan(math.pi/4))

#>> tan(π/4) =  0.9999999999999999

Tính chu vi hình tròn bằng số pi trong python

import math

r = 8

print("Chu vi = ", 2 \* r \* math.pi)

##> Chu vi =  50.26548245743669

Tính diện tích hình tròn bằng số pi trong python

import math

r = 8

print("Dien tich = ", r \* r \* math.pi)

##> Dien tich =  201.06192982974676

## Kiểm tra số chẵn lẻ

C1: chia phần trăm cho 2

def check\_odd\_even(n):

    #flag = 1 => số lẻ

    #flag = 0 => số chẵn

    flag = 1

    if( n % 2 == 0):

        flag= 0

    return flag

C2: AND với 1

def check\_odd\_even\_bit(n):

    #flag = 1 => số lẻ

    #flag = 0 => số chẵn

    flag = 0

    if( n & 1 == 1 ):

        flag= 1

    return flag

## Kiểm tra số nguyên, số thực

### Kiểm tra số nguyên số thực trong python | Hàm isinstance()

Cú pháp:

isinstance(num, type)

Trong đó num chính là số cần kiểm tra , và type là kiểu cần kiểm tra. type có thể là int hoặc float.

Hàm isinstance() sẽ trả về True nếu số đã cho thuộc kiểu type chỉ định , và ngược lại trả về False trong các trường hợp còn lại.

VD:

i = 6

f = 6.78

print(isinstance(i, int))

# True

print(isinstance(i, float))

# False

print(isinstance(f, int))

# False

print(isinstance(f, float))

# True

Lưu ý là trong trường hợp số nguyên được viết có phần thập phân, ví dụ như số 1 viết thành 1.0 hoặc 1.000 chẳng hạn thì hàm isinstance() sẽ coi đây là số thực trong Python.

print(isinstance(1, int))

# True

print(isinstance(1.00, int))

# False

print(isinstance(1.00, float))

# True

### Kiểm tra số thực có phải là số nguyên trong python | Phương thức is\_integer()

is\_integer() là một phương thức cài sẵn trong kiểu dữ liệu float, do đó chúng ta có thể sử dụng phương thức này với một số thực bất kỳ để kiểm tra coi nó có phải là số nguyên hay không.

Phương thức float.is\_integer() sẽ trả về True nếu số đã cho thuộc là số nguyên, và ngược lại trả về False trong các trường hợp còn lại.

VD:

f1 = 1.234

print(f1.is\_integer())

# False

f2 = 100.0

print(f2.is\_integer())

# True

## Kiểm tra số nguyên tố

Số nguyên tố là số lớn hơn 1 và chỉ có đúng hai ước số là 1 và chính nó.

Để làm được điều này, chúng ta chỉ cần kiểm tra trong phạm vi lớn hơn 1 và nhỏ hơn số đã cho có tồn tại ước số nào không, và nếu tồn tại dù chỉ một ước số thì có thể khẳng định số đã cho không phải là số nguyên tố.

VD

def check\_prime\_number(n):

    #flag = 0 => không phải số nguyên tố

    #flag = 1 => số nguyên tố

    flag = 1

    if (n <2):

        flag = 0

        return flag  #Số nhỏ hơn 2 không phải số nguyên tố => trả về 0

    #Sử dụng vòng lặp while để kiểm tra có tồn tại ước số nào khác không

    for p in range(2, n):

        if n % p == 0:

            flag = 0

            break #Chỉ cần tìm thấy 1 ước số là đủ và thoát vòng lặp

    return flag

## Kiểm tra số chính phương

Số chính phương hay còn gọi là số hình vuông là số tự nhiên có căn bậc hai là một số tự nhiên, hay nói cách khác, số chính phương bằng bình phương của một số tự nhiên

Cách 1: Kiểm tra các số trong khoảng [0, n).

def find\_square\_number(n):

    #flag = 1 => số chính phương

    #flag = 0 => không phải số chính phương

    flag = 0

    #Tìm số bất kỳ nhỏ hơn hoặc bằng n mà bình phương bằng n

    if any(i\*\*2 == n for i in range(n+1)):

        flag = 1

    return flag

Cách 2: Kiểm tra xem căn bậc 2 của n có phải là số nguyên hay không.

def find\_square\_number\_2(n):

    #flag = 1 => số chính phương

    #flag = 0 => không phải số chính phương

    flag = 0

    #Kiểm tra căn bậc 2 của số đó có phải số tự nhiên hay không

    if (n \*\* .5).is\_integer(): ##  if (math.sqrt(n)).is\_integer():

        flag = 1

    return flag

## Số hoàn hảo

Số hoàn hảo (hay còn gọi là số hoàn chỉnh, số hoàn thiện hoặc số hoàn thành) là một số nguyên dương mà tổng các ước nguyên dương chính thức của nó (số nguyên dương bị nó chia hết ngoại trừ nó) bằng chính nó.

def find\_perfect\_num(n):

    ## flag = 1 => số số hoàn hảo

    ## flag = 0 => không phải số hoàn hảo

    flag = 0

    #Tìm ước số

    divisor= [i for i in range(1,n) if n % i == 0]

    #Tìm tổng ước số

    total = 0

    for i in range(len(divisor)):

        total += divisor[i]

    #So sánh

    if total == n:

        flag = 1

    return flag

## Số đối xứng

số nguyên n gọi là số đối xứng nếu đọc từ trái qua phải, hay từ phải qua trái đều được số giống nhau. Ví dụ: 11,121,101 là các số đối xứng.

#Hàm kiểm tra số đối xứng trong python

def symmetrical\_num(n):

    # flag = 1 => số đối xứng

    # flag = 0 => không phải số đối xứng

    n = str(n)

    flag =0

    if ( n[::-1] == n):

      flag = 1

    return flag

# Biến

## Khái niệm

Có hai trường phái định nghĩa biến trong python như sau:

Trường phái đầu tiên coi biến trong python giống như một cái hộp để lưu trữ dữ liệu. Các dữ liệu này có thể là số hoặc chuỗi mà bạn có thể ưu trữ vào biến và sử dụng nhiều lần. Kết quả của các phép xử lý như tính toán giá trị số, chỉnh sửa chuỗi ký tự sẽ tạm thời được giữ vào biến và dùng để sử dụng cho chương trình sau này.

Trường phái thứ hai coi biến trong python giống như thẻ ghi địa chỉ của dữ liệu. Các dữ liệu được lưu giữ tại các vị trí riêng biệt trong bộ nhớ với địa chỉ khác nhau, và biến trong python là thẻ dùng để ghi địa chỉ của dữ liệu đó trong bộ nhớ. Khi sử dụng dữ liệu, chúng ta sẽ truy cập vào địa chỉ được ghi trên biến của dữ liệu đó.

## Khởi tạo

Cú pháp:

name = value

Trong đó

* name là tên biến được đặt theo quy tắc đặt tên biến trong Python mà bạn đã học trong bài Biến trong python là gì.
* Dấu = dùng để gán biến trong python. Giá trị value bên phải dấu = sẽ được gán vào biến ở bên trái.
* value là giá trị dùng để khai báo biến. Giá trị này có thể là chuỗi, số, hoặc các kiểu đối tượng khác trong python.

Khi bạn khởi tạo biến trong python, Python sẽ tự động xác định kiểu dữ liệu của biến được khai báo.

Kiểm tra kiểu dữ liệu sau khi khai báo biến bằng hàm typle()

myvar = 1234

print(type(myvar))

# <class 'int'>

mystr = "hello"

print(type(mystr))

# <class 'str'>

print(symmetrical\_num(n))

## Xoá biến

Cú pháp:

del tên biến

VD:

num = 10

print(num)

del num

# Lệnh điều kiện

## If

Cú pháp:

if biểu-thức-điều-kiện:

câu lệnh 1

câu lệnh 2

câu lệnh 3

if biểu-thức-điều-kiện:

câu lệnh xử lý nếu biểu-thức-điều-kiện là True (đúng)

else:

câu lệnh xử lý nếu biểu-thức-điều-kiện là False (sai)

if biểu-thức-điều-kiện-1:

câu lệnh xử lý nếu biểu-thức-điều-kiện-1 là True (đúng)

elif biểu-thức-điều-kiện-2:

câu lệnh xử lý nếu biểu-thức-điều-kiện-2 là True (đúng)

elif biểu-thức-điều-kiện-3:

câu lệnh xử lý nếu biểu-thức-điều-kiện-3 là True (đúng)

…

else:

câu lệnh xử lý nếu tất cả các biểu thức điều kiện đều False (sai)

## Toán tử so sánh

|  |  |  |
| --- | --- | --- |
| Toán tử | Biểu thức điều kiện | Ý nghĩa |
| == | x == y | x và y bằng nhau |
| != | x != y | x và y không bằng nhau |
| > | x > y | x lớn hơn y |
| < | x < y | x nhỏ hơn y |
| >= | x >= y | x bằng hoặc lớn hơn y |
| <= | x <= y | x bằng hoặc nhỏ hơn y |
| in | x in y | phần tử x tồn tại trong y |
| not in | x not in y | phần tử x không tồn tại trong y |

Tham khảo: <http://laptrinhcanban.com/python/nhap-mon-lap-trinh-python/lenh-dieu-kien-trong-python/toan-tu-so-sanh-trong-python/>

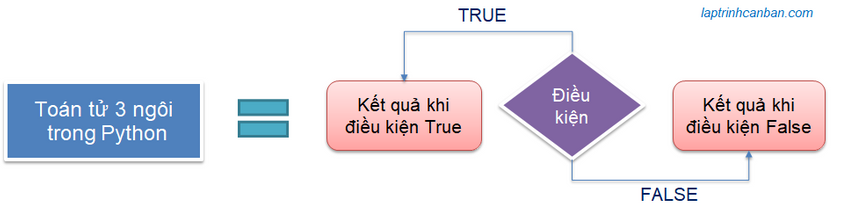
## Toán tử logic

|  |  |  |
| --- | --- | --- |
| Toán tử | Biểu thức | Ý nghĩa |
| AND | X and Y | True nếu cả X và Y đều đúng |
| OR | X or Y | True nếu ít nhất một trong hai vế X hoặc Y đúng |
| NOT | not X | True nếu X sai và False nếu X đúng |

Thứ tự ưu tiên

not > and > or

## Toán tử 3 ngôi



Cú pháp:

(true\_value) if (condition) else (false\_value);

VD:

x = "OK" if n == 10 else "NG"

# Vòng lặp

## For

Cú pháp:

for variable in iterable :

Câu lệnh trong khối

for variable in iterable :

Câu lệnh 1

Câu lệnh 2

…

VD:

member\_list = ['Hoa', 'Thảo', 'Mai']

for name in member\_list:

    print(name)

print("\nEnd")

#> Hoa

#> Thảo

#> Mai

#> End

## While

Cú pháp:

while biểu-thức-điều-kiện :

Câu lệnh 1 trong khối while

Câu lệnh 2 trong khối while

…

VD:

num = 1

while (num < 20):

    num = num \* 3

    print(num)

#>> 3

#>> 9

#>> 27

### While else break

VD:

num = 0

total = 0

while num < 4:

    if num == 2:

        print('!!BREAK!!')

        break

    print(num)

    total+=num

    num += 1

else

    print("Tổng =",total)

#> 0

#> 1

#> !!BREAK!!

Else được thực hiện khi điều kiện của while là sai. Tuy nhiên, nếu gặp break thì chương trình kết thúc vòng lặp và bỏ qua cả khối lệnh trong else.

## Break và continue

Break: thoát vòng lặp

Continue: bỏ qua các lệnh đằng sau và thực hiện lần lặp tiếp theo.

## Pass

Truyền qua và không thực hiện gì.

Pass dùng để đảm bảo cấu trúc bên trong hàm hoặc vòng lặp là hợp lệ.

## Range()

Cú pháp:

range(start=0, stop, step=1)

Trong đó:

* start [Optional (không bắt buộc)]. An integer number specifying at which position to start. Default is 0
* stop [Required (bắt buộc)]. An integer number specifying at which position to stop (not included).
* step [Optional]. An integer number specifying the incrementation. Default is 1

range trả về dải đối tượng trong khoảng [start=0; stop) với bước nhảy mặc định bằng 1.

VD:

# print first 5 integers

# using python range() function

for i in range(5):

    print(i, end=" ")

print()

0 1 2 3 4

## True, False trong python

VD:

if True:

    print(True)

if not False:

    print(False)

True

False

Trong python có định nghĩa True False hỗ trợ trong các câu lệnh điều khiện.

# Chuỗi

## Khai báo:

### Khai báo sử dụng dấu nháy kép “” hoặc ‘’

"Chuỗi ký tự"

Hoặc

'Chuỗi ký tự'

### Khai báo chuỗi nhiều dòng bằng 3 nháy đơn hoặc 3 nháy kép

"""abc

def

xyz"""

Hoặc

'''abc

def

xyz'''

### Khai báo chuỗi nhiều dòng chứa dấu nháy đơn ' hoặc dấu nháy kép "

Phương pháp: Khai báo trong dấu nháy khác.

VD: Khi khai báo chuỗi bằng 3 dấu nháy kép thì các dấu nháy đơn, nháy kép, 3 dấu nháy đơn đều được coi là ký tự bình thường.

VD:

print(""" She's already study with "you". '''good job''' """)

She's already study with "you". '''good job'''

### Ứng dụng của chuỗi nhiều dòng trong python

Ngoài cách sử dụng chuỗi nhiều dòng như là một chuỗi thông thường, bạn cũng có thể sử dụng chuỗi nhiều dòng được khai báo bởi ba dấu nháy để viết chú thích nhiều dòng trong python.

Bản chất của cách viết này là chúng ta khai báo chuỗi nhiều dòng trong python bằng ba dấu nháy và không thêm lệnh xử lý gì với chuỗi mới khai báo.

Do đó chuỗi nhiều dòng vừa được khai báo sẽ chỉ có tác dụng lưu trữ thông tin mà không ảnh hưởng gì đến chương trình.

## Độ dài chuỗi

### Tính độ dài chuỗi bằng len:

Cú pháp:

len(str)

Lưu ý kết quả của hàm len trong python là số ký tự có trong chuỗi chứ không phải là số bite có trong chuỗi, do đó kể cả các ký tự được tạo bởi 2 bite trở lên như tiếng Việt có dấu hay tiếng Nhật zenkaku, thì hàm len cũng đưa ra kết quả chính xác độ dài chuỗi trong python .

VD:

# Tính độ dài chuỗi chứa ký tự 1 bite trong python

print(len('Hello'))

#>> 5

# Tính độ dài chuỗi chứa ký tự 2 bite trong python

print(len('Chào'))

#>> 4

# Tính độ dài chuỗi chứa ký tự 2 bite trong python

print(len('ｔｏｕｋｙo'))

#>> 6

## Nối chuỗi

Cú pháp:

"Chuỗi ký tự 1" + "Chuỗi ký tự 2"

### Nối chuỗi bằng phương thức join()

Cú pháp:

str.join(iterable)

Trong đó str là ký tự phân cách giữa các phần tử khi nối chúng lại, join dùng để gọi tên phương thức, iterable là list hoặc tuple.

VD:

# Tổng quan

# Nối chuỗi là phần tử của tuple

str = ", "

iterable = ("Java", "Python", "PHP")

print (str.join(iterable))

#>> Java, Python, PHP

# Nối chuỗi là phần tử của list

print(",".join(["Blue", "Red", "Green"]))

#>> Blue,Red,Green

## Lặp chuỗi

Cú pháp:

"Chuỗi ký tự " \* số lần lặp

## Cắt chuỗi (Slice string)

Cú pháp:

str[ begin : end : step=1]

Trong đó:

* str là chuỗi cần cắt
* begin là index của ký tự bắt đầu cắt chuỗi
* end là index của ký tự kết thúc cắt chuỗi. Chuỗi cắt không bao gồm ký tự này.
* Step là bước nhảy.

Khoảng cắt: [begin; end). Với bước nhảy mặc định bằng 1.

Lưu ý:

* Nếu lược bỏ begin: mặc định cắt từ đầu chuỗi
* Nếu lược bỏ end: mặc định cắt đến cuối chuỗi
* Index có thể là số âm. VD: -1 tương ứng với ký tự cuối cùng.

VD:

str = "bigcityboy"

print(str[3:7])

#>> city

# Lược bỏ begin

print(str[:7])

#>> bigcity

# lược bỏ end

print(str[3:])

#>> cityboy

# index âm

print(str[:-1])

#>> bigcitybo

# index âm

print(str[:-2])

#>> bigcityb

# sử dụng step

print(str[::2])

#>> bgiyo

## Tách chuỗi

### Tách chuỗi bằng split()

Cú pháp:

str.split(sep, maxsplit)

Trong đó :

* sep (viết tắt của separator) là ký tự phân cách dùng để tách chuỗi str ban đầu ra các chuỗi nhỏ. Nếu không chỉ định thì python mặc định sep là ký tự trống.
* maxsplit là số lần tách lớn nhất. Nếu không chỉ định thì python mặc định số lần tách là vô hạn.

Lưu ý:

* Các ký tự trống có thể là khoảng trắng, tab \t, …
* Split nhặt ra các phần tử không phải là khoảng trống và xếp vào một danh sách.

VD:

print("  My First \tLove".split())

#>> ['My', 'First', 'Love']

print("Orange,Lemon,Apple".split(","))

#>> ['Orange', 'Lemon', 'Apple']

print("A B C D E".split(" ", 2))

#>> ['A', 'B', 'C D E']

### Tách chuỗi bằng splitlines()

Cú pháp:

str.splitlines([keepends=False])

Trong đó: keepends: giữ ký tự xuống dòng.

Splitlines() dùng tách chuỗi bằng một ký tự xuống dòng và thu về kết quả là một list có các phần tử là các chuỗi nhỏ vừa được tách ra.

Các ký tự trong chuỗi được coi là ký tự xuống dòng như bảng sau đây:

* \n xuống dòng
* \r quay về đầu dòng
* \r\n xuống dòng + quay về đầu dòng
* \v or \x0b Tab thẳng
* \f or \x0c ngắt trang
* \x1c ngắt tệp
* \x1d Ngắt nhóm
* \x1e Ngắt bản ghi
* \x85 xuống dòng (mã điều khiển C1)
* \u2028 ngắt dòng
* \u2029 Phân cách đoạn văn

VD:

str2 = """\

Hello

My name is Kiyoshi

Thank you"""

print(str2)

#>> Hello

#>> My name is Kiyoshi

#>> Thank you

print(str2.splitlines())

#>> ['Hello', 'My name is Kiyoshi', 'Thank you']

VD: Giữ lại ký tự xuống dòng

str1 = "\nOrange\nLemon\nApple\n"

print(str1)

#>> Orange

#>> Lemon

#>> Apple

print(str1.splitlines(True))

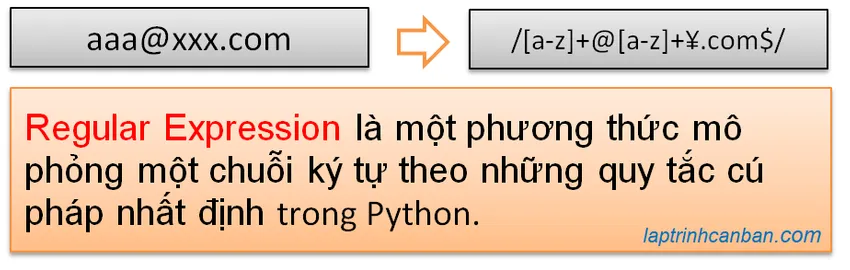
#>> ['\n', 'Orange\n', 'Lemon\n', 'Apple\n']

## Regular Expression

### Khái niệm

Regular Expression (viết tắt RegEx) hay còn gọi là biểu thức chính quy là một phương thức mô phỏng một chuỗi ký tự theo những quy tắc cú pháp nhất định.

Nguyên tắc hoạt động của biểu thức RegEx là so khớp dựa vào khuôn mẫu, khuôn mẫu được xây dựng từ các quy tắc căn bản của biểu thức RegEx.



### Regular expression module

Python có một gói bult-in (tích hợp) được gọi là re. Nó được sử dụng để làm việc với Regular expressions (các biểu thức chính quy)

Import re module:

import re

### Hàm RegEx

|  |  |
| --- | --- |
| Function | Description |
| findall | Returns a list containing all matches |
| search | Returns a Match object if there is a match anywhere in the string |
| split | Returns a list where the string has been split at each match |
| sub | Replaces one or many matches with a string |

### Siêu ký tự (Metacharacters)

|  |  |  |
| --- | --- | --- |
| Character | Description | Example |
| [] | A set of characters | "[a-m]" |
| \ | Signals a special sequence (can also be used to escape special characters) | "\d" |
| . | Any character (except newline character) | "he..o" |
| ^ | Starts with | "^hello" |
| $ | Ends with | "planet$" |
| \* | Zero or more occurrences | "he.\*o" |
| + | One or more occurrences | "he.+o" |
| ? | Zero or one occurrences | "he.?o" |
| {} | Exactly the specified number of occurrences | "he.{2}o" |
| | | Either or | "falls|stays" |
| () | Capture and group |  |

VD:

import re

txt = "the abc123 in the bcd456 better than the abc123"

x = re.findall("a[a-z]\*", txt)

print(x)

#>>['abc', 'an', 'abc']

x = re.search("a[a-z]\*", txt)

print(x)

#>><re.Match object; span=(4, 7), match='abc'>

print(x.start())

#>>4

x = re.findall("\d+", txt)

print(x)

#>>['123', '456', '123']

x = re.findall("^the.\*abc", txt)

print(x)

#>>['the abc123 in the bcd456 better than the abc']

Lưu ý:

* Để sử dụng []: tham khảo bảng set.
* Để sử dụng \: tham khảo bảng Special Sequences

### Chuỗi đặc biệt(Special Sequences)

|  |  |  |
| --- | --- | --- |
| Character | Description | Example |
| \A | Returns a match if the specified characters are at the beginning of the string | "\AThe" |
| \b | Returns a match where the specified characters are at the beginning or at the end of a word  (the "r" in the beginning is making sure that the string is being treated as a "raw string") | r"\bain" r"ain\b" |
| \B | Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word  (the "r" in the beginning is making sure that the string is being treated as a "raw string") | r"\Bain" r"ain\B" |
| \d | Returns a match where the string contains digits (numbers from 0-9) | "\d" |
| \D | Returns a match where the string DOES NOT contain digits | "\D" |
| \s | Returns a match where the string contains a white space character | "\s" |
| \S | Returns a match where the string DOES NOT contain a white space character | "\S" |
| \w | Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore \_ character) | "\w" |
| \W | Returns a match where the string DOES NOT contain any word characters | "\W" |
| \Z | Returns a match if the specified characters are at the end of the string | "Spain\Z" |

### Bộ (sets)

Một bộ là một tập hợp các ký tự bên trong một cặp dấu ngoặc vuông []

|  |  |
| --- | --- |
| Set | Description |
| [arn] | Returns a match where one of the specified characters (a, r, or n) is present |
| [a-n] | Returns a match for any lower case character, alphabetically between a and n |
| [^arn] | Returns a match for any character EXCEPT a, r, and n |
| [0123] | Returns a match where any of the specified digits (0, 1, 2, or 3) are present |
| [0-9] | Returns a match for any digit between 0 and 9 |
| [0-5][0-9] | Returns a match for any two-digit numbers from 00 and 59 |
| [a-zA-Z] | Returns a match for any character alphabetically between a and z, lower case OR upper case |
| [+] | In sets, +, \*, ., |, (), $,{} has no special meaning, so [+] means: return a match for any + character in the string |

### Hàm findall()

Hàm findall()trả về một danh sách chứa tất cả các kết quả khớp. Nếu không tìm thấy chuỗi nào khớp, trả về danh sách trống [].

VD:

import re

txt = "The rain in Spain"

x = re.findall("ai", txt)

print(x)

['ai', 'ai']

### Hàm search()

Hàm search() tìm kiếm và trả về một chuỗi khớp tìm thấy đầu tiên nếu có kết quả khớp. Nếu không tìm thấy chuỗi nào khớp, trả về None.

import re

txt = "The rain in Spain"

x = re.search("\s", txt)

print(x)

print("The first white-space character is located in position:", x.start())

<re.Match object; span=(3, 4), match=' '>

The first white-space character is located in position: 3

### Hàm split()

Hàm split() trả về một danh sách trong đó chuỗi đã được chia ở mỗi lần khớp.

VD

import re

txt = "The rain in Spain"

x = re.split("\s", txt)

print(x)

['The', 'rain', 'in', 'Spain']

Bạn có thể kiểm soát số lần xuất hiện bằng cách chỉ định maxsplit tham số:

import re

txt = "The rain in Spain"

x = re.split("\s", txt, 1)

print(x)

['The', ' rain in Spain']

Trong chuỗi txt bên trên, khoảng trắng đầu tiên được tạo bởi hai ký tự space. Tuy nhiên do đặt maxsplit bằng 1 nên chỉ một kỹ tự space đầu tiên được chọn là ký tự phân tách, còn ký tự space còn lại được giữ nguyên trong phần tử thứ 2 được trả về.

### Hàm sub()

Đặt lại các chuỗi khớp bằng chuỗi chỉ định.

VD:

import re

txt = "The rain in Spain"

x = re.sub("\s", "9", txt)

print(x)

The9rain9in9Spain

### Phân biệt các ký tự/chuỗi bắt đầu/kết thúc với ^, $, /A, /b.

<https://www.w3schools.com/python/python_regex.asp>

1. phân biệt /b với ^ và /A.

/b là bắt đầu hoặc kết thúc của một từ bất kỳ trong chuỗi tìm kiếm, còn ^ và /A là bắt đầu của cả chuỗi dùng để tìm kiếm. Do đó, /b có thể tìm kiếm ở bất kỳ vị trí nào trong chuỗi và có thể cho ra nhiều kết quả với re.findall(). Còn ^ và /A chỉ xét mỗi vị trí đầu của chuỗi và re.findall() chỉ cho ra danh sách với tối đa là một kết quả.

VD1:

import re

s = """the abc is abc is abc bcd abcbcd

abc is bcd is abc"""

l = re.findall(r"\babc.\*abc\b", s)

print(l)

#>> ['abc is abc is abc', 'abc is bcd is abc']

l = re.findall(r"\Aabc.\*abc\b", s)

print(l)

#>> []

l = re.findall(r"^abc.\*abc\b", s)

print(l)

#>> []

Lưu ý:

* /b là tìm kiếm trong từ. một từ được định nghĩa là nằm giữa 2 khoảng trắng. do đó, khi tìm kiếm abc\b (abc ở cuối từ), abcbcd tuy cũng có abc nhưng không nằm cuối từ mà nằm cuối từ là bcd, do dó, abcbcd không khớp với mẫu.
* Trong lần tìm kiếm đầu, kết quả cho ra là abc is abc is abc thay vì abc is abc.

(Dự đoán lý do, không chắc chắn) Nguyên do là theo biểu thức chính quy, thay vì tìm từ có abc đứng đầu, sau đó tìm lần lượt các ký tự cho đến khi gặp abc thì trình tự thực hiện lại là tìm từ abc đứng đầu, sau đó lấy tất cả các ký tự trừ newline (dấu chấm là bất kỳ ký tự nào trừ newline) được abc is abc is abc bcd abcbcd, cuối cùng loại bỏ phần đuôi đến khi gặp từ có kết thúc là abc được abc is abc is abc.

2. ^ và /A

^ và /A không có sự khác biệt.

^ và /A đều không bỏ qua khoảng trắng.

VD2

import re

s = """the abc is bcd"""

l = re.findall(r"\Ath.\*", s)

print(l)

#>> ['the abc is bcd']

l = re.findall(r"^th.\*", s)

print(l)

#>> ['the abc is bcd']

# lỗi khoảng trắng

l = re.findall(r"^ th.\*", s)

print(l)

#>> []

### Tham khảo:

<https://topdev.vn/blog/regex-la-gi/>

<https://www.w3schools.com/python/python_regex.asp>

## Tách số trong chuỗi

<http://laptrinhcanban.com/python/nhap-mon-lap-trinh-python/thao-tac-voi-chuoi-string-trong-python/tach-so-trong-chuoi-python/>

### Tách dãy số trong chuỗi python dưới dạng một list | hàm re.findall()

Cú pháp:

re.findall ( r'\d+' , str)

VD:

import re

s = 'Thứ 6 ngày 25 tháng 6 năm 2021, 21:30'

m = re.findall(r'\d+', s)

print(m)

#>> ['6', '25', '6', '2021', '21', '30']

print("Ngày",m[1])

print("Tháng",m[2])

print("Năm",m[3])

#>> Ngày 25

#>> Tháng 6

#>> Năm 2021

### Tách số trong chuỗi python dưới dạng một chuỗi | hàm re.sub()

Cú pháp:

re.sub ( r'\D' , '', str)

import re

s1 = '1ab23cdef456'

m1 = re.sub(r'\D','', s1)

print(m1)

#>> 123456

s2 = 'Thứ 6 ngày 25 tháng 6 năm 2021, 21:30'

m2 = re.sub(r'\D','', s2)

print(m2)

#>> 625620212130

Lưu ý: re.sub() không làm thay đổi chuỗi ban đầu.

### Tách số đầu tiên trong chuỗi python | hàm re.search()

Cú pháp:

re.search ( r'\d' , str)

VD:

import re

s1 = '1ab23cdef456'

m1 = re.search(r'\d', s1)

print(m1)

#>> <re.Match object; span=(0, 1), match='1'>

print(m1.group())

#>> 1

s2 = 'Thứ 6 ngày 25 tháng 6 năm 2021, 21:30'

m2 = re.search(r'\d', s2)

print(m2.group())

#>> 6

### Tách dãy số đầu tiên xuất hiện trong chuỗi python | hàm re.search() python

Cú pháp:

re.search ( r'\d+' , str)

VD:

import re

s = 'Năm 2021, thứ 6 ngày 13 21:30'

m = re.search(r'\d+', s)

print(m)

#>> <re.Match object; span=(4, 8), match='2021'>

print(m.group())

#>> 2021

Sau đó, chúng ta có thể lấy vị trí bắt đầu và kết thúc của dãy số trong chuỗi bằng cách dùng phương thức start() và end() như sau:

#Lấy vị trí xuất hiện của dãy số

print(m.start())

#>> 4

#Lấy vị trí kết thúc của dãy số

print(m.end())

#>> 8

## Xoá ký tự trong chuỗi

<http://laptrinhcanban.com/python/nhap-mon-lap-trinh-python/thao-tac-voi-chuoi-string-trong-python/xoa-ky-tu-trong-chuoi-python/>

### strip() xoá các ký tự đầu và cuối trong chuỗi

Cú pháp:

new\_str = org\_str.strip(chars = space/newline )

Trong đó:

* org\_str là chuỗi ban đầu.
* chars là tập hợp các ký tự muốn xóa ở cả hai đầu chuỗi org\_str. Mặc định xoá khoảng trắng và xuống dòng

Phương thức strip() xét lần lượt các ký tự trong org\_str từ 2 đầu và xóa đi cho đến khi gặp phải khí tự không nằm trong chars. Kết quả lưu vào biến mới (new\_str). Và strip() không làm thay đổi chuỗi gốc.

VD3:

# Xoá khoảng trắng 2 đầu chuỗi

org\_str = "  hello world  "

new\_str = org\_str.strip()

print('[' + new\_str + ']')

#>> [hello world]

# xoá các ký tự khoảng trắng, l và o

org\_str = "loool hello world loool"

new\_str = org\_str.strip(" lo")

print('[' + new\_str + ']')

#>> [hello world]

### Rstrip() và lstrip()

Tương tự strip, lstrip và rstrip lần lượt dùng để xoá các ký tự ở đầu chuỗi và cuối chuỗi.

VD:

#lstrip ----------------------------------------------------------------

# Xoá khoảng trắng ở đầu chuỗi

org\_str = "  hello world  "

new\_str = org\_str.lstrip()

print('[' + new\_str + ']')

#>> [hello world  ]

# Xoá các ký tự khoảng trắng, l và o ở đầu chuỗi

org\_str = "loool hello world loool"

new\_str = org\_str.lstrip(" lo")

print('[' + new\_str + ']')

#>> [hello world loool]

#rstrip ----------------------------------------------------------------

# Xoá khoảng trắng ở cuối chuỗi

org\_str = "  hello world  "

new\_str = org\_str.rstrip()

print('[' + new\_str + ']')

#>> [  hello world]

# Xoá các ký tự khoảng trắng, l và o ở cuối chuỗi

org\_str = "loool hello world loool"

new\_str = org\_str.rstrip(" lo")

print('[' + new\_str + ']')

#>> [loool hello world]

### Sử dụng replace() để xoá các ký tự trong cả chuỗi

Cú pháp:

new\_str = org\_str.replace(old\_substr, new\_substr)

Trong đó:

* old\_substr là chuỗi con cũ cần thay đổi
* new\_substr là chuỗi con cũ được thay đổi.

VD:

# Xoá các ký tự khoảng trắng, o và l trong chuỗi

org\_str = "  hello world  "

new\_str = org\_str.replace(" ", "")

new\_str = new\_str.replace("o", "")

new\_str = new\_str.replace("l", "")

print('[' + new\_str + ']')

#>> [hewrd]

### Sử dụng slice để xoá chuỗi

VD:

# Xoá ký tự đầu chuỗi

org\_str = "hello world"

new\_str = org\_str[1:]

print('[' + new\_str + ']')

#>> [ello world]

# Xoá ký tự cuối chuỗi

org\_str = "hello world"

new\_str = org\_str[:-1]

print('[' + new\_str + ']')

#>> [hello worl]

## Tìm chuỗi con trong chuỗi

### Find()

Tìm vị trí chuỗi con xuất hiện đầu tiên trong chuỗi và trả về -1 nếu không tìm được.

Cú pháp:

str.find(sub, start=0, end=[-1]+1)

Trong đó:

* sub là chuỗi con cần tìm vị trí.

Find() tìm vị trí xuất hiện lần đầu tiên của chuỗi con trong đoạn [start; end). Mặc định tìm từ vị trí đầu tiên đến hết chuỗi.

VD:

org\_str = "Good school"

# Tìm vị trí của oo trong chuỗi

print(org\_str.find("oo"))

#>> 1

print(org\_str.find("oo", 2))

#>> 8

# oo xuất hiện lần thứ 2 tại vị trí 8.

# Tuy nhiên khu vực tìm kiểm [2; 8) nên kết quả không tìm thấy và trả về -1

print(org\_str.find("oo", 2, 8))

#>> -1

### Rfind()

Tương tự find() nhưng tìm theo chiều ngược lại (từ cuối chuỗi lên đầu).

Cú pháp:

str.rfind(sub, start=0, end=[-1]+1)

VD:

org\_str = "Good school"

# Tìm vị trí của oo trong chuỗi

print(org\_str.rfind("oo"))

#>> 8

print(org\_str.rfind("oo", 0, 8))

#>> 1

# oo xuất hiện tại 2 vị trí 1 và 8.

# Tuy nhiên khu vực tìm kiểm [2; 8) nên kết quả không tìm thấy và trả về -1

print(org\_str.rfind("oo", 2, 8))

#>> -1

### Index()

Tương tự find, index() cũng tìm vị trí xuất hiện lần đầu tiên của chuỗi con nhưng sẽ trả ra lỗi nếu không tìm thấy vị trí.

Cú pháp:

str.index(sub, start=0, end=[-1]+1)

VD:

org\_str = "Good school"

# Tìm vị trí của oo trong chuỗi

print(org\_str.index("oo"))

#>> 1

print(org\_str.index("oo", 2))

#>> 8

# oo xuất hiện tại vị trí 1 và 8.

# Tuy nhiên khu vực tìm kiểm [2; 8) nên kết quả không tìm thấy và trả về lỗi

print(org\_str.index("oo", 2, 8))

"""

>> Traceback (most recent call last):

  File "d:\PROGRAMMING\_LANGUAGE\Python\_Tutorial\Code\hello\_world\highline.py", line 12, in <module>

    print(org\_str.index("oo", 2, 8))

ValueError: substring not found

"""

### Rindex()

Rindex() tương tự index() nhưng tìm kiếm theo chiều ngược lại (từ cuối chuỗi lên đầu chuỗi)

Cú pháp:

str.rindex(sub, start=0, end=[-1]+1)

VD:

org\_str = "Good school"

# Tìm vị trí của oo trong chuỗi

print(org\_str.rindex("oo"))

#>> 8

print(org\_str.rindex("oo", 0, 8))

#>> 1

# oo xuất hiện tại vị trí 1 và 8.

# Tuy nhiên khu vực tìm kiểm [2; 8) nên kết quả không tìm thấy và trả về lỗi

print(org\_str.rindex("oo", 2, 8))

"""

>> Traceback (most recent call last):

  File "d:\PROGRAMMING\_LANGUAGE\Python\_Tutorial\Code\hello\_world\highline.py", line 12, in <module>

    print(org\_str.index("oo", 2, 8))

ValueError: substring not found

"""

## Index dương, index âm

### index dương

Ký tự đầu tiên của chuỗi có index là 0, sau đó tăng dần 1, 2, 3... theo thứ tự về phía sau.

Ví dụ index dương của các ký tự trong chuỗi bigcityboy như sau:

b i g c i t y b o y

-------------------

0 1 2 3 4 5 6 7 8 9

### index âm

Ký tự cuối cùng( ngoài cùng bên tay phải) của chuỗi có index là -1, sau đó giảm dần -1, -2,-3... về phía đầu( phía tay trái).

Ví dụ index âm của các ký tự trong chuỗi bigcityboy như sau:

b i g c i t y b o y

--------------------------------

-10 -9 -8 -7 -6 -5 -4 -3 -2 -1

## Thay thế ký tự trong chuỗi

### Replace()

Cú pháp:

new\_str = org\_str.replace(old\_substr, new\_substr, count=[max])

Trong đó:

* old\_substr là chuỗi con cũ cần thay đổi
* new\_substr là chuỗi con cũ được thay đổi.
* count là số lần thay thế. Mặc định count bằng max (thay thế tất cả)

VD:

org\_str = "Yooh! Good school!"

# Thay đổi toàn bộ oo thành OO

new\_str = org\_str.replace("oo", "OO")

print(new\_str)

#>> YOOh! GOOd schOOl!

# Thay đổi 2 lần oo thành OO

new\_str = org\_str.replace("oo", "OO", 2)

print(new\_str)

#>> YOOh! GOOd school!

## Đếm số lần xuất hiện chuỗi con trong chuỗi.

### Count()

Cú pháp:

str.count(sub, start=0, end=[-1]+1)

Trong đó:

* sub là chuỗi con cần đếm

VD:

org = "Yooh! Good school!"

# đếm số lần xuất hiện oo

print(org.count("oo"))

# đếm số lần xuất hiện oo trong khoảng [0, 14)

print(org.count("oo", 0, 14))

## Kiểm tra chuỗi có phải là chữ hay không trong python

### Isascii()

Kiểm tra chuỗi có là ký tự thuộc ascii không

Cú pháp:

str.isascii()

Ký tự ASCII chứa cả dấu chấm ., dấu phẩy , và dấu gạch ngang -.

VD:

print("Hello".isascii())

#>> True

print("8052".isascii())

#>> True

print("東京".isascii())

#>> False

print("Việt Nam".isascii())

#>> False

### Isalpha()

Kiểm tra chuỗi có là chữ nằm trong alphabet.

Tiếng Việt có dấu trả về False, hán tự tiếng Nhật hay tiếng Trung lại trả về True.

Cú pháp:

str.isalpha()

Nếu như tất cả ký tự trong chuỗi là ký tự chữ, thì kết quả True sẽ được trả về.

VD:

print("Hello".isalpha())

#>> True

print("8052".isalpha())

#>> False

Theo như website chính thức của python thì ký tự Alphabetic là những ký tự được định nghĩa là “Letter” trong bảng mã Unicode, có nghĩa là những ký tự mà có thuộc tính category là một trong “Lm”, “Lt”, “Lu”, “Ll”, hoặc “Lo”. Và thuộc tính của những ký tự này khác với quy chuẩn của Unicode Standard.

Dựa theo định nghĩa này thì tiếng Việt có dấu, hoặc số viết bởi tiếng Nhật zenkaku sẽ không được tính là Alphabetic, do đó khi dùng phương thức isalpha trong python thì kết quả False sẽ được trả về.

Điều kỳ lạ là hán tự tiếng Nhật hay tiếng Trung lại được chấp nhận trong thuộc tính này.

print("Việt Nam".isalpha())

#>> False

print("京都".isalpha())

#>> True

print("Ａｐｐｌｅ".isalpha())

#>> True

print("７６５".isalpha())

#>> False

print("七五三".isalpha())

#>> True

print("".isalpha())

#>> False

### Isalnum()

Kiểm tra chuỗi có phải là chữ hoặc số

Cú pháp:

str.isalnum()

Nếu như tất cả ký tự trong chuỗi là ký tự alphabet hoặc số thì kết quả True sẽ được trả về.

VD:

print("Hello".isalnum())

#>> True

print("100years".isalnum())

#>> True

print("8052".isalnum())

#>> True

print("Việt Nam".isalnum())

#>> False

print("京都".isalnum())

#>> True

print("Ａｐｐｌｅ".isalnum())

#>> True

print("７６５".isalnum())

#>> True

print("七五三".isalnum())

#>> True

print("".isalnum())

#>> False

## Kiểm tra chuỗi có phải là số

Tham khảo: <http://laptrinhcanban.com/python/nhap-mon-lap-trinh-python/thao-tac-voi-chuoi-string-trong-python/kiem-tra-chuoi-co-phai-la-so-hay-khong-trong-python/>

### Isdecimal()

Kiểm tra xem chuỗi có phải số thập phân

Cú pháp:

str.isdecimal()

VD:

print("75120".isdecimal())

#>> True

print("75F2".isdecimal())

#>> False

# ký tự 2 byte như tiếng Nhật zenkaku

print("０９８２".isdecimal())

#>> True

print("７５ａ０".isdecimal())

#>> False

# dấu âm

print("-8052".isdecimal())

#>> False

print("3.14".isdecimal())

#>> False

print("7,524".isdecimal())

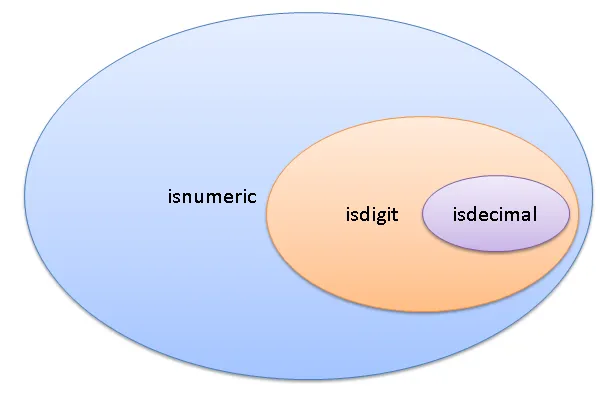
#>> False

print("".isdecimal())

#>> False

### So sánh isdecimal, isdigit, isnumeric

|  |  |  |  |
| --- | --- | --- | --- |
| phương thức | isdecimal | isdigit | isnumeric |
| dạng dữ liệu | số thập phân | số thập phân hoặc Digit | số thập phân hoặc Digit hoặc Numeric |
| ví dụ | “12345” “12” “98201” | “12345” “1233” “3” ③ | “12345” “½¼” “½” “12345½” “3” ③ “七五三” “拾伍” |
| kết quả | “true” nếu tất cả ký tự là số thập phân | “true” nếu tất cả ký tự là số thập phân hoặc Digit | “true” nếu tất cả ký tự là số thập phân, Digit hoặc Numeric |
| xem thêm | [isdecimal](http://laptrinhcanban.com/python/nhap-mon-lap-trinh-python/thao-tac-voi-chuoi-string-trong-python/kiem-tra-chuoi-co-phai-la-so-hay-khong-trong-python/#isdecimal) | [isdigit](http://laptrinhcanban.com/python/nhap-mon-lap-trinh-python/thao-tac-voi-chuoi-string-trong-python/kiem-tra-chuoi-co-phai-la-so-hay-khong-trong-python/#isdigit) | [isnumeric](http://laptrinhcanban.com/python/nhap-mon-lap-trinh-python/thao-tac-voi-chuoi-string-trong-python/kiem-tra-chuoi-co-phai-la-so-hay-khong-trong-python/#isnumeric) |



## Chuyển số thành chuỗi:

Cú pháp:

Str(num)

VD:

str1 = str(234.5)

print(type(str1), str1)

#>> <class 'str'> 234.5

## Chuyển list thành chuỗi:

### Sử dụng join()

VD:

mylist = ["a", "b", "c", 1]

mystring = "".join([str(char) for char in mylist])

print(mystring)

#>> abc1

### Sử dụng RegEx

import re

mylist = [1, ['a', 'b'], 3, 4, (5,6)]

mystring = "".join([str(char) for char in mylist])

print(mystring)

#>> 1['a', 'b']34(5, 6)

new\_string = re.sub(r"[^a-zA-Z0-9]","",mystring)

print(new\_string)

#>> 1ab3456

### Sử dụng hàm map()

Chúng ta cũng có thể sử dụng hàm map() để chuyển list sang trong Python. Hàm map() trong Python sẽ chuyển các phần tử trong list chỉ định sang dạng chuỗi và tạo ra một trình lặp. Sau đó chúng ta cũng sử dụng phương thức join() để lấy ra và nối các chuỗi là phần tử của trình lặp này để tạo ra chuỗi string kết quả.

VD:

mylist1 = ["a", "b", "c", 1]

mystring1 = "".join(map(str, mylist1))

print(mystring1)

#>> abc1

mylist2 = [1, 2, 3, 4]

mystring2 = "".join(map(str, mylist2))

print(mystring2)

#>> 1234

## Kiểm tra chữ hoa, chữ thường

### Kiểm tra chữ hoa

Cú pháp:

str.isupper()

nếu như tất cả ký tự trong chuỗi đều là chữ hoa, thì kết quả True sẽ được trả về. Và False trong các trường hợp còn lại.

print("APPLE".isupper())

#>> True

print("Hello".isupper())

#>> False

Đối với những ký tự không phân biệt được chữ hoa chữ thường như số hoặc kanji tiếng Nhật thì sẽ được bỏ qua khi kiểm tra bằng phương thức islower trong python, như ví dụ sau:

print("ORANGE100%".isupper())

#>> True

print("MAX値".isupper())

#>> True

### Kiểm tra chữ thường

Str.islower().

Sử dụng giống str.isupper().

### Kiểm tra tiêu đề

Str.istitle()

Sử dụng giống str.isupper().

Title() là kiểu chữ đầu viết hoa, các chữ còn lại viết thường.

print("Music Video".istitle())

#>> True

print("THE Movie".istitle())

#>> False

Chi tiết các trường hợp đặc biệt, tham khảo:

<http://laptrinhcanban.com/python/nhap-mon-lap-trinh-python/thao-tac-voi-chuoi-string-trong-python/kiem-tra-chu-hoa-chu-thuong-trong-python/>

## Chuyển chữ hoa thành chữ thường và ngược lại trong python

### Chuyển chữ hoa thành chữ thường trong python

Cú pháp:

str.lower()

VD:

print("Hello".lower())

#>> hello

print("APPLE".lower())

#>> apple

### Chuyển chữ thường thành chữ hoa trong python

Cú pháp:

str.upper()

VD:

print("Hello".upper())

#>> HELLO

print("apple".upper())

#>> APPLE

### Phương thức capitalize trong python

Chúng ta sử dụng phương thức capitalize trong python để viết hoa ký tự đầu tiên của chuỗi và chuyển tất cả các chữ hoa khác thành chữ thường và tạo thành chuỗi mới với cú pháp sau đây:

str.capitalize()

VD:

print("hello python".capitalize())

#>> Hello python

print("MY FAVORITE THINGS".capitalize())

>> My favorite things

### Phương thức title trong python

Chúng ta sử dụng phương thức title trong python để viết hoa ký tự đầu tiên của cụm từ trong chuỗi và chuyển tất cả chữ hoa khác thành chữ thường và tạo thành chuỗi mới với cú pháp sau đây:

str.title()

VD:

print("hello python".title())

#>> Hello Python

print("MY FAVORITE THINGS".title())

#>> My Favorite Things

### Phương thức swapcase trong python

Chúng ta sử dụng phương thức swapcase trong python để chuyển tất cả chữ hoa thành chữ thường và tất cả chữ thường thành chữ hoa và tạo thành chuỗi mới với cú pháp sau đây:

str.swapcase()

VD:

print("Hello".swapcase())

#>> hELLO

print("apple".swapcase())

#>> APPLE

## Căn lề

### Căn lề trái một chuỗi trong Python | phương thức ljust()

Cú pháp:

org\_str.ljust(width,fillchar)

Trong đó:

* width là chiều dài chỉ định của chuỗi kết quả sau khi căn lề. Giá trị của này có thể lớn hơn hoặc nhỏ hơn độ dài ban đầu của chuỗi str.
* fillchar là một ký tự dùng để lấp chỗ trống.

VD:

str1="học lập trình"

print("[" + str1.ljust(20) + "]")

#>> [học lập trình       ]

str2 ="10082019"

print("[" + str2.ljust(10) + "]")

#>> [10082019  ]

print("Apple".ljust(8, "\_"))

#>> Apple\_\_\_

Lưu ý là phương thức ljust() không thay đổi chuỗi ban đầu, mà dùng chuỗi ban đầu để tạo ra chuỗi mới được căn trái mà thôi.

### Căn lề giữa một chuỗi trong Python |phương thức center()

Cú pháp:

str.center(width,fillchar)

Trong đó str là chuỗi cần căn giữa, center để gọi phương thức, còn width để chỉ định chiều dài chuỗi kết quả và fillchar là một ký tự dùng để lấp chỗ trống trong chuỗi nếu có.

VD:

str1="học lập trình"

print("[" + str1.center(20) + "]")

#>> [   học lập trình    ]

str2 ="10082019"

print("[" + str2.center(20) + "]")

#>> [      10082019      ]

print("Apple".center(8, "\_"))

#>> \_Apple\_\_

Lưu ý là phương thức center() cũng không thay đổi chuỗi ban đầu, mà dùng chuỗi ban đầu để tạo ra chuỗi mới được căn trái mà thôi.

### Căn lề phải một chuỗi trong Python |phương thức rjust()

Cú pháp:

str.rjust(width,fillchar)

Sử dụng như center()

### Chỉ định width nhỏ hơn hoặc bằng độ dài chuỗi ban đầu khi sử dụng các phương thức ljust, center, rjust

Nếu như chúng ta chỉ định giá trị width - độ dài chuỗi kết quả nhỏ hơn hoặc bằng với độ dài chuỗi ban đầu, kết quả trả về sẽ chính là chuỗi ban đầu.

### Căn lề phải một chuỗi và lấp khoảng trắng trong chuỗi bằng số 0 | phương thức zfill()

Cú pháp:

str.zfill(width)

Trong đó width là độ dài của chuỗi kết quả thu về của phương thức zfill. Và nếu độ dài này lớn hơn độ dài của chuỗi str thì ký tự 0 sẽ được dùng để lấp các ký tự còn thiếu.

VD:

print("89".zfill(5))

#>> 00089

print("abc".zfill(6))

#>> 000abc

Trong trường hợp width - độ dài của chuỗi kết quả nhỏ hơn hoặc bằng với độ dài của chuỗi ban đầu thì chuỗi ban đầu sẽ được trả về.

Bạn có thể thấy phương thức zfill cũng cho kết quả giống với khi chúng ta dùng phương thức rjustr với đối số fillchar bằng 0.

Tuy nhiên điều khác biệt ở đây là nếu chúng ta chỉ định dấu + hoặc - vào trước chuỗi ký tự str, thì phương thức zfill sẽ lấp chỗ trống bằng ký tự số 0 bắt đầu từ sau dấu + hoặc -, còn phương thức rjust với đối số fillchar bằng 0 sẽ chỉ đơn giản lấp chỗ trống từ trước dấu + hoặc - mà thôi. Hãy xem kỹ ví dụ sau đây:

print("-123".zfill(8))

#>> -0000123

print("+975".zfill(8))

#>> +0000975

print("-123".rjust(8,"0"))

#>> 0000-123

print("+975".rjust(8,"0"))

#>> 0000+975

## Đảo ngược chuỗi

### Đảo ngược chuỗi trong python| Phương thức reverse()

Phương thức reverse() vốn được sử dụng để đảo ngược list trong Python. Tuy nhiên chúng ta lại không thể sử dụng phương thức này trực tiếp với chuỗi được.

Để đảo ngược chuỗi ban đầu, chúng ta cần chuyển chuỗi sang list, sau đó đảo ngược list thu được trước khi chuyển lại list đó về dạng chuỗi.

VD:

org\_str = 'aiueo'

new\_str\_list = list(org\_str)

new\_str\_list.reverse()

new\_str = ''.join(new\_str\_list)

print(org\_str)

print(new\_str)

#>> aiueo

#>> oeuia

### Đảo ngược chuỗi trong python | Hàm reversed()

Tương tự như với phương thức phương thức reverse(), chúng ta cũng không thể sử dụng trực tiếp hàm reversed() với kiểu chuỗi được.

Tuy nhiên khác với phương thức reverse() không thể sử dụng với tuple thì hàm reversed() lại có thể sử dụng với tuple được, do đó để đảo ngược chuỗi trong python, chúng ta có 2 sự lựa chọn là chuyển chuỗi sang tuple hoặc list, sau đó đảo ngược tuple hoặc list thu được trước khi chuyển lại kết quả về dạng chuỗi.

VD:

org\_str = 'aiueo'

new\_str\_list = tuple(reversed(org\_str))

new\_str = ''.join(new\_str\_list)

print(org\_str)

print(new\_str)

#>> aiueo

#>> oeuia

Tương tự, chúng ta cũng có thể đảo ngược chuỗi trong python bằng cách ứng dụng hàm reversed() và list như sau:

org\_str = 'aiueo'

new\_str\_list = list(reversed(org\_str))

new\_str = ''.join(new\_str\_list)

print(org\_str)

print(new\_str)

#>> aiueo

#>> oeuia

### Đảo ngược chuỗi trong python | Cắt (slice) chuỗi

orinary\_str = org\_str = 'aiueo'

new\_str = orinary\_str[::-1]

print(orinary\_str)

print(new\_str)

#>> aiueo

#>> oeuia

## So sánh chuỗi

### Kiểm tra mã unicode của một ký tự

Sử dụng hàm ord()

VD:

print(ord("a"))

#> 97

print(ord("b"))

#> 98

print(ord("A"))

#> 65

### So sánh chuỗi bằng toán tử

Cách thức: so sánh mã unicode của từng ký tự trong 2 chuỗi cho đến khi có sự chênh lệch.

VD:

print('abc' == 'abc')

##> True

print('abc' == 'def')

##> False

print('abc' != 'abc')

##> False

print('abc' != 'def')

##> True

print('a' < 'b')

#>> True

print('aa' <= 'ab')

#>> True

print('abc' > 'abcd')

#>> False

print('abc' >= 'abcd')

#>> False

### Kiểm tra chuỗi con với toán tử in

print('cc' in 'bb-cc-dd')

#>> True

print('1' in '234')

#>> False

### startswith(), endswith() Kiểm tra đầu chuỗi và cuối chuỗi có bắt đầu bằng một chuỗi con khác

VD:

s = 'aaa-bbb-ccc'

print(s.startswith('aaa'))

##> True

print(s.startswith('bbb'))

#>> False

s = 'aaa-bbb-ccc'

print(s.endswith('ccc'))

##> True

print(s.endswith('bbb'))

#>> False

### So sánh không phân biệt hoa thường.

Sử dụng upper() và lower()

VD:

s1 = 'abcd1'

s2 = 'ABCD1'

print(s1 == s2)

#>> False

print(s1.lower() == s2.lower())

#>> True

print(s1.upper() == s2.upper())

#>> True

### Lấy ra các phần tử giống nhau trong 2 chuỗi| sử dụng set và toán tử and

s1 = 'abcde'

s2 = 'cde'

s3 = 'abcdef'

print( set(s1) and set(s2) )

#>> {'e', 'd', 'c'}

print( set(s1) & set(s3) )

#>> {'d', 'b', 'e', 'c', 'a'}

print(type(set(s1) & set(s3)))

#>> <class 'set'>

### Lấy ra các phần tử khác nhau trong 2 chuỗi| sử dụng set và toán tử ^ (XOR) (tìm phần bù)

VD:

s1 = 'abcde'

s2 = 'cde'

s3 = 'abcdef'

print( set(s1) ^ set(s2) )

#>> {'b', 'a'}

print( set(s1) ^ set(s3) )

#>> {'f'}

print(type(set(s1) ^ set(s3)))

#>> <class 'set'>

## Ký tự đặc biệt (escape sequence)

### Bảng ký tự đặc biệt

| **số thứ tự** |  | **ký tự đặc biệt** | **Ý nghĩa** |
| --- | --- | --- | --- |
| 0 |  | \ | xuống dòng khi viết chuỗi nhiều dòng |
| 1 |  | \\ | dấu gạch chéo ngược (\) |
| 2 |  | \' | dấu nháy đơn (‘) |
| 3 |  | \" | dấu nháy kép (“) |
| 4 |  | \a | [ký tự chuông BEL trong chuỗi](https://en.wikipedia.org/wiki/Bell_character) |
| 5 |  | \b | ký tự backspace BS trong chuỗi |
| 6 |  | \f | ký tự Form feed FF trong chuỗi |
| 7 |  | \n | ký tự xuống dòng trong chuỗi |
| 8 |  | \r | ASCII CF |
| 9 |  | \t | ký tự TAB |
| 10 |  | \v | ASCII VT |
| 11 |  | \000 | ASCII ký tự 000 của hệ cơ số 8 |
| 12 |  | \xhh | ASCII ký tự xhh của hệ cơ số 16 |
| 13 |  | \N {name} | Ký tự có tên trong cơ sở dữ liệu Unicode |
| 14 |  | \uxxx | Unicode ký tự xxx của hệ cơ số 16 |
| 15 |  | \Uxxxxxxxx | Unicode ký tự xxxxxx của hệ cơ số 32 |

VD: xuống dòng khi viết chuỗi nhiều dòng

print("Tối nay hẹn em 12h \

bên bờ hồ")

Tối nay hẹn em 12h bên bờ hồ

### Loại bỏ ký tự đặc biệt

Cách 1: Sử dụng isalnum() để tìm ký tự đặc biệt, sau đó loại bỏ ký tự đó đi.

string = "Hey! What's up bro?"

new\_arr= []

for char in string:

    if (char.isalnum() == True):

        new\_arr.append(char)

new\_string = ''.join(new\_arr)

print(new\_string)

#>> HeyWhatsupbro

Cách 2: sử dụng filter() và isalnum()

Hàm filter()

Cú pháp: filter(function, iterable)

Hàm filter() trả về một trình vòng lặp là các mục được lọc thông qua một hàm để kiểm tra xem mục đó có được chấp nhận hay không.

string = "Hey! What's up bro?"

new\_string = ''.join(filter(str.isalnum, string))

print(new\_string)

#>> HeyWhatsupbro

Cách 3: Sử dụng RegEx

import re

string = "Hey! What's up bro?"

new\_string = re.sub(r"[^a-zA-Z0-9]","",string)

print(new\_string)

#>> HeyWhatsupbro

## Chuỗi thô (raw string)

Ở trong python, nếu chúng ta thêm ký tự r hoặc R vào trước một chuỗi chứa chuỗi thoát, thì những chuỗi thoát trong đó sẽ không bị coi là ký hiệu đặc biệt mà sẽ được coi như là một ký tự bình thường trong chuỗi.

VD:

rs = r'a\tb\nA\tB'

print(rs)

a\tb\nA\tB

Lưu ý:

Nếu bạn viết chuỗi thô với một số lẻ dấu gạch chéo ngược \ khi kết thúc chuỗi? kết quả, lỗi EOL sẽ bị trả về như sau

rpath2 = r'C:\Windows\system32\'

File "Main.py", line 1

rpath2 = r'C:\Windows\system32\'

^

SyntaxError: EOL while scanning string literal

Khắc phục: Kết thúc chuỗi thô bằng một số chẵn dấu gạch chéo ngược.

rpath2 = r'C:\Windows\system32'   # kết thúc bởi 0 dấu `\\`

path2 = r'C:\\Windows\\system32\\' # kết thúc bởi 2 dấu `\\`

rpath2\_2 = r'C:\Windows\system32' + '\\' # kết thúc bởi 2 dấu `\\`

## Định dạng chuỗi

Tham khảo: <http://laptrinhcanban.com/python/nhap-mon-lap-trinh-python/thao-tac-voi-chuoi-string-trong-python/dinh-dang-chuoi-trong-python-bang-toan-tu-dinh-dang/>

### Định dạnh bằng toán tử %

Cú pháp:

chuỗi chứa định dạng % giá trị

chuỗi chứa định dạng % (giá trị 1, giá trị 2, giá trị 3…)

“chuỗi %(key1)format … %(keyn)format” % {‘key1’:value, … ‘keyn’:value}

Format:

%[flags][width][.precision]specifier

Trong đó:

* flags chứa các chỉ định có tác dụng căn chỉnh, thêm dấu , đệm số 0 v.v..
* width: độ dài tối thiểu của kết quả xuất
* .precision : độ chính xác khi xuất số thực (số chữ số đằng sau dấu phẩy số thực sau khi làm tròn)
* specifier là định dạng của chuỗi kết quả

|  |  |
| --- | --- |
| flag | Ý nghĩa |
| - | Căn trái (mặc định không có dấu trừ là căn phải) |
| + | Thêm dấu cho số |
| 0 | Đệm số 0 cho đủ độ dài kết quả xuất |

VD:

# %[flags][width][.precision]specifier

# chiều rộng bằng 8

# căn phải (mặc định)

print("Hi!\_\_%8s\_\_" % "LyLy")

#>> Hi!\_\_    LyLy\_\_

# căn trái

print("Hi!\_\_%-8s\_\_" % "LyLy")

#>> Hi!\_\_LyLy    \_\_

# đệm 0 (chỉ có tác dụng với số)

# căn trái

print("\_\_%08d\_\_" % 12)

#>> \_\_00000012\_\_

# đệm 0, width = 8 (tính cả dấu chấm), làm tròn 3 chứ số sau đấu chấm

# căn trái

print("\_\_%08.3f\_\_" % 12.34567)

#>> \_\_0012.346\_\_

VD:

print("Hi! %s" % "LyLy")

#>> Hi! LyLy

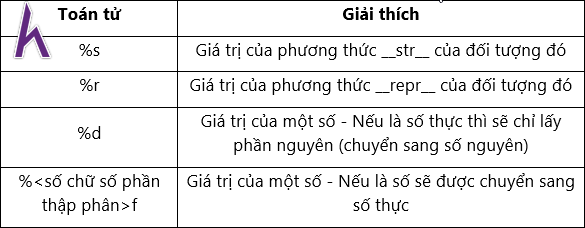
print("Tên tôi là %-8s , tuổi là %03d" % ("Kiyoshi",30))

#>> Tên tôi là Kiyoshi  , tuổi là 030

print('Tôi là %(key1)s, năm nay %(key2)d' % {'key1':'hao','key2':20})

#>> Tôi là hao, năm nay 20

Toán tử định dạng



| **Định dạng** | **Ý nghĩa** |
| --- | --- |
| d | Số nguyên thập phân có dấu |
| i | Số nguyên thập phân có dấu |
| o | Số bát phân có dấu |
| u | Kiểu mới của d |
| x | Số thập lục phân có dấu(chữ thường) |
| X | Số thập lục phân có dấu(chữ hoa) |
| e | số biểu diễn dưới dạng ký hiệu khoa học |
| E | số biểu diễn dưới dạng ký hiệu khoa học |
| f | Ký hiệu điểm cố định |
| F | Ký hiệu điểm cố định |
| g | Số điểm nổi |
| G | Số điểm nổi |
| c | ký tự |
| r | chuyển đối ký tự bằng hàm repr() |
| s | chuyển đối ký tự bằng hàm str() |
| a | chuyển đối ký tự bằng hàm ascii() |

### Định dạnh chuỗi bằng f

Cú pháp:

f“abc{value:pattern}xyz”

Trong đó:

* Ký tự f hoặc F đầu chuỗi biểu thị việc chúng ta sẽ sử dụng chuỗi f để định dạng chuỗi trong python.
* pattern là định dạng chuỗi.
* abc và xyz là các ký tự bất biến trong chuỗi cần định dạng, và {value:pattern} là phần trong chuỗi cần được định dạng.

pattern

Định dạng căn lề

s = 'abc'

print(f'right : {s:\_>8}')

print(f'center: {s:\_^8}')

print(f'left  : {s:\_<8}')

# right : \_\_\_\_\_abc

# center: \_\_abc\_\_\_

# left  : abc\_\_\_\_\_

Lấp chỗ trống bằng số 0

i = 1234

print(f'zero padding: {i:08}')

# zero padding: 00001234

Dấu phân tách số

i = 1234

print(f'comma: {i:,}')

# comma: 1,234

Độ chính xác của phần phân số

f = 12.3456

print(f'digit(decimal): {f:.3f}')

print(f'digit(all)    : {f:.3g}')

# digit(decimal): 12.346

# digit(all)    : 12.3

Kiểu giá trị e

print(f'exponen: {f:.3e}')

# exponen: 1.235e+01

Dạng phần trăm

f = 0.123

print(f'percent: {f:.2%}')

# percent: 12.30%

Kiểu bin, oct, hex

print(f'bin: {i:b}')

print(f'oct: {i:o}')

print(f'hex: {i:x}')

# bin: 10011010010

# oct: 2322

# hex: 4d2

print(f'bin: {i:#b}')

print(f'oct: {i:#o}')

print(f'hex: {i:#x}')

# bin: 0b10011010010

# oct: 0o2322

# hex: 0x4d2

Viết biểu thức tính toán

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

# 3 + 4 = 7

print(f'{a} \* {b} = {a \* b}')

# 3 \* 4 = 12

print(f'{a} / {b} = {a / b:.2e}')

# 3 / 4 = 7.50e-01

Sử dụng dictionary

d = {'key1': 3, 'key2': 4}

#Sử dụng key dictionary với phương thức format

print('{0[key1]}, {0[key2]}'.format(d))

#Sử dụng key dictionary với chuỗi f

print(f'{d["key1"]}, {d["key2"]}')

### Định dạng chuỗi bằng format

Cú pháp:

pattern. format ( value1 , value2 , value3 , …)

VD:

# thay thế theo thứ tự

print("{} is {} years old".format('Kiyoshi','30'))

#>Kiyoshi is 30 years old

# Định index cho trường thay thế

print("{1} and {0} and {2}".format('a','b', 'c'))

#>> b and a and c

# Định dạnh theo kiểu key - value

print("{b} and {a} and {c}".format(a='a',b='b', c='c'))

#>> b and a and c

# value là biểu thức

print("{} + {} = {}".format(1, 2, 1+2))

#>> 1 + 2 = 3

Sử dụng dictionary

d = {'key1': 3, 'key2': 4}

#Sử dụng key dictionary với phương thức format

print('{0[key1]}, {0[key2]}'.format(d))

#Sử dụng key dictionary với chuỗi f

print(f'{d["key1"]}, {d["key2"]}')

Định dạng các trường thay thế

pattern:

[index/key]:[Align][Sign][Width][Grouping\_option][.Precision][#][Type]

Trong đó:

* Align: căn lề
* Sign: dấu
* Width: độ dài tối thiểu của kết quả xuất
* Grouping\_option: nhóm cụm 3 số và phân tách bằng dấu phẩy , hoặc dấu gạch dưới \_
* .precision : độ chính xác khi xuất số thực (số chữ số đằng sau dấu phẩy số thực sau khi làm tròn).
* Dấu #: Định dạng chuỗi với cách biểu diễn khác nhau của số.
* type là định dạng của chuỗi kết quả.

type

| **Type** | **Định dạng kiểu dữ liệu** | **Ví dụ** |
| --- | --- | --- |
| s | Kiểu chuỗi | {:s} |
| b | Kiểu số nhị phân | {:b} |
| c | Kiểu ký tự số | {:c} |
| d | Kiểu số thập phân | {:d} |
| o | Kiểu số bát pâhn | {:0} |
| x | Kiểu số thập lục phân | {:x} |
| X | Kiểu số thập lục phân | {:X} |
| n | Kiểu số nguyên | {:n} |
| e | Kiểu mũ | {:e} |
| E | Kiểu mũ | {:E} |
| f | Kiểu thập phân cố định | {:f} |
| F | Kiểu thập phân cố định | {:F} |
| g | Kiểu làm tròn số | {:g} |
| G | Kiểu làm tròn số | {:G} |
| % | Kiểu phần | {:%} |

align

| **Ký hiệu** | **Định dạng** | **Ví dụ** |
| --- | --- | --- |
| < | căn trái chuỗi | {:<7d} |
| > | căn phải chuỗi | {:>7d} |
| ^ | căn giữa chuỗi | {:^7d} |
| fill | lấp chỗ trống | {:0^7d} |

Sign

| **Ký hiệu** | **Định dạng** | **Ví dụ** |
| --- | --- | --- |
| + | Định dạng số dương (có thể sử dụng cho cả số âm) | {:+d} |
| - | Định dạng số âm ( chỉ sử dụng cho số âm) | {:-d} |
| ' ' | Thêm khoảng trống vào số dương | {: d} |

VD:

# pattern: [index/key]:[Align][Sign][Width][Grouping\_option][.Precision][#][Type]

# Định dạng kiểu dữ liệu (type) ---------------

# chuỗi, nhị phân, bát phân, thập lục phân, phần trăm

print("{:s}, {:b}, {:o}, {:x}, {:%}".format("string", 31, 31, 31, 0.12))

print("{:s}, {:#b}, {:#o}, {:#x}, {:%}".format("string", 31, 31, 31, 0.12))

#>> string, 11111, 37, 1f, 12.000000%

#>> string, 0b11111, 0o37, 0x1f, 12.000000%

# Định dạng độ rộng tối thiểu (width), căn lề, đệm 0 -----------

# căn trái, căn phải, giữa, đệm 0

print("\_{:<8}\_ \_{:>8}\_ \_{:^8}\_ \_{:08}\_".format("hello", "hello", "hello", 123.4))

#>> \_hello   \_ \_   hello\_ \_ hello  \_ \_000123.4\_

# Định dạng dấu(sign) (không có ý nghĩa trong python) -------------

print("{:-} {:-} {} {:+}".format(123.4, -123.4, -123.4, -123.4))

#>> 123.4 -123.4 -123.4 -123.4

# Định dạng dấu phân tách cụm 3 số cho dễ nhìn (group\_option) --------------

# định dạng với số nguyên

d = 12345678

print("{:,}  {:\_}  {:014,}".format(d, d, d))

#>> 12,345,678  12\_345\_678  00,012,345,678

# định dạng với số thập phân

f = 1234.5678

print("{:,}  {:014,}  {:014,.5f}".format(f, f, f))

#>> 1,234.5678  0,001,234.5678  0,001,234.56780

# định dạng .precision ----------------------------------------------------

# cần type = f

f = 1234.5678

print("{:.2f}  {:014.5f}  {:014,.5f} ".format(f, f, f))

#>> 1234.57  00001234.56780  0,001,234.56780

print("{:.2}  {:014.5}  {:014,.5} ".format(f, f, f))

#>> 1.2e+03  000000001234.6  0,000,001,234.6

## Các phương thức xử lý chuỗi

### Ljust()

Căn trái chuỗi

Cú pháp:

str.ljust(width,fillchar=space)

* width là chiều dài chỉ định của chuỗi kết quả.
* fillchar là một ký tự dùng để lấp chỗ trống.

Lưu ý là chỉ có thể chỉ định giá trị fillchar bằng một ký tự, nếu không lỗi TypeError sẽ xảy ra.

print("["+ "hello".ljust(8) + "]")

#>> [hello   ]

print("["+ "hello".ljust(8, "\_") + "]")

#>> [hello   ]

### Rjust()

Căn phải chuỗi

Cú pháp:

str.rjust(width,fillchar=space)

VD:

print("["+ "hello".rjust(8) + "]")

#>> [   hello]

print("["+ "hello".rjust(8, "\_") + "]")

#>> [\_\_\_hello]

### Zfill()

Zero fill: đệm số 0 vào số.

Cú pháp:

str.zfill(width)

VD:

print("["+ "+123.456".zfill(8) + "]")

#>> [+123.456]

print("["+ "-123.456".zfill(8) + "]")

#>> [-123.456]

### Lower()

Chuyển toàn bộ thành chữ thường

Cú pháp:

str.lower()

VD:

print("123abcDEF@#".lower())

#>> 123abcdef@#

### Upper()

Chuyển toàn bộ thành chữ hoa

Cú pháp:

str.upper()

VD:

print("123abcDEF@#".upper())

#>> 123ABCDEF@#

### Swapcase()

Chuyển đổi chữ thường và chữ hoa

Cú pháp:

str.swapcase()

VD:

print("hello YOU @# 123".swapcase())

#>> HELLO you @# 123

### Capitalize()

Viết hoa chữ cái đầu

str.capitalize()

VD:

print("hello you @# 123".capitalize())

#>> 123ABCDEF@#

### Title()

Viết hoa các chữ cái đầu của từ, các chữ cái còn lại viết thường

Cú pháp:

str.title()

VD:

print("hello YOU @# 123".title())

#>> Hello You @# 123

### Join()

Ghép iterable thành chuỗi

Cú pháp:

str.join(iterable)

Trong đó

* str là ký tự phân cách dùng để nối chuỗi
* join là tên phương thức
* iterable là list hoặc tuple chứa phần tử là các chuỗi cần nối.

Lưu ý: các phần tử trong iterable phải là kiểu chuỗi

VD:

# join list

print(" and ".join(["123", "hello", "world"]))

#>> 123 and hello and world

# join tuple

print(" and ".join(("123", "hello", "world")))

#>> 123 and hello and world

# List

## Khởi tạo

5 phương pháp tạo list:

* Sử dụng cặp dấu [] : chỉ định trực tiếp giá trị phần tử trong list
* Sử dụng biến : chỉ định gián tiếp giá trị phần tử trong list
* Sử dụng toán tử \* : lặp list và tạo ra list mới.
* Sử dụng hàm list() : chuyển string, tuple , dictionary sang list trong Python
* Sử dụng hàm range() : tạo ra list có các phần tử là một dãy số liên tục

VD:

# tạo list thông thường

a = 12; b = 2.34

list1 = [123, "hello", 123.456, ["str", 123], a, b]

# Tạo list trống

list2 = []

list3 = list()

# lặp list

list4 = ["123", "456"]

list5 = list4 \* 3

print(list5)

#>> ['123', '456', '123', '456', '123', '456']

# tạo list bằng hàm list() ---------------------

# tạo từ iterable khác

tuple1 = (123, 456)

list6 = list(tuple1)

print(list6)

#>> [123, 456]

# tạo từ chuỗi

str1 = "hello world"

list7 = list(str1)

print(list7)

#>> ['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']

# Tạo list bằng hàm range() ---------------------------------

nums1 = list(range(5))

print(nums1)

#>> [0, 1, 2, 3, 4]

nums2 = list(range(-5, 6))

print(nums2)

#>> [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]

nums3 = list(range(1, 11, 2))

print(nums3)

#>> [1, 3, 5, 7, 9]

## Len()

Đếm số phần tử trong list

VD:

a = 12; b = 2.34

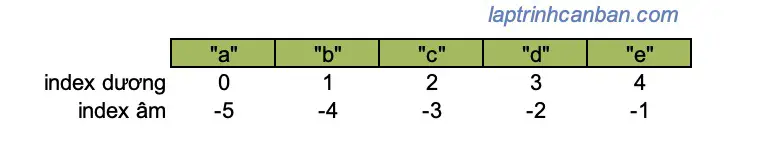
list1 = [123, "hello", 123.456, ["str", 123], a, b]

print(len(list1))

#>> 6

## Index

Index có index dương và index âm



### Truy cập bằng index

VD:

a = 12; b = 2.34

list1 = [123, "hello", 123.456, ["str", 123], a, b]

print(list1[0])

#>> 123

print(list1[:2])

#>> [123, 'hello']

print(list1[:])

#>> [123, 'hello', 123.456, ['str', 123], 12, 2.34]

print(list1[::-1])

#>> [2.34, 12, ['str', 123], 123.456, 'hello', 123]

print(list1[-1])

#>> 2.34

### Thay đổi giá trị bằng index

VD:

list1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# thay đổi 1 giá trị ---------------

list1[-1] = 99

print(list1)

#>>[0, 1, 2, 3, 4, 5, 6, 7, 8, 99]

# thay đổi dải giá trị ------------

# cùng số lượng phần tử

list1[6:8] = [66, 77]

print(list1)

#>>[0, 1, 2, 3, 4, 5, 66, 77, 8, 99]

# khác số lượng phần tử

list1[1:5] = [11, 22]

print(list1)

#>>[0, 11, 22, 5, 66, 77, 8, 99]

## List comprehension: phương pháp viết nội hàm

Cú pháp đơn giản:

[expression for variable in iterable]

Trong đó:

* iterable là một đối tượng chứa nhiều phần tử trong python, ví dụ như một list, tuple, range hay set.
* variable là một biến để gán giá trị phần tử lấy ra từ iterable.
* expression là một biểu thức để tính toán với biến variable.

VD:

iter1 = ("hello", "world")

list1 = [number for number in iter1]

print(list1)

#>> ['hello', 'world']

list2 = [number\*\*2 for number in range(0,6)]

print(list2)

#>> [0, 1, 4, 9, 16, 25]

Cú pháp có điều kiện:

[expression for variable in iterable if if\_expression]

Trong đó:

* iterable là một đối tượng chứa nhiều phần tử trong python, ví dụ như một list, tuple, range hay set.
* variable là một biến để gán giá trị phần tử lấy ra từ iterable.
* if\_expression là biểu thức điều kiện để xét có tiếp tục thực hiện vòng lặp for hay không.
* expression là một biểu thức để tính toán với biến variable.

VD:

list1 = [number for number in range(0,6) if number%2==0]

print(list1)

#>> [0, 2, 4]

## Xoá phần tử

### Pop()

Lấy và xoá phần tử trong list

Cú pháp:

list.pop(index=-1)

VD:

list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]

end = list1.pop()

print(end)

#>> 9

print(list1)

#>> [1, 2, 3, 4, 5, 6, 7, 8]

index1 = list1.pop(1)

print(index1)

#>> 2

print(list1)

#>> [1, 3, 4, 5, 6, 7, 8]

### Del

Xoá phần tử trong phạm vi chỉ định

Cú pháp:

del range

Trong đó: range là phạm vi phần tử cần xoá.

VD:

list1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Xoá 1 phần tử

del list1[-1]

print(list1)

#>> [0, 1, 2, 3, 4, 5, 6, 7, 8]

# Xoá nhiều phần tử

del list1[2:6]

print(list1)

#>> [0, 1, 6, 7, 8]

# Xoá toàn bộ các phần tử

del list1[:]

print(list1)

#>> []

# Xoá list, giải phóng bộ nhớ

del list1

print(list1)

'''

Traceback (most recent call last):

  File "d:\PROGRAMMING\_LANGUAGE\Python\_Tutorial\Code\hello\_world\highline.py", line 15, in <module>

    print(list1)

NameError: name 'list1' is not defined

'''

### Remove()

Xoá phần tử đầu tiên trong list giống với giá trị chỉ định

Cú pháp:

list.remove(value)

VD:

list1 = [0, 1, 2, 3, 0, 1, 2, 3]

# Xoá 1 phần tử

list1.remove(2)

print(list1)

#>> [0, 1, 3, 0, 1, 2, 3]

### Clear()

Xoá toàn bộ các phần tử (không xoá biến list)

Cú pháp:

list.clear()

VD:

list1 = [0, 1, 2, 3, 0, 1, 2, 3]

# Xoá toàn bộ phần tử

list1.clear()

print(list1)

#>> []

### Xoá các phần tử trùng lặp | sử dụng set

VD:

mylist = ['red', 'blue', 'yellow', 'black' , 'blue']

unique\_list=list(set(mylist))

print(unique\_list)

#>> ['blue', 'red', 'black', 'yellow']

## List.sort() Xắp xếp phần tử

Xắp xếp phần tử

List.sort(reverse=False)

Trong đó:

* reverse = False: Xắp xếp theo chiều tăng dần
* reverse = True: Xắp xếp theo chiều giảm dần

VD:

list1 = [0, 2, 3, 1, 2, 4, 6, 4]

list1.sort()

print(list1)

#>> [0, 1, 2, 2, 3, 4, 4, 6]

list1.sort(reverse=True)

print(sorted(list1, reverse=True))

#>> [6, 4, 4, 3, 2, 2, 1, 0]

## So sánh phần tử

### So sánh 2 list giống nhau

print(["hello", "world"] == ["hello", "world"])

#>> True

print(["world", "world"] == ["world", "hello"])

#>> False

print(["hello", 123] != ["hello", "world"])

#>> True

print(["world", 123] == ["world", "hello"])

#>> False

### So sánh 2 list và lấy ra các phần tử giống nhau | sử dụng set và toán tử &

Để so sánh 2 list trong python và lấy các phần tử giống nhau, chúng ta cần chuyển 2 list đã cho về kiểu set (tập hợp), sau đó mới tìm ra phần tử giống nhau giữa chúng thông qua phép tìm giao hai tập hợp bằng toán tử & trong python.

l1 = [0,1,2,3,'Bob']

l2 = [0,1,5,3]

l3 = [0,1,'Bob',3]

print( set(l1) & set(l2) )

#>> {0, 1, 3}

print( set(l1) & set(l3) )

#>> {0, 1, 3, 'Bob'}

print(type(set(l1) & set(l3)))

#>> <class 'set'>

### So sánh 2 list và lấy ra các phần tử giống nhau | sử dụng set và toán tử ^

Để so sánh 2 list trong python và lấy các phần tử khác nhau, chúng ta cần chuyển 2 list đã cho về kiểu set (tập hợp), sau đó mới tìm ra phần tử khác nhau giữa chúng thông qua phép tìm phần bù giữa hai tập hợp bằng toán tử ^ trong python.

l1 = [0,1,2,3,'Bob']

l2 = [0,1,5,3]

l3 = [0,1,'Bob',4]

print( set(l1) ^ set(l2) )

#>> {2, 5, 'Bob'}

print( set(l1) ^ set(l3) )

#>> {2, 3, 4}

print(type(set(l1) ^ set(l3)))

#>> <class 'set'>

## Thêm phần tử

### Append()

Thêm một phần tử vào cuối list

Cú pháp:

list.append(element)

l = [0, 1, 3]

l.append(1)

print(l)

#>> [0, 1, 3, 1]

l.append("abc")

print(l)

#>> [0, 1, 3, 1, 'abc']

l.append((1, 2))

print(l)

#>> [0, 1, 3, 1, 'abc', (1, 2)]

### Extend()

Thêm nhiều phần tử từ một iterable khác vào cuối list

Cú pháp:

list.extend(iterable)

l = [0, 1, 3]

l.extend([1, 2])

print(l)

#>> [0, 1, 3, 1, 2]

l.extend("abc")

print(l)

#>> [0, 1, 3, 1, 2, 'a', 'b', 'c']

# Hàm khác

## Min()

Tìm phần tử có giá trị nhỏ nhất.

Cú pháp:

min(iterable)

VD:

list1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(min(list1))

#>> 0

list2 = ["hello", "world", "there", "are"]

print(min(list2))

#>> are

Với chuỗi, so sánh giá trị theo bảng mã UNICODE

## Max()

Tìm phần tử có giá trị lớn nhất

Cú pháp:

max(iterable)

VD:

list1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(max(list1))

#>> 9

list2 = ["hello", "world", "there", "are"]

print(max(list2))

#>> world

Với chuỗi, so sánh giá trị theo bảng mã UNICODE

## Sorted()

Xắp xếp các phần tử của iterable.

Cú pháp:

sorted(iterable, reverse=False)

Trong đó:

* reverse = False: Xắp xếp theo chiều tăng dần
* reverse = True: Xắp xếp theo chiều giảm dần

VD:

list1 = [0, 2, 3, 1, 2, 4, 6, 4]

print(sorted(list1))

#>> [0, 1, 2, 2, 3, 4, 4, 6]

print(sorted(list1, reverse=True))

#>> [6, 4, 4, 3, 2, 2, 1, 0]

# Tham khảo

Lập trình căn bản:

<https://laptrinhcanban.com/cpp/lap-trinh-cpp-co-ban/>