**Lập trình là gì**

Khoa học máy tính

Dung dòng lệnh

Thuật toán

Để tạo thành ngôn ngứ máy

Tạo thành các ứng dụng

VD PM:

Phần mềm là một cái ứng dụng đc xây dựng bởi thuật toán, dòng lệnh để phục vụ một mục đích nào đó

Java: hệ thống lớn

C#: FrameWork

C++: Chuyên về game đồ họa

C: hệ thống nhúng

PHP: Lập trình Web nhanh gọn và đơn giản

Python:   
+ Tạo App

+ Tạo Web

+ Xây dựng hệ thống

+ Mạnh trong AI ( Trí tuệ nhân tạo)

Python là gì:

Là một ngôn ngữ lập trình bậc cao. (Gần giống với ngôn ngữ của con người)

+ Ngôn ngữ hướng thủ tục và ngôn ngữ hướng đối tượng

-Hướng thủ tục: chú ý vào các chức năng

-Hướng đối tượng: chú ý vào một thực thể

+ Ngôn ngữ thông dịch: (interprinter)dịch từng dòng cốt 1. Dịch dòng 1 nếu ok thì sẽ chạy đến dòng 2, nếu dòng 2 có lỗi thì nó sẽ dừng luôn.

+ Ngôn ngữ biên dịch là nó sẽ chỉ chạy khi các dòng lệnh ko có lỗi.

**Chương 2: Cú Pháp**

2.1:Biến

Khi định nghĩa biến a thì biến a sẽ trỏ đến 1 địa chỉ trong máy tính.

Khai báo: tên biến = value

Có thể gán nhiều giá trị cho 1 biến

\* **Toán tử số học**

+ Cộng

+ Trừ

+ Nhân

+ Chia

+ Lấy số dư %

+ Lấy số nguyên int(a/b) hoặc //

+ Lấy số mũ \* Ví dụ 2\*\*3=8

\* **Toán tử quan hệ**

**>,<,=,<=,>=,==,!= (<>)**

**\* Toán tử gán**

a=+=10 tức là a=a+10

a %= 10 Lấy a chia cho 10 và lấy phần dư rồi gán cho a

**\* Toán tử Logic**

**AND** : true <-> khi cả 2 là true, false khi một trong 2 sai và cả 2 sai

**Sử dụng toán tử \* để đại diện cho các toán tử còn lại**

Ví dụ:

a,\*b,c=1,2,3,4

Kết quả in ra là a=1

b=[2,3]

b=4

OR : true khi một trong 2 true, flase khi cả 2 đều false

**NOT** Đảo ngược kết quả. True thành False và ngược lại

\* **Toán tử thành viên**

**In/Not in**

Kiểm tra một đối tượng có hoặc không nằm trong đối tượng khác hay không.Giá trị trả về là **True** hoặc **False**

Cấu trúc điều khiển

Chú ý: Trong python thì nó sẽ có 1 đặc điểm đó là block code. Tức là phải có indentation và mỗi một dòng trong block code sẽ phải Indentation giống nhau.

Ví dụ: if a<=4

Print (a) (Print phải thụt dòng khoảng 4 dấu space so với if . Và các dòng lệnh sau cũng như vậy

**Import: Dùng để import các thư viện trong Python**

Ví dụ: import datetime

Datetime là thư viện về ngày tháng và thời gian

Bao gồm các hàm về xử lý thời gian và hiển thị thời gian

Bao gồm nhiều phương thức:

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| **date.replace(year=self.year, month=self.month, day=self.day)** | Trả về một đối tượng **date** với cùng giá trị, ngoại trừ các giá trị bị thay thế cho bởi tham số.  Ví dụ: d == date(2002, 12, 31),  d.replace(day=26) == date(2002, 12, 26). |
| date.timetuple() | Trả về một **time.struct\_time** như được trả về bởi **time.localtime()**. Trong đó giờ, phút và giây bằng 0, và cờ **DST**là -1.  **d.timetuple()** tương đương với **time.struct\_time((d.year, d.month, d.day, 0, 0, 0, d.weekday(), yday, -1))**, trong đó **yday = d.toordinal() - date(d.year, 1, 1).toordinal() + 1** là số thứ tự ngày trong năm, ngày 1-1 được gọi là ngày thứ 1. |
| date.toordinal() | Trả về số thứ tự của ngày theo lịch Gregorian, theo đó ngày 1 tháng 1 của năm 1 có thứ tự là 1.  Đối với một đối tượng date bất kỳ d, **date.fromordinal(d.toordinal()) == d**. |
| date.weekday() | Trả về ngày trong tuần có kiểu integer, với thứ hai là 0, thứ ba là 1, .. và chủ nhật là 6.  Ví dụ, **date(2002, 12, 4).weekday() == 2**, thứ tư. Xem thêm **isoweekday()**. |
| date.isoweekday() | Trả về ngày trong tuần là số có kiểu integer (Theo tiêu chuẩn ISO), với thứ hai là 1, thứ ba là 2 và chủnhậtlà7.  Ví dụ, **date(2002, 12, 4).isoweekday() == 3**, Thứ tư. Xem thêm **weekday()**, **isocalendar()**. |
| date.isocalendar() | Trả về một Tuple có 3 phần tử, (ISO year, ISO week number, ISO weekday). |
| date.isoformat() | Trả về một String đại diện ngày tháng theo định dạng ISO 8601, ‘YYYY-MM-DD’.  Ví dụ, **date(2002, 12, 4).isoformat() == '2002-12-04'**. |
| date.\_\_str\_\_() | Đối với đối tượng **date** d, **str(d)** tương đương với **d.isoformat()**. |
| date.ctime() | Trả về String đại diện **date**, ví dụ ***date(2002, 12, 4).ctime() == 'Wed Dec 4 00:00:00 2002'***.  **d.ctime()** tương đương với **time.ctime(time.mktime(d.timetuple()))**. Hàm **ctime()** được gọi phù hợp với tiêu chuẩn C khi chạy trên các nền tảng (flatform). |
| date.strftime(format) | Trả về string đại diện date, theo định dạng (format) cho bởi tham số. Các mã định dạng cho giờ phút giây có giá tri 0.  Xem thêm hàm **strftime()** và **strptime()** của module **time**. |
| date.\_\_format\_\_(format) | Tương tự với **date.strftime()**. |

- Viết đối tượng date dạng xâu "2015-10-23"

>string\_date = str(my\_date)

- Định dạng lại date theo mong muốn

>string\_date = my\_date.strftime('%m/%d/%Y') # 23/10/2015

- Gọi lại đối tượng date từ 1 xâu dạng "2015-10-23"

>my\_date = datetime.date(\*[int(i) for i in string\_date.split("-")])

**Chú ý: Các hàm xử lý ngày tháng khi thay đổi thì phải gán lại cho biến ban đầu**

**Ví dụ:** my\_year1=my\_year1.replace(day=int(day3+1))

**Đọc tài liệu hướng dẫn**

B1: Đọc hướng dẫn và xác định được yêu cầu chi tiết ( không hiểu thì ghi lại)

B2:+ Phân tích yêu cầu

+ Tìm hiểu cách làm, thuật toán, thư viện,…

+ Xây dựng cấu trúc

+ Code

+ Test – kiểm tra các chức năng

**Vòng lặp Loop**

1. **Vòng lặp FOR**

For value in range((10)

Phương thức range(Start,Stop, step)

Start là điểm đầu

Stop là điểm cuối

Step là bước nhảy

**Chú ý với trường hợp index âm**

Ví dụ:

a="abcABCCD"

index = lent(a)

for index in range(index - 1, -1, -1):

Hàm sẽ chạy như sau:

sau khi chạy vòng đầu đến vòng thứ 2 hàm sẽ + vào bước step, nếu như tổng mà nhỏ hơn cái stop thì nó sẽ vẫn thực thi lệnh, nhưng nếu tổng mà nằm ngoài Stop thì nó sẽ không thực hiện. Đặc biệt chú ý với trường hợp duyệt âm của các chuỗi tring. Còn khi duyệt xuôi thì cái len(str) không phải + 1 ký tự vì string hay list đánh số ký tự chữ cái từ số 0.

Hàm enumerate là một hàm hỗ trợ lặp của vòng lặp For

Enumerate(lst,index)

Lst là đối tượng thuộc kiểu list

Index là vị trí mà ta muốn vòng lặp bắt đầu chạy

Ví dụ: enumerate(lst\_number,1): Chạy từ vị trí thứ 1, mặc định chạy từ 0

for index, value in enumerate(lst\_number):

1. While

Break: Thoát vòng lặp

Pass: Bỏ qua một điều kiện của vòng lặp While

Continue: Bỏ qua một vòng lặp khi thỏa mãn điều kiện sau đó vòng lặp lại chạy với phần tử tiếp theo

## Kiểu Dữ Liệu

**Mutable: (có thể thay đổi được)**

**Immutable: không biến đổi được**

1. **Number – immutable**: int(10),float(2.5),complex(a+bj) - j là căn bậc 2 của -1

Abs(x): tính giá trị tuyệt đối. int(x): convert(ép kiểu)

1. **String – immutable**

* Truy cập vào string: print(str[0:4])
* Update String:

+ print(update string: “,str[0:7] + “ABC”

* Toán tử với string

+

\*

IN

NOT IN

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Mô tả** | **Ví dụ** |
| + | Nối (concatenate) 2 string, tạo thành một string mới. | "Hello" +"Python" ==> "Hello Python" |
| \* | Tạo một string mới bằng cách nối (concatenate) nhiều lần bản copy của cùng môt string. | "Hello"\*2 ==> "HelloHello" |
| [] | Trả về ký tự tại vị trí cho bởi chỉ số. | a = "Hello" a[1] ==> "e" |
| [ : ] | Trả về một chuỗi con chứa các ký tự cho bởi phạm vi (range) | a = "Hello" a[1:4] ==> "ell" a[1: ] ==> "ello" |
| in | Trả về True nếu ký tự tồn tại trong string đã cho. | a = "Hello" 'H' in a ==> True |
| not in | Trả về True nếu ký tự không tồn tại trong string đã cho. | a = "Hello" 'M' not in a ==> True |
| r/R | Chuỗi thô (Raw String) - Ngăn chặn ý nghĩa thực tế của các ký tự thoát (Escape character). Cú pháp cho chuỗi thô giống hệt với chuỗi thông thường ngoại trừ **"toán tử chuỗi thô"**, chữ **"r"** đứng trước dấu ngoặc kép. **"R"** có thể là chữ thường (r) hoặc chữ hoa (R) và phải được đặt ngay trước dấu trích dẫn đầu tiên. | print (r'\n\t') ==> \n\t print (R'\n\t') ==> \n\t |
| % | Định dạng string | Xem tại phần dưới. |

Các phương thức

1, Capitalize().

Hàm này có tác dụng in hoa chữ cái đầu tiên của chuỗi.

**VD**:

string = "toidicode.com"

print(string.capitalize());

# Kết quả: Toidicode.com

2, Center().

Hàm này có tác dụng trả về chuỗi được hiển thị ở giữa một chuỗi.

**Cú Pháp**:

string.center(len, char)

**Trong đó**:

* string là chuỗi các bạn cần chuyển đổi.
* len là số lượng ký tự của chuỗi mới.
* char là ký tự sẽ hiển thị ở 2 bên chuỗi cũ. Mặc định nó sẽ là khoảng trắng.

**VD**: Nếu khó hiểu thì mời mọi người cùng xem qua ví dụ sau:

string = "toidicode.com"

print(string.center(20));

# Kết quả: Toidicode.com

print(string.center(20, '\*'));

# Kết quả:\*\*\*toidicode.com\*\*\*\*

**Chú ý**: Nếu len nhỏ hơn độ dài chuỗi cần xử lý thì, hàm này sẽ trả về chuỗi ban đầu.

3, Count().

Hàm này có tác dụng đếm xem trong chuỗi có bao nhiêu ký tự cần tìm.

**Cú Pháp**:

string.count(sub, start, end)

**Trong đó**:

* sub là chuỗi các bạn cần tìm kiếm và đếm.
* start là index bắt của chuỗi cần tìm. Mặc định thì strart = 0.
* end là index kết thúc của chuỗi cần tìm.  Mặc định thì end = len() của chuỗi.

**VD**:

string = "toidicode.com"

print(string.count('i'));

# Kết quả: 2

print(string.count('i', 3));

# Kết quả: 1

4, encode().

Hàm này có tác dụng encode (mã hóa) một chuỗi.

**Cú pháp**:

string.encode(type, mode)

**Trong đó**:

* type là kiểu encode của string. Mặc định sẽ là utf-8
* mode là chế độ báo lỗi nếu có khi encode. Python hỗ trợ 6 dạng mode như sau:
  + strict - Chế độ nghiêm ngặt, nó sẽ hiển thị lỗi dưới UnicodeDecodeError exception. Đây là chế độ mặc định.
  + ignore - bỏ qua tất cả các lỗi nếu có.
  + replace - nó sẽ thay thế lỗi bằng dấu ?.
  + xmlcharrefreplace - chèn tham chiếu XML.
  + backslashreplace - Chèn chuỗi \uNNNN.
  + namereplace - Chèn chuỗi \N{...}.

**VD**:

string = "toidicode.com"

print(string.encode());

# Kết quả: b'toidicode.com'

5, decode().

Hàm này có tác dụng decode (gải mã) chuỗi trông Python.

**Cú Pháp**:

string.decode(type, mode)

Về phần type và mode thì hoàn toàn giống ở encode nhé mọi người!

**VD**:

string = b'toidicode.com'

print(string.decode());

# Kết quả: toidicode.com

6, endswith().

Hàm này có tác dụng kiểm tra xem chuỗi hoặc khoảng chuỗi có được kết thúc bằng ký tự nào đó hay không. Nó sẽ trả về True nếu đúng và False nếu sai.

**Cú pháp**:

string.endswith(str, start, end)

**Trong đó**:

* str là chuỗi các bạn cần xác thực xem có phải chuỗi kết thúc không.
* strart là index bắt đầu chuỗi cần so sánh. Mặc định thì start = 0.
* end là index kết thúc chuỗi cần so sánh.  Mặc định thì end = len().

**VD**:

string = 'toidicode.com'

print(string.endswith('m'));

# Kết quả: True

print(string.endswith('m', 3, 10));

# Kết quả: False

7, expandtabs().

Hàm này có tác dụng tìm kiếm thay thế \t bằng các ký tự khoảng trắng.

**Cú Pháp**:

string.expandtabs(len)

**Trong đó**: len là số lượng khoảng trắng mà bạn muốn thay thế cho một \t. Mặc định thì len = 8.

**VD**:

string = 'toidicode.com\thoc lap trinh'

print(string.expandtabs());

# Kết quả: toidicode.com hoc lap trinh

8, find().

Hàm này có tác dụng tìm kiếm một chuỗi trong một chuỗi hoặc khoảng chuỗi. Nó sẽ trả về là vị trí bắt đầu của chuỗi tìm được trong chuỗi nếu tìm thấy và nếu không tìm thấy nó sẽ trả về  -1.

**Cú pháp**:

string.find(str, start, end)

**Trong đó**:

* str là chuỗi các bạn cần xác thực xem có phải chuỗi kết thúc không.
* start là index bắt đầu chuỗi cần so sánh. Mặc định thì start = 0.
* end là index kết thúc chuỗi cần so sánh.  Mặc định thì end = len().

**VD**:

string = 'toidicode.com'

print(string.find('di'));

# Kết quả: 3

9, index().

Hàm này tương tự như hàm find() chỉ khác duy nhất là nếu như không tìm thấy thì hàm này sẽ gọi exception.

**VD**:

string = 'toidicode.com'

print(string.index('vuthanhtai'));

# Kết quả: ValueError: substring not found

10, isalnum().

Hàm này có tác dụng kiểm tra xem một chuỗi có phải là chứa duy nhất các ký tự chữ hoặc chuỗi hay không? Nó sẽ trả về True nếu chuỗi chỉ chứa các ký tự chữ hoặc số. Và ngược lại nó sẽ trả về False nếu chuỗi chứa ký tự khác chuỗi và số.

**VD**:

string = 'toidicode'

print(string.isalnum());

# Kết quả: True

string = 'toidicode.com'

print(string.isalnum());

# Kết quả: False

11, isalpha().

Hàm này có tác dụng kiểm tra xem một chuỗi có phải là chứa duy nhất các ký tự chữ hay không? Nó sẽ trả về True nếu chuỗi này chỉ chứa duy các ký tự chữ trong bảng chữ cái, và sẽ trả về False nếu nó chứa số hoặc ký tự đặc biệt khác.

**VD**:

string = 'toidicode96'

print(string.isalpha());

# Kết quả: False

string = 'toidicodecom'

print(string.isalpha());

# Kết quả: True

12, isdigit().

Hàm này có tác dụng kiểm tra xem một chuỗi có phải là chứa duy nhất các chữ số hay không? Nó sẽ trả về True nếu đúng và False nếu sai.

**VD**:

string = 'toidicode96'

print(string.isdigit());

# Kết quả: False

string = '12051996'

print(string.isdigit());

# Kết quả: True

13, islower().

Hàm này có tác dụng kiểm tra xem một chuỗi có phải là in thường hay không? Nó sẽ trả về True nếu đúng và False nếu sai.

**VD**:

string = 'toidicode.com'

print(string.islower());

# Kết quả: True

string = '12051996'

print(string.islower());

# Kết quả: False

string = '9toidicode.com6'

print(string.islower());

# Kết quả: True

string = '9Toidicode.com6'

print(string.islower());

# Kết quả: False

14, isupper().

Hàm này có tác dụng kiểm tra xem một chuỗi có phải là in Hoa hay không? Nó sẽ trả về True nếu đúng và False nếu sai.

**VD**:

string = 'TOIDICODE.COM'

print(string.isupper());

# Kết quả: True

string = '12051996'

print(string.isupper());

# Kết quả: False

string = '9TOIDICODE.COM6'

print(string.isupper());

# Kết quả: True

string = '9Toidicode.com6'

print(string.isupper());

# Kết quả: False

15, isnumeric()

Hàm này có tác dụng kiểm tra xem một chuỗi có phải chỉ chứa duy nhất các ký tự số hay không? Nó sẽ trả về True nếu đúng và False nếu sai.

**VD**:

string = 'TOIDICODE.COM'

print(string.isnumeric());

# Kết quả: False

string = '12051996'

print(string.isnumeric());

# Kết quả: True

string = '9TOIDICODE.COM6'

print(string.isnumeric());

# Kết quả: False

string = '9Toidicode.com6'

print(string.isnumeric());

# Kết quả: False

16, isspace().

Hàm này có tác dụng kiểm tra xem một chuỗi có phải chỉ chứa duy nhất các ký tự khoảng trắng không? Nó sẽ trả về True nếu đúng và False nếu sai.

**VD**:

string = ' '

print (string.isspace());

# Kết quả: True

string = 'Vu Thanh Tai'

print (string.isspace());

# Kết quả: False

17, istitle().

Hàm này có tác dụng kiểm tra xem một chuỗi có phải là title hay không, chuỗi title là chuỗi có các chữ cái đầu đều được in hoa. Nó sẽ trả về True nếu đúng và ngược lại False nếu sai.

**VD**:

string = 'vu thanh Tai'

print(string.istitle())

# Kết quả: False

string = 'Vu Thanh Tai'

print(string.istitle())

# Kết quả: True

18, join().

Hàm này có tác dụng join squence bởi string.

**Cú pháp**:

string.join(squence)

**Trong đó**: squence là string, list,... mà bạn cần join lại với nhau bởi chuỗi string.

**VD**:

string\_one = ' '

string\_two = 'TAI'

print(string\_one.join(string\_two))

# Kết quả: T A I

string\_one = '-'

string\_two = ['T','D','C',]

print(string\_one.join(string\_two))

# Kết quả: T-D-C

19, len().

Hàm này có tác dụng trả về độ dài của chuỗi.

**VD**:

string = "Vu Thanh Tai"

print(len(string))

# Kết quả: 12

20, ljust().

Hàm này có tác dụng trả về một chuỗi với độ dài length được xác định, nếu chuỗi được chọn nhỏ hơn width thì nó sẽ sử dụng char để bù chỗ thiếu đó về phía bên phải của chuỗi.

string.ljust(length, char)

**Trong đó**:

* length là độ dài của chuỗi mới cần in ra.
* char là ký tự sẽ bù vào chuỗi mới nếu chuỗi cũ không đủ length. Mặc định thì char = khoảng trắng.

**VD**:

string = "Vu Thanh Tai"

print(string.ljust(17, "-"))

# Kết quả: Vu Thanh Tai-----

21, rjust().

Tương tự hàm ljust() nhưng chỉ có điều là nó sẽ bù về phía bên trái của chuỗi.

**VD**:

string = "Vu Thanh Tai"

print(string.rjust(17, "-"))

# Kết quả: -----Vu Thanh Tai

22, lower().

Hàm này có tác dụng chuyển đổi chuỗi về dạng in thường.

**VD**:

string = "Vu Thanh Tai"

print(string.lower())

# Kết quả: vu thanh tai

23, upper().

Hàm này có tác dụng chuyển đổi chuỗi sang dạng in hoa.

**VD**:

string = "Vu Thanh Tai"

print(string.upper())

# Kết quả: VU THANH TAI

24, lstrip().

Hàm này có tác dụng loại bỏ đi các ký tự char ở phía đầu của chuỗi.

**Cú Pháp**:

string.lstrip(char)

**Trong đó**: char là ký tự bạn muốn loại bỏ. Mặc định thì char sẽ bằng khoảng trắng (white space).

**VD**:

string = " Vu Thanh Tai"

print(string.lstrip())

# Kết quả: Vu Thanh Tai

string = "----Vu Thanh Tai"

print(string.lstrip('-'))

# Kết quả: Vu Thanh Tai

25, rstrip().

Tương tự như lstrip(), chỉ khác là rstrip nó sẽ loại bỏ char ở phần cuối của chuỗi.

**VD**:

string = "Vu Thanh Tai "

print(string.rstrip())

# Kết quả: Vu Thanh Tai

string = "Vu Thanh Tai----"

print(string.rstrip('-'))

# Kết quả: Vu Thanh Tai

26, strip().

Hàm này là sự kết hợp của lstrip() và rstrip(). Nó sẽ lại bỏ các ký tự char ở cả hai đầu của chuỗi.

**VD**:

string = " Vu Thanh Tai "

print(string.strip())

# Kết quả: Vu Thanh Tai

string = "----Vu Thanh Tai----"

print(string.strip('-'))

# Kết quả: Vu Thanh Tai

27, rfind().

Tương tự như hàm find(), nhưng hàm này nó sẽ trả về index của chuỗi cuối cùng tìm được trong chuỗi. Cú pháp sử dụng tương tự hàm find().

**VD**:

string = "Vu Thanh Tai"

print(string.rfind('T'))

# Kết quả: 9

28, rindex().

Tương tự như hàm index(),nhưng hàm này nó sẽ trả về index của chuỗi cuối cùng tìm được trong chuỗi. Cú pháp sử dụng tương tự hàm index().

**VD**:

string = "Vu Thanh Tai"

print(string.rindex('T'))

# Kết quả: 9

29, replace().

Hàm này có tác dụng tìm kiếm và thay thế chuỗi tìm được bằng chuỗi mới.

**Cú Pháp**:

string.replace(old,new,max)

**Trong đó**:

* old là chuỗi mà bạn cần tìm kiếm trong string.
* new là chuỗi mà bạn cần thay thế cho chuỗi old tìm được.
* max là số lượng từ có thể thay thế tối đa.

**VD**:

string = "Chao \*!"

print(string.replace('\*', 'Tai'))

# Kết quả: Chao Tai!

string = "A A A"

print(string.replace('A', 'Tai', 2))

# Kết quả: Tai Tai A

30, max().

Hàm này trả về chữ cái có độ sắp xếp cuối cùng theo bảng chữ cái alphabet nằm trong chuỗi.

**VD**:

string = "Vu Thanh Tai"

print(max(string))

# Kết quả: u

31, min().

Hàm này trả về chữ cái có độ sắp xếp đầu tiên theo bảng chữ cái alphabet nằm trong chuỗi.

**VD**:

string = "vuthanhtai"

print(min(string))

# Kết quả: a

32, title().

Hàm này có tác dụng chuyển đổi chuỗi sang kiểu title (xem ở trên).

**VD**:

string = "vu thanh tai"

print(string.title())

# Kết quả: Vu Thanh Tai

33, swapcase().

Hàm này có tác dụng chuyển đổi chuỗi sang dạng nghịch đảo của nó (nghịch đảo ở đây là hoa - thường).

**VD**:

string = "vu thanh tai"

print(string.swapcase())

# Kết quả: VU THANH TAI

string = "Vu Thanh Tai"

print(string.swapcase())

# Kết quả: vU tHANH tAI

34, zfill().

Hàm này có tác dụng như hàm ljust() , nhưng nó sẽ chỉ thêm được các ký tự zero (số 0) và trước chuỗi thôi.

**VD**:

string = "vu thanh tai"

print(string.zfill(17))

# Kết quả: 00000vu thanh tai

35, isdecimal().

Hàm này có tác dụng **gần như** hàm isdigit(), nó sẽ trả về True nếu chuỗi cần kiểm tra chỉ chứa các số thập phân, và ngược lại....

**VD**:

string = "Vu Thanh Tai 96"

print(string.isdecimal())

# Kết quả: False

string = "12051996"

print(string.isdecimal())

# Kết quả: True

36, split().

Hàm này có tác dụng tách chuỗi thành mảng bởi các char.

**Cú Pháp**:

string.split(char, max)

**Trong đó**:

* char là ký tự các bạn tìm và tách chuỗi bởi nó. Mặc định thì char = khoảng trắng.
* max là số lượng chuỗi tách tối đa.

**VD**:

string = "Vu Thanh Tai"

print(string.split())

# Kết quả: ['Vu', 'Thanh', 'Tai']

string = "Vu Thanh Tai"

print(string.split('a'))

# Kết quả: ['Vu Th', 'nh T', 'i']

string = "Vu Thanh Tai"

print(string.split(' ', 1))

# Kết quả: ['Vu', 'Thanh Tai']

37, splitlines().

Hàm này sẽ tách chuỗi bởi các ký tự \n.

**Cú pháp**:

string.splitlines(max)

**Trong đó**: max là số lần có thể cắt tối đa.

**VD**:

string = "Vu\nThanh\nTai"

print(string.splitlines())

# Kết quả: ['Vu', 'Thanh', 'Tai']

38, startswith().

Hàm này có tác dụng kiểm tra xem chuỗi hoặc khoảng chuỗi có được bắt đầu bằng ký tự nào đó hay không. Nó sẽ trả về True nếu đúng và False nếu sai.

**Cú pháp**:

string.startswith(str, start, end)

1. **List – mutable**

Khai báo lst\_a=[“a”,”b”,”c”]

Phần tử trong list có thể là nhiều kiểu dữ liệu

Ví dụ:

lst\_C = ["A", "B", "C",['a','b','c'], 1, 2, 3.1]

Hàm:

For data in list()

**If isinstance(data, list) : Kiểm tra data có phải là một list không hoặc có thể kiểm tra nhiều loại dữ liệu khác**

for x in data

print(x)

* Toán tử với list

+ Cộng 2 list với nhau

\* Nhân đôi list

Phương thức join(str): chèn **1 ký tự** hoặc **1** **string** vào trong một list **mà các phần tử của list đó thuộc dạng string.**

Ví dụ: lst = [‘a’, ‘b’, ‘c’]

“-“.join(lst) => a-b-c (Kết quả)

Một số phương thức trong mảng

Python có một danh sách các phương thức để chúng ta thao tác với mảng/ list

1.append ():

Thêm một phần tử vào cuối mảng/ list

2.clear ():

Xóa toàn bộ phần tử ra khỏi mảng/ list

## copy ():

1. Trả về một bản sao của mảng/ list
2. Ví dụ:
3. fruits = ["apple", "banana", "cherry"]
4. x = fruits.copy()
5. print (x)
6. # In ra màn hình: ['apple', 'banana', 'cherry']

## 4.count ():

Trả về số lượng phần tử có giá trị là tham số truyền vào<h2>

Tìm xem trong mảng có bao nhiêu phần tử giống tham số truyền vào

Ví dụ:

1. points = [1, 4, 2, 9, 7, 8, 9, 3, 1]
2. x = points.count(9)
3. print (x)
4. *# In ra màn hình: 2*

## 5.extend ():

Thêm các phần tử của một mảng/ list vào cuối mảng/ list hiện tại

Ví dụ:

fruits = ['apple', 'banana', 'cherry']

cars = ['Ford', 'BMW', 'Volvo']

fruits.extend(cars)

print (fruit)

# In ra màn hình: ['apple', 'banana', 'cherry', 'Ford', 'BMW', 'Volvo']

## 6.index ():

Trả về chỉ số của phần tử đầu tiên có trong mảng/ list<h2>

Ví dụ:

fruits = ['apple', 'banana', 'cherry']

**x** = fruits.**index**("cherry")

**print** (**x**)

*# In ra màn hình: 2*

## 7.insert ():

Thêm một phần tử vào trong mảng/ list tại vị trí là tham số truyền vào

Ví dụ:

fruits = ['apple', 'banana', 'cherry']

x = fruits.insert(1, "orange")

print(x)

# In ra màn hình: ['apple', 'orange', 'banana', 'cherry']

## 8.pop ():

Lấy một phần tử ra khỏi mảng/ list tại vị trí là tham số truyền vào

## 9.remove ():

Lấy một phần tử đầu tiên ra khỏi mảng/ list có giá trị là tham số truyền vào

## 10.reverse ():

Đảo ngược mảng/ list

Ví dụ:

fruits = ['apple', 'banana', 'cherry']

fruits.reverse()

print(fruits)

# In ra màn hình: ['cherry', 'banana', 'apple']

## 11.sort ():

Sắp xếp lại mảng/ list

Đối với chuỗi thì sort() sẽ sắp xếp theo thứ tự alphabet.

**12. Del**

Xóa một phần tử trong list

del lst[start:stop]

Chú ý cách lấy xử lý phần tử trong list

**Ví dụ lst[1:3] = [“X”,”Y”] thì nó chỉ thay đổi chỉ số 1 và chỉ số 2**

**Còn nếu lst[1:4] = [“X”,”Y”] nó sẽ thay thế phần tử 1, 2 và sẽ xóa phần tử thứ 3**

**13. Phương thức zip**

**Ví dụ**

lst\_A = ["A", "B", "C"]

lst\_C = ["A", "B", "C",['a','b','c'], 1, 2, 3.1]

lst\_E = list(zip(lst\_A, lst\_B))

**kết quả: ['A', 'X', 'Y', 'B', 'C', ['a', 'b', 'c'], 1, 2, 3.1][('A', 'A'), ('B', 'B'), ('C', 'C')]**

1. Tuple – immutable

Là một dãy các giá trị trong Python và các giá trị của nó không thay đổi dc

Cách truy cập phần tử trong Tuple giống như List

Các toán tử cũng giống List: +, \*, NOT, NOT IN

1. Dictionary – mutable: Một tập hợp các phần tử là các cặp key - value

Dic = {key1: value, key2:value, …}

Các key không nên trùng nhau. Nếu trùng nhau thì khi in ra các phần tử thì nó chỉ in ra phần tử cuối cùng trong các key bằng nhau đó.

+ Truy cập vào phần tử:

Cách 1: Truy cập theo key của DIC

Ví dụ:

print("ID : %s - Name : %s" % (data1["ID"], data1["Name"]))

Cách này có thể thay đổi được giá trị của Dic thông qua phép gán

Cách 2: Hàm dic.get(key) nếu key (key) có trong **dic** thì nó sẽ trả về giá trị, còn ngược lại thì nó sẽ trả về **NONE**

**Hoặc** dic.get(“key”,”giá trị thay thế”) ”) nếu key (key) có trong **dic** thì nó sẽ trả về giá trị, còn ngược lại thì nó sẽ trả về **“giá trị thay thế”**

\* Cách truy cập theo key có thể thay đổi được giá trị của Dic thông qua phép gán, còn cách 2 sử dụng hàm get thì chỉ duyệt phần tử và không thay đổi được giá trị.

+ Xóa phần tử

del dct[key]

Duyệt Dic bằng vòng For

Ví dụ:

For key, value in data1.items()

Print(“{} : {}”.format(key, value))

|  |  |
| --- | --- |
| **STT** | **Hàm và Miêu tả** |
| 1 | [**Hàm cmp(dict1, dict2)**](https://vietjack.com/python/dictionary_cmp_trong_python.jsp)  So sánh các phần tử của cả hai dict |
| 2 | [**Hàm len(dict)**](https://vietjack.com/python/dictionary_len_trong_python.jsp)  Độ dài của dict. Nó sẽ là số item trong Dictionary này |
| 3 | [**Hàm str(dict)**](https://vietjack.com/python/dictionary_str_trong_python.jsp)  Tạo ra một biểu diễn chuỗi có thể in được của một dict |
| 4 | [**Hàm type(variable)**](https://vietjack.com/python/dictionary_type_trong_python.jsp)  Trả về kiểu của biến đã truyền. Nếu biến đã truyền là Dictionary, thì nó sẽ trả về một kiểu Dictionary |

Các phương thức đã được xây dựng sẵn cho Dictionary trong Python:

|  |  |
| --- | --- |
| **STT** | **Phương thức và Miêu tả** |
| 1 | [**Phương thức dict.clear()**](https://vietjack.com/python/dictionary_clear_trong_python.jsp)  Xóa tất cả phần tử của *dict* |
| 2 | [**Phương thức dict.copy()**](https://vietjack.com/python/dictionary_copy_trong_python.jsp)  Trả về bản sao của *dict* |
| 3 | [**Phương thức fromkeys(seq,value1)/ fromkeys(seq)**](https://vietjack.com/python/dictionary_fromkeys_trong_python.jsp)  Được sử dụng để tạo một Dictionary mới từ dãy seq và value1. Trong đó dãy seq tạo nên các key và tất cả các key chia sẻ các giá trị từ value1. Trong trường hợp value1 không được cung cấp thì value của các key được thiết lập là None |
| 4 | [**Phương thức dict.get(key, default=None)**](https://vietjack.com/python/dictionary_get_trong_python.jsp)  Trả về giá trị của key đã cho. Nếu key không có mặt thì phương thức này trả về None |
| 5 | [**Phương thức dict.has\_key(key)**](https://vietjack.com/python/dictionary_has_key_trong_python.jsp)  Trả về true nếu key là có mặt trong Dictionary, nếu không là false |
| 6 | [**Phương thức dict.items()**](https://vietjack.com/python/dictionary_items_trong_python.jsp)  Trả về tất cả các cặp (key-value) của một Dictionary |
| 7 | [**Phương thức dict.keys()**](https://vietjack.com/python/dictionary_keys_trong_python.jsp)  Trả về tất cả các key của một Dictionary |
| 8 | [**Phương thức dict.setdefault(key, default=None)**](https://vietjack.com/python/dictionary_setdefault_trong_python.jsp)  Tương tự get(), nhưng sẽ thiết lập dict[key]=default nếu *key* là không tồn tại trong dict |
| 9 | [**Phương thức dict.update(dict2)**](https://vietjack.com/python/dictionary_update_trong_python.jsp)  Được sử dụng để thêm các item của dictionary 2 vào Dictionary đầu tiên |
| 10 | [**Phương thức dict.values()**](https://vietjack.com/python/dictionary_values_trong_python.jsp)  Trả về tất cả các value của một Dictionary |

**Có 3 kiểu duyệt theo vòng lặp của DIC**

- Kiểu list key:

list(dic.keys()): Trả về 1 disk\_keys (Dạng list) chứa tất cả các key của DIC.(Phải dùng ép kiểu list() để chuyển về dạng list chuẩn)

- Kiểu list value:

list(dic.values()): : Trả về 1 disk\_values (Dạng list) chứa tất cả các value của DIC.(Phải dùng ép kiểu list() để chuyển về dạng list chuẩn)

- Kiểu dic.items(): Trả về 1 list chứa các cặp giá trị key – value

+ Có 2 cách để loại bỏ một phần tử ra khỏi một dictionary.

1. Sử dụng toán tử **del**

Ví dụ: del contacts["John"]

1. Sử dụng phương thức **\_\_delitem\_\_(key)**

contacts.\_\_delitem\_\_( "Tom")

**Chú ý: Khi sử dụng câu lệnh rẽ nhánh khi ta muốn kiểm tra giá trị của list này và list kia ta nên sử dụng hàm NOT hoặc NOT IN để kiểm tra data của list 1 có nằm trong list 2 hay ko. Hơn là sử dụng cách sử dụng 2 vòng lặp kết hợp với với câu lệnh rẽ nhánh.**

**- Ta có thể sử dụng lệnh print(x,end=”-“) để chèn một ký tự giữa các phần tử của một List, một String, hoặc một Tuple,…**

**- Khi muốn thực hiện xóa phần tử nào đó một list hoặc dictionary khi đang duyệt vòng lặp thì ta nên dùng vòng lặp ngược để không bị thay đổi chỉ số index (loại bỏ lỗi out of range)**

**- Khi sử dụng hàm copy() với những kiểu dữ liệu mà bên trong nó có chứa kiểu dữ liệu khác thì ta chỉ copy được với kiểu dữ liệu đầu tiên**

**Ví dụ: dic1 = {“ID”: 1, “sản phẩm”:{“Name” : “Nokia”, “Price” : 300}}**

**Thì khi ta thực hiện hàm copy – dic2 = dic1.copy() thì khi thay đổi giá trị của dic2 ở phần ID thì dic1 không thay đổi, nhưng ta thay đổi giá trị “sản phẩm” của dic2 thì giá trị “sản phẩm” của dic1 cũng sẽ thay đổi. Tức là thực hiện lệnh copy() ban đầu chỉ áp dụng với dữ liệu dic1 ban đầu còn kiểu dữ liệu dic “sản phẩm” thì sẽ không áp dụng. Vậy tức là trong dic1 có bao nhiêu kiểu dữ liệu thì ta phải copy() bấy nhiêu lần tương ứng đặc biệt với những kiểu dữ liệu mutable. Như ở đây ta phải thực hiện 1 lần copy() nữa- dic2[“sản phẩm”] = dic1[“sản phẩm”].copy(). Hoặc sử dụng thư viện import copy và sử dụng hàm deepcopy(dic1)**

**Ví dụ:** dic\_increat = copy.deepcopy(product06)

**Function**

+ Đoạn lệnh được xây dựng thực hiện một công việc nào đó

+ Tái sử dụng được

+ Dễ quản lý, chỉnh sửa

+ Che giấu được thông tin

Cấu trúc hàm: def tenham(thamso1, thamso2,…)

return

thamso1,thamso2: đối số truyền vào

Return: để trả về giá trị, một hàm có thể có giá trị return hoặc không. Nếu không có thì hàm đấy mặc định trả về giá trị là none

Trong hàm đầu tiên phải có phần giải thích về ý nghĩa của hàm bao gồm tên hàm, các tham só và return trả về giá trị được đặt trong cặp dấu “”” “””

Các loại hàm:

- Hàm với tham số bắt buộc: khi gọi hàm thì có bao nhiêu pararameter trong phần định nghĩa thì phải truyền vào bấy nhiêu.

- Hàm với tham số mặc định: Hàm có nhiều tham số, bao gồm các tham số bắt buộc và các tham số có giá trị mặc định (Default – argument)

- Hàm có tham số với độ dài thay đổi: là hàm có thể truyền vào bao nhiêu tham số cũng được và không cần phụ thuộc vào định nghĩa hàm của nó.

Def function\_name(param1, para2, \*tenparam)

- Hàm Keywords: Hàm phải đi cùng với các từ khóa

- Hàm nặc danh là hàm chỉ trả về giá trị là một biểu thức, có thể có 0 hoặc nhiều tham số truyền vào. Từ khóa **lambda**

1. Hàm nặc danh có thể có 0 hoặc nhiều tham số, nhưng trong thân hàm chỉ có duy nhất một biểu thức (expression). Giá trị của biểu thức chính là giá trị trả về của hàm. Nhưng không được sử dụng từ khóa **'return'** ngay trước biểu thức.
2. Danh sách các tham số cách nhau bởi dấu phẩy, và không được đặt trong cặp dấu ngặc tròn ( ).
3. Trong thân của hàm nặc danh, bạn không thể truy cập các biến bên ngoài, bạn chỉ có thể truy cập các tham số của nó.
4. Hàm nặc danh không thể gọi trực tiếp hàm print, bởi vì **lambda** đòi hỏi một biểu thức.

Tác dụng của hàm nặc danh là khi chạy xong hàm thì nó sẽ xóa khỏi bộ nhớ, giúp tiết kiệm bộ nhớ.

**Tên hàm.\_\_code\_\_.co\_nlocals : phương thức đếm số biến locals trong một hàm**

- Global variables: biến toàn cục

Muốn thay đổi biến global trong hàm thì trong hàm chỉ cần khai báo biến global trong hàm kèm với từ khóa global ở trước biến

Vd: def ham()

global g\_variable

- Local variables: biến cục bộ

**Việc biến có thể hay không còn phụ thuộc vào tính năng mutable hay immutable nữa. Nếu là biến kiểu mutable (list, dic) thì khi thay đổi trong hàm thì giá trị của nó cũng thay đổi khi đó gọi là kiểu tham chiếu tức là trỏ đến địa chỉ, còn với các biến immutable thì là tham trị(copy giá trị của biến immutable)**

Kiến thức về class, module, commandline

**Class**

#### **1. Tổng quan về OOP**

* Class (lớp): Được định nghĩa cho một đối tượng bao gồm một tập các attributes (thuộc tính) đặc trưng cho tất cả các đối tượng của lớp. Attributes gồm: data members và methods được gọi thông qua ký hiệu dấu chấm.
* Instance: Một hiện thực cụ thể của một lớp, có thể gọi là thực thể
* object: instance duy nhất chứa cấu trúc dữ liệu được định nghĩa bởi class.(Ví dụ: khai báo lớp HocSinh ta sẽ có object HocSinh, sau đó tạo biến nguyen\_van\_a = HocSinh() thì có thêm instance nguyen\_van\_a)
* Methods: Hàm thực thi
* Data members: Biến chứa dữ liệu gồm: class variable & instance variable
* Class variable: Biến dùng chung cho tất cả các đối tượng của lớp, được định nghĩa trong lớp mà không nằm trong methods (hàm thực thi) nào cả. Các biến này không được sử dụng thường xuyên.
* Instance variable: biến được định nghĩa bên trong methods và chỉ thuộc về các instance (đối tượng thực thể của lớp).
* Function overloading: method định nghĩa các phép toán nhiều instances tham gia (Ví dụ cộng 2 instances, so sánh lớn hơn nhỏ hơn của 2 instances, ...).
* Operator overloading: Phép toán cần nhiều method tham gia.
* Inheritance: Việc chuyển giao các đặc tính của một lớp kế thừa từ lớp khác.
* Instantiation: Việc tạo ra một instance của một class.

1. Khái niệm:

+ Đối tượng: Là một tập hợp các vật/ việc hoặc chủ thể nào đấy có cùng 1 hoặc nhiều các đặc điểm chung

2. Lập trình hướng đối tượng

+ Khái niệm: Tập trung phát triển các chức năng theo đặc điểm chung của 1 đối tượng.

3. Class(Lớp):

+ Khái niệm: Nguyên mẫu để định nghĩa một đối tượng nào đó.

+ Thành phần:

Thuộc tính Attribute: Là các biến ở trong class

Phương thức Method. (Hàm trong class)

4. Cú pháp:

class tên\_class(tên class cha)

“””

Doc string

“””

**pass: Tạo ra một đối tượng và không làm gì cả**

**Tất cả các đối tượng trong python đều là các class. Ví dụ khai báo 1 chuỗi str = “abc” thì str thuộc kiểu string và lớp string. Nó sẽ có thể sử dụng các phương thức trong lớp string ví dụ: str.split()**

+ Hàm khởi tạo

def \_\_init\_\_(self)

self: là đại diện cho class

- Phương thức (Method):

+ Instance method: sv= Sinhvien()

sv.instance\_method(“OK”)

**với hàm instance\_method thì chỉ có đối tượng instance mới có thể gọi đến nó**

def instance\_method(self, status):

print("Instance method có chứa thông tin trạng thái làm việc là: %s" %status)

+ Class method:

**Hàm classmethod thì cả đối tượng instance và class đều có thể gọi được đến nó nhưng nó vẫn bind cho class tức là sẽ không ảnh hưởng hay tác dụng gì đến các instance**

@classmethod

def class\_method(cls):

print("Class”)

+ Static Method:

@staticmethod

... def static\_bar(x):

... print("executing static\_bar(%s)" % x)

Riêng static method là trường hợp đặc biệt, mặc dù nó là một phương thức nhưng dù gọi foo.static\_bar hay Foo.static\_bar thì kết quả trả về vẫn là hàm ban đầu không hề được "bind" bất cứ một đối tượng nào.

**Static method không cho truy cập vào các thuộc tính của class**

**Hàm def \_\_str\_\_(self) (Hàm chạy khi ta gọi in một đối tượng instance)**

**Print(“Đây là hàm str”)**

**Print(SV1)**

**Hàm def \_\_del\_\_(self) (Hàm hủy một đối tượng)**

**del SV1**

**4 Đặc điểm của lập trình hướng đối tượng(OPP)**

1. Đóng gói:(Encapsulation): Phạm vi truy cập của biến, phương thức

Class employee()

name = "Tên mặc định" #Public

\_position = "Staff" #Protected

\_\_salary = 0.0 #Private

Ví dụ: print(employee.\_\_salary) => Lỗi (bởi vì đây là biến private và nó bị truy cập từ bên ngoài)

Biến private có cú pháp \_\_tên biến, nó không thể truy cập trực tiếp từ bên trong thông qua thuộc tính mà phải truy cập thông qua phương thức. Hoặc có thể truy cập theo dạng:

counter = JustCounter()

print counter.\_JustCounter\_\_secretCount

**Ta có thể thêm hoặc thay đổi giá trị một thuộc tính (biến public) vào Class: Sinhvien.sdt = giá trị. Thì tương đương trong class Sinhvien có thêm một thuộc tính mới (Biến class) là sdt. Và các thể hiện của class Sinhvien có thể truy cập đến.**

2. Tính kế thừa: (Inheritance) Là việc truyền các đặc trưng của một lớp cho các lớp khác mà kế thừa từ lớp ban đầu.

Dùng lệnh super().tenphuongthuc để truy cập đến phương thức của lớp cha hoặc super(lopcon, self).tenphuongthuc

Khi muốn truy cập trực tiếp từ phương thức mà trùng tên với phương thức của lớp con thì ta có thể truy cập theo phương thức dog\_01.\_\_class\_\_.\_\_bases\_\_[0].display(dog\_01.\_\_class\_\_.\_\_bases\_\_[0]))

**Đối tượng instance được truy cập vào mọi thành phần của class. Còn Class chỉ truy cập được biến và hàm class. Từ khóa self để đại diện cho một thể hiện instance, còn từ khóa cls trong class method để đại diện cho chính Class đấy.**

3. Đa hình: (Polymorphism): Trong class cha và class con thì vẫn có thể có những phương thức trùng tên nhau nhưng lại xử lý các công việc khác nhau

Khi muốn truy cập trực tiếp từ phương thức mà trùng tên với phương thức của lớp con thì ta có thể truy cập theo phương thức dog\_01.\_\_class\_\_.\_\_bases\_\_[0].display(dog\_01.\_\_class\_\_.\_\_bases\_\_[0]))

4. Trừu tượng (Abstraction): Là khả năng chương trình bỏ qua 1 đối tượng nào đó của thông tin hay nói cách khác là nó có khả năng tập trung vào những điểm cần thiết.

- Một class abstraction là 1 class mà bên trong nó chứa ít nhất 1 phương thức trừu tượng.

**+ Chỉ được khai báo và không được viết code trong nó**

Đặc điểm:

**- Một abstraction thì không được khởi tạo từ một object từ nó**

**- Một class mà kế thừa từ một abstraction thì phải khai báo lại mọi phương thức trừu tượng của abstraction class đó (Tức là phải khai báo lại mọi phương thức trừu tượng của lớp cha tại lớp con dưới dạng phương thức thông thường)**

Cú pháp:

from abc import ABC

**Một Class abstraction tương tự như một đầu việc mà đối tượng lớp con thừa kế nó phải làm tất cả các đầu việc (tương đương với các phương thức trừu tượng của lớp cha)**

from abc import ABC, abstractclassmethod

# Khai báo

class ClassName(ABC):

def \_\_init\_\_(self, name, age, city):

self.name = name

self.age = age

self.city = city

@abstractclassmethod

def abstraction\_name(self):

pass

class Person(ClassName):

def \_\_init\_\_(self, name, age, city):

super().\_\_init\_\_(name, age, city)

def abstraction\_name(self):

print("%s - %s - %s" %(self.name, self.age, self.city))

person\_01 = Person("Hoàng", 32, "Hà Nội")

**\* Từ khóa self để đại diện cho một đối tượng của 1 class. Khi 1 class kế thừa từ 1 class, thì khi 1 đối tượng được tạo ra từ class con và truy cập đến các hàm của class cha thì khi đó self của các hàm trong class cha lại là đối tượng được tạo ra từ class con**

# [Ý nghĩa của if \_\_name\_\_ == ‘\_\_main\_\_’: trong Python?](https://daynhauhoc.com/t/y-nghia-cua-if---name-----main---trong-python/8315)

# Tức là kiểm tra xem chương trình có đang chạy từ file chính hay không.

# Thuộc tính \_\_name\_\_ để biết được tên chương trình chính, nếu chương trình đang chạy từ chính file đó thì \_\_name sẽ bằng \_\_main\_\_.

# Ví dụ ta viết:

print(\_\_name\_\_)

# cả trong file A.py và file B.py. sau đó từ file A to import B. Vậy khi chạy chương trình từ A nó sẽ in ra 2 dòng đó là:

# B

# \_\_main\_\_

# Còn khi chạy trực tiếp từ file đó thì nó sẽ cho ra là main, còn chạy từ file import module thì nó sẽ trả về tên của Module import đó

**Từ đây ta có thể xây dựng một chương trình mà chỉ khi nào chạy trực tiếp từ file mà ta muốn thì đoạn code mà ta muốn mới được thực thi.**

**Ta sẽ cho đoạn code đó nằm trong điều kiện:**

if \_\_name\_\_ == '\_\_main\_\_':