OpenDSA Export Script for Google Spreadsheets

Initial Set Up

A. Given Access to a Google Spreadsheet

This is the quickest and easiest way to get started. Once given access to a workbook through google that already has the script on it. You can just start working on it with the spreadsheet already formatted to the script's needs.

B. Copy an existing Google Sheet

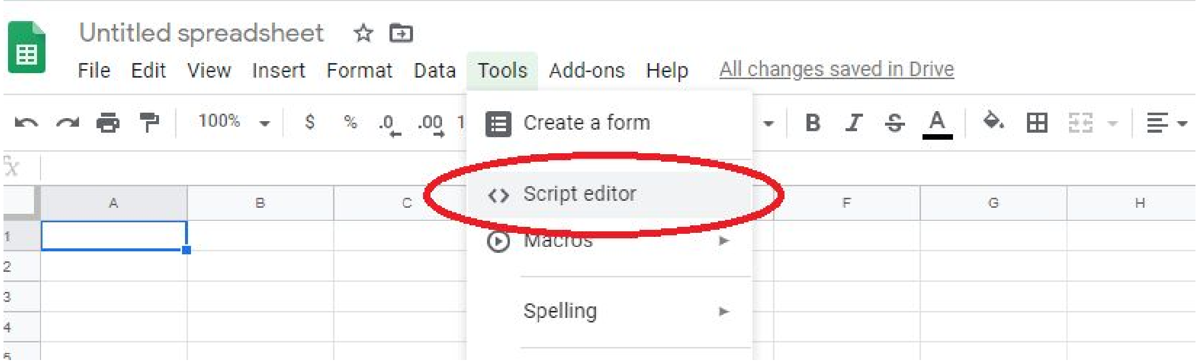
In your Google Account Drive copy an existing Google Sheet with the script on it. This will carry over to the new Google Sheet

C. Creating a new spreadsheet

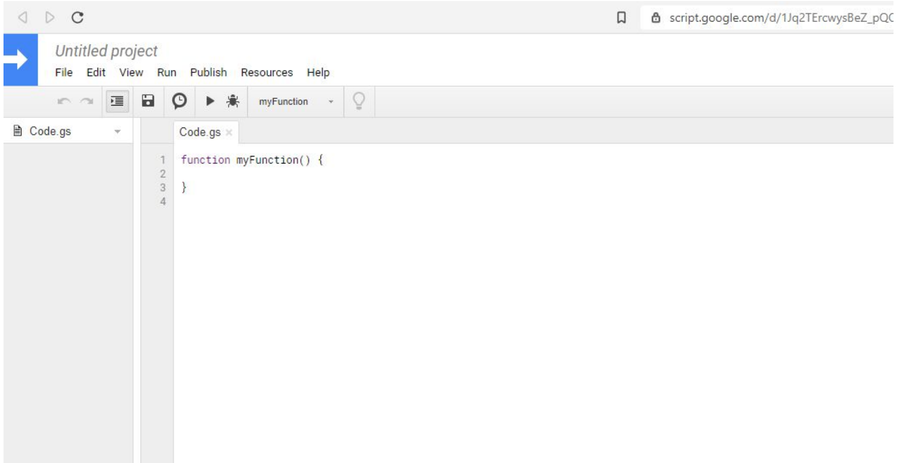
1. Create a new Google Sheet in Drive using a Google Account.
2. Open Script Editor:

   Going to Tools from main top menu

   Select Script editor from drop down menu


Untitled spreadsheet
File Edit View Insert Format Data Tools Add-ons Help
All changes saved in Drive
100%
Create a form
Script editor
Macros
Spelling
A
B
C
F
G
H

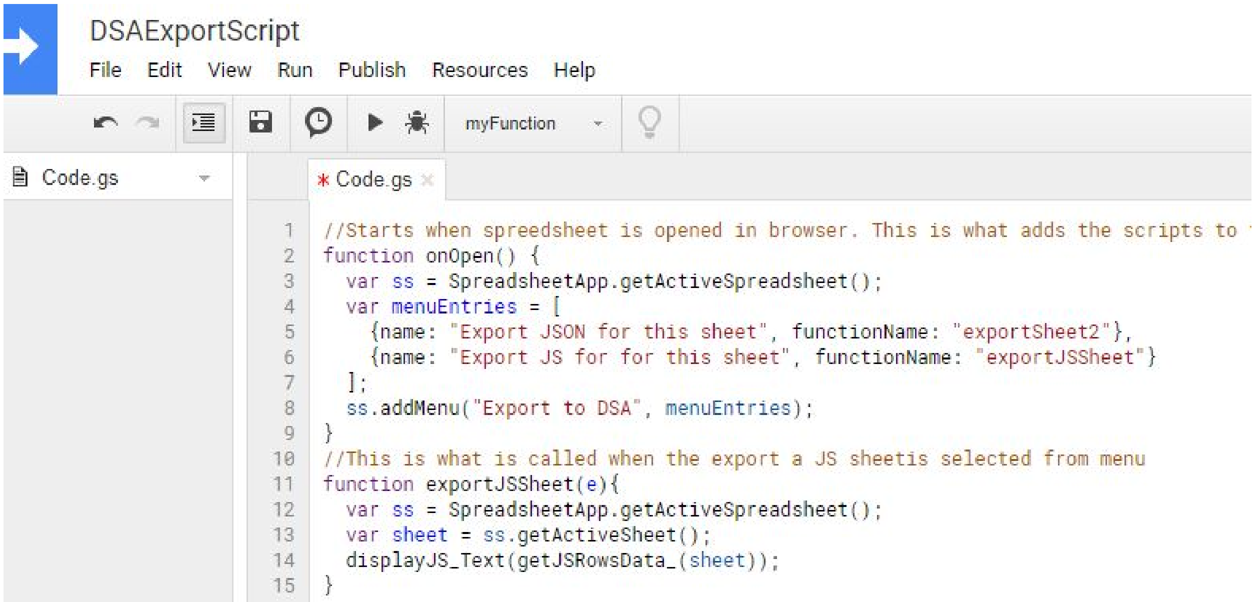
3. A new browser window will open and will look like below.


script.google.com/d/1Jq2TErcwysBeZ_pQC
Untitled project
File Edit View Run Publish Resources Help
myFunction
Code.gs
Code.gs

```
function myFunction() {

}
```

4. Make a Project Name. This will be required to save our script. The name can be anything you like though. For this tutorial it will be called DSAExportScript.
5. *Optional: By default google will make a blank .gs script file called Code.gs You can either use this file, make a new file or rename it.*
6. The script will need to be copy and pasted into a .gs file at this location.
   Obtain a copy of the .gs script
   a. There is a copy of the script at the end of this tutorial. (However for most updated version use the git repository version)
   b. The most updated version of the script will be in the dsa Git repository under
   Paste the contents of the scripts into the .gs file

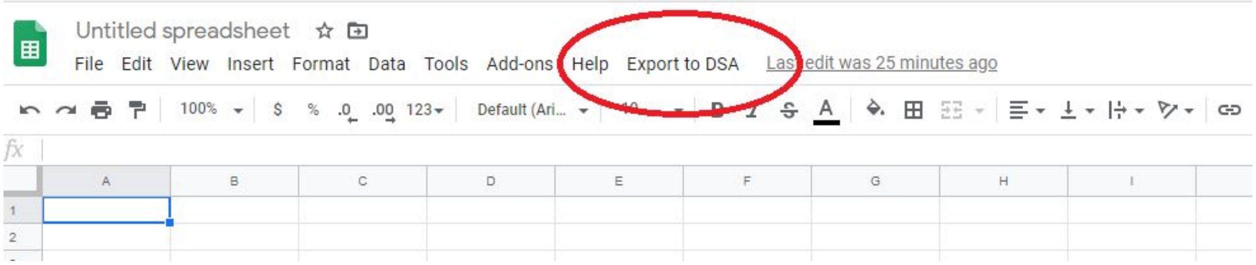


```
//Starts when spreedsheet is opened in browser. This is what adds the scripts to
function onOpen() {
  var ss = SpreadsheetApp.getActiveSpreadsheet();
  var menuEntries = [
    {name: "Export JSON for this sheet", functionName: "exportSheet2"},
    {name: "Export JS for for this sheet", functionName: "exportJSSheet"}
  ];
  ss.addMenu("Export to DSA", menuEntries);
}
//This is what is called when the export a JS sheetis selected from menu
function exportJSSheet(e){
  var ss = SpreadsheetApp.getActiveSpreadsheet();
  var sheet = ss.getActiveSheet();
  displayJS_Text(getJSRowsData_(sheet));
}
```

Save the updated .gs file by hitting the save icon (floppy disk icon).

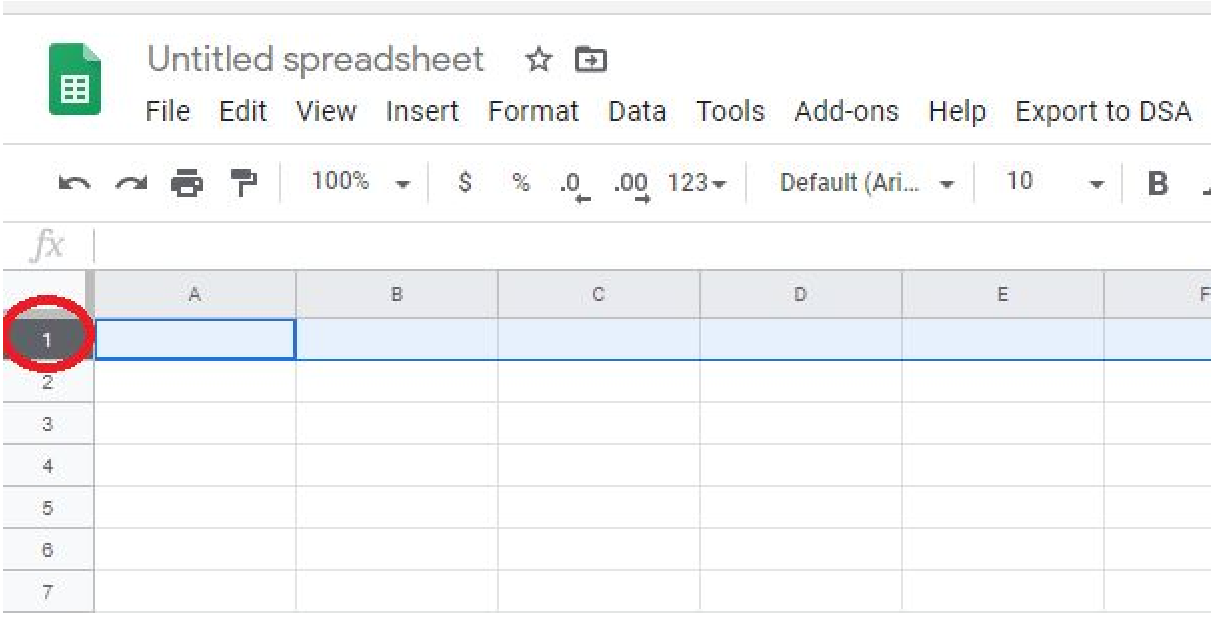
The script editor window can now be closed. You will not need to do anything in the editor anymore unless updating the script file.

7. Go back to the original Google Sheet window. Manually refresh the page by hitting the refresh icon on your browser or F5. After reloading a new item should be added to the menu at top. **Note: That this item will not appear immediately and will load in after all other items 5 to 10 seconds later.**

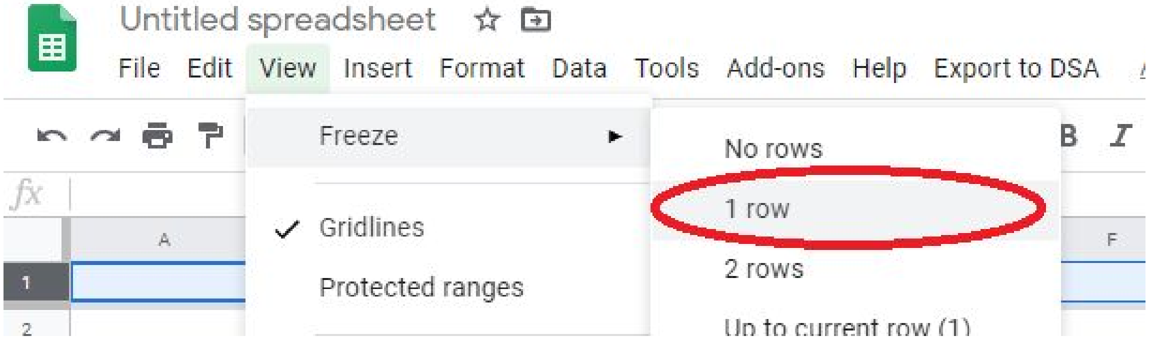

Untitled spreadsheet
File Edit View Insert Format Data Tools Add-ons Help Export to DSA
Last edit was 25 minutes ago
100%
Default (Ari...
A B C D E F G H I
1
2

The script requires some specific formatting in the sheet.

8. First let's freeze the first row. This will make the top row (our headers) stay on top when scrolling down for easier reading. **Note: The script looks at the frozen row to know where to start with the headers so this must be done.**
    a. Left click the number 1 to select the whole row.


Untitled spreadsheet
File Edit View Insert Format Data Tools Add-ons Help Export to DSA
100%
Default (Ari...
10
A B C D E F
1
2
3
4
5
6
7

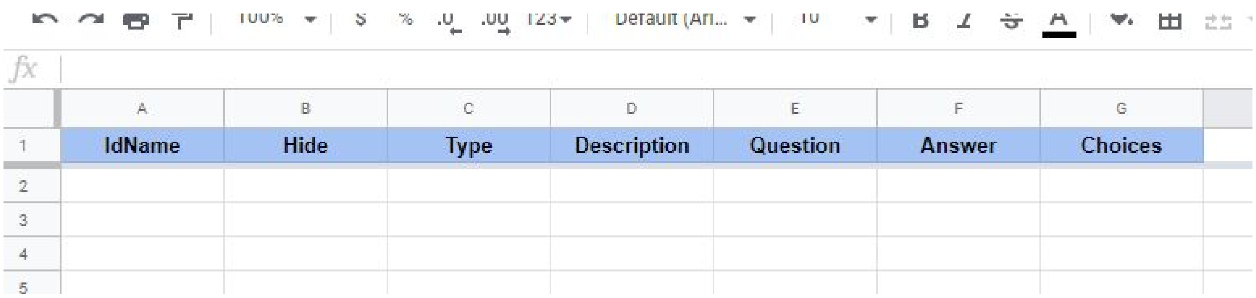
    b. On the top menu bar open the dropdown View and select Freeze. We are going to just freeze one row (the selected row) as in the picture below.



9. The script looks for headers in the first 7 columns. These must be named as:

   IdName Hide Type Description Question Answer Choices

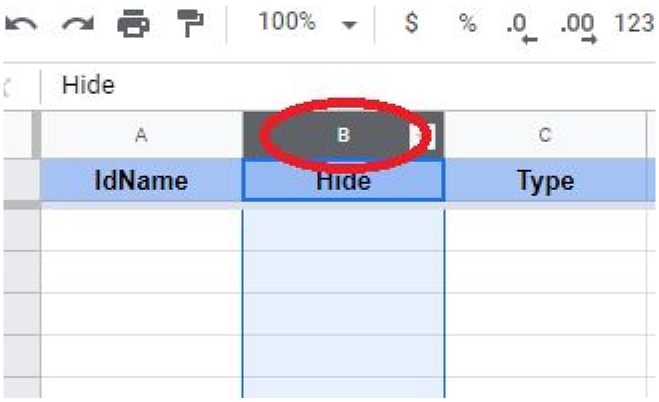
   Restrictions: The script uses these **EXACT** names but the **ORDER** does not matter. There is one exception to this rule and that is the first column “IdName” which can be named anything but has to be the first column.



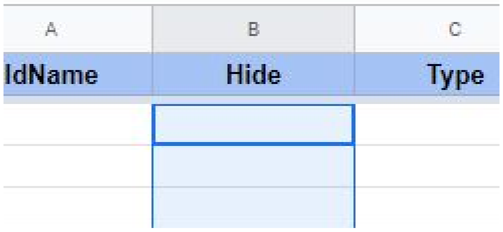
	A	B	C	D	E	F	G
1	IdName	Hide	Type	Description	Question	Answer	Choices
2							
3							
4							
5							

10. The script is now Fully functional and can now be used. See the next section on how to use the script. However these next steps are HIGHLY recommended as they will make entry much simpler.
11. Add a checkbox for the Hide column.

    A) Select the whole column by hitting the letter above the Cell called Hide



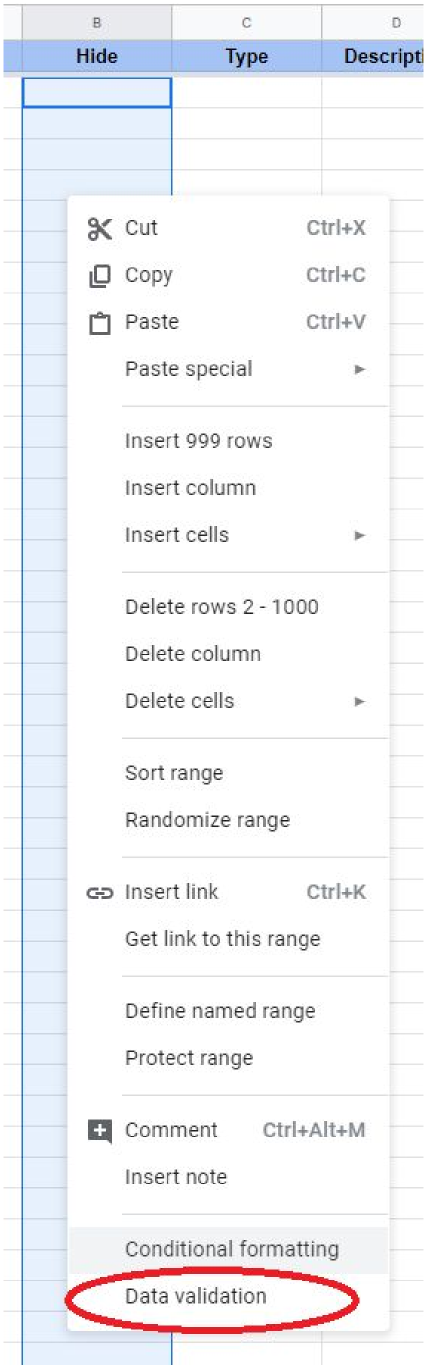
    B) Unselect the top row by holding Ctrl key and left click the cell called hide.



    Note: This is to make the column header not get applied to the next step.

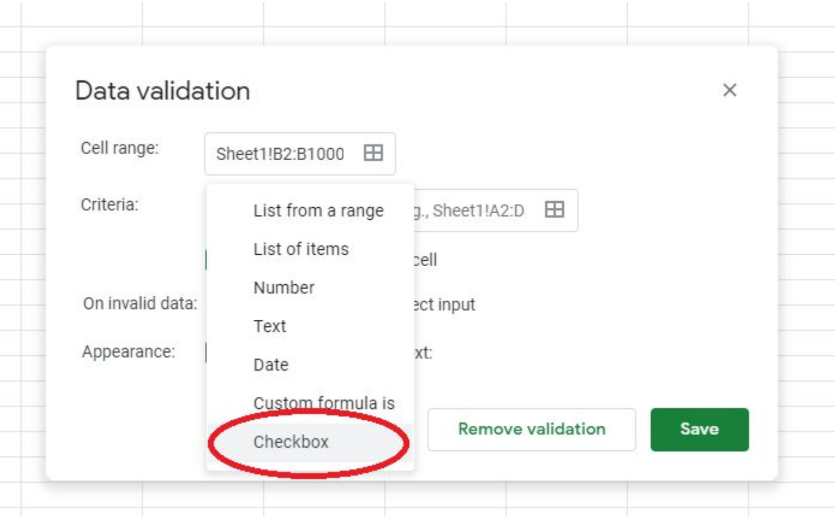
C) Right click inside the highlighted blue area to get a popup menu

D) Select Data validation (all the way at the bottom) as shown on the right



E) A popup window will emerge on Data Validation

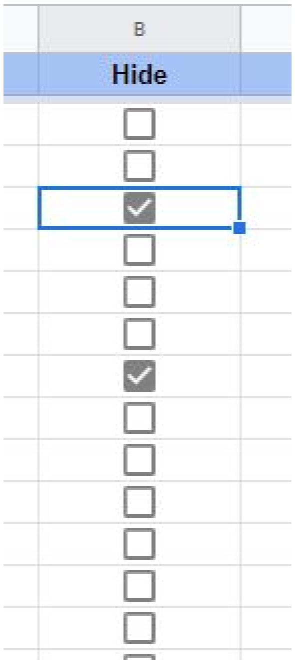
F) Under criteria click the dropdown menu and select Checkbox as shown below.


Data validation

Cell range: Sheet1!B2:B1000

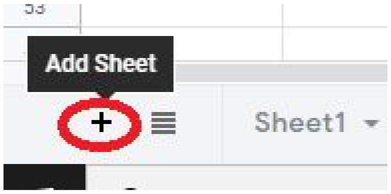
G) Now hit the Save button on the Data validation window.

H) Now the entire column should have checkboxes. As seen in picture below.



12. Add a dropdown for Type

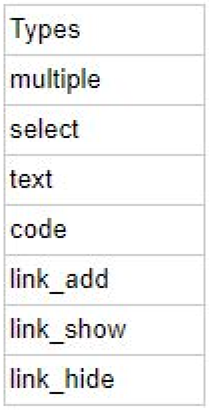
A) Create a new sheet. Hit the plus sign in the lower left corner of the browser window.



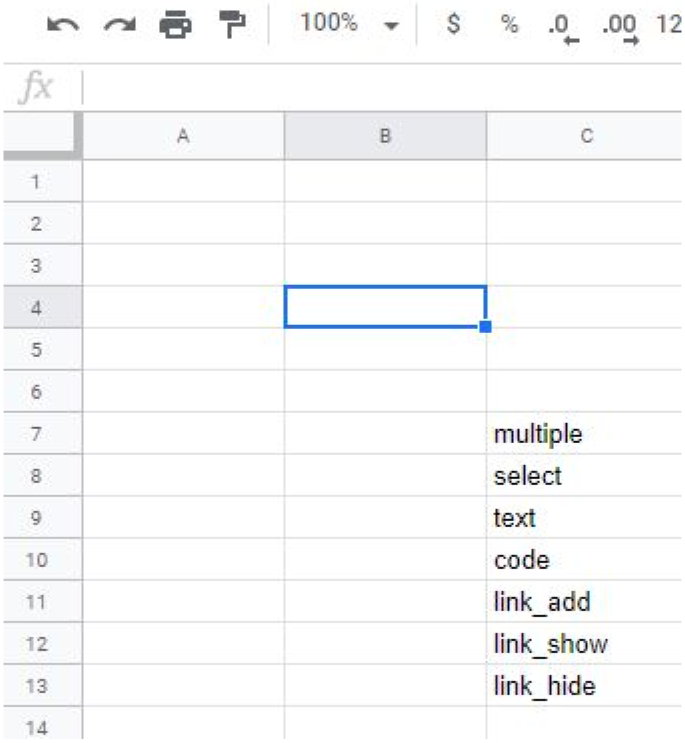
B) Click the tab created for the new sheet to switch to it.

C) Add the following words below on the left in a column somewhere on that sheet.

Note: Where we place them does not matter.



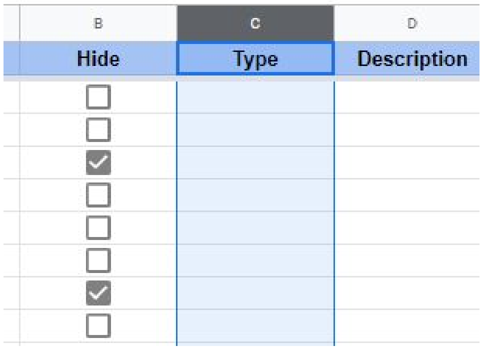
Types
multiple
select
text
code
link_add
link_show
link_hide



	A	B	C
1			
2			
3			
4			
5			
6			
7			multiple
8			select
9			text
10			code
11			link_add
12			link_show
13			link_hide
14			

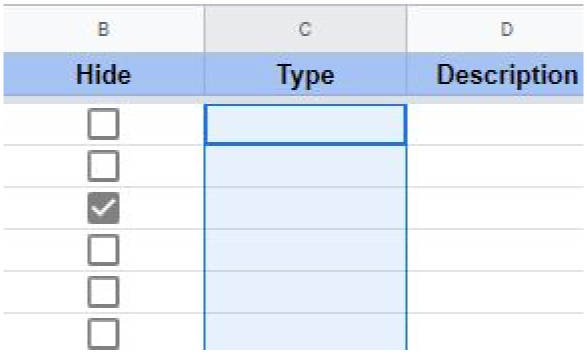
D) Now go back to the original first sheet.

E) Select the whole column by hitting the letter above the Cell called Type



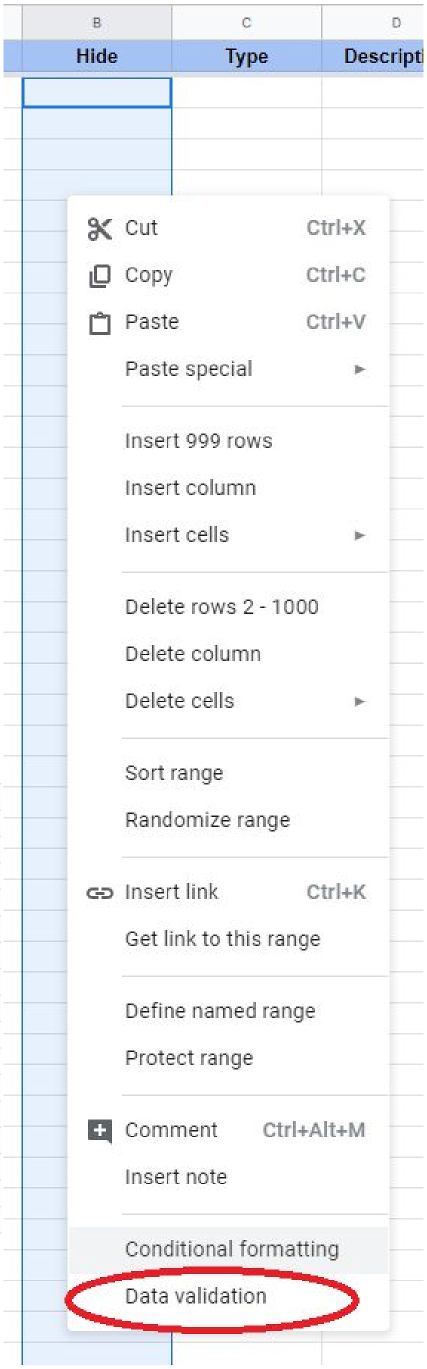
B	C	D
Hide	Type	Description
☐		
☐		
☑		
☐		
☐		
☐		
☑		
☐		

F) Unselect the top row by holding Ctrl on keyboard and left click the cell called Type.



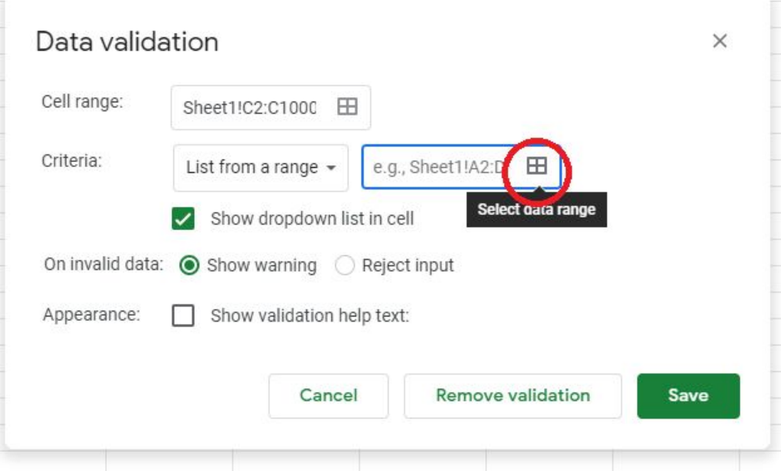
G) Right click inside the highlighted blue area to get a popup menu

H) Select Data validation (all the way at the bottom) as shown on the right



I) A popup window will emerge on Data Validation

J) Now select the grid in the 2nd box under criteria



Data validation

Cell range: Sheet1!C2:C1000

Criteria: List from a range ▾ e.g., Sheet1!A2:D

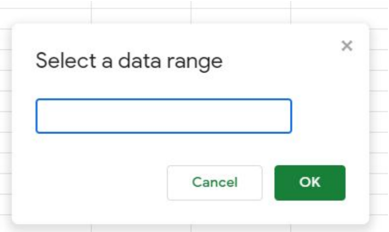
Select data range

☑ Show dropdown list in cell

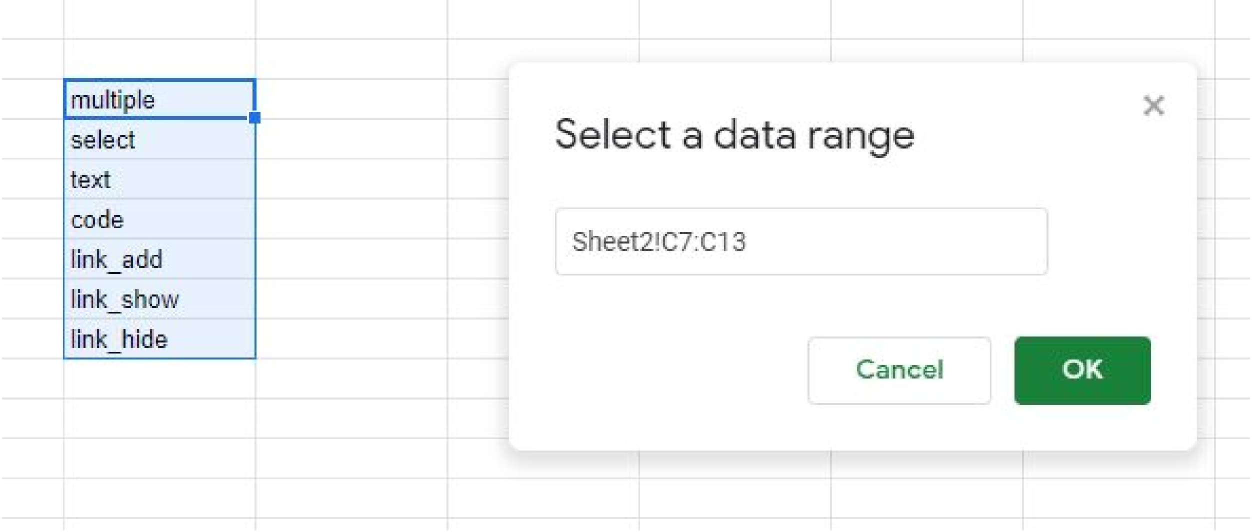
On invalid data: ● Show warning ○ Reject input

Appearance: □ Show validation help text:

K) A window now pops up to get the selected area. You can manually type this next step but it is easier to just do by clicking.



L) Now click the 2nd sheet we made earlier and highlight the area of the text you entered as shown below. Once they are all selected hit the OK button.

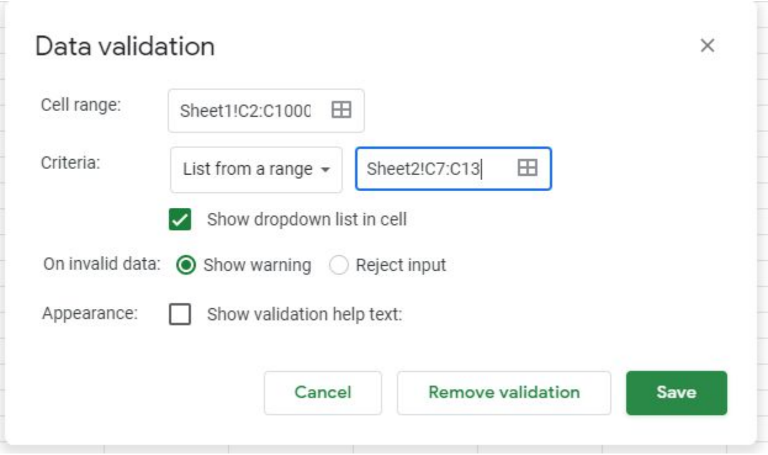


multiple
select
text
code
link_add
link_show
link_hide

Select a data range

Sheet2!C7:C13

M) You will now return the validation step. Hit the Save button.



Data validation

Cell range: Sheet1!C2:C1000

Criteria: List from a range ▾ Sheet2!C7:C13

☑ Show dropdown list in cell

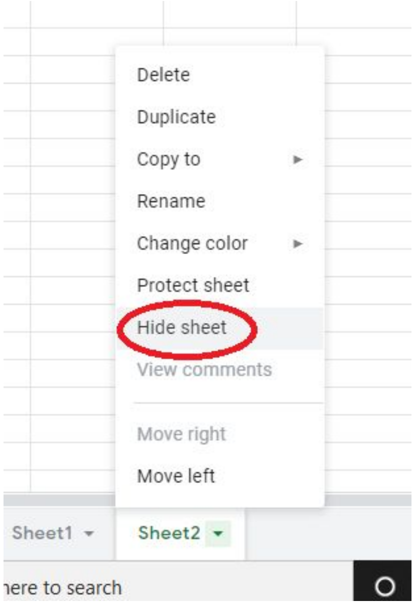
On invalid data: ● Show warning ○ Reject input

Appearance: □ Show validation help text:

Note: You can turn on the Reject Input on invalid data. This will make it so wrong data can never be entered vs seeing a warning when it is bad input.

N) You can now hide the second sheet so hit will not appear in your workspace. This will not stop the drop down menus from working.

O) The dropdown selection will now be enabled for Type column.



Script Code

```
//Starts when the spreadsheet is opened in the browser. This is what adds the scripts to the toolbar menu in google spreadsheet
function onOpen() {
  var ss = SpreadsheetApp.getActiveSpreadsheet();
  var menuEntries = [
    {name: "Export JSON for this sheet", functionName: "exportSheet2"},
    {name: "Export JS for for this sheet", functionName: "exportJSSheet"}
  ];
  ss.addMenu("Export to openDSA", menuEntries);
}
//This is what is called when the export a JS sheetis selected from menu
function exportJSSheet(e){
  var ss = SpreadsheetApp.getActiveSpreadsheet();
  var sheet = ss.getActiveSheet();
  displayJS_Text(getJSRowsData_(sheet));
}
//This is what is called when the export a JSON sheet is selected from menu
function exportSheet2(e) {
  var ss = SpreadsheetApp.getActiveSpreadsheet();
  var sheet = ss.getActiveSheet();
  var json = JSON.stringify(getRowsData(sheet), null, 4);
   displayJSON_Text(json);
}
//Grabs all the rows under the header row
function getRowsData(sheet) {
  var headersRange = sheet.getRange(1, 1, sheet.getFrozenRows(), sheet.getMaxColumns());
  var headers = headersRange.getValues()[0];
  var dataRange = sheet.getRange(sheet.getFrozenRows()+1, 1, sheet.getMaxRows(), sheet.getMaxColumns());
  var objects = getObjects(dataRange.getValues(), normalizeHeaders_(headers));
  return objects;
}
//Grabs all the rows under the header row
function getJSRowsData_(sheet) {
  var headersRange = sheet.getRange(1, 1, sheet.getFrozenRows(), sheet.getMaxColumns());
  var headers = headersRange.getValues()[0];
  var dataRange = sheet.getRange(sheet.getFrozenRows()+1, 1, sheet.getMaxRows(), sheet.getMaxColumns());
  var text = getJSText(dataRange.getValues(), normalizeHeaders_(headers));
  return text;

}
//Returns the JSon text to be displayed in html
// Arguments:
//   - Data: an object containing the row entries
//   - key: the column header names for the frozen row
// Returns an object to convert into a JSON.
function getObjects(data, keys) {

  var jsonObjects = {};
  //Loop through rowdata
  for (var i = 0; i < data.length; ++i) {
    var rowObject = {};
    var hasData = false;
    //Set name of json to refName
    //This is the first column on spreed sheet
    if(isCellEmpty_(data[i][0])){
        continue;
    }
    for (var j = 1; j < data[i].length; ++j) {
      var cellData = data[i][j];
      if (isCellEmpty_(cellData)) {
        continue;
      }
      if(keys[j] == "hide"){
        //Do not add to json entire row when hide column is true
        if(cellData == true){ break;}
        //Do not add hide column to json when false
        else{ continue;}
      }
      //Do not make json of when a type that is JS export mode only
      if(keys[j] == "type"){
        //These types should not be added to json, they are javascript only
        if(cellData == "link_add" ||cellData == "link_show" || cellData == "link_hide" || cellData == "code" || cellData == "text"){ break;}
      }
       //Check for answer column - format of ';' for multiple entries
      if(keys[j] == "answer"){
       rowObject[keys[j]] = cellData.split(";");

      }
      //Check for answer column - format of ';' for multiple entries
      else if(keys[j] == "choices"){
        //Change the delimter
        var tempStr  = cellData.replace(new RegExp(';','g'), "tempDelim");
        tempStr  = tempStr.replace(new RegExp('<','g'), "&lt;");
        tempStr = tempStr.replace(new RegExp('>','g'),"&gt;");
        //rowObject[keys[j]] = cellData.split(";");
        rowObject[keys[j]] =  tempStr.split("tempDelim");

      }
      //Check for description column - format for < and > and "
      else if(keys[j] == "description"){
        //Change the text to be json able
         var textStr = cellData.replace(new RegExp('<','g'), "&lt;");
         textStr = textStr.replace(new RegExp('>','g'),"&gt;");
        //rowObject[keys[j]] = cellData.split(";");
        rowObject[keys[j]] =  textStr;

      }
      //Skipp data for columns more than 7
      //This is to make sure the exported raw string columns do not get added
      else if( j > 7){
        continue;
      }
      else{

        rowObject[keys[j]] = cellData;
      }
      hasData = true;
    }
    if (hasData) {
      jsonObjects[data[i][0]] = rowObject;

    }
  }
  //Setup data for json
  var translationObj = {};
  var enObj = {};
  translationObj["translations"] = enObj;
  enObj["en"] = jsonObjects;


  return translationObj;
}
//Returns the JS text to be displayed in html
// Arguments:
//   - Data: an object containing the row entries
//   - key: the column header names for the frozen row
// Returns a rawText for JS file.
function getJSText(data, keys) {
  var ss = SpreadsheetApp.getActiveSpreadsheet();
  var sheet = ss.getSheets()[0];
  var text = "$(document).ready(function() {\n\"use strict\";\nvar av_name = \""+ SpreadsheetApp.getActiveSpreadsheet().getActiveSheet().getName() +"\";\nvar av = new JSAV(av_name);\nvar Frames = PIFRAMES.init(av_name);\nvar config = ODSA.UTILS.loadConfig({ av_name: av_name }),\n\tinterpret = config.interpreter,\n\tcode = config.code;\nvar goNext = false;\n";
  //Init display
  var init = "\nav.displayInit();"
  var frameCount = 1;
 for (var i = 0; i < data.length; ++i) {
    var hasData = false;
    //Set name of json to refName
    //This is the first column on spreed sheet
    if(isCellEmpty_(data[i][0])){
        continue;
    }
   //Loop through all columns in row.
   for (var j = 1; j < data[i].length; ++j) {
      var cellData = data[i][j];
     //Do not add to js when hide column is true
      if(keys[j] == "hide" && cellData == true){
        break;
      }

     if(keys[j] == "type"){
       if(cellData == "code"){
         //Add the comment (description column)
         text += "\n//" + data[i][3] + "\n";
         //he code (question column)
         text += data[i][4];

       }
       else if(cellData == "link_add"){
         text+= "\n//"+ data[i][3];
         text+= "\nvar url"+data[i][0]+ "=\"" + data[i][4] + "\";"
         text+= "\nvar "+data[i][0]+"= new av.ds.FA({center:true , url:url"+data[i][0]+"});";
       }
       else if(cellData == "link_show"){
         text+= "\n"+data[i][0]+".show();";
       }
       else if(cellData == "link_hide"){
         text+= "\n"+data[i][0]+".hide();";
       }
       else if(cellData == "text"){
         text+= "\n//Frame "+ frameCount++;
         var textStr  =  data[i][3].replace(new RegExp(/\\/,'g'), "\\\\");
         textStr  =  textStr.replace(new RegExp('"','g'), "\\\"");
         textStr  = textStr.replace(new RegExp('<','g'), "&lt;");
         textStr = textStr.replace(new RegExp('>','g'),"&gt;");
         text+= "\nav.umsg(\""+textStr+"\");";
         if(frameCount > 2)
           text+= "\nav.step();";
         else
           text += init;


       }
       //Add as a question
       else{
         //Check if first frame
         if(frameCount == 1)   text += init;
         text += "\n//Frame "+ frameCount++ + "\nav.umsg(Frames.addQuestion(\"" + data[i][0] + "\"));\nav.step();";
       }
     }
   }//End of looping through spreedsheet rows
 }
  text += "\n\nav.recorded();\n});"
  return text;
}
function displayJS_Text(text) {
//  var style = "<style> h1 {  color: blue;  font-family: verdana;  font-size: 300%;} .tag:before{content: '<'}.tag:after{content: '>'}</style>";
 // var test = "<h1 class=\"tag\">test</h1>";
  //text = "&lt</pre>; : &lt;<test>" + text;
  var output = HtmlService.createHtmlOutput("<textarea style='width:100%;' rows='20'>" + text + "</textarea>");

  var ss = SpreadsheetApp.getActiveSpreadsheet();
  var sheet = ss.getActiveSheet();
  var cell = sheet.getRange("I3");
  cell.setValue(text);

  output.setWidth(500);
  output.setHeight(500);
  SpreadsheetApp.getUi()
      .showModalDialog(output, 'Exported JS');
}

function displayJSON_Text(text) {
  var output = HtmlService.createHtmlOutput("<textarea style='width:100%;' rows='20'>" + text + "</textarea>");

  var ss = SpreadsheetApp.getActiveSpreadsheet();
  var sheet = ss.getActiveSheet();
  var cell = sheet.getRange("K3");
  cell.setValue(text);

  output.setWidth(500)
  output.setHeight(500);
  SpreadsheetApp.getUi()
      .showModalDialog(output, 'Exported JSON');
}
// getRowsData iterates row by row in the input range and returns an array of objects.
// Each object contains all the data for a given row, indexed by its normalized column name.
// Arguments:
//   - sheet: the sheet object that contains the data to be processed
//   - range: the exact range of cells where the data is stored
//   - columnHeadersRowIndex: specifies the row number where the column names are stored.
//       This argument is optional and it defaults to the row immediately above range;
// Returns an Array of objects.
function getRowsData_(sheet) {
  var headersRange = sheet.getRange(1, 1, sheet.getFrozenRows(), sheet.getMaxColumns());
  var headers = headersRange.getValues()[0];
  var dataRange = sheet.getRange(sheet.getFrozenRows()+1, 1, sheet.getMaxRows(), sheet.getMaxColumns());
  var objects = getObjects_(dataRange.getValues(), normalizeHeaders_(headers));
  return objects;

}

// For every row of data in data, generates an object that contains the data. Names of
// object fields are defined in keys.
// Arguments:
//   - data: JavaScript 2d array
//   - keys: Array of Strings that define the property names for the objects to create
function getObjects_(data, keys) {
  var objects = [];
  for (var i = 0; i < data.length; ++i) {
    var object = {};
    var hasData = false;
    for (var j = 0; j < data[i].length; ++j) {
      var cellData = data[i][j];
      if (isCellEmpty_(cellData)) {
        continue;
      }
      object[keys[j]] = cellData;
      hasData = true;
    }
    if (hasData) {
      objects.push(object);
    }
  }
  return objects;
}

// Returns an Array of normalized Strings.
// Arguments:
//   - headers: Array of Strings to normalize
function normalizeHeaders_(headers) {
  var keys = [];
  for (var i = 0; i < headers.length; ++i) {
    var key = normalizeHeader_(headers[i]);
    if (key.length > 0) {
      keys.push(key);
    }
  }
  return keys;
}

// Normalizes a string, by removing all alphanumeric characters and using mixed case
// to separate words. The output will always start with a lower case letter.
// This function is designed to produce JavaScript object property names.
// Arguments:
//   - header: string to normalize
// Examples:
//   "First Name" -> "firstName"
//   "Market Cap (millions) -> "marketCapMillions
//   "1 number at the beginning is ignored" -> "numberAtTheBeginningIsIgnored"
function normalizeHeader_(header) {
  var key = "";
  var upperCase = false;
  for (var i = 0; i < header.length; ++i) {
    var letter = header[i];
    if (letter == " " && key.length > 0) {
      upperCase = true;
      continue;
    }
    if (!isAlnum_(letter)) {
      continue;
    }
    if (key.length == 0 && isDigit_(letter)) {
      continue; // first character must be a letter
    }
    if (upperCase) {
      upperCase = false;
      key += letter.toUpperCase();
    } else {
      key += letter.toLowerCase();
    }
  }
  return key;
}
// Returns true if the character char is alphabetical, false otherwise.
function isAlnum_(char) {
  return char >= 'A' && char <= 'Z' ||
    char >= 'a' && char <= 'z' ||
    isDigit_(char);
}
// Returns true if the character char is a digit, false otherwise.
function isDigit_(char) {
  return char >= '0' && char <= '9';
}

// Given a JavaScript 2d Array, this function returns the transposed table.
// Arguments:
//   - data: JavaScript 2d Array
// Returns a JavaScript 2d Array
// Example: arrayTranspose([[1,2,3],[4,5,6]]) returns [[1,4],[2,5],[3,6]].
function arrayTranspose_(data) {
  if (data.length == 0 || data[0].length == 0) {
    return null;
  }

  var ret = [];
  for (var i = 0; i < data[0].length; ++i) {
    ret.push([]);
  }

  for (var i = 0; i < data.length; ++i) {
    for (var j = 0; j < data[i].length; ++j) {
      ret[j][i] = data[i][j];
    }
  }

  return ret;
}

// Returns true if the cell where cellData was read from is empty.
// Arguments:
//   - cellData: string
function isCellEmpty_(cellData) {
  return typeof(cellData) == "string" && cellData == "";
}
```