関数に 可変長引数(*args) を渡す時の注意

* args

> args は、慣習なので、tpl でもよいし、分かりやすい名前を使ってもOKです。

私が、今日期待通りに行かなかった例を書いていこうと思います。

割と起こり得ることだと思うので、書くことにしました。

自分が書いたコードを、それを説明するためだけのコードに書き換えました。

以下のコードの挙動を考えてみましょう。

```
# howto_args.py

lst1 = [1, 2, 3]
lst2 = [4, 5, 6]


# リストの最初の値を返す関数
def return_first(lst):
    return lst[0]


# 複数のリストを受け取って、それぞれの最初の値を返す関数
def check_first(*arg):
    for elm in arg:  # ここでは、arg に * を付けないこと
        print(return_first(elm))


# lst1 と lst2 を、タプルにしました
lst_group = (lst1, lst2)

""" 以下のコードの挙動を考えてみましょう """
# 1 (lst1, lst2) で渡す
check_first((lst1, lst2))

# 2 lst_group で、渡す
```

```
check_first(lst_group)

# ３ 「lst1, lst2」 で、渡す
check_first(lst1, lst2)

# ４ *lst_group で、渡す（*が付いている）
check_first(*lst_group)
```

`*`の働き

- 関数定義
  - `*args` は、`複数の引数を一つにまとめる`(パッキング)をする
- 関数内での働き
  - 展開して(ばらして)、タプルとして扱う
- 関数に渡す時
  - 展開(アンパック)して渡す
- 関数内で使う時の、表記方法
  - * を付けない
    - ×：*args
    - ○： args

これを念頭に考えてみましょう：

このコードを実行すると、以下のようになります。　　期待通りの挙動は、３ ４ です

```
# howto_args.py

lst1 = [1, 2, 3]
lst2 = [4, 5, 6]

# リストの最初の値を返す関数
def return_first(lst):
    return lst[0]

# 複数のリストを受け取って、それぞれの最初の値を返す関数
def check_first(*arg):
```

```
    for elm in arg:  # ここでは、argに、* を付けないこと
        print(return_first(elm))

# lst1 と lst2 を、タプルにしました
lst_group = (lst1, lst2)

""" 以下のコードの挙動を考えてみましょう """
# 1 (lst1, lst2) で渡す
check_first((lst1, lst2))  # [1, 2, 3]

# 2 lst_group で、渡す
check_first(lst_group)  # [1, 2, 3]

# 3 「lst1, lst2」 で、渡す
check_first(lst1, lst2)
"""
1
4
"""

# 4 *lst_group で、渡す（*が付いている）
check_first(*lst_group)
"""
1
4
"""
```

＃１の時　　期待外れ

関数の引数である`*args`は、引き数をまとめる働きがあります。

すなわち、この場合、もうまとめようがないので、

要素が一つのタプルを受け取ることになります。

つまり、1個のタプルを受け取っているということです。

タプルは要素が一つであっても、タプルです

- print(type((1,)))　➡️　<class 'tuple'>

そして、そのタプルの要素の一つ目が、lst1 なので [1, 2, 3] を返します。

＃２の時　　期待外れ

この場合も、１と同じことをやっているわけです。

つまり、渡しているのは一つなので、リストが二つ入っている一つのタプルを、

渡していることになるわけです。

＃３の時　　期待通り

この時は、lst1 と lst２ の、

2つをまとめて(パッキングして)受け取っていることになります。

ですから、関数内で使う時は、2つの要素があるタプルとして、扱います。

ここでは、2個の要素をパッキングして、受け取っています。

＃４の時　　期待通り

この場合も、３と、結果的にはおなじです。

関数に渡す時*を付けて、**lst_group**としています。

これは、*は、**関数定義で使う時**は、**複数の引数を一つにまとめる**ように働き、

関数を呼び出して使う時は、**要素をアンパッキング(展開)する**ことになっています。

つまり、３の時と同じことをしていることになります。

以上、*args の使い方を、復習してみました

END