【課題 11：count_pairs.py】

collections モジュールの Counter クラスを利用して，課題 02 と同様に，配布した「poke_names.txt」を読み込み，3 列目に書かれたタイプのペアをカウントし，出現回数が多いペアから順に出力するコードを実装せよ．当該ファイルは，各行において番号，名前，タイプ，進化先がタブで区切られており，3 列目のタイプは半角スペースで複数のタイプが区切られている．この課題では，3 列目のタイプが 2 つある行が対象となる．

採点の都合上，以下の要件を満たすこととする：

- collections.Counter を用いる；
- 読み込みファイルのパスは，コマンドライン引数により指定する；
- 不要なモジュールは import しない；
- デバッグ用の print など必要ないものはコメントアウトし，出力が図 1 になるようにする；

```
PS C:\Users\admin\Desktop\ProA2> python .\lec06\kadai11\count_pairs.py lec06/data/poke_names.txt
[(frozenset({'ひこう', 'ノーマル'}), 10),
 (frozenset({'どく', 'くさ'}), 9),
 (frozenset({'じめん', 'いわ'}), 8),
 (frozenset({'むし', 'どく'}), 7),
 (frozenset({'ひこう', 'むし'}), 5),
 (frozenset({'みず', 'いわ'}), 5),
 (frozenset({'エスパー', 'みず'}), 4),
 (frozenset({'ひこう', 'ほのお'}), 3),
 (frozenset({'フェアリー', 'ノーマル'}), 3),
 (frozenset({'ひこう', 'どく'}), 3)]
```

図 1　課題 11 の実行例

リスト 1　count_pairs.py

```
1  import sys
2  from collections import Counter
3  from pprint import pprint
4
5  def read_types(file_path):
6      types = list()
7      with open(file_path, "r", encoding="utf8") as rfo:
8          for row in rfo:
9              _, _, typ, *_ = row.rstrip().split("\t")
10             types.append(frozenset(typ.split()))
11     return types
12
13
14 if __name__ == "__main__":
15     types = read_types(sys.argv[1])
16     counter = Counter([type_set for type_set in types if len(type_set) == 2])
17     pprint(counter.most_common(10))
```

【課題 12：evolution.py】

list クラスを継承し，進化先リストを表す EvolvList クラスを定義せよ．list クラスを継承しているので，リストとしての性質，機能を持つ．例えば，append メソッドにより当該クラスのインスタンスに要素を追加できる．

【「if __name__ == "__main__"」以下の説明】

(1) フシギダネの進化先リスト tane をインスタンス化し，tane に進化先のフシギソウを表す kusa を append メソッドにより追加している；
(2) フシギダネはフシギソウにしか進化できないため，リスト tane には 1 つの要素のみ append している；
(3) この例は実際と異なり，フシギバナはさらにフシギスギに進化するが，フシギスギはこれ以上進化しないため，フシギバナを表す hana には，名前文字列"フシギスギ"を append している；
(4) 当該リストクラスの evolution() メソッドにより，図 2 のように，進化先の名前を進化の段階に合わせてインデントして print している；

上記のように直接進化先を要素に格納する特殊なリストを表す EvolveList クラスを実装せよ．採点の都合上，以下の要件を満たすこととする：

(1) list クラスを継承する；
(2) このクラスのインスタンスは，直接進化先（フシギダネの直接進化先はフシギソウのみ）を要素として持つリストである；
(3) ただし，直接進化先がさらに進化先を持っている場合は，進化先ポケモンの EvolvList インスタンスを，さらなる進化先がない場合は，進化先ポケモンの名前文字列を要素とするリストである；
(4) イニシャライザでは，インスタンス変数 title に名前を設定する；
(5) evolution() メソッドを定義する；
　（a）進化の段階（例：フシギダネは 0，フシギソウは 1，フシギバナは 2）を表すパラメータ dankai を持つ；
　（b）dankai のデフォルト値を 0 とする；
　（c）dankai の値に合わせたインデント（全角スペース）を定義する；
　（d）定義したインデントと名前を print する；
　（e）当該リストの要素（EvolvList インスタンス or 文字列）に対して，
　　i. EvolvList インスタンスなら（さらに進化するなら），その evolution() メソッドを呼び出す；
　　ii. 文字列なら（それ以上進化しないなら），その名前文字列を print する；
(6) evolution.py にあらかじめ書かれている部分は，変更しない；
(7) 不要なモジュールは import しない；
(8) デバッグ用の print など必要ないものはコメントアウトし，出力が図 2 になるようにする；

```
フシギダネ/
  フシギソウ/
    フシギバナ/
      フシギスギ
[[['フシギスギ']]]
ナゾノクサ/
  クサイハナ/
    ラフレシア
    キレイハナ
[['ラフレシア', 'キレイハナ']]
イーブイ/
  シャワーズ
  サンダース
  ブースター
  エーフィ
  ブラッキー
  リーフィア
  グレイシア
  ニンフィア
['シャワーズ', 'サンダース', 'ブースター', 'エーフィ', 'ブラッキー', 'リーフィア', 'グレイシア', 'ニンフィア']
```

図 2　課題 12 の実行例

リスト 2　evolution.py

```
class EvolvList(list):
    def __init__(self, title):
        self.title = title

    def evolution(self, dankai=0):
        offset = "  "*dankai
        print(f"{offset}{self.title}/")

        for item in self:
            if hasattr(item, "evolution"):
                item.evolution(dankai+1)
            else:
                print(f"{offset}  {item}")


if __name__ == "__main__":
    tane = EvolvList("フシギダネ")
    kusa = EvolvList("フシギソウ")
    hana = EvolvList("フシギバナ")
    tane.append(kusa)
    kusa.append(hana)
    hana.append("フシギスギ")
    tane.evolution()
    print(tane)

    nazo = EvolvList("ナゾノクサ")
    kusai= EvolvList("クサイハナ")
    nazo.append(kusai)
    kusai.append("ラフレシア")
    kusai.append("キレイハナ")
    nazo.evolution()
    print(nazo)

    eevee = EvolvList("イーブイ")
    eevee.append("シャワーズ")
    eevee.append("サンダース")
    eevee.append("ブースター")
    eevee.append("エーフィ")
    eevee.append("ブラッキー")
    eevee.append("リーフィア")
    eevee.append("グレイシア")
    eevee.append("ニンフィア")
    eevee.evolution()
    print(eevee)
```

【課題 13：unique_type.py】

dict クラスを継承し，異なるタイプからなるポケモンのチームを表す Team クラスを定義せよ．Team クラスは，キーをポケモンの名前，値をタイプ（set で表す）とした dict である．通常の dict は，キーは一意である必要があるが，値が重複しても問題ない．一方，今回実装する Team クラスは，値も一意である必要がある．もし，すでに登録されている値を再度登録しようとした場合，ValueError が発生するように実装する．これにより，一意なタイプパターンからなるチームを作ることができる．なお，単独の「set(”どく”)」タイプと「set(”どく”, ”くさ”)」タイプは異なるタイプパターンとする．

上記のように，キーだけでなく，値にも一意性の制約を持つ特殊な辞書を表す Team クラスを実装せよ．採点の都合上，以下の要件を満たすこととする：

(1) dict クラスを継承する；
(2)「辞書 [キー] = 値」の形式で辞書に値を設定できるように，特殊メソッドを定義する；
(3) この特殊メソッドでは，以下の条件のとき，ValueError を raise する：
　(a)「値」がすでに登録されており，「キー」が既に登録されている，かつ，「辞書 [キー] != 値」の場合（※「辞書 [キー] == 値」なら，同じエントリーを再度登録しようとしているだけなので，エラーではない）；
　(b)「値」がすでに登録されており，「キー」がまだ登録されてない場合；
(4) unique_type.py にあらかじめ書かれている部分は，変更しない；
(5) 読み込みファイルのパスは，コマンドライン引数により指定する；
(6) 不要なモジュールは import しない；
(7) デバッグ用の print など必要ないものはコメントアウトし，出力が図 3 になるようにする；

```
このタイプのポケモンはすでにチームに存在します
このタイプのポケモンはすでにチームに存在します
このタイプのポケモンはすでにチームに存在します
60
{'アーボ': {'どく'},
 'イシツブテ': {'じめん', 'いわ'},
 'ウソッキー': {'いわ'},
 'ウパー': {'じめん', 'みず'},
 'ウリムー': {'こおり', 'じめん'},
 'エアームド': {'はがね', 'ひこう'},
 'オムナイト': {'みず', 'いわ'},
 'カイリュー': {'ひこう', 'ドラゴン'},
```

図 3　課題 13 の実行例：例外が発生した箇所の一部と最後の pprint の一部を表示している．pprint で出力しているため，キーがあいうえお順で表示されている．タイプパターンの異なり数は 60 である．

リスト 3　unique_type.py

```
import sys
from pprint import pprint

def read_file(file_path):
        titles, types = [], []
        with open(file_path, "r", encoding="utf8") as rfo:
            for row in rfo:
                _, tit, typ, *_ = row.rstrip().split("\t")
                titles.append(tit)
                types.append(set(typ.split()))
        return titles, types

class Team(dict):
    def __setitem__(self, key, value):
        if value in self.values():
            if ((key in self and self[key] != value) or (key not in self)):
                raise ValueError("このタイプのポケモンはすでにチームに存在します")
        super().__setitem__(key, value)


if __name__ == "__main__":
    titles, types = read_file(sys.argv[1])
    team = Team()
    for title, type_set in zip(titles, types):
        try:
            team[title] = type_set
        except Exception as e:
            print(e)

    print(len(team))
    pprint(team)
```

【課題 14：zukan.py】

list クラスを継承し，Monster クラスのインスタンスのみを append できる特殊なリスト Zukan クラスを実装せよ．

採点の都合上，以下の要件を満たすこととする：

(1) 出力結果を参考にして，ポケモン 1 体を表す Monster クラスを，本日習った，namedtuple または dataclass を使用して定義する；

(2) Monster クラスのインスタンスのみを append できる Zukan クラスを定義する；

（a）list クラスを継承する；

（b）Monster クラスのインスタンスのみを追加できるように，append メソッドをオーバーライドする；

（c）Monster クラスのインスタンス以外が append されそうになった時，ValueError を raise する；

(3) zukan.py にあらかじめ書かれている部分は，変更しない；

(4) 読み込みファイルのパスは，コマンドライン引数により指定する；

(5) 不要なモジュールは import しない；

(6) デバッグ用の print など必要ないものはコメントアウトし，出力が図 4 になるようにする；

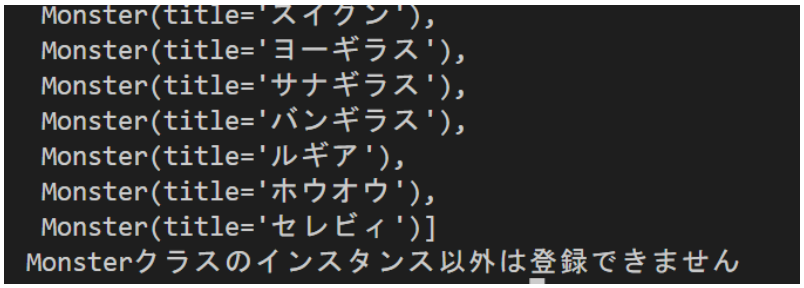

図 4　課題 14 の実行例

リスト 4　zukan.py

```
1  import sys
2  from pprint import pprint
3  from dataclasses import dataclass
4  #from collections import namedtuple
5
6  def read_names(file_path):
7      titles = list()
8      with open(file_path, "r", encoding="utf8") as rfo:
9          for row in rfo:
10             _, tit, _, *_ = row.rstrip().split("\t")
11             titles.append(tit)
12     return titles
13
14
15 @dataclass
16 class Monster:
17     title: str
18
19
20 class Zukan(list):
21     def append(self, value):
22         if not isinstance(value, Monster):
23             raise ValueError("Monsterクラスのインスタンス以外は登録できません")
24         super().append(value)
25
26
27 if __name__ == "__main__":
28     titles = read_names(sys.argv[1])
29     zukan = Zukan()
30     #Monster = namedtuple("Monster", ("title"))
31     for title in titles:
32         zukan.append(Monster(title))
33     pprint(zukan)
34     try:
35         zukan.append(251)
36     except Exception as e:
37         print(e)
```