2022年10月17日(月) 5限
@研究棟A302

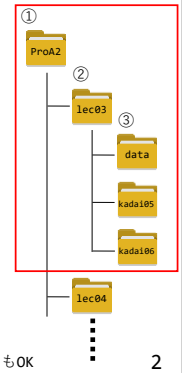
プログラミングA2
第3回

演習課題

1

準備

1. デスクトップなどの任意の場所に
   プログラミングA2用のフォルダを
   作成する
2. 1.の下に
   第3回授業用のフォルダを
   作成する
3. 2.の下に
   データファイル用のフォルダ，
   課題05，課題06用のフォルダを
   作成する



※必ずしも，このようなディレクトリ構造でなくてもOK

2

注意点

- 採点の都合上，
  指示された要件を満たすコードを実装すること
- 授業時間中に提出された場合は2点満点
- 授業時間外に提出された場合は1点満点
- 授業時間中に1問も提出できなかった場合は，
  学修支援センターで指導・チェックを受けること
  ※チェックだけでもOK
  ※1問だけ終わっていない状況でも，LSCに行くことはもちろんOK
- 授業時間外の質問は，Slackにて伏見が受け付ける
  （24時間365日営業）

3

3

5限課題チェックの流れ

1. **受講生：**課題ができたら，#授業中_課題チェック依頼 に
   依頼文を投稿する
   - 例：「課題03のチェックをお願いします」
2. **TASA：**スレッドに返信する で応答する
   - 例：「チェックするので，DM送ってください」
3. **受講生：**担当TASAにDMで課題プログラムを送信する
4. **TASA：**
   1. DMでフィードバック（正解／不正解ならどこが悪いのかを簡単に）
   2. 済を付ける
5. **受講生：**Moodleに提出する
6. **TASA：**Moodleに点数を入力する
   （あとで一気に入力する場合もある）

4

4

時間外課題チェックの流れ

時間内に1問も提出できなかった場合は，
学修支援センター(LSC)で指導・チェックを受けること
・意図：理解できていない可能性が高い→理解度を高める
・利点：加点が保証される(独自で提出して間違っていたら0点)

1. **受講生：**課題ができたら，Moodleに提出する

※時間外提出の場合は，Slackでのチェック依頼不要

- LSCでチェック済の場合：
  2. **教員：**その日のうちにMoodleに**加点**する
- LSCで未チェックの場合：
  2. **TASA：**次回講義時間中に**採点**，Moodleに点数を入力する

5

5

授業中質疑応答の流れ

- **緊急案件の場合**
  - 挙手によりTASAを呼ぶ
- **上記以外の場合**
  - Slackの #授業中_質問ボックス に質問内容を書き込む．
  - 込み入った内容でテキスト化するのが面倒なときは，
    「対面希望」とおおよその座席を書き込む．

基本的にSlackに投稿された順に対応する

6

6

【課題 05：sort_name.py】

ポケモンの名前を辞書順にソートするコードを実装せよ．配布したポケモンデータファイル「poke_names.txt」を読み込み，ポケモンの名前を抽出する．当該ファイルは 1 行が 1 匹のポケモンのデータであり，ポケモン番号，名前，タイプ，進化先がタブ区切りで書かれている．

辞書順のソートには，名前文字列を構成する 1 文字に対してスコアを算出し，その重み付き和により名前文字列のスコアを算出する．ord(1 文字) 関数を用いることで，「ア」ほど小さい，「ン」ほど大きな値を得ることができる．そして，数値の場合と同様に，先頭文字（上の位の文字）ほど大きな重みを掛けて足し合わせることで，名前文字列のスコアを計算できる．例えば，名前文字列「アイウ」のスコアは，（「ア」のスコア ×1 桁目の重み）＋（「イ」のスコア ×2 桁目の重み）＋（「ウ」のスコア ×3 桁目の重み）で求める．この時，1 桁目の重みは基数の 5 乗，2 桁目の重みは基数の 4 乗，3 桁目の重みは基数の 3 乗とするのが自然である．基数は，ord() 値の最大とするとシンプルである．ちなみに，10 進数の基数は 10，2 進数の基数は 2 である．

- 今回の対象となる名前文字列の長さの最大値は，「5」である．つまり，5 桁である．
- 今回の対象となる名前文字列に含まれる文字の中で，ord() の値の最大値は，「12540」である（環境によって異なるかも）．

採点の都合上，以下の要件を満たすこととする：

- ファイル読み込み用の関数を定義する；
  - 読み込み元のファイルパスを受け取るパラメータを持つ；
  - 2 列目の名前だけを抽出し，リストに格納する；
  - 名前のリストを返す；
- 結果書き込み用の関数を定義する；
  - 書き込み先のファイルパスと，書き込み対象のリストを受け取るパラメータを持つ；
  - リストの要素（この課題ではソート後の名前）を 1 行ずつ出力する；
- 名前文字列のスコアを計算する関数を定義する；
  - 名前文字列を受け取るパラメータを持つ；
  - 名前文字列を構成する 1 文字ずつに対して，ord() 関数によりアイウエオ順に高くなるスコアを求める；
  - 名前文字列の先頭（上の桁）ほど大きな重みを掛けながら，構成文字すべてのスコアを足し合わせることで，名前文字列のスコアを算出する；
  - ※実際の辞書では，伸ばす音を表す長音符号「ー」は，前の文字の母音により音を決めるが，処理が煩雑なので，今回の課題ではその事実は無視する．すなわち，「ー」は ord(”ー”) により変換された値をそのまま使う；
  - 名前文字列のスコアを返す；
- (1) 読み込みファイルと (2) 書き込みファイルのパスは，コマンドライン引数により，**この順番**で指定する；
- sorted() 関数，または，sort() メソッドに対して，名前文字列のスコアを計算する関数オブジェクトを，キーワード引数により指定する；
- 書き込み関数の 2 つの引数は，キーワード引数により指定する；
- 出力の形式は，Moodle にアップロードしてある「sort_names.txt」と全く同じようにする；

【課題 06：distance.py】

自分が選んだポケモンと最も類似するポケモンを求めるコードを実装せよ．配布した種族値ファイル「base_stats.txt」と名前ファイル「poke_names.txt」を読み込み，全 251 匹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，コマンドライン引数で指定した番号のポケモンを自分のポケモンとし，自分のポケモン以外の全ポケモンとの距離を求め，距離が最も小さなポケモンを最類似ポケモンとして出力する．

ポケモン間の距離は，6 次元の種族値ベクトル（6 要素からなるリスト）$\mathbf{x} = [x_1, \ldots, x_6]$ と $\mathbf{y} = [y_1, \ldots, y_6]$ 間の

(1) Manhattan：$\sum_{i=1}^{6} |x_i - y_i|$
(2) Euclidean：$\sqrt{(\sum_{i=1}^{6} |x_i - y_i|^2)}$
(3) Chebyshev：$\max_{1 \le i \le 6}\{|x_i - y_i|\}$

により求める．そして，各距離定義を表すラムダ式と自分のポケモンを引数として渡すことで，全ポケモンとの距離を計算し最類似ポケモンを求める関数を定義する．

採点の都合上，以下の要件を満たすこととする：

- 指定ポケモンとそれ以外のポケモンの距離を計算し，最類似ポケモンを求める関数 calc_similar_monster() を定義する；
  - 距離関数（ラムダ式）オブジェクトと指定ポケモン（この課題では自分のポケモン）のインスタンスを受け取るパラメータを持つ；
  - 引数として渡された距離関数に基づき，指定ポケモンとそれ以外との距離を計算する；
  - 距離が最小のポケモンインスタンスと距離の値をタプルとして返す；
- 距離のラムダ式を calc_similar_monster() の引数として渡すことで，各距離定義に基づく最類似ポケモンを求める；
  - 3 つの距離定義に基づき，引数として渡された 2 つのリスト間の距離を計算するラムダ式を定義する；
  - 距離の名前（「Manhattan」など）をキー，距離のラムダ式を値とした辞書を構築する；
  - 辞書に対する for 文の中で，ラムダ式と自分のポケモンを引数として指定して calc_similar_monster() を呼び出す；
  - 戻り値として返ってきた最類似ポケモンとその距離を標準出力する；
- pokemon.py には変更を加えない；
- 種族値ファイル「base_stats.txt」と名前ファイル「poke_names.txt」を読み込む関数は，リスト 1 を使用する（Moodle にある pokemon.py にすでに書かれている）；
- (1) 種族値ファイルのパス，(2) 名前ファイルのパス，(3) 自分のポケモン番号は，コマンドライン引数により，**この順番**で指定する；
- 出力の形式は，図 1 のようにする；

```
PS C:\Users\admin\Desktop\ProA2> python .\lec03\kadai06\distance.py .\lec03\data\base_stats.txt .\lec03\data\poke
_names.txt 2
僕のポケモン：フシギソウ
Manhattan距離：一番類似するのはベイリーフで，距離は34
Euclidean距離：一番類似するのはクサイハナで，距離は22.538855339169288
Chebyshev距離：一番類似するのはフシギダネで，距離は15
PS C:\Users\admin\Desktop\ProA2> python .\lec03\kadai06\distance.py .\lec03\data\base_stats.txt .\lec03\data\poke
_names.txt 10
僕のポケモン：キャタピー
Manhattan距離：一番類似するのはビードルで，距離は20
Euclidean距離：一番類似するのはビードルで，距離は10.0
Chebyshev距離：一番類似するのはビードルで，距離は5
PS C:\Users\admin\Desktop\ProA2> python .\lec03\kadai06\distance.py .\lec03\data\base_stats.txt .\lec03\data\poke
_names.txt 144
僕のポケモン：フリーザー
Manhattan距離：一番類似するのはカメックスで，距離は50
Euclidean距離：一番類似するのはスイクンで，距離は23.452078799117146
Chebyshev距離：一番類似するのはスイクンで，距離は15
PS C:\Users\admin\Desktop\ProA2>
```

図 1　課題 06 の実行例：

リスト 1　pokemon.py

```python
def read_stats(file_path):
    stats = []
    with open(file_path, "r") as rfo:
        for row in rfo:
            row = row.rstrip()
            stats.append([int(col) for col in row.split(" ")])
    return stats

def read_names(file_path):
    names, types, evols = [], [], []
    with open(encoding="utf8", file=file_path, mode="r") as rfo:
        for row in rfo:
            _, nam, typ, *evo = row.rstrip().split("\t")
            names.append(nam) # 名前の文字列をappend
            types.append(typ.split(" ")) # タイプのリストをappend
            evols.append([e for e in evo]) # 進化先のリストをappend

    return names, types, evols


class Monster:
    def __init__(self, title):
        self.title = title
        self.stats = []

    def __repr__(self):
        return self.title
```