```
# Wrestler Class

import sys
from random import *
import os;

# Wrestler object.  Contains the data for a wrestler.
class wrestler:
    def __init__(self, id, fname, lname, school,  weight):
        self.start = False;
        self.id = id;
        self.fname = fname;
        self.lname = lname;
        self.school = school;
        self.weight = weight;
        self.abname = self.fname + " " + self.lname + "(" + self.school +")";

    def __str__(self):
        return "Wrestler " + str(self.id) + ": " + self.abname + " " + str(self.weight);

# Helper Methods

# Prints the wrestlers in each group.  Will flag any group that is less than 3 wrestlers
def printGroups(groups):
    count = 0;
    for group in groups:
        count += 1;
        groupMaxWeight = getMaxWeight(group);
        #groupMinWeight = getMinWeight(group)* (1+allowance);
        size = len(group);
        if size == 1:
            size = ("****************************************** This is a one man bracket.  please
fix this.");
        elif int(size) <= 3:
            size = "****************************************** Bracket Size of " + str(size);
        else:
            size = str(size);

        print("Bracket " + str(count) + " Max weight: " + str(groupMaxWeight) + "\tSize: " + size);

        firsthalf = [0, 3, 4, 7];
        secondhalf = [1, 2, 5, 6];

# Testing if two wrestlers from same school are wrestling one another.

        if len(group) == 8:
            for i in range(0, len(firsthalf)-1):
                for j in range(i+1, len(firsthalf)):
                    if group[firsthalf[i]].school == group[firsthalf[j]].school:
                        print("****************************************** " +
str(group[firsthalf[i]]) + " and " + str(group[firsthalf[j]]) + " are in the first half of the
bracket!");
            for i in range(0, len(secondhalf)-1):
                for j in range(i+1, len(secondhalf)):
                    if group[secondhalf[i]].school == group[secondhalf[j]].school:
                        print("****************************************** " +
str(group[secondhalf[i]]) + " and " + str(group[secondhalf[j]]) + " are in the second half of the
bracket!");

        for i in range(0, len(group)-1):
            for j in range(i + 1, len(group)):
                if group[i].school == group[j].school:
                    print("****************************************** " + str(group[i]) + " and "
+ str(group[j]) + " are in the same bracket!");
```

```
        for wrestler in group:
            print("\t" + str(wrestler));

# Gets the heaviest wrestler for the group
def getMaxWeight(group):
    max = group[0].weight;
    for wrestler in group:
        if wrestler.weight > max:
            max = wrestler.weight;
    return max;

# Gets the lightest wrestler for the group
def getMinWeight(group):
    min = 10000;
    for wrestler in group:
        if wrestler.weight <  min:
            min = wrestler.weight;
    return min;

# Sorts the list of wrestlers by weight given list of wrestlers
def sortByWeight(wrestlers):
    wrestlers.sort(key=lambda x: x.weight, reverse=False);
    sortedArr = sorted(wrestlers, key=lambda x: x.weight, reverse=False);
    return sortedArr;

# Sorts the groups by weight given list of groups
def sortGroupsByWeight(groups):
    groups.sort(key=lambda x: getMaxWeight(x), reverse=False);
    sortedArr = sorted(groups, key=lambda x: getMaxWeight(x), reverse=False);
    return sortedArr;

# Prints the stats of the a list of groups.  Shows the amount of each size group and total
def printStats(groups, errors):
    print("\nStats:")
    stats = [];
    total = [];
    for i in range(0, 3):
        total.append(0);
    for i in range(0, 500):
        stats.append(0);
    index = 0;
    for group in groups:
        index += 1;
        if len(group) == 1:
            errors += "\nBracket " + str(index) + " is a one man group, fix this before writing to
RRG file";
        stats[len(group)] += 1;
    for i in range(0, len(stats)):
        if stats[i] != 0:
            percent = float(stats[i]*100)/len(groups);
            total[0] += stats[i];
            total[1] += percent;
            total[2] += i * stats[i];
            print("Size " + str(i) + ":\t" + str(stats[i]) + "\t" + str(round(percent, 2)) + "%" +
"\t# Wrestlers: " + str(i * stats[i]));
    print(".................\nTotal:\t" + str(total[0]) + "\t" + str(total[1]) + "%\t" + "#
Wrestlers: " + str(total[2]));
    return errors;

# Optimization Methods

# Given a list of wrestlers, creates groups of size n using the allowance.  returns the list of
groups.
def optimizeByWeightAllowance(wrestlers, n, allowance):
```

```
    groups = [];
    group = [];
    count = 0;
    maxweight = 0;
    for wrestler in wrestlers:
        if count == 0 or wrestler.start == True:
            if wrestler.start == True and group != []:
                groups.append(group);
                count = 0;
            maxweight = wrestler.weight * (1 + allowance);
            group = [wrestler];
            count += 1;
        elif wrestler.weight < maxweight and count < n:
            group.append(wrestler);
            count += 1;
        else:
            groups.append(group);
            group = [wrestler];
            maxweight = wrestler.weight * (1 + allowance);
            count = 1;
    groups.append(group);
    return groups;

# Performs the commands from the .ovr file on the wrestlers list.  These commands are "START"
"OVERRIDE" and "REMOVE"
def makePreAdjustments(wrestlers, txt):
    lines = txt.split("\n");
    for line in lines[1:len(lines)-1]:
        commands = line.split(":");
        if "START" in commands[0]:
            for wrestler in wrestlers:
                if wrestler.id == commands[1]:
                    wrestler.start = True;
        elif "OVERRIDE" in commands[0]:
            for wrestler in wrestlers:
                if wrestler.id == commands[1]:
                    wrestler.weight = float(commands[2]);
        elif "REMOVE" in commands[0]:
            wrestlerindex = 0;
            for i in range(0, len(wrestlers)):
                if wrestlers[i].id == commands[1]:
                    wrestlerindex = i;
            wrestlers.pop(wrestlerindex);
    return wrestlers;

# Performs the commands from the .ovr file on the wrestlers list.  These commands are "SWAP""
def makePostAdjustments(groups, txt):
    lines = txt.split("\n");
    for line in lines[1:len(lines)-1]:
        commands = line.split(":");
        if "SWAP" in commands[0]:
            groupindex = 0;
            group1 = 0;
            wrestler1 = 0;
            group2 = 0;
            wrestler2 = 0;
            for group in groups:
                wrestlerindex = 0;
                for wrestler in group:
                    if wrestler.id == commands[2]:
                        group1 = groupindex;
                        wrestler1 = wrestlerindex;
                    if wrestler.id == commands[1]:
                        group2 = groupindex;
                        wrestler2 = wrestlerindex;
```

```
                    wrestlerindex += 1;
                groupindex += 1;
            groups[group2][wrestler2], groups[group1][wrestler1] = groups[group1][wrestler1],
groups[group2][wrestler2]
    return groups;

# Execution Functions

# Writes RRGs to the /RRGs folder.  Also adds to error message if there are any errors
def createRRGs(groups, templates, errors):
    canrun = True;
    for group in groups:
        if len(group) == 1: canrun = False;
    index = 0;
    if canrun == True:
        os.system("cd RRGs/\nrm *");
        for group in groups:
            index += 1;
            if index < 10:
                printstr = "0" + str(index);
            else: printstr = str(index);
            temp = templates[len(group)];
            found = False;
            for i in range(len(group), 50):
                if templates[i] != 0:
                    if found == False and len(group) <= i:
                        temp = templates[i];
                        found = True;
            for i in range(0, len(group)):
                temp = temp.replace("WRESTLER" + str(i+1), group[len(group) - i - 1].abname);
            for i in range(0, 50):
                temp = temp.replace("WRESTLER" + str(i+1), "BYE (BYE)");
            strindex = str(index);
            temp = temp.replace("::", ":BR" + printstr + ":");
            if index < 10:
                strindex = "0" + str(index);
            fh = open("RRGs/BR" + strindex + ".txt", "w+");
            fh.write(temp);
            fh.close();
    else: errors += "\nRRGs were not created because there is a one man group";
    return errors;

# Executable Code

# Parameters for Tournament Execution.  Most are taken from the command line input/.cfg file.
bracketsmallmax is not in use.
tourneyname = sys.argv[1];
brackettype = "";
bracketsize = 0;
bracketallowance = float(sys.argv[2]);
writeRRGs = sys.argv[3];
bracketsmallmax = 0;
errors = "Errors:";
bracketadj = "";
enddata = "";
templates = [];
for i in range(0, 256):
    templates.append(0);

# Open the configuration file
configopened = True;
try:
    cfg = open(tourneyname + ".cfg", "r");
except:
    errors += "\ncannot open " + tourneyname + ".cfg";
```

```
    configopened = False;

# Sets the execution parameters for the tournament
if configopened == True:
    config = cfg.read();
    lines = config.split("\n");
    for line in range(0,len(lines) -1):
        param = lines[line].split(":");
        if param[0] == "brackettype": brackettype = param[1];
        elif param[0] == "bracketsize": bracketsize = int(param[1]);
        elif param[0] == "bracketsmallmax": bracketsmallmax = int(param[1]);
        elif param[0] == "usebracket":
            num = int(param[1][2:]);
            templatefile = open("templates/" + param[1] + ".rrg");
            template = templatefile.read();
            templates[num] = template;

if writeRRGs != "w": prwrite = "editting";
else: prwrite = "writing";
enddata += "Input > " + brackettype + " " + tourneyname + ": size: " + str(bracketsize) + ",
bracketallowance: " + str(bracketallowance *100) + "%, smallest bracket size: " +
str(bracketsmallmax) + ", action: " + prwrite;

# Open the .ovr file.  If it doesn't exist, the output will say it can't find the file but the
program will still run

try:
    file = open(tourneyname + ".ovr", "r");
    bracketadj = file.read();
    adjdata = bracketadj.split("\n",1)[1];
    enddata += "\nOverride file:\n" + adjdata[0:len(adjdata)-1];
except:
    errors += "\ncannot open " + tourneyname + ".ovr";
wrestlers = [];

op = [];

csvopened = True;

try:
    csv = open(tourneyname + ".csv");
except:
     errors += "\ncannot open " + tourneyname + ".csv";
     csvopened = False;

if csvopened == True:
    data = csv.read();
    lines = data.split("\n");
    index = 0;
    for line in range(0,len(lines) -1):
        index += 1;
        elems = lines[line].split(",");
        w = wrestler(str(index), elems[0], "", elems[1], float(elems[2]));
        #w = wrestler(str(index), elems[0], elems[1], elems[2], float(elems[3]));
        wrestlers.append(w);
    wrestlers = makePreAdjustments(wrestlers, bracketadj);
    wrestlers = sortByWeight(wrestlers);
    op = optimizeByWeightAllowance(wrestlers, bracketsize, bracketallowance);
    op = makePostAdjustments(op, bracketadj);
printGroups(op);
errors = printStats(op, errors);

print(enddata);
if writeRRGs == "w": errors = createRRGs(op, templates, errors);
if errors == "Errors:":
```

```
    errors += " None, okay to write to RRG file";
print(errors);
```