4.10 オブジェクトの保存と読込み： pickle モジュール

Python 処理系に標準的に添付されている pickle モジュールを利用すると，オブジェクト（データ構造）を**バイト列**[200] （バイナリ形式のデータ列）に変換（**シリアライズ**）する，あるいはその逆の変換を行うことができる．このモジュールが提供する代表的な関数を表 38 に挙げる．

表 38: pickle モジュールが提供する代表的な関数

関数	説明
dumps(**オブジェクト**)	「オブジェクト」をバイト列に変換したものを返す．
dump(**オブジェクト**, **ファイル**)	「オブジェクト」をバイト列に変換して 「ファイル」に保存する．
loads(**バイト列**)	「バイト列」を展開して元のオブジェクトにして返す．
load(**ファイル**)	dump 関数により保存されたファイルを読込み， 元のオブジェクトにして返す．

例. リスト⇔バイト列の変換

```
>>> import pickle  Enter     ←モジュールの読込み
>>> d = [1,2,3,4,5,6,7,8,9,10]  Enter     ←リストの用意
>>> bn = pickle.dumps(d)  Enter     ←バイト列に変換
>>> bn  Enter     ←内容確認
b'¥x80¥x03]q¥x00(K¥x01K¥x02K¥x03K¥x04K¥x05K¥x06K¥x07K¥x08K¥tK¥ne.' ←バイト列になっている
>>> pickle.loads(bn)  Enter     ←リストに戻す
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]     ←元のリストになっている
```

dumps, dump 関数によって得られたバイト列はデータ型に関する情報などを含んでおり，元のデータに復元することができる．

次に，定義されたクラスのインスタンスとして生成されたオブジェクトをファイルに保存し，それを再び読み込んで元のオブジェクトに復元する例を示す．次に示すサンプルプログラム pickle02class.py は三角関数の表を生成してプロットするためのクラス Data を定義している．（グラフの描画には matplotlib ライブラリ[201] を使用する）

プログラム： pickle02class.py

```
1  # coding: utf-8
2  import math                          # 数学関数のモジュール
3  import matplotlib.pyplot as plt      # グラフ描画ライブラリ
4
5  #--- クラス定義 ---
6  class Data: # 三角関数の表を扱うクラス
7      #--- コンストラクタ ---
8      def __init__(self):
9          self.tri = dict()   # 辞書として空の表を生成
10     #--- データの生成 ---
11     def gen(self):
12         for ix in range(629):
13             x = ix / 100.0      # 定義域
14             ysin = math.sin(x); ycos = math.cos(x)      # 正弦関数，余弦関数
15             self.tri[x] = [round(ysin,3),round(ycos,3)] # を生成してリストにする
16     #--- データ列の取り出し ---
17     def qx(self):
18         return list( self.tri.keys() )
19     def qsin(self):
20         return [self.tri[x][0] for x in self.qx()]
21     def qcos(self):
22         return [self.tri[x][1] for x in self.qx()]
23     #--- グラフのプロット ---
24     def plotTriangle(self):
25         plt.figure( figsize=(6,2.8) )
```

[200]「2.7.3.4 バイト列の扱い」（p.105）参照のこと．

[201]Python でグラフを描画する場合によく用いられるライブラリ（公式インターネットサイト：https://matplotlib.org/）
拙書「Python3 ライブラリブック」でも基本的な使用方法を解説しています．

```
26          plt.plot( self.qx(), self.qsin() );      plt.plot( self.qx(), self.qcos() )
27          plt.show()
```

Data クラスは三角関数表を辞書オブジェクト tri として保持するもので，インスタンス生成後に gen メソッドによりデータを生成する．また，plotTriangle メソッドによりグラフを描画する．このクラスを用いて，

1) データの生成，グラフの描画，データの保存
2) 保存されたデータの復元（Data クラス），グラフの描画

の処理を行う例（サンプルプログラム pickle02-1.py，pickle02-2.py）を示す．

プログラム：pickle02-1.py（データの生成，グラフ描画，データの保存）

```
1  # coding: utf-8
2  import pickle
3
4  #--- クラス定義読込み ---
5  from pickle02class import Data
6
7  #--- データの生成とプロット ---
8  d = Data()           # オブジェクトの生成
9  d.gen()              # データの生成
10 d.plotTriangle()     # グラフのプロット
11
12 #--- pickleによるデータ保存 ---
13 f = open('pickle02.dat','wb')   # バイナリファイルを作成
14 pickle.dump( d, f )             # 保存
15 f.close()
```

プログラム：pickle02-2.py（データの読込み，グラフの描画）

```
1  # coding: utf-8
2  import pickle
3
4  #--- pickleによるデータ読込み ---
5  f = open('pickle02.dat','rb')   # バイナリファイルとして開く
6  d = pickle.load( f )            # 読込み
7  f.close()
8
9  #--- グラフのプロット ---
10 d.plotTriangle()
```

これらプログラムを実行すると，Data クラスのオブジェクトがバイナリファイル pickle02.dat として保存される．特に重要な点として，読込み側のプログラム pickle02-2.py においてクラスの定義の記述が無いことが挙げられる．pickle モジュールの機能によって保存された pickle02.dat にはデータのクラス定義の関連情報が含まれており[注]，グラフ描画のための plotTriangle メソッドが実行できることが確認される．

両方のプログラムの実行において図 36 のようなグラフが表示される．

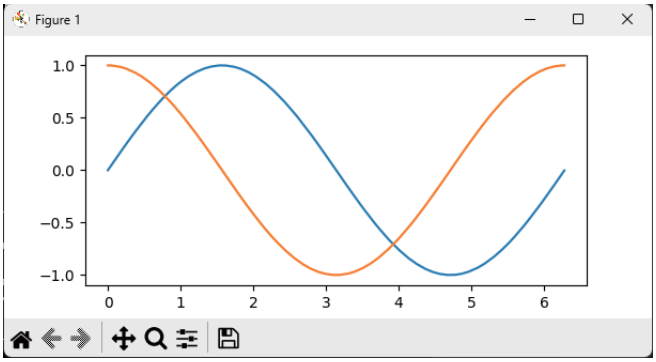

図 36: pickle02-1.py，pickle02-2.py で描画されるグラフ

注意）

先の例では，クラス Data の定義本体は pickle02.dat には含まれておらず，pickle02class.py の内容を参照している．具体的には，保存対象のオブジェクトをシリアライズしたバイト列データの中に，そのオブジェクトのクラス名や，そのクラスがどのモジュール（ファイル）に保存されているかの関連情報が保持されている．従って，Python 言語処理系が必要なモジュールファイルを見つけることができない場合はエラー（ModuleNotFoundError）が起こる．例えば先の例において，pickle02class.py がモジュールとして参照できない状況にあるとこの問題が起こる．

4.10.1 シリアライズと復元のカスタマイズ方法

Python のオブジェクトを dump，dumps でシリアライズし，load，loads で復元する処理において，通常はその処理過程をユーザが意識することはない．しかし，Python 3.10 以前の版では，スロットベースのクラス[202] のインスタンスをこの方法で処理することができず．シリアライズと復元の過程をユーザがカスタマイズする必要があった[203]．また現在の版の Python においても，特殊な形式のオブジェクトを pickle で扱う場合は，処理過程をユーザが独自にカスタマイズしなければならないことがある．

次に示すサンプル pickle03class.py では，スロットベースのクラス Pos を定義している．このクラスは，2 次元座標上の点を x，y の属性で現し，その原点 (0,0) からの距離を属性 _distance で表すものである．またこのクラスは，そのインスタンスをシリアライズと復元する際の処理のカスタマイズをしている．具体的には，クラス Pos に特殊メソッド__getstate__，__setstate__，__reduce__ を実装してカスタマイズしている．

プログラム：pickle03class.py（クラス定義）

```
class Pos:
    __slots__ = ('x', 'y', '_distance')     # スロットとして属性を限定
    # コンストラクタ
    def __init__(self, px, py):
        print('__init__: コンストラクタの実行')
        self.x = px
        self.y = py
        self._distance = (px**2 + py**2)**0.5
    # シリアライズの際の状態取得：すべての属性情報を辞書にして返す
    def __getstate__(self):
        state = {'x': self.x, 'y': self.y, '_distance': self._distance}
        print('__getstate__: 保存のための状態取得 :',state)
        return state
    # 復元時のインスタンスの再構成：状態辞書から属性を再構成
    def __setstate__(self, state):
        print('__setstate__: 復元に用いる状態辞書 :',state)
        self.x = state['x']
        self.y = state['y']
        self._distance = state['_distance']
    # シリアライズの際に埋め込む情報：コンストラクタとその引数，状態辞書
    def __reduce__(self):
        print('__reduce__が起動')
        return (self.__class__, (self.x, self.y), self.__getstate__())
    # 状態表示
    def show(self):
        print( f'  show: {self.x=}, {self.y=}, distance={self._distance}' )
```

このクラス定義のカスタマイズの要点を以下に説明する．

■ シリアライズの対象となるオブジェクトの属性情報の収集：__getstate__ メソッド

シリアライズの対象となるオブジェクトの属性とその値の組を辞書の形で返す処理として実装する．

■ 復元処理：__setstate__ メソッド

属性とその値の組の辞書から実際にインスタンスの状態を構成する処理を実装する．

■ シリアライズに必要な機能とデータの収集：__reduce__ メソッド

__reduce__ メソッドは，オブジェクトを実際にシリアライズする際に呼び出されるもので，復元時にオブジェク

[202]巻末付録「I スロットベースのクラス」(p.451）で解説する．

[203]Python 3.11 からはその必要がなくなった．

トを再構成するための関数（通常は当該クラスのコンストラクタ）と，属性情報辞書（通常は__getstate__で生成）を収集する．

__reduce__ メソッドは，再構成用の関数，それに渡すための引数並びのタプル，属性情報辞書をタプルにして返す形にする．

pickle03class.py に記述されている Pos クラスでは，上記の特殊メソッド __getstate__，__setstate__，__reduce__ やコンストラクタが呼び出されたことを print 関数で知らせる出力をしている．Pos クラスのインスタンスをシリアライズする処理，復元する処理を次の pickle03.py で実行する．

プログラム：pickle03.py（シリアライズと復元）

```
1  import pickle
2  from pickle03class import Pos
3
4  print('■ インスタンス生成')
5  p1 = Pos(3,4)    # インスタンス生成
6  p1.show()        # 状態確認
7
8  f = open('pickle03.dat','wb')
9  print('■ シリアライズして保存します')
10 pickle.dump(p1,f)
11 f.close()
12
13 f = open('pickle03.dat','rb')
14 print('■ 復元します')
15 p2 = pickle.load(f)
16 f.close()
17
18 p2.show()
```

このプログラムを実行した際の出力の例を次に示す．

実行例．ターミナルのコマンドラインで実行（Python 3.13）

```
■ インスタンス生成
__init__: コンストラクタの実行
   show: self.x=3, self.y=4, distance=5.0
■ シリアライズして保存します
__reduce__が起動
__getstate__: 保存のための状態取得：{'x': 3, 'y': 4, '_distance': 5.0}
■ 復元します
__init__: コンストラクタの実行
__setstate__: 復元に用いる状態辞書：{'x': 3, 'y': 4, '_distance': 5.0}
   show: self.x=3, self.y=4, distance=5.0
```

処理の流れ，各メソッドが実行されたタイミングが確認できる．

4.10.2　シリアライズされたバイナリデータのプロトコルレベル

pickle の機能は Python の版の進展に従ってアップデートされてきており，オブジェクトのシリアライズの手法もアップデートされてきている．過去の pickle によって作成されたバイナリデータには，作成当時の手法（**pickle のプロトコル**：表 39）の情報が含まれており，新しい版の pickle はその情報（**プロトコルレベル**）を読み取って，復元処理を行う．

表 39: pickle のプロトコルレベル

プロトコルレベル	対応する Python の版	主な特徴
0	全ての版	ASCII 形式（テキスト）
1	全ての版	最初のバイナリ形式（後方互換性のために存在）
2	Python 2.3 以降	新スタイルクラスなどの効率化
3	Python 3.0 以降	bytes 型対応など（これ以降 Python 2 には非対応）
4	Python 3.4 以降	大容量への対応と高速化
5	Python 3.8 以降	アウト-オブ-バンドバッファなどの新機能

シリアライズ処理の際，デフォルトでは処理系における最新のプロトコルレベルが採用される（例外もある）が，詳しくは次に説明する方法で調べると良い．

4.10.2.1　シリアライズされたバイナリデータのプロトコルレベルを調べる方法

標準ライブラリである pickletools が提供する genops 関数を用いることで，シリアライズされたバイナリデータを解析することができる．この関数はシリアライズされたバイナリデータのファイルから次々と構成要素を読み取るもので，「オペレーションコード，引数，要素の位置」を組みを返すイテレータとなる．

「オペレーションコード」の name 属性が 'PROTO' である場合の「引数」が，当該ファイルのプロトコルレベルである．また，古いプロトコルレベルのファイルはこの情報を持っていないこともある．このことを応用して，バイナリデータファイルのプロトコルレベルを調べる関数を実装したモジュールファイルを pickleProtocol.py に示す．

プログラム：pickleProtocol.py

```
1 import pickletools
2
3 def pickleProtocol(path):
4     with open(path, 'rb') as f:
5         for op, arg, _ in pickletools.genops(f):
6             if op.name == 'PROTO':
7                 return arg
8     return None     # プロトコル0/1（PROTOがない場合）はNone
```

このモジュールを用いて，先の pickle03.py が作ったバイナリデータファイル pickle03.dat を解析する例を次に示す．

例. pickle03.dat のプロトコルレベルを調べる

```
>>> from pickleProtocol import pickleProtocol  Enter    ←上記モジュールの読込み
>>> pickleProtocol('pickle03.dat')  Enter    ←ファイル pickle03.dat のプロトコルレベルを調べる
4                                              ←結果
```

また，OS のコマンドシェルでも pickletools を実行して解析処理が行える．（次の例）

例. コマンド操作でバイナリファイルの情報を解析（Windows 環境の PSF 版 Python の場合）

```
C:\Users\katsu\Python> py -m pickletools pickle03.dat  Enter    ←解析開始
    0:  \x80 PROTO      4                  ←プロトコルレベルの情報
    2:  \x95 FRAME      71
   11:  \x8c SHORT_BINUNICODE 'pickle03class'
                  :
              （以下省略）
                  .
```

4.10.2.2　シリアライズ時のプロトコルレベルの指定

dump，dumps に引数 'protocol=' を与えることで，シリアライズのプロトコルレベルを指定することができる．

例. プロトコルレベルを指定したシリアライズ

```
>>> import pickle  Enter    ←モジュールの読込み
>>> s0 = pickle.dumps( list(range(100000)), protocol=0 )  Enter  ←プロトコルレベル 0 を指定
>>> s1 = pickle.dumps( list(range(100000)), protocol=1 )  Enter  ←プロトコルレベル 1 を指定
>>> s2 = pickle.dumps( list(range(100000)), protocol=2 )  Enter  ←プロトコルレベル 2 を指定
>>> s3 = pickle.dumps( list(range(100000)), protocol=3 )  Enter  ←プロトコルレベル 3 を指定
>>> s4 = pickle.dumps( list(range(100000)), protocol=4 )  Enter  ←プロトコルレベル 4 を指定
>>> s5 = pickle.dumps( list(range(100000)), protocol=5 )  Enter  ←プロトコルレベル 5 を指定
```

このようにしてシリアライズされたバイナリデータ s0〜5 のデータサイズを調べる．（次の例）

例. データサイズを調べる（先の例の続き）

```
>>> len(s0)  Enter
788896
>>> len(s1)  Enter
368876
>>> len(s2)  Enter
368878
```

```
>>> len(s3)  Enter
368878
>>> len(s4)  Enter
368931
>>> len(s5)  Enter
368931
```

このように，プロトコルレベルに応じてシリアライズされたデータのサイズが異なることがあることがわかる．

dump の引数に 'protocol=' を与えてバイナリファイルを作成する例を pickle04-1.py に示す．これは 10,000 個の素数のリストを gmpy2 モジュール[204] を用いて作成し，それを各プロトコルレベルでシリアライズして別々のファイルとして保存するものである．

プログラム：pickle04-1.py

```
import pickle
import gmpy2

# 素数10,000個のリストを作成
n = 0
pLst = []
for _ in range(10000):
    n = gmpy2.next_prime(n)
    pLst.append( int(n) )

# プロトコルレベル毎にシリアライズして保存
for p in range(6):
    fname = 'pickle04_'+str(p)+'.dat'
    f = open(fname,'wb')
    pickle.dump(pLst,f,protocol=p)
    f.close()
```

このプログラムで作成したバイナリデータのファイルのプロトコルレベルとファイルサイズを調べるプログラムを pickle04-2.py に示す．

プログラム：pickle04-2.py

```
import pickletools
import os

# ファイルのプロトコルレベルを調べる関数
def pickleProtocol(path):
    with open(path, 'rb') as f:
        for op, arg, _ in pickletools.genops(f):
            if op.name == 'PROTO':
                return arg
    return None     # プロトコル0/1（PROTOがない場合）はNone

# ファイル毎に調査
print('file-name\tp-lev.\tfile-size\n--------------------------------')
for p in range(6):
    fname = 'pickle04_'+str(p)+'.dat'
    f = open(fname,'rb')
    pl = pickleProtocol(fname)
    f.close()
    fs = os.path.getsize(fname)
    print( f'{fname}\t{pl}\t{fs}' )
```

このプログラムを実行した例を次に示す．

実行例.

```
file-name       p-lev.   file-size
--------------------------------
pickle04_0.dat  None     78988
pickle04_1.dat  None     36886
pickle04_2.dat  2        36888
pickle04_3.dat  3        36888
pickle04_4.dat  4        36896
pickle04_5.dat  5        36896
```

[204]標準モジュールではないので，pip や conda などを用いてインストールする必要がある．(https://pypi.org/project/gmpy2/)

4.10.3 使用上の注意事項

先の pickle03class.py の例からわかるように，pickle でシリアライズされたバイナリデータを復元する際に，復元対象のオブジェクトのクラスのコンストラクタ（__init__）や __reduce__，__setstate__ といった特殊メソッドの中に記述されたコードが自動的に実行される．従って，クラス定義の中に悪意のあるコードが記述されていれば，データの復元時にそれらが実行されてしまう．従って，pickle の利用はセキュリティ上のリスクとなり得るので注意が必要である．

pickle は，Python 上で作成したオブジェクトをバイナリデータに変換するための簡便な方法であるが，素性のわからない第三者が pickle で作成したバイナリデータは，セキュリティ上の観点から復元するべきではない．従って pickle の利用は，Python 利用者が個人の利用の範囲に限る，あるいは，信頼できる緊密な開発者の間での利用に限るべきである．

上に述べたセキュリティ上のリスクは，ユーザが実装したクラス定義の内容から来るものであるが，標準ライブラリとして利用できる機能を応用して作られたバイナリデータは更に別の形でのセキュリティリスクとなり得る．例えば次に示す pickleBgen.py で作成されるバイナリファイル pickleB.dat を復元する場合，クラス Bomb の定義ファイル pickleBgen.py が無くても，作成者が仕組んだコードが実行される．

プログラム：pickleBgen.py

```
1  import pickle, os
2
3  class Bomb:
4      def __reduce__(self):
5          # 復元時にこの関数と引数が呼ばれる
6          return ( os.system, ('echo [爆弾作動！]',) )
7
8  # シリアライズして保存
9  if __name__ == '__main__':
10     f = open('pickleB.dat','wb')
11     bad_pickle = pickle.dump(Bomb(),f)
12     f.close()
```

このプログラムを実行した結果として出来上がったバイナリファイル pickleB.dat を読み込んで復元する例を次に示す．

例. バイナリデータを復元するだけで実行されるコード

```
>>> import pickle  Enter      ←モジュールの読込み
>>> f = open('pickleB.dat','rb')  Enter      ←ファイルを開いて
>>> m = pickle.load(f)  Enter      ←復元処理を行うだけで
[爆弾作動！]          ← os.system('echo [爆弾作動！]') が実行される
>>> f.close()  Enter
```

os モジュールは Python の標準ライブラリであり，OS に関する様々な処理を実行する機能を提供する．上の実行例では，メッセージを表示するだけにとどまっているが，悪意を持ってこれを応用すると，システムに対する大きな脅威となる．