



LOGO

NHẬP MÔN LẬP TRÌNH CHO KHOA HỌC DỮ LIỆU

Bài 4. Xử lý dữ liệu trong Python

Nội dung

1

Kiểu dữ liệu tuần tự (sequential datatype)

2

String (chuỗi)

3

List (danh sách)

4

Tuple (hàng)

5

Range (miền)

Kiểu dữ liệu tuần tự (sequential datatype)

- Kiểu dữ liệu tuần tự: kiểu dữ liệu chứa bên trong nó các dữ liệu con nhỏ hơn và thường được xử lý bằng cách lấy ra từng phần-tử-một (bằng vòng for)
 - Các kiểu dữ liệu chứa bên trong nó các dữ liệu nhỏ hơn thường được gọi là các container (bộ chứa)
 - Khái niệm “tuần tự” nhấn vào việc xử lý từng phần tử một, nhưng không nhất thiết đây là cách xử lý duy nhất
- Có 3 kiểu tuần tự thông dụng là **list**, **tuple** và **range**
- Có nhiều kiểu khác như **string**, **bytes**, **bytearray**,... hoặc các lập trình viên có thể tự tạo kiểu riêng theo nhu cầu

String (chuỗi)

- Một chuỗi được xem như một hàng (tuple) các chuỗi con độ dài 1
 - Trong python không có kiểu kí tự (character)
 - Nội dung của chuỗi không thay đổi được, khi ghép thêm nội dung vào chuỗi thực chất là tạo ra chuỗi mới
- Hàm `len(s)` trả về độ dài (số chữ) của s
- Phép toán với chuỗi:
 - Phép nối chuỗi (+): `s="Good" + " " + "Morning!"`
 - Phép nhân bản (*): `s="AB" * 3` # số nguyên
 - Kiểm tra nội dung: `sin '1ABABABCD'` # True

String (chuỗi)

Chỉ mục trong chuỗi

- Các phần tử (các chữ) trong chuỗi được đánh số thứ tự và có thể truy cập vào từng phần tử theo chỉ số. Python duy trì 2 cách đánh chỉ mục khác nhau:
 - Đánh từ trái qua phải: chỉ số đánh từ 0 trở đi cho đến cuối chuỗi
 - Đánh từ phải qua trái: chỉ số đánh từ -1 giảm dần về đầu chuỗi

Đ	H	T	H	U	Y	L	O	I
0	1	2	3	4	5	6	7	8
-9	-8	-7	-6	-5	-4	-3	-2	-1

String (chuỗi)

Cắt chuỗi

- Dựa trên chỉ mục, phép cắt chuỗi cho phép lấy nội dung bên trong của chuỗi bằng cú pháp như sau
 - `<chuỗi>[vị trí A : vị trí B]`
 - `<chuỗi>[vị trí A : vị trí B : bước nhảy]`
- Giải thích:
 - Tạo chuỗi con bắt đầu từ `<vị-trí-A>` đến trước `<vị-trí-B>`
 - Tức là chuỗi con sẽ không gồm vị trí B
 - Nếu không ghi `<vị-trí-A>` thì mặc định là lấy từ đầu
 - Nếu không ghi `<vị-trí-B>` thì mặc định là đến hết chuỗi
 - Nếu không ghi `<bước-nhảy>` thì mặc định bước là 1
 - Nếu `<bước-nhảy>` giá trị âm thì sẽ nhận chuỗi ngược lại

String (chuỗi)

Cắt chuỗi

```
s = '0123456789'
```

```
print(s[3:6])
```

```
# 345
```

```
print(s[3:])
```

```
# 3456789
```

```
print(s[:6])
```

```
# 012345
```

```
print(s[-7:-4])
```

```
# 345
```

```
print(s[-4:-7])
```

```
#
```

```
print(s[-4:-7:-1])
```

```
# 654
```

```
print(s[:len(s)])
```

```
# 0123456789
```

```
print(s[:len(s)-1])
```

```
# 012345678
```

```
print(s[:])
```

```
# 0123456789
```

```
print(s[len(s)::-1])
```

```
# 9876543210
```

```
print(s[len(s)-1::-1])
```

```
# 9876543210
```

```
print(s[len(s)-2::-1])
```

```
# 876543210
```


String (chuỗi)

Định dạng chuỗi

- Dùng toán tử %: <chuỗi> %(<các tham số>)
 - Bên trong <chuỗi> có các kí hiệu đánh dấu nơi đặt lần lượt các tham số
 - Nếu đánh dấu %s: thay thế bằng tham số dạng chuỗi
 - Nếu đánh dấu %d: thay thế bằng tham số dạng nguyên
 - Nếu đánh dấu %f: thay thế bằng tham số dạng thực

- Ví dụ:

"Chao %s, gio la %d gio" %('txnam', 10)

"Can bac 2 cua 2 = %f" %(2**0.5)

"Can bac 2 cua 2 = %10.3f" %(2**0.5)

"Can bac 2 cua 2 = %10f" %(2**0.5)

"Can bac 2 cua 2 = %.7f" %(2**0.5)

String (chuỗi)

Định dạng chuỗi

- Dùng toán tử %: <chuỗi> %(<các tham số>)
 - Bên trong <chuỗi> có các kí hiệu đánh dấu nơi đặt lần lượt các tham số
 - Nếu đánh dấu %s: thay thế bằng tham số dạng chuỗi
 - Nếu đánh dấu %d: thay thế bằng tham số dạng nguyên
 - Nếu đánh dấu %f: thay thế bằng tham số dạng thực

- Ví dụ:

```
"Chao %s, gio la %d gio" %('txnam', 10)
```

```
"Can bac 2 cua 2 = %f" %(2**0.5)
```

```
"Can bac 2 cua 2 = %10.3f" %(2**0.5)
```

```
"Can bac 2 cua 2 = %10f" %(2**0.5)
```

```
"Can bac 2 cua 2 = %.7f" %(2**0.5)
```

String (chuỗi)

Định dạng chuỗi

- Python cho phép định dạng chuỗi ở dạng f-string

```
myname = 'DHTL'
```

```
s = f'This is {myname}.'
```

```
# 'This is DHTL.'
```

```
w = f'{s} {myname}'
```

```
# 'This is DHTL. DHTL'
```

```
z = f'{{s}} {s}'
```

```
# '{s} This is DHTL.'
```

- Mạnh mẽ nhất là định dạng bằng format

```
# điền lần lượt từng giá trị vào giữa cặp ngoặc nhọn
```

```
'a: {}, b: {}, c: {}'.format(1, 2, 3)
```

```
# điền nhưng không lần lượt
```

```
'a: {1}, b: {2}, c: {0}'.format('one', 'two', 'three')
```

```
'two same values: {0},
```

```
{0}'.format(1, 2)
```

```
# điền và chỉ định từng giá trị
```

```
10 '1: {one}, 2: {two}'.format(one=111, two=222)
```

String (chuỗi)

Định dạng chuỗi

- Định dạng bằng format cho phép căn lề phong phú

căn giữa: 'aaaa'

'{: ^10}'.format('aaaa')

căn lề trái: 'aaaa'

'{: <10}'.format('aaaa')

căn lề phải 'aaaa'

'{: >10}'.format('aaaa')

căn lề phải, thay khoảng trắng bằng -: '-----aaaa'

'{: ->10}'.format('aaaa')

căn lề trái, thay khoảng trắng *: 'aaa*****'

'{: * <10}'.format('aaaa')

căn giữa, thay khoảng trắng bằng +: '+++aaaa+++'

'{: + ^10}'.format('aaaa')

String (chuỗi)

Các phương thức chuỗi

- Các phương thức chỉnh dạng
 - `capitalize()`: viết hoa chữ cái đầu, còn lại viết thường
 - `upper()`: chuyển hết thành chữ hoa
 - `lower()`: chuyển hết thành chữ thường
 - `swapcase()`: chữ thường thành hoa và ngược lại
 - `title()`: chữ đầu của mỗi từ viết hoa, còn lại viết thường
- Các phương thức căn lề
 - `center(width [,fillchar])`: căn lề giữa với độ dài width
 - `rjust(width [,fillchar])`: căn lề phải
 - `ljust(width [,fillchar])`: căn lề trái

String (chuỗi)

Các phương thức chuỗi

- Các phương thức chỉnh dạng
 - `capitalize()`: viết hoa chữ cái đầu, còn lại viết thường
 - `upper()`: chuyển hết thành chữ hoa
 - `lower()`: chuyển hết thành chữ thường
 - `swapcase()`: chữ thường thành hoa và ngược lại
 - `title()`: chữ đầu của mỗi từ viết hoa, còn lại viết thường
- Các phương thức căn lề
 - `center(width [,fillchar])`: căn lề giữa với độ dài width
 - `rjust(width [,fillchar])`: căn lề phải
 - `ljust(width [,fillchar])`: căn lề trái

String (chuỗi)

Các phương thức chuỗi

- Các phương thức chỉnh dạng
 - `capitalize()`: viết hoa chữ cái đầu, còn lại viết thường
 - `upper()`: chuyển hết thành chữ hoa
 - `lower()`: chuyển hết thành chữ thường
 - `swapcase()`: chữ thường thành hoa và ngược lại
 - `title()`: chữ đầu của mỗi từ viết hoa, còn lại viết thường
- Các phương thức căn lề
 - `center(width [,fillchar])`: căn lề giữa với độ dài width
 - `rjust(width [,fillchar])`: căn lề phải
 - `ljust(width [,fillchar])`: căn lề trái

List (danh sách)

- List = dãy các đối tượng (một loại array đa năng)
- Khai báo trực tiếp bằng cách liệt kê các phần tử con đặt trong cặp ngoặc vuông ([]), ngăn cách bởi phẩy

[1, 2, 3, 4, 5] # list 5 số nguyên

['a', 'b', 'c', 'd'] # list 4 chuỗi

[[1, 2], [3, 4]] # list 2 list con

[1, 'one', [2, 'two']] # list hỗn hợp

[] # list rỗng

- Kiểu chuỗi (str) trong python có thể xem như một list đặc biệt, bên trong gồm toàn các str độ dài 1

List (danh sách)

Khởi tạo danh sách

- Tạo list bằng constructor

```
l1 = list([1, 2, 3, 4]) # list 4 số nguyên
l2 = list('a bc')       # list 3 chuỗi con
l3 = list()             # list rỗng
```

- Tạo list bằng listcomprehension

```
# list 1000 số nguyên từ 0 đến 999
X = [n for n in range(1000)]
# list gồm 10 listcon là các cặp [x, x2]
# với x chạy từ 0 đến 9
Y = [[x, x*x] for x in range(10)]
```

List (danh sách)

Phép toán

- Giữa list và str có sự tương đồng nhất định
 - List cũng hỗ trợ 3 phép toán: ghép nối (+), nhân bản (*) và kiểm tra nội dung (in)
 - List sử dụng hệ thống chỉ mục và các phép cắt phần con tương tự như str
- Điểm khác biệt là nội dung của list có thể thay đổi

```
l1 = list([1, 2, 3, 4])
```

```
l1[-1] = list('abc')
```

```
print(l1)
```

```
# [1, 2, 3, ['a', 'b', 'c']]
```

List (danh sách)

Chỉ mục

- Chỉ mục của danh sách bắt đầu từ 0 đến $n-1$
- Truy cập đến các phần tử:

`<ten_list>[index]`

- List không lưu trữ các phần tử một cách trực tiếp tại chỉ mục. Sự thực là một tham chiếu được lưu trữ tại mỗi chỉ mục mà tham chiếu tới đối tượng được lưu trữ ở đâu đó trong bộ nhớ. Điều này là do một số đối tượng có thể lớn hơn một số đối tượng khác và vì thế chúng được lưu trữ tại một vị trí bộ nhớ khác.

List (danh sách)

Chỉ mục

- Ví dụ 4.1:

```
list1 = ['vatly', 'hoahoc', 1997, 2000];
```

```
list2 = [1, 2, 3, 4, 5, 6, 7];
```

```
print ("list1[0]: ", list1[0])    # list1[0]: vatly
```

```
print ("list2[1:5]: ", list2[1:5]) # list2[1:5]: [2, 3, 4, 5]
```

- Gán giá trị cho phần tử: <ten_list>[index]=<giatri>

- Ví dụ 4.2:

```
list = ['vatly', 'hoahoc', 1997, 2000];
```

```
print ("Giá trị có sẵn tại chỉ mục thu 2 : ",list[2])
```

```
list[2] = 2001;
```

```
print ("Giá trị mới tại chỉ mục thu 2 : ",list[2])
```

List (danh sách)

Phương thức

- Một số phương thức thường hay sử dụng
 - `count(sub, [start, [end]])`: đếm số lần xuất hiện của sub
 - `index(sub[, start[, end]])`: tìm vị trí xuất hiện của sub, trả về `ValueError` nếu không tìm thấy
 - `clear()`: xóa trắng list
 - `append(x)`: thêm x vào cuối list
 - `extend(x)`: thêm các phần tử của x vào cuối list
 - `insert(p, x)`: chèn x vào vị trí p trong list
 - `pop(p)`: bỏ phần tử thứ p ra khỏi list (trả về giá trị của phần tử đó), nếu không chỉ định p thì lấy phần tử cuối

List (danh sách)

Phương thức

- Một số phương thức thường hay sử dụng
 - `copy()`: tạo bản sao của list (tương tự `list[:]`)
 - `remove(x)`: bỏ phần tử đầu tiên trong list có giá trị x, báo lỗi `ValueError` nếu không tìm thấy
 - `reverse()`: đảo ngược các phần tử trong list
 - `sort(key=None, reverse=False)`: mặc định là sắp xếp các phần tử từ bé đến lớn trong list bằng cách so sánh trực tiếp giá trị

```
x = "Trương Xuân Nam".split()
x.sort(key=str.lower)
print(x)
```

List (danh sách)

- Ví dụ 4.3: Thêm 1 phần tử vào list

```
list1=[10,"hoang",'z']
```

```
print ("Cac phan tu cua List la: ")
```

```
print (list1)
```

```
list1.append(10.45)
```

```
print ("Cac phan tu cua List sau khi phu them la: ")
```

```
print (list1)
```

- Ví dụ 4.4: Xóa 1 phần tử list

```
list1 = ['vatly', 'hoahoc', 1997, 2000];
```

```
print (list1)
```

```
del (list1[2]);
```

```
print ("Cac phan tu cua List sau khi xoa gia tri tai chi muc 2 : ",list1)
```


List (danh sách)

Hàm

Hàm	Miêu tả
<code>len(list)</code>	Trả về độ dài của list
<code>max(list)</code>	Trả về phần tử có giá trị lớn nhất trong list
<code>min(list)</code>	Trả về phần tử có giá trị nhỏ nhất trong list
<code>list(seq)</code>	Chuyển đổi một tuple thành list

Tuple (hàng)

Tạo Tuple

- Tuple = dãy các đối tượng (list), nhưng không thể bị thay đổi giá trị trong quá trình tính toán
- Như vậy str giống tuple nhiều hơn list
- Khai báo trực tiếp bằng cách liệt kê các phần tử con đặt trong cặp ngoặc tròn (), ngăn cách bởi dấu phẩy

(1, 2, 3, 4, 5)

tuple 5 số nguyên

('a', 'b', 'c', 'd')

tuple 4 chuỗi

(1, 'one', [2, 'two'])

tuple hỗn hợp

(1,)

tuple 1 phần tử

()

tuple rỗng

Tuple (hàng)

Phân biệt Tuple với List

- Tuple hỗ trợ 3 phép toán: +, *, in
- Tuple cho phép sử dụng chỉ mục giống List
- Các phương thức thường dùng của tuple
 - `count(v)`: đếm số lần xuất hiện của v trong tuple
 - `index(sub[, start[, end]])`: tương tự như str và list
- Tuple khác list ở điểm nào?
 - Chiếm ít bộ nhớ hơn
 - Nhanh hơn

Range (miền)

- Chúng ta đã làm quen với range khi dùng vòng for
 - `range(stop)`: tạo miền từ 0 đến `stop-1`
 - `range(start, stop[, step])`: tạo miền từ `start` đến `stop-1`, với bước nhảy là `step`
 - Nếu không chỉ định thì `step = 1`
 - Nếu `step` là số âm sẽ tạo miền đếm giảm dần (`start > stop`)
- Vậy range khác gì một tuple đặc biệt
 - Range chỉ chứa số nguyên
 - Range nhanh hơn rất nhiều
 - Range chiếm ít bộ nhớ hơn
 - Range vẫn hỗ trợ chỉ mục và cắt (nhưng khá đặc biệt)

LOGO

CẢM ƠN!