20163079-ngj04_out

File ../files_to_test/20163079-ngj04.py differs from original due to handling of invisible characters

PROJECT CODE

```
##    Name:              Joshua Ng
##    Student Number:    20163079
##    version:           25/5/2017

##    project2.py is a program to be used to decide an election for a single
##    seat under the Preferential Voting system used in the Western
##    Australia Legislative Assembly (the Lower House).

import os.path

##    getCandidates(f) returns a list containing the candidates' names from
##    the file f.The names will be one per line with no extraneous
##    characters. Disregard any blanklines in f. If f doesn't exist, it prints
##    an error message and return an empty list. For example,
##    getCandidates("candidates.txt") = ["Major Clanger", "Soup Dragon",
##    "Froglet", "Iron Chicken", "The Cloud"].
##    >>> getCandidates(f) returns ["Major Clanger", "..], || print(error), []
def getCandidates(f):
    candidates = []
    if os.path.isfile(f):
        file_f = open(f, "r")
        for names in file_f:
            if names not in ['\n', '\r\n']:
                candidates.append(names.strip())
        file_f.close()
    else:
        print("file doesn't exist")
    return candidates

##    storeCandidateData returns a dictionary assigning 1,2,3,...etc to
##    each Candidates' data.
##    e.g. storeCandidateData(["Major Clanger", "..]) returns
##    [(1, 'Major Clanger'), (2, ...]
def storeCandidateData(candidates):
    data = {}
    for i in range(0, len(candidates)):
        data[i+1] = candidates[i]
    return data

##    listOfLists returns a list of lists of size number.
def listOfLists(size):
    list_of_lists = []
    for i in range (size):
        list_of_lists.append([])
    return list_of_lists

##    parseVote(s, option) returns the vote from s. option enables
##    Optional Preferential Voting.
##    Return 0 for an empty vote (Optional Preferential Voting).
##    Returns -1 if there are any non-digits.
##    e.g. ^IparseVote("15") = parseVote(" 15 ") = 15.
##            parseVote("", True) = parseVote(" ", True) = 0,
##            parseVote("", False) = parseVote("no") = parseVote("1 5") = -1,
def parseVote(s, option=False):
    vote = s.strip()
    if (vote == "") & option:
        return 0
    elif not vote.isdigit():
        return -1
    else:
        return int(s)

##    parsePaper(s, n, option) returns the votes from the ballot paper s in an
##    election with n candidates, plus an error message. option enables
##    Optional Preferential Voting.
```

20163079-ngj04_out

```
##    If s is formal, it returns the list of numbers found in s and the empty
##    string. If s is informal, return an empty list of numbers and the
##    appropriate string below.
##    i.e. parsePaper("3, 1, 2", 3) = ([3, 1, 2], "")
##    i.e. parsePaper(", , ,", 4) = parsePaper("0, 0", 4) = ([ ], "blank"),
##    i.e. parsePaper("3, -8, 1", 3) = parsePaper("3, 1.8, 2", 3) = parsePaper
##    ("pointless, 2, 1", 1) = ([ ], "non-digits"),
##    i.e. parsePaper("1,2,3,4,5", 4) = ([ ], "too long"),
##    i.e. parsePaper("6,1,2,3,4",5) = ([ ], "too high")
##    i.e. parsePaper("1, 1", 2) = ([ ], "duplicate")
##    i.e parsePaper("1, , , ", 4, True) = ([1], "")
##
##    parsePaper(s,n): >> ([3, 1, 2], ""), >> ([ ], "blank"),
##                     >> ([ ], "non-digits"), >> ([ ], "too long")
##                     >> ([ ], "too high"), >> ([ ], "duplicate")
def parsePaper(s, n, option=False):
    ballot = s.split(",")
    len_ballot = len(ballot)
    votes = []
    blank_vote = 0

    if len_ballot > n:
        return ([], "too long")

    for vote in ballot:
        parse_vote = parseVote(vote, option)
        if parse_vote == -1:
            return ([], "non-digits")
        elif parse_vote > n:
            return ([], "too high")
        elif (parse_vote in votes) & (parse_vote != 0):
            return ([], "duplicate")
        elif parse_vote == 0:
            blank_vote+=1
            votes.append(0)
        else:
            votes.append(parse_vote)

    if blank_vote == len_ballot:
        return ([], "blank")

    return (votes, "")

##    getPapers(f, n, option) returns a list containing the ballot papers from
##    the file f, in  an election with n candidates. Each line of the file
##    is treated as a separate paper. If f doesn't exist, it will print error
##    message and return the empty list. option enables Optional Preferential
##    Voting. For example:
##    getPapers("smallfile.txt", 4) = [[1, 2, 3, 4], [1, 3, 2, 4]...
##    getPapers("smallfile.txt", 4, True) = [[3,2,1], [...
def getPapers(f, n, option=False):
    papers = []
    if os.path.isfile(f):
        file_f = open(f, "r")
        for line in file_f:
            paper = []
            paper = parsePaper(line, n, option)
            if paper[0] != []:
                papers.append(paper[0])
        file_f.close()
    else:
        print("file doesn't exist")
    return papers

##    orderedPaper(p) returns a list of candidates in order of the
##    voterM-bM-^@M-^Ys preferences. For example:
##    orderedPaper([2,4,5,3,1]) >> [5,1,4,2,3]
##    orderedPaper([4,5,0,3,1]) >> [5,4,1,2]
##    @param p is the list of the voterM-bM-^@M-^Ys preferences
##    @returns list of candidates
```

20163079-ngj04_out

```
def orderedPaper(p):
    ordered = []
    preferences = []
    for i in range(len(p)):
        if p[i] != 0:
            preferences.append((p[i],i))
    preferences.sort()
    for i in range(len(preferences)):
        ordered.append(preferences[i][1]+1)
    return ordered

##    orderedPapers(ps) returns an ordered list of papers, ps. E.g.
##    orderedPapers([[2,4,5,3,1], [4,5,0,3,1]]) = [[5, 1, 4, 2, 3],
##    [5, 4, 1, 2]]
def orderedPapers(ps):
    papers = []
    for p in ps:
        papers.append(orderedPaper(p))
    return papers

##    countVotes(cs, ct, ps) updates the dictionary, cs, of candidatesM-bM-^@M-^Y
##    lists of votes with the ballot papers ps. The total votes for each
##    candidate, ct, will also be updated.
##    For example, a = {1: [], 2:[], 3:[], 4:[], 5:[]}
##    b = {1:0, 2:0, 3:0, 4:0, 5:0}
##    countVotes(a, b, [[1,3,4,5,2], [4,2,5,3,1]])
##    list(a.items()) = [(1, [[1, 3, 4, 5, 2]]), (2, []), (3, []), (4,
##    [[4, 2, 5, 3, 1]]), (5, [])]
def countVotes(cs, ct, ps):
    for p in ps:
        while p != []:
            if not p[0] in cs:
                p.pop(0)
            else:
                cs[p[0]].append(p)
                break
    for c in cs:
        ct[c] = len(cs[c])

##    printCount(ct, cn, n) displays the ct, the candidates total votes.
##    cn is the candidates names. n is the number of counts.
def printCount(ct, cn, n):
    print("Count {}".format(n))
    for i in range(len(ct)):
        print("{0}\t{1}".format(ct[i][1], cn[ct[i][0]]))
    print()

##    printCandidate prints the winner or the eliminated candidates.
def printCandidate(ct, cn, win):
    if(win == False):
        for i in ct:
            print("Candidate {} has the smallest number of votes and is "
                  "eliminated from the count".format(cn[i[0]]))
        print()
    if(win == True):
        print()
        print("Candidate {} is elected".format(cn[ct[0][0]]))

##    byTotalVotes is a helper method to sort candidate_total_list
def byVotes(pair):
    return pair[1]

##    main will require the file-name for the file of candidates names
##    and the file-name of the file of ballots option enables Optional
##    Preferential Voting.
def main(candidates_file_name, ballots_file_name, optional=False):
    #get candidates
    candidates = getCandidates(candidates_file_name)
    if candidates == []:
        print("no candidates")
```

```
        return
    number_candidates = len(candidates)

    #get papers
    papers = getPapers(ballots_file_name, number_candidates, optional)
    if papers == []:
        print("no votes")
        return

    #setup candidate data, number of eligible votes, winner condition
    candidate_names = storeCandidateData(candidates)
    list_of_lists = listOfLists(number_candidates)
    candidate_voters = storeCandidateData(list_of_lists)
    list_of_lists = listOfLists(number_candidates)
    for i in list_of_lists:
        i.append(0)
    candidate_total_votes = storeCandidateData(list_of_lists)
    total_votes = len(papers)
    winner = False

    #count votes
    voter_pool = orderedPapers(papers)
    for count in range(1, number_candidates):
        countVotes(candidate_voters, candidate_total_votes, voter_pool)

        candidate_total_list = list(candidate_total_votes.items())
        candidate_total_list.sort(key=byVotes, reverse=True)

        #print result
        printCount(candidate_total_list, candidate_names, count)

        #check if candidate has <50% of eligible votes
        if candidate_total_list[0][1] > total_votes//2:
            winner = True
            printCandidate(candidate_total_list, candidate_names, winner)
            break

        #check to see if there are more than one lowest candidate
        low_total = candidate_total_list[-1][1]
        lowest = []
        for i  in candidate_total_list:
            if i[1] == low_total:
                lowest.append(i)

        #check for a tie
        if len(lowest) == len(candidate_total_list):
            winner = True
            candidate_total_list.sort()
            printCandidate([candidate_total_list[-1]], candidate_names, winner)
            break

        #check if there are only two candidates.
        if len(candidate_total_list) < 3:
            winner = True
            printCandidate(candidate_total_list, candidate_names, winner)
            break

        #print candidate winner or eliminated canidates
        printCandidate(lowest, candidate_names, winner)

        #update pool of elegible votes to be recounted.
        voter_pool = []
        for i in lowest:
            voter_pool += candidate_voters[i[0]]
            del candidate_voters[i[0]]
            del candidate_total_votes[i[0]]

if __name__ == "__main__":
    main("candidates.txt", "papers2.txt", True)
```

20163079-ngj04_out

```
-----------------------------------

END TO END TESTING

Handling of Formal Votes, full preferential

Test 0: Full Preferential, 50% on first count (stop before end). candidate file: candidat
es.txt  papers file: papers1.txt
Count 1
11      Major Clanger
5       Soup Dragon
3       The Cloud
2       Froglet
1       Iron Chicken

Candidate Iron Chicken has the smallest number of votes and is eliminated from the count

Count 2
11      Major Clanger
5       Soup Dragon
4       The Cloud
2       Froglet

Candidate Froglet has the smallest number of votes and is eliminated from the count

Count 3
12      Major Clanger
6       Soup Dragon
4       The Cloud


Candidate Major Clanger is elected


---------------------------
Test 1: Full Preferential, one vote for one and only candidate. candidate file: candidate
s1.txt  papers file: papers2.txt
```

```
---------------------------
Test 2: Full Preferential, two horse race, equal votes. candidate file: candidates2.txt
papers file: papers3.txt
Count 1
2       Major Clanger
2       Soup Dragon


Candidate Soup Dragon is elected


---------------------------
Test 3: Full Preferential, distribution of prefs, grind out to last pair. candidate file:
 candidates.txt  papers file: papers4.txt
Count 1
8       Major Clanger
5       Soup Dragon
3       The Cloud
2       Froglet
1       Iron Chicken

Candidate Iron Chicken has the smallest number of votes and is eliminated from the count

Count 2
8       Major Clanger
5       Soup Dragon
4       The Cloud
2       Froglet

Candidate Froglet has the smallest number of votes and is eliminated from the count
```

20163079-ngj04_out

```
Count 3
9       Major Clanger
6       Soup Dragon
4       The Cloud

Candidate The Cloud has the smallest number of votes and is eliminated from the count

Count 4
11      Major Clanger
8       Soup Dragon


Candidate Major Clanger is elected


----------------------------
Test 4: Full Preferential, distribution of prefs, candidate with zero 1 prefs. candidate
file: candidates.txt  papers file: papers6.txt
Count 1
11      Major Clanger
5       Soup Dragon
4       The Cloud
2       Froglet
0       Iron Chicken

Candidate Iron Chicken has the smallest number of votes and is eliminated from the count

Count 2
11      Major Clanger
5       Soup Dragon
4       The Cloud
2       Froglet

Candidate Froglet has the smallest number of votes and is eliminated from the count

Count 3
12      Major Clanger
6       Soup Dragon
4       The Cloud


Candidate Major Clanger is elected


----------------------------
Test 5: Full Preferential, distribution of prefs, equal smallest, grind out to last pair.
 candidate file: candidates.txt  papers file: papers5.txt
Count 1
11      Major Clanger
5       Soup Dragon
4       The Cloud
2       Froglet
2       Iron Chicken

Candidate Froglet has the smallest number of votes and is eliminated from the count
Candidate Iron Chicken has the smallest number of votes and is eliminated from the count

Count 2
12      Major Clanger
6       Soup Dragon
6       The Cloud

Candidate Soup Dragon has the smallest number of votes and is eliminated from the count
Candidate The Cloud has the smallest number of votes and is eliminated from the count

Count 3
24      Major Clanger
```

20163079-ngj04_out

```
Candidate Major Clanger is elected


---------------------------
Handling of Formal Votes, full preferential

Test 6: Optional Preferential, effective first past post (less than 50% of pool). candida
te file: candidates.txt  papers file: papers7.txt
Count 1
4       Major Clanger
3       Soup Dragon
2       Froglet
0       Iron Chicken
0       The Cloud

Candidate Iron Chicken has the smallest number of votes and is eliminated from the count
Candidate The Cloud has the smallest number of votes and is eliminated from the count

Count 2
4       Major Clanger
3       Soup Dragon
2       Froglet

Candidate Froglet has the smallest number of votes and is eliminated from the count

Count 3
4       Major Clanger
3       Soup Dragon


Candidate Major Clanger is elected


---------------------------
Test 7: Optional Preferential, two horse race, equal votes. candidate file: candidates2.t
xt  papers file: papers8.txt
Count 1
2       Major Clanger
2       Soup Dragon


Candidate Soup Dragon is elected


---------------------------

Handling of Informal Votes

Test 8: 1 formal, one informal (2, 2 votes). candidate file: candidates.txt  papers file:
 papers9.txt
Count 1
1       Major Clanger
0       Soup Dragon
0       Froglet
0       Iron Chicken
0       The Cloud


Candidate Major Clanger is elected


---------------------------
Test 9: 1 informal vote, none else. candidate file: candidates.txt  papers file: papers10
.txt
no votes


---------------------------
Test 10: 1 formal, 1 informal vote (includes 0). candidate file: candidates.txt  papers f
ile: papers11.txt
```

20163079-ngj04_out

```
Count 1
1       Major Clanger
1       Soup Dragon
0       Froglet
0       Iron Chicken
0       The Cloud

Candidate Froglet has the smallest number of votes and is eliminated from the count
Candidate Iron Chicken has the smallest number of votes and is eliminated from the count
Candidate The Cloud has the smallest number of votes and is eliminated from the count

Count 2
1       Major Clanger
1       Soup Dragon


Candidate Soup Dragon is elected


---------------------------
Test 11: 1 formal, 1 informal vote (too long). candidate file: candidates.txt  papers fil
e: papers12.txt
Count 1
1       Major Clanger
0       Soup Dragon
0       Froglet
0       Iron Chicken
0       The Cloud


Candidate Major Clanger is elected


---------------------------
Test 12: 1 formal, 1 informal vote (truncated, and optpref, no 1 pref). candidate file: c
andidates.txt  papers file: papers14.txt
Count 1
2       Major Clanger
0       Soup Dragon
0       Froglet
0       Iron Chicken
0       The Cloud


Candidate Major Clanger is elected


---------------------------
```