Create — Applications from Ideas
Written Response Submission Template

Submission Requirements

2. Written Responses

Submit one PDF document in which you respond directly to each prompt. Clearly label your responses **2a – 2d in order. Your response to all prompts combined must not exceed 750 words, exclusive of the Program Code.**

Program Purpose and Development

2a. Identify the programming language and identify the purpose of your program. Explain your video using one of the following:

- A written summary
- of what the video illustrates OR
- An audio narration in your video. If you choose this option, your response to the written summary should read, “The explanation is located in the video.”

(Approximately 150 words)

Insert response for 2a in the text box below.

Our program was coded in python, using various built-in libraries, the Easy GUI Library and the Google Sheets API. Our program’s purpose is to automate a process that is currently performed manually by humans. The program will be used to help our School’s Stage Crew team. Currently, people hosting events at the school request the technical assistance of Stage Crew, through a Google Form. A member of Stage Crew then must look at this data, which is outputted to a Google Sheet, assess how many ‘Senior Crew’ members and ‘Junior Crew’ members are needed to help run the event, and what time they must arrive to set up (known as the Call Time), depending on the equipment needs of the event. Then, they copy this information to a separate spreadsheet, where other members of the Stage Crew sign up to help with the event. Our program automates this process, so that this can all be accomplished with just a click of a button in our GUI. In the submitted video, we fill out the Event Requests google form. Then, our program first determines if there are new events, then it reads the data from these new events, it selects the relevant information, processes the data to determine the number of crew members needed and the Call Time, and then copies the relevant data to a different spreadsheet, in the right spot.

2b. Describe the incremental and iterative development process of your program, focusing on two distinct points in that process. Describe the difficulties and/or opportunities you encountered and how they were resolved or incorporated. In your description clearly indicate whether the development described was collaborative or independent. At least one of these points must refer to independent program development; the second could refer to either collaborative or independent program development. *(Approximately 200 words)*

Insert response for 2b in the text box below.

We began by splitting our program into all the abstractions and functions necessary, and writing each one. Then once they were all written and tested using generalized data, I wrote the script to call all the functions and put them together - the main() function. As part of this process, there were numerous opportunities and difficulties. One algorithm we wrote calculated the Call Time for an event. To do this, we had to convert times into integers that we could perform arithmetical operations upon. At first, we converted times to a list, and wrote a complex function to perform arithmetic on the time. Then, during the “debugging” phase, I found the datetime library, a built-in Python library to do exactly this. I realized that this was an opportunity to make our code much more efficient, consistent and easier to debug. So, I independently rewrote all the functions that processed times, to use this function. Another opportunity, discovered during the original development of all the abstractions and functions was the discovery of the difflib library, which I used, completely independently, to create function that could evaluate if a cell was similar to another cell. To integrate this, I had to test many different cell values to determine a percentage threshold for this similarity algorithm, that works with out data. This library simplified code that written without the library would have been very complex. A difficulty we had was that the Google API had very complex documentation, and was very difficult to work with. I had to do a lot of research and trial and error, to a get a reliable way of integrating with this API.

2c. Capture and paste an image or images of your program code segment that implements the most complex algorithm you wrote. (marked with a **color border** below)

```
def main(values): #Written by: EK, Debugged by both WZ and EK
    lastIndex=len(values)-1
    #2. Loop through the necessary rows and write the relevant data to a dictionary - newEvents
    newEvents={}
    for event in values[newRowFinder('lastrow.txt'):len(values)]:#Only updates rows that have not been updated previously. See newRowFinder() .
        newEvents.setdefault(event[2], {'dateOfEvent': event[3],
                                        'eventCoordinator': event[4],
                                        'staffSupervisor': event[5],
                                               'eventType': event[6],
                                               'eventStart': event[7],
                                               'houseOpen':event[10],
                                               'curtains':event[13],
                                        'podium':event[14],
                                        'additionalFurniture': event[15],
                                        'audienceQs':event[16],
                                        'AvMedia':event[22],
                                        'Lighting':event[24]
                                        }
                                )
    #3. Write values to correct location in other file
    #First, read the Event List File
    rangeName = 'Event List'
    result = service.spreadsheets().values().get(spreadsheetId=spreadsheetId, range=rangeName).execute()
    values = result.get('values', [])

    #Create a dictionary called 'months' with all the indexes of where months begin
    months={'July':len(values)}
    for rows in values[1:]:
        if rows[0] !='':
            months.setdefault(rows[0])
            months[rows[0]]=values.index(rows)

    #Begin the copy
    for event in newEvents:
        #Find where to insert
        eventMonth=findMonth(newEvents[event]['dateOfEvent'])
        nextMonth=nxtMonth(newEvents[event]['dateOfEvent'])

        for rows in values[months[eventMonth]:months[nextMonth]]:
            #print(rows[1], "compared with", findDay(newEvents[event]['dateOfEvent']))
            if int(rows[1])==findDay(newEvents[event]['dateOfEvent']): #Checks to see if an event on the day of the event exists - if yes, it knows the event row to insert
                rowToInsert=values.index(rows)+1
                indextoInsert=values.index(rows)
                break
            elif int(rows[1])>findDay(newEvents[event]['dateOfEvent']): #Checks to see if an event after the day of the event exists, setting the index to insert as 1 row above
                rowToInsert=values.index(rows)
                indextoInsert=values.index(rows)-1
                break
            else:
                rowToInsert=months[nextMonth]
                indextoInsert=months[nextMonth]-1

        if not isSimilar(values[indextoInsert][2],event): #Checks to see if an event with the same name exists on that date
            #If yes, be prepared to overwrite it
            #Othwerwise, create a new row below the existing row, to add this new event
            rowToInsert+=1
            indextoInsert+=1
            #InsertNewRow using InsertDimensionRequest
            requests=[]
            requests.append({"insertRange": { "range":{
                "sheetId": 1034566956,
                "startRowIndex": rowToInsert-1,
                           "endRowIndex": rowToInsert,
                                                  "startColumnIndex": 0,
                                                  "endColumnIndex": 12, },
                                           "shiftDimension": "ROWS",
                                           }})
            body = {"requests": requests}
            response = service.spreadsheets().batchUpdate(spreadsheetId=spreadsheetId,body=body).execute()


        """Perform the copy using the spreadsheet.values collection's BatchUpdate request"""
        ct=callTime(newEvents[event]['houseOpen'], newEvents[event]['eventType'])
        batch_update_values_request_body = {
            # How the input data should be interpreted.
            'value_input_option': 'USER_ENTERED',

                    # The new values to apply to the spreadsheet.
                           'data': [
                               {"range": str(rangeName)+'!B'+str(rowToInsert), "majorDimension": "ROWS", "values": [[findDay(newEvents[event]['dateOfEvent'])]]},
                               {"range": str(rangeName)+'!C'+str(rowToInsert),"values": [[event]]},
                               {"range": str(rangeName)+'!D'+str(rowToInsert),"values": [["Event Coordinator: " + str(newEvents[event]['eventCoordinator']) +"\n" +"Staff Supervisor: " + str(newEvents[event]['staffSupervisor'])]]},
                                   {"range": str(rangeName)+'!E'+str(rowToInsert),"values": [[ct[1]]]},
                                   {"range": str(rangeName)+'!G'+str(rowToInsert),"values": [[ct[2]]]},
                                   {"range": str(rangeName)+'!J'+str(rowToInsert),"values": [[formatTime(ct[0])]]},
                                   {"range": str(rangeName)+'!K'+str(rowToInsert),"values": [["Type: "+str(newEvents[event]['eventType'])
                                                                                             +"\n Event Starts: "+ str(newEvents[event]['eventStart'])
                                                                                             +"\n House Opens: "+str(newEvents[event]['houseOpen'])
                                                                                             +"\n Curtains: "+str(newEvents[event]['curtains'])
                                                                                    +"\n Podium: "+str(newEvents[event]['podium'])
                                                                                    +"\n Audience Questions: "+str(newEvents[event]['audienceQs'])
                                                                                    +"\n Lights On: "+str(newEvents[event]['Lighting'])
                                                                                    +"\n Additional AV Media: "+str(newEvents[event]['AvMedia'])
                                                                                    +"\n Additional Furniture: "+str(newEvents[event]['additionalFurniture'])
                                                                                    ]]}
                               ],

        }

        request = service.spreadsheets().values().batchUpdate(spreadsheetId=spreadsheetId, body=batch_update_values_request_body)
        response = request.execute()
        print('Updated Row ' +str(rowToInsert) +' for '+event)

#Mission Accomplished
        print ('All Updates Complete!!')

        saveLastRow('lastrow.txt', lastIndex) #Saves the last row to the file
```

Your algorithm should integrate several mathematical and logical concepts.
Describe the mathematical and logical concepts used to develop the algorithm.
Explain the complexity of the algorithm and how it functions in the program.
(Approximately 200 words)

Insert text response for 2c in the plain box below.

The most complex function I (In the code, comments with the letters EK identify code written by me, while the letters WZ identify code written by my partner) wrote individually was the 106-line main() function. My partner helped with debugging, but I was the only one to write or edit code within this function. This is an algorithm that integrates with the Google Sheets API, to accomplish the core purpose of our program – processing the data from one Google Sheet and copying it to another. This algorithm integrates with most of the other smaller abstractions within. The definition statements of all the smaller functions can be found within Part 3, the program code. All these smaller algorithms are noteworthy because when put together as part of this main() algorithm they allow the program to perform its main purpose. A noteworthy integrated algorithm is the newRowFinder() algorithm, which reads the last row updated in the Google Sheet, which was previously stored in a text file. There are also the findMonth() and the findDay() algorithms, which process a date from the Google Sheets format and output the month as a word or the day as an integer, respectively. Another noteworthy integrated algorithm is the IsSimilar() algorithm, which takes two strings as inputs and uses the difflib library to calculate a ratio for how similar they are to each other, returning True if this ratio is equal to or greater than 0.55 and False otherwise. This is used to determine if an event is already in the Event List (the destination spreadsheet), and evaluate whether to update an existing event or create a new event. Finally, the callTime() algorithm calculates the time to arrive in order to setup, as well as the number of both “junior” and “senior” crew needed. The formatTime() algorithm is then used to convert the time type object into a string that is formatted in 12-hour format. Lastly, the saveLastRow() algorithm saves the index of the last row that was updated in a text file, so it can be accessed by the program later.

2d. Capture and paste an image or images of the program code segment that contains an abstraction you developed (marked with a matching **blue color border** below)

```
83  def isSimilar(a, b): #Written by: EK
84      similarity= SequenceMatcher(None, a, b).ratio() # This funcion (SequenceMatcher) Came from the built in difflib library.
        https://d.python.org/2/library/difflib.html#sequencematcher-objects
85      if similarity >=0.55:
86          return True
87      else:
88          return False
```

Your abstraction should integrate mathematical and logical concepts.
Explain how your abstraction helped manage the complexity of your program.
(Approximately 200 words)

Insert text response for 2d in the plain box below.

An abstraction that I developed is the isSimilar() abstraction, which takes two strings as inputs and evaluates if they are similar. This function integrates the difflib library's SequenceMatcher().ratio() functions which compute a ratio to express the similarity of two strings. This operates based on the following formula Similarity Ratio= 2 · Matches/Total Characters in Both Strings. 'Matches' is calculated by iterating through each character in a string and checking to see if it exists in the other string. If it does, 1 would be added to Matches. My abstraction integrates takes the ratio computed by the SequenceMatcher().ratio() function and compares it to a threshold similarity ratio of 0.55. If over 0.55, values 'a' and 'b' are considered similar, and True is returned. Otherwise, the values are considered not similar and False is returned. The alternative of not using this function would be an added if-else comparison in the middle of the code, and an extra global variable to store the result of the if-else comparison. By not having to have this process in the middle of the code, and have an extra global variable, this abstraction simplifies the complexity of the code, and the number of lines of code needed.