Bài 36: KIỂU DỮ LIỆU FUNCTION TRONG PYTHON - ĐỆ QUY (RECURSION)

Xem bài học trên website để ủng hộ Kteam: Kiểu dữ liệu Function trong Python - Đệ quy (recursion)

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage How Kteam nhé!

Dẫn nhập

Trong bài trước, Kteam đã giới thiệu đến bạn KIỂU DỮ LIỆU FUNCTION TRONG PYTHON – FUNCTIONAL TOOLS.

Và ở bài này Kteam sẽ lại tìm hiểu với các **KIỂU DỮ LIỆU FUNCTION – ĐỆ QUY** trong Python.

Nội dung

Để đọc hiểu bài này tốt nhất bạn cần:

- Cài đặt sẵn MÔI TRƯỜNG PHÁT TRIỂN CỦA PYTHON.
- Xem qua bài CÁCH CHẠY CHƯƠNG TRÌNH PYTHON.

- Nắm CÁCH GHI CHÚ và BIẾN TRONG PYTHON.
- CÁC KIỂU DỮ LIỆU ĐƯỢC GIỚI THIỆU TRONG PYTHON
- CÂU ĐIỀU KIỆN IF TRONG PYTHON

Bạn và Kteam sẽ cùng tìm hiểu những nội dung sau đây

- Giới thiệu về đệ quy
- Minh họa đệ quy bằng cách tính tổng
- Đệ quy theo phong cách Python
- Đệ quy và vòng lặp

Giới thiệu về đệ quy

Đệ quy là một mảng kiến thức nâng cao, ở Python thì nó không thường xuyên được dùng đến, do cách xử lí của Python có thể sử dụng những cấu trúc vòng lặp đơn giản mà không cần dùng tới đệ quy. Nhưng dù sao thì đây cũng là một kĩ thuật khá hữu dụng mà bạn đọc nên biết. Nó cũng chỉ đơn giản là việc chính nó gọi nó.

Minh họa đệ quy bằng cách tính tổng

Ta sẽ tính tổng các phần tử của một **list** (hoặc một sequence nào đó) bằng cách dùng **đệ quy** (Ví dụ này chỉ là minh họa, thực tế khi làm bạn nên sử dụng hàm **sum**)

```
>>> def cal_sum(lst):
...     if not lst: # tương đương if len(lst) == 0:
...             return 0
...     else:
...             return lst[0] + cal_sum(lst[1:])
...
>>> cal_sum([1, 2, 3, 4])
10
```

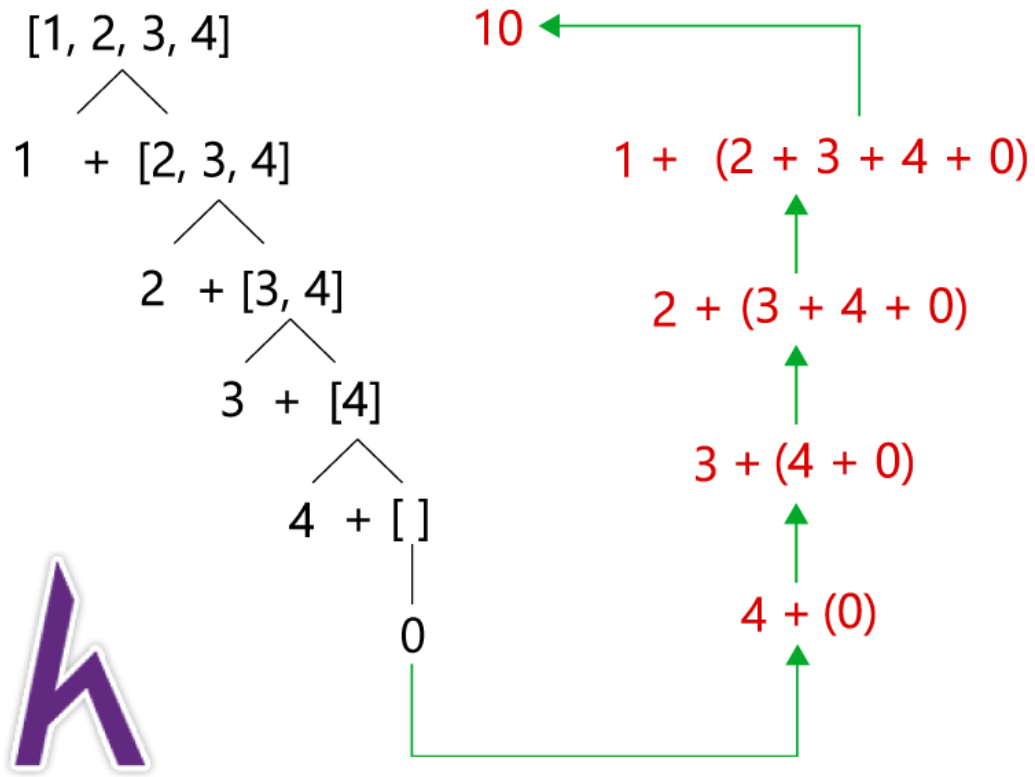
```
>>> cal_sum([1, 2, 3, 4, 5])
15
```

Ở ví dụ trên, ta liên tục gọi lại hàm cal_sum với **argument** là phần còn lại của **List** tính từ **index 1**. Ở mỗi lần gọi hàm, ta để lại giá trị **index 0** ở **List** để khi trong **List** không còn phần tử nào ta sẽ trả về số 0 để kết thúc đệ quy.

Nếu bạn thấy vẫn còn chưa hiểu rõ thì cũng đừng lo lắng, ai cũng đều cảm thấy khó hiểu và điều này thường xuyên xảy ra với những bạn mới học. Những lúc như thế này, bạn nên để thêm một cái hàm print để xem cụ thể là chuyện gì xảy ra

```
>>> def cal_sum(lst):
...     print(lst)
...     if not lst:
...             return 0
...     else:
...             return lst[0] + cal_sum(lst[1:])
...
>>> cal_sum([1, 2, 3, 4])
[1, 2, 3, 4]
[2, 3, 4]
[3, 4]
[4]
[]
10
```

Nếu bạn vẫn còn khó hiểu hãy vẽ ra giấy. Đâyq là cách mà mình đã làm



Đệ quy theo phong cách Python

Bạn còn nhớ cú pháp if/else trong **lambda** không? Nó còn có tên khác là **ternary expression**. Bạn có thể áp dụng để sử dụng nó để đệ quy:

```
>>> def cal_sum(lst):
...     return 0 if not lst else lst[0] + cal_sum(lst[1:])
...
>>> cal_sum([1, 2, 3])
6
```

Giả sử một list có **n** phần tử thì với đệ quy như trên cần phải có **n + 1** lần **return**, ta có thể giảm bớt xuống còn **n** lần **return** bằng cách:

```
>>> def cal_sum(lst):
...     return lst[0] if len(lst) == 1 else lst[0] + cal_sum(lst[1:])
...
>>> cal_sum([1, 2, 3])
6
```

> **Lưu ý:** cách này không sử dụng được trong trường hợp **container rỗng**

Hoặc ta có thể sử dụng **packing argument:**

```
>>> def cal_sum(lst):
...     idx0, *r = lst # idx0, r = lst[0], lst[1:]
...     return idx0 if not r else idx0 + cal_sum(r)
...
>>> cal_sum([1, 2, 3])
6
```

> **Lưu ý:** cách này cũng không sử dụng được khi **container là rỗng**. Tuy nhiên điểm lợi của nó cũng như cách vừa nãy là ta có thể cộng không chỉ số mà là chuỗi, hoặc list

```
>>> cal_sum(['a', 'b', 'c'])
'abc'
>>> cal_sum([[1, 2], [3, 4], [5, 6]])
[1, 2, 3, 4, 5, 6]
```

Đệ quy cũng có thể chuyển hướng. Hãy xem ví dụ sau. Một hàm gọi một hàm khác, sau đó lại gọi lại hàm đã gọi nó.

```
>>> def cal_sum(lst):
...     if not lst: return 0
...     return call_cal_sum(lst)
...
>>> def call_cal_sum(lst):
...     return lst[0] + cal_sum(lst[1:])
...
>>> cal_sum([1, 2, 3, 4, 5])
15
```

Đệ quy và vòng lặp

Ở những ví dụ trước, nếu phải chọn vòng lặp hay là đệ quy để xử lí thì Kteam khuyên bạn đọc nên chọn vòng lặp. Python chú trọng việc làm đơn giản hóa mọi việc như là vòng lặp vì nó theo một cách bình thường đơn giản. Vòng lặp cũng không yêu cầu bạn phải tạo ra một hàm mới có thể sử dụng được. Và thêm nữa, đệ quy còn thua vòng lặp ở mặt hiệu quả về bộ nhớ và thời gian thực hiện.

Kết luận

Qua bài viết này, bạn đã biết thêm về đệ quy. Ngoài ra, bạn cũng đã nắm được kha khá các kiến thức cơ bản của Python thông qua khóa LẬP TRÌNH PYTHON CƠ BẢN. Hy vọng khóa học sẽ là nền tảng tốt để bạn có thể tiếp tục tự nghiên cứu hoặc tiến tới các khóa học Python khác.

Cảm ơn bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quyên "**Luyện tập – Thử thách – Không ngại khó**".