Biến và các KDL cơ bản

tenBien = giaTri

**Trong đó**:

* tenBien là tên của biến mà các bạn muốn đặt. Tên biến này không được bắt đầu bằng số hay các ký tự đặc biệt, mà chỉ được bắt đầu bằng chữ cái hoặc ký tự \_ và nó có phân biệt hoa thường.
* giaTri là giá trị của biến mà bạn muốn gán.

Python cũng giống như một số các ngôn ngữ bậc cao khác, khi ta khai báo biến thì kiểu dữ liệu của nó sẽ **tự động được detect**. Vì vậy nên chúng ta cũng không phải quá vất vả khi khai báo 1 biến.

Trong một trường hợp nào đó mà bạn muốn chuyển đổi kiểu dữ liệu của một biến, thì Python cũng hỗ trợ bạn qua các hàm cơ bản sau:

* float(data) chuyển đổi sang kiểu số thực.
* int(data,base) chuyển đổi sang kiểu số, trong đó base là kiểu hệ số mà các bạn muốn chuyển đổi sang (tham số này có thể bỏ trống).
* str(data) chuyển đổi sang dạng chuỗi.
* complex(data) chuyển đổi sang kiểu phức hợp.
* tuple(data) chuyển đổi sang kiểu Tuple.
* dict(data) chuyển đổi sang kiểu Dictionary.
* hex(data) chuyển đổi sang hệ 16.
* oct(data) chuyển đổi sang hệ 8.
* chr(data) chuyển đổi sang dạng ký tự.

Chuỗi trong Python

* Các kí tự đặc biệt trong chuỗi:

\n ngắt xuống dòng và bắt đầu dòng mời.

\t đẩy nội dung phía sau nó cách 1 tab.

\a chuông cảnh báo.

\b xóa bỏ khoảng trắng phía trước nó.

Ngoài ra bạn cũng có thể sử dụng để in ra các ký tự đặc biệt khác bằng việc sử dụng theo cú pháp \xnn, với n là 0->9, hoặc a->f hoặc A->F.

Format chuỗi: print("%type" %(binding))

**Trong đó**:

* type là các kiểu dữ liệu các bạn muốn binding và thay thế vào vị trí đó.
* binding là giá trị mà các bạn muốn binding vào vị trí được xác định trong chuỗi.

Type là các kiểu sau:

|  |  |
| --- | --- |
| **Cú pháp fomat** | **Mô tả** |
| %c | character |
| %s | chuỗi |
| %i | số nguyên |
| %d | số nguyên |
| %u | số nguyên |
| %o | bát phân |
| %x | thập lục phân (in thường) |
| %X | thập lục phân (in hoa) |
| %e | số mũ  (với e thường) |
| %E | số mũ  (với e hoa) |
| %f | số thực |
| %g | dạng rút gọn của %f and %e |
| %G | dạng rút gọn của %f and %E |

Truy cập tới từng giá trị của chuỗi

Chuỗi trong Python được lưu trữ vào trong các ô nhớ với mỗi ô nhớ tương đương với một ký tự đơn (khác với các ngôn ngữ khác) và các ký tự này được xếp liên tiếp với nhau. Do đó kiểu dữ liệu chuỗi trong Python có thể được truy xuất đến từng ký tự trong nó (các ngôn ngữ khác không có, PHP7.1.X mới hỗ trợ ở đây là reverse index string).

Để truy cập đến từng ký tự bên trong chuỗi, các bạn sử dụng cú pháp sau:

stringName[index]

**Trong đó**:

* stringName là tên của biến chứa chuỗi, hoặc chuỗi.
* index là vị trí của ký tự bạn muốn lấy ra. Index này hỗ trợ chúng ta truy xuất được cả 2 chiều của chuỗi nếu:
  + Tính từ đầu thì nó bắt đầu từ 0.
  + Tính từ cuối thì nó bắt đầu từ -1.

Nếu trong trường hợp các bạn muốn lấy nội dung của một đoạn chuỗi trong chuỗi đó thì có thể sử dụng cú pháp sau:

stringName[start:end]

**Trong đó**:

* stringName là tên của biến chứa chuỗi, hoặc chuỗi.
* start là vị trí của ký tự bắt đầu lấy, nếu để trống start thì nó sẽ lấy từ 0.
* end là vị trí kết thúc (nó sẽ lấy trong khoảng từ start đến < end), nếu để trống end thì nó sẽ lấy đến hết chuỗi.

Các hàm xử lý chuỗi

## 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.
* 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.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 tác 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)

**Trong đó**:

* str là chuỗi các bạn cần xác thực xem có phải chuỗi bắt đầu 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.startswith('t'));

# Kết quả: True

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

# Kết quả: False

## 39, maketrans().

Hàm này có tác dụng tạo ra các translation cho chuỗi. Dùng kết hợp với phương thức translate().

**Cú Pháp**:

string.maketrans(in, out)

**Trong đó**:

* in là chuỗi các ký tự các bạn cần tìm.
* out là chuỗi chứa các ký tự các bạn cần thay thế.

## 40, translate().

Hàm này có tác dụng thực thi việc dịch chuỗi. Dùng kết hợp với phương thức **makestrans()**.

**VD**:

inputs = "abcdefghijklmnopqrstuxyz";

outputs = "ABCDEFGHIJKLMNOPQRSTUXYZ";

string = "Vu Thanh Tai";

trans = string.maketrans(inputs, outputs)

print(string.translate(trans))

# Kết quả: VU THANH TAI

# List trong Python

List trong Python là một dạng dữ liệu cho phép lưu trữ nhiều kiểu dữ liệu khác nhau trong nó, và chũng ta có thể truy xuất đến các phần tử bên trong nó thông qua vị trí của phần tử đó trong list. Ở đây, nếu như bạn nào đã tìm hiểu qua một ngôn ngữ nào đó thì có thể coi list trong Python như một mảng tuần tự trong các ngôn ngữ khác.

Để khai báo một list trong Python thì chúng ta sử dụng cặp dấu [] và bên trong là các giá trị của list.

[value1, value2,..., valueN]

**Trong đó**: value1, value2,..., valueN là các giá trị của list.

Trong trường hợp bạn muốn in ra một phần của list thì bạn sử dụng cú pháp sau:

list[start:end]

**Trong đó**:

* list là tên của biến chứa list.
* start là ví trí bắt đầu lấy ra list con. Nếu để trống thì nó sẽ lấy từ đầy list.
* end là vị trí kết thúc. Nếu để trống thì nó sẽ lấy đến phần tử cuối cùng của list.

### Update

Để sửa giá trị của các phần tử trong list thì các bạn chỉ cần truy cập đến phần tử mà mình cần sửa đổi và tiến hành gán giá trị mới cho nó.

**VD**: Sửa name thứ 2 trong list ở ví dụ trên thành 1996.

name = ['Vu Thanh Tai', 'Nguyen Van A', 'Nguyen Thi E']

print(name)

# ['Vu Thanh Tai', 'Nguyen Van A', 'Nguyen Thi E']

name[1] = 1996

print(name)

# ['Vu Thanh Tai', 1996, 'Nguyen Thi E']

### Delete

Để xóa một hoặc nhiều phần tử trong list thì các bạn cần truy cập đến phần tử cần xóa và dùng hàm del để xóa. Và sau khi chúng ta xóa phần tử trong list thì index của list sẽ được cập nhật lại.

**VD**: Xóa phần tử thứ 3 trong list trên.

name = ['Vu Thanh Tai', 'Nguyen Van A', 'Nguyen Thi E']

print(name)

# ['Vu Thanh Tai', 'Nguyen Van A', 'Nguyen Thi E']

del name[2]

print(name)

# ['Vu Thanh Tai', 'Nguyen Van A']

Do list có thể chứa nhiều kiểu dữ liệu khác nhau lên chúng ta hoàn toàn có thể khai báo một list chứa một hoặc nhiều list khác nhau.

**VD**:

option = [12,5,1996]

myList = ['Vu Thanh Tai', option]

print(myList)

# ['Vu Thanh Tai', [12, 5, 1996]]

Và cứ như thế chúng ta có thể lồng N list khác vào trong list.

Đối với list lồng nhau như này thì chúng ta chũng truy xuất đến các phần tử như bình thường, theo cấp từ ngoài vào trong.

**VD**: Mình sẽ truy cập vào phần tử dầu tiên trong list option.

option = [12,5,1996]

myList = ['Vu Thanh Tai', option]

print(myList)

# ['Vu Thanh Tai', [12, 5, 1996]]

subList = myList[1] # [12, 5, 1996]

subList[0] # 12

# hoặc có thể viết ngắn gọn như sau

myList[1][0] # 12

## 1, list().

Hàm này có tác dụng chuyển đổi kiểu dữ liệu của một biến sang dạng list.

**Cú pháp**:

list(data)

**Trong đó**, data là biến chứa tuple bạn cần chuyển đổi.

**VD**:

string = "Vu Thanh Tai"

print(list(string))

# Ket Qua: ['V', 'u', ' ', 'T', 'h', 'a', 'n', 'h', ' ', 'T', 'a', 'i']

tup = ('A', 'B', 'C')

print(list(tup))

# Ket Qua: ['A', 'B', 'C']

## 2, len().

Hàm này trả về số lượng phần tử có trong list.

**Cú pháp**:

len(list)

**Trong đó,**list là list mà các bạn cần đếm.

**VD**:

list = ['A', 'B', 'C']

print(len(list))

#Kết quả: 3

## 3, max().

Hàm này sẽ trả về phần tử có giá trị lớn nhất trong list. Nếu là chuỗi thì nó sẽ trả về phần tử có độ dài chuỗi dài nhất, nếu là số thì nó sẽ trả về phần tử có số lớn nhất.

**Cú pháp**:

max(list)

**Trong đó,**list là list mà các bạn cần kiểm tra.

**VD**:

list = ['A', 'B', 'C']

print(max(list))

#Kết quả: C

list = ['1', '3', '2']

print(max(list))

#Kết quả: 3

## 4, min().

Hàm này sẽ trả về phần tử có giá trị nhỏ nhất trong list. Nếu là chuỗi thì nó sẽ trả về phần tử có độ dài chuỗi ngắn nhất, nếu là số thì nó sẽ trả về phần tử có số nhỏ nhất.

**Cú pháp**:

min(list)

**Trong đó,**list là list mà các bạn cần kiểm tra.

**VD**:

list = ['A', 'B', 'C']

print(min(list))

#Kết quả: A

list = ['1', '3', '2']

print(max(list))

#Kết quả: 1

## 5, append().

Phương thức này có tác dụng thêm phần vào cuối của một list.

**Cú pháp**:

mylist.append(obj)

**Trong đó**:

* mylist là list mà các bạn cần thêm phần tử.
* obj là phần tử mà bạn muốn thêm vào mylist.

**VD**:

list = ['A', 'B', 'C']

list.append('D')

print(list)

# Kết quả: ['A', 'B', 'C', 'D']

list.append(('E', 'F'))

print(list)

# Kết quả: ['A', 'B', 'C', 'D', ('E', 'F')]

## 6, extend().

Hàm này có tác dụng kế thừa lại các phần tử của list2 và thêm vào trong list1.

**Cú pháp**:

list1.extend(list2)

**Trong đó**:

* list1 là list mà bạn muốn kế thừa từ một list khác (ở đây là list2).
* list2 là list được sử dụng để cho list khác kết thừa (ở đây là list1).

**VD**:

list = ['A', 'B', 'C']

list.extend('D')

print(list)

# Kết quả: ['A', 'B', 'C', 'D']

list.extend(('E', 'F'))

print(list)

# Kết quả: ['A', 'B', 'C', 'D', 'E', 'F']

Lưu ý: Ở đây mình đã cố tình để ví dụ của phương thức append() và extend() là giống nhau để cho các bạn thấy được sự khác biệt giữa 2 phương thức này (xem kỹ nhé! không nhận ra được thì comment mình sẽ trả lời :D).

## 7, count().

Phương thức này có tác dụng đếm số lần xuất hiện của một thành phần trong list!

**Cú pháp**:

mylist.count(val)

**Trong đó**:

* mylist là list mà các bạn cần kiểm tra.
* val là phần tử mà bạn muốn tìm và đếm trong list mylist.

**VD**:

list = ['A', 'B', 'C']

print(list.count('A'))

# Kết quả: 1

## 8, index().

Phương thức này có tác dụng trả về index xuất hiện đầu tiên của phần tử mà bạn muốn tìm và nếu như không tìm thấy thì nó sẽ gọi [exception](https://toidicode.com/exception-trong-python-356.html).

**Cú Pháp**:

mylist.index(val)

**Trong đó**:

* mylist là list mà các bạn cần kiểm tra.
* val là phần tử mà bạn muốn tìm trong list mylist.

**VD**:

list = ['A', 'B', 'C']

print(list.index('B'))

# Kết quả: 1

print(list.index('D'))

# Kết quả: ValueError: 'D' is not in list

## 9, insert().

Phương thức có tác dụng thêm phần tử vào vị trí index của list, và các phần tử sau index đó sẽ được đẩy về phía sau.

**Cú pháp**:

mylist.insert(index, val)

**Trong đó**:

* mylist là list mà các bạn cần thêm.
* index là vị trí mà bạn muốn thêm phần tử val vào.
* val là phần tử mà bạn muốn thêm vào trong list mylist.

**VD**:

list = ['A', 'B', 'C']

list.insert(0, 'Z')

print(list)

# Kết quả: ['Z', 'A', 'B', 'C']

list.insert(2, 'D')

print(list)

# Kết quả: ['Z', 'A', 'D', 'B', 'C']

## 10, reverse().

Phương thức này có tác dụng đảo ngược vị trí của các phần tử trong list.

**Cú pháp**:

mylist.reverse()

**Trong đó**, mylist là list mà các bạn muốn đảo ngược.

**VD**:

list = ['A', 'B', 'C']

list.reverse()

print(list)

# Kết quả: ['C', 'B', 'A']

## 11, remove().

Phương thức này có tác dụng xóa phần tử khỏi list.

**Cú Pháp**:

mylist.remove(val)

**Trong đó**:

* mylist là list mà các bạn cần xóa phần tử.
* val là phần tử mà bạn muốn muốn xóa trong list mylist.

**VD**:

list = ['A', 'B', 'C']

list.remove('C')

print(list)

# Kết quả: ['A', 'B']

## 12, pop().

Phương thức này có tác dụng xóa bỏ phần tử trong list dựa trên index của nó.

**Cú pháp**:

mylist.pop(index)

**Trong đó**:

* mylist là list mà các bạn cần xóa phần tử.
* index là index của phần tử mà bạn muốn muốn xóa trong list mylist. Mặc định thì index = mylist[-1] (phần tử cuối cùng trong list).

**VD**:

list = ['A', 'B', 'C', 'D', 'E']

list.pop()

print(list)

# Kết quả: ['A', 'B', 'C', 'D']

list.pop(2)

print(list)

# Kết quả: ['A', 'B', 'D']

## 13, sort().

Phương thức này có tác dụng sắp xếp lại các phần tử trong list theo một thứ tự xác định.

**Cú pháp**:

mylist.sort(reverse, key)

**Trong đó**:

* mylist là list mà các bạn muốn sắp xếp.
* reverse là một boolean cấu hình kiểu sắp xếp. Nếu reverse = True thì list sẽ được sắp xếp từ lớn đến bé, nếu reverse = False thì list sẽ được sắp xếp theo thứ tự từ bé đến lớn. Mặc định thì reverse = False.
* key là callback def để xử lý list hoặc là một lamda function (thường được dùng để sắp xếp các list tuple hoặc dictionary).

**VD**:

list = ['A', 'C', 'B', 'E', 'D']

list.sort()

print(list)

# Kết quả: ['A', 'B', 'C', 'D', 'E']

list.sort(reverse=True)

print(list)

# Kết quả: ['E', 'D', 'C', 'B', 'A']

def custom\_sort(elem):

return elem[1]

list = [(1, 2), (5, 7), (7, 100), (4, 4)]

list.sort(key=custom\_sort)

print(list)

# Kết quả: [(1, 2), (4, 4), (5, 7), (7, 100)]

## 14, clear().

Phương thức này có tác dụng xóa bỏ hết tất cả các phần tử trong list.

**Cú pháp**:

mylist.clear()

**Trong đó**,  mylist là list mà bạn muốn xóa bỏ hết phần tử.

**VD**:

list = ['A', 'C', 'B', 'E', 'D']

list.clear()

print(list)

# Kết quả: []

# Tuple trong Python

Tuple trong Python là một kiểu dữ liệu dùng để lưu trữ các đối tượng không thay đổi về sau (giống như hằng số). Còn lại thì cách lưu trữ của nó cũng khá giống như kiểu dữ liệu list mà [bài trước](https://toidicode.com/list-trong-python-346.html) chúng ta đã được tìm hiểu.

Để khai báo một enum thì mọi người sử dụng cú pháp sau:

(val1, val2,.., valn)

**Trong đó,** val1, val2,.., valn là các giá trị của tuple.

Nếu bạn khai báo 1 biến chứa các giá trị mà không được bao quang bởi dấu () thì Python cũng nhận định nó là một tuple (nhưng mình khuyên mọi người lên sử dụng cách đầu tiên cho code được tường minh).

Và nếu như bạn muốn khai báo 1 tuple trống thì bạn chỉ cần khai báo như sau:

a = ();

Còn nếu như tuple của bạn chỉ chứa duy nhất một giá trị thì bắt buộc bạn phải thêm một dấu , nữa đằng sau giá trị đó.

**VD**:

a = (10,)

Để truy cập đến các phần tử trong Tuple thì các bạn thực hiện tương tự như đối với chuỗi và list.

* Các phần tử trong Tuple được đánh dấu từ **0** theo chiều từ trái qua phải.
* Và ngược lại từ **-1** theo chiều từ phải qua trái.

Và nếu như bạn muốn lấy ra một tuple con trong tuple hiện tại thì bạn có thể sử dụng cú pháp sau (giống với list và string):

tupleName[start:end]

**Trong đó**:

* start là vị trí bắt đầu lấy. Nếu để trống start thì nó sẽ lấy từ đầu Tuple.
* end là vị trí kết thúc. Nếu để trống end thì nó sẽ lấy đến hết Tuple.

### Thêm mới phần tử.

Thực ra đây chỉ là cách lách luật thôi, chứ một tuple đã được khai báo thì chúng ta chỉ được gọi và không được sửa đổi hay thêm mới bất cứ một cái gì cả. Nhưng chúng ta có thể tạo ra được một tuple mới từ các tuple đã có bằng biểu thức + hai tuple.

**VD**: Mình sẽ ghép 2 tuple day1 và day2 thành tuple day.

day1 = ('monday', 'tuesday', 'wednesday')

day2 = ('thursday', 'friday', 'saturday' , 'sunday')

day = day1 + day2

print(day)

# ('monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunday')

Cũng giống như list, bạn cũng có thể khai báo các tuple lồng nhau.

**VD**:

day1 = ('monday', 'tuesday', 'wednesday')

day2 = ('thursday', 'friday', 'saturday' , 'sunday', day1)

# day = day1 + day2

print(day2)

# ('thursday', 'friday', 'saturday', 'sunday', ('monday', 'tuesday', 'wednesday'))

print(day2[4][0]) # monday

Và bạn có thể lồng bao nhiêu cấp cũng được. Và lồng bất cứ một kiểu dữ liệu nào cũng ok.

# Dictionary trong Python

iểu dữ liệu dictionary trong Python là một kiểu dữ liệu lưu trữ các giá trị chứa key và value , nhìn một cách tổng quát thì nó giống với **Json**. Và đối với kiểu dữ liệu này thì các giá trị bên trong nó không được sắp xếp theo một trật tự nào cả.

Để khai báo một dictionary chúng ta sử dụng cặp dấu {} theo cú pháp sau:

{key1: value1, key2: value2,..., keyN: valueN}

**Trong đó**, key1: value1, key2: value2,..., keyN: valueN là các key và giá trị của kiểu dữ liệu dictionary. Và tên của key thì các bạn phải tuân thủ theo một số quy tắc sau:

* Các phần tử đều phải có **key**.
* Và **Key** chỉ có thể là số hoặc chuỗi.
* **Key** phải là duy nhất, nếu không nó sẽ nhận giá trị của phần tử có key được xuất hiện cuối cùng.
* **Key** khi đã được khai báo thì không thể đổi được tên.
* **Key** có phân biệt hoa thường.

-Để truy cập đến các phần tử trong dictionary thì các bạn sử dụng cú pháp sau:

dicName[key]

**Trong đó**:

* dicName là tên của của dictionary.
* key là tên của key các bạn muốn lấy ra trong dictionary.

Để xóa một phần tử trong dictionary thì chúng ta sử dụng hàm del và chọn phần tử cần xóa.

Và nếu như bạn muốn xóa tất cả các phần tử bên trong dictionary thì bạn sử dụng phương thức clear theo cú pháp:

dictName.clear();

Trong đó, dictName là dictionary mà bạn muốn xóa hết phần tử.

Cũng giống như kiểu dữ liệu list, tuple thì trong dicrtionary các bạn cũng có thể lồng bất kỳ kiểu dữ liệu nào bạn thích vào trong nó.

**VD**: Mình sẽ lồng một dictionary vào trong dictionary và đồng thời truy vấn luôn đến dictionary con.

person = {

'name': 'Vũ Thanh Tài',

'option': {

'age': 22,

'male': True,

'status': 'alone'

}

}

print(person['option']['age'])

# 22

# Câu lệnh rẽ nhánh trong Python

If-else dịch ra tiếng Việt có nghĩa là **nếu thì** và nó ở trong lập trình cũng thế. Các điều kiện mà mệnh đề if đưa ra trả về là đúng (True) thì nó sẽ thực thi code bên trong mệnh đề if và ngược lại nếu điều kiện đó sai thì nó sẽ thực hiện code trong mệnh đề else.

**Cú pháp của câu lệnh if-else**:

if condition:

#code

else:

#code

**Trong đó**, condition là điều kiện của mệnh đề if.

Trong thực tế thì không phải lúc nào chúng ta cũng có 2 trường hợp nếu thì. Mà có đôi lúc tồn tại vô vàn điều kiện khác nhau và để giải quyết điều này thì trong Python có cung cấp thêm co chúng ta mệnh đề if-elif-else với cú pháp như sau:

if condition:

# code

elif condition2:

# code

elif condition3:

# code

else:

#code

**Trong đó,** thì sau các mệnh đề if và elif sẽ tồn tại các điều kiện condition tương ứng, và nếu như các điều kiện đó đúng thì code ở trong mệnh đề đó sẽ được thực hiện.

Trong python k hỗ trợ switch case

# Vòng lặp trong Python

Vòng lặp for ở trong Python có tác dụng lặp các biến dữ liệu có trong list , tuple hoặc string,... Sử dụng cú pháp như sau:

for variable in data:

# code

**Trong đó**:

* variable là các biến tạm dùng để chứa dữ liệu sau mỗi lần lặp.
* data là một list, tuple hoặc string,... chứa giá trị cần lặp.

Vòng lặp while trong Python dùng để lặp các dữ liệu mà giá trị ngừng có của nó là chưa biết trước.

**Cú pháp sử dụng**:

while condition:

# code

**Trong đó,**condition là điều kiện quyết định vòng lặp while có được chạy hay không. Nếu condition trả về giá trị là True thì vòng lặp while mới được thực thi, và ngược lại thì nó sẽ không thực thi nếu condition trả về False.

hông thường trong một số trường hợp vòng lặp của chúng ta sẽ có thể không cần thực thi code trong một số vòng lặp cụ thể hay là cần nhảy lần vòng lặp đó và thực thi các lần lặp tiếp theo, và để làm được điều đó thì trong Python có hỗ trợ chúng ta 3 keyword để tác động đến vòng lặp là:

* break - break giúp chúng ta chấm dứt vòng lặp tại thời điểm nó xuất hiện và các code cùng cấp phía sau nó sẽ không được thực thi nữa.
* continue - giúp chúng ta nhảy qua lần lặp hiện tại và chuyển đến lần lặp tiếp theo, các code cùng cấp phía sau nó cũng sẽ không được thực hiện.

# Hàm trong Python

Để khai báo một hàm trong Python thì chúng ta sử dụng keyword def với cú pháp như sau:

def ten\_ham(param...):

#code

**Trong đó**:

* ten\_ham là tên của hàm mà bạn muốn đặt. Lưu ý: Tên hàm không được bắt đầu bằng số và không được chứa các ký tự đặc biệt **trừ** ký tự \_
* param... là các tham số bạn muốn truyền vào hàm, nếu không có tham số thì để trống trường này.

Để gọi một hàm đã được khai báo rồi, thì chúng ta sử dụng cú pháp sau:

ten\_ham()

#hoặc

ten\_ham(param...)

**Trong đó**:

* ten\_ham là tên của hàm là chúng ta muốn gọi.
* param... là các tham số chúng ta muốn truyền vào trong hàm.

Python cũng cung cấp cho chúng ta thiết lập luôn giá trị mặc định của tham số khi khai báo hàm. Bằng cách sử dụng dấu = với cú pháp như sau:

def ten\_ham(param = defaultValue):

# code

**Trong đó**:

* defaultValue là giá trị mặc định của tham số đó mà bạn muốn gán.
* Khi một biến được khai báo ở trong hàm thì nó chỉ có thể được sử dụng ở trong hàm đó thôi.
* **VD**:
* def say\_hello():
* a = "Hello"
* print(a)
* print(a)
* # Lỗi: name 'a' is not defined
* Và chúng ta cũng không thể nào thay đổi giá trị của biến (biến bình thường) mà tác động ra ngoài hàm được.
* **VD**:
* a = "Hello Guy!"
* def say(a):
* a = "Toidicode.com"
* print(a)
* say(a)
* # KQ: Toidicode.com
* print(a)
* # KQ: Hello Guy!
* Nhưng nếu như biến mà có kiểu dữ liệu là list thì chúng ta lại có thể là được điều đó.
* **VD**:
* a = [5, 10, 15]
* def change(a):
* a[0] = 1000
* print(a)
* change(a)
* # KQ: [1000, 10, 15]
* print(a)
* # KQ: [1000, 10, 15]
* Ngoài những cách hoạt động của biến mình đã trình bày ở phần 6 ra thì chúng ta còn có 1 cách nữa để có thể tác động đến các biến bên ngoài hàm khi đang ở trong hàm. Đó là sử dụng global variable - biến toàn cầu, khi một biến là global thì chúng ta có thể gọi và tác động đến nó từ bất kỳ đâu trong chương trình.
* Để khai báo một biến là global thì chúng ta chỉ cần thêm keyword global trước tên của nó như sau:
* global tenbien

Trên thực tế, không phải lúc nào chúng ta cũng biết được chính xác số lượng biến truyền vào trong hàm. Chính vì thế trong Python có cũng cấp cho chúng ta khai báo một param đại diện cho các biến truyền vào hàm bằng cách thêm dấu \* vào trước param đó.

Để sử dụng exception trong Python thì các bạn sử dụng lệnh try except theo cú pháp sau:

try:

# code

except exceptionName:

# code

**Trong đó:**

* exceptionName là tên của các exception mà bạn muốn bắt và xử lý (xem các exception ở phần 2)**.**

Nếu khối lệnh trong try có 1 lỗi gì đó xảy ra thì chương trình sẽ tìm đến các except phía dưới và nếu một except nào thỏa mãn thì nó sẽ thực thi code trong khối except đó.

**VD**: Mình sẽ hạn bắt lỗi của ví dụ trên bằng exception.

Như các bạn đã thấy ở dòng báo lỗi trong ví dụ trên nó có đoạn ZeroDivisionError:... thì ở đây nếu như chúng ta không biết hết các exception trong Python thì có thể dựa vào dòng lỗi đó và bắt theo, trong trường hợp này exception được gọi chính là ZeroDivisionError.

def sum(a, b):

return a / b

try :

print(sum(6, 0))

except ZeroDivisionError:

print('Co loi xay ra!')

#ket qua: Co loi xay ra!

Và đối với mỗi khối lệnh except thì bạn cũng có thể bắt nhiều excetion trên một lần khai báo. Bằng việc đặt các exception cách nhau bở một dấu ,

**VD**: Bắt nhiều exception trên một lần khai báo.

try :

# code

except (ZeroDivisionError, RuntimeError):

# code

Hoặc bạn cũng có thể khai báo nhiều except trong một khối lệnh try except.

**VD**: Bắt nhiều exception.

try :

# code

except ZeroDivisionError:

# code

except RuntimeError:

# code

Và đương nhiên bạn cũng có thể lồng các khối try except lại với nhau:

**VD**:

try :

# code

except ZeroDivisionError:

try :

# code

except StandardError:

# code

except RuntimeError:

# code

## 2, Finally.

Nếu như trong khối lệnh try except bạn muốn sẽ có 1 đoạn lệnh**chắc chắn sẽ được thực thi** cho dù try đúng hay sai, thì bạn sẽ phải khai báo thêm khối lệnh finally vào cuối khối lệnh try except theo cú pháp sau:

try:

# code

except:

# code

finally:

# code

Finally trong Python thường được dùng để clear data mà trong quá trình try except tạo ra.

**VD**:

def sum(a, b):

return a / b

try:

print(sum(6, 0))

except ZeroDivisionError:

print('Co loi xay ra!')

finally:

print('finally duoc goi!')

# Ket qua:

# Co loi xay ra!

# finally duoc goi!

## 3, Các exception có sẵn trong Python.

Dưới đây là danh sách các exception mặc định trong Python.

|  |  |
| --- | --- |
| **Exception Name** | **Chú Thích** |
| **Exception** | Đây là lớp cơ sở cho tất cả các exception, nó sẽ xuất hiện khi có bất cứ một lỗi nào xảy ra. |
| **StopIteration** | Xuất hiện khi phương thức next() của interator không trỏ đến một đối tượng nào. |
| **SystemExit** | Xuất hiện khi dùng phương thức sys.exit() |
| **StandardError** | Lớp cơ sở cho tất cả các exception. |
| **ArithmeticError** | Xuất hiện khi có lỗi tính toán giữa các số với nhau |
| **OverflowError** | Xuất hiện khi thực hiện tính toán và giá trị của nó vượt quá ngưỡng giới hạn cho phép của kiểu dữ liệu. |
| **FloatingPointError** | Xuất hiện khi tính toán float thất bại. |
| **ZeroDivisonError** | Xuất hiện khi thực hiện phép chia cho 0. |
| **AssertionError** | Xuất hiện trong trường hợp lệnh assert thất bại. |
| **AttributeError** | Xuất hiện khi không tồn tại thuộc tính này, hoặc thiếu tham số truyền vào nó. |
| **EOFError** | Xuất hiện khi không có dữ liệu từ hàm input() hoặc cuối file. |
| **ImportError** | Xuất hiện khi lệnh import thất bại. |
| **KeyboardInterrupt** | Xuất hiện khi ngắt trình biên dịch. |
| **LookupError** | Lớp cơ sở cho tất cả các lỗi về lookup. |
| **IndexError** | Xuất hiện khi index không tồn tại trong list, string,... |
| **KeyError** | Xuất hiện khi key không tồn tại trong dictionary. |
| **NameError** | Xuất hiện khi một biến không tồn tại trong phạm vi bạn gọi nó. |
| **EnvironmentError** | Xuất hiện khi có bất kỳ một lỗi nào ngoài phạm vị của Python. |
| **IOError** | Xuất hiện khi xử dụng input/ output thất bại, hoặc  mở file không thành công. |
| **OSError** | Xuất hiện khi có lỗi từ hệ điều hành. |
| **SyntaxError** | Xuất hiện khi chương trình có lỗi cú pháp. |
| **IndentationError** | Xuất hiện khi bạn thụt dòng không đúng. |
| **SystemError** | Xuất hiện khi trình biên dịch có vấn đề nhưng mà nó lại không tự động exit. |
| **SystemExit** | Xuất hiện khi trình biên dịch được thoát bởi sys.exit(). |
| **TypeError** | Xuất hiện khi thực thi toán tử hoặc hàm mà kiểu dữ liệu bị sai so với kiểu dữ liệu đã định nghĩa ban đầu. |
| **ValueError** | Xuất hiện khi chúng ta build 1 function mà kiểu dữ liệu đúng nhưng khi chúng ta thiết lập ở tham số là khác so với khi truyền vào. |
| **RuntimeError** | Xuất hiện khi lỗi được sinh ra không thuộc một danh mục nào. |
| **NotImplementedError** | Xuất hiện khi một phương thức trừu tượng cần được thực hiện trong lớp kế thừa chứ không phải là lớp thực thi |
| **UnboundLocalError** | Xuất hiện khi chúng ta cố tình truy cập vào một biến trong hàm hoặc phương thức, nhưng không thiết lập giá trị cho nó. |

Để mở file trong Python chúng ta sử dụng hàm open với cú pháp như sau:

open(filePath, mode, buffer)

**Trong đó**:

* filePath là đường dẫn đến địa chỉ của file.
* mode là thông số thiết lập chế độ chúng ta mở file được cấp những quyền gì? Mặc địn mode sẽ bằng r (xem các mode ở dưới).
* buffer là thông số đệm cho file mặc định thì nó sẽ là 0.

Các chế độ mode.

|  |  |
| --- | --- |
| Mode | Chú thích |
| r | Chế độ chỉ được phép đọc. |
| rb | Chế độ chỉ được phép đọc nhưng cho định dạn nhị phân. |
| r+ | Chế độ này cho phép đọc và ghi file, con trỏ nó sẽ nằm ở đầu file. |
| rb+ | Chế độ này cho phép đọc và ghi file ở dạng nhị phân, con trỏ sẽ nằm ở đầu file. |
| w | Chế độ ghi file, nếu như file không tồn tại thì nó sẽ tạo mới file và ghi nội dung, còn nếu như file đã tồn tại nó sẽ ghi đè nội dung lên file cũ. |
| wb | Tương tự chế độ w nhưng đối với nhị phân. |
| w+ | Mở file trong chế độ đọc và ghi. còn lại như w. |
| wb+ | Giống chế độ w+ nhưng đối với nhị phân |
| a | Mở file trong chế độ ghi tiếp. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung, và nếu như file chưa tồn tại thì nó sẽ tạo một file mới và ghi nội dung vào đó. |
| ab | Tương tự a nhưng đối với nhị phân. |
| a+ | Mở file trong chế độ đọc và ghi tiếp nội dung, còn lại cơ chế giống chế độ a. |
| ab+ | Tương tự chế độ a+ nhưng đối với nhị phân. |

**VD**: Mình sẽ đọc file readme.md có cùng cấp với file index.py của mình.

|-- readme.md

|-- index.py

code file index.py

# index.py

open('readme.md')

Lúc này hàm open sẽ trả về một object chứa các các phươn thức để tác động đến file.

### Đóng file.

Để đóng một file đang được mở, thì chúng ta sử dụng phương thức close() với cú pháp như sau:

fileObject.close()

**Trong đó,**fileObject là đối tượng mà chúng ta thu được khi sử dụng hàm open().

Để đảm bảo quy chế đóng mở và giải phóng bộ nhớ cho chương trình thì các bạn phải luôn nhớ đống file khi kết thúc phiên làm việc.

### Đọc file.

Sau khi đã mở được file ra rồi, để đọc được file thì chúng ta sử dụng phương thức read với cú pháp:

fileObject.read(length);

**Trong đó**:

* fileObject là đối tượng mà chúng ta thu được khi sử dụng hàm open().
* length là dung lượng của dữ liệu mà chúng ta muốn đọc, nếu để trống tham số này thì nó sẽ đọc hết file hoặc nếu file lớn quá thì nó sẽ đọc đến khi giới hạn của bộ nhớ cho phép.

**VD**: Mình sẽ đọc và xuất ra dữ liệu đọc được trong file readme ở trên.

# mo file

file = open('readme.md')

# doc file

data = file.read();

# dong file

file.close()

# in du lieu doc duoc

print(data)

### Ghi file.

Để ghi được file thì bạn phải chắc chắn là đang mở file ở các chế độ cho phép ghi. Và sử dụng phương thức write với cú pháp sau:

fileObject.write(data)

Trong đó:

* fileObject là đối tượng mà chúng ta thu được khi sử dụng hàm open().
* data là dữ liệu mà chúng ta muốn ghi vào trong file.

**VD**:  Mình sẽ ghi dữ liệu vào file readme.md

# mo file o che do ghi

file = open('readme.md','w')

# ghi file

file.write('Vu Thanh Tai - toidicode.com')

# dong file

file.close()

### Các thuộc tính trong file.

Nếu như bạn cần biết thêm các thông số của file hiện tại thì bạn có thể tham khảo thêm các thuộc tính trong object file mà Python đã cung cấp sẵn cho chúng ta. Các thuộc tính như sau:

Trong các trường hợp dưới đây: giả sử file là kết quả thu được từ hàm open().

|  |  |
| --- | --- |
| Thuộc tính | Chú thích |
| file.name | Trả về tên của file đang được mở. |
| file.mode | Trả về chế độ mode của file đang được mở. |
| file.closed | Trả về true nếu file đã được đóng, và false nếu file chưa đóng. |

**VD**: Mình sẽ in ra thông số của file readme.md ở trên.

file = open('readme.md','w')

print( file.name )

# readme

print( file.mode )

# w

print( file.closed )

# False

file.close()