X. Python 文法（2）

目的

Python でより大規模なプログラム開発する方法を学ぶ

14. Python の文法：ユーザー定義関数

１つのプログラムが長くなると，プログラムを読むのが難しくなるし，開発も大変になる。プログラムを部品に分け，分業で開発することができれば，開発が省力化できる。これを行う仕組みの基礎の１つが関数である。すでに Python 標準の入出力関数や数学関数などについては取り上げた。この関数は開発者が自ら定義してプログラムに組み込むことができる。

14.1 ユーザー定義関数と引数および戻り値

(1) def 文による関数の定義

```
>>> def addfunc( a, b ):
>>>     c = a + b
>>>     return c
>>>
>>> x = 1.0
>>> y = 2.0
>>> z = addfunc( x, y )
>>> print( z )
```

・「`def` *`funcname`*`( arg1, arg2,… ):`」のように書くと**関数** *funcname* が定義される。
・「`arg1, arg2,…`」は**引数** (arguments)で，関数への入力。
・関数内はインデントして書く。
・変数は関数呼び出す側と関数内でローカルに定義される（別々に値を保存)。
・引数内の変数を関数内部で書き換えても，呼び出した側の値に影響しない。
　Fortran などでは引数になっている変数の値が書き変わるので，仕様が異なる。
・「`return c`」で関数内の処理を終わって呼び出し側に戻る。
・戻るときに関数の値として *c* の値が返される。これを**戻り値**(return value)という。
・「`z = addfunc( x, y )`」で戻り値が変数 *z* に代入される。

(2) 戻り値のない関数

```
>>> def printname( name ):
>>>     print( 'Your name is ', name )
```

```
>>>     return
>>>
>>> myname = input( 'My name is ' )
>>> printname( myname )
```

・キーボードから名前を入力すると，画面に名前を入力する関数。

(3) 引数も戻り値もない関数

```
>>> def printhoge():
>>>     print( 'hoge~~~~' )
>>>     return
>>>
>>> printhoge
>>> printhoge
>>> printhoge
```

・呼び出すと「hoge~~~~」と画面に出力する関数。
・関数内部で処理が完結している。

14.2 グローバル変数

(1) グローバル変数の定義

```
>>> def prtnhoge( ):
>>>     nhoge = n * 'hoge'
>>>     print( nhoge )
>>>     return
>>>
>>> global n
>>> n = int( input( '# of repeats = ' ) )
>>> prtnhoge( )
```

・「global *n*」で**グローバル変数** *n* を定義。
・グローバル変数は 1 組で実行されるプログラム群で共通の値を持つ。
・呼び出し側で global 宣言した変数は関数内では**読み込み専用**。
　グローバル変数を関数内で書き換えようとすると実行時にエラーになる。

(2) グローバル変数を使って値を戻す

```
>>> def addfunc( a, b ):
>>>     global c
>>>     c = a + b
```

```
>>>     return
>>>
>>> x  = 1.0
>>> y  = 2.0
>>> c  = 1.5
>>> c0 = c
>>> addfunc( x, y )
>>> print( 'c is changed from', c0,  'to', c )
```

・**グローバル宣言を関数内**で行うと，関数内で書き換えたグローバル変数値を呼び出し側に渡すことができる。
・呼び出し側に戻したい変数が多いときなどに便利かもしれない。
・プログラムのファイルが別になるとグローバル宣言していることを忘れてしまったなどで起きるバグに注意が必要。

14.3 モジュール

(1) 関数プログラムを個別ファイルにする方法

メインモジュール（呼び出し側プログラム）：func_test.py

```
import functest_mod
x = [ 1.0, 2.0 ]
y = [ 3.0, 4.0 ]
z = [ 0.0, 0.0 ]
print( x[ 0 ], x[ 1 ] )
print( y[ 0 ], y[ 1 ] )
z = functest_mod.addfunc( x, y )
print( z )
```

関数モジュール：functest_mod.py

```
def addfunc( x, y ):
    xy = [ 0.0, 0.0 ]
    xy[ 0 ] = x[ 0 ] + y[ 0 ]
    xy[ 1 ] = x[ 1 ] + y[ 1 ]
    return xy
```

・２つのプログラムを同じフォルダの別ファイル「*prog_1*.py」「*modulename*.py」などに保存する。
・「*prog_1*.py」を実行すると「*modulename*.py」の中の関数が呼び出される。
・ここでは，「func_test.py」と「functest_mod.py」。
・プログラムの部品を**モジュール**とよび，関数のモジュールを**関数モジュール**という。

・「import *modulename*」でモジュールをインポート
・「*modulename.funcname*( )」で関数を呼び出す。
・ここでは,「z = functestmod.addfunc(x, y)」。

１５．クラス

クラスとは型のことである。整数型や文字型などの変数，ファイル型オブジェクトなど初めから定義されているクラスもあるが，プログラム作成者が定義できる。リスト型変数やファイル型オブジェクトと同様，定義したクラスにメソッドを持たせることもできる。

15.1 クラスの基本

(1) 変数のインスタンス

```
>>> import datetime
>>> ymd = datetime.date( 2020, 6, 10 )
>>> print( ymd )
>>> y = ymd.year
>>> m = ymd.month
>>> d = ymd.day
>>> print( y, '年', m, '月',  d, '日' )
```

・「datetime.date( )」は datetime モジュールにある日付を生成する関数。
・入力は年・月・日の数値。
・「ymd」は **date 型変数**として定義される
・「ymd.year」は date 型変数から年の値を取り出す。
・「ymd.year」の.year を**インスタンス**，インスタンスを持つ変数を**インスタンス変数**という。

(2) クラスとインスタンスの定義

クラス呼び出し側プログラム：class_score.py

```
import smyclass as smc
st1 = smc.seiseki( 'Sato', 80, 70, 90 )
p11 = st1.pt1
p12 = st1.pt2
p13 = st1.pt3
print( p11, p12, p13 )
pm1 = ( p11 + p12 + p13 ) / 3
print( 'average of', st1.name, pm1 )
```

クラスモジュール：smyclass.py

```
class seiseki:
    def __init__( self, name, pt1, pt2, pt3 ):
        self.name = name
        self.pt1 = pt1
        self.pt2 = pt2
        self.pt3 = pt3
```

・「import smyclass as smc」はクラスモジュール smyclass をインポート。
・「st1 = smc.seiseki( )」で**クラス変数**へ値を代入。
・「p11 = st1.pt1」クラス変数 seiseki の pt1 インスタンスの値を代入
・「class seiseki:」はクラス変数 seiseki を定義することを宣言。
・「def __init__( ):」でクラス変数の内容（インスタンス）の定義。
・「self」の部分は外から見えない変数で，適当な名前でよい。

15.2 クラスのメソッド

(1) クラスのメソッドの定義

クラス呼び出し側プログラム：class_score_2.py

```
import smyclass2 as smc
st1 = smc.seiseki( 'Sato', 80, 70, 90 )
p11 = st1.pt1
p12 = st1.pt2
p13 = st1.pt3
print( p11, p12, p13 )
st1.avr( )
```

クラスモジュール：smyclass2.py

```
class seiseki:
    def __init__( self, name, pt1, pt2, pt3 ):
        self.name = name
        self.pt1 = pt1
        self.pt2 = pt2
        self.pt3 = pt3

       def avr( self ):
           avpt = ( self.pt1 + self.pt2 + self.pt3 ) / 3
            print( 'average of', self.name, avpt )
```

・「st1.avr( )」は**クラスメソッド**を呼び出し。

・「`def avr( self ):`」でクラスメソッドを定義。
・「`self`」の部分はクラスを定義した変数と同じにする。

16．プログラム例

例7． $y = \sin x$ のテイラー展開

$\sin x$ を組み込み関数とテイラー展開により 1 周期計算し，画面にプロットする。

In [5]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import math

#  input maximum order
nmax  = int( input( 'Maxmum order n of Taylor Series? ' ) )
mx = int( input( 'Number of intervals in x? ' ) )

#  Maxmum value of aux. index k
kmax = ( nmax + 1 ) // 2
print( 'max k (n = 2k - 1) = ', kmax )

# interval of x
dx = 2 * math.pi / mx

# Initialize list variables
fn =  [ 0.0 for n in range( 0, nmax+1 ) ]     # factorial k
a  = [ 0.0 for k in range( 0, kmax+1 ) ]      # Tayor coefficient

x  =  [ 0.0 for i in range( 0, mx+1 ) ]       # x_i
s1 =  [ 0.0 for i in range( 0, mx+1 ) ]       # sin x by math module function
s2 =  [ 0.0 for i in range( 0, mx+1 ) ]       # sin x by Taylor Series

# calculate coefficients of Talyor Series
fn[ 0 ] = 1
for n in range( 1, nmax+1 ):
   fn[ n ] = fn[ n - 1 ]  * n
   print( n, fn[ n ] )

for k in range( 1, kmax+1 ):
   n      = 2 * k - 1
   sig    = ( -1 )**( k - 1 )
   a[ k ]  = sig / fn[ n ]
   print( n, k, sig, fn[ n ], a[ k ] )

# calculate sine function
for i in range( 0, mx+1 ):
   x[ i ]  = dx * i
   s1[ i ] = math.sin( x[ i ] )    # by math module function

   s2[ i ] = 0.0
   for k in range( 1, kmax+1 ):
      n = 2 * k - 1
      s2[ i ] = s2[ i ] + a[ k ] * x[ i ] **n   # by Taylor Series

# graph
plt.plot( x, s1 )
plt.plot( x, s2 )
plt.grid( color = '0.8' )
plt.show
```

```
Maxmum order n of Taylor Series? 13
Number of intervals in x? 100
max k (n = 2k - 1) =  7
1 1 1 1 1.0
3 2 -1 6 -0.16666666666666666
5 3 1 120 0.008333333333333333
7 4 -1 5040 -0.0001984126984126984
9 5 1 362880 2.7557319223985893e-06
11 6 -1 39916800 -2.505210838544172e-08
13 7 1 6227020800 1.6059043836821613e-10
```

Out[2]: <function matplotlib.pyplot.show(*args, **kw)>

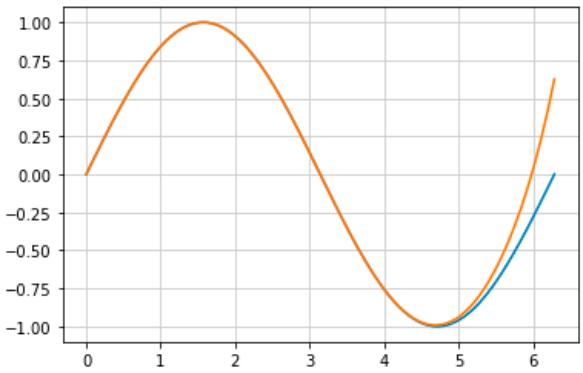

図 16.1　テイラー展開により $\sin x$ を求めるプログラムとそのグラフ出力

例 8. $y = \sin x$ のテイラー展開

$\sin x$ を組み込み関数とテイラー展開により 1 周期計算し，CSV ファイルに保存する。
処理を関数に分けてモジュール化している。

(1) メインモジュール

```
import math
import sineTaylor as st

#  input maximum order
nmax = int( input( 'Maxmum order n of Taylor Series? ' ) )
mx   = int( input( 'Number of intervals in x? ' ) )

#  Maxmum value of aux. index k
kmax = ( nmax + 1 ) // 2
print( 'max k (n = 2k - 1) = ', kmax )

# interval of x
dx = 2 * math.pi / mx

# Initialize list variables
# a  = [ 0.0 for k in range( 0, kmax+1 ) ]        # Tayor coefficient

x  = [ 0.0 for i in range( 0, mx+1 ) ]          # x_i
s1 = [ 0.0 for i in range( 0, mx+1 ) ]          # sin x by math module function
s2 = [ 0.0 for i in range( 0, mx+1 ) ]          # sin x by Taylor Series

# Call the function to calculate Taylor coefficinets

a = st.coef_Tsine( nmax, kmax )

#for n in range( 1, nmax+1 ):
#    print( n, a[ n ] )

# calculate sine function
for i in range( 0, mx+1 ):
    x[ i ]  = dx * i
    s1[ i ] = math.sin( x[ i ] )              # by math module function
    s2[ i ] = st.tysine( a, x[ i ], nmax )

fsin = open( 'sin_taylor.csv', 'w' )
for i in range( 0, mx+1 ):
    print( x[ i ], ',', s1[ i ], ',', s2[ i ], file = fsin )
```

図 16.2　テイラー展開により $\sin x$ を求めるプログラム：メインモジュール

・テイラー級数の係数の計算と級数の計算を関数モジュール化
・関数を使用した方がプログラムをすっきりとした形で書ける
・関数の名前，引数や戻り値に注意
・画面キャプチャは jupyter note book のテキストエディタ

(2) 関数モジュール

```
def coef_Tsine( nmax, kmax ):
    # calculate coefficients of Taylor Series

    print( 'in coefTsine ', nmax, kmax )

    # initailize local list variables
    fn = [ 0.0 for n in range( 0, nmax+1 ) ]        # factorial k
    a  = [ 0.0 for k in range( 0, nmax+1 ) ]        # Tayor coefficient

    fn[ 0 ] = 1
    for n in range( 1, nmax+1 ):
        fn[ n ] = fn[ n - 1 ]  * n
        # print( n, fn[ n ] )

    for k in range( 1, kmax+1 ):
        n      = 2 * k - 1
        sig    = ( -1 )**( k - 1 )
        a[ n ]  = sig / fn[ n ]
        print( n, k, sig, fn[ n ], a[ n ] )

    return a

def tysine( a, x, nmax ):
    # calculate sine value

#    for n in range( 1, nmax+1 ):
#        print( n, a[ n ] )

    ts = 0.0
    for n in range( 1, nmax+1, 2 ):
        ts = ts + a[ n ] * x **n                    # summation to make Taylor Series

    return ts
```

図 16.3　テイラー展開により $\sin x$ を求めるプログラム：関数モジュール

・２つの関数「coef_Tsine」「tysine」からなるモジュール
・関数内の変数はすべてローカル変数を用いていることに注意。
・引数やリスト型変数の初期化位置などに注意。
・添字の扱いを一体型プログラムと少し変えてある。