【課題 07：shiritori.py】

しりとりの要領でポケモンの名前を順に抽出する iterable なクラスとそのイテレータクラスを実装せよ．

採点の都合上，以下の要件を満たすこととする：

(1) iterable な Zukan クラスを定義する；
  （a）Zukan クラスが iterable になるように，適切な特殊メソッドを追加する；
  （b）ただし，Zukan クラス自体はイテレータではないことに注意する；
(2) iterable な Zukan クラスに対応するイテレータクラスを定義する；
  （a）イニシャライザで，Zukan クラスのインスタンスを受け取り，インスタンス変数に設定する；
  （b）イニシャライザで，Zukan クラスのインスタンスから名前文字列をランダムに 1 つ選び，インスタンス変数に設定する（これが，しりとりの最初の単語となる）；
  （c）Zukan クラスのインスタンス（iterable）から，現在選ばれている単語（名前文字列）の最後の文字と等しい頭文字を持つ単語をすべて抽出する（これが，次の単語の候補となる）；
  （d）候補単語がなかった場合は，適切な例外を raise する；
  （e）候補単語がある場合は，その中からランダムに選択した単語を次の単語とする；
(3) 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；
(4) 配布した shiritori.py にあらかじめ書かれている部分は，変更しないこと；
(5) 出力の形式は，図 1 のようにする；

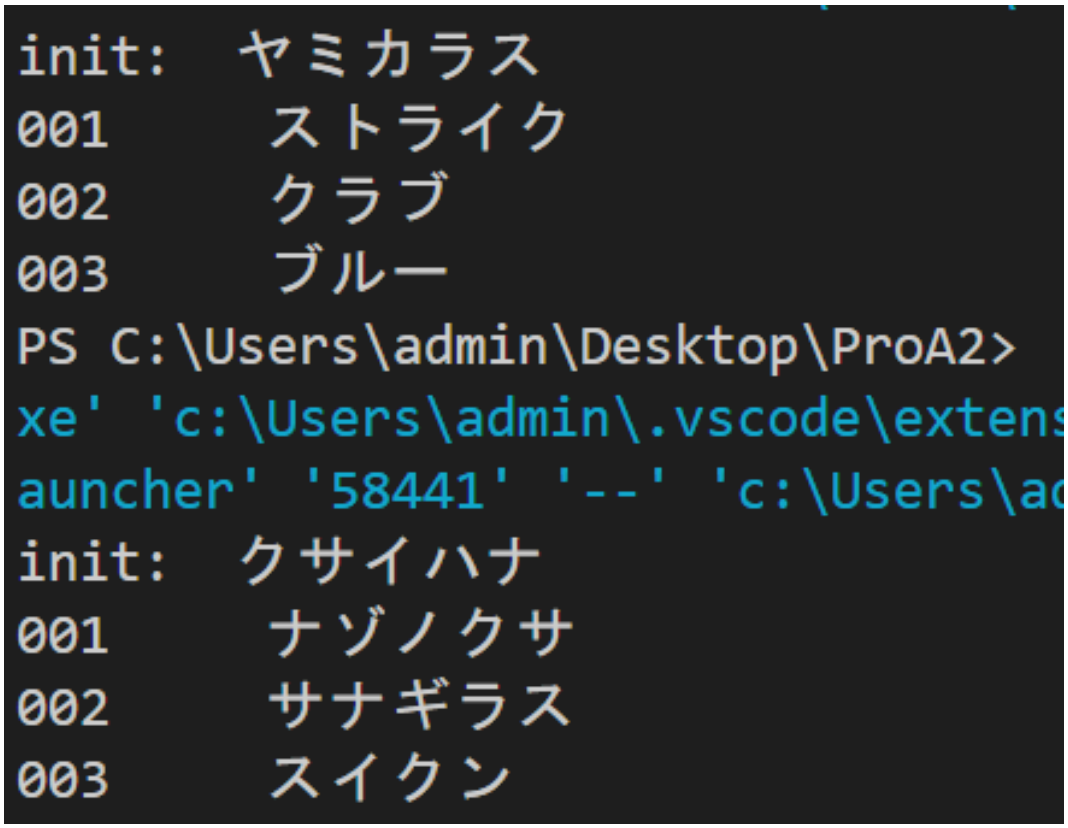

図 1　課題 07 の実行例：

リスト 1 shiritori.py

```
import sys
from random import choice

class Zukan:
    def __init__(self, file_path):
        self.titles = __class__.read_file(file_path)

    def __iter__(self):
        return ZukanIterator(self)

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


class ZukanIterator():
    def __init__(self, zukan):
        self.source = zukan
        self.current = choice(self.source.titles)
        print("init: ", self.current)

    def __iter__(self):
        return self

    def __next__(self):
        next_candidates = []
        for title in self.source.titles:
            if title[0] == self.current[-1]:
                next_candidates.append(title)
        if len(next_candidates) == 0:
            raise StopIteration()
        self.current = choice(next_candidates)
        return self.current


if __name__ == "__main__":
    zukan = Zukan(sys.argv[1]) # lec04/data/poke_names.txt
    for i, z in enumerate(zukan, 1):
        print(f"{i:03d}\t{z}")
```

【課題 08：party.py】

ポケモン 3 体からなるパーティーを構築し，スコアが最も高いパーティーを探すコードを実装せよ．配布した名前ファイル「poke_names.txt」と種族値ファイル「base_stats.txt」を読み込み，全 251 種のポケモンインスタンスを要素とするリストを作成する．そして，251 種から 3 体選ぶ全組合せに対して，スコアを計算する．ポケモンパーティーのスコアは，3 体のポケモンの種族値の合計値とする．

251 種のポケモンから 3 体を選ぶ組合せの総数は ${}_{251}\mathrm{C}_3 = 2,604,125$ であり，膨大となるため，全組合せを要素とするコンテナ（リストなど）を構築することなく，itertools モジュールの combinations 関数を利用すること．

採点の都合上，以下の要件を満たすこととする：

(1) ポケモン 1 体を表す Monster クラスはすでに定義されたものを使用する；
(2) 全ポケモンのリストを表す iterable なクラス MonsterList を定義する；
   （a）【済】静的メソッド read_files() は，名前ファイル「poke_names.txt」と種族値ファイル「base_stats.txt」を読み込み，名前文字列と種族値リストから Monster クラスのインスタンスを生成し，それらを要素とするリストを返すように定義してある；
   （b）【済】イニシャライザは，read_files() の戻り値を，インスタンス変数に設定している；
   （c）MonsterList クラスが iterable になるように，適切な特殊メソッドを定義する．※ただし，今回はインデックスによって要素にアクセスできるようにする（つまり，__iter__() ではない）；
(3) ポケモンパーティーに対して，スコア（種族値の合計）を計算する関数を定義する；
(4) 3 ポケモンからなるパーティーを生成して，最大値を求めるジェネレータ関数を定義する；
   （a）itertools の combinations 関数によるイテレータを用いて，全パーティーを 1 つずつ生成する；
   （b）※総数が非常に大きいため，全パーティーを要素とするコンテナは作らないこと；
   （c）生成したパーティーに対して，上で定義した関数によりスコアを計算する；
   （d）現在の（仮の）最大値と比較して，スコアが高いパーティーが出現したら，ジェネレータ関数の呼び出し元に，パーティーとスコアを返す；
   （e）呼び出し元では，戻された値（パーティーとスコア）を print する；
(5) 名前ファイルのパス，種族値ファイルのパスは，コマンドライン引数により，**この順番**で指定する；
(6) 出力の形式は，図 2 のようにする；

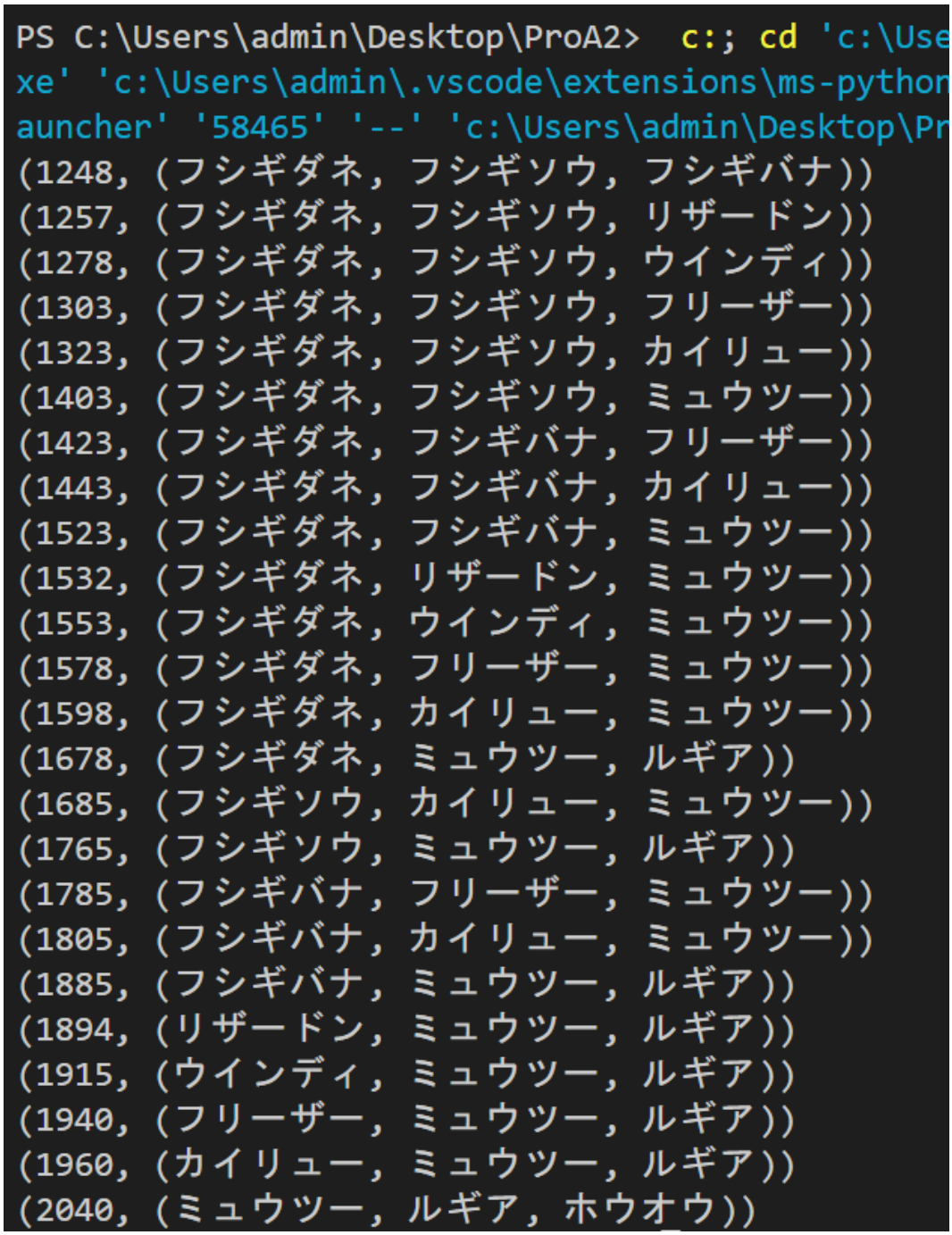

図 2　課題 08 の実行例：

リスト 2　party.py

```
1  from itertools import combinations
2  import sys
3
4  class Monster:
5      def __init__(self, title, stats):
6          self.title = title
7          self.stats = stats
8
9      def __repr__(self):
10         return self.title
11
```

```
12
13 class MonsterList:
14     def __init__(self, file_path1, file_path2):
15         self.monsters = __class__.read_files(file_path1, file_path2)
16
17     def __getitem__(self, idx):
18         return self.monsters[idx]
19
20     @staticmethod
21     def read_files(file_path1, file_path2):
22         titles = []
23         with open(file_path1, "r", encoding="utf8") as rfo:
24             for row in rfo:
25                 _, tit, _, *_ = row.rstrip().split("\t")
26                 titles.append(tit)
27
28         stats = []
29         with open(file_path2, "r") as rfo:
30             for row in rfo:
31                 row = row.rstrip()
32                 stats.append([int(col) for col in row.split(" ")])
33
34         monsters = [Monster(title, stat) for title, stat in zip(titles, stats)]
35         return monsters
36
37
38 def party_score(tpl):
39     s = 0
40     for mon in tpl:
41         s += sum(mon.stats)
42     return s
43
44
45 def max_party(lst):
46     max_val = 0
47     arg_max = None
48     for tpl in combinations(lst, 3):
49         val = party_score(tpl)
50         if val > max_val:
51             arg_max = tpl
52             max_val = val
53             yield max_val, arg_max
54     return max_val, arg_max
55
56
57 if __name__ == "__main__":
58     monster_lst = MonsterList(sys.argv[1], sys.argv[2]) # lec04/data/poke_names.txt, lec04/data/
          base_stats.txt
59     party_generator = max_party(monster_lst)
60     for res in party_generator:
61         print(res)
```