Bài 30: KIỂU DỮ LIỆU FUNCTION TRONG PYTHON - PACKING VÀ UNPACKING ARGUMENTS

Xem bài học trên website để ủng hộ Kteam: Kiểu dữ liệu Function trong Python - Packing và unpacking arguments

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 - Positional và keyword argument

Và ở bài này Kteam sẽ lại tìm hiểu với các bạn **Kiểu dữ liệu Function trong Python - Packing và unpacking arguments.**

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
- VÒNG LẶP WHILE và VÒNG LẶP FOR TRONG PYTHON
- NHẬP XUẤT TRONG PYTHON

Trong bài này, chúng ta sẽ cùng tìm hiểu những nội dung sau đây:

- Unpacking arguments với *
- Packing arguments với *
- Unpacking arguments với **
- Packing arguments với **

Unpacking arguments với *

Giả sử, bạn có một hàm thế này

```
>>> def kteam(k, t, e, r):
...     print(k)
...     print(t, e)
...     print('end', r)
...
```

Bạn thấy đấy! Hàm này gồm 4 parameter và không có **default argument**. Vậy nên khi gọi hàm này, bạn buộc phải đưa vào 4 **argument**.

Nhưng bạn có một vấn đề, 4 argument cần truyền vào khi gọi hàm này lại nằm trong một List.

```
>>> lst = ['123', 'Kteam', 69.96, 'Henry']
```

Chả sao cả, bạn có thể lấy từng phần tử (element) trong list ra một cách dễ dàng, sau đó bạn có thể gọi hàm kteam như thế này.

```
>>> kteam(lst[0], lst[1], lst[2], lst[3])
123
Kteam 69.96
end Henry
```

Phức tạp vấn đề lên một chút nào! Sẽ ra sao nếu bạn có 50 parameter và phải gõ hết 50 **indexing** để truyền vào cho hàm khi gọi nó?

Lập trình viên lười lắm, họ không làm chuyện đó đâu. Vậy nên, Python cho phép làm điều đó đơn giản chỉ bằng một dấu *

```
>>> kteam(*lst)
123
Kteam 69.96
end Henry
```

Khi bạn sử dụng *, bạn không chỉ có thể **unpack** được các List mà bên cạnh đó bạn có thể **unpack** các container khác như Tuple, Chuỗi, Generator, Set, Dict (chỉ lấy được key).

Lưu ý:

Pass argument bằng cách **unpacking** argument như thế này là đang truyền vào dưới dạng positional argument. Hãy cẩn thận sử dụng kĩ thuật này với những hàm có parameter dạng keyword-only argument.

```
>>> def a(*, s, d):
...     print(s, d)
...
>>> a(*('a', 'b'))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: a() takes 0 positional arguments but 2 were given
```

Trong trường hợp container của bạn **unpack** các giá trị có trong container nhưng vẫn chưa đủ yêu cầu của hàm, thì bạn có thể truyền thêm:

```
>>> kteam(*('a', 'b', 'c'), 'd')
a
b c
end d
```

Packing arguments với *

Bạn còn nhớ hàm **print** chứ? Nó có khả năng nhận được bao nhiêu argument cũng được. Làm sao nó làm được như thế?

Đó chính là nhờ packing argument. Đôi lúc, bạn sẽ không thể biết trước được bạn sẽ pass vào bao nhiêu argument. Việc kiểm soát điều đó đôi lúc là bất khả thi.

Khi bạn sử dụng packing argument. Đồng nghĩa với việc bạn nhờ một biến đi gói tất cả các giá trị truyền vào cho hàm bằng positional argument thành một Tuple. Nếu không có gì để gói (bạn không truyền vào bất cứ argument nào) thì bạn sẽ nhận được một tuple rỗng. Để giao nhiệm vụ cho một biến làm công việc này, bạn đặt một dấu * trước nó.

```
>>> def kteam(*args):
...     print(args)
...     print(type(args))
...
>>> kteam('Kteam', 69.96, 'Henry')
('Kteam', 69.96, 'Henry')
<class 'tuple'>
>>> kteam(*(x for x in range(7))) # unpack sau đó là pack
(0, 1, 2, 3, 4, 5, 6)
<class 'tuple'>
```

Lưu ý:

> Bạn không nên nhầm lẫn việc này với việc force keyword-only argument
>
> **Không** được phép để 2 parameter cùng làm nhiệm vụ packing argument trong một hàm

Nếu sau một packing parameter còn có những parameter khác, khi gọi hàm muốn truyền giá trị vào cho các parameter sau packing parameter bạn cần phải sử dụng keyword argument.

```
>>> def f(*args, kter):
...     print(kter)
...
>>> f('1', '2')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: f() missing 1 required keyword-only argument: 'kter'
>>> f('1', '2', kter='3')
3
```

Bạn có thể sử dụng kĩ thuật này khi khai báo biến. Kteam sẽ nói về vấn đề này ở một bài khác.

Ở những ví dụ trên các bạn có thể thấy Kteam sử dụng biến packing có tên là args. Đó không phải là ngẫu nhiên mà là một quy ước nhỏ của các Pythonist với nhau. Thường người ta sẽ sử dụng biến có tên là args (viết gọn của arguments) để làm biến packing.

Trong Python, có rất nhiều quy ước là những luật bất thành văn như cách đặt tên biến, cách định dạng code, cách đặt tên file. Bạn sẽ biết thêm ở một bài khác của Kteam.

Unpacking arguments với **

Ta thử **unpacking** một Dict chỉ với một dấu *

```
>>> dic = {'name': 'Kteam', 'member': 69}
>>> def kteam(a, b):
...     print(a)
...     print(b)
...
>>> kteam(*dic)
name
member
```

Như bạn thấy, chúng ta chỉ lấy được key thôi.

Với Dict, thì nó phức tạp hơn một xíu khi mỗi phần tử là một cặp **key** và **value**. Vậy nên một dấu * là không đủ nội công để **unpack** hết được các giá trị. Do đó ta phải nhờ đến một cặp **.

Nếu bạn **unpacking** một Dictionary để truyền argument vào cho hàm khi gọi hàm thì đây chính là dạng keyword argument. Vậy nên bạn phải chắc chắn rằng parameter với key là giống nhau.

```
>>> dic = {'name': 'Kteam', 'member': 69}
>>> def kteam(a, b):
...     print(a)
...     print(b)
...
>>> kteam(**dic)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: kteam() got an unexpected keyword argument 'name'
>>> def kteam(name, member):
...     print('name =>', name)
...     print('member =>', member)
...

>>> kteam(**dic)
name => Kteam
member => 69
```

Packing arguments với **

Đã có **unpacking** với ** thì cũng phải có packing. Khác với dấu * là gói những positional argument thì ** lại gói các keyword argument. Và đương nhiên, nó sẽ gói trong một Dict. Nếu không truyền gì vào sẽ là một dict rỗng.

```
>>> def kteam(**kwargs):
...     print(kwargs)
...     print(type(kwargs))
...

>>> kteam(name='Kteam', member=69)
{'name': 'Kteam', 'member': 69}
<class 'dict'>
```

Tên biến kwargs (viết gọn của keyword arguments) cũng là một quy ước đặt tên.

```
>>> def kteam(**kwargs):
...     for key, value in kwargs.items(): # phương thức items của kiểu dữ liệu Dict
...         print(key, '=>', value)
...
>>> kteam(id=3424, name='Henry', age=18, lang='Python')
id => 3424
name => Henry
age => 18
lang => Python
```

Lưu ý:

> bạn không được phép bỏ các positional parameter sau biến packing mà có ** giống như với *.

```
>>> def f(**a, b):
  File "<stdin>", line 1
    def f(**a, b):
              ^
SyntaxError: invalid syntax
```

Nhờ những kiến thức trên, bạn có thể có một hàm cực kì linh hoạt như sau

```
>>> def best_function_ever(*args, **kwargs):
...     # việc còn lại của bạn là thỏa sức biến tấu
...     pass
...
```

Bạn hãy nắm chắc kĩ thuật này, tuy đơn giản nhưng lại được sử dụng rất nhiều.

Củng cố bài học

Đáp án bài trước

Bạn có thể tìm thấy câu hỏi của phần này tại CÂU HỎI CỦNG CỐ trong bài KIỂU DỮ LIỆU FUNCTION TRONG PYTHON – Phần 2.

Đáp án: Ta dùng lệnh help

```
>>> help(sorted)
```

Ta sẽ được cấu trúc của hàm **sorted** như sau:

```
sorted(iterable, /, *, key=None, reverse=False)
```

Vậy nên các positional only là **iterable**, còn keyword only là **key** và **reverse**.

Kết luận

Qua bài viết này, Bạn đã biết thêm về hàm trong Python qua các khái niệm packing và unpacking arguments.

Ở bài viết sau. Kteam sẽ tiếp tục giới thiệu thêm với các bạn về KIỂU DỮ LIỆU FUNCTION TRONG PYTHON - BIẾN LOCALS VÀ GLOBALS

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 quên **"Luyện tập – Thử thách – Không ngại khó"**.