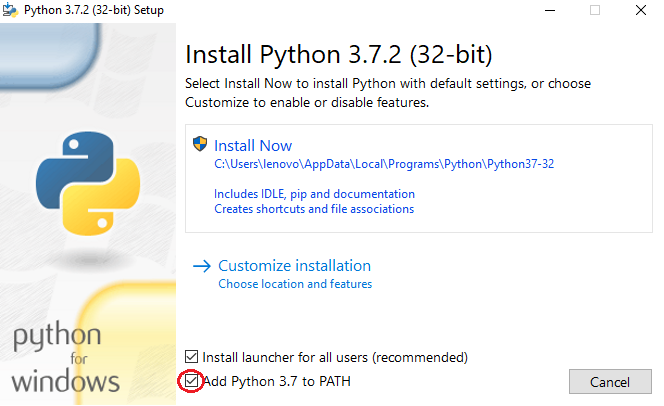
https://dttvn0010.github.io/python\_tutorial/basic/

Cài đặt môi trường phát triển python

Cài đặt python

* Download và cài đặt python tại :[https://www.python.org/downloads/](https://www.google.com/url?q=https://www.python.org/downloads/&sa=D&ust=1550420047593000)
* Lưu ý khi cài đặt: Tại màn hình cửa sổ đầu tiên khi chạy chương trình cài đăt, nên chọn option “**Add Python 3.x to PATH**”, để thuận lợi cho việc chạy python từ cửa sổ cmd sau khi cài đặt



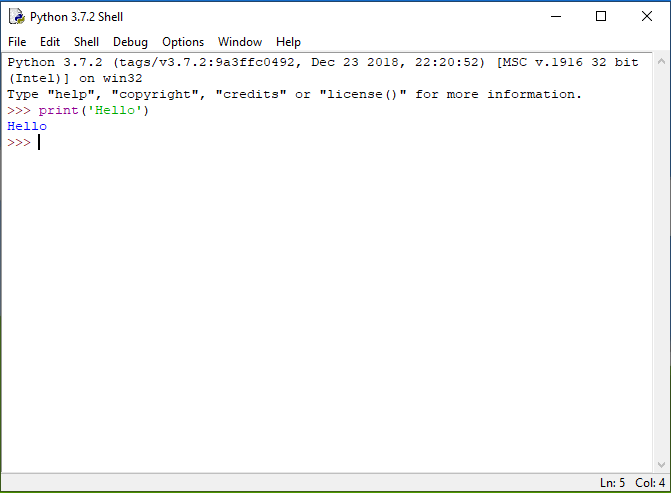
Thêm lựa chọn “Add Python to PATH” khi cài đặt Python

Tạo mới và chạy thử một chương trình

Từ menu start của Windows tìm ứng dụng IDLE và mở ứng dụng này

Gõ vào cửa sổ một lệnh để chạy thử:

print('Hello')



Cửa sổ ứng dụng IDLE

**Chạy chương trình từ File :**

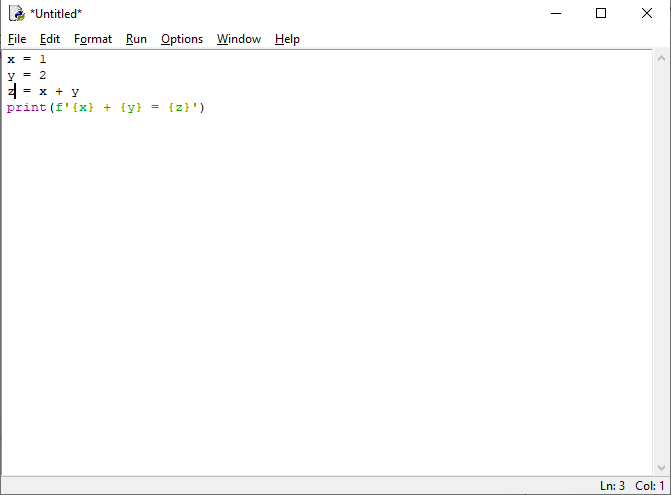
Từ Menu **File**, chọn New File, chương trình sẽ tạo ra một cửa sổ soạn thảo mới, nhập vào cửa sổ này nội dung một chương trình đơn giản, ví dụ:

x = 1

y = 2

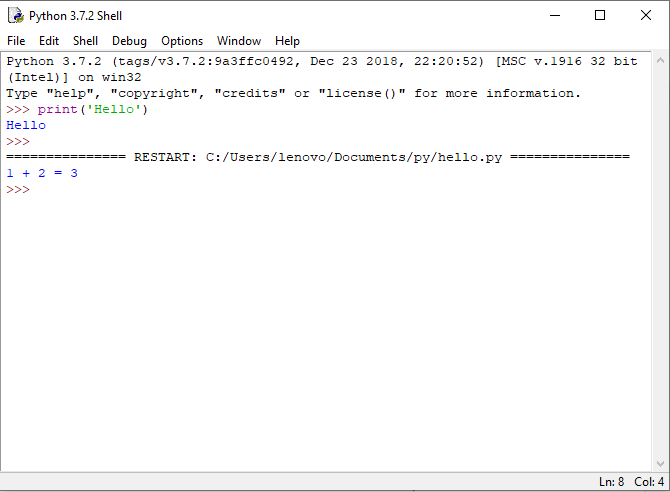
z = x + y

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



Tạo mới một chương trình

Lưu lại chương trình thành file (ví dụ hello.py), sau đó chạy chương trình bằng cách từ Menu **Run**, chọn **Run Module (F5)**. Chương trình sẽ chạy và in kết quả ra cửa sổ IDLE:



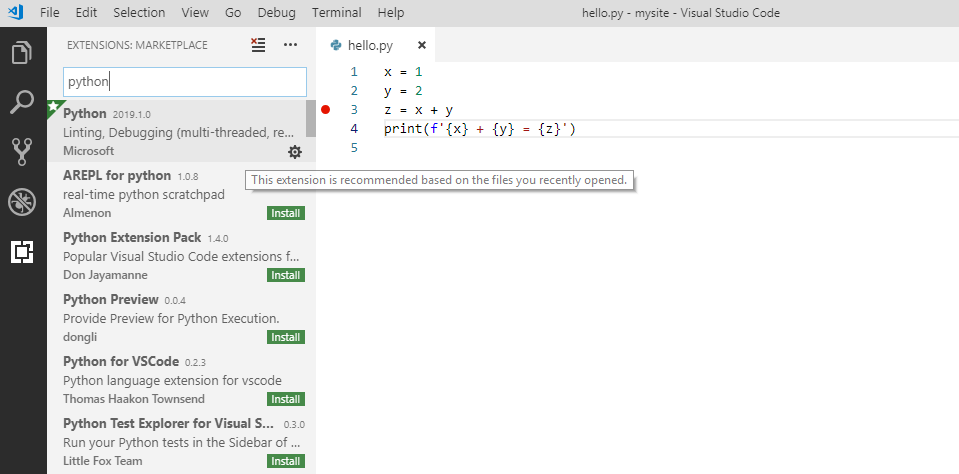
Kết quả chạy chương trình được in ra cửa sổ IDLE

Cài đặt Python IDE

IDLE cho phép soạn thảo và chạy chương trình Python nhưng không hỗ trợ nhiều các chức năng phát triển ứng dụng. Với các ứng dụng python có nhiều file, nên sử dụng IDE khác. Có thể sử dụng một trong các IDE:

* Pycharm
* Eclipse + Pydev
* Visual Studio Code

Pycharm được xem là IDE hỗ trợ Python đầy đủ nhất, tuy nhiên Visual Studio Code tương đối nhẹ trong khi hỗ trợ Python tương đối tốt. Muốn sử dụng Visual Studio Code để phát triển ứng dụng Python, cần cài đặt Python Extension cho Visual Studio Code:



Cài đặt python extension cho Visual Studio Code

Lập trình Python cơ bản

Khai báo biến

Trong python, việc khai báo biến được thực hiện bằng cách gán giá trị khởi đầu cho một biến:

bien = <gia\_tri>

Ví dụ :

x = 1

y = 2

z = x + y

Sau khi được khai báo, giá trị của biến có thể được thay đổi bằng cách gán lại một biểu thức mới cho biến:

      x = 1

      x = 2

      x = x + 1

Để xóa một biến khỏi bộ nhớ, sử dụng lệnh del :

del x, y, z

Vào ra dữ liệu

Đọc dữ liệu từ bàn phím

Lệnh input được sử dụng để nhập dữ liệu từ bàn phím. Dữ liệu trả về của lệnh là một biến string chứa dòng văn bản mà người dùng đã nhập vào:

giatri = input('Nhập giá trị cho biến : ')

Ví dụ 1

|  |
| --- |
| ten = input('Mời bạn cho biết tên của bạn : ') print('Xin chào ', ten) |

Ví dụ 2:

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

In dữ liệu ra màn hình

Lệnh print được sử dụng để in dữ liệu ra màn hình, cú pháp của lệnh này như sau:

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

Ví dụ:

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

**L**ệnh điều khiển if

Cấu trúc lệnh if

Các cấu trúc if trong Python:

* if <điều kiện> :

    <Lệnh>

* if <điều kiện> :

    <Lệnh 1>

else:

    <Lệnh 2>

* if <điều kiện 1> :

    <Lệnh 1>

elif <điều kiện 2>  :

    <Lệnh 2>

elif <điều kiện 3>  :

    <Lệnh 3>

else:

    <Lệnh 4>

Lưu ý:

* Cuối dòng các lệnh if/elif/else là dấu hai chấm. Python sử dụng dấu hai chấm để bắt đầu một khối lệnh con
* Các dòng của khối lệnh con bên dưới if/elif/else được viết lùi vào một số khoảng trắng (thường là 4 khoảng trắng hoặc 1 tab) so với dòng cha. Các dòng của một khối lệnh phải viết thẳng hàng với nhau.

Điều kiện logic trong lệnh if

Điều kiện logic trong lệnh if về bản chất là một biểu thức, nhưng chỉ nhận một trong hai giá trị True/False . Điều kiện này thường được tạo ra từ các phép toán :

* Các phép so sánh :
* Lớn hơn : >
* Nhỏ hơn : <
* Lớn hơn hoặc bằng : >=
* Nhỏ hơn hoặc bằng : <=
* Bằng nhau : ==
* Khác nhau : !=
* Phép phủ định : not
* Phép và : and
* Phép hoặc : or

Một số ví dụ minh họa

Ví dụ 1:

Nhập vào từ bàn phím 2 số và in ra giá trị nhỏ nhất của 2 số đó

|  |
| --- |
| a = input('Số thứ nhất : ') a = float(a)  b = input('Số thứ hai : ') b = float(b)  **if** a < b:   print(a)  **else**:   print(b) |

Ví dụ 2:

Chỉ số BMI (Body mass index) dùng để đánh giá thân hình của một người. Chỉ số này được đo bằng tỉ số giữa cân nặng (tính theo kg) và bình phương của chiều cao (tính theo mét). Dựa vào chỉ số BMI, người ta có thể đánh giá được thân hình của một người:

* BMI nhỏ hơn 15 : thân hình quá gầy
* BMI từ 15 đến 16 : thân hình gầy
* BMI từ 16 đến 18.5 : thân hình hơi gầy
* BMI từ 18.5 đến 25 : thân hình bình thường
* BMI từ 25 đến 30 : thân hình hơi béo
* BMI từ 30 đến 35: thân hình béo
* BMI trên 35: thân hình quá béo

Nhập vào giá trị chiều cao và cân nặng của một người từ bàn phím và cho biết tình trạng thân hình của người đó.

|  |
| --- |
| height = input('Chiều cao (mét) : ') height = float(height)  mass = input('Cân nặng (kg) : ') mass = float(mass)  bmi = mass / (height \* height)  **if** bmi < 15:    print('Thân hình quá gầy')  **elif** bmi < 16:    print('Thân hình gầy')  **elif** bmi < 18.5:    print('Thân hình hơi gầy')  **elif** bmi < 25:    print('Thân hình bình thường')  **elif** bmi < 30:    print('Thân hình hơi béo')  **elif** bmi < 35:    print('Thân hình béo')  **else**:    print('Thân hình quá béo') |

Vòng lặp for

Cấu trúc vòng lặp for

Lệnh for được dùng để thực hiện một vòng lặp với số lần lặp biết trước. Cấu trúc lệnh for như sau:

for bienLap in tapGiaTri:

    <khoi\_lenh>

Cách biểu diễn tập giá trị lặp trong vòng lặp for

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

range(<end>)

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

Ví dụ:

range(5) → 0, 1, 2, 3, 4

* Sử dụng cấu trúc :

range(start, end)

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

Ví dụ:

range(1, 5) → 1, 2, 3, 4

* Sử dụng cấu trúc :

range(start, end, increment)

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

Ví dụ:

range(0, 10, 2) → 0, 2, 4, 6, 8

**Một số ví dụ minh họa**

Ví dụ 1:

Tính tổng các số từ 1 đến 100

|  |
| --- |
| S = 0 **for** i **in** range(1, 101):    S += i  print('S = ', S) |

Ví dụ 2:

Viết chương trình in ra bảng cửu chương

|  |
| --- |
| **for** i **in** range(2, 10):    **for** j **in** range(2, 10):        print(i, '\*', j, '=', i\*j)    print() |

Vòng lặp while

Cấu trúc vòng lặp while

Lệnh lặp while dùng để thực hiện một vòng lặp với số lần lặp không biết trước. Cấu trúc lệnh while như sau:

while dieukien:

    <khoi\_lenh>

Một số ví dụ minh họa

Ví dụ 1:

Kiểm tra một số có phải số nguyên tố không

|  |
| --- |
| x = int(input('x='))  i = 2 **while** i \* i <= x:    **if** x % i == 0:        j = x // i        print(x, '=', i , '\*', j)        exit()    i += 1  print(x, ' là số nguyên tố') |

Ví dụ 2:

Chuyển một số từ hệ thập phân sang hệ nhị phân

|  |
| --- |
| x = int(input('x='))  s = ''  **while** x > 0:    i = x % 2    s = str(i) + s    x = int(x/2)  print(s) |

Kiểu dữ liệu số

Trong Python có 2 kiểu số : số nguyên (int) và số thực (float)

Kiểu số nguyên (int)

Số nguyên là các giá trị số không chứa dấu phảy:

x = 1

y = 2

z = -10000

Các phép tính trên kiểu số nguyên:

* Phép cộng (+)
* Phép trừ (-)
* Phép nhân (\*)
* Phép chia (/)
* Phép chia lấy phần nguyên (//)
* Phép lấy phần dư (%)
* Phép lũy thừa (\*\*)

Với Python 3, phép chia 2 số nguyên cho kết quả là một số thập phân:

>>> print(4/3)

1.3333333333333333

Phép tính dạng rút gọn:

Các phép tính +=, -=, \*=, /=, //=, %=, \*\*=, … được sử dụng để tính giá trị phép tính trên 2 biến và gán kết quả vào biến thứ nhất:

* a += b  tương đương với  a = a + b
* a -= b tương đương với a = a - b
* ....

Phép lấy giá trị tuyệt đối

Hàm abs được dùng để lấy giá trị tuyệt đối một số:

>>> print(abs(-10))

10

Kiểu số thập phân(float)

Số thập phân là các số có chứa dấu phảy :

x = 1.0

y = 10.5

z = 9e38

Các phép tính trên số thập phân

Tương tự số nguyên, số thập phân cũng có các phép tính +, -, \*, //, /, %, \*\*, abs, các phép tính rút gọn. Ngoài ra trên số thập phân còn có các phép tính sau:

Phép làm tròn số

Hàm round được dùng để làm tròn một số thập phân:

>>> print(round(4.5))

4

Hàm này còn cho phép làm tròn tới một số lượng chữ số nhất định sau dấu phảy:

>>> print(round(4/3, 2)) # làm tròn đến 2 chữ số sau dấu phảy

1.33

Các hàm toán học

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

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

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

import math

Ví dụ:

|  |
| --- |
| **import** math print(math.sin(math.pi/2)) print(math.sqrt(4)) |

Kiểu dữ liệu string

Khai báo biến string

Trong Python, giá trị của kiểu dữ liệu String được đặt trong cặp 2 dấu nháy đơn (') hoặc 2 dấu nháu kép (") :

hoten = 'Nguyễn Văn An'

diachi = "Hà Nội"

Một số hàm xử lý trên dữ liệu String

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

Ví dụ:

>>> str(100)

'100'

* Phép cộng 2 giá trị string :

Ví dụ:

>>> 'Chào ' + 'bạn'

'Chào bạn'

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

Ví dụ:

>>> 'Chào bạn'.lower()

'chào bạn'

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

Ví dụ:

>>> 'Chào bạn'.upper()

'CHÀO BẠN'

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

Ví dụ:

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

'Tôi sống ở Huế'

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

Ví dụ:

>>> 'Hà Nội'.split()

['Hà', 'Nội']

>>> 'Hà Nội, Việt Nam'.split(',')

['Hà Nội', ' Việt Nam']

* Hàm **strip** : Bỏ đi các kí tự trống ở đầu và cuối string:

Ví dụ:

>>> '   Hà Nội    '.strip()

'Hà Nội'

* Hàm **len** : trả về số lượng ký tự của một giá trị string

Ví dụ:

>>> len('Chào bạn')

8

Truy nhập từng kí tự của string

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

c = text[index]

Bản thân một giá trị string có thể xem là một tập hợp kí tự và có thể dùng vòng lặp for chạy qua tập kí tự này

Ví dụ :

text = 'Chào bạn'

for c in text:

    print( c )

Một số phép tính trên kí tự:

* Hàm ord : lấy mã thứ tự của một kí tự trong bảng mã kí tự.

Ví dụ:

>>> ord('A')

65

* Hàm chr : chuyển ngược từ thứ tự của một kí tự trong bảng kí tự thành giá trị kí tự.

Ví dụ:

>>> chr(65)

'A'

Substring

Mỗi đoạn nhỏ của một giá trị string được gọi là substring. Bản thân mỗi substring cũng là một giá trị kiểu string.

Các cách để lấy ra substring trong Python:

* Cú pháp:

text[start:end]

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

Ví dụ:

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

Chào

* Cú pháp:

text[start:]

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

Ví dụ:

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

bạn

* Cú pháp:

text[:end]

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

Ví dụ:

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

Chào

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

Ví dụ:

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

bạn

Format string

Để ghép nhiều giá trị string thành một có thể dùng phép cộng string như đã trình bày ở phía trên:

|  |
| --- |
| x = 1 y = 2 z = 3 st = 'Tổng của ' + str(x) + ' và ' + str(y) + ' là ' + str(z) print(st) |

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

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

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

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

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

Một số ví dụ minh họa

Ví dụ 1:

Kiểm tra tính hợp lệ của mật khẩu (là một giá trị string). Mật khẩu hợp lệ nếu:

* Có ít nhất 6 kí tự
* Chứa ít nhất một chữ cái (a-z hoặc A-Z)
* Chứa ít nhất một chữ số (0-9)

|  |
| --- |
| password = input('Nhập mật khẩu : ')  **if** len(password) < 6:    print('Mật khẩu quá ngắn')    exit()  chuaChuCai = **False** **for** c **in** password:    C = c.upper()    **if** (C >= 'A' **and** C <= 'Z'):        chuaChuCai = **True**        **break**  **if** **not** chuaChuCai:    print('Mật khẩu cần chứa ít nhất một chữ cái (a-z/A-Z)')    exit()  chuaChuSo = **False** **for** c **in** password:    **if** c >= '0' **and** c <= '9':        chuaChuSo = **True**        **break**  **if** **not** chuaChuSo:    print('Mật khẩu cần chứa ít nhất một chữ số (0-9)')    exit()  print('Mật khẩu hợp lệ !') |

Ví dụ 2 :

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

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

Kiểu dữ liệu List và Tuple

Kiểu dữ liệu List

Kiểu dữ liệu List dùng để chứa một dãy nhiều phần tử. Khi khai báo giá trị, các phần từ của một List được đặt trong cặp dấu [ ].

Ví dụ:

dayso = [1, 3, 5, 7, 9]

danhsach\_hocsinh = ["Nguyễn Văn An", "Nguyễn Chí Cường", "Nguyễn Mạnh Tuấn"]

diachi = [322, "Tây Sơn", "Hà Nội"]

Các phần tử của một List có thể không cùng kiểu dữ liệu.

Một số phép tính trên kiểu dữ liệu List

* Hàm **len** : trả về số phần tử của một List

Ví dụ:

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

5

* Hàm **append** : Thêm mới một phần tử vào List

Ví dụ:

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

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

* Hàm **remove** : Xóa một phần tử khỏi List

Ví dụ:

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

[1, 2, 4, 5]

* Hàm **extend** : Ghép một List con vào cuối một List

Ví dụ:

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

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

* Phép cộng 2 List : trả về một List mới bằng cách ghép 2 List với nhau

Ví dụ:

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

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

* Hàm **reverse** : đảo ngược một List

Ví dụ:

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

[5, 4, 3, 2, 1]

* Phép toán **in** : Kiểm tra một phần tử có nằm trong List

Ví dụ:

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

True

* Phép toán **not in** : Kiểm tra một phần tử có không nằm trong List

Ví dụ:

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

False

**Truy nhập đến từng phần tử của List**

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

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

Một số ví dụ minh họa

Ví dụ 1:

In ra các số nguyên tố nhỏ hơn 1000

|  |
| --- |
| ds = []  **for** x **in** range(2, 1000):    nguyento = **True**     **for** p **in** ds:        **if** x % p == 0:             nguyento = **False**             **break**         **if** p \*p > x:             **break**         **if** nguyento:        ds.append(x)  print(ds) |

Ví dụ 2:

In ra 10 dòng đầu của tam giác Pascal (các hệ số của khai triển nhị thức (a+b)n)

(a+b)0 = 1                     → 1

(a+b)1 = a + b                → 1  1

(a+b)2 = a2+ 2ab + b2         → 1  2  1

(a+b)3 = a3+ 3a2b + 3ab2+ b3  → 1  3  3  1

…

|  |
| --- |
| N = 10 heso = []  **for** i **in** range(N):    heso.append(1)    **for** j **in** range(i-1, 0, -1):        heso[j] += heso[j-1]     print(heso) |

Kiểu dữ liệu Tuple

Dữ liệu kiểu bộ nhóm (Tuple) tương tự với kiểu danh sách (List), chỉ khác là các phần tử sau khi khai báo thì không sửa (thêm, xóa) được. Các phần tử của dữ liệu kiểu nhóm được đặt trong cặp dấu ( )

dayso = (1, 3, 5, 7, 9)

danhsach\_hocsinh = ("Nguyễn Văn An", "Nguyễn Chí Cường", "Nguyễn Mạnh Tuấn")

Các phép tính trên Tuple

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

* Hàm **len** : trả về số phần tử của một Tuple
* Phép cộng 2 Tuple : trả về một Tuple mới bằng cách ghép 2 Tuple với nhau
* Phép toán **in** : Kiểm tra một phần tử có nằm trong Tuple
* Phép toán **not in** : Kiểm tra một phần tử có không nằm trong Tuple
* Truy nhập phần tử qua index : tương tự như List

Kiểu dữ liệu Set

Kiểu dữ liệu Set dùng để chứa các phần tử của một tập hợp. So với kiểu List, kiểu Set có điểm khác là:

* Trong một Set không có 2 phần tử cùng giá trị.
* Không thể dùng index để truy nhập phần tử của một Set

Các phép tính trên Set

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

Ví dụ:

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

{1, 2, 3}

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

Ví dụ:

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

{1, 3}

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

Ví dụ:

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

True

* Phép **not in** : kiểm tra giá trị có không nằm trong tập hợp

Ví dụ:

>>> s = set([1, 2, 3]); print(3 not in s)

False

* Hàm **union** : trả về hợp của 2 Set

Ví dụ:

>>> s1 = set([1, 2, 3]); s2 = set([3, 4, 5]); print(s1.union(s2))

{1, 2, 3, 4, 5}

* Hàm **intersection** : giao của 2 Set

Ví dụ:

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

{3}

* Hàm **difference** : hiệu của 2 Set

Ví dụ:

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

{1, 2}

Một số ví dụ minh họa

Ví dụ 1:

|  |
| --- |
| s1 = set([1, 2]) s1.add(3) s1.remove(1)  **if** 1 **not** **in** s1:    print("1 không nằm trong tập hợp ", s1)  **if** 2 **in** s1:   print("2 nằm trong tập hợp ", s1)  s2 = set([3, 4]) s3 = set.union(s1, s2) print(s3)  s4 = set.intersection(s1, s2) print(s4) |

Ví dụ 2:

|  |
| --- |
| weekdays = set(['Thứ hai', 'Thứ ba', 'Thứ tư', 'Thứ năm', 'Thứ sáu']) weekends = set(['Thứ bảy', 'Chủ nhật'])  day = "Thứ ba"  **if** day **in** weekdays:   print(day, ' là ngày làm việc')  **if** day **in** weekends:   print(day, ' là ngày nghỉ cuối tuần') |

Kiểu dữ liệu Dictionary

Kiểu dữ liệu Dictionary dùng để chứa một bảng biến đổi 1-1 giữa 2 tập hợp :

* Tập nguồn (hay tập khóa (key)) : chứa các thông tin cần tra cứu
* Tập đích (hay tập giá trị (value)) : chứa thông tin của mỗi key từ tập nguồn

Khai báo kiểu dữ liệu Dictionary

Cú pháp khai báo kiểu Dictionary trong Python:

d = {

    <key1> : <value1>,

    <key2> : <value2>,

    ...

}

Ví dụ:

d = {"một" : 1, "hai" : 2, "ba" : 3}

Truy nhập phần tử của Dictionary

Để truy nhập đến phần tử có key là <key> trong Dictionary, Python sử dụng phép toán []:

d[<key>]

Ví dụ:

>>> d = {"một" : 1, "hai" : 2, "ba" : 3}; print(d['một'])

1

Trong trường hợp nếu giá trị <key> không nằm trong tập nguồn của Dictionary, Python sẽ đưa ra thông báo Exception:

>>> d = {"một" : 1, "hai" : 2, "ba" : 3}; print(d['bốn'])

Traceback (most recent call last):

  File "<pyshell#118>", line 1, in <module>

    d = {"một" : 1, "hai" : 2, "ba" : 3}; print(d['bốn'])

KeyError: 'bốn'

Để tránh gặp lỗi này, có thể dùng hàm **get** của Dictionary để truyền giá trị mặc định trong trường hợp <key> không nằm trong tập nguồn của Dictionary:

d.get(<key>, <gia\_tri\_mac\_dinh>)

Ví dụ:

>>> d = {"một" : 1, "hai" : 2, "ba" : 3}; print(d.get('bốn', -1))

-1

Một số ví dụ minh họa

Ví dụ 1:

Đếm tần suất của các từ trong một câu

|  |
| --- |
| text = 'Một năm có mười hai tháng, tháng hai có hai mươi tám ngày, các tháng còn lại có ba mươi hoặc ba mươi mốt ngày'  text = text.lower() **for** c **in** ['.', ',' , ':']:    text = text.replace(c, ' ')  words\_count = {}  **for** word **in** text.split():    words\_count[word] = words\_count.get(word, 0) + 1  **for** word **in** words\_count:    print(word , ' : ', words\_count[word]) |

Ví dụ 2:

Chuyển phát âm tiếng Việt của một số trong phạm vi 1-99 thành giá trị số đó. Ví dụ : tám mươi lăm → 85

|  |
| --- |
| bang\_so1 = {'một' : 1, 'hai' : 2, 'ba' : 3, 'bốn' : 4, 'năm' : 5, 'sáu' : 6, 'bảy' : 7, 'tám' : 8, 'chín' : 9, 'mười' : 10}  bang\_so2 = {'một' : 1, 'hai' : 2, 'ba' : 3, 'bốn' : 4, 'lăm' : 5, 'sáu' : 6, 'bảy' : 7, 'tám' : 8, 'chín' : 9}  bang\_so3 = {'mươi' : 0, 'mốt' : 1, 'hai' : 2, 'ba' : 3, 'bốn' : 4, 'tư' : 4, 'lăm' : 5, 'sáu' : 6, 'bảy' : 7, 'tám' : 8,'chín' : 9}  text = input('text=') words = text.lower().split() N = len(words)  **if** N == 1:    **if** words[0] **in** bang\_so1:        print(bang\_so1[words[0]])            **else**:        print('Không tồn tại số')            exit()  chuc, donvi = -1, -1  **if** (N == 3 **and** words[1] == 'mươi') **or** N == 2:    chuc = bang\_so1.get(words[0], -1)    donvi = bang\_so3.get(words[-1], -1)  **if** N == 2 **and** words[0] == 'mười':    chuc = 1    donvi = bang\_so2.get(words[1], -1)  **if** chuc >= 0 **and** donvi >= 0:    print(10 \* chuc + donvi) **else**:    print('Không tồn tại số') |

Hàm (Function)

Khai báo hàm

Cấu trúc để khai báo hàm trong Python:

def tenham(<danh\_sach\_bien>):

    <noi\_dung\_ham>

    return <ket\_qua>

Ví dụ:

|  |
| --- |
| **def** **square**(x):   **return** x\*x  **for** i **in** range(1,11):   print(f'{i} \* {i} = {square(i)}') |

Hàm trả về nhiều giá trị

Một hàm có thể trả về nhiều giá trị, danh sách các giá trị trả về ở được ngăn cách nhau bởi dấu phảy

Ví dụ:

|  |
| --- |
| **def** **calcAreaAndPerimeter**(width, height):    S = width \* height    P = 2\*(width + height)    **return** S, P  A, P = calcAreaAndPerimeter(5, 4) print('A = ', A) print('P = ', P) |

Trong trường hợp này, có thể hiểu kết quả trả về là một giá trị kiểu Tuple

Giá trị mặc định của tham số trong hàm

Khi khai báo hàm, có thể  khai báo giá trị mặc định cho các tham số. Khi gọi hàm, nếu một tham số bị thiếu nhưng đã được khai báo giá trị mặc định trong khai báo hàm thì tham số đó sẽ được truyền vào giá trị mặc định.

Ví dụ:

|  |
| --- |
| **def** **getAddress**(street, city, country='Việt Nam'):    **return** f'{street}, {city}, {country}'          print(getAddress('Ba Đình', 'Hà Nội')) |

Truy nhập File

Mở và đóng file

Mở file

Lệnh mở file trong Python:

f = open(<file\_name>, <mode>, encoding=<encoding>)

Trong đó :

* <file\_name> : tên file cần mở
* <mode> : Chế độ mở file, các giá trị thường dùng:
* 'r' : Mở file để đọc, đây là giá trị mặc định
* 'rb' : Mở file để đọc dưới dạng nhị phân (binary)
* 'w' : Tạo file mới để ghi
* 'wb' : Tạo file mới để ghi dưới dạng nhị phân (binary)
* 'a' :  Mở file để ghi thêm vào cuối (append)
* 'ab' : Mở file để ghi thêm vào cuối dưới dạng nhị phân
* <encoding>: Chế độ encoding với file văn bản
* None : giá trị mặc định, tùy thuộc từng hệ thống
* 'utf8' : mã hóa Unicode dạng UTF-8

Đóng file

f.close()

Đọc và ghi file

Đọc file

* Đọc toàn bộ nội dung file:

data = f.read()

Nếu file được mở theo chế độ văn bản ('r'), kết quả đọc file là một biến string. Nếu file được mở theo chế độ nhị phân ('rb'), kết quả đọc file là một biến có kiểu dữ liệu bytes.

* Đọc toàn bộ các dòng của file (với file văn bản):

Nếu mở file theo chế độ văn bản ('r') có thể đọc nội dung file theo từng dòng với lệnh :

lines = f.readlines()

Kết quả trả về là một List, mỗi phần tử là một string chứa nội dung các dòng của file

* Đọc từng dòng của file:

Nếu mở file theo chế độ văn bản ('r') có thể đọc từng dòng của file với cấu trúc:

for line in f:

    # process line

Ghi file:

* Với file văn bản, dữ liệu ghi vào file cần có kiếu string

f.write(<string>)

Ví dụ:

f = open('test.txt', 'w')

f.write('hello')

f.close()

* Với file nhị phân, dữ liệu ghi vào file cần có kiểu bytes

f.write(<bytes>)

Ví dụ:

f = open('test.dat', 'wb')

f.write('hello'.encode())

f.close()

Truy xuất file với cấu trúc 'with'

Sau khi mở file với lệnh open, phải đóng file với lệnh close để tránh bị thiếu nội dung file khi kết thúc chương trình. Để tránh việc quên đóng file sau khi mở, có thể dùng cấu trúc **with**:

with open(<file\_name>, <mode>, <encoding>) as f:

    # Process file

Với cấu trúc **with**, file sẽ tự động đươc đóng khi chương trình chạy xong khối lệnh bên trong **with**.

Ví dụ:

|  |
| --- |
| **with** open('test.txt', 'w') **as** f:    f.write('Line1\n')    f.write('Line2\n') |

**Một số ví dụ minh họa**

Ví dụ 1:

Đọc vào một file văn bản, tạo ra một file văn bản mới chứa các dòng của file nguồn, bỏ đi các dòng trống.

File input.txt:

Một năm có 365 hoặc 366 ngày

Năm thường có 365 ngày

Năm nhuận có 366 ngày

File output.txt:

Một năm có 365 hoặc 366 ngày

Năm thường có 365 ngày

Năm nhuận có 366 ngày

|  |
| --- |
| infile = open('input.txt', encoding='utf-8') outfile = open('output.txt', 'w', encoding='utf-8')  **for** line **in** infile:    **if** line.strip() != '':        outfile.write(line)  infile.close() outfile.close() |

Ví dụ 2:

Một file csv chứa bảng điểm một môn học của các học sinh một lớp học. Mỗi dòng của file là thông tin điểm của một học sinh bao gồm : họ tên, điểm hệ số 1, điểm hệ số 2, điểm hệ số 3, các giá trị này ngăn cách nhau bởi dấu phảy. Viết chương trình để đọc file csv đầu vào và tạo ra một file csv mới có thêm cột điểm trung bình.

File input.csv

Nguyễn Văn An, 8, 7, 8  
Nguyễn Văn Bình, 6, 6, 8  
Nguyễn Thị Chi, 8, 8, 9  
Lê Văn Cường, 8, 7, 9  
Phạm Thu Trang, 7, 8, 8

File output.csv

Nguyễn Văn An, 8, 7, 8,  7.7

Nguyễn Văn Bình, 6, 6, 8,  7.0

Nguyễn Thị Chi, 8, 8, 9,  8.5

Lê Văn Cường, 8, 7, 9,  8.2

Phạm Thu Trang, 7, 8, 8,  7.8

|  |
| --- |
| fi = open('input.csv', encoding='utf-8') fo = open('output.csv', 'w', encoding='utf-8')  **for** line **in** fi:    hoten, diemhs1, diemhs2, diemhs3 = line.split(',')    diemhs1 = int(diemhs1)    diemhs2 = int(diemhs2)    diemhs3 = int(diemhs3)    diem\_tb = (diemhs1 + 2\*diemhs2 + 3\*diemhs3) / 6    fo.write(f'{hoten}, {diemhs1}, {diemhs2}, {diemhs3}, {round(diem\_tb,1)}\n')  fi.close() fo.close() |

**Lập trình hướng đối tượng với Class**

Khai báo Class trong Python

Trong Python, một Class thường có cấu trúc như sau:

class ClassName:

    # Hàm khởi tạo đối tượng

    def \_\_init\_\_(self, var1, var2, ...):

        self.var1 = var1

        self.var2 = var2

        ....

    # Các method

    def method1(self, ...)

        # Nội dung hàm

    def method2(self, ...)

        # Nội dung hàm

Ví dụ:

class Person:

    def \_\_init\_\_(self, name, address):

        self.name = name

        self.address = address

    def sayHello(self):

        print(f'Xin chào, tên tôi là {self.name}')

    def updateAddress(self, newAddress):

        self.address = newAddress

Khởi tạo và sử dụng đối tượng

Cú pháp để khởi tạo một đối tượng của một Class:

obj = ClassName(var1, var2, …)

Để truy nhập đến phương thức của đối tượng:

obj.method1(...)

obj.method2(...)

Để truy nhập đến thuộc tính đối tượng:

obj.attr1

Ví dụ:

person = Person('Nguyễn Văn An', 'Hà Nội')

person.sayHello()

person.updateAddress('TP. HCM')

print(f'{person.name} sống tại {person.address}')

Thừa kế

Để khai báo một class mới thừa kế từ một class đã có:

class ChildName(ParentClass):

     ....

Lớp con sẽ kế thừa các thuộc tính và phương thức của lớp cha.

Ngoài ra lớp con có thể “ghi đè” phương thức đã có ở lớp cha bằng cách khai báo một phương thức cùng tên.

Ví dụ:

|  |
| --- |
| **class** **Person**:    **def** **\_\_init\_\_**(self, name, address):        self.name = name        self.address = address            **def** **sayHello**(self):        print(f'Xin chào, tên tôi là {self.name}')                **def** **updateAddress**(self, newAddress):        self.address = newAddress  **class** **Student**(Person):    **def** **\_\_init\_\_**(self, name, address, schoolYear):        self.name = name        self.address = address        self.schoolYear = schoolYear            **def** **sayHello**(self):        print(f'Xin chào, tên tôi là {self.name}, học khóa {self.schoolYear}')     student = Student('Nguyễn Văn An', 'Hà Nội', 2019) student.sayHello() |

Xử lý đa luồng

Thread

Thread được sử dụng để tạo ra một luồng xử lý song song với luồng xử lý chính của chương trình.

Ví dụ:

|  |
| --- |
| **import** time **from** threading **import** Thread  **def** **threadFunc**():    time.sleep(2)    print('Inside thread')          thr = Thread(target=threadFunc) thr.start()  time.sleep(5) print('Inside main program') |

Chương trình trên tạo ra một thread chạy song song với chương trình chính. Trong chương trình chính, khi gặp lệnh **time.sleep(5)**,  chương trình sẽ  ngừng trong 5 giây mà không thực hiện hoạt động nào. Tuy nhiên, bên trong thread các hoạt động vẫn được thực hiện, do đó khi chạy chương trình chúng ta vẫn thấy lệnh **print** bên trong thread in dữ liệu ra màn hình trong khoảng 5 giây ngừng hoạt động của chương trình chính .

Có thể tạo nhiều thread một lúc, khi đó các thread sẽ chạy song song với nhau. Ví dụ:

|  |
| --- |
| **from** threading **import** Thread  **def** **threadFunc**(threadId):    print(f'Inside thread {threadId}')     **for** i **in** range(5):    thr = Thread(target=threadFunc, args=(i,))    thr.start() |

Đồng bộ giữa các thread : Khi các thread chạy song song nhau, trình tự thực hiện các đoạn chương trình bên trong các thread sẽ không phụ thuộc lẫn nhau. Trong trường hợp cần đợi cho một thread thực hiện xong trước khi bắt đầu thực hiện các công việc khác, có thể sử dụng lệnh **thread.join()**như trong ví dụ sau:

|  |
| --- |
| **import** time **from** threading **import** Thread  **def** **threadFunc**():    time.sleep(2)    print('Inside thread')          thr = Thread(target=threadFunc) thr.start()  time.sleep(1) thr.join()      # đợi thread thực hiện xong  print('Thread completed.') print('In main program') |

Nhược điểm của thread:

* Mặc dù các thread chạy song song với nhau nhưng tại một thời điểm chỉ một thread được chạy. Nói cách khác, dù nhiều thread được tạo ra, chương trình chỉ sử dụng một core CPU để chạy các thread theo trình tự luân phiên nhau. Nguyên nhân của hiện tượng này là do Python sử dụng **GIL** (**Global interface lock**) để tránh hiện tượng nhiều thread cùng làm thay đổi một đối tượng tại một thời điểm dẫn đến hiện tượng dữ liệu bị phá hỏng. Bên trong thread, những đoạn nào không sử dụng GIL thì có thể thực hiện đồng thời trên nhiều core CPU, những đoạn nào sử dụng GIL thì sẽ luân phiên nhau thực hiện trên một core CPU.

Các đoạn code Python thông thường đều sử dụng GIL. Chỉ một số hàm của các thư viện của Python không sử dụng GIL như các hàm truy nhập IO (bàn phìm, File, Download, Upload qua internet …). Do đó về cơ bản, việc sử dụng nhiều Thread không giúp tăng tốc độ xử lý chương trình.

Process

Process cũng giống như Thread cho phép xử lý các tác vụ song song với nhau, nhưng cho phép các công việc được thực hiện đồng thời trên nhiều core CPU một lúc.

Ví dụ:

|  |
| --- |
| **from** multiprocessing **import** Process  **def** **processFunc**(pid):        s = 0    **for** i **in** range(10000000):        s += 1            print(f'Process id : {pid}, result : {s}')  **if** \_\_name\_\_ == '\_\_main\_\_':    processes = []        **for** i **in** range(5):        p = Process(target=processFunc, args=(i,))        processes.append(p)        p.start()            **for** p **in** processes:        p.join()        print('All process completed') |

Nếu so với cách sử dụng Thread, đoạn chương trình trên sẽ thực hiện nhanh hơn vì các tác vụ tính toán được thực hiện trên nhiều core CPU cùng lúc.

Nhược điểm của Process:

* So với Thread, khi sử dụng process thì các tác vụ được phân phối đều trên các core CPU. Tuy nhiên nhược điểm của Process là không thể truy nhập đến một đối tượng chung của chương trình. Khi Process được tạo ra, nó clone toàn bộ các biến của chương trình chính, do đó bên trong mỗi process, các biến có tên giống với biến của chương trình chính thực chất là các biến mới.

Ví dụ sau minh họa sự khác nhau giữa Thread và Process :

* Sử dụng Thread:

|  |
| --- |
| **from** threading **import** Thread  a = [0] \* 5  **def** **threadFunc**(id):    s = 0    **for** i **in** range(1000):        s += 1            a[id] = s  **if** \_\_name\_\_ == '\_\_main\_\_':    threads = []        **for** i **in** range(5):        t = Thread(target=threadFunc, args=(i,))        threads.append(t)        t.start()            **for** thr **in** threads:        thr.join()            print(a) |

Chương trình khi chạy sẽ in ra kết quả là  [1000, 1000, 1000, 1000, 1000], đúng như mong đợi

* Sử dụng process:

|  |
| --- |
| **from** multiprocessing **import** Process  a = [0] \* 5  **def** **processFunc**(id):        s = 0    **for** i **in** range(1000):        s += 1            a[id] = s  **if** \_\_name\_\_ == '\_\_main\_\_':    processes = []        **for** i **in** range(5):        p = Process(target=processFunc, args=(i,))        processes.append(p)        p.start()            **for** p **in** processes:        p.join()        print(a) |

Chương trình khi chạy sẽ in ra kết quả là [0, 0, 0, 0, 0]. Nguyên nhân là do biến **a** ở trong process là một biến mới, không phải biến **a** khai báo ở đầu chương trình.

Pool

Pool cũng giống như Process, tức cho phép các tác vụ xử lý được thực hiện trên nhiều core CPU. Pool thường được sử dụng thay cho Process khi cần tập hợp kết quả các Process với nhau vào một biến chung của chương trình chính (điều mà Process không làm được như ở ví dụ minh họa phía trên)

Ví dụ:

|  |
| --- |
| **from** multiprocessing **import** Pool  **def** **poolFunc**(id):    print('Process id : ', id)    s = 0    **for** i **in** range(1000):        s += 1            **return** s     **if** \_\_name\_\_ == '\_\_main\_\_':    p = Pool(5)    a = p.map(poolFunc, list(range(5)))    print(a) |

Lập trình với một số thư viện thông dụng của Python

Lấy tham số từ dòng lệnh

Để lấy tham số từ dòng lệnh, có thể sử dụng thư viện **sys**. Với một chương trình, danh sách tham số truyền vào dòng lệnh được lấy từ biến:

sys.argv : ~  một list string, mỗi phần tử là một tham số dòng lệnh

Trong đó sys.argv[0] luôn là tên file .py được chạy, sys.argv[1] trở đi là các tham số truyền vào chương trình.

Ví dụ:

|  |
| --- |
| # test\_args.py  **import** sys  **for** arg **in** sys.argv:    print(arg) |

Khi chạy chương trình trên, các tham số từ dòng lệnh sẽ được in ra màn hình, ví dụ:

> python test\_args.py 1 2 3 abc

test\_args.py

1

2

3

abc

Trong trường hợp muốn đặt tên tường minh cho các tham số từ dòng lệnh, có thể sử dụng thư viện **argparse**:

|  |
| --- |
| # test\_args.py  **import** argparse  parser = argparse.ArgumentParser() parser.add\_argument('--ip', '-i', default='127.0.0.1') parser.add\_argument('--port', '-p', type=int, default=8080)  args = parser.parse\_args() print(f'{args.ip}:{args.port}') |

Khi chạy chương trình, để truyền tham số từ dòng lệnh, có thể sử dụng các lệnh

python test\_args.py              # Các tham số trong chương trình sẽ lấy mặc định

python test\_args.py -p 8000                             # Truyền vào tham số port

python test\_args.py --ip 192.168.100.1                  # Truyền vào tham số ip

python test\_args.py --port 8000 -i 192.168.1.100        # Truyền vào cả 2 tham số

python test\_args.py -p 8000 -i 192.168.1.100            # Truyền vào cả 2 tham số

Thao tác với file và thư mục

Lấy đường dẫn thư mục làm việc hiện tại:

|  |
| --- |
| **import** os print(os.getcwd()) |

Thay đổi thư mục làm việc:

|  |
| --- |
| **import** os os.chdir('C:\\Python37') |

Tạo mới thư mục:

|  |
| --- |
| **import** os os.mkdir('new\_dir') |

Đổi tên thư mục:

|  |
| --- |
| **import** os os.rename('dir','new\_dir') |

Xóa thư mục:

|  |
| --- |
| **import** os **import** shutil  path = 'C:\\dir'  **if** os.path.exists(path):    shutil.rmtree(path) |

Lấy đường dẫn tuyệt đối của một file trong thư mục:

|  |
| --- |
| **import** os  dir\_name = 'C:\\' file\_name = 'test.txt' file\_path = os.path.join(dir\_name, file\_name)  print(file\_path) |

Lấy tên file từ đường dẫn tuyệt đối:

|  |
| --- |
| **import** os  file\_path = 'C:\\test.txt' file\_name = os.path.basename(file\_path)  print(file\_name) |

Lấy tên cơ bản và tên mở rộng từ tên file:

|  |
| --- |
| **import** os  file\_name ='test.txt' name, ext\_name = os.path.splitext(file\_name)  print(name, ext\_name) |

Kiểm tra file có tồn tại:

|  |
| --- |
| **import** os  file\_path = 'C:\\test.txt'  **if** os.path.exists(file\_path):    print(f'File {file\_path} có tồn tại') **else**:    print(f'File {file\_path} không tồn tại') |

Xóa file:

|  |
| --- |
| **import** os  file\_path = 'C:\\test.txt'  **if** os.path.exists(file\_path):    os.remove(file\_path) |

Copy file:

|  |
| --- |
| **from** shutil **import** copyfile  copyfile('src\_file.ext', 'dst\_file.ext') |

Lấy danh sách các file và thư mục con trong một thư mục:

|  |
| --- |
| **import** os  dir = '.' items = os.listdir(dir) files = [f **for** f **in** items **if** os.path.isfile(os.path.join(dir, f))] subdirs = [d **for** d **in** items **if** os.path.isdir(os.path.join(dir, d))] |

Duyệt cây thư mục:

|  |
| --- |
| **import** os  dir = '.'  **for** root, dirs, files **in** os.walk(dir):    print('Thư mục : ', os.path.join(dir, root))        print(' + Danh sách file : ')        **for** name **in** files:        print(os.path.join(dir, root, name))            print(' + Danh sách thư mục con : ')    **for** name **in** dirs:        print(os.path.join(dir, root, name)) |

Làm việc với ngày/giờ và thời gian

Lấy timestamp hiện tại (số giây từ 1/1/1970 đến hiện tại)

|  |
| --- |
| **import** time  timestamp = time.time()  print(timestamp) |

Lấy thời gian hiện tại

|  |
| --- |
| **from** datetime **import** datetime  now =datetime.now()  print(now) print(now.year, now.month, now.day, now.hour, now.minute, now.second, now.microsecond)  print(now.timestamp()) |

Lấy ngày hiện tại :

|  |
| --- |
| **from** datetime **import** date  today = date.today()   print(today) print(today.year, today.month, today.day, today.weekday()) |

Tạo thời gian từ các thành phần ngày/tháng/năm/giờ/phút/giây:

|  |
| --- |
| **from** datetime **import** datetime  dt = datetime(2019,1,1,23,59,59) print(dt.year, dt.month, dt.day, dt.hour, dt.minute, dt.second) |

Tạo ngày từ các thành phần ngày/tháng/năm:

|  |
| --- |
| **from** datetime **import** date  dt = date(2019,1,1) print(dt.year, dt.month, dt.day) |

Chuyển đổi từ thời gian sang string:

|  |
| --- |
| **from** datetime **import** datetime  now = datetime.now()  print(now.strftime('%d-%m-%Y')) print(now.strftime('%d/%m/%Y %H:%M:%S')) |

Chuyển đổi string sang thời gian:

|  |
| --- |
| **from** datetime **import** datetime  dt1 = datetime.strptime('01/01/2019', '%d/%m/%Y') print(dt1)  dt2 = datetime.strptime('01-01-2019 23:59:59', '%d-%m-%Y %H:%M:%S') print(dt2) |

Khoảng thời gian

Tính khoảng thời gian giữa 2 thời điểm :

|  |
| --- |
| **from** datetime **import** datetime  dt1 = datetime(2019, 2, 1) dt2 = datetime(2019, 2, 28, 23, 59, 59)  duration = dt2 - dt1  days = duration.days hours = duration.seconds // 3600 minutes = (duration.seconds %  3600) // 60 seconds = duration.seconds % 60  print(f'Duration : {days} days, {hours} hours,  {minutes} minutes, {seconds} seconds') |

Cộng một mốc thời điểm với một khoảng thời gian :

|  |
| --- |
| **from** datetime **import** datetime, timedelta  dt1 = datetime(2019, 2, 1) duration = timedelta(days=30, seconds=3600) dt2 = dt1 + duration  print(dt2) |

Chạy chương trình của hệ thống

Chạy một chương trình của hệ thống :

|  |
| --- |
| **import** os os.system('calc.exe') |

Chạy một chương trình hệ thống và lấy kết quả của chương trình sau khi kết thúc:

|  |
| --- |
| **import** subprocess output = subprocess.getoutput("dir") print(output) |

Khởi động một chương trình và theo dõi trạng thái cho đến khi chương trình thực hiện xong :

|  |
| --- |
| **import** subprocess **import** time  proc = subprocess.Popen('ping 127.0.0.1', stdout=subprocess.PIPE)   **while** **True**:    time.sleep(1)    poll = proc.poll()            **if** poll == **None**:            # program is running        **while** **True**:            line = proc.stdout.readline().decode()                        **if** line:                print(line)            **else**:                **break**    **else**:                       # program completed        print('Subprocess completed.')        **break** |

Thao tác với Json

Thư viện json cho phép chuyển đổi giữa json string và object của python. Hai hàm thường sử dụng là:

* json.loads : chuyển json string thành object
* json.dumps : chuyển object thành json string

Ví dụ:

|  |
| --- |
| **import** json  students = [    { 'name' : 'Nguyễn Văn An', 'address' : 'Hà Nội'},    { 'name' : 'Nguyễn Thị Bình', 'address' : 'TP.HCM'}, ] print(json.dumps(students))  jsonstring = '{"name" : "Nguyễn Văn An", "address" : "Hà Nội"}' student = json.loads(jsonstring) print(student['name'], student['address']) |

Thao tác với Xml

Chuyển từ xml sang object của python

|  |
| --- |
| **import** xml.etree.ElementTree **as** ET  xml = """    <students>        <student>            <name>Nguyễn Văn An</name>            <address>Hà Nội</address>        </student>        <student>            <name>Nguyễn Thị Bình</name>            <address>TP.HCM</address>        </student>    </students> """  root = ET.fromstring(xml) **for** student **in** root:        name = student.find('name').text    address = student.find('address').text    print(name, address) |

Chuyển object của Python sang xml

|  |
| --- |
| **import** xml.etree.cElementTree **as** ET  students = [    { 'name' : 'Nguyễn Văn An', 'address' : 'Hà Nội'},    { 'name' : 'Nguyễn Thị Bình', 'address' : 'TP.HCM'}, ]   root = ET.Element("students")  **for** student **in** students:    tag = ET.SubElement(root, "student")    ET.SubElement(tag, "name").text = student['name']    ET.SubElement(tag, "address").text = student['address']  tree = ET.ElementTree(root) tree.write("output.xml", encoding="UTF-8") |

Base64 encode và decode

Encode base64:

|  |
| --- |
| **import** base64  msg = 'Xin chào' msg\_bytes = msg.encode('utf-8') b64\_msg\_bytes = base64.b64encode(msg\_bytes) b64\_msg = b64\_msg\_bytes.decode()  print(b64\_msg) |

Decode base64:

|  |
| --- |
| **import** base64  msg = 'WGluIGNow6Bv' msg\_bytes = msg.encode() decoded\_msg\_bytes = base64.b64decode(msg\_bytes) decoded\_msg = decoded\_msg\_bytes.decode('utf-8')  print(decoded\_msg) |

**TCP/IP Socket**

Thư viện socket cho phép xây dựng các chương trình kết nối theo mô hình Client/Server qua Socket

Server:

|  |
| --- |
| # server.py  **import** socket  HOST = '127.0.0.1' PORT = 8080  s = socket.socket() s.bind((HOST, PORT)) s.listen()  print(f'Listen at {HOST}:{PORT}')  **while** **True**:    conn, addr = s.accept()    print('Connected from ', addr)    data = conn.recv(4096)    clientID = data.decode()    conn.send(f'Hello {clientID}'.encode())    conn.close() |

Client :

|  |
| --- |
| # client.py  **import** socket  HOST = '127.0.0.1' PORT = 8080 CLIENT\_ID = 'Client1'  s = socket.socket()  s.connect((HOST, PORT)) s.send(CLIENT\_ID.encode()) data = s.recv(4096) print(data.decode()) s.close() |

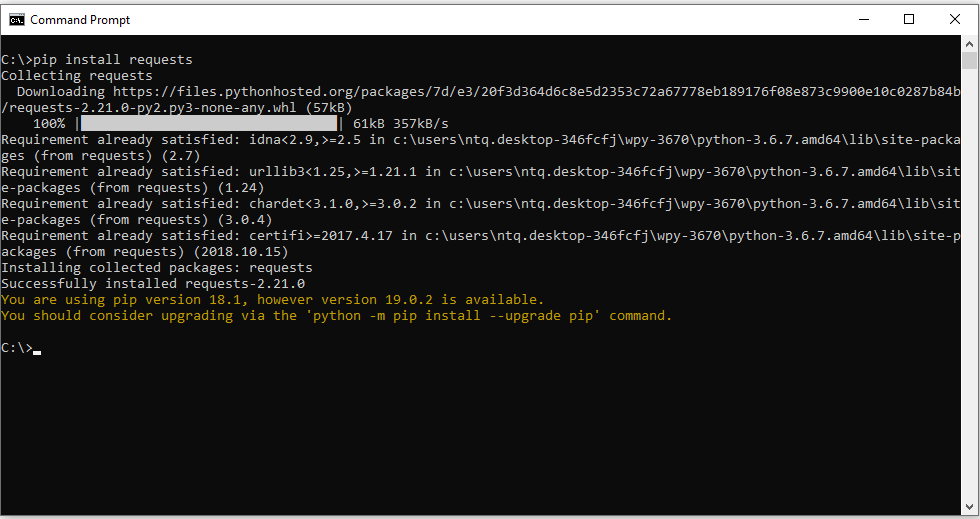
Gọi API tới http server qua GET/POST

Để thực hiện GET/POST tới http server, cách đơn giản nhất là sử dụng thư viện request.

Cài đặt thư viện:

Sử dụng pip để cài đặt thư viện, mở cửa sổ cmd và gõ lệnh:

pip install requests



Cài đặt thư viện requests bằng pip từ cửa sổ cmd

Sử dụng 'requests' để gọi API của remote server:

Để dễ hình dung, chúng ta cài đặt flask và tạo một ứng dụng server đơn giản để test.

Cài đặt flask:

pip install flask

Tạo ứng dụng server đơn giản để test:

|  |
| --- |
| # server.py  **from** flask **import** Flask, request, jsonify app = Flask(\_\_name\_\_)  @app.route('/test\_get') **def** **getMethod**():        **return** 'Hello world!'  @app.route('/test\_post', methods = ['POST']) **def** **postMethod**():    name = request.json['name']        **return** jsonify({'message' : f'Hello {name}!'})   **if** \_\_name\_\_ == '\_\_main\_\_':   app.run(debug = **True**, port=8080) |

Sau khi khởi động chương trình server.py, server sẽ lắng nghe ở cổng 8080

Chương trình client sử dụng thư viện requests để gọi API server như sau:

|  |
| --- |
| # client.py  **import** json **import** requests  # test GET method result = requests.get('http://localhost:8080/test\_get').text print(result)  # test POST method data = {'name' : 'new world'} result = requests.post(url='http://localhost:8080/test\_post',                        data=json.dumps(data),                        headers={'content-type' : 'application/json'}).text                                  result = json.loads(result) print(result['message']) |

**Kết nối CSDL MySQL**

Thư viện MySQLdb cho phép kết nối với cơ sở dữ liệu MySQL

Cài đặt thư viện :

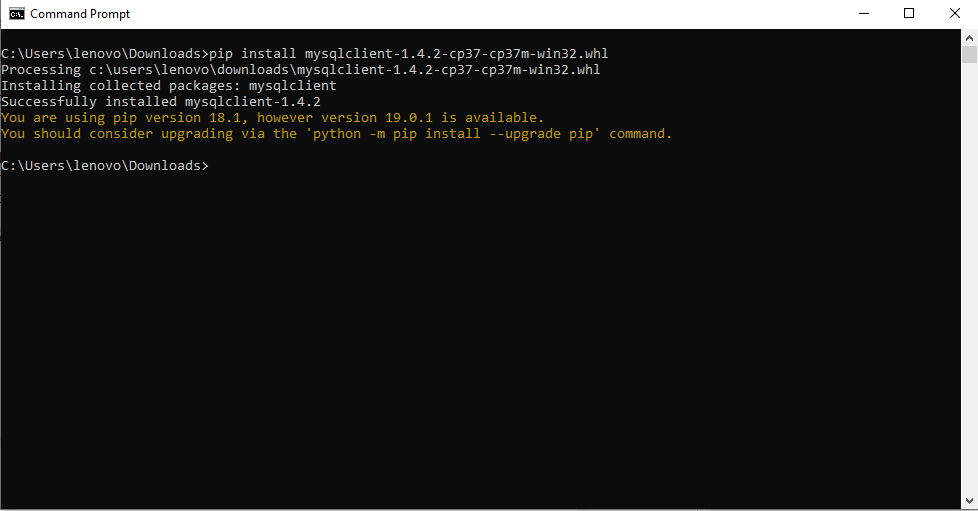
* Download thư viện tại : [https://www.lfd.uci.edu/~gohlke/pythonlibs/#mysqlclient](https://www.google.com/url?q=https://www.lfd.uci.edu/~gohlke/pythonlibs/%23mysqlclient&sa=D&ust=1550420047840000)

Lựa chọn theo đúng phiên bản Python và kiến trúc cpu (32 bit/64bit) của phiên bản Python đã cài đặt.

* Sử dụng pip để cài đặt thư viện:

Mở cửa sổ cmd, di chuyển vào trong thư mục chứa file thư viện đã download, gõ lệnh sau để cài đặt thư viện:

pip install <tên file thư viện đã download>



Sử dụng pip để cài đặt thư viện mysqlclient

Sử dụng thư viện để kết nối với database:

|  |
| --- |
| **import** MySQLdb  db = MySQLdb.connect("localhost", "admin", "1234", "testdb" )  print('Connected to database') |

Các tham số cần truyền vào để kết nối đến database theo thứ tự là : host, username, password, dbname

Tạo mới bảng trong database

|  |
| --- |
| **import** MySQLdb  db = MySQLdb.connect("localhost", "admin", "1234", "testdb" )  cursor = db.cursor()  cursor.execute("DROP TABLE IF EXISTS STUDENT")  cursor.execute("""CREATE TABLE STUDENT (         id INT NOT NULL AUTO\_INCREMENT,         name  VARCHAR(50),         address  VARCHAR(100),         PRIMARY KEY(id))""")  db.close() |

Thêm mới bản ghi

|  |
| --- |
| **import** MySQLdb  db = MySQLdb.connect("localhost", "admin", "1234", "testdb" )  cursor = db.cursor()  **try**:   cursor.execute('''INSERT INTO STUDENT(name,address)                     VALUES ('Nguyen Van An', 'Ha Noi')''')                                             cursor.execute('''INSERT INTO STUDENT(name,address)                     VALUES ('Nguyen Thi Binh', 'TP.HCM')''')                                             db.commit() **except** Exception **as** e:   print(str(e))   db.rollback()  db.close() |

Đọc dữ liệu

|  |
| --- |
| **import** MySQLdb  db = MySQLdb.connect("localhost", "admin", "1234", "testdb" )  cursor = db.cursor()  **try**:    cursor.execute("SELECT name, address FROM STUDENT")    results = cursor.fetchall()     **for** name, address **in** results:             print(name, address)                  **except** Exception **as** e:   print(str(e))  db.close() |