AI Prompts:

* Hello, I am currently starting a new coding project where I am building a web application using VSCode on mac. Essentially, right now I want to have some sort of form that has 6 different questions with a dropdown menu to answer each question. For the dropdown menus on the questions, I want to give them the functionality such that when one dropdown option is chosen on one of the questions, the chosen option disappears from the drop down menus on any of the other questions. In total the form should start with 13 dropdown options and then once you finish the form there should be 7 options left. How can I implement this?
* Hi, I am currently starting a project where I am going to create a spreadsheet where the user will enter in 6 different choices (in 6 different rows) and then press a button (or it just does it automatically) that will automatically populate a table to the right of this input area. The population of the table to the right will be very simple. Whatever the user entered in for the choice will simply be copy and pasted to a designated cell on the right table. How can I make this happen with excel? Will I need to use VBA?

Hi, I am starting a new Microsoft Excel VBA project that I would like you to provide the VBA code for. The purpose of my project is to create a spreadsheet/program that will calculate what NCAA College Ski Racing teams get to set the courses for each of the races during the NCAA skiing season.

To start out, I will explain to you the rules of the NCAA sets and give you information about the NCAA ski teams and ski season. In a NCAA ski season there are 6 races, and the races for the NCAA are called “Carnivals”, hence “Carnival 1”, “Carnival 2”, …, “Carnival 6”. For each of the 6 different races during the season, there will be a unique host team of that race. In total, there are 13 NCAA ski teams: “UVM”, “CBC”, “HAR”, “DAR”, “WIL”, “MID”, “SLU”, “BAT”, “SMC”, “UNH”, “CSC”, “BC”, and “PSU” (like I said above, only 6 of these schools will host a race per season). At each carnival there are 6 different sets, or times when a specific college ski team will be assigned to set the racecourse. These six different sets for each race are “GS 1”, “GS 2”, “MSL 1”, “WSL 1”, “MSL 2”, “WSL 2”. In each NCAA ski season, each ski team will be assigned with either 3 or 2 sets, for the whole season. For example, “UVM” gets 3 sets per season while “BC” only gets 2. The entire list of the teams and how many sets they get per season can be observed: “UVM” : 3, “CBC” : 3, “HAR” : 3, “DAR” : 3, “WIL” : 3, “MID” : 3, “SLU” : 3, “BAT” : 3, “SMC” : 3, “UNH” : 3, “CSC” : 2, “BC” : 2, and “PSU” : 2. In total, since there are six races for the NCAA ski season, and there are six different sets per race, there are a total of 36 course sets per season.

If a specific ski team is a host for a race, they will automatically be given the slot to set the “GS 2” and either “MSL 1” or “WSL 1” (alternating each race) of their hosted “Carnival”. Since there are 6 total races, and during each race 2 of the 6 total sets are taken by the hosted team, that means that after the hosts are designated their “automatic” sets, being 2 sets \* 6 races = 12 sets, there will be 36 – 12 = 24 open sets left. Note that since each host team will automatically set 2 times for their hosted race, If a team such as “BC”, with only 2 sets per season, is a host, then they will extinguish all their sets for the season during their hosted “Carnival” and if a team such as “UVM”, with 3 sets per season, is a host, then they will only have one more set left for the season.

As can be observed above, each host for each “Carnival” will be given two slots to set at their hosted “Carnival” – these sets being “GS 2” and either “MSL 1” or “WSL 1” (alternating each race). After each host team is designated their “automatic” home sets, there will be 24 open sets left for the season. These last 24 open sets for the season will be randomly filled by teams that have sets “left”. For example, let’s say that “UVM” and “BC” are “Carnival” hosts, and “SMC” is not. After the hosting schools are given their “automatic” home sets, “UVM” will have 1 set left, and “BC” will have 0 sets left since “UVM” and “BC” are both host schools, and “SMC” will still have 3 sets left since it isn’t a host school. From here on out, the last 24 open sets for the season will be randomly filled by teams that have sets “left” until all the open sets are filled, and all of the teams have run out of sets, which should occur at the same time.

So far, I have some implementation ideas for this project and have started writing some code. First, I have created a “Hosts” table spanning cells E2 to F8. In the column E3 to E8 I have “Carnival 1” through “Carnival 6”. In the column F3 to F8, the user will type in the corresponding hosts for each carnival. Underneath the “Hosts” table there is a Button labeled “Execute” which will be connected to some VBA code.

The way the spreadsheet/VBA should be set up is as follows:

Hello, I would like to tell you about a “Game” that I have, and I would like you to solve it.

In my game there are “teams”, “picks”, “moves”, and a “grid”.

There are a total of 13 teams. 10 of the teams have 3 moves and 3 of the teams have 2 moves. In total there are 36 moves.

There is a 6X6 grid with a total of 36 grid spaces.

There are 6 picks at the start of the game. Each pick must be for a team and must be of a team that hasn’t gotten picked already.

Whatever teams get picked, they get their name posted to 2 specific spots on the grid, and they get two moves taken away. Once a team gets all of its moves taken away, it cannot put its name down on the grid anymore.

Pick #1 (team) get’s it’s name posted to row 1 at grid spots (1,2) and (1,3).

Pick #2 (team) get’s it’s name posted to row 2 at grid spots (2,2) and (2,4).

Pick #3 (team) get’s it’s name posted to row 3 at grid spots (3,2) and (3,3).

Pick #4 (team) get’s it’s name posted to row 4 at grid spots (4,2) and (4,4).

Pick #5 (team) get’s it’s name posted to row 5 at grid spots (5,2) and (5,3).

Pick #6 (team) get’s it’s name posted to row 6 at grid spots (6,2) and (6,4).

After the initial picks are made, and the picked team’s get their names posted to the grid, there should be 12 filled grid spaces, and 24 open grid spaces left. This means that for the teams, in total there should be 24 moves left. If a team that only had two moves was one of the teams that happened to get picked, then it doesn’t have any moves anymore (since both of it’s moves were taken away) and it cannot put its name down on the grid anymore.

Find a way to extinguish all the rest of the moves that the teams have, by placing their names on the table, without violating any of the rules.

Key rules:

1. A “picked” team (one of the 6 teams that get picked at the start of the game), cannot have its name placed anywhere in the row directly below where its name was “automatically” placed. For example, pick #1 (team) automatically gets its name posted to grid spot (1,2) and (1,3), so it now cannot have its name posted to anywhere on the 2nd row. Similarly, Pick #4 (team) automatically gets its name posted to grid spot (4,2) and (4,3), so it now cannot have its name posted to anywhere on the 5th row.
2. After a team get its name posted to the grid, for the rest of the game its name cannot go in the same column that its name was just posted to.

Winning the game is having the grid completely filled (legally) with all the teams having all their moves extinguished.

Can you write me a python program that tests if this game is theoretically possible to be solved? Please have the python program print out the grid so I can get a visual of what the solved or unsolved grid looks like.

.

.

.

This looks great, thank you. Now, can you modify this code such that it prints out 5 unique solutions for solving the grid given the same starting picks? I want it to print out a solution each time that it has one and if it is not able to get 5 total solutions, that is fine, and I want it to time out at a certain point (say after 15 seconds or so).

.

.

.

It looks like that based on that output that you gave me, these aren’t 5 unique solutions, but instead just 5 of the same solutions. Can you rewrite the code such that it checks and makes sure the solution hasn’t already been found?

.

.

.

I think that you are missing some parts from the initial code. For example, GRID\_SIZE is never defined on the new code that you gave me. Can you make sure that it has all of its necessary parts?

.

.

.

*# Set up initial grid with picks*

picked\_teams = list(teams.keys())[:6] *# First 6 teams picked*

In this line of code it picks the first six teams, now I want to pick teams from team 6 to team 11. How can I do this?

.

.

.

Now, can you change this Javascript code such that the the program will take user input at the beginning for the 6 teams that are picked. The user type in the first 6 teams that will be “picked”. Similar to how the original code is written, the options will be “Team\_1”, “Team\_2”, “Team\_3”, …, “Team\_13”.

.

.

.

I want to make a few tweaks to this function:

*// Function to try filling grid using remaining moves*

function fillGrid(remainingTeams) {

if (remainingTeams.length === 0) {

return true; *// All moves completed successfully*

}

const team = remainingTeams[0];

let movesLeft = teams[team];

*// Try placing remaining moves for current team*

for (let row = 0; row < GRID\_SIZE; row++) {

for (let col = 0; col < GRID\_SIZE; col++) {

if (movesLeft > 0 && isValidMove(team, row, col)) {

grid[row][col] = team;

teams[team] -= 1; *// Deduct move*

if (fillGrid(teams[team] > 0 ? remainingTeams : remainingTeams.slice(1))) {

return true;

}

*// Undo move if unsuccessful*

grid[row][col] = ' ';

teams[team] += 1;

}

}

}

return false;

}

Here I see that each time a new “team” is picked and an attempt is made to put that team into the grid, the neam team that is picked is just taken from the beginning of the remainingTeams array. Hence: “const team = remainingTeams[0];”. Now I want to change this code so it randomly picks a new team to try out of the remainingTeams array each time. Can you change the code to do this?

.

.

.

Thank you – that function performed well. However, I just noticed that the row exclusion for the initial pick rule is being violated in my results. For example, in my results this time “SMC” was picked as the first pick, but for some reason, on the second row I notice that “SMC” is posted in spot (6,2). How can I stop this from happening? Here is all the code I have right now:  
  
const readline = require('readline');

*// Constants*

const GRID\_SIZE = 6;

const TOTAL\_TEAMS = 13;

*// Define grid*

let grid = Array.from({ length: GRID\_SIZE }, () => Array(GRID\_SIZE).fill(' '));

*// Define teams and moves*

let teams = {};

teams['UVM'] = 3;

teams['CBC'] = 3;

teams['HAR'] = 3;

teams['DAR'] = 3;

teams['WIL'] = 3;

teams['MID'] = 3;

teams['SLU'] = 3;

teams['BAT'] = 3;

teams['SMC'] = 3;

teams['UNH'] = 3;

teams['CSC'] = 2;

teams['BC'] = 2;

teams['PSU'] = 2;

*// for (let i = 0; i < TOTAL\_TEAMS; i++) {*

*// teams[`Team\_${i + 1}`] = i < 10 ? 3 : 2; // First 10 teams with 3 moves, last 3 teams with 2 moves*

*// }*

*// Create interface for input*

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

*// Function to get user input for picked teams*

function getPickedTeams(callback) {

let pickedTeams = [];

const askTeam = (i) => {

if (i < 6) {

rl.question(`Enter the name of picked team ${i + 1} (e.g., UVM, SMC,..., DAR): `, (team) => {

if (!teams.hasOwnProperty(team) || pickedTeams.includes(team)) {

console.log('Invalid team. Please enter a unique team name.');

askTeam(i); *// Ask again*

} else {

pickedTeams.push(team);

askTeam(i + 1); *// Move to next team*

}

});

} else {

callback(pickedTeams);

}

};

askTeam(0);

}

*// Initial picks and their corresponding positions*

const initialPicksPositions = [

[[1, 2], [1, 3]],

[[2, 2], [2, 4]],

[[3, 2], [3, 3]],

[[4, 2], [4, 4]],

[[5, 2], [5, 3]],

[[6, 2], [6, 4]]

];

*// Get picked teams from user input*

getPickedTeams(pickedTeams => {

*// Set initial picks on grid and deduct moves*

pickedTeams.forEach((team, idx) => {

teams[team] -= 2; *// Deduct initial 2 moves for picked teams*

initialPicksPositions[idx].forEach(pos => {

const [row, col] = [pos[0] - 1, pos[1] - 1]; *// Convert to 0-indexed for grid*

grid[row][col] = team;

});

});

*// Constraints: row exclusion for each initial pick*

let rowExclusions = {};

pickedTeams.forEach((team, idx) => {

rowExclusions[team] = idx + 1;

});

*// Function to print grid*

function printGrid(grid) {

grid.forEach(row => console.log(row.join(" ")));

console.log();

}

*// Helper to check if a move is valid*

function isValidMove(team, row, col) {

*// Rule 1: Row exclusion after the initial pick row*

if (rowExclusions[team] === row + 1) {

return false;

}

*// Rule 2: Column exclusion for already occupied columns*

for (let r = 0; r < GRID\_SIZE; r++) {

if (grid[r][col] === team) {

return false;

}

}

return grid[row][col] === ' ';

}

*// // Function to try filling grid using remaining moves*

*// function fillGrid(remainingTeams) {*

*// if (remainingTeams.length === 0) {*

*// return true; // All moves completed successfully*

*// }*

*// const team = remainingTeams[0];*

*// let movesLeft = teams[team];*

*// // Try placing remaining moves for current team*

*// for (let row = 0; row < GRID\_SIZE; row++) {*

*// for (let col = 0; col < GRID\_SIZE; col++) {*

*// if (movesLeft > 0 && isValidMove(team, row, col)) {*

*// grid[row][col] = team;*

*// teams[team] -= 1; // Deduct move*

*// if (fillGrid(teams[team] > 0 ? remainingTeams : remainingTeams.slice(1))) {*

*// return true;*

*// }*

*// // Undo move if unsuccessful*

*// grid[row][col] = ' ';*

*// teams[team] += 1;*

*// }*

*// }*

*// }*

*// return false;*

*// }*

*// Function to try filling grid using remaining moves*

function fillGrid(remainingTeams) {

if (remainingTeams.length === 0) {

return true; *// All moves completed successfully*

}

*// Randomly select a team from remainingTeams*

const randomIndex = Math.floor(Math.random() \* remainingTeams.length);

const team = remainingTeams[randomIndex];

let movesLeft = teams[team];

*// Try placing remaining moves for current team*

for (let row = 0; row < GRID\_SIZE; row++) {

for (let col = 0; col < GRID\_SIZE; col++) {

if (movesLeft > 0 && isValidMove(team, row, col)) {

grid[row][col] = team;

teams[team] -= 1; *// Deduct move*

*// Create a new array excluding the current team if its moves are exhausted*

const newRemainingTeams = teams[team] > 0 ? remainingTeams : remainingTeams.filter(t => t !== team);

if (fillGrid(newRemainingTeams)) {

return true;

}

*// Undo move if unsuccessful*

grid[row][col] = ' ';

teams[team] += 1;

}

}

}

return false;

}

*// Solve and print result*

console.log("Initial Grid after picks:");

printGrid(grid);

*// Start solving from remaining teams*

let remainingTeams = Object.keys(teams).filter(team => teams[team] > 0);

let solutionFound = fillGrid(remainingTeams);

console.log(solutionFound ? "Final Grid:" : "No solution found. Partial grid:");

printGrid(grid);

*// Close readline interface*

rl.close();

});

.

.

.

Now, this is going to be the biggest next step in my project. The reason I have created this code above is to eventually move this Javascript code into Google App Script and connect it with a google sheet. The ultimate idea for my Google Sheet is to have a small input box where I type the 6 picked team names, and then an “execute” button that takes those 6 picked team names in from the sheet, runs the script on the team names, and then outputs the final grid in a 6X6 sheet table. Can you make this for me?

.

.

.

Here is some Google AppsScript code that I am using for a Google sheet of mine.

function onOpen() {

const ui = SpreadsheetApp.getUi();

ui.createMenu('Pick Sets')

.addItem('Execute', 'runTeamPicker')

.addToUi();

}

function runTeamPicker() {

const sheet = SpreadsheetApp.getActiveSpreadsheet().getActiveSheet();

// Read picked teams from the sheet (cells C3 to C8)

const pickedTeams = [];

for (let i = 3; i <= 8; i++) {

const team = sheet.getRange(`C${i}`).getValue().trim();

if (team) {

pickedTeams.push(team);

}

}

// If less than 6 teams are picked, alert the user

if (pickedTeams.length < 6) {

SpreadsheetApp.getUi().alert('Please enter 6 unique teams.');

return;

}

// Constants

const GRID\_SIZE = 6;

const TOTAL\_TEAMS = 13;

// Define grid

let grid = Array.from({ length: GRID\_SIZE }, () => Array(GRID\_SIZE).fill(' '));

// Define teams and moves

let teams = {

'UVM': 3, 'CBC': 3, 'HAR': 3, 'DAR': 3, 'WIL': 3,

'MID': 3, 'SLU': 3, 'BAT': 3, 'SMC': 3, 'UNH': 3,

'CSC': 2, 'BC': 2, 'PSU': 2

};

// Set initial picks on grid and deduct moves

pickedTeams.forEach((team, idx) => {

teams[team] -= 2; // Deduct initial 2 moves for picked teams

const initialPicksPositions = [

[[1, 2], [1, 3]], [[2, 2], [2, 4]], [[3, 2], [3, 3]],

[[4, 2], [4, 4]], [[5, 2], [5, 3]], [[6, 2], [6, 4]]

];

initialPicksPositions[idx].forEach(pos => {

const [row, col] = [pos[0] - 1, pos[1] - 1]; // Convert to 0-indexed for grid

grid[row][col] = team;

});

});

// Constraints: row exclusion for each initial pick

let rowExclusions = {};

pickedTeams.forEach((team, idx) => {

rowExclusions[team] = idx + 1;

});

// Helper to check if a move is valid

function isValidMove(team, row, col) {

// Rule 1: Row exclusion after the initial pick row

if (rowExclusions[team] === row + 1) {

return false;

}

// Rule 2: Column exclusion for already occupied columns

for (let r = 0; r < GRID\_SIZE; r++) {

if (grid[r][col] === team) {

return false;

}

}

return grid[row][col] === ' ';

}

// Function to try filling grid using remaining moves

function fillGrid(remainingTeams) {

if (remainingTeams.length === 0) {

return true; // All moves completed successfully

}

// Randomly select a team from remainingTeams

const randomIndex = Math.floor(Math.random() \* remainingTeams.length);

const team = remainingTeams[randomIndex];

let movesLeft = teams[team];

// Try placing remaining moves for current team

for (let row = 0; row < GRID\_SIZE; row++) {

for (let col = 0; col < GRID\_SIZE; col++) {

if (movesLeft > 0 && isValidMove(team, row, col)) {

grid[row][col] = team;

teams[team] -= 1; // Deduct move

// Create a new array excluding the current team if its moves are exhausted

const newRemainingTeams = teams[team] > 0 ? remainingTeams : remainingTeams.filter(t => t !== team);

if (fillGrid(newRemainingTeams)) {

return true;

}

// Undo move if unsuccessful

grid[row][col] = ' ';

teams[team] += 1;

}

}

}

return false;

}

// Solve and print result

let remainingTeams = Object.keys(teams).filter(team => teams[team] > 0);

let solutionFound = fillGrid(remainingTeams);

// Output final grid to the sheet (starting from B1 E3)

for (let row = 0; row < GRID\_SIZE; row++) {

for (let col = 0; col < GRID\_SIZE; col++) {

sheet.getRange(row + 3, col + 6).setValue(grid[row][col]); // Columns B to G

}

}

if (!solutionFound) {

SpreadsheetApp.getUi().alert("No solution found. Partial grid displayed.");

} else {

SpreadsheetApp.getUi().alert("Final grid has been generated.");

}

}

How can I add functionality to this code such that if two of the same team names are taken in from the Sheet into the program, the program stops running and prints out an error handling message to the google sheet user?

Hello, here is the google AppScript code that I am using to execute a function in my Google sheet:

I now want to add more functionality to the section where initial picks are made:

*// Set initial picks on grid and deduct moves*

pickedTeams.forEach((team, idx) => {

teams[team] -= 2; *// Deduct initial 2 moves for picked teams*

const initialPicksPositions = [

[[1, 2], [1, 3]], [[2, 2], [2, 4]], [[3, 2], [3, 3]],

[[4, 2], [4, 4]], [[5, 2], [5, 3]], [[6, 2], [6, 4]]

];

initialPicksPositions[idx].forEach(pos => {

const [row, col] = [pos[0] - 1, pos[1] - 1]; *// Convert to 0-indexed for grid*

grid[row][col] = team;

});

});

I want to change the functionality here so that every time the script is ran, a random “heads or tails” coin flip (or something similar) is made. If heads, then the initial pick position will be:

const initialPicksPositions = [

[[1, 2], [1, 3]], [[2, 2], [2, 4]], [[3, 2], [3, 3]],

[[4, 2], [4, 4]], [[5, 2], [5, 3]], [[6, 2], [6, 4]]

];

And if tails, the intialPicksPositions will be:

const initialPicksPositions = [

[[1, 2], [1, 4]], [[2, 2], [2, 3]], [[3, 2], [3, 4]],

[[4, 2], [4, 3]], [[5, 2], [5, 4]], [[6, 2], [6, 3]]

];

Can you change the code for me and make this happen? Also the random “heads or tail” generater doesn’t actually have to be heads or tails – do whatever is best practice. It cann bee between true or false, 0 or 1 etc. Thank you.