

Bảng cheat Python

chỉ những điều cần bản

Được tạo bởi: Arianne Colton và Sean Chen

tổng quan

- Python phân biệt chữ hoa chữ thường
- Chỉ số Python bắt đầu từ 0
- Python sử dụng khoảng trắng (tab hoặc dấu cách) để thụt lề. mã thay vì sử dụng dấu ngoặc nhọn.

GIÚP ĐỖ

Trợ giúp Trang chủ	trợ giúp()
Chức năng Trợ giúp	trợ giúp(str.replace)
Trợ giúp mô-dun Trợ giúp	(lại)

MÔ-DUN (THƯ VIỆN AKA)

Mô-dun Python chỉ đơn giản là một tệp '.py'

Liệt kê nội dung mô-dun	thư mục(mô-dun1)
mô-dun tải	nhập mô-dun1
Hàm gọi từ Mô-dun module1.func1()	

' Câu lệnh nhập sẽ tạo một không gian tên mới và thực thi tất cả các câu lệnh trong tệp .py được liên kết trong không gian tên đó. Nếu bạn muốn tải nội dung của mô-dun vào không gian tên hiện tại, hãy sử dụng 'from module1 import * '

Các loại vô hướng

Kiểm tra kiểu dữ liệu: type(variable)

SÁU LOẠI DỮ LIỆU THƯỜNG ĐƯỢC SỬ DỤNG

- int/long* - int lớn tự động chuyển thành long
- float* - 64 bit, không có loại 'double'
- bool* - Đúng hay Sai
- str* - Giá trị ASCII trong Python 2.x và Unicode trong Python 3
 - Chuỗi có thể ở trong dấu ngoặc đơn/đối/ba
 - Chuỗi là một chuỗi các ký tự nên có thể được xử lý như các chuỗi ký tự khác.
 - Ký tự đặc biệt có thể được thực hiện thông qua \ hoặc lời nói đầu Với r

```
str1 = r'this\?ff'
```
 - Định dạng chuỗi có thể được thực hiện theo một số cách

```
template = '%.2f %s haha %d'; str1 = mẫu % (4.88, 'hola', 2)
```

Các loại vô hướng

* str(), bool(), int() và float() cũng là các hàm truyền kiểu rõ ràng.

5. NoneType(None) - Giá trị 'null' của Python (Chỉ tồn tại một phiên bản của đối tượng Không tồn tại)

- None không phải là từ khóa dành riêng mà là một phiên bản duy nhất của 'NoneType'
- Không có giá trị mặc định chung cho các đối số hàm tùy chọn:

```
def func1(a, b, c = Không)
```
- Cách dùng phổ biến của None :

```
nếu biến là None :
```

6. datetime - mô-dun 'datetime' python tích hợp cung cấp các loại 'datetime', 'date', 'time'.

- 'datetime' kết hợp thông tin được lưu trữ trong 'ngày' và 'thời gian'

Tạo ngày giờ từ Chuỗi	dt1 = ngày giờ. strptime('20091031', '%Y%m%d')
Nhận đối tượng 'ngày'	dt1.date()
Nhận đối tượng 'thời gian'	dt1.time()
Định dạng ngày giờ thành chuỗi	dt1.strftime('%m/%d/%Y %H%M')
Thay đổi trường giá trị	dt2 = dt1.replace(phút = 0, giây = 30)
Nhận sự khác biệt	khác biệt = dt1 - dt2 # diff là đối tượng 'datetime.timedelta'

Lưu ý: Hầu hết các đối tượng trong Python đều có thể thay đổi được ngoại trừ 'chuỗi' và 'bộ dữ liệu'

Cấu trúc dữ liệu

Lưu ý: Tất cả lệnh gọi hàm không nhận, tức là list1.sort() các ví dụ bên dưới là các thao tác tại chỗ (không tạo đối tượng mới) trừ khi có ghi chú khác.

TUPLE

Chuỗi một chiều, có độ dài cố định, không thay đổi của các đối tượng Python thuộc bất kỳ loại nào.

Cấu trúc dữ liệu

Tạo bộ dữ liệu	tup1 = 4, 5, 6 hoặc tup1 = (6,7,8)
Tạo Tuple lồng nhau	tup1 = (4,5,6), (7,8)
Chuyển đổi chuỗi hoặc Trình vòng lặp nối Tuple	bộ dữ liệu([1, 0, 2])
Ghép các bộ dữ liệu	tup1 + tup2
Giải nén Tuple a, b, c	= tup1

Ứng dụng của Tuple

Hoán đổi biến	b, a = a, b
---------------	-------------

DANH SÁCH

Chuỗi một chiều, có độ dài thay đổi, có thể thay đổi (tức là nội dung có thể được sửa đổi) của các đối tượng Python thuộc bất kỳ loại nào.

Tạo danh sách	list1 = [1, 'a', 3] hoặc list1 = list(tup1)
Danh sách nối *	list1 + list2 hoặc list1.extend(list2)
Nối vào cuối danh sách	list1.append('b')
Chèn vào danh sách cụ thể	list1.insert(posIdx, Vị trí 'b')
Nghịch đảo của chèn	valueAtIdx = list1.pop(posIdx)
Xóa giá trị đầu tiên khỏi danh sách	list1.remove('a')
Kiểm tra tư cách thành viên	3 trong danh sách1 => Đúng
Sắp xếp danh sách	list1.sort()
Sắp xếp với người dùng-Chức năng cung cấp	list1.sort(key = len) # sắp xếp theo độ dài

Việc nối danh sách bằng cách sử dụng '+' rất tốn kém vì phải tạo một danh sách mới và sao chép các đối tượng. Vì vậy, mở rộng() là thích hợp hơn.

Chèn đất về mặt tính toán so với nối thêm.

*** Việc kiểm tra xem danh sách có chứa một giá trị chậm hơn rất nhiều so với dicts và set hay không khi Python thực hiện quét tuyến tính trong đó các danh sách khác (dựa trên bảng băm) trong thời gian không đổi.

Chèn đất về mặt tính toán so với nối thêm.

Việc nối danh sách bằng cách sử dụng '+' rất tốn kém vì phải tạo một danh sách mới và sao chép các đối tượng. Vì vậy, mở rộng() là thích hợp hơn.

Tích hợp 'mô-dun chia đôi'

- Thực hiện tìm kiếm nhị phân và chèn vào danh sách đã sắp xếp.
- 'bisect.bisect' tìm vị trí ở đó 'bisect.insort' thực sự chèn vào vị trí đó.

† CẢNH BÁO: các hàm mô-dun chia đôi không kiểm tra xem danh sách đã được sắp xếp hay chưa, làm như vậy sẽ tốn kém về mặt tính toán. Vì vậy, sử dụng chúng trong danh sách chưa sắp xếp sẽ thành công mà không gặp lỗi nhưng có thể dẫn đến kết quả không chính xác.

CẮT CHO LOẠI TRÌNH TỰ

† Các loại trình tự bao gồm 'str', 'array', 'tuple', 'list', v.v.

Danh sách	ký hiệu1(bắt đầu:dừng]
	list1(bắt đầu:dừng[bước
	(thứ bước này được sử dụng) 5

Cấu trúc dữ liệu

Ghi chú:

- Chỉ mục 'bắt đầu' được bao gồm, nhưng chỉ mục 'dừng' thì KHÔNG.
- bắt đầu/dừng có thể được bỏ qua khi mặc định là bắt đầu/kết thúc.

Áp dụng 'bước':

Lấy mọi yếu tố khác	danh sách1[:2]
Đảo ngược một chuỗi	str1[::-1]

DICTIONARY (Bản đồ băm)

Tạo lệnh	dict1 = {'key1': 'value1', 2 :3, 2}
Tạo Dict từ sự liên tiếp	dict(zip(keyList, valueList))
Nhận/Đặt/Chèn phần tử	dict1['key1']* dict1['key1'] = 'newValue'
Nhận với giá trị mặc định	dict1.get('key1', giá trị mặc định)
Kiểm tra xem khóa có tồn tại không	'key1' trong dict1
Xóa phần tử	del dict1['key1']
Nhận danh sách khóa	dict1.keys()
Nhận danh sách giá trị	dict1.values()
Cập nhật giá trị	dict1.update(dict2) # giá trị dict1 được thay thế bằng dict2

Ngoại lệ 'KeyError' nếu khóa không tồn tại.

get() theo mặc định (còn gọi là không có 'default(value') sẽ trả về 'Không' nếu khóa không tồn tại.

Trả về danh sách các khóa và giá trị theo cùng thứ tự. Tuy nhiên, thứ tự không phải là thứ tự cụ thể nào, hay còn gọi là nó rất có thể không được sắp xếp.

Các loại khóa chính tả hợp lệ

- Khóa phải bất biến như kiểu vô hướng (int, float, string) hoặc bộ dữ liệu (tất cả các đối tượng trong bộ dữ liệu cũng cần phải bất biến)
- Thuật ngữ kỹ thuật ở đây là 'khả năng băm', kiểm tra xem một đối tượng có thể băm được bằng hash('this is string'), hash([1, 2]) - việc này sẽ thất bại.

BỘ

- Một tập hợp là một tập hợp không có thứ tự của các phần tử ĐỘC ĐÁO.
- Bạn có thể coi chúng giống như các lệnh nhưng chỉ là các phim.

Tạo bộ	set([3, 6, 3]) hoặc {3, 6, 3}
Tập hợp con kiểm tra	set1.issubset (set2)
Thử nghiệm siêu tập	set1.issuperset (set2)
Bộ kiểm tra có nội dung giống nhau	tập1 == tập2

- Thiết lập các thao tác:

Liên minh (còn gọi là 'hoặc')	tập1 tập2
Giao lộ (còn gọi là 'và')	tập1 & tập2
Sự khác biệt	tập1 - tập2
Sự khác biệt đối xứng (còn gọi là 'xor')	set1 ^ tập2

chức năng

Python được truyền bằng tham chiếu, các đối số của hàm được truyền bằng tham chiếu.

- Hình thức cơ bản:

```
def func1(posArg1, keywordsArg1 = 1, ..):
```

Ghi chú:

- Đối số từ khóa PHẢI tuân theo đối số vị trí.
- Python theo mặc định KHÔNG phải là "đánh giá lười biếng", các biểu thức được đánh giá ngay lập tức.

- Cơ chế gọi hàm:

- Tất cả các chức năng đều cục bộ trong phạm vi cấp độ mô-đun. Xem phần "Mô-đun".
- Bên trong, các đối số được đóng gói thành một bộ và dict, hàm nhận một bộ 'args' và dict 'kwargs' rồi giải nén bên trong.

- Cách sử dụng phổ biến của "Hàm là đối tượng":

```
def func1(ops = [str.strip, user_ định_func, ..], ..): cho hàm trong ops: value = function(value)
```

GIÁ TRỊ TRẢ LẠI

- Không có giá trị nào được trả về nếu kết thúc hàm mà không gặp câu lệnh return.
- Trả về nhiều giá trị thông qua MỘT đối tượng bộ dữ liệu

```
trả về (giá trị1, giá trị2) giá trị1, giá trị2 = func1(..)
```

CHỨC NĂNG Nặc Danh (AKA LAMBDA)

- Chức năng ẩn danh là gì? Một hàm đơn giản bao gồm một câu lệnh.

```
lambda x : x * 2 # def func1(x): trả về x * 2
```

- Ứng dụng hàm lambda: 'curring' hay còn gọi là lấy các hàm mới từ hàm hiện có bằng ứng dụng đối số từng phần.

```
ma60 = lambda x : pd.rolling_mean(x, 60)
```

CHỨC NĂNG HỮU ÍCH (DÀNH CHO CẤU TRÚC DỮ LIỆU)

- Enumerate trả về một chuỗi các bộ giá trị (i, value) trong đó tối thiểu là chỉ mục của mục hiện tại.

```
đối với i, giá trị trong liệt kê (bộ sưu tập):
```

- Ứng dụng: Tạo ảnh xạ chính tả của giá trị của một chuỗi (được coi là duy nhất) tới các vị trí của chúng trong chuỗi.

- Sorted trả về một danh sách được sắp xếp mới từ bất kỳ dãy nào được sắp xếp([2, 1, 3]) => [1, 2, 3]

- Ứng dụng:

```
được sắp xếp(set('abc bcd')) => [' ', 'a', 'b', 'c', 'd'] # trả về các ký tự duy nhất được sắp xếp
```

- Zip ghép nối các phần tử của một số danh sách, bộ dữ liệu hoặc các chuỗi khác để tạo thành danh sách bộ dữ liệu:

```
zip(seq1, seq2) => [('seq1_1', 'seq2_1'), (..), ..]
```

- Zip có thể lấy số dãy tùy ý. Tuy nhiên, số phần tử nó tạo ra được xác định bởi chuỗi "ngắn nhất".

- Ứng dụng : Lập đồng thời trên nhiều trình tự:

```
với tôi, (a, b) trong liệt kê(zip(seq1, seq2)):
```

- Giải nén - một cách khác để nghĩ về điều này là chuyển đổi danh sách các hàng thành danh sách các cột.

```
seq1, seq2 = zip(*zipOutput)
```

- Đảo ngược lặp lại các phần tử của chuỗi theo thứ tự ngược lại.

```
danh sách (đảo ngược (phạm vi (10))) '
```

* đảo ngược() trả về trình vòng lặp, list() biến nó thành một danh sách.

Kiểm soát và dòng chảy

- Toán tử cho điều kiện trong 'if else':

Kiểm tra xem hai biến có phải là cùng một đối tượng không	var1 là var2
, , , là đối tượng khác nhau Kiểm tra xem hai biến có cùng giá trị không	var1 không phải là var2 var1 == var2

CẢNH BÁO: Sử dụng toán tử 'và', 'hoặc', 'không' cho các điều kiện phức hợp, không phải &&, ||, !.

- Cách sử dụng phổ biến của toán tử 'for':

Lặp lại một bộ sưu tập (tức là danh sách hoặc bộ dữ liệu) hoặc một trình vòng lặp. , , Nếu các phần tử là các chuỗi, có thể	cho phần tử trong iterator: cho a, b, c trong iterator:
---	---

được "giải nén" 3. 'pass' - câu lệnh no-op. Được sử dụng trong các khối không có hành động nào được thực hiện.

- Biểu thức bậc ba - hay còn gọi là "nếu khác" ít dài dòng hơn

- Hình thức cơ bản:

```
value = true-expr nếu điều kiện khác false-expr
```

- Không có câu lệnh switch/case, thay vào đó hãy sử dụng if/elif.

Hướng đối tượng Lập trình

- "object" là gốc của mọi loại Python
- Mọi thứ (số, chuỗi, hàm, lớp, mô-đun, v.v.) đều là một đối tượng, mỗi đối tượng có một "loại". Biến đối tượng là một con trỏ tới vị trí của nó trong bộ nhớ.
- Tất cả các đối tượng đều được tính tham chiếu.

```
sys.getrefcount(5) => x a = 5, b = a # Điều này tạo ra một 'tham chiếu' đến đối tượng ở bên phải của =, do đó cả a và b đều trỏ đến 5 sys.getrefcount(5) => x + 2 del(a); sys.getrefcount(5) => x + 1
```

- Mẫu đơn cơ bản của lớp :

```
lớp MyObject(đối tượng): # 'self' tương đương với 'this' trong Java/C++ def __init__(bản thân, tên): self.name = tên def memberFunc1(self, arg1): @staticmethod def lớpFunc2(arg1): obj1 = MyObject('name1') obj1.memberFunc1('a') MyObject.classFunc2('b')
```

- Công cụ tương tác hữu ích:

```
dir(variable1) # liệt kê tất cả các phương thức có sẵn trên đối tượng
```

Chuỗi chung hoạt động

Nối Danh sách/bộ dữ liệu với Đầu phần cách	' , '.join(['v1', 'v2', 'v3']) => 'v1, v2, v3'
Chuỗi định dạng	string1 = 'Tên tôi là {0} {name}' newString1 = chuỗi1. format('Sean', name = 'Chen')
Tách chuỗi	tháng chín = '-'; chuỗi1list1 = string1.split(tháng 9)
Nhận chuỗi con bắt đầu = 1; chuỗi1[bắt đầu:8]	
Đệm chuỗi với số không	tháng = '5'; tháng.zfill(2) => '05' tháng = '12'; tháng.zfill(2) => '12'

Xử lý ngoại lệ

- Mẫu cơ bản:

```
thứ: ngoại trừ ValueError là e: ngoại trừ (TypeError, AnotherError): ngoại trừ: Cuối cùng: # dọn dẹp, ví dụ đóng db
```

- Đưa ra ngoại lệ theo cách thủ công

```
raise AssertionError # xác nhận không thành công raise SystemExit # yêu cầu thoát chương trình raise RuntimeError(' Thông báo lỗi: ..')
```

Danh sách, Bộ và Dict Hiểu biết

Đường cú pháp giúp mã dễ đọc và viết hơn

- Hiểu danh sách

- Tạo một danh sách mới một cách chính xác bằng cách lọc các phần tử của một bộ sưu tập và chuyển đổi các phần tử đi qua bộ lọc thành một biểu thức ngắn gọn.
- Hình thức cơ bản:

```
[expr forval incollection ifcondition]
```

Một phím tắt cho:

```
kết quả = [] cho val trong bộ sưu tập: nếu điều kiện: result.append(expr)
```

Điều kiện lọc có thể được bỏ qua, chỉ để lại biểu thức.

- Đọc chính tả • Dạng cơ bản :

```
{key-expr : value-expr cho giá trị trong bộ sưu tập nếu có điều kiện}
```

- Đặt mức độ hiểu

- Dạng cơ bản: giống như List Comprehension ngoại trừ dấu ngoặc nhọn thay vì []

- Hiểu về danh sách lồng nhau

- Hình thức cơ bản:

```
[expr cho val trong bộ sưu tập cho InnerVal trong val if condition]
```