Criteria C: Development

Basic Description:
The goal of this project was to create a website for my father for him to easily input and view player statistics for the basketball teams he manages, that could also store data between visits.

Explanation of Solution:
I decided to create a web application using the google interface to connect the website to a google spreadsheet on the backend, since it was an easy way to store table data, and would automatically retain the information given. Each player's data is stored in a singular sheet, which contains a table of their games on the y-axis, and their statistics for those games on the x-axis, with season totals and averages on the bottom two rows.

The program uses both HTML, for user interfacing forms and the shown stat table, and JavaScript, to communicate with the base spreadsheet and alter inputs/outputs as needed for readability and usefulness. The website is public, and can be accessed by anyone with the link, though I don't plan to share it with the public yet due to lacking some security.

Techniques used in project:
Below are examples of the techniques used in this project, as well as explanations of their function in the overall program:

- Dynamic sheet communication and alteration
- Dynamic array and calculation creation
- Array iteration and methodical alteration
- Dynamic html form alteration and creation
- Nested loops with arrays for table creation
- Parameter control and cross-file communication

The following paragraphs detail these techniques, and the context they were used in:
Dynamic sheet communication and alteration, Dynamic array calculation and creation:

```
function processForm(coord, val, gamers, namers, player){
  var sheet=ss.getSheetByName(''+player+'')
    if (!sheet) {
    Logger.log('Sheet ' + player + ' not found');
    return;
  }
  if (gamers==999){
    var added=(sheet.getLastRow()-2)
    sheet.insertRowAfter(added)
    if (sheet.getRange('A3').getValue()=="vs example (adding a game will delete this)"){
      sheet.deleteRow(2)
    }
    gamers=added+1;
    sheetcalcs=[[namers,0,0,'=B'+gamers+'/C'+gamers+'',0, 0, '=E'+gamers+'/F'+gamers+'',0,0,'=H'+gamers+'/I'+gamers+'','=(B'+gamers+'*2)+E'
    +gamers+'+(H'+gamers+'*3)', '=K'+gamers+'/(2*(C'+gamers+'+(0.44*F'+gamers+')))',0,0,0,0,0,'=N'+gamers+'/Q'+gamers+'']]
    sheet.getRange('A'+gamers+':R'+gamers+'').setValues(sheetcalcs)
  }
  trueplace=coord+gamers
  sheet.getRange(''+trueplace+'').setValue(val);
  sheet.getRange("A1").setValue(player)
  return(getDetail(player))
}
```

This function, processForm, is meant to alter the base spreadsheet in response to the user. It is run after the user's input form for a game's stats are submitted, and is given five parameters: coord (the inputted column/stat), val (the inputted value of the stat), gamers (the inputted game), namers (the name of a new game, if necessary), and player (the previously inputted player).

After communicating with the base google sheet of the web app to find the specific player sheet, and making sure the sheet exists, the function then checks if a new game was inputted, which is true if the value of gamers is 999, rather than a pre-existing row name. If there is a new game, the function adds a new sheet row two below the bottom (the two lowest being whole season calculations), deletes the example row if it still exists, and renames the game to the new name. Then, still within the if statement, the function embeds the sheet's calculations present in every row, dynamically altering them based on the row. The calculations are used to find percentages and other harder calculations and present them without user complications.

Finally, the function locates the stat alteration inputted by the user by concatenating the column and row submitted (stat and game), makes the changes in the sheet, and quickly makes sure the player's name is in the top left corner, before returning the new sheet composition through getDetail, which is returned to make the new user-seen table.

Array iteration and methodical alteration:

```
function getDetail(name) {
  var sheet=ss.getSheetByName(''+name+'')
  if (sheet==null){
    Logger.log('null sheet')
    dupeSheet(name)
  }
  var sheet=ss.getSheetByName(''+name+'')
  var data= sheet.getRange('A1:R'+sheet.getLastRow()+'').getValues();
  for (let i=0; i<data.length; i++){
    for (let j=0; j<data[i].length; j++){
      if (typeof data[i][j]==='number'){
        if (j==3 || j==6 || j==9 || j==11){
          data[i][j]=Math.round(data[i][j] * 1000) / 10+"%"
        } else if (j==17){
          data[i][j]=Math.round(data[i][j]*10)/10
        } else if (i==data.length-1){
        data[i][j]=Math.round(data[i][j]*100)/100
        }
      }
      if (typeof data[i][j] === 'string' && data[i][j].substring(0,1)==="#"){
        data[i][j]="N/A"
      }
    }
  }
  Logger.log([data,name])
  return([data,name])
}
```

This function, getDetail, is meant to find and shape the sheet data to give to the html table builder. After finding the sheet based on its player parameter, it first checks if the sheet exists to create a new sheet through the dupeSheet function, which separately creates and names the sheet. Then, the function reads all of the selected sheet's data, converts it into an array, and then alters it for readability.

To alter it, a for loop iterates through the array, first checking the items are numbers that are fractions (number mod 1 !=0) and altering any percentage fractions (rows 3, 6, 9, 11), to be a percentage to the nearest hundredth. It then checks specifically row 17 (the assist to turnover ratio) to alter to a non percentage to the nearest hundredth. Lastly for numbers, the function makes sure the last row, which is always the season averages, rounds to the nearest hundredth. Finally, the iteration also checks for any items starting with a #, which tend to be math errors in sheets (like dividing by zero), to change them to "N/A" for a better viewing experience.

Finally the function returns the newly altered array to be used in creating the user-seen table.

Dynamic html form alteration and creation:

```
function gameMaker(response){
  console.log(response);
  var newform=document.getElementById('gamers')
  var games=response
  console.log(games)
   while (newform.length > 1) {
     newform.remove(1);
     }
  for (let i=0; i<(games.length-3); i++){
    var gameoption=document.createElement('option')
    gameoption.textContent=games[i]
    gameoption.value=i+2
    newform.appendChild(gameoption);
  }
}
```

This function, gameMaker, which runs inside a script in the html section, populates a user form based on the response of a player's y-axis, or their games. The function starts by selecting a pre-existing form element and then clears all options inside of it except the first, which is always the selection for a new game. This is to prevent duplication as the most recent game information is added.

This addition occurs next, where a for loop iterates through each given game in the array and creates a new form option for it. Some weird math has to occur in the process to leave out the top (header) row, and the bottom two (season synthesis) rows.

In the end, the form is dynamically filled to show a proper selection of possible games for the user to select and potentially alter.

Nested loops with arrays for table creation:

```
function buildTable(response) {
  console.log('works?'+response)
  var interior=document.getElementById("output")
  interior.replaceChildren()
  for (let i=0; i<response[0].length; i++){
    let row = document.createElement("tr");
    for (let j=0; j<response[0][i].length; j++){
      let col = document.createElement("td");
      col.innerHTML=response[0][i][j]
      row.appendChild(col);
    }
    interior.appendChild(row);
  }
  getHeaders(response[1])
}
```

This function, buildTable, also exists in a script in html, and is given a technically three dimensional array, though the first dimension is only used to maintain the active player sheet to be given somewhere else.

The other two dimensions are based on sheets data from getDetail, and are used to construct the table that the viewer sees by selecting an html table with the id “output”, clearing it to prevent duplication from repeated calls, and then filling it based on the array.

By using a nested for loop, the function creates a row for each row in the second dimension, and iterates through creating sections for each column in the third dimension, ending with a full table showing the optimal data to the user.

Parameter control and cross-file communication:

```
function loadPage(event, formObject){
  event.preventDefault();
  var ourguy=formObject.players.value
  console.log(ourguy)
  if (ourguy==998){
    ourplayer=formObject.namey.value
  }else{
    ourplayer=ourguy
  }
  console.log(ourplayer+'works?')
  google.script.run.withSuccessHandler(buildTable).getDetail(ourplayer)
  getHeaders(ourplayer);
  login()
  console.log('works?')
}
```

This function is run after the player sheet is chosen by the user, and uses the form response to both set the global variable “ourplayer” to either the chosen pre-existing player or a newly added player’s name based on the formObject given. It then calls the attached js file in the web app program to find the player’s data (which also creates the new player’s sheet if necessary), runs the function to create the user table based on that data, and finally reruns the login function to recreate the player form to support any potential changes.

Helpful sources for project:
https://developers.google.com/apps-script/reference/spreadsheet/spreadsheet-Google Sheets web app documentation of functions
https://developers.google.com/apps-script/reference/html-More general web app API documentation
https://chatgpt.com/-**Only used** to create project CSS display at the end of the project, **not used** for any HTML/Javascript programs