**Python Journal Template**

**Directions:** Follow the directions for each part of the journal template. Include in your response all the elements listed under the Requirements section. Prompts in the Inspiration section are not required; however, they may help you to fully think through your response.

Remember to review the Touchstone page for entry requirements, examples, and grading specifics.

**Name: Monet Paul**

**Date: 03/16/2025**

**Final IDE Program Share Link: https://onlinegdb.com/1hHjuUzTS**

Complete the following template. Fill out all entries using complete sentences.

## PART 1: Defining Your Problem

|  |
| --- |
| **Task**  State the problem you are planning to solve.  **Requirements**   * Describe the problem you are trying to solve for. * Describe any input data you expect to use. * Describe what the program will do to solve the problem. * Describe any outputs or results the program will provide.   **Inspiration**  When writing your entry below ask yourself the following questions:   * Why do you want to solve this particular problem? * What source(s) of data do you believe you will need? Will the user need to supply that data, or will you get it from an external file or another source? * Will you need to interact with the user throughout the program? Will users continually need to enter data in and see something to continue? * What are your expected results or what will be the end product? What will you need to tell a user of your program when it is complete? |
| Financial planners handwritten for a check-by-check setup is hard to do on paper every month and accurate. Users must recalculate the dates of bills when the month is long or short. They would have to move the bill to the next check period and recalculate the bill’s amount for that check week. The problem to solve is to remove extra work by calculating the bills’ dates and amount for the check period and displaying the results where it shows the breakdown. I will build a program that will do all the calculations of the bills and put the bills in the correct check period to pay them. When the program starts, it provides a sample file and will show an example to the user and explain how this tool will help and what they will input into the program. Once the user confirms they are ready, the program will ask them to put in some information on their bills, check periods, and if more than one person is paying the bills. After the program is done with its calculation for one year, it will display results and ask the user if they would want the result in a txt file and be downloaded. User will have better control over bills and improve on financial planning by providing information of their bills, check period and number of people paying the bills in this program. |

## PART 2: Working Through Specific Examples

|  |
| --- |
| **Task**  Write down clear and specific steps to solve a simple version of your problem you identified in Part 1.  **Requirements**  Complete the three steps below **for at least two distinct examples/scenarios**.   * State any necessary data for your simplified problem. * Write clear and specific steps in English (not Python) detailing what the program will do to solve the problem. * Describe the specific result of your example/scenario.   **Inspiration**  When writing your entry below ask yourself the following questions:   * Are there any steps that you don’t fully understand? These are places to spend more time working out the details. Consider adding additional smaller steps in these spots. * Remember that a computer program is very literal. Are there any steps that are unclear? Try giving the steps of your example/scenario to a friend or family member to read through and ask you questions about parts they don’t understand. Rewrite these parts as clearly as you can. * Are there interesting edge cases for your program? Try to start one of your examples/scenarios with input that matches this edge case. How does it change how your program might work? |
| <Write your journal entry response here>    Example 1: Simple User Scenario (1 person paying bills and only fixed bills)   1. User will be asked how many people are paying bills. They will enter 1. 2. Users will be asked their name, the date when they got their last paid check and the amount of check. 3. Users will be asked how many bills they have and enter the numbers of bills. 4. User will be asked if they have any variable/fluctuate bills. User will enter no. 5. A for loop which the length will be based on number of fixed bills will run with the following questions to the user   5a. User will be asked the name of the bill.  5b. User will be asked what day out the month is the due.  5c. User will be asked what is estimate or exact amount of this bill .   1. After user enter all data needed for the loop, a function will run where it will calculate the check periods all the way to next year and same thing for the fixed bills as well. All information will be stored. 2. Next running another function will be to compare check periods to the due dates of bills and calculating which bill will be under what check period and always make sure the total bill amount is not over the check amount. It will scan to see if the bill is over 50% of the check period and if it is, they will add a note in the storage variable would say “this bill will take all or most of your check, you will have to split this amount in smaller chucks”. 3. The result will be put in a file for the user - here’s an small example   “  \*\*Username’s financial planner\*\*  ------------------------------------------------------------------------------  \*\*Check period: 01/20/2025\*\* Amount:$1,200.00 total bills: $374.98  -Spotify  Due Date: 01/25/2025  Amount: 11.99  -Netflix  Due Date: 01/28/2025  Amount: 12.99  -**Rent (exclude from total bills)**  Due Date: 02/01/2025  Amount: 1,200.00  \*\*\*Note: this bill will take all or most of your check, you will have to split this amount in smaller chucks.  -Credit Card  Due Date: 02/04/2025  Amount:$350.00  -----------------------------------------------------------------------------------  \*\*Check period: 02/05/2025\*\* Amount:$1,200.00 total bills: $174.98  -Adobe  Due Date: 02/10/2025  Amount: 49.99  -Xbox live  Due Date: 02/15/2025  Amount: 24.99  -AI Fun  Due Date: 02/07/2025  Amount: $100.00  “  Example 2: Challenge User Scenario (2 people paying bills and fixed and variable bills)   1. User will be asked how many people are paying bills. They will enter 2. 2. A for loop which the length will be based on number of people paying the bills will run with the following questions to the user.   2a. Users will be asked the person’s name, the date when the person got their last paid check and the amount of the check.   1. Users will be asked how many bills they have and enter the numbers of bills. 2. User will be asked if they have any variable/fluctuate bills. User will enter yes. 3. User will be asked how many variable/fluctuate bills. User will enter the numbers of those bills. 4. A for loop which the length will be based on number of variable/fluctuate bills will run with the following questions to the user   6a. User will be asked the name of the bill.  6b. User will be asked what the date was their last due date of this bill.  6c. User will be asked what is estimate amount of this bill .   1. A for loop which the length will be based on number of fixed bills will run with the following questions to the user   7a. User will be asked the name of the bill.  7b. User will be asked what day out the month is the due.  7c. User will be asked what is estimate or exact amount of this bill .   1. After the user enter all data needed for the loops, a function will run where it will calculate the check periods all the way to next year and same thing for the fixed bills as well. All information will be stored. 2. A function for variable/fluctuate bills, where it calculate a future estimate date and give an range of dates for all the variable/fluctuate bills for the whole year. All information will be stored. 3. Next running another function will be to compare check periods to the due dates of the bills and calculating which bill will be under what check period and always make sure the total bill amount is not over the check amount. It will scan to see if the bill is over 50% of the check period and if it is, they will add a note in the storage variable would say “this bill will take all or most of your check, you will have to split this amount in smaller chucks”. 4. The result will be put in a file for the user - here’s an small example   “  \*\*Alice Johnson & John Smith's Financial Planner\*\*  ------------------------------------------------------------------------------  \*\*Alice Johnson's Check period: 01/21/2025\*\* Amount: $1,300.00 Total bills: $418.94  - Apple Music  Due Date: 01/26/2025  Amount: $9.99  - Hulu  Due Date: 01/27/2025  Amount: $14.99  - Disney+  Due Date: 01/28/2025  Amount: $7.99  - Amazon Prime  Due Date: 01/28/2025  Amount: $12.99  - **Rent (excluded from total bills)**  Due Date: 02/01/2025  Amount: $1,250.00  \*\*\*Note: this bill will take all or most of your check, you will have to split this amount in smaller chunks.  - Student Loan  Due Date: 02/03/2025  Amount: $372.98  ------------------------------------------------------------------------------  \*\*John Smith's Check period: 02/05/2025\*\* Amount: $1,300.00 Total bills: $422.99  - Car Payment  Due Date: 02/05/2025  Amount: $422.99  ------------------------------------------------------------------------------  \*\*Alice Johnson's Check period: 02/06/2025\*\* Amount: $1,250.00 Total bills: $197.96  - Canva  Due Date: 02/11/2025  Amount: $49.99  - PlayStation Plus  Due Date: 02/16/2025  Amount: $24.99  - Google Play Store  Due Date: 02/12/2025  Amount: $7.99  - HBO Max  Due Date: 02/17/2025  Amount: $14.99  - Fitness Class  Due Date: 02/08/2025  Amount: $100.00  ------------------------------------------------------------------------------  \*\*John Smith's Check period: 02/20/2025\*\* Amount: $1,300.00 Total bills: $518.94  - \*\*Phone bill\*\*  Due Date: 02/22/2025  Amount: $100.00  - Apple Music  Due Date: 02/26/2025  Amount: $9.99  - Hulu  Due Date: 02/27/2025  Amount: $14.99  - Disney+  Due Date: 02/28/2025  Amount: $7.99  - Amazon Prime  Due Date: 02/28/2025  Amount: $12.99  - **Rent (excluded from total bills)**  Due Date: 03/01/2025  Amount: $1,250.00  \*\*\*Note: this bill will take all or most of your check, you will have to split this amount in smaller chunks.  - Student Loan  Due Date: 03/03/2025  Amount: $372.98 |

## PART 3: Generalizing Into Pseudocode

|  |
| --- |
| **Task**  Write out the general sequence your program will use, including all specific examples/scenarios you provided in Part 2.  **Requirements**   * Write pseudocode for the program in English but refer to Python program elements where they are appropriate. The pseudocode should represent the full functionality of the program, not just a simplified version. Pseudocode is broken down enough that the details of the program are no longer in any paragraph form. One statement per line is ideal.   **Help with writing pseudocode**   * Here are a few links that can help you write pseudocode with examples. Remember to check out part 3 of the Example Journal Template Submission if you have not already. Note: everyone will write pseudocode differently. There is no right or wrong way to write it other than to make sure you write it clearly and in as much detail as you can so that it should be easy to convert it to code later.   + <https://www.geeksforgeeks.org/how-to-write-a-pseudo-code/>   + <https://www.wikihow.com/Write-Pseudocode>   **Inspiration**  When writing your entry below ask yourself the following questions:   * Do you see common program elements and patterns in your specific examples/scenarios in Part 2, like variables, conditionals, functions, loops, and classes? These should be part of your pseudocode for the general sequence as well. * Are there places where the steps for your examples/scenarios in Part 2 diverged? These may be places where errors may occur later in the project. Make note of them. * When you are finished with your pseudocode, does it make sense, even to a person that does not know Python? Aim for the clearest description of the steps, as this will make it easier to convert into program code later. |
| <Write your pseudocode here>  planner equal {  peopleNum: “ ”,  peopleArray : [ ],  totalBills: “ ”,  vfBillNum: “ ”,  vfBillArray: [ ],  fixedBillNum: “ ”,  fixedBillArray: [ ]  }  Function input ()  Ask user to enter how many people are paying bills  Store input into planner.peopleNum  if planner.peopleNum is greater than 1  For loop with length based on planner.peopleNum  personInfo equal { name: “ ”, checkDate: “ ”, checkAmount:“ ”, futurePeriods: [ ] }  Ask User to enter person[i + 1] name  Store input in personInfo.name  Ask User to enter the date when person[i + 1] got their last paid check (personInfo.name)  Store input in personInfo.checkDate  Ask User to enter the amount of person[i + 1]’s check (personInfo.name)  Store input in personInfo.checkAmount  personInfo will be pushed to planner.peopleArray  else  personInfo equal { name: “ ”, checkDate: “ ”, checkAmount:“ ”, futurePeriods: [ ] }  Ask User to enter their name  Store input in personInfo.name  Ask User to enter the date when they got their last paid check  Store input in personInfo.checkDate  Ask User to enter the amount of check  Store input in personInfo.checkAmount  personInfo will be pushed to planner.peopleArray  Ask user to enter the number of bills  Store input to planner.totalBills  Ask User if they have any variable/fluctuate bills (yes or no)  Store input in answerVfBills  if answerVfBills equal yes  Ask user to enter the number of variable/fluctuate bills  Store input in planner.vfBillNum  For loop with length based on planner.vfBillNum  billInfo equal { name: “ ”, billDate: “ ”, billAmount:“ ”, futurePeriods: [ ] }  Ask User to enter variable/fluctuate bill[i + 1] name  Store input in billInfo.name  Ask User to enter the date of their last due date for this bill (billInfo.name)  Store input in billInfo.billDate  Ask User to enter the estimated amount of bill (billInfo.name)  Store input in billInfo.billAmount  billInfo will be pushed to planner.vfBillArray  else  Store 0 in planner.vfBillNum  planner.fixedBillNum equal planner.totalBills minus planner.vfBillNum  For loop with length based on planner.fixedBillNum  billInfo equal { name: “ ”, billDay: “ ”, billAmount:“ ”, futurePeriods: [ ] }  Ask User to enter fixed bill[i + 1] name  Store input in billInfo.name  Ask User to enter the day of the month this bill is due (billInfo.name)  Store input in billInfo.billDay  Ask User to enter the estimated or exact amount of bill (billInfo.name)  Store input in billInfo.billAmount  billInfo will be pushed to planner.fixedBillArray  Function calculation()  For loop with length based on planner.peopleArray  For loop with length 26  nextPeriod equal “ ”  previous equal nextPeriod not equal “ ” or planner.peopleArray[i].checkDate  futureDate equal previous plus 2 weeks (14 days)  push futureDate to planner.peopleArray[i].futurePeriods  nextPeriod equal futureDate  For loop with length based on planner.fixedBillNum  todayMonth equal get today month  todayYear equal get today year  standardBillDate equal todayMonth plus planner.fixedBillArray[i].billDay plus todayYear  push standardBillDate to planner.fixedBillArray[i].futurePeriods  For loop with length 11  nextPeriod equal “ ”  current equal nextPeriod not equal “ ” or standardBillDate  futureBillDate equal current plus 30 days  push futureBillDate to planner.fixedBillArray[i].futurePeriods  nextPeriod equal futureBillDate  For loop with length based on planner.vfBillNum  lastDate equal planner.vfBillArray[i].billDate  For loop with length 11  range equal {min: “ ”, middle: “ ” , max: “ ” }  nextDate equal “ ”  currentDate equal nextDate not equal “ ” or lastDate  middleDate equal currentDate plus 30 days  range.middle equal middleDate  minDate equal middleDate minus 2 days  range.min equal minDate  maxDate equal middleDate plus 8 days  range.max equal maxDate  push range to planner.vfBillArray[i].futurePeriods  nextDate equal middleDate  allChecks equal [ ]  For loop based on planner.peopleArray  For loop based on planner.peopleArray[i].futurePeriods  checkLayout equal { name: “ ”, date: “ ”, amount: “ ”}  checkLayout.name equal planner.peopleArray[i].name  checkLayout.date equal planner.peopleArray[i].futurePeriods[j]  checkLayout.amount equal planner.peopleArray[i].amount  push checkLayout to allChecks  organizedChecks equal allChecks sort by earlier date  allBills equal [ ]  For loop based on planner.fixedBillArray  For loop based on planner.fixedBillArray[i].futurePeriods  billLayout equal { name: “ ”, date: “ ”, amount: “ ”, type: “fixed” }  billLayout.name equal planner.fixedBillArray[i].name  billLayout.date equal planner.fixedBillArray[i].futurePeriods[j]  billLayout.amount equal planner.fixedBillArray[i].amount  push billLayout to allBills  For loop based on planner.vfBillArray  For loop based on planner.vfBillArray[i].futurePeriods  billLayout equal { name: “ ”, date: {}, amount: “ ”, type: “variable/fluctuate” }  billLayout.name equal planner.vfBillArray[i].name  billLayout.date equal planner.vfBillArray[i].futurePeriods[j]  billLayout.amount equal planner.vfBillArray[i].amount  push billLayout to allBills  organizedBills equal allBills sort by earliest date  displayOutput equal [ ]  For loop with length based on organizedChecks  totalBillAmount equal 0  maxAmount equal organizedChecks[i].amount  periodLayout equal {check: {name: “ ”, date: “ ”, amount: “ ”}, bills: [ ], totalBills:“ ”, overAmount: “ ”}  periodLayout.check.name equal organizedChecks[i].name  periodLayout.check.date equal organizedChecks[i].date  periodLayout.check.amount equal organizedChecks[i].amount  For loop with length based on organizedBills  fixedbillDetail equal {name: “ ”, date: “ ”, amount: “ ”, type: “Fixed”, note:“ ”}  vfBillDetail equal {name: “ ”, date:{}, amount: “ ”, type:“variable/fluctuate”, note:“ ”}  if organizedBills[j].type equal “variable/fluctuate”  compareDate equal organizedBills[j].date.min  if compareDate is greater or equal to organizedChecks[i].date and is less than organizedChecks[i + 1].date  vfBillDetail.name equal to organizedBills[j].name  vfBillDetail.date equal to organizedBills[j].date  vfBillDetail.amount equal to organizedBills[j].amount  if organizedBills[j].amount is greater than or equal to half of organizedChecks[i].Amount  vfBillDetail.note equal to “this bill will take all or most of your check, you will have to split this amount in smaller chunks”  push vfBillDetail to periodLayout.bills  else  totalBillAmount + organizedBills[j].amount  push vfBillDetail to periodLayout.bills  else  if organizedBills[j].date is greater or equal to organizedChecks[i].date and is less than organizedChecks[i + 1].date  fixedbillDetail.name equal to organizedBills[j].name  fixedbillDetail.date equal to organizedBills[j].date  fixedbillDetail.amount equal to organizedBills[j].amount  if organizedBills[j].amount is greater than or equal to half of organizedChecks[i].Amount  fixedbillDetail.note equal to “this bill will take all or most of your check, you will have to split this amount in smaller chunks”  push fixedbillDetail to periodLayout.bills  else  totalBillAmount + organizedBills[j].amount  push fixedbillDetail to periodLayout.bills  push totalBillAmount to periodLayout.totalBills  over equal maxAmount minus totalBillAmount  if over is less than 0  push over to periodLayout.overAmount  push periodLayout to displayOutput  Main function  Call input ()  Call calculation()  create a file and store displayOutput in file and design the format of how to display the information  output the results |

## PART 4: Testing Your Program

|  |
| --- |
| **Task**  While writing and testing your program code, describe your tests, record any errors, and state your approach to fixing the errors.  **Requirements**   * For at least one of your test cases, describe how your choices for the test helped you understand whether the program was running correctly or not.   For each error that occurs while writing and testing your code:   * Record the details of the error from your IDE. A screenshot or copy-and-paste of the text into the journal entry is acceptable. * Describe what you attempted in order to fix the error. Clearly identify what approach was the one that worked.   **Inspiration**  When writing your entry below ask yourself the following questions:   * Have you tested edge cases and special cases for the inputs of your program code? Often these unexpected values can cause errors in the operation of your program. * Have you tested opportunities for user error? If a user is asked to provide an input, what happens when they give the wrong type of input, like a letter instead of a number, or vice versa? * Did the outcome look the way you expected? Was it formatted correctly? * Does your output align with the solution to the problem you coded for? |
| <Record your errors and fixes here>  For my test cases, I used while loops for my conditions that I wanted to accept for the program to run. I designed the program to re-ask the questions if the user enters invalid input. In the mix of bugs and errors, I realized I need to make a class function handler for validation in the while loop for more details like decimals with try and except statements.  Error 1:  I got an error that says something is wrong with the first input for the function input. This is what the console said      While looking over the code, I forgot to add the colon and realized that I can't name the function input because input is a method.    I changed the name of the function to inputInformation    Now I don’t get the error message.  Error 2:  When I enter a non-number like a. I get an error where the program can’t run.    Looking at the code, I have broken down the question into three parts. I created a while loop to keep asking until it’s in the correct format as a number and user didn’t enter a blank.  I ensured the if statement reads an integer value, not a string.    Error 3: The program will not accept a comma in the money amount asked for a check. It will keep asking for the money amount.        I added more in the NumberHander, I split the values by the comma and put them back together as a number without the comma.    It now works |

## PART 5: Commenting Your Program

|  |
| --- |
| **Task**  Submit your full program code, including thorough comments describing what each portion of the program should do when working correctly.  **Requirements**   * The purpose of the program and each of its parts should be clear to a reader that does not know the Python programming language.   **Inspiration**  When writing your entry, you are encouraged to consider the following:   * Is each section or sub-section of your code commented to describe what the code is doing? * Give your code with comments to a friend or family member to review. Add additional comments to spots that confuse them to make it clearer. |
| <Copy your full program code here, including comments>  ----- main.py file----  # This program will help with organizing the bills and checks for the user  #import numberHandler class from classesFunctions file  from classesFunctions import NumberHandler  #import the following libraries to help with the program datetime, timedelta, and re  from datetime import datetime, timedelta  import re  # This  varible will help with storing the information from the user  planner = {       "peopleNum": "",       "peopleArray" : [ ],       "totalBills" : "",       "vfBillNum" : "",      "vfBillArray": [ ],       "fixedBillNum": "",       "fixedBillArray": [] }  # This function will help with getting the information from the user  def inputInformation():      #this  variable will help with checking the correct format of the date      pattern = r"^(0[1-9]|1[0-2])/([0-2][0-9]|3[0-1])/(\d{4})$"        # while loop check correct format  if it's a number      while planner["peopleNum"] == "" or planner["peopleNum"].isnumeric() == False:       # asking about people who are  paying the bills information       planner["peopleNum"] = input("Number of people who are paying bills?:")      #for loop to get the information of the people      for i in range(int(planner["peopleNum"])):          #this  variable will help with storing the information of the person and  set up  the layout of the organization later in the code          personInfo = {                  "name": "",                  "checkDate": "",                  "checkAmount": "",                  "futurePeriods": []              }          #check to make sure name is not empty          while personInfo["name"] == "":              personInfo["name"] = input("Enter your name: " if int(planner["peopleNum"]) == 1  else f"Enter person {i + 1} name: ")          #add check correct format of date          while personInfo["checkDate"] == "" or re.match(pattern, personInfo["checkDate"]) == None:              personInfo["checkDate"] = input("Enter the date when you got your last paid check(mm/dd/yyyy):"if int(planner["peopleNum"]) == 1  else f"Enter the date when {personInfo['name']} got their last paid check(mm/dd/yyyy):")              #check to make sure the amount is a number          while personInfo["checkAmount"] == "" or not NumberHandler.is\_number(personInfo["checkAmount"]):              personInfo["checkAmount"] = input("Enter the amount of your check $: " if int(planner["peopleNum"]) == 1  else f"Enter the amount of {personInfo['name']}'s check $: ")          #add the information to the array          planner["peopleArray"].append(personInfo)  # asking about  bills      # check correct format if it's a number      while planner["totalBills"] == "" or planner["totalBills"].isnumeric() == False:       planner["totalBills"] = input("Enter the number of total bills:")      # check correct format if it's a number      while planner["vfBillNum"] == "" or planner["vfBillNum"].isnumeric() == False or int(planner["vfBillNum"]) > int(planner["totalBills"]):       planner["vfBillNum"] = input("Enter the number of variable/fluctuate bills(can't be over the total bills):")      if int(planner["vfBillNum"]) > 0:         #for loop to get the information of the variable/fluctuate bills          for i in range(int(planner["vfBillNum"])):              #this billInfo variable will help with storing the information of the variable/fluctuate bills and  set up  the layout of the organization later in the code              billInfo = {                  "name": "",                  "billDate": "",                  "billAmount": "",                  "futurePeriods": []              }              #check to make sure name is  fill out              while billInfo["name"] == "":               billInfo["name"] = input(f"Enter variable/fluctuate bill {i + 1} name:")              #add check correct format of date              while billInfo["billDate"] == "" or re.match(pattern, billInfo["billDate"]) == None:               billInfo["billDate"] = input(f"Enter the date of the last due date for this bill(mm/dd/yyyy):")              #check correct format  if it's a number              while billInfo["billAmount"] == "" or not NumberHandler.is\_number(billInfo["billAmount"]):               billInfo["billAmount"] = input(f"Enter the estimated amount of {billInfo['name']} $:")              #add the information to the array              planner["vfBillArray"].append(billInfo)      #We will calculate the fixed bills based on the total bills and the variable/fluctuate bills      planner["fixedBillNum"] = int(planner["totalBills"]) - int(planner["vfBillNum"])     #for loop to get the information of the fixed bills      for i in range(int(planner["fixedBillNum"])):          #this billInfo variable will help with storing the information of the fixed bills and  set up  the layout of the organization later in the code          billInfo = {              "name": "",              "billDay": "",              "billAmount": "",              "futurePeriods": []          }          #add check to make sure name is  fill out          while billInfo["name"] == "":           billInfo["name"] = input(f"Enter fixed bill {i + 1} name: ")          #add check correct format  if it's a number          while billInfo["billDay"] == "" or billInfo["billDay"].isnumeric() == False or int(billInfo["billDay"]) >= 31:           billInfo["billDay"] = input(f"Enter the day of the month this bill is due:")          #add check correct format  if it's a number          while billInfo["billAmount"] == "" or not NumberHandler.is\_number(billInfo["billAmount"]):           billInfo["billAmount"] = input(f"Enter the estimated or exact amount of {billInfo['name']} $:")          planner["fixedBillArray"].append(billInfo)  def calculation():      #the following code will help get the  future periods for the checks      for i in range(len(planner["peopleArray"])):          # this nextPeriod variable is a placeholder for the next period          nextPeriod = ""          # for loop to get the future periods for the checks          for j in range(26):              #this checkDateSetup variable will help with storing the information of the future periods for the checks and the  format of the date              checkDateSetup= {"format": "", "compare": ""}              previous = nextPeriod if nextPeriod !="" else planner["peopleArray"][i]["checkDate"]              #this formatPrevious variable will help with the format of the date              formatPrevious = datetime.strptime(previous, "%m/%d/%Y")              #this  futureDate variable will help with getting the future date              futureDate = formatPrevious + timedelta(days=14)              checkDateSetup["format"] = futureDate.strftime("%m/%d/%Y")              checkDateSetup["compare"] = futureDate              #add the information to the array              planner["peopleArray"][i]["futurePeriods"].append(checkDateSetup)              #set the next period to the future date              nextPeriod = futureDate.strftime("%m/%d/%Y")        #the following code will help get the  future periods for the fixed  bills      for i in range(int(planner["fixedBillNum"])):          todayDate = datetime.today()          todayMonth = todayDate.month          todayYear = todayDate.year          # this  month variable will help with getting the correct format of the month          month = todayMonth if todayMonth > 9 else f'0{todayMonth}'          day = planner['fixedBillArray'][i]['billDay']         # this  day variable will help with getting the correct format of the day          day = int(day) if int(day) > 9 else f'0{int(day)}'          # this  standardBillDate variable will help with getting the correct format of the date          standardBillDate = f"{month}/{day}/{todayYear}"          # this nextPeriod variable is a placeholder for the next period          nextPeriod = ""          for j in range(12):              #this dateSetup variable will help with storing the information of the future periods for the fixed bills and the  format of the date              dateSetup = {"format": "","compare": ""}              current = nextPeriod if nextPeriod != "" else standardBillDate              #this startDate variable will help with the format of the date              startDate = datetime.strptime(current, "%m/%d/%Y")              nextMonth = startDate.month if j == 0 else (startDate.month + 1)              nextYear = startDate.year              # this  section of code will help with getting the correct format of the month and year              if nextMonth > 12:                  nextMonth = 1                  nextYear = startDate.year + 1              monthWithZero = nextMonth if nextMonth > 9 else f'0{nextMonth}'              futureBillDate = f"{monthWithZero}/{day}/{nextYear}"              dateSetup["format"] = futureBillDate              dateSetup["compare"] = datetime.strptime(futureBillDate, "%m/%d/%Y")             #add the information to the array              planner["fixedBillArray"][i]["futurePeriods"].append(dateSetup)              nextPeriod = futureBillDate     #the following code will help get the  future periods for the variable/fluctuate bills      for i in range(int(planner["vfBillNum"])):          lastDate = planner["vfBillArray"][i]["billDate"]          #this nextDate variable is a placeholder for the next date          nextDate = ""          for j in range(11):              #this rangeSetup variable will help with storing the information of the future periods for the variable/fluctuate bills and the  format of the              rangeSetup = {"min": "", "middle": "", "max": "", "compare": ""}              currentDate = nextDate if nextDate !="" else lastDate              formatCurrentDate = datetime.strptime(currentDate, "%m/%d/%Y")             #this middleDate variable will help with getting the middle date              middleDate = formatCurrentDate + timedelta(days=30)              rangeSetup["middle"] = middleDate.strftime("%m/%d/%Y")              #this minDate variable will help with getting the min date              minDate = middleDate - timedelta(days=2)              rangeSetup["min"] = minDate.strftime("%m/%d/%Y")              rangeSetup["compare"] = minDate             #this maxDate variable will help with getting the max date              maxDate = middleDate + timedelta(days=8)              rangeSetup["max"] = maxDate.strftime("%m/%d/%Y")              #add the information to the array              planner["vfBillArray"][i]["futurePeriods"].append(rangeSetup)              nextDate = middleDate.strftime("%m/%d/%Y")      #the following code will help with organizing the checks     #this allChecks variable will help with storing the information of the checks      allChecks = []      for i in range(len(planner["peopleArray"])):          for j in range(len(planner["peopleArray"][i]["futurePeriods"])):              #this checkLayout variable will help with storing the information of the checks              checkLayout = {"name": "", "date": "", "amount": ""}              checkLayout["name"] = planner["peopleArray"][i]["name"]              checkLayout["date"] = planner["peopleArray"][i]["futurePeriods"][j]              checkLayout["amount"] = planner["peopleArray"][i]["checkAmount"]              allChecks.append(checkLayout)      #this organizedChecks variable will help with organizing the checks from the allChecks variable      organizedChecks = sorted(allChecks, key=lambda x: x["date"]["compare"])      #the following code will help with organizing the bills      #this allBills variable will help with storing the information of the bills      allBills = []      # for loop for  fixed bills      for i in range(len(planner["fixedBillArray"])):          for j in range(len(planner["fixedBillArray"][i]["futurePeriods"])):              billLayout = {"name": " ", "date": " ", "amount": " ", "type": "fixed"}              billLayout["name"] = planner["fixedBillArray"][i]["name"]              billLayout["date"] = planner["fixedBillArray"][i]["futurePeriods"][j]              billLayout["amount"] = planner["fixedBillArray"][i]["billAmount"]              allBills.append(billLayout)      # for loop for  variable/fluctuate bills      for i in range(len(planner["vfBillArray"])):          for j in range(len(planner["vfBillArray"][i]["futurePeriods"])):              billLayout = {"name": " ", "date": {}, "amount": " ", "type": "variable/fluctuate"}              billLayout["name"] = planner["vfBillArray"][i]["name"]              billLayout["date"] = planner["vfBillArray"][i]["futurePeriods"][j]              billLayout["amount"] = planner["vfBillArray"][i]["billAmount"]              allBills.append(billLayout)      #this organizedBills variable will help with organizing the bills from the allBills variable      organizedBills = sorted(allBills, key=lambda x: x["date"]["compare"])      # This loop will help with displaying the information in the correct format for  the file      #this displayOutput variable will help with storing the information of the display      displayOutput = []      # for loop  to put the  bills under the correct check      for i in range(len(organizedChecks)):          totalBillAmount = 0          #this maxAmount variable will help with getting the max amount of the check          maxAmount = float(organizedChecks[i]["amount"])          #this periodLayout variable will help with storing the information of the display and the  format of the display          periodLayout = {"check": {"name": " ", "date": " ", "amount": " "}, "bills": [], "totalBills": " ", "overAmount": "none"}          periodLayout["check"]["name"] = organizedChecks[i]["name"]          periodLayout["check"]["date"] = organizedChecks[i]["date"]["format"]          periodLayout["check"]["amount"] = organizedChecks[i]["amount"]          # for loop to put the  bills under the correct check          for j in range(len(organizedBills)):              #this fixedbillDetail variable will help with storing the information of the fixed bills and the  format of the display              fixedbillDetail = {"name": " ", "date": " ", "amount": " ", "type": "Fixed", "note": " "}              #this vfBillDetail variable will help with storing the information of the variable/fluctuate bills and the  format of the display              vfBillDetail = {"name": " ", "date": {}, "amount": " ", "type": "variable/fluctuate", "note": " "}             #this compareDate variable will help with getting the compare date              compareDate = organizedBills[j]["date"]["compare"]              #this if statement will help with organizing the bills based if it's a variable/fluctuate or fixed bill              if organizedBills[j]["type"] == "variable/fluctuate":                  if compareDate >= organizedChecks[i]["date"]["compare"] and (i + 1 < len(organizedChecks) and compareDate < organizedChecks[i + 1]["date"]["compare"]):                      vfBillDetail["name"] = organizedBills[j]["name"]                      vfBillDetail["date"] = organizedBills[j]["date"]                      vfBillDetail["amount"] = organizedBills[j]["amount"]                      if float(organizedBills[j]["amount"]) >= 0.5 \* float(organizedChecks[i]["amount"]):                          vfBillDetail["note"] = "---this bill will take all or most of your check, you will have to split this amount in smaller chunks---"                          periodLayout["bills"].append(vfBillDetail)                      else:                          totalBillAmount +=  float(organizedBills[j]["amount"])                          periodLayout["bills"].append(vfBillDetail)              else:                  if compareDate >= organizedChecks[i]["date"]["compare"] and (i + 1 < len(organizedChecks) and compareDate < organizedChecks[i + 1]["date"]["compare"]):                      fixedbillDetail["name"] = organizedBills[j]["name"]                      fixedbillDetail["date"] = organizedBills[j]["date"]["format"]                      fixedbillDetail["amount"] = organizedBills[j]["amount"]                      if float(organizedBills[j]["amount"]) >= 0.5 \* float(organizedChecks[i]["amount"]):                          fixedbillDetail["note"] = "---this bill will take all or most of your check, you will have to split this amount in smaller chunks---"                          periodLayout["bills"].append(fixedbillDetail)                      else:                          totalBillAmount += float(organizedBills[j]["amount"])                          periodLayout["bills"].append(fixedbillDetail)                  else:                      continue          #add the information to the array and round the total bill amount          periodLayout["totalBills"] =  round(totalBillAmount, 2)         #this over variable will help with getting the over amount          over = maxAmount - totalBillAmount          if over < 0:              periodLayout["overAmount"] = round(over, 2)          #add the information to the array          displayOutput.append(periodLayout)      #return the information      return displayOutput  #this function will help with displaying the information in the correct format for  the file and  run the program  def output():        #this userAnswer variable will help with getting the user input      userAnswer = ""      #this while loop is the main loop for the program      while userAnswer.lower() != "exit" and userAnswer.lower() != "enter":       #this userinput variable will help with getting the user input and give the user instructions       userinput = input("Welcome to the Financial Planner\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\nThis program will help you organize your bills and checks.\nThis program will ask some questions so please gather all of your bills and information.\nIf you want to look at an example of the output please look at the sample.txt file.\nPlease follow the instructions below.\nWhen entering the date please use the following format: mm/dd/yyyy\nWhen entering the amount please use numbers only.\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\nTo continue type enter or type exit to leave the program:")       userAnswer = userinput      if userAnswer.lower() == "exit":          return      else:       #getting the information from the user       inputInformation()      #getting the results       result= calculation()       #putting  results in a file       with open("output.txt", "w") as file:          #this file.write will help with writing the information in the file and it  put it in a layout to be easy to read          file.write(f"Financial Planner Todays date: {datetime.today().strftime('%m/%d/%Y')}\n\n")          file.write("\*\*Variable/fluctuate bills Minimum Days Between Bills are 28 days and the Maximum Days Between Bills are 35 days.\*\*\n \*\*\*Rare Extremes are 15 days(starting or ending services) and 45 days(In unusual circumstances, such as major system overhauls or natural disasters) between bills.\*\*\*\n\n")          for i in range(len(result)):              file.write("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")              file.write("Check " if int(planner["peopleNum"]) == 1 else f"Check for {result[i]['check']['name']} ")              file.write(f"on {result[i]['check']['date']} amount:${result[i]['check']['amount']}\n")              for j in range(len(result[i]['bills'])):                  if result[i]['bills'][j]['type'] == "variable/fluctuate":                      file.write(f"Variable/fluctuate bill:{result[i]['bills'][j]['name']} between {result[i]['bills'][j]['date']['min']}-{result[i]['bills'][j]['date']['max']} amount:${result[i]['bills'][j]['amount']}\n{result[i]['bills'][j]['note']}\n")                  else:                      file.write(f"Fixed bill:{result[i]['bills'][j]['name']} on {result[i]['bills'][j]['date']} amount:${result[i]['bills'][j]['amount']}\n{result[i]['bills'][j]['note']}\n")              file.write(f"Total bills:${result[i]['totalBills']}\n")              file.write(f"Over amount:${result[i]['overAmount']}\n" if result[i]['overAmount'] != "none" else "No over amount\n")              file.write("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n\n")          file.close()          #this print statement will help with letting the user know the information has been saved in the file          print("The information has been saved in output.txt file")  #run the program  output()  ---classesFunctions.py file---  class NumberHandler:  # This function checks if a string is a number with a comma for this class      def is\_number(s):          try:              withoutCommaList= s.split(",")              numberTogether= "".join(withoutCommaList)              float(numberTogether)              return True          except ValueError:              return False |

## PART 6: Your Completed Program

|  |
| --- |
| **Task**  Provide the IDE share link to your full program code.  **Requirements**   * The program must work correctly with all the comments included in the program.   **Inspiration**   * Check before submitting your touchstone that your final version of the program is running successfully. |
| <Provide the link to your program here>  https://onlinegdb.com/1hHjuUzTS |