Lập trình Python

Bài 5: Kiểu tuần tự trong python, phần 2

Tóm tắt nội dung bài trước

- Vật chứa là các loại dữ liệu đặc biệt, có thể chứa bên trong nó các loại dữ liệu con
- Kiểu tuần tự là loại vật chứa mà ta có thể duyệt từng phần tử con bên trong nó theo một thứ tự nào đó
- Chuỗi (str) là một dãy các str con độ dài 1 kí tự
  - Nhiều phép toán: nối chuỗi (+), nhân bản (*), kiểm tra (in)
  - So sánh hai chuỗi theo thứ tự từ điển
  - Hệ thống chỉ mục theo 2 chiều, trái sang phải và phải sang trái
  - Phép cắt chuỗi: tạo chuỗi mới theo vị trí đầu cuối
  - Ba kiểu định dạng chuỗi: định dạng (%), f-string và hàm format
  - Nhiều phương thức hỗ trợ thao tác nội dung chuỗi
- Python có các hàm chuyển đổi giữa số và kí tự unicode

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 4

List (danh sách)

Bất biến (immutable) và Khả biến (mutable)

- Bất biến = không thay đổi, các loại dữ liệu bất biến thông dụng trong Python: bool, int, float, str, tuple và frozenset
- Khả biến = có thể thay đổi, các loại dữ liệu khả biến thông dụng trong Python gồm: list, set, dict
- Chúng ta vẫn thay đổi giá trị của int, tại sao nói "bất biến"
  - Python không thực sự thay đổi giá trị của int, phần mềm tạo vùng nhớ chứa giá trị mới và cho biến "trỏ" tới vùng đó
- Ví dụ để hiểu rõ cơ chế này:

```
n = 100
print(n, id(n))      # 100 và id của biến n
n = n + 1
print(n, id(n))      # 101 và id của n thay đổi so với trên
```

Giới thiệu và khai báo

- List = dãy các đối tượng (một loại array đa năng)
- Các phần tử con trong list không nhất thiết phải cùng kiểu dữ liệu
- Khai báo trực tiếp: liệt kê các phần tử con đặt trong cặp ngoặc vuông ([]), ngăn cách bởi dấu 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 hỗ 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

Khởi tạo list

- Tạo list bằng constructor (hàm tạo)

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

- Tạo list bằng list comprehension (bộ suy diễn danh sách) một đoạn mã ngắn trả về các phần tử thuộc list

```
# list 1000 số nguyên từ 0 đến 999
X = [n for n in range(1000)]
# list gồm 10 list con là các cặp [x, x^2]
# với x chạy từ 0 đến 9
Y = [[x, x*x] for x in range(10)]
```

So sánh 2 list: theo thứ tự từ điển (như str)

```
a = [1, 2, 3]
b = [1, 2, 3, 4]
c = [1.5]
d = ['a', 'b', 'c']
print(a > b)                # False
print(a == b)               # False
print(a < b)                # True
print(a + [4] == b)         # True
print(c <= b)               # False
print(d != a)               # True
print(d == list('abc'))     # True
print(d >= c)               # Lỗi
```

Phép toán, chỉ mục và cắt

- 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: nội dung của list có thể thay đổi

```
# khởi tạo list ban đầu
l1 = list([1, 2, 3, 4])
# thay đổi giá trị của phần tử cuối cùng
l1[-1] = list('abc')
# in nội dung list: [1, 2, 3, ['a', 'b', 'c']]
print(l1)
```

Chỉ mục, lát cắt, xóa dữ liệu với list

```
a = list('abcde')
print(a)              # ['a', 'b', 'c', 'd', 'e']
a[-1] = [0, 5, 9]     # thay đổi phần tử cuối cùng
print(a)              # ['a', 'b', 'c', 'd', [0, 5, 9]]
a[1:3] = [1, 2, 3]    # thay đổi một đoạn
print(a)              # ['a', 1, 2, 3, 'd', [0, 5, 9]]
print(a[2::-1])       # [2, 1, 'a']
a[2::-1] = [0]        # lỗi, đoạn ngược không thể thay đổi
del a[2::-1]          # xóa đoạn con từ 2 trở về đầu
print(a)              # [3, 'd', [0, 5, 9]]
del a                 # xóa biến a
print(a)              # lỗi, a không tồn tại
```

Các phương thức của list

- 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, phát sinh lỗi 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

Các phương thức của list

- 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)
```

Ví dụ về sắp xếp với list

```
a = ['123', '13', '90', "-1", -100]
a.sort()              # lỗi, vì các phần tử không cùng kiểu
a[-1] = '-100'        # thay đổi phần tử cuối thành dạng chuỗi
a.sort()              # xếp tăng dần theo so sánh chuỗi
print(a)              # ['-1', '-100', '123', '13', '90']
a.sort(key=int)       # chuyển thành số nguyên rồi sắp xếp
print(a)              # ['-100', '-1', '13', '90', '123']
a.sort(key=int, reverse=True)
print(a)              # ['123', '90', '13', '-1', '-100']
```

Duyệt list với vòng lặp for

```
my_list = ['foo', 'bar', 'baz', 'noo’]

# duyệt các phần tử, cách làm đơn giản nhất
for item in my_list:
    print(item)

# duyệt các phần tử theo giá trị chỉ mục
for i in range(len(my_list)):
    print(i, my_list[i])

# duyệt theo giá trị chỉ mục nhưng nhanh hơn
# với cách sử dụng hàm enumerate
for i, item in enumerate(my_list):
    print(i, item)
```

Ví dụ về làm việc với list

```
# tạo và in nội dung của một list
danhsach = ["apple", "banana", "cherry"]
for x in danhsach:
    print(x)
# thêm một phần tử vào cuối list và lại in ra
danhsach.append("orange")
print(danhsach)
# thêm một phần tử vào giữa list và in ra
danhsach.insert(1, "orange")
print(danhsach)
# sắp xếp lại danh sách và in ra
danhsach.sort()
print(danhsach)
```

Một số thao tác thông dụng với list

1. Tạo một list
2. Duyệt list
3. Truy cập phần tử theo chỉ số
4. Thay đổi nội dung phần tử
5. Cắt list
6. Thêm phần tử vào list
7. Bỏ phần tử khỏi list
8. Tìm kiếm phần tử trong list
9. Sắp xếp các phần tử trong list
10. List hỗ hợp

Phần 5

Tuple (hàng)

Tuple là một dạng readonly list

- 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 phẩy

```
(1, 2, 3, 4, 5)              # tuple 5 số nguyên
('a', 'b', 'c', 'd')         # tuple 4 chuỗi
(1, 'one', [2, 'two'])       # tuple hỗ hợp
(1,)                         # tuple 1 phần tử
()                           # tuple rỗng
```

Khai báo tuple không nhất thiết phải dùng ()

```
t0 = 'a', 'b', 'c'              # tuple 3 chuỗi
t1 = 1, 2, 3, 4, 5              # tuple 5 số nguyên
t2 = ('a', 'b', 'c')            # tuple 3 chuỗi
t3 = (1, 'one', [2, 'two'])     # tuple hỗn hợp
t4 = 1, 'one', [2, 'two']       # tuple hỗn hợp
t5 = ()                         # tuple rỗng
t6 = 'a',                       # tuple một phần tử
t7 = ('a')                      # KHÔNG phải tuple (là chuỗi)
t8 = ('a',)                     # tuple một phần tử
t9 = ('a', )                    # tuple, không đúng chuẩn python
```

Tuple và list nhiều điểm giống nhau

- Tuple có thể tạo bằng constructor hoặc generator (bộ sinh) – một cách viết tương tự list comprehension
- Tuple hỗ trợ 3 phép toán: +, *, in
- Tuple cho phép sử dụng chỉ mục và cắt
- 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

Bộ sinh của tuple chỉ dùng được 1 lần

```
# t0 viết giống như bộ suy diễn danh sách
t0 = (c for c in 'Hello!’)
# tạo t1 từ t0
t1 = tuple(t0)
# tạo t2 từ t0
t2 = tuple(t0)
print(t0)   # <generator object <genexpr> at XXXX>
print(t1)   # ('H', 'e', 'l', 'l', 'o', '!')
print(t2)   # () <~~ như vậy t0 chỉ dùng được một lần
```

Tính bất biến của kiểu tuple

```
a = (1, 2, [3, 4], 'abc')
print(a[2])                 # [3, 4]
a[2].append('xyz')          # a[2] trở thành [3, 4, 'xyz']
print(a)                    # (1, 2, [3, 4, 'xyz'], 'abc')
a[2] = '123'                # lỗi
a = (1, 2, 3)               # ok: thay đổi a thành tuple mới
a += (4, 5)                 # ok: a thay đổi thành tuple mới
print(a)                    # (1, 2, 3, 4, 5)
del a                       # ok: xóa hoàn toàn a
```

Hàm dựng sẵn làm việc với list và tuple

- Hàm all(X): trả về True nếu tất cả các phần tử của X đều là True hoặc tương đương với True hoặc x rỗng
- Hàm any(X): trả về True nếu có ít nhất một phần tử của X là True hoặc tương đương với True
- Hàm len(X): trả về số lượng phần tử của X
- Hàm list(X): tạo một list gồm các phần tử con của X
- Hàm max(X): trả về giá trị lớn nhất trong X
- Hàm min(X): trả về giá trị nhỏ nhất trong X
- Hàm sorted(X): trả về danh sách mới gồm các phần tử của X đã được sắp xếp
- Hàm sum(X): trả về tổng các phần tử trong X

Phần 6

Range (miền)

Range là một tuple đặc biệt?

- 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)
  - Không nên coi Range là một Tuple!!!

Phần 7

Bài tập

Bài tập

1. Người dùng nhập từ bàn phím liên tiếp các từ tiếng Anh viết tách nhau bởi dấu cách. Hãy nhập chuỗi đầu vào và tách thành các từ sau đó in ra màn hình các từ đó theo thứ tự từ điến.
2. Người dùng nhập từ bàn phím chuỗi các số nhị phân viết liên tiếp được nối nhau bởi dấu phấy. Hãy nhập chuỗi đầu vào sau đó in ra những giá trị được nhập.
3. Nhập số n, in ra các số nguyên dương nhỏ hơn n có tổng các ước số thực sự lớn hơn chính nó.
4. Nhập vào một chuỗi từ người dùng, kiểm tra xem đó có phải địa chỉ email hợp lệ hay không?
5. Nhập n, in n dòng đầu tiên của tam giác pascal

Bài tập

6. Hãy nhập số nguyên n, tạo một list gồm các số fibonacci nhỏ hơn n và in ra
   - Dãy fibonacci là dãy số nguyên được định nghĩa một cách đệ quy như sau: f(0)=0, f(1) = 1, f(1<n) = f(n-1) + f(n-2)
7. Tạo tuple P gồm các số nguyên tố nhỏ hơn 1 triệu
   - Số nguyên tố là số tự nhiên có 2 ước số là 1 và chính nó.
8. Liệt kê các chuỗi độ dài ít hơn N của tuple X gồm các chuỗi được định nghĩa như sau:
   - Chuỗi M = '()' thuộc X
   - Nếu chuỗi A thuộc X thì chuỗi (A) cũng thuộc X
   - Nếu chuỗi A và B thuộc X thì chuỗi AB cũng thuộc X