



Lập trình Python

Bài 4: Kiểu tuần tự trong python, phần 1

Tài liệu này phân phối dưới giấy phép Creative Commons Attribution 4.0
(bất kỳ ai cũng đều có quyền tự do sử dụng, chia sẻ, sao chép, phân phối, phân phối lại, áp dụng, trích xuất, tùy biến, mở rộng, thương mại hóa,... miễn là ghi nhận công của các tác giả ban đầu của tài liệu)



Tóm tắt nội dung bài trước

- Python chỉ có 2 loại vòng lặp là **while** và **for** (thay vì 3 loại như C/C++)
- Lặp **while** thực hiện lặp chừng nào điều kiện còn đúng
- Lặp **for** cho phép duyệt từng phần tử của một danh sách
- Hàm **range** hỗ trợ tạo các miền số nguyên để phép duyệt bằng for trở nên ngắn gọn hơn
- Cả **while** và **for** đều hỗ trợ các cú pháp mở rộng:
 - Lệnh **break** để kết thúc vòng lặp hiện tại
 - Lệnh **continue** để bắt đầu một vòng lặp mới
 - Khối **else** được thực hiện nếu vòng lặp kết thúc không do break



Nội dung

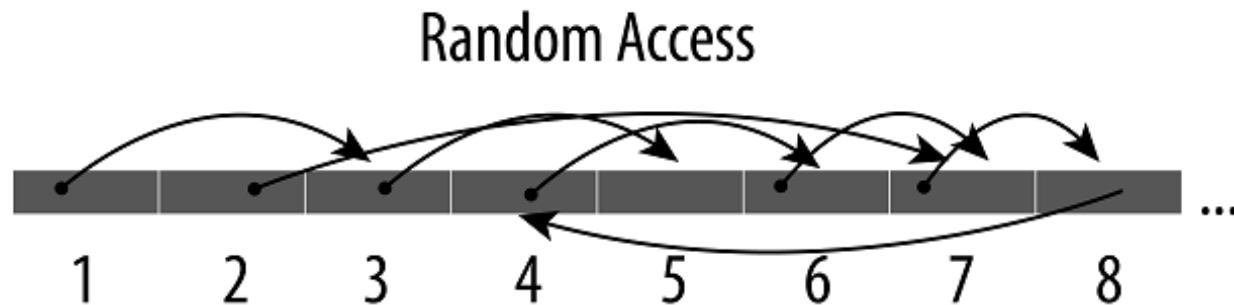
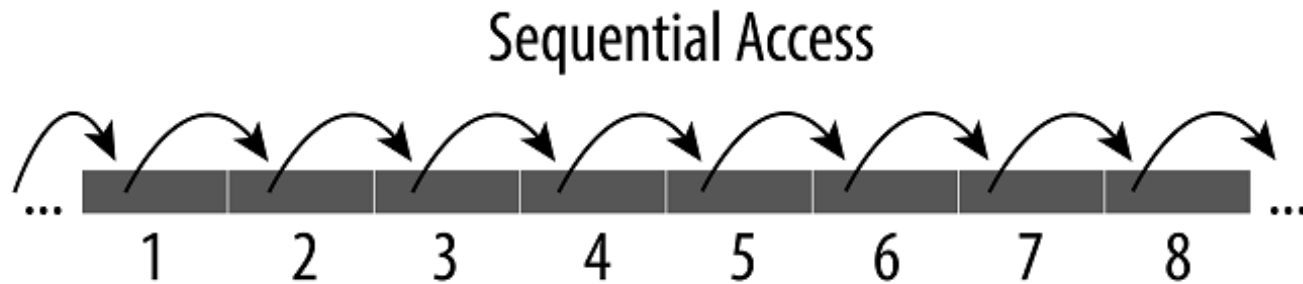
1. Kiểu dữ liệu tuần tự (sequential data type)
2. String (chuỗi)
3. Bài tập về xử lý chuỗi
4. List (danh sách)
5. Tuple (hàng)
6. Range (miền)
7. Bài tập về dữ liệu tuần tự



Phần 1

Kiểu dữ liệu tuần tự (sequential data type)

- Hai loại đặc trưng cơ bản trong điều khiển
 - Sequential access: truy cập tuần tự
 - Random access: truy cập ngẫu nhiên
- Tuần tự: khá thông dụng trong cuộc sống, chẳng hạn như xếp hàng, xử lý dây chuyền, lưu trữ trong băng từ,...





- Kiểu dữ liệu tuần tự trong python: những kiểu dữ liệu chứa bên trong nó các dữ liệu con và thường được xử lý bằng cách lấy ra từng phần-tử-một theo thứ tự nào đó (thường là 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



Phần 2

String (chuỗi)



- Một chuỗi được xem như một hàng (tuple) các chuỗi con độ dài 1
 - Trong python không có khái niệm 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
 - Chuỗi trong python hỗ trợ các kí tự unicode, vì vậy có thể sử dụng tiếng Việt (cũng như tiếng Hán, tiếng Nhật,...) thoải mái
- Hàm `len(s)` trả về độ dài (số kí tự) của s, hàm trả về 0 nếu chuỗi rỗng
- Phép toán với chuỗi:
 - Phép nối chuỗi (+): `s = "Good" + " " + "Morning!"`
 - Phép nhân bản (*): `s = "AB" * 3` `# ABABAB`
 - Kiểm tra nội dung: `s in '1ABABABCD'` `# True`



Ví dụ về phép nối chuỗi

```
a = 'Hello'
b = 'World'
print(a + b)                # "HelloWorld"
c = 'Hello' 'Kitty'
print(c)                    # "HelloKitty"
d = '1'
'2'
'3'
print(d)                    # "1"
e = ('1'
     '2'
     '3')
print(e)                    # "123"
print(d + '23')             # "123"
print(e '23')               # lỗi
```

Ví dụ về phép nhân bản và kiểm tra nội dung



```
print('abc' * 3)           # 'abcabcabc'
print(5 * '10')            # '1010101010'
print('ab' + 'cd' * 3)     # 'abcdcdcd'

print('bcd' in 'abcdef')   # True
print('bcd' in 'abc' * 3)  # False
print('abc' not in 'ab' + 'cd') # False
```



Phép so sánh giữa 2 chuỗi

- Phép so sánh chuỗi trong Python sử dụng trật tự từ điển, khi hai chuỗi A và B được so sánh với nhau, Python thực hiện theo quy tắc sau:
 - So sánh lần lượt từng cặp từ trái qua phải, tương ứng theo từng ký tự
 - Ký tự của chuỗi nào lớn hơn thì chuỗi tương ứng lớn hơn
 - Nếu hai ký tự giống nhau thì chuyển sang cặp tiếp theo
 - Nếu một chuỗi còn ký tự nhưng chuỗi kia đã hết thì chuỗi đã hết nhỏ hơn



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:
 - Từ trái qua phải: chỉ số đánh từ 0 tăng dần đến cuối chuỗi
 - Từ phải qua trái: chỉ số đánh từ -1 giảm dần về đầu chuỗi
 - Hai cách đánh chỉ mục này có thể sử dụng lẫn lộn với nhau, chẳng hạn: lấy từ vị trí 1 đến vị trí -2 của chuỗi ĐHTHUYLOI ta sẽ được chuỗi HTHUYL

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



- 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

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
```



- 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
 - Có thể thêm tham số chỉ độ rộng của định dạng (xem ví dụ)

- 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)
```



- 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

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




■ 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')
```



Các phương thức của 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



Các phương thức của chuỗi

- Các phương thức cắt phần dư
 - `strip([chars])`: loại bỏ những ký tự đầu hoặc cuối chuỗi thuộc vào danh sách `[chars]`, hoặc ký tự trống
 - `rstrip([chars])`: làm việc như `strip` nhưng cho bên phải
 - `lstrip([chars])`: làm việc như `strip` nhưng cho bên trái
- Tách chuỗi
 - `split(sep, maxsplit)`: tách chuỗi thành một danh sách, sử dụng dấu ngăn cách `sep`, thực hiện tối đa `maxsplit` lần
 - Tách các số nhập vào từ một dòng: `input("Test: ").split(',')`
 - `rsplit(sep, maxsplit)`: thực hiện như `split` nhưng theo hướng ngược từ phía cuối chuỗi



■ Các phương thức khác

- `join(list)`: ghép các phần tử trong list bởi phần gạch nối là nội dung của chuỗi, ví dụ: `'-'.join(('1', '2', '3'))`
- `replace(old, new [,count])`: thế nội dung old bằng nội dung new, tối đa count lần
- `count(sub, [start, [end]])`: đếm số lần xuất hiện của sub
- `startswith(prefix)`: kiểm tra đầu có là prefix không
- `endswith(prefix)`: kiểm tra cuối có là prefix không
- `find(sub[, start[, end]])`: tìm vị trí của sub (-1: không có)
- `rfind(sub[, start[, end]])`: như find nhưng tìm từ cuối
- `islower()`, `isupper()`, `istitle()`, `isdigit()`, `isspace()`, `isalpha()`, `isnumeric()`
- `index(sub)` giống find, nhưng sinh ngoại lệ nếu không tìm thấy



Các hàm dựng sẵn hỗ trợ chuyển đổi

- Hàm `chr(n)`: chuyển đổi giá trị số sang mã unicode
- Hàm `ord(c)`: chuyển đổi kí tự unicode sang giá trị số
- Hàm `len(s)`: trả về độ dài (số kí tự) của chuỗi
- Hàm `str(v)`: chuyển đổi giá trị của biến `v` sang thể hiện ở dạng chuỗi
- Ví dụ:

```
s = '€'          # s chứa kí tự euro có mã 8364
print(ord(s))    # in ra mã của s (8364)
print(chr(8364)) # in ra kí tự ứng với mã 8364
n = 3+4j         # n là số phức 3+4i
print(n)         # tự động chuyển n sang chuỗi và in ra
print(len(n))    # LỖI
print(len(str(n))) # in ra độ dài của n dạng chuỗi
```



Phần 3

Bài tập về xử lý chuỗi



Bài tập về xử lý chuỗi

1. Nhập một chuỗi từ bàn phím, kiểm tra xem nó có tận cùng bởi 3 dấu chấm than hay không (!!!), nếu không thì hãy thêm dấu chấm than vào cuối để chuỗi có tận cùng là 3 dấu chấm than.
2. Nhập dãy số từ bàn phím (các số được gõ trên cùng một dòng, cách nhau bởi dấu cách hoặc tab), in ra dãy số vừa nhập.
3. Nhập một tên người từ bàn phím, hãy tách phần họ và tên riêng của người đó và in chúng ra màn hình (giả thiết họ và tên riêng chỉ gồm một âm).
4. Nhập một chuỗi từ bàn phím, hãy loại bỏ tất cả các chữ số khỏi chuỗi và in lại nội dung chuỗi mới ra màn hình.

Bài tập về xử lý chuỗi



5. Nhập một dãy các từ từ bàn phím, hãy in ra từ dài nhất trong dãy vừa nhập, in ra mọi từ có cùng độ dài nhất.
6. Nhập một dãy các từ từ bàn phím, thống kê xem có những chữ cái nào xuất hiện trong dãy và mỗi chữ xuất hiện bao nhiêu lần?
7. Nhập chuỗi S gồm toàn chữ số và số nguyên N, chỉ ra cách xóa đúng N kí tự khỏi S để được số có giá trị lớn nhất.
8. Nhập chuỗi S, hãy thay thế tất cả các chữ số trong S bằng kí tự hỏi chấm (?), sau đó in lại S ra màn hình
9. Nhập chuỗi S, kiểm tra xem chuỗi S có là dạng đối xứng hay không?