Week 2: List

Time-Series Team

Ngày 16 tháng 6 năm 2025

Trong tuần này (Week 2) chúng ta được học về Data Structure. Với các ứng dụng có dữ liệu lớn, số lượng biến nhiều thì dữ liệu kiểu cấu trúc có vai trò vô cùng quan trong. Một trong những kiểu dữ liệu cấu trúc quan trọng nhất trong Python chính là List. Trong bài viết này, chúng ta sẽ cùng tìm hiểu về List với các nội dung sau:

- ***Phần I: List***
- ***Phần II: List methods***
- ***Phần III: List built-in function***
- ***Phần IV: List Loops***
- ***Phần V: 2D List***

Phần I: List

0.1 Lý thuyết

Danh sách (list) là một chuỗi các giá trị. Tuy nhiên, khác với chuỗi(string) các giá trị là ký tự, ở danh sách chúng có thể là bất ký loại nào (int, float, string, thậm chí là 1 list khác). Các giá trị trong list được gọi là phần tử (element).

Có nhiều cách để khởi tạo một danh sách mới, các đơn gian nhất là đặt các phần tử vào trong dấu ngoặc vuông [] như cú pháp dưới đây:

```
list_name = [element-1, ..., element-n]
```

Python Code

```
# Create a list
data = [4, 5, 6, 7, 8, 9]
print(data)
print(type(data))
print(len(data))
```

Output

```
[4, 5, 6, 7, 8, 9]
<class 'list'>
6
```

Một số đặc điểm nổi bật của List trong Python:

- Một danh sách nằm trong 1 danh sách khác gọi là danh sách lồng nhau.
- Một danh sách không chưa phần tử nào được gọi là danh sách rỗng.
- Hàm len trả về độ dài của 1 danh sách. Độ dài của một danh sách rỗng là 0.

0.2 List index

Chỉ số của list hoạt động giống như chỉ số chuỗi:

```
list_name →[   1    2    3    4    5    6   ]
            0    1    2    3    4    5
```

- index (chỉ mục) là độ lệch tính từ đầu list
- chỉ mục phải là một số nguyên - nếu không, bạn sẽ gặp lỗi TypeError
- Index âm dùng để đếm lùi từ cuối list, phần tử cuối cùng của list có index = -1
- Nếu bạn cố gặng đọc hoặc ghi một phần tử không tồn tại trong list, bạn sẽ gặp lôi IndexError

Một số phép toán trên danh sách:

1. Toán tử + nối các danh sách lại với nhau
2. Toán tử * lặp lại 1 danh sách với số lần nhất định
3. **min** tìm phần tử nhỏ nhất trong danh sách
4. max tìm phần tử lớn nhất trong danh sách
5. Không có toán tử số học nào khác hoạt động với list, nhưng hàm tích hợp sẵn **sum** có thể trả về tổng của các phần tử trong list.

0.3 List Slicing

Slicing là kỹ thuật cho phép trích xuất một phần (một list con) từ list gốc. Việc này được thực hiện bởi cú pháp:

```
list[start:end:step]
```

Python Code

```
# Create a list
data = [4, 5, 6, 7, 8, 9]
print(data[2:4])   # => [6, 7]
print(data[::2])   # => [4, 6, 8]
print(data[1::])
# => [5, 6, 7, 8, 9]
```

-
- **start index:** có giá trị mặc định là `0`
- **end index:** có giá trị mặc định là `len(list)`
- **step:** có giá trị mặc định là `1`

Cho lst = [1,2,3,4,5,6,7,8,9]

Nếu bạn bỏ qua chỉ mục đầu tiên thì phần cắt sẽ bắt đầu từ đầu danh sách. lst[:2] => [1,2]
Nếu bạn bỏ qua chỉ số thứ 2, phần cắt sẽ kéo dài tới cuối danh sách. lst[2:] => [3,4,5,6,7,8,9]
Nếu bạn bỏ qua cả 2, phần cắt sẽ là một bản sao của list. lst[:] => [1,2,3,4,5,6,7,8,9]
Nếu bạn bỏ qua chỉ mục cuối cùng(step), nó sẽ có giá trị mặc định là 1.

Phần II: List Methods

Nhờ tính chất mutable, list trong Python đi kèm với nhiều phương thức (methods) và hàm (functions) tích hợp sẵn cho phép thực hiện các thao tác với list python một cách dễ dàng. Dưới đây là những thao tác phương thức cơ bản và quan trọng nhất.

.append()	+ Adds an element at the end of the list	**.sort()**	+ Sorts the list
.insert()	+ Adds an element at the specified position	**.reverse()**	+ Reverses the order of the list
.extend()	+ Add the elements of a list (or any iterable), to the end of the current list	**.copy()**	+ Returns a copy of the list
.remove()	+ Removes the item with the specified value	**.count()**	+ Returns the number of elements with the specified value
.pop()	+ Removes the element at the specified position	**.index()**	+Returns the index of the first element with the specified value
.clear()	+ Removes all the elements from the list		

0.4 Thêm phần tử vào list

Để thêm một item(phần tử) vào cuối danh sách, hãy sử dụng phương thức **append()**. Phương thức **append(value)**, sẽ thêm 1 phần tử có giá trị value vào cuối list và trả về none.

```
lst= [4, 5, 6, 7, 8, 9]
print(lst)
print(lst.append(8))
print(lst)

>>>
[4, 5, 6, 7, 8, 9]
None
[4, 5, 6, 7, 8, 9, 8]
```

Để thêm item tại chỉ mục chỉ định , hãy sử dụng phương thức **insert()**. Phương thức **insert(index, value)** sẽ thêm 1 phần tử có giá trị value vào vị trí index chỉ định, và trả về none.

```
lst= [4, 5, 6, 7, 8, 9]
print(lst)
print(lst.insert(3,0))
print(lst)

>>>
[4, 5, 6, 7, 8, 9]
None
[4, 5, 6, 0, 7, 8, 9]
```

Để thêm nội dung của seq vào cuối list hãy sử dụng phương thức extend(). Phương thức này sẽ thêm tất cả các phần tử của seq vào cuối list.

```
fruits = ["apple", "banana", "guava"]
```

```
fruits.extend([1,2,3])
print(fruits)

>>>
['apple', 'banana', 'guava', 1, 2, 3]
```

0.5 Xóa phần tử khỏi list

Để xóa phần tử ra khỏi list, ta sử dụng phương thức **pop()**. Phương thức **pop(index)** sẽ thực hiện xóa phần tử thứ index ra khỏi list và trả về giá trị của phần tử đó.

```
lst= [4, 5, 6, 7, 8, 9]
print(lst)
print(lst.pop(1))
print(lst)

>>>
[4, 5, 6, 7, 8, 9]
5
[4, 6, 7, 8, 9]
```

Để xóa một phẩn tử ra khỏi list mà chỉ biết giá trị cần xóa, không biết index của giá trị đó hãy sử dụng phương thức **remove()**. Phương thức **remove(value)** sẽ thực hiện xóa giá trị value đầu tiên tìm thấy ra khỏi danh sách.

```
lst= [4, 5, 6, 7, 8, 9]
print(lst)
print(lst.remove(9))
print(lst)

>>>
[4, 5, 6, 7, 8, 9]
None
[4, 5, 6, 7, 8]
```

Để xóa toàn bộ phần tử trong list hãy sử dụng phương thức **clear()**. Phương thức này trả về một list rỗng.

```
fruits = ["apple", "banana", "guava"]
fruits.clear()
print(fruits)

>>>
[]
```

0.6 Một số methods thường dùng khác với list

Để kiểm tra chỉ mục lần đầu tiên xuất hiện của phần tử hãy dung phương thức **index()**.

```
fruits = ["apple", "banana", "guava"]
print(fruits)
fruits.index("apple")

>>>
['apple', 'banana', 'guava']
0
```

Để đảo ngược thứ tự các phần tử trong list hãy sử dụng phương thức **reverse()**.

```
fruits = ["apple", "banana", "guava"]
print(fruits)
fruits.reverse()
print(fruits)

>>>
['apple', 'banana', 'guava']
['guava', 'banana', 'apple']
```

Để sắp xếp danh sách theo thứ tự tăng dần hãy dùng phương thức **sort()**.

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)

>>>
['banana', 'kiwi', 'mango', 'orange', 'pineapple']
```

Để sắp xếp theo thứ tự giảm dần, hãy sử dụng phương thức **sort()** với đối số từ khóa **reverse = True**:

```
thislist = [100, 50, 65, 82, 23]
thislist.sort(reverse = True)
print(thislist)

>>>
[100, 82, 65, 50, 23]
```

Trong python, bạn không thể copy list bằng cách nhập lst1 = lst2, bởi vì lst2 chỉ là 1 tham chiếu đến lst1 và những thay đổi trong 1 trong 2 list sẽ tự động thực hiện trong list còn lại, điều này là điều không mong muốn. Vì vậy, Để copy list hãy sử dụng phương thức **copy()**.

```
fruits = ["apple", "banana", "guava"]
fruits_2 = fruits.copy()
print(fruits)
print(fruits_2)

fruits.append(2)
print(fruits)
print(fruits_2)

>>>
['apple', 'banana', 'guava']
['apple', 'banana', 'guava']
['apple', 'banana', 'guava', 2]
['apple', 'banana', 'guava']
```

Để đếm số lần phẩn tử xuất hiện trong list, hãy sử dụng phương thức count().

```
fruits = ["apple", "banana", "guava","apple" ]
fruits.count("apple")

>>>
2
```

Phần III: List built-in function

Một số hàm tích hợp sẵn trên list bao gồm:

- Hàm len() trả về độ dài của list

  ```
  thislist = ["apple", "banana", "cherry"]
  print(len(thislist))
  ```

```
>>>
3
```

- Hàm min() trả về phần tử nhỏ nhất trong list

```
thislist = [100, 50, 65, 82, 23]
print(min(thislist))

>>>
23
```

- Hàm max() trả về phần tử lớn nhất trong list

```
thislist = [100, 50, 65, 82, 23]
print(max(thislist))

>>>
100
```

- Hàm sum(iterable, start) trả về tổng các phần tử trong danh sách

```
data = [6, 5, 7, 1, 9, 2]
print(sum(data))  # Output: 30
print(sum(data,7))  # Output: 37
```

- Hàm zip() gom các phần tử theo cặp index từ nhiều list

```
data1 = [1, 2, 3]
data2 = [5, 6, 7]
for v1, v2 in zip(data1, data2):
    print(v1, v2)
>>>
1 5
2 6
3 7
```

- Hàm reversed() đảo ngược thứ tự list, trả về iterator

```
data = [6, 1, 7]
for v in reversed(data):
    print(v)
>>>
7
1
6
```

- Hàm sorted() sắp xếp list mới (không thay đổi list gốc)

```
data = [6, 5, 7, 1, 9, 2]
sorted(data)
sorted(data, reverse=True)
>>>
[1, 2, 5, 6, 7, 9]
[9, 7, 6, 5, 2, 1]
```

- Hàm enumerate() duyệt qua từng phần tử trong list, kèm theo vị trí (index) của nó. (có thể thay đổi index ban đầu)

```
data = [6, 1, 7]
for i, v in enumerate(data):
    print(i, v)
>>> output:
0 6
1 1
2 7

for i, v in enumerate(data, 7):
    print(i, v)
>>>output
7 6
8 1
9 7
```

Phần IV: Vòng lặp List

Bạn có thể lặp qua các phần tử trong danh sách bằng cách sử dụng vòng lặp for.

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)

>>>output
apple
banana
cherry
```

Bạn cũng có thể lặp qua các phần tử danh sách bằng cách tham chiếu đến số chỉ mục của chúng(sử dụng các hàm range()và len())

```
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(thislist[i])

>>>output
apple
banana
cherry
```

Bạn có thể lặp qua các mục danh sách bằng cách sử dụng vòng lặp while.

```
thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):
  print(thislist[i])
  i = i + 1

>>>output
apple
banana
cherry
```

Khi sử dụng vòng lặp với list, bạn có thể sử dụng List Comprehension. List Comprehension cung cấp cú pháp ngắn nhất để lặp qua các danh sách.

```
thislist = ["apple", "banana", "cherry"]
[print(x) for x in thislist]

>>>output
apple
banana
cherry
```

Khi sử dụng List Comprenhension, hãy nhớ cú pháp sau:

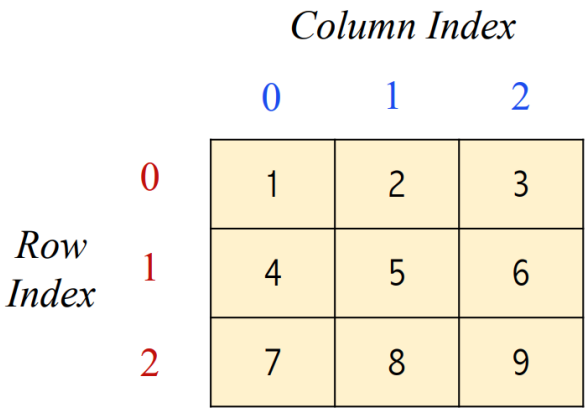
List comprehension with condition:

- `[expression for x in data if condition]`
  (Lọc phần tử thoả điều kiện)
- `[expression_true if condition else expression_false for x in data]`
  (Thực hiện phân nhánh if-else cho mỗi phần tử)

Phần V: 2D List

1 2D list

2D List (hay ma trận 2 chiều) là danh sách các danh sách (list of lists).Có thể tưởng tượng nó giống như bảng (table) với nhiều hàng (rows) và cột (columns).



Khai báo 2D list:

```
2D_lst = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Truy cập phần tử của 2D list:

```
2D_list[row][col]
```

```
print(2D_lst[0][0])
```

=> Kết quả là : 1

2 Một số thao tác với 2D list

2.1 Duyệt qua 2D List

Để thực hiện duyệt qua 2D list, hãy dùng 2 vòng lặp for lồng nhau.

- Duyệt theo hàng:

```
for row in 2D_lst:
    for val in row:
        print(val, end=" ")
    print()
```

- Duyệt theo index:

```
num_rows = len(2D_lst)
num_cols = len(2D_lst[0])

for r in range(num_rows):
    for c in range(num_cols):
        print(m[r][c], end=" ")
    print()
```

2.2 Cập nhật phần tử

Cũng tương tự như 1D list, chúng ta vẫn sự dụng phép gán: 2D_lst[1][1] = 0

2.3 Tính toán trên 2 ma trận (Hadamard Product)

Hadamard Product là phép nhân từng phần tử tương ứng của hai ma trận cùng kích thước.Không giống như phép nhân ma trận chuẩn (dot product), Hadamard là phép nhân element-wise:

$$\overset{G}{\begin{bmatrix}3 & 5 & 7\\ 4 & 9 & 8\end{bmatrix}} \circ \overset{H}{\begin{bmatrix}1 & 6 & 3\\ 0 & 2 & 9\end{bmatrix}} = \overset{N}{\begin{bmatrix}3\times1 & 5\times6 & 7\times3\\ 4\times0 & 9\times2 & 8\times9\end{bmatrix}}$$

```
A = [[1, 2], [3, 4]]
B = [[5, 6], [7, 8]]
rows = len(A)
cols = len(A[0])

result = [[0]*cols for _ in range(rows)]

for r in range(rows):
    for c in range(cols):
        result[r][c] = A[r][c] * B[r][c]

print(result)
```

=>Kết quả là: [[5, 12], [21, 32]]

Phần VI: Một vài ứng dụng

Bạn đã bao giờ phải tính tổng liên tục trong một danh sách rất dài chưa? Trong AI, xử lý dữ liệu hay thi code competition, điều này cực kỳ thường gặp. Vậy làm sao để tính tổng cực nhanh?

Trong phần này, chúng ta sẽ khám phá một ý tưởng cực hay: Áp dụng khái niệm tích phân trong toán học vào danh sách số trong Python để tính tổng nhanh chóng với độ phức tạp O(1)!

Độ phức tạp là gì?

"Big O notation" là cách để mô tả độ phức tạp thuật toán. O(n) nghĩa là thời gian xử lý tỷ lệ với số phần tử n.

Vậy O(1) lại là gì?

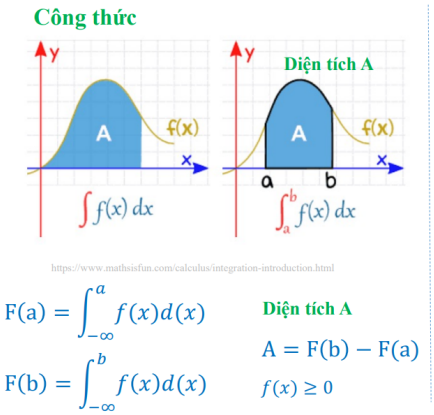
O(1) nghĩa là thời gian xử lý không đổi, nhanh tức thì.

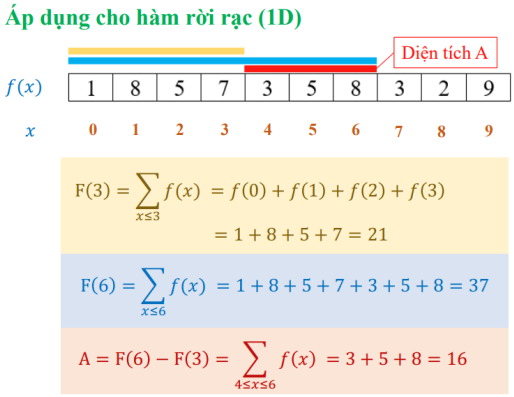
3 Tính tổng dữ liệu trong danh sách

Giả sử ta có một danh sách biểu diễn một hàm số ngẫu nhiên:
f = [1, 8, 5, 7, 3, 5, 8, 3, 2, 9]
Và ta muốn tính tổng giá trị từ vị trí a đến b (a x b): sum(f[a:b+1])
Nhưng bài toán đặt ra: liên tục nhiều lần, trên dữ liệu rất lớn. Khi đó, ta cần tối ưu.

4 Tích phân trong Giải tích

Cùng nhớ lại kiến thức toán:





Tích phân trên đoạn [a, b] là diện tích dưới đường cong.

Trong dữ liệu rời rạc, ta cũng có thể xem từng giá trị như cột dữ liệu: F(3), F(6) Vậy Area = F(6)-F(3), Tương đương: tính tổng từ x=4 đến x=6

5 Xây dựng

Tạo một danh sách mới F sao cho: F[x] = f[x] + F[x-1]

x	$f(x)$	$F(x)$
0	1	1
1	8	9
2	5	14
3	7	21
4	3	24
5	5	29
6	8	37
7	3	40
8	2	42
9	9	51

Khi đã có F, ta tính nhanh: Area = F[b] - F[a-1]

```
def prefix_sum(data):
    F = [0] * len(data)
    F[0] = data[0]
    for i in range(1, len(data)):
        F[i] = F[i-1] + data[i]
    return F

def range_sum(F, a, b):
    return F[b] - (F[a-1] if a > 0 else 0)

f = [1, 8, 5, 7, 3, 5, 8, 3, 2, 9]
F = prefix_sum(f)
print(" T ng   t   x=4    n    x=6:", range_sum(F, 4, 6))
```

=>Kết quả là: Tổng từ x=4 đến x=6: 16

Nhờ prefix sum array, ta chỉ mất O(n) để tiền xây dựng, nhưng về sau mỗi lần tính tổng ta chỉ cần O(1), vì sau khi đã có prefix sum, việc tính tổng từ a đến b chỉ là phép trừ:F[b] - F[a-1], bất kể n lớn bao nhiêu → chạy rất nhanh.