IX. Python 文法（1）

目的

プログラミング言語 Python を正しく記述するための文法を学ぶ

7．Python の基本概念

7.1 Python の基本的な書き方

(1) 先頭の空白を入れてはいけない
(2) `print` などの予約語や変数の途中に空白を入れてはいけない
(3) インデントは 4 文字以上の空白，意味を持つ
(4) 文末に「\（バックスラッシュ）」を入れると次の行に続く
(5) 文字列を入力するコーテーションマークはシングル「`'strings'`」，ダブル「`"strings"`」どちらでも良いが，同じものを使う
(6) #以降の行はコメント，プログラムの理解を助ける注意書き
(7) 複数コメント行は「"""〜"""」（〜は 1 行〜複数行）
(8) 式の途中などに空白を上手に入れるとプログラムがきれいで見やすくなる

7.2 オブジェクト

(1) データと機能をまとめたものをオブジェクトという。
(2) たとえば，アプリのメニューのように，「保存」などの文字列データやクリックするとファイルに書き込む機能など
(3) オブジェクトを組み合わせていくと，効率的にプログラムの開発ができる。いわば，プログラムのアセンブリパーツという感じ。

7.3 変数

(1) 数値や文字列などのデータを格納しておく場所を変数という。
(2) 変数に値を入れるには，代入文や入力関数を使う。
(3) 変数に複数の値を持たせたものをリストという。Fortran などの配列変数に相当。

7.4 関数

(1) 関数はプログラムで使われる定型的な処理をまとめたもの。
(2) 基本「`a = func( )`」という形で書く。
(3) 「`( )`」の中を**引数**(argument)という。`math.pi` のように引数のない関数もある。
(4) 数学関数など多くの関数などは処理の後，戻り値(return value)を返す。

(5) 数学関数などはライブラリが必要。関数名が「*libirary.fun*」になる。
(6) プログラムの部品の小さい単位のような感じ。
(7) 元々組み込まれているもの，ライブラリにあるもの以外に自分で定義することもできる。

7.5 クラス

(1) 「型」のこと。整数型・浮動小数型・文字型のほか，Python 標準のクラスとして日付型・時間型・ファイル型がある。
(2) 用意されている型のほか，自分で定義することも可能。

8．Python の文法：変数と演算

8.1 変数

(1) 変数とは？

```
>>> a = 3
>>> b = 3.0
>>> c = '3.00'
>>> print ( a, b, c )
```

・変数：数値や文字列などのデータを格納しておく場所。

```
a = 3
```

のような式は等式ではない。「=」で結ばれた式を**代入文**といい，右辺の値や計算結果を左辺の変数に入れる命令である。

(2) 変数の型（クラス）

今度は，b に再度代入する。

```
>>> b = 3.00
>>> print ( a, b, c )
```

・上の(1)(2)で違い生じたであろうか。
・これは，変数の型によって説明することができる。
　a: 整数型
　b: 浮動小数型
　c: 文字型
・python では**変数の型が代入の際，自動判別**される。
・ほかに「論理（真偽）型」「リスト型」「ファイル型」…がある。

問：(1)(2)の違いが生じない原因を「変数の型」によって説明せよ。

8.2 算術演算

(1) 四則演算

```
>>> 1 + 2
>>> 5 * 3
>>> 10 / 3
>>> 10 // 3
>>> 10 % 3
>>> 2 ** 4
```

(2) 四則演算

「+」 加算
「-」 減算
「*」 積算
「**」 べき乗
「/」 除算
「//」 除算の商（整数）
「%」 剰余
「( )」括弧

・べき乗は Excel のように「^」ではないので注意
　演算の順序は数学と同様：括弧→べき乗→積・除→加・減

(3) 変数の使用

```
>>> a = 3.5
>>> b = 6
>>> a * b
```

・浮動小数型と整数型の演算のときは，浮動小数型に自動変換される

(4) 複合演算

```
>>> a = 3
>>> a += 3
>>> a
>>> a = 3
>>> a = a + 3
>>> a
```

・ある変数に演算して，同じ変数に代入(上書き)する演算。総和の計算などに使用。

「+=」 a += b　a = a + b

「-=」 a -= b　a = a - b

「*=」 a *= b　a = a * b

「/=」 a /= b　a = a / b

「//=」a //= b　a = a // b

「%=」 a %= b　a = a % b

「**=」a **= b　a = a ** b

8.3 文字列の連結式

(1) 文字の連結式

```
>>> a = 'My name is '
>>> b = 'your name.'　(your name のところはあなたの名前)
>>> c = a + b
>>> print ( c )
```

・My name is …という英文が出力されたはずである。

・文字変数を「足す」と**文字列が連結**される。

(2) 文字型と整数型の加算

```
>>> a = 'I am '
>>> b = ##　(##のところはあなたの年齢の数字)
>>> c = 'years old.'
>>> d = a + b + c
>>> print ( d )
```

・これは d の式のところでエラーになる。

・一度，数値型に決まった変数が文字型に**自動変更されることはない**からである。

・エラーにならないようにするには

```
>>> b = '##'
>>> d = a + b + c
>>> print ( d )
```

とやり直すとエラーは出ない。

```
>>> b = str( 10 )
```

・このように型変換関数もある

(3) 文字型と整数型の掛け算は可

```
>>> 'Hoge' * 3
```

```
>>> 'HogeHogeHoge'
```

8.4 論理演算

(1) 比較演算

```
>>> a = 5
>>> a < 5
>>> a >= 5
```

・比較演算は式が真か偽か判断する。
・出力は True か False
・単独で使うことは少なく，多くの場合，分岐処理で使用

(2) 比較演算子

「==」　等しい
「!=」　等しくない
「<」　小なり
「<=」　以下
「>」　大なり
「<=」　以上

(3) 複合論理演算子

「and」 論理積
「or」　論理和
「not」 否定(反転)

・２つの比較演算式を結びつける。だいたい and と or だけで間に合う。

```
>>> a = 5
>>> a > 4 and a < 6
>>> a > 4 or a < 6
>>> not( a > 4 and a < 6 )
```

8.5 リスト

(1) リスト

```
>>> a = [ 1, 2, 3, 4, 5, 6 ]
>>> a
```

・１つの変数に多数のデータを持たせるものを**リスト**という。
・Fortran や C の配列変数のような役目

(2) リストのインデックス（添字）

```
>>> a[ 2 ]
```

・リストのインデックス（添字）は 0 から始まる。6 個のときは 0〜5

(3) 空のリスト

```
>>> a = [ ]
```

・「[ ]」内に何も書かないと**空のリスト**（長さゼロのリスト）a [i]が作られる。
・代入文で代入するとエラーになる。
・値を代入するときは，リストの**メソッド**「.append」を用いてリストを付加をする。

(4) リストの初期化

```
>>> a = [ 0.0 for i in range( 0, n+1 ) ]
```

・インデックス i = 0〜n に対して a [i] = 0.0 が代入される。

(5) リストの連結

```
>>> b = [ 7, 8, 9, 10 ]
>>> c = a + b
>>> c
```

・２つのリストが連結されて 1 つになる。

(6) 多次元のリスト

```
>>> a = [ [ 0.0 for i in range( 0, nx+1 ) ] for j in range( 0,ny+1 ) ]
>>> for j in range( 0, ny+1):
>>>     for i in range( 0, nx+1 ):
>>>         print( i, j, a[ i ][ j ] )
```

・Excel シートのように行と列を持つリストを２次元のリストという。
・２つの添字を持つ a_{ij} ような変数を表すのに便利。
・「a[i][j]」のように引用する。
・３つや４つの添字を持つリストも作ることができる。

９．Python の文法：関数

9.1 関数

(1) 型変換関数

```
>>> a = '6.4e6'
>>> float( a )
```

```
>>> a
```

・変数の型を変換する関数である。

`int( a )` ： 整数型へ変換

`float( a )` ：浮動小数型へ変換

`str( a )` ： 文字型へ変換

(2) 出力関数

```
>>> a = 3.0
>>> b = 2.0
>>> print( a, b )
```

・引数にある変数の値を画面やファイル表示する。

・1 行ずつ書き込みを行う。1 行終わると (改行コードを書き) 改行する。

(3) 入力関数

```
>>> a = input( 'a = , ' )
>>> b = int( input( 'b = , ' ) )
>>> c = float( input( 'c = , ' ) )
>>> a, b, c
```

・値をキーボードやファイルから引数にある変数へ読み込む。

・「`'a = '`」の部分は入力を促すプロンプトとして画面に表示される。

・入力された**文字列**が戻り値

・整数型や浮動小数型に変換する関数が必要

(4) 補助入出力関数

```
>>> f1 = open( '/Users/username/dirname/filename', 'w' )
>>> print( x, y, file = f1 )
>>> f1.close( )
```

・open は変数 f1 にファイルを結びつける関数。f1 は**ファイル型オブジェクト**。

・「/Users/*username*/*dirname*/*filename*」のところはディレクトリ（フォルダ）のパス＋ファイル名。

・ここでのディレクトリ名は絶対パス（「/」から始まっている）が相対パスでも良い。その場合，実行するディレクトリの位置に注意。

・同じディレクトリの場合は「*dirname*」は不要。

・Windows 標準ではディレクトリ区切りは「\」「¥」だが，Python では「/」を使う。

・print 関数において，「file=f1」で書き込み先を f1 で結び付けられたファイルに指定している。

・「f1.close( )」は f1 に対するファイル操作の**メソッド**（後述）でファイルを閉じる。

(5) 数学関数

```
>>> import math
>>> p1 = meth.pi
>>> y = math.sin( p1 / 6.0 )
>>> p, y
```

・数学関数を呼び出すときには数学ライブラリをプログラムにインポートする
必要がある。
・「import math」でインポートしている。「math」のところはライブラリ名。
・「math.pi」：π（引数なし），「math.sin( )」：sine

(6) ユーザー定義関数
・ユーザーが独自に機能を作る関数。プログラムの部品になる。
・Fortran だと，関数のほかサブルーチンというものがある。C だと関数や void 関数。

10. Python の文法：メソッド

10.1 リストとメソッド

(1) リストへの値の追加

```
>>> d = [ 1, 3, 5, 7, 9 ]
>>> d.append( 13 )
>>> d
```

・リストにはリストを操作する機能がある。
・機能のことを**メソッド**という。
・「.append」はリストの後ろに値を加える。

(2) リストへの値の挿入

```
>>> d.insert( 5, 11 )
>>> d
```

・メソッド「.insert」はリストに値を挿入する。
・最初の数値「5」がインデックス，コンマの後ろ「11」が挿入値である。

(3) リストの値の削除

```
>>> d.pop( 3 )
```

```
>>> d
>>> d.remove( 3 )
>>> d
```

・メソッド「.pop(3)」はインデックス 3 の削除(取り出し),「.remove」は値 3 を削除。

(4) 文字列の分割

```
>>> line = '1.0, 2.0'
>>> a, b = line.split( ',' )
>>> a
>>> b
>>> c = float( a )
>>> d = float( b )
>>> c
>>> d
```

・「.split( )」は文字列を( )内に指定した文字列で分割するメソッド。
・文字型変数 line に入っている文字列「1.0, 2.0」を「,」で分割し，a と b に代入。
・a と b も文字変数。
・スペース区切りは「.split(' ')」とする。
・CSV やスペース切りファイルを読み込むときに使う。

10.2 ファイルとメソッド

(1) ファイルへの出力

```
>>> f1 = open( 'dirname/filename', 'w' )
>>> f1.write( str( x ) + '  ' + str( y ) + '\n' )
```

・「f1.write( )」はファイル型オブジェクト f1 に対するメソッドで，ファイルへ
書き込む。
・9.2 (4)の print 関数とほぼ同じ結果が得られる。
・( )内は文字列・文字型変数あるいは文字の連結式
・引数は 1 つでなければならない。つまり，複数の変数をかけない。
・また，自動改行もしない。
・このため，文字列にして複数の変数やスペース，改行コードを連結している。
最後の「'\n'」は改行コードの文字表示。日本語 Windows では「¥」と表示。
・macOS では「¥」だとエラーになるので，必ず,「\」を使う。
・他に,「.writeline( )」や「.writelines( )」がある。

(2) ファイルのクローズ

```
>>> f1 = open( '/dirname/filename', 'w' )
>>> f1.close( )
```

・「f1.close( )」はファイルの最後に EOF (end of file)コードを書き，ファイルを閉じる。
・ファイルはプログラム終了前に閉じる必要があるので，必須である。

(3) ファイルの 1 行読み込み

```
>>> f1 = open( '/dirname/filename', 'w' )
>>> row = f1.readline( )
>>> a, b = row.split( ',' )
>>> a, b
```

・「f1.readline( )」はファイルの 1 行だけを読む。
・「f1.readlines( )」はファイル全体を 1 つの文字列として読む。

11．Python の文法：分岐処理

プログラムは基本的に上の行から下の行へ順番に処理される。しかし，場合によって処理の仕方を変えたいことは多く発生する。このような処理が分岐処理である。分岐処理の要素は, 条件の判定をするための比較演算を組み合わせた論理式と分岐後の処理からなる。

11.1 if 文による制御

(9) if~else 文

```
>>> scor = int( input( 'Your score ? ' ) )
>>> if scor >= 60:
>>>     print( 'Congratulation! You are passed.' )
>>> else:
>>>     print( 'You are not passed.' )
```

・「if *condition equation* (*logical equation*):」のように書く。
・判定後の処理の部分はインデントする。
・if 文の直後は条件式が「True」の場合の処理。
・「else:」以下は条件式が「False」の場合の処理。

(10)　if~elif~else 文

```
>>> scor = int( input( 'Your score ? ' ) )
```

```
>>> if scor >= 90:
>>>     print( 'Your score is S.' )
>>> elif scor >= 80:
>>>     print( 'Your score is A.' )
>>> elif scor >= 70:
>>>     print( 'Your score is B.' )
>>> elif scor >= 60:
>>>     print( 'Your score is C.' )
>>> else:
>>>     print( 'Your score is D.' )
```

・「elif」は else if を略したもの。上の条件が当てはまらないもの(偽)の中での条件。

(11)　if 文のネスト

```
>>> scor1 = int( input( 'Your test score ? ' ) )
>>> if scor1 >= 60:
>>>     print( 'Congratulation! You are passed' )
>>> else:
>>>     print( 'Submit the additional report' )
>>>     repo = input( 'Did you submit? ' )
>>>     if repo == 'y' or repo == 'yes':
>>>         print( 'You are passed' )
>>>     else:
>>>         print( 'You are not passed' )
```

・if 文を入れ子にすると複雑な条件判定に対応できるようになる。
・入れ子を**ネスト**という。
・２つめの条件式の論理演算子「or」は論理和。
　「y」「yes」２通りの入力どちらでも「True」になる。

12. Python の文法：繰り返し処理

同じ処理・演算を異なるデータや値に対して行うことを繰り返し処理という。これが高速にできることがコンピューターを使う理由の最大のものである。繰り返し処理の要素は，繰り返し行う処理そのものと，繰り返しを行う回数や条件の制御からなる。

12.1 回数によるループ制御

(1) for 文

```
>>> for num in range( 1, 11 ):
>>>     print( num, 'time repeats' )
```

・「for *integer_num* in range(*first_num*, *final_num* +1):」のように書く。
・繰り返す処理の部分はインデントする。
インデントしていない行が下にあると，そこからループ外になる。
・「*integer_num*」は整数型変数でカウンターの役目をする。
・「range( )」は range 関数。リスト[1, 2,…10]を生成して for に渡している。
・カウンターの最初は 「*first_num*」。書かないときは自動的に 0。
・カウンターの最後は「*final_num*」。
つまりカウンターが「*final_num* +1」になるとループを脱出する。

12.3 条件によるループ制御

(1) while 文

```
>>> num = 1
>>> while num < 11:
>>>     print( num, 'time repeats' )
>>>     num += 1
```

・「while *condition equation*:」のように書く。
・英語の while 節と同じで，条件式が「True」の間，処理を繰り返す。
・「for」と同様繰り返す処理の部分はインデントする。

(2) 強制ループ脱出

```
>>> a = 5.0
>>> b = 1.1
>>> eps = 1.0e-6
>>> for iter in range( 1, 101 ):
>>>     a = a / b
>>>     print( iter, a )
>>>     if a < eps:
>>>         break
```

・条件式「a < eps」が「True」になったとき，ループを中止して脱出。
・脱出後はループ以下の処理を行う。

(3) ループ内処理スキップ

```
>>> for n in range( 1, 100 ):
>>>     if n % 2 == 0:
```

```
>>>         continue
>>>     sn = s + n
>>> print( 'Summation of odd number to ', n, ' is ', sn )
```

・奇数の総和を求めるプログラム,
・条件式「b % 2 == 0」が「True」(偶数)のとき,「c = c + b」をスキップして「for」に戻る

13. プログラム例

例 1． $y = \sin x$ のプロット

$\sin x$ を 1 周期計算し，画面にプロットする。

In [6]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import math
"""
plot sine curve
"""
# input amplitude
a = float( input( 'amplitude =  ' ) )

# number of calculating points
n = 50
dx = 1.0 / n

#  initialize lists
x = [ 0.0 for i in range ( 0, n+1 ) ]
y = [ 0.0 for i in range ( 0, n+1 ) ]
# x = [ ]
# y = [ ]
#for i in range ( 0, n+1 ):
#    x.append ( 0.0 )
#    y.append ( 0.0 )

#  Calculate x and sin x
for i in range( 0, n+1 ):
    x[ i ] = dx * i
    y[ i ] = a * math.sin( 2.0 * math.pi * x[ i ] )

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

```
amplitude =  1.0
```

Out[6]: <function matplotlib.pyplot.show(*args, **kw)>

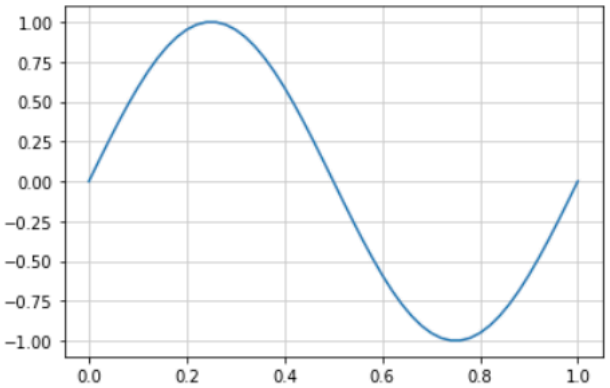

図 13.1　$y = \sin x$ をプロットするプログラム

例２． リサジュー図形のプロット

リサジュー図形を計算し，画面にプロットする。

In [17]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import math
#
#  plot Ressajous figure
#
# input parameters
n1 = int( input( 'frequency n1 = ' ) )
n2 = int( input( 'frequency n2 = ' ) )
p_diff = float( input( 'phase defference [degree] = ' ) )

a  = 1.0

n = 100
dt = 1.0 / n
p1 = p_diff / 360.0
m = n * n1 * n2

#  initialize lists

x = [ ]
y = [ ]

for i in range ( 0, m+1 ):
    x.append ( 0.0 )
    y.append ( 0.0 )

#  Calculate x and sin x

for i in range( 0, m+1 ):
    t = dt * i
#    print( i, t )
    x[ i ] = a * math.sin( 2.0 * math.pi * n1 * t )
    y[ i ] = a * math.sin( 2.0 * math.pi * n2 * ( t - p1 ) )

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

```
frequency n1 = 2
frequency n2 = 3
phase defference [degree] = 90
```

Out[17]: <function matplotlib.pyplot.show(*args, **kw)>

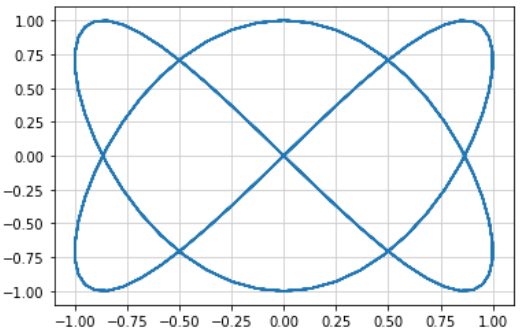

図 13.3　リサジュー図形をプロットするプログラムとその出力

例３． $y = \sin x$ の計算結果を保存

$\sin x$ を 1 周期計算し，ファイルに保存する。

```
In [*]: import math

        #  plot sine curve

        a = float( input( 'amplitude = ' ) )

        #  initialize lists

        x = [ ]
        y = [ ]

        n = 50
        dx = 1.0 / n

        for i in range ( 0, n+1 ):
            x.append ( 0.0 )
            y.append ( 0.0 )

        for i in range( 0, n+1 ):
            x[ i ] = dx * i
            y[ i ] = a * math.sin( 2.0 * math.pi * x[ i ] )

        file1 = open ( '/Users/nakakuki/Documents/python/sine1.csv', 'w' )
        for i in range ( 0, n+1 ):
            file1.write( str( x[ i ] ) + ' , ' + str( y[ i ] ) + '\n' )
        #    file1.writeline( str( x[ i ] ) + ' , ' + str( y[ i ] ) )
        file1.close( )

        file2 = open ( '/Users/nakakuki/Documents/python/sine2.dat', 'w' )
        for i in range ( 0, n+1 ):
            print( x[ i ], y[ i ], file = file2 )
        file2.close( )

        amplitude =
```

図 13.3　$y = \sin x$ を計算して，結果をファイルに保存するプログラム

・「/Users/nakakuki/Documents/python」は各自の PC に合わせて変更する。

・コンマ区切り(.csv)とスペース区切り(.dat)のファイル２つを作る。

例４． $y = \sin x$ の計算結果を読み込み・プロット

例３で保存したファイルを読み込んで画面にプロットするプログラム。

In [11]:
```
import matplotlib.pyplot as plt
import math
#
#  plot sine curve
#

#  initialize lists

x = [ ]
y = [ ]

n = 50

# reset list
for i in range ( 0, n+1 ):
    x.append ( 0.0 )
    y.append ( 0.0 )

# open file
f1 = open( 'sine2.csv', 'r' )

# read file

for i in range( 0, n+1 ):
    row = f1.readline( )
    a, b = row.split( ',' )

    x[ i ]  = float( a )
    y[ i ]  = float( b )

f1.close( )

for i in range ( 0, n+1 ):
    print( i, x[ i ], y[ i ] )

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

```
0 0.0 0.0
1 0.02 0.2506664671286085
2 0.04 0.4973797743297096
3 0.06 0.7362491053693558
4 0.08 0.9635073482034306
```

図 13.4　$y = \sin x$ をファイルから読み込んでプロットするプログラム

「/Users/nakakuki/Documents/python」は各自の PC に合わせて変更する。

例５． $y = \sin x$ の計算結果を読み込み・プロット

例４のプログラムの別の書き方。

In [4]:
```
import matplotlib.pyplot as plt
import math
#
#  plot sine curve
#

#  initialize lists

x = [ ]
y = [ ]

n = 50

# reset list
for i in range ( 0, n+1 ):
    x.append ( 0.0 )
    y.append ( 0.0 )

# open file
f1 = open( 'sine1.csv', 'r' )

#  read x and sin x
i = 0
for line in f1.readlines( ):
    a, b = line.split( ',' )
    x[ i ]  = float( a )
    y[ i ]  = float( b )

    i = i + 1

f1.close( )

for i in range ( 0, n+1 ):
    print( i, x[ i ], y[ i ] )

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

```
0 0.0 0.0
1 0.02 0.12533323356430426
2 0.04 0.2486898871648548
3 0.06 0.3681245526846779
4 0.08 0.4817536741017153
```

図 13.5　$y = \sin x$ をファイルから読み込んでプロットするプログラム

・open 関数のファイル名は各自の PC に合わせて変更する。

・ファイル型オブジェクトは for 文でループさせると，１行ずつ読み込む。

・「`.readlines( )`」は省略しても同じ動作になる。

例６． Monte-Carlo 法による円周率の計算

２つの乱数を発生して，円内に入る確率から円周率を推定する。

In [5]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import random as rnd
import math
"""
calculate pi by Monte-Carlo method
"""
# radius
r = 1.0
# number of particles
np = int( input( 'number of particles = ' ) )

# initialize lists
x     = [ 0.0 for i in range( np ) ]
y     = [ 0.0 for i in range( np ) ]
iflag = [ 0   for i in range( np ) ]

# plot points and count points in the inside of circle
cnt = 0
for i in range( np ):
    x[ i ] = rnd.random( )
    y[ i ] = rnd.random( )

    yc = math.sqrt( r**2 - x[ i ]**2 )
    if y[ i ] < yc:
        cnt = cnt + 1
        iflag[ i ] = 1
    else:
        iflag[ i ] = 0

# calculate possibility and pi
p  = cnt / np
pi = p * 4
print ( 'estimated pi value is ', pi )

# plot particles
for i in range( np ):
    if iflag[ i ] == 1:
        plt.scatter( x[ i ], y[ i ], marker = '.', c = 'r' )
    else:
        plt.scatter( x[ i ], y[ i ], marker = '.', c = 'b' )

plt.axis( 'equal' )
plt.show
```

図 13.6　Monte-Carlo 法により円周率を求めるプログラム

```
number of particles = 4000
estimated pi value is  3.167
```

Out[5]: <function matplotlib.pyplot.show(*args, **kw)>

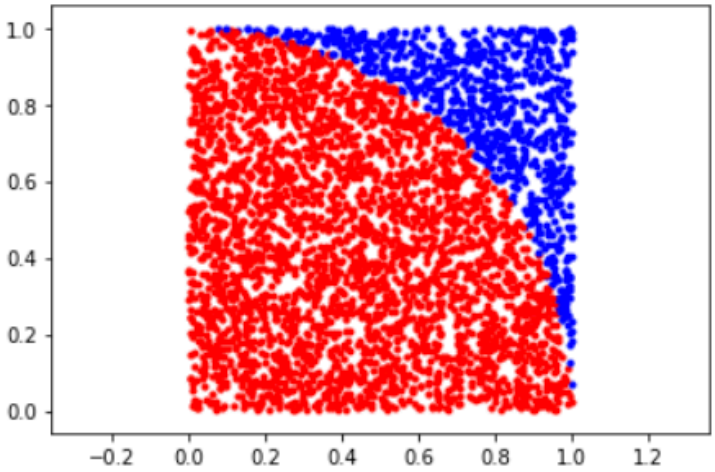

図 13.7　得られた円周率の値と 2 つの乱数を座標とするプロット：赤が円内

・「`inport random as rnd`」で乱数ライブラリを `rnd` としてインポート。
・「`rnd.random( )`」は 0 から 1 の一様乱数を発生する関数。
・「`yc = math.sqrt(… )`」で円の外周の y の値を計算
・「`if y < yc:`」で円の中か外かを判定している。整数型リスト「`iflag[ ]`」に記録。
・「`plt.scatter( )`」は散布図プロット。「`plt.`」はライブラリ `mathplotlib.pyplot` をインポートした略称

例７． 惑星の体積と平均密度

赤道半径，極半径，質量を入力して，惑星の体積と平均密度を計算する。

In [7]:
```
import math
"""
calculate volume and mean density of the planet
"""
# imput parameters: radius and mass
a = float( input( 'equator radius [km] = ' ) )
b = float( input( 'polar radius     [km] = ' ) )
m = float( input( 'mass             [kg]  = ' ) )

# Calculate volume
a = a * 1000
b = b * 1000
V = 4 / 3 * math.pi * a**2 * b

# Calculate mean density
rho = m / V

# Volume in kg^3
V = V / 1.0e9

# Output calculate values
print( 'volume            = ', V, ' [km^3]' )
print( 'mean density     = ', rho, ' [kg m^-3]' )
```

```
equator radius [km] = 6378
polar radius    [km] = 6357
mass            [kg]  = 5.972e24
volume            =  1083202991015.2313  [km^3]
mean density     =  5513.278720180367  [kg m^-3]
```

図 13.8　惑星の体積と平均密度を求めるプログラム

A1. テキストエディタ

メモ帳は最小限の機能しかなく, 使いにくいので別のテキストエディタを使用すると良いかもしれない。下記は使用してみたもの。

A1.1 TeraPad

・Windows 用のシンプルなテキストエディタ。
・軽くて使いやすい。
・ダウンロード元は，https://tera-net.com/library/tpad.html

A1.2 Visual Studio Code

・Microsoft が開発・配布しているテキストエディタ。
・Windows, macOS, Linux に対応。
・プログラム実行や様々な言語に対応したプラグインなど機能多彩。
・日本語 OS でもバックスラッシュが「\」で表示される。
・ダウンロード元は，https://code.visualstudio.com/download

A1.3 Jedit Ω

・macOS 用のテキストエディタ。
・有料版と無料版がある。
・軽くて機能も豊富で使いやすい。
・ダウンロード元は，　http://www.artman21.com/jp/jeditOmega/

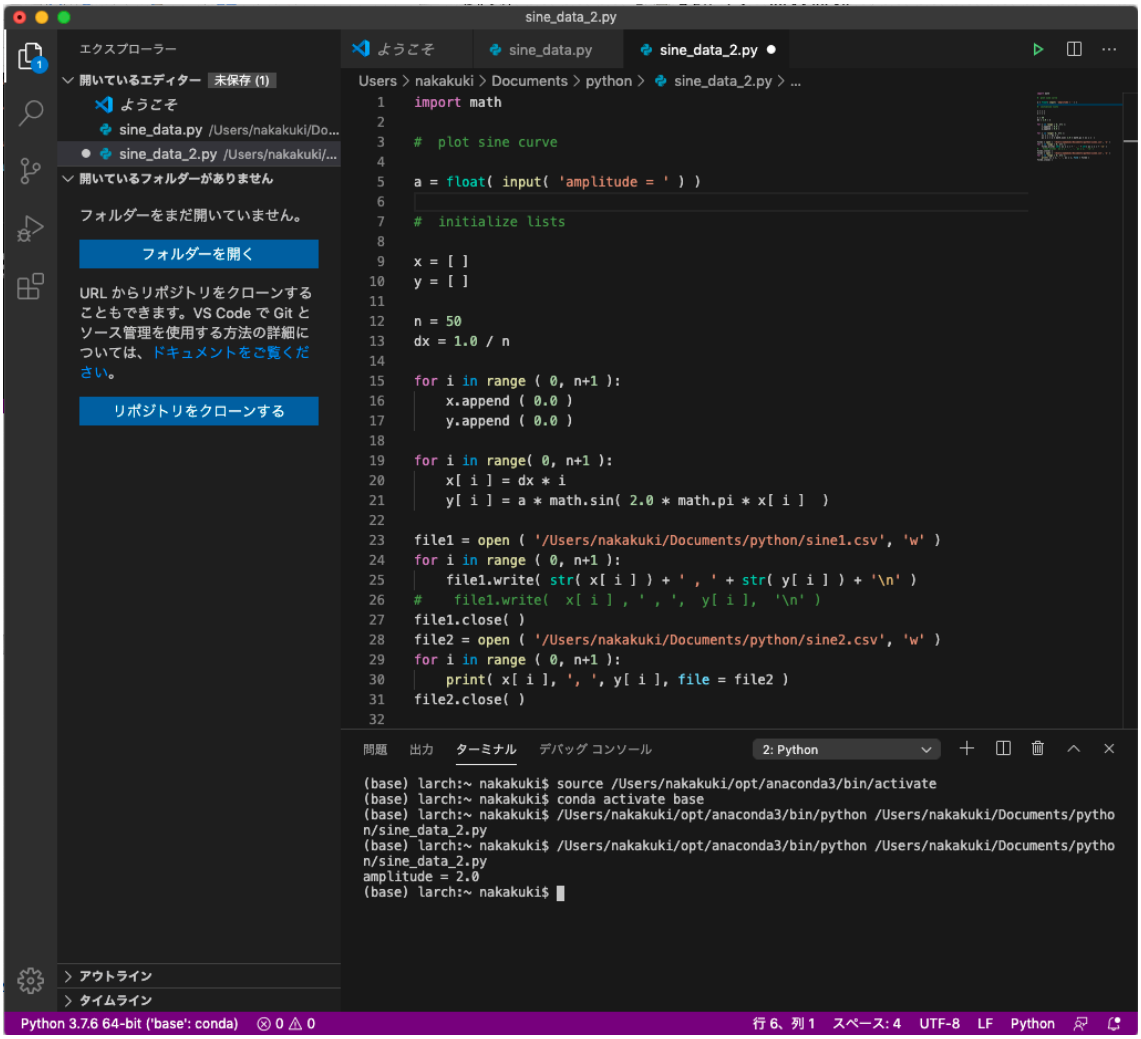

図 A1.1 Visual Code Studio で $y = \sin x$ を計算して２通りの方法でファイルに書き込むプログラムを開発・テスト実行