B1201

1. Pythonの可変長引数の使い方

Pythonコードを見て「何だこれ？」とつまずきやすいものの 1 つに可変長引数があります。可変長引数の種類は位置引数版とキーワード引数版の二種類あります。

1.1 可変長位置引数 `*args`

関数に渡される位置引数の数が予めわからない場合は、`*args` を指定します。argsという名前は変数名のルールに沿っていれば自由につけてください。

以下の例では、myf という関数に可変長引数 `dogs` を渡し、関数の中では変数 `dogs` がどういう値を持っていて、どういう型なのかを調べます。

```
def myf(*dogs):
  print(dogs)
  print(type(dogs))

myf("Pochi","Taro","Jiro","Tama",)
# 実行結果は以下のようになるはずです。
# ('Pochi', 'Taro', 'Jiro', 'Tama')
# <class 'tuple'>
```

この実行結果を見ると、dogs はtuple 型です。

ちなみにですが、`(1,2,3)` も `1,2,3` も `(1,2,3,)` も `1,2,3,` も全部同じ tuple です。区切り文字だけでtupleとみなされますが、カンマの優先順位が非常に低いために優先度を上げるために丸括弧 `()` で囲うことが多いそうです。

```
t1=(1,2,3)
t2=1,2,3
t3=(1,2,3,)
t4=1,2,3,
print(type(t1),t1)
print(type(t2),t2)
print(type(t3),t3)
print(type(t4),t4)
# 結果はどれも以下のようになるはずです。
# <class 'tuple'> (1, 2, 3)
```

1.2 可変長キーワード引数 `**kwargs`

キーワード引数の数が予めわからない場合は可変長キーワード引数 `**kwargs` を指定します。名前 `kwargs` は先の `args` と同様に自由に変えられます。

可変長キーワード引数は関数内部でどのような型の変数として扱われるかを見ていきましょう。

```
def water(**forms):
  print(forms)
  print(type(forms))

water(壱ノ型='水面斬り',弐ノ型='水車',参ノ型='流流舞い',肆ノ型='打ち潮',)
# 実行結果は以下のようになるはずです。
# {'壱ノ型': '水面斬り', '弐ノ型': '水車', '参ノ型': '流流舞い', '肆ノ型': ' 打ち潮'}
# <class 'dict'>
```

この実行結果のように、可変長キーワード引数は辞書型変数として扱われます。

ちなみにの話として、辞書の場合も最後のカンマの有無に関しては、どちらでも同じ内容の辞書を作成してくれます。中括弧`{}`は省略できません。

```
print({1:1,2:2} == {1:1,2:2,})
# True
```

1.3 位置引数、可変長位置引数、キーワード引数、可変長キーワード引数が混在する場合

位置引数、可変長位置引数、キーワード引数、可変長キーワード引数を関数の引数に混在させる場合は、この順番で配置してください。

```
def f(x,y,*z,a=0,b='zero',**c):
    print('f\'s arguments are...')
    print(x)
    print(y)
    print(z)
    print(a)
    print(b)
    print(c)

f(1,2)
# f's arguments are...
# 1
# 2
# ()
# 0
# zero
# {}
f(1,2,3,4,5)
# f's arguments are ...
# 1
# 2
# (3, 4, 5)
# 0
# zero
# {}
f(2,5,a=1,b='one',c=2,d='two')
# f's arguments are ...
# 2
# 5
# ()
# 1
# one
```

```
# {'c': 2, 'd': 'two'}
```

2. ライブラリの自作方法とimportの仕方

汎用性が高く使いやすい関数は、ライブラリ化してみんなにシェアすると、みんながハッピーになれます。みんなが同じライブラリを使えば、誰かがバグ報告をしてくれたりして、計算の信頼性も上がっていきます。

そのような理由で、ライブラリの作成方法とそれを利用する方法についてここでは学びます。サンプルプログラムとして mylib.py とmain.py を同梱してありますので、それらのソースコードを理解しつつ、それぞれを実行してみましょう。

mylib.py の内容は以下の通りです。コマンドプロンプトで python mylib.py と実行すると`if`文のところが実行されます。(We are HAPPY! というメッセージが出力されると思います。) `if` 文以降のコードは、モジュールとしてではなく通常のpythonプログラムとして動作させる場合にだけ動作しますので、debug用のコードとして利用できます。モジュールとして呼び出されるときには実行されません。

```
def get_happy():
  return 'HAPPY!!'

if __name__ == '__main__':
  print('We','are',get_happy())
```

一方で main.py というファイルには、次のように記述されています。

```
import mylib
print('You are',mylib.get_happy())
```

main.pyの1行目は同じフォルダにある mylib.py の中をimportしています。これでmylib.pyの中で定義された関数やclass オブジェクト（自作オブジェクト）をmain.pyでも定義したことになります。もしファイル名がmy_library_bbxxxxxxxx.py のように非常に長くなっているなら、

```
# main1.py
import my_library_bbxxxxxxxx
print('You are',my_library_bbxxxxxxxx.get_happy())
```

ということになってしまいますが、こういう長い名前を避けたいなら

```
# main2.py
import my_library_bbxxxxxxxx as ml
print('You are',ml.get_happy())
```

というように`as` を使うと短い名前が有効になります。また、ローカルの関数名に付け替えてしまうことでも短い名前にできます。命令文やファイル名は他人が使う時は詳しい方が意味がわかりやすいのですが、入力するときは楽をしたいことが多いでしょうから、適宜入力量を減らして動作するプログラムを「さっ」と作るようにして、かなり汎用性が高いプログラムについては他人に配布することも考慮して長くても詳しい変数名や関数名を使用するようにすると良いと思います。

```
# main3.py
import my_library_bbxxxxxxxx
mlgh=my_library_bbxxxxxxxx.get_happy
print('You are', mlgh())
```

3. 再帰的な関数定義

前回の課題で苦戦している方が多いと感じましたので、もっと簡単な例を紹介してから、ステップアップしていくために例をいくつか与えます。

3.1 `1,2,3,...,n` の総和 `S_n`

ここでは `1,2,3,...,n-1,n` の総和 `S_n` を再帰的に計算する関数 `sum(n)=S_n` を例に解説します。

$$S_0 = 0.0, \quad S_n = \sum_{k=1}^{n} k = \frac{n*(n+1)}{2}.$$

ですから、直接右辺を計算したほうが軽いのですが、今回は再帰的定義の練習と考えてください。（※右辺の公式は検証に使いましょう。）

再帰的な定義は次のように数式で書けます。

$$S_0 = 0.0, \quad S_n = n + S_{n-1} \quad (n \geq 1).$$

これをPythonコードに起こしてみます。

```
def sum(n):
    if n==0:
        return 0.0
    else:
        return n+sum(n-1)
```

この関数を実行します。

```
print(sum(100))
# 5050.0
```

なぜきちんと計算できているのでしょうか？以下では `sum(2)` のケースで順にプログラムの動作を追っていきましょう。もちろん `sum(2)=0.0+1+2=3.0` が期待される値です。

1. `sum(2)` が実行されます。
    1. `n=2` と設定されます。
    2. `n==0` は `False` ですので `else:` ブロックが実行されます。
    3. `n+sum(n-1)=2+sum(1)` を評価するために、`sum(1)` の値が必要です。`sum(1)` が実行されます。
        1. `n=1` と設定されます。
        2. `n==0` は `False` ですので `else:` ブロックが実行されます。
        3. `n+sum(n-1)=1+sum(0)` を評価するために、`sum(0)` の値が必要です。`sum(0)` が実行されます。
            1. `n=0` と設定されます。
            2. `n==0` は `True` ですので、`return 0.0` が実行されます。
        4. `sum(0)=0.0` が判明します。`n+sum(n-1)=1+sum(0)=1+0.0=1.0` が評価されました。
        5. `return 1.0` が実行されます。
    4. `sum(1)=1.0` が判明します。`n+sum(n-1)=2+sum(1)=2+1.0=3.0` が評価されました。
    5. `return 3.0` が実行されます。

2. `sum(2)` の値は `3.0` でした。

3.2 `a_0,a_1,...,a_{n-1}` の総和 `S_n`

より一般に数列 `a` の総和を求めるプログラムも考えてみましょう。さきほどと違うのは一般項が `k` ではなく、`a_k` になっているところです。

$$S_0 = 0.0, \quad S_n = \sum_{k=0}^{n-1} a_k = a_0 + a_1 + \cdots + a_{n-1} \quad (n \geq 1).$$

なお、添字が `0,1,...,n-1` なのは、python のtupleのindexに合わせているからです。この数式を再帰的に定義すると次のように書けます。

$$S_0 = 0.0, \quad S_n = a_{n-1} + S_{n-1} \quad (n \geq 1).$$

`a_k=k+1` の場合が3.1 節での総和ですから、このプログラムはやや一般化されています。

```
def sum(*a):
  n=len(a)
  if n==0:
    return 0.0
  else:
    return a[n-1]+sum(*a[:n-1])

print(sum(1,2,3,))
```

`sum(1,2,)` を呼び出した時の動作を順を追って解説します。

1. `sum(1,2,)` が実行されます。値がでるまで待ちます。
    1. `a` の値として `(1,2,)` が登録されます。
    2. `n=len(a)` の値は `2` です。
    3. `n==0` は `False` ですから `else:` 節が実行されます。
    4. `a[n-1]` =a[1] の値は `2` です。
    5. `a[:n-1]=a[0:1]` の値は `(1,)` です。
    6. `sum=(*(1,))=sum(1,)` が実行されます。値が出るまで待ちます。
        1. `a` の値として `(1,)` が登録されます。
        2. `n=len(a)` の値は `1` です。
        3. `n==0` は `False` ですから `else:` 節が実行されます。
        4. `a[n-1]` =a[0] の値は `1` です。
        5. `a[:n-1]=a[0:0]` の値は `()` です。
        6. `sum(*())=sum()` が実行されます。値が出るまで待ちます。
            1. `a` の値として `()` が登録されます。
            2. `n=len(a)` の値は `0` です。
            3. `n==0` は `True` ですから `return 0.0` が実行されます。
        7. `sum()=0.0` がわかりました。
        8. `a[0]+sum()=1+0.0=1.0` です。
        9. `return 1.0` が実行されます。
    7. `sum(1,)=1.0` がわかりました。

8. `a[0]+sum(1,)=2+1.0=3.0` です。
9. `return 3.0` が実行されます。

2. `sum(1,2,)=3.0` がわかりました。

以下の問いでは、関数定義と動作確認のコードは B1201_lib_bbxxxxxxxx.py の中で記述してください。またB1201_lib_bbxxxxxxxx.py をimportして B1201_main_bbxxxxxxxx.py の中で利用してください。python B1201_main_bbxxxxxxxx.py の実行結果は B1201.txt に保存してください。提出するときは bbxxxxxxxxは自分の学籍番号に置き換えてください。(import ファイル名もbbxxxxxxxxの部分を学籍に置き換えてください)提出するファイルは、ライブラリ B1201_lib_bbxxxxxxxx.py, メインルーチン B1201_main_bbxxxxxxxx.py, メインルーチンを実行した時の出力結果 B1201.txt の３つです。

問(0) 可変長引数`x`の個数を返す`mylen(*x):` 関数を作成してください。

メインルーチンでは、print(mylen(1,1,1,1,1,1,)) を動かしてください。

問(i) nの階乗を再帰的に定義してください。

$$0! = 1.0, \quad n! = n * (n-1)! \quad (n \geq 1).$$

メインルーチンでは、 0!,1!,2!,...,10! を表示させてください。

問(ii) 二項係数 c(n,k) を再帰的に定義してください。

$$c(0,0) = 1.0, \quad c(n,k) = \begin{cases} 1.0 & (k = 0, n) \\ c(n-1,k-1) + c(n-1,k) & (1 \leq k \leq n-1) \end{cases}, \quad (n \geq 1).$$

n=0,1,...,10 まで（k=0,...,n）パスカルの三角形をコマンドプロンプトに表示させてください。（見栄えが良いように工夫してください。）

前回の課題で既に階乗の定義や二項係数の定義を再帰的な定義で提出している人も、モジュール化の練習になりますのでやってください。