B1301

1. namedtuple

- 名前付きタプルと呼ばれるコレクション型です。
- 使用する前に`from collections import namedtuple`と宣言する必要があります。
- イミュータブルなオブジェクトのように振る舞います。（書き換え不能な構造体）
- オブジェクトより空間・時間方向で効率が良いです。
- hogehoge['item'] という呼び出しではなく hogehoge.item と呼び出せるようになります。
- 辞書のキーとして使えます。

書式

```
from collections import namedtuple
コンテナ名=namedtuple('コンテナ名', '属性1_key 属性2_key ... 属性n_key')
コンテナ名.属性i_key で呼び出せる。
```

name・number・sex 属性を持つStudentコンテナを namedtuple で作成すると次のようなコードになります。

```
from collections import namedtuple
Student=namedtuple('Student','name number sex')
taro=Student('Taro Chodai',12345678,'male')
hanako=Student('Hanako Nagasaki',12345679,'female')
print(taro.name)
print(taro.number)
print(taro.sex)
print(hanako.name)
print(hanako.number)
print(hanako.sex)
```

2. クラスを使った実装

クラスはオブジェクト（データ）の型を自作することができます。オブジェクトは属性（変数）を集めただけではなく、そのオブジェクトを操作するための処理（メソッド関数）も一緒に定義します。

- クラスで作成したオブジェクトはミュータブルです（書き換え可能です）。
- ライブラリをimportしなくても定義可能です。
- 複雑なことができますので、空間・時間的には効率が悪いです。
- インスタンス変数やインスタンスメソッドの呼び出し方は インスタンス名.属性 です。クラス変数やクラスメソッドの呼び出し型はクラス名.属性です。クラスの定義内ではインスタンス名は self です。
- 他の組み込み型もオブジェクトです。
- 使いこなすにはオブジェクト指向プログラミングの概念を理解する必要がありますが、抽象的なのでわかりにくいという評価が多いです。一方手続き型プログラミング（今まで学んだ範囲）は直感的でわかりやすいそうです。

クラスを使った実装を先のStudentコンテナのケースで説明します。

```
```

```
class Student: # class Student(Object): と等価
    def __init__(self,name,number,sex):
        self.name=name
        self.number=number
        self.sex=sex

taro=Student('Taro Chodai',12345678,'male')
hanako=Student('Hanako Nagasaki',12345679,'female')
print(taro.name)
print(taro.number)
print(taro.sex)
print(hanako.name)
print(hanako.number)
print(hanako.sex)
```

この定義では、初期設定用のメソッド `__init__(self,name,number,sex)` でインスタンス変数 `self` の属性に引数で与えられた値を代入しています。インスタンス変数 `self` はオブジェクトのインスタンス（実際にメモリに展開されたもの）です。`taro` が指し示しているオブジェクトと `hanako` を指し示す変数として `self` が使われています。

一度作成したクラスは、更に自分用にカスタマイズできます。

```
class Joho_Student(Student):
    base_number=12345500 #クラス変数
    def __init__(self,name,sex,shozoku_course):
        Joho_Student.base_number+=1 #クラス変数を参照するときはクラス名.クラス変数
        super().__init__(name,Joho_Student.base_number,sex) #(1)
        self.shozoku_course=shozoku_course
    def aisatsu(self):
        print('Hello, I am '+self.name+'in '+self.shozoku_course+' course. My
number is'+str(self.number)) #個人で処理が異なる場合はインスタンスメソッドで処理する。
    def aisatsu2():
        print('Hello, We are Joho Students!') #クラス全員に共通する処理はクラスメソッ
ドという。

Jiro=Joho_Student('Jiro Joho','male','IS')
Saiko=Joho_Student('Saiko Deta','female','DS')
Joho_Student.aisatsu2() # クラスメソッドはクラス名.クラスメソッド()で呼び出す。
Jiro.aisatsu()
Saiko.aisatsu()
```

クラス変数を用いるとクラスに共通する変数を管理することができます。主な用途は通し番号の自動付与です。クラス変数は明示的に宣言しますがインスタンス特有の属性は self.属性名に値を代入して定義します。呼び出し方もクラス変数やクラスメソッドはクラス名を指定しますが、インスタンス変数やインスタンスメソッドはインスタンス名を指定します。super()は継承元のクラスを指定します。#(1) の行は継承元の初期化ルーチンを再利用しています。

2.1 クラスの使いどころ

- 動作は同じ、属性は異なる複数のインスタンスを必要とするときは、クラスが最も役にたちます。
- クラスは継承をサポートしますが、モジュールはサポートしません。
- なにか１つ「だけ」を必要として複数ないことを保証したいときは、モジュールが良いです。詳しくはシングルトンを調べて下さい。

- 複数の値を持つ変数があり、これらを複数の関数に引数として渡せるときには、それをクラスとして定義した方がすっきりすることがあります。
- 問題にとってもっとも単純な方法を使ってください。データ構造を作り込みすぎないようにしましょう。オブジェクトよりもタプルの方が良いことがあります。組み込みデータ型、数値、文字列、タプル、リスト、集合、辞書をもっと使いましょう。これらは既にオブジェクト化されています。よく探せば欲しい機能が備わっているかもしれません。そしてコレクションライブラリにも注意を払いましょう。
- ゲッター・セッター関数よりも単純な方法でアクセスしましょう。アクセスを制御したい場合はPythonでは守りきれません。詳しくはマングリングを調べて下さい。マングリングは親クラスと子クラスの名前衝突回避のための記法ですから、Javaのアクセス制御ほど強固ではありません。

2.2 クラスの特徴

添付した資料1.pdfも参照してください。色んな言葉が飛び交っていますが、自主学習した人のためのキーワードとして利用してください。詳しくはISコースのプログラミング言語論や他のプログラミング科目でいずれ学びます。

問(i) namedtuple を使って、属性として所有者氏名name、小銭入れのコイン枚数 c1, c5, c10, c50, c100, c500 を持ったWalletコンテナを作成してください。Takada財布コンテナのインスタンス変数の値は、name='Hiroyuki Takada', お金は小さい順から、4,1,3,1,2,1 枚ずつとしてください（この場合の合計金額は789円です)。また、namedtuple型コンテナを引数にとり、財布の合計金額を返す関数 sum を実装してください。動作させるプログラムも作成してください。なお、プログラムはライブラリ化させなくても構いません。

問(ii) 問(i) のデータを cWallet型としてclass定義してください。財布の合計金額を求める `sum()` メソッドもcWalletに実装して下さい。クラスを定義したら、問(i)と同様にしてTakada財布とあなたの財布のインスタンスを作成して実装した関数を動作させてください。各財布の合計額を表示するようにしてください。

提出物は作成したプログラム b1301_bbxxxxxxxx.py と出力の結果b1301.txtです。