B1101

この課題では、自作関数を定義して使用する方法を２通り、関数の亜種であるジェネレータの定義の仕方を解説します。自作関数を作成して呼び出す課題を解いてもらいます。

1. 関数

何度も同じような複雑な処理が複数箇所に現れると入力するのが大変だと思うことがあります。関数定義はそのような場合に入力を少なくするのに役立ちます。

1.1 関数定義の書式

関数の書式はおおよそ以下の通りです。

```
def 自作関数名(引数１、引数２, ...):
  関数の処理
  return 返り値１、返り値２、...
```

自作関数名に使用していい文字は変数と同じでアルファベット（大文字小文字）とアンダースコア_と数字の組み合わせです。ただし、数字から始まる名前は駄目です。また既にpythonで予約されている単語は使用できません（for, while,True, False 等）。日本語もUTF8の中で使える文字がないわけではありませんが使えない文字もあるようです。

1.2 引数と返り値

引数について解説します。

引数を必要としない関数に対しては引数を書きません。またreturn 文の無い関数（返り値がない関数）も定義ができます。

```
def d():
  print('DEBUG!!')
```

この関数を呼び出すときは、次のように呼び出します。

```
d()
```

引数を 1 つ持つ関数は引数用の変数を 1 つ書きます。返り値は return 文を使って表します。次の関数は x をfloat型の値が入ってくると期待し、引数 x が正なら x の値を、そうでなければ 0.0 を返します。

```
def ReLU(x):
  x=float(x)
  if x>0.0:
    return x
  else:
    return 0.0
```

引数を２つ以上とる関数や複数の返り値を返す関数も定義できます。その場合は変数をカンマで区切って列挙します。例えばn個の玉のうちk個の玉を選ぶ組み合わせの数 nCk は引数n,kをとり返り値が１つです。

```
def combination(n,k):
  # n個からk個選ぶ組み合わせの数を算出して result に格納
  return result
```

以下の例は、tuple で複数の値を与えたうち、最小値、中央値、最大値、和、平均、分散、偏差の組を返します。

```
def statistics(t):
  # 処理は省略
  return min, center, max, sum, mean, variance, deviation # ()が省略されたtuple
```

引数には実は 2 種類あって、位置引数とキーワード引数があります。これまでの例は位置引数の例です。

1.2.1 位置引数

位置引数は、その位置にかかれたオブジェクトが該当する位置引数に引き渡されます。例えば、先に定義した combination(n,k) 関数を呼び出す場合には、例えば以下のように呼び出します。

```
combination(4,2)
```

この例では関数内部では n=4, k=2 として扱われます。これが逆に書かれていると、n=2, k=4 として扱われていまいます。このように位置がとても重要です。

（コラム）python における引数や返り値の引き渡しは必ず参照渡しです。なぜなら、pythonにおけるデータはあらゆるものがオブジェクトだからです。オブジェクトデータはたくさんの情報の集まりになっています。int型の1ですら、1という数値以外の情報を持っています。これらをコピーして渡す値渡しでは効率が悪くなります。参照すなわちオブジェクトのデータがメモリのどこにあるか（id()関数で取得可能）を伝える参照渡しの方が効率がよくなります。

1.2.2 キーワード引数

キーワード引数はデフォルト値を指定した引数です。デフォルト値というのは、その引数を省略した場合に、設定される値のことです。キーワード引数の書式は引数を定義するところで `変数名=デフォルト値` の形で与えます。

次の例は位置引数 L とデフォルト値が False のキーワード引数 debug の関数定義の例です。

```
def sum(L,debug=False):
    n=len(L)
    result=0
    for i in range(n):
        result+=L[i]
        if debug:
            print(i,L[i],result) # 途中経過の表示
    return result
```

以下は debug を省略した呼び出し例です。省略すると debug の値は False に設定されますので、途中経過を表示しなくなります。

```
y=(3,2,5,9,1)
goukei=sum(y)
print(goukei)
# 20
```

debugを省略しないでTrueを指定すると、途中経過も表示されます。

```
y=(3,2,5,9,1)
goukei=sum(y,debug=True)
print(goukei)
# 0 3 3
# 1 2 5
# 2 5 10
# 3 9 19
# 4 1 20
# 20
```

1.3 ローカル変数とグローバル変数

python の変数にはローカル変数とグローバル変数という類別方法があります。

ローカル変数は関数定義内で代入された変数のことを指します。ローカル変数は関数を実行している間のみ参照できます。関数外からローカル変数を参照することはできません。例えば、次のコードは変数`x`が見つかりません。

```
def func():
  x=3
  return x

print(x) # x は関数定義以外では定義されていない
```

グローバル変数はローカル変数以外の関数定義の外部で代入された変数のことです。グローバル変数は関数内部からも参照できます。次のコードは、動きます。

```
a='ABC'
def func():
  print(a)
func()
# 'ABC' が表示されます。
```

しかしながら、関数定義内で代入で a が再定義されると、グローバル変数ではなくなります。

```
a='ABC'
def func():
  a='DEF'
  print(a)
func()
# 'DEF' が表示されます。
```

なお、関数定義内で`a`をグローバル変数として扱いたい場合は`global a`というように明示します。

```
a='ABC'
def func():
  global a
  a='DEF'
  print(a)
func()
# 'DEF' が表示されます。
```

このコードの場合は a はグローバル変数ですので、 'DEF' という文字列オブジェクトに変数 a を関連付け直します。

2. 無名関数 lambda 式

```
def f(x1,x2,x3,...,xn):
  return 計算式
```

のようにreturn する計算式が 1 行でかける場合、次のようなlambda式でも表現可能です。

```
f=lambda x1,x2,x3,...,xn:計算式
```

`f=` の部分は無名関数 lambda の関数オブジェクトに f という名前をつけています。def を使った関数は最初から名前をつけています。関数を一度定義して、 2 度使い回さない場合は `f=` のように名前をつけずに使います。主に関数の引数に無名関数を直接書き込むときは名前をつけないで済みます。

以下はf(x)がxの二乗を返す場合の定義です。単純な関数の定義はlambda 式の方が簡単に書けます。

```
f=lambda x:x**2
```

次の例の `vectorize()` 関数は関数を引数にとり、返り値も関数になっています。ちょっとややこしいですが、 1 変数にしか対応しない `f` から多変数 `L` に対応する `vectorize(f)()` という関数を作成しています。このように元の関数を壊さずに機能拡張をする関数のことをデコレータ（飾り付け装置）といいます。`vectorize` はデコレータの一種です。

```
def vectorize(f):
  def result(L):
    R=[]
    for x in L:
      R.append(f(x))
    return R
  return result

F=vectorize(lambda x:x**2)
s=tuple(range(10))
t=F(s)
print(s)
# (0,1,2,3,4,5,6,7,8,9)
print(t)
# (0,1,4,9,16,25,36,49,64,81)
```

この例のように関数の引数に直接関数を定義する方が単純なときに、lambda式は便利です。

3. ジェネレータ

関数定義における return は一回しか値を返せませんが、range(10) は0,1,2,3,4,5,6,7,8,9 という１０個の値を返しているように見えます。このような何度も値を出す装置をジェネレータといいます。ジェネレータは関数定義とほとんど同じ書式ですが、 return 文の代わりに yield 文を用いて値を吐き出します。

```
def expr(x):
  if x>0:
    return 'x<end'
  else:
    return 'x>end'

def myrange(start,end,step=1):
  x=start
  while eval(expr(step)):
    yield x
    x+=step

print(list(myrange(0,10)))
print(list(myrange(1,11)))
print(list(myrange(10,0,-1)))
```

（コラム）ここで `eval()` という関数を使っています。この関数は文字列をpython言語の命令とみなして実行します。例えば、 `s='print(x)'` はただの文字列でしかありませんが、 `eval(s)` とするとprint(x) を実行してくれます。文字列操作を通じて、プログラムを作るプログラムを書く時にとても便利です。例えば'x**n' を 'n*x**(n-1)' に直す操作を文字式の微分公式として覚えさせて、後で数値に直したいときには `eval()` によって命令文として実行してしまうことができます。

問(i) 引数 n を持ち、n!を返り値に持つ関数 Factorial(n) を定義して、 Factorial(10) の値を表示するプログラムを書いてください。

これらの計算を行うプログラムb1101.pyとその出力結果b1101.txt を提出してください。