// Created by Chase Cummins, Big Event 2022: (520) 256-9739 for questions if needed, but please read the comments first.

/\*

This algorithm is used to match Operations Staff with partners. Please read this before trying to use it because it is specific

in what it needs to work. Assuming you are matching off the same parameters we did when creating it: sex (boys to girls or girls to girls), car (one car between them), schedule (some # of similar hours free in schedule), experience (not allowing two non-rookies to be together), and personality (extrovert, introvert, or moderate where 2 people of the same personality cannot be matched together excluding moderates), you should be able to just change the few constants below to the proper numbers and run it.

The algorithm WILL NOT WORK if you input an odd number of people because it is impossible to match each person with another if there is an odd number. We simply took a person with a lot of matches after personality matching (look in the logs) out, and manually matched them to a group of 3 after. This algorithm can be rerun as many times as you want if you don't like the matches output because it will generate a new combination every time (as long as you delete the output sheet before rerunning). We used a google form to collect the info from the Ops Staff and as long as the format doesn't change largely from what it was, it should work fine. There should be an example for you to use or copy off of in the folder this file is in.

I have commented "CHANGE" next to anything that you might need to change to make the algorithm work, assuming you are inputting good data and there are an even number of people inputted. Use CTRL + F to find them but they are all at the top of the program. If you have questions or something isn't working, it's probably your input, but my phone number is at the top if needed.

\*/

// These are all Zero-Indexed

const colSex = 3; // CHANGE to column # of sex (male, female, or other)

const colCar = 4; // CHANGE to column # of car (yes or no)

const colSched = 5; // CHANGE to column # of Monday schedule column (Assuming tues-fri are the columns right after)

// All schedules need to be formatted exactly the same or algorithm will say there are no matches

const colExp = 10; // CHANGE to column # of experience (rookie or non-rookie)

const colPerson = 11; // CHANGE to column # of personality (extrovert, introvert, or moderate)

var numHoursNeeded = 4; // CHANGE: Start high and lower until no eliminations: "Someone died: false" in Execution Log

// Algorithm will run forever if someone(people) are eliminated. We used 4 in original.

const ss = SpreadsheetApp.getActiveSpreadsheet(); // the spreadsheet

const sheet = ss.getSheetByName("Form Responses 1"); // sheet with all the people and their info CHANGE to sheet name

const table = sheet.getDataRange().getValues(); // 2D array containing the entire sheet including header row - Zero-indexed

// This holds all the possible matches and matches are deleted from here whenever they don't work

var matches = [];

// This can be used to automate reruns of the algorithm if less hours are needed to stop eliminations of people

// var copyOfScheduleMatches = [];

function myFunction() {

createPossibleMatches();

matchCars();

matchSchedules(numHoursNeeded);

matchExperience();

matchPersonality();

var toPrint = partnerMatch();

printResults(toPrint);

deleteDuplicates();

}

// Comment telling what it is doing

function deleteDuplicates() {

var sheet2 = ss.getSheetByName('Output');

let data = sheet2.getDataRange().getValues();

for (let j = 2; j < data.length; ++j) { // Comment why this is 2

for (let k = 2; k < data.length; ++k) { // Same as above^^

if (sheet2.getRange(k, 2).getValues().toString() == sheet2.getRange(j, 1).getValues().toString()) {

sheet2.deleteRow(k);

--data.length;

--k;

// Logger.log("Deleted row " + k + ", which had '" + data[k] + "'");

// Uncomment this^ if you want to track which duplicates are being deleted

}

}

}

}

// Comment telling what it is doing

function printResults(finalMatches) {

var newSheet = ss.insertSheet('Output');

var headers = newSheet.getRange('A1:B1');

var values = [

["Partner 1 Name", "Partner 2 Name"]]

headers.setValues(values);

headers.setFontWeight('bold');

let partner1Match = newSheet.getRange(2, 1, table.length, 1); // Explain the get range numbers or use a constant

let partner2Match = newSheet.getRange(2, 2, matches.length, 1);

let partner1Names = sheet.getRange(2, 2, table.length, 1).getValues();

partner1Match.setValues(partner1Names); // Print the name of everyone into column 1

var finalMatchNames = [];

var finalMatchesSubArrays = [];

for (let i = 0; i < finalMatches.length; ++i) {

let matchIndex = finalMatches[i];

let matchName = table[matchIndex][1]; // Why is this 1? Use a constant

finalMatchNames.push(matchName);

}

while (finalMatchNames.length > 0) {

var match = finalMatchNames.splice(0, 1);

finalMatchesSubArrays.push(match);

}

partner2Match.setValues(finalMatchesSubArrays);

newSheet.autoResizeColumns(1, 2); // Explain this too

}

// Comment telling what it is doing

function onOpen() {

let ui = SpreadsheetApp.getUi();

ui.createMenu('Fun Button')

.addItem('Match that shit', 'myFunction')

.addToUi();

}

/\*

After all matches have been reduced by match functions this final function will find one of many

possible combinations of partner to partner matches so that each person is matched to 1 and only 1 person.

It will either return an array of matches or will run forever if input isn't right.

\*/

function partnerMatch() {

/\*

This function randomly pick a person and one of their matches and continuously calls findPartner

until it finds a set that works for every person, which will vary on how many times it calls.

\*/

let totalCount = 0; // Total amount of runs needed to find a matching set

var finalMatches; // Stores the output of findParter and will be set to null if a matching set wasn't found

// or to an array of matches if matches are found

// This creates a mark array to store previously matched people

var mark = new Array(matches.length);

mark.fill(-1);

do {

mark.fill(-1); // Resets the mark array after every fun of findPartner DO NOT DELETE

//Randomize i and j for input

i = Math.floor(Math.random() \* matches.length);

j = Math.floor(Math.random() \* matches[i].length);

finalMatches = findPartner(i, j, mark);

// Counts the number of people unmatched after each run

let count = 0;

for (let i = 0; i < mark.length; ++i) {

if (mark[i] == -1) ++count;

}

totalCount++;

Logger.log("Matches short: " + count);

} while (finalMatches == null);

if (finalMatches != null) {

Logger.log("Runs needed to find match: " + totalCount);

Logger.log("Matches Found:");

Logger.log(finalMatches);

return finalMatches;

}

}

// A recursive function that matches a person to a partner and randomly picks the next person and partner to match

function findPartner(i, j, finalMatches) {

// Check for availability of match[i][j] in array

if (finalMatches[i] == -1 && finalMatches[matches[i][j] - 1] == -1) {

finalMatches[i] = matches[i][j];

// Subtract one because a person's matches aren't zero-indexed and add one because

// the indices of people in the table array are +1 because of the header row

finalMatches[matches[i][j] - 1] = i + 1;

}

else {

if (finalMatches.includes(-1)) { // Checks if there are unmatched people

if (finalMatches[i] != -1) { // Checks if i is matched already

i = findI(i, finalMatches); // Finds new i to match

// Finds a match for person i and if it can't returns all the way out to partnerMatch

if ((j = findJ(i, j, finalMatches)) == null) {

return null;

}

} else {

// Same as above if statement

if ((j = findJ(i, j, finalMatches)) == null) {

return null;

}

}

//Sets each partner to each other

finalMatches[i] = matches[i][j];

finalMatches[matches[i][j] - 1] = i + 1;

}

else {

return finalMatches;

}

}

// Picks new random i and j for input

i = Math.floor(Math.random() \* matches.length);

j = Math.floor(Math.random() \* matches[i].length);

return findPartner(i, j, finalMatches);

}

// Finds a new i for findPartner that isn't taken already

function findI(i, finalMatches) {

let mark = new Array(matches.length);

mark.fill(-1);

while (finalMatches[i] != -1) {

mark[i] = 1;

if (!mark.includes(-1)) {

return null;

}

i = Math.floor(Math.random() \* matches.length);

}

return i;

}

// Finds a new j for findPartner that isn't taken already

function findJ(i, j, finalMatches) {

let mark = new Array(matches[i].length);

mark.fill(-1);

while (finalMatches[matches[i][j] - 1] != -1) {

mark[j] = 1;

if (!mark.includes(-1)) {

return null;

}

j = Math.floor(Math.random() \* matches[i].length);

}

return j;

}

// Eliminates matches where people's personalities are the same, excluding moderates.

function matchPersonality() {

for (let i = 0; i < matches.length; ++i) { // Goes through every guy

for (let j = 0; j < matches[i].length; ++j) { // Goes through every girl in guy matches

// If both people are introverts/extroverts, then it removes their match so

// that two of the same personality are not together

if (table[i + 1][colPerson] == "Extroverted" && table[matches[i][j]][colPerson] == "Extroverted" ||

table[i + 1][colPerson] == "Introverted" && table[matches[i][j]][colPerson] == "Introverted") {

let temp = matches[i][j] - 1; // Subtract one because indices correspond to table array with header row

matches[i].splice(j, 1);

// This deletes the corresponding match so you don't have to check a matches compatibility twice

// Add one because i corresponds to matches indices, not the table array indices with header row

matches[temp].splice(matches[temp].indexOf(i + 1), 1);

// If you splice it removes the element and therefore when j increments in the loop

// it will have skipped the following element so we decrement it if we splice.

--j;

}

}

}

Logger.log("Matches after personality matching:");

let someoneDied = false;

// Prints out all the matches of each person after matching personality

// If someone has zero matches, it outputs that someone died

for (let i = 0; i < matches.length; ++i) {

Logger.log("Person " + (i + 1) + ": " + matches[i]);

if (matches[i].length == 0) someoneDied = true;

}

Logger.log("Someone died: " + someoneDied);

}

// Eliminates matches where two non-rookies are matched together

function matchExperience() {

for (let i = 0; i < matches.length; ++i) { //Goes through every guy

for (let j = 0; j < matches[i].length; ++j) { // Goes through every girl in guy matches

// If both people have been on Ops Staff before, then it removes their match so

// that two non-rookies won't be together

if (table[i + 1][colExp] == "Yes" && table[matches[i][j]][colExp] == "Yes") {

let temp = matches[i][j] - 1; // Subtract one because indices correspond to table array with header row

matches[i].splice(j, 1);

// This deletes the corresponding match so you don't have to check a matches compatibility twice

// Add one because i corresponds to matches indices, not the table array indices with header row

matches[temp].splice(matches[temp].indexOf(i + 1), 1);

// If you splice it removes the element and therefore when j increments in the loop

// it will have skipped the following element so we decrement it if we splice.

--j;

}

}

}

Logger.log("Matches after experience matching:");

Logger.log(matches);

}

// Eliminates matches where a match doesn't have the necessary number of matching available hours

function matchSchedules(numHoursNeeded) {

const daysInWeek = 5;

// Loops through every person

for (let i = 0; i < matches.length; ++i) {

// Loops through each match

for (let j = 0; j < matches[i].length; ++j) {

var hoursMatched = 0;

// Loops through days in week

for (let numDay = 0; numDay < daysInWeek; ++numDay) {

// Takes schedule and splits it into an array of strings

var firstSchedule = table[i + 1][colSched + numDay].split(", ");

var secondSchedule = table[matches[i][j]][colSched + numDay].split(", ");

// Checks first schedule vs second schedule

for (let l = 0; l < firstSchedule.length; ++l) {

for (let m = 0; m < secondSchedule.length; ++m) {

if (firstSchedule[l] == secondSchedule[m]) {

++hoursMatched;

break;

}

}

}

}

// Splices, which will skip element unless you decrement it to check the same index again

if (hoursMatched < numHoursNeeded) {

let temp = matches[i][j] - 1; // Subtract one because indices correspond to table array with header row

matches[i].splice(j, 1);

// This deletes the corresponding match so you don't have to check a match's compatibility twice

// Add one because i corresponds to matches indices, not the table array indices with header row

matches[temp].splice(matches[temp].indexOf(i + 1), 1);

// It will skip an element if you do not decrement after splicing

// since it removes the element from the array

--j;

}

}

}

Logger.log("Matches after schedule matching:");

Logger.log(matches);

}

// Eliminates matches where neither of them have a car

function matchCars() {

for (let i = 0; i < matches.length; ++i) { //Goes through every person

for (let j = 0; j < matches[i].length; ++j) { // Goes through every person's matches

if (table[i + 1][colCar] != "Yes" && table[matches[i][j]][colCar] != "Yes") { // Add one because table array has header

let temp = matches[i][j] - 1; // Subtract one because indices correspond to table array with header row

matches[i].splice(j, 1);

// This deletes the corresponding match so you don't have to check a matches compatibility twice

// Add one because i corresponds to matches indices, not the table array indices with header row

matches[temp].splice(matches[temp].indexOf(i + 1), 1);

// If you splice it removes the element and therefore when j increments in the loop

// it will have skipped the following element so we decrement it if we splice.

--j;

}

}

}

Logger.log("Matches after matching cars:")

Logger.log(matches);

}

// This function generates all possible matches for each person matching every man to every female

// and every female to every other person besides themselves; "others" are treated as female.

function createPossibleMatches() {

var notMen = []; // Stores indices of everyone that isn't a male

var notMenCount = 0;

var allPeople = []; // Stores indices of everyone

var allPeopleCount = 0;

for (let i = 1; i < table.length; ++i) { // Starts at 1 to ignore header row

// Creates an array of all that are not men

if (table[i][colSex] != "Male") {

notMen[notMenCount++] = i;

}

allPeople[allPeopleCount++] = i; // Creates array of all people

}

Logger.log("Not men:");

Logger.log(notMen);

Logger.log("All:");

Logger.log(allPeople);

let index = 0;

for (let i = 1; i < table.length; ++i) {

if (table[i][colSex] == "Male") {

matches[index] = notMen.slice(); // This will put all non-men as possible matches to the men

}

else {

matches[index] = allPeople.slice(); // This will put all people as possible matches for non-men

// We have to delete each person's match to themselves

matches[index].splice(matches[index].indexOf(index + 1), 1);

}

++index;

}

Logger.log("Initial Matches:");

Logger.log(matches);

}