提出日：2024/5/31

プログラミング演習　第７回演習レポート

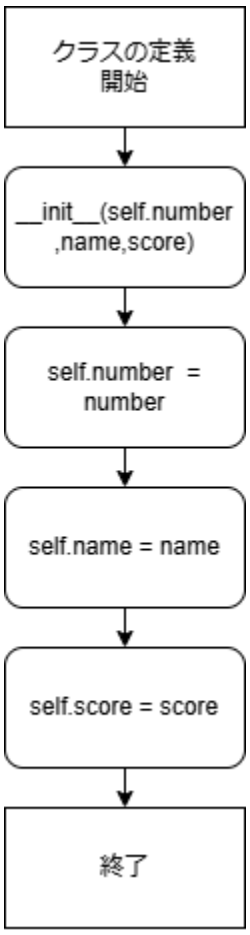
担当教員：杉本　千佳先生
所属：理工学部　数物・電子情報系学科
電子情報システムEP
学年・クラス：２年　Fe1
学籍番号：2364092
氏名：熊田　真歩

（１）課題番号：基本課題２

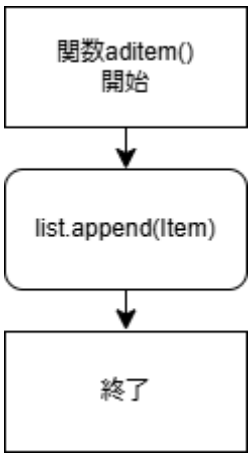
課題名：整列過程におけるデータの比較回数とデータの入れ換え回数を表示し、整列データをファイルに書き込む

（２）プログラムのフローチャート（関数ごと）

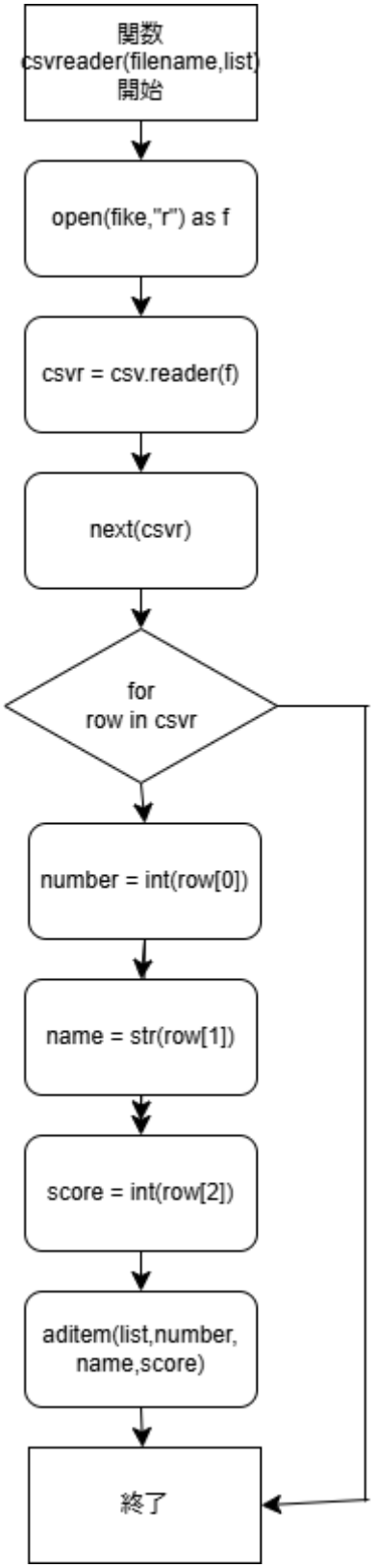
・クラスの定義関数



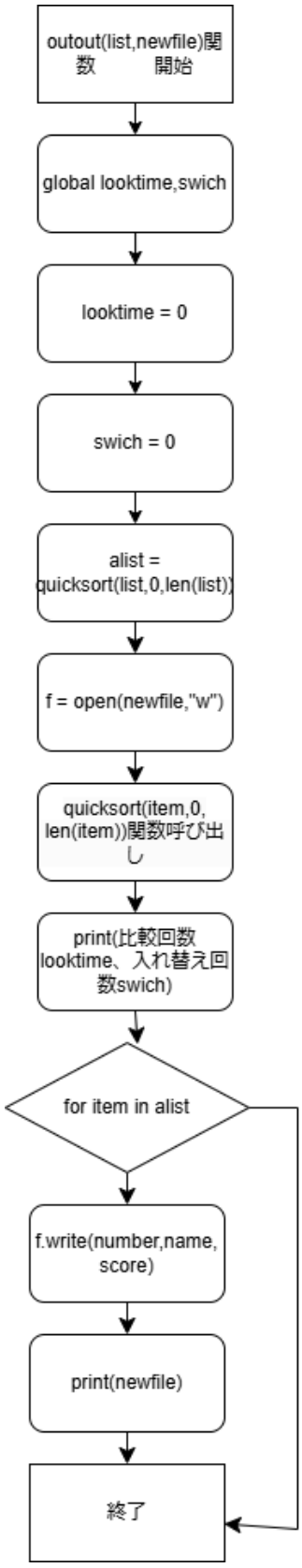
・aditem(item_list,number:int,name:str,score:int)関数



・csvreader(filename:str, item_list)関数



・outout(item_list,newfile)関数



・quicksort 関数

（3）アルゴリズムが「正しいこと」である説明

本プログラムでは学籍番号、名前、成績が記入された CSV ファイルに対してクイックソートを用いて成績順に並び替え、ファイルに書き込むという事を行った。ここで、入力ファイル名に関しては入力時のミスによるエラーを回避するために入力式ではなく予め入力してあるという形を採用した。よって、以下では入力ファイルが存在するという条件のもと、アルゴリズムの正しさについて議論する。

まず、すべての入力に対してプログラムは停止するということについてだが、上記で述べた通り本プログラムでは入力する内容は入出力のファイル名のみであり、入力では存在する CSV ファイル入れればよい。しかし、CSV は学籍番号、名前、成績の要素からなるものでなくてはいけない。この条件がそろう CSV ファイルであればアルゴリズムはどんな入力に対しても停止する。

次にすべての入力に対してアルゴリズムは「正しい答え」を出力することについて議論する。上記と同じ通り、ここですべての入力とは条件を満たす CSV が存在し、それが入力ファイルとして存在しているという条件下のもと議論する。課題ではデータ数が 8,288, 2240 個の時に対してプログラムを実行し、いずれの場合も正しい出力結果が得られた。すなわち、データ数の量によらずこのプログラムは正しい答えを出力することが分かる。すなわちこの条件に当てはまる CSV ファイルであればすべての入力に対してアルゴリズムは正しい答えを出力する。

（4）ソース・プログラムの説明

```
import csv           #csv ファイルを用いるための設定
looktime = 0         #looktime(比較回数)の初期値は 0
swich = 0            #swich(入れ替え回数)の初期値は 0

class Item:          #クラス、Item の定義
  def __init__ (self, number:int, name:str, score:int):#クラスの設定
   self.number = number       #生徒番号の設定
   self.name = name           #名前の設定
   self.score = score         #成績の設定

def aditem(item_list,number:int,name:str,score:int):#クラスに要素を追加する関数
  item_list.append(Item(number,name,score))#リストに要素を追加

def csvreader(filename:str, item_list):  #CSV ファイルを読み込む関数
  with open(filename,'r') as f:          #指定されたファイルを開く
    csvr = csv.reader(f)                #CSV ファイルの読み込み
```

```
    next(csvr)                          #読み込みをスキップ
    for row in csvr:                    #CSVファイルの1行ずつに対して
      number = int(row[0])              #学籍番号はint型
      name = str(row[1])                #名前はstr型
      score = int(row[2])               #成績はint型
      aditem(item_list, number, name, score)   #これらをリストに追加

def outout(item_list,newfile):          #出力関数
  global looktime,swich                 #looktimeとswichをグローバル変数とする
  looktime = 0                          #looktimeの初期値は0
  swich = 0                             #swichの初期値は0
  alist = quicksort(item_list,0, len(item_list)-1)         #alistの値をクイックソートで並び
                                                            替えた関数とする
  f = open(newfile,"w")                                    #newfileをfとして開く
  quicksort(item_list,0, len(item_list)-1)                  #クイックソート関数の呼び出し
  print("比較回数="+str(looktime),"入れ替え回数="+str(swich)) #比較回数と入れ替え回数の出力
  for item in alist:                                       #alistのそれぞれのitemに対して
    f.write(f"{item.number:4},{item.name:16},{item.score}¥n")#ファイルに内容を書き込む
  print("出力ファイル名:"+str(newfile))                     #出力ファイル名の出力

def quicksort(a:list, left:int,right:int):   #順序を入れ替えるクイックソート関数
    global looktime,swich                    #looktime(比較回数),swich(入れ替え回数)をグロ
                                               ーバル変数とする
    if left < right:                         #rightがleftより大きい時
        i, j = left, right                   #i=left,j=rightとする
        x = a[(left + right) // 2].score     #リストの中央にあるスコアの値をxに代入
        while True:                          #条件が正しい間
            while(a[i].score < x):           #i番目リストのスコアがリストの中央のスコアより小さ
                                               い時(中央より左側について)
                looktime += 1                #比較回数を1回増やす
                i += 1                       #iをインクリメントすることで比較対象を次の値へとずら
                                             す(右にずらす)
            while(x < a[j].score):           #i番目リストのスコアがリストの中央のスコアより小さ
                                             い時(中央より右側について)
                looktime += 1                #比較回数を1回増やす
```

```
            j -= 1                          #j を負にインクリメントすることで比較対象を次の値へ
                                              とずらす（左にずらす）
        if (i <= j):                        #i の値が j 以下になったら
            if i < j:                       #j の値が i の値よりも大きい時
               swich += 1                   #入れ替え回数の値を 1 回増やす
               a[i], a[j] = a[j], a[i] #リストの i 番目のデータと j 番目のデータを入れ替える
            i += 1                          #i の値を増やす
            j -= 1                          #j の値を減らす
        if i > j:                           #i の値が j の値よりも大きい時
            break                           # ループ終了
    if left < j:                            #最も左の値が j よりも小さい時
        quicksort(a, left, j)               #quicksort(a, left, j)関数の呼び出し
    if i < right:                           #最も左の値が i よりも大きい時
        quicksort(a , i, right)             #quicksort(a , i, right)関数の呼び出し
    return a                                # a に値を返す
item_list = []                              #item_list の初期値は空
csvreader('data_8.csv', item_list)          #入力ファイル
outout(item_list,"output_8.txt")            #出力ファイル
```

（5）考察

本プログラムでは前回に引き続き、CSV ファイルの内容の書き換えを行った。しかし、第 6 回とは異なり、CSV ファイルのデータの入れ替えを行う際にクイックソートを用いて並び替えを行った。ここで前回の入れ替え方法とクイックソートを用いた入れ替え方法を比較してそのメリット・デメリットについて考察する。以下に data_8.csv, data_288.csv, data_2240.csv の第 6 回と第 7 回の入れ替え方法でファイルの内容を並び替えた時の比較回数と入れ替え回数を示す。

	第 6 回のアルゴリズム	クイックソート(7 回)
data_8.csv	比較：21　替え：5	比較：18　替え：5
data_288.csv	比較：41041　替え：282	比較：3074　替え：818
data_2240.csv	比較：2505441 替え：2216	比較：27147 替え：11494

表から明らかにわかることとして、第 6 回で採用したアルゴリズムは比較回数がクイックソートに比べ多い。しかし、入れ替え（表中の替え）回数はクイックソートの方がより多い傾向にある。ここで「効率の良いプログラム」とはこの表中の比較回数のより少ないもの

である。比較して入れ替えなくとも、作業に要する時間は比較すればかかることになるからである。ここで、クイックソートではなぜ比較回数がより少なくて済むのかという点について考察する。クイックソートは中央の値をピンで固定することで、二つのグループに分割し、比較を行う。これにより比較回数は初めからずらしていくプログラムに比べ比較回数がより少なくなることが分かる。表からも明らかなように、このクイックソートはそのデータ数が多いほど効果を発揮する。膨大なデータ数を扱う際には不可欠なアルゴリズムであることが分かる。また、本プログラムではクイックソートはリストの中の数を扱ったため、right の値は len(item_list)-1 と－ 1 をしたことに注意した。そして left の値は 0 とした。

プログラムの改善の余地について考察する。本プログラムでは、CSV ファイルが学籍番号、名前、成績の三要素からなるものでないと正常に稼働しない。この条件以外のものでも稼働するようなプログラムは非常に複雑であるが、この条件を満たさない場合はエラーメッセージを表示する等の処理が加わるとより汎用性の高いプログラムとなると考えた。

（6）参考文献

・クイックソートに関して
①アルゴリズム、スライドサンプルプログラム

②Zenn：ダーマン「Python でクイックソート完全解説」
https://zenn.dev/yutabeee/articles/e8fb2847cfc980　　2024/05/31 アクセス

③スズメの本棚「Python でクイックソートを実装してみた」
https://tech-shelf.hatenablog.com/entry/algorithm/quicksort　　2024/05/31 アクセス

（7）感想

前回のプログラムではクラスの内部に関数を作成することを試みてアルゴリズムを組んだが、本プログラムではそれを改良し関数をクラスの外部に作成した。結果論であるが、関数はクラスの外部に設定しする方がより明確でコードを組むのも容易であった。クイックソートはアルゴリズムのサンプルプログラムのものをそのまま用いたが、意味の理解がはっきりできていなかったクイックソートの使い方が明確にわかるようになり、やはり実際に習ったプログラムを活用してコードを書いてみる重要性を感じた。また、比較回数と入れ替え回数ではクイックソートのメリットを実感でき良い学びへとつながった。しかしながら、クイックソートに関してはアルゴリズムを自分で作成したのではなく、アルゴリズムのサンプルプログラムを用いたため、自分でクイックソートを作れるようになったかというと怪しい、というか書けない。すなわち、これを自力でしっかりかけるようにすることは今後の課題であると思った。