Program 5

Raj Khanderia and Alexander Bean

ISA 201

05/02/2022

Table of Contents

Summary

The billing system created opens with a menu that first allows a user to enter billing data, open an ad-hoc report, or end the program. When “enter billing data” is selected, the program allows the user to enter employee data such as name, hourly rate, and hours worked for week 1 through week 4. Each input has a set of validation rules to ensure the intended information is being input. If invalid input is entered, the program will prompt for re-entry. With this data, the program determines if overtime is worked, and prints an invoice that shows the employee name, rate, average weekly hours, total billable hours, regular hours, overtime hours (if any), and total amount due. The program will then ask the user if they want to enter more data for another employee. If yes, the program starts again from the employee's name. If no, the program will return the user to the menu. As the employee billing data is being entered, it is being written to a text file called “billing.txt”. After employee data has been written, the program is capable of producing an ad-hoc report for the employer reading from the billing.txt file. The user has the option to open this ad-hoc report when returned to the menu. This ad-hoc report shows a list with each employee’s name, rate, hours worked for each week, total hours worked, and the total amount due. After the employee data list, the ad-hoc report also displays the total billable payment due, total billable hours, and average billable hours for all employees entered. All of the programs used in the billing system use functions that are stored in the billingModule.py file.

The billingModule.py file contains the functions that are called throughout the billing system. Many functions were created to orchestrate the running of the program. The functions provided to the team through class instruction are formMyList(), writeBillingFile(), resetBillingFile(), readWeeklyHours(), readHourlyRate(), and readEmployeeName(). These functions were provided to the team in previous assignments and class videos. They accept arguments with variables located in local program main()s, or hold strings with prompt instructions for the user (see appendices for function demonstrations) The team developed helper functions to be used in the integration of program4 and program5, which were overtimeCalc(), totalPayYesOT(), and totalPayNoOT(). The overtimeCalc() function consists of six lines and takes arguments for rate and totalHours from program 5. An overtime cutoff value constant from program4 is stated, and then an if-else suite is created. If the totalHours argument provided is greater than the cutoff variable, totalPayYesOT() is run with arguments including cutoff. If not, totalPayNoOT() is run with arguments excluding cutoff. Both functions are assigned to the *totalPay* variable, which is then returned.

These two helper functions left program4 unchanged while reducing the lines of code within program5. However, totalPayYesOT() does carry duplicate code from the body of program4.main(). The team spent time debating ways to remove this duplicate code while still producing the proper ad-hoc report output with the correct calculations. However, the duplicate code was left in place after the team determined the cost of overhauling the whole system just to replace four lines of duplicate code was not worth the effort. Other solutions provided, like creating a second .txt file written in program4 with the total hours and total billable pay calculations, were extremely inefficient and had poor design. The program operated as intended with a few duplicate lines and was not worth the cost of rewriting it (see appendices for function demonstration).

The program4.py file is the program that asks the user to input the employee data, determines whether there is overtime, and prints an invoice for all employees entered. This file starts by calling the resetBillingFile(), defining the constants, and defining the while loop validator. Then a while loop is created using the validator. In this while loop, helper functions readEmployeeName() and readHourlyRate() are called once, and readWeeklyHours() four times. These ask for input, validate it, and assign it to variables. After this, total hours and average hours are calculated, and writingBillingFile() is called to write the data to the billing.txt file. Then an if-else statement is used to determine whether there is or is not overtime. If there is calculated overtime, then the overtime rate, overtime hours, and overtime amount is then calculated using helper functions. The totalPayYesOT() function is then called and assigned to totalPay, and the invoice amount is calculated with that. Then the work statement and the invoice format are constructed. If no overtime is worked, then overtime hours and overtime amount are set to zero, and the totalPayNoOT() function is run and set to equal the *totalPay* variable. Then, the work statement and invoice format is constructed. The program will then print all the output. Finally, the program will ask the user if they want to input another employee. If there is the proper input, the program returns them to the top of the while loop. If not, the program ends.

The program5.py file creates the ad-hoc report using the data entered into program4.py, which was written to the billing.txt file. This program starts by defining new variables used in the program. The two that include specific design choices are the *dataPrint* and *rowSkip* variables. The *dataPrint* variable is used and added to throughout the program to build the statement for the data labels, starting with the heading when first defined. The purpose of creating *dataPrint* is to only have to create and define a single variable to create the ad-hoc report statement. Previous iterations included three variables that all added together, but the team found this to be inefficient and opted to use solely one variable to build the print statement. The single variable resulted in shorter and more efficient code. The *rowSkip* variable is an important design choice used to update arguments for the reading and formatting of data in *myList*. The variable increases by a value of 6, assigned to the *ROW_SKIP* variable, at the end of each iteration of the while loop. The addition of this value impacts the next iteration and alters the range of data read to a new “row”–a new employee and their set of data–in the data from *myList*. Like *dataPrint*, *rowSkip* was initially composed of three variables and was condensed into one for the sake of efficiency.

Program5’s processing begins and enters a try suite and sets the variable *myList* to the function formMyList(), which reads from the billing.txt file and removes the newlines. If there is no file to read in formMyList(), there is an exception clause for a “FileNotFoundError”. This results in a “No employees on file” string added onto the *dataPrint* variable, and the variable is printed. If there is a file to read in formMyList(), the file is read, formatted, and formed into a list. The program then enters a while loop that continues until the variable *rowSkip* is greater than the length of *myList*. The program establishes variables for each element read in the list, converting some to strings or floats depending on the required formatting for the ad-hoc report. When a full row of data is read, the *totalPay* variable is assigned to a value produced by the overtimeCalc() function, which accepts arguments for the employee rate and totalHours, which are calculated in the while loop. After the function is run and *totalPay* is assigned, the *dataPrint* variable then has calculated values read in the row of the list. Then, the *finalHours* and *finalPay* values are given additional values for the totalized section of the ad-hoc report, and the *rowSkip* variable is given a value to skip the row just read. The while loop begins again until the value of the *rowSkip* variable becomes greater than the length of myList. When this is done, dataPrint adds the formatted total billable due, total billable hours, and average billable hours data calculated and adds it to the variable. The dataPrint line is printed. Then, the user is brought back to the menuModule prompt.

The menuModule.py file is the program that creates the menu that is shown to the user, which allows data to be entered, or to open the ad-hoc report, or end the program. First, the loop control is set to true. Then a while loop is to run the menu. In the while loop, the options are shown to the user, and they are prompted to enter what they want to do. If “0” is entered, the loop control is changed to false and the program is ended. If “1” is entered, program4.main() is called. If “2” is entered, program5.main() is called. If no available options are entered, or an empty or invalid string is entered an error message is displayed and the user is prompted again.

After completing program5, the team reflected upon concepts learned throughout the assignment. Using the skills learned in the previous program assignments, the team learned how to successfully create unique helper functions that can be used in multiple programs. This made calculating values between programs easy and tested the team’s ability to code unique functions. Another lesson learned was understanding that while some of the written code may not be optimal, it is acceptable so long as the team is ready to defend the structure of the suboptimal code. The team understands that some parts of the code could be written differently and more efficiently, but the team can also defend those design choices and understand exactly why it was written that way.

Appendix 1: Source Code

program4.py

```
# -----------------------------------------------------------------
# Author: Alexander Bean &  Raj Khanderia
# Program: Program4
#
# Description:
# This is a payroll program used to determine the monthly billable
# pay to an employee, depending on their hours worked and their rate,
# including any overtime hours. Includes billing functions, an external
# write to a .txt file, and a loop to enter and write multiple employees
# -----------------------------------------------------------------

#input

import billingModule

def main():
    billingModule.resetBillingFile()
    again = 'y'
    TOTAL_WEEKS = 4.00
    OVERTIME_CUTOFF = 160.00
    RATE_INCREASE = 1.05
    while again == 'y':
        employee = billingModule.readEmployeeName("Employee Name: ")
        rate = billingModule.readHourlyRate("Hourly Rate: ")
        week1 = billingModule.readWeeklyHours("Enter hours worked for week 1: ")
        week2 = billingModule.readWeeklyHours("Enter hours worked for week 2: ")
        week3 = billingModule.readWeeklyHours("Enter hours worked for week 3: ")
        week4 = billingModule.readWeeklyHours("Enter hours worked for week 4: ")
        totalHours = week1 + week2 + week3 + week4
        averageHours = totalHours/TOTAL_WEEKS
#processing
        billingModule.writeBillingFile(employee, rate, week1, week2, week3, week4)
        if totalHours > OVERTIME_CUTOFF:
            overtimeRate = round(rate * RATE_INCREASE, 2)
            overtime = totalHours - OVERTIME_CUTOFF
            overtimeAmount = round(overtime * overtimeRate, 2)
            totalPay = billingModule.totalPayYesOT(rate, totalHours, OVERTIME_CUTOFF)
            invoiceAmount = totalPay - overtimeAmount
            workStatement = '\n'+employee+' worked '+ format(overtime, ',.2f')+ \
                            ' hours of overtime.\n'
            billableHours = 'Overtime Hours: '+\
                    format(overtime, ',.2f')+ ' @ $' +\
                    format(overtimeRate,',.2f')+ ' = $' +\
                    format(overtimeAmount, ',.2f')+\
                '\nRegular Hours: '+\
                    format(totalHours - overtime, ',.2f')+ ' @ $' +\
                    format(rate, ',.2f')+ ' = $' +\
                    format(invoiceAmount, ',.2f')
        else:
            overtime = 0.00
            overtimeAmount = 0.00
            totalPay = billingModule.totalPayNoOT(rate, totalHours)
            workStatement = '\n'+employee+' worked no overtime.\n'
            billableHours = 'Regular Hours: '+\
                    format(totalHours - overtime, ',.2f')+ ' @ $' +\
                    format(rate, ',.2f')+ ' = $' +\
                    format(totalPay, ',.2f')
#output
        print(workStatement, '\nInvoice')
        print('Resource: ',employee,'\tAverage weekly hours: ',\
                format(averageHours, ',.2f'))
        print('\nTotal billable hours: ',\
                format(totalHours, ',.2f'), '\tRate: $',\
                format(rate, ',.2f'), sep='')
        print(billableHours, '\nAmount Due: $',\
                format(totalPay, ',.2f'), sep='')

        again = input('\nEnter another employee? ("y" = yes): ')
```

program5.py

```
# ----------------------------------------------------------------
# Author: Alexander Bean & Raj Khanderia
# Program: Program5
#
# Description:
# This is a program that reads data from the "billing.txt" file
# created by Program4 and builds an ad-hoc report with all employees
# entered in the .txt file as a list. It shares functions with
# program4 nested in the overtimeCalc() function. "Except" clause
# failsafes are also put in place to prevent code from breaking.
# ----------------------------------------------------------------

import billingModule

def main():
#input
    NAME_LINE = 0
    RATE_LINE = 1
    FIRST_WEEK = 2
    FULL_ROW = 6
    finalHours = 0
    finalPay = 0
    rowSkip = 0
    dataPrint = '\nEmployee\tRate\tWeek 1\tWeek 2\tWeek 3'+\
                '\tWeek 4\tHours\tTotal\n'
#processing
    try:
        myList = billingModule.formMyList()
        while rowSkip < len(myList):
            totalHours = 0
            empName = myList[NAME_LINE + rowSkip] + '\t'
            empRate = format(float(myList[RATE_LINE + rowSkip]), ',.2f')
            empList = empName+'\t$'+str(empRate)
            for count in range(FIRST_WEEK, FULL_ROW):
                elem = format(float(myList[count + rowSkip]), ',.2f')
                totalHours += float(elem)
                empList += '\t' + str(elem)
            totalPay = billingModule.overtimeCalc(float(empRate), totalHours)
            dataPrint += empList+'\t'+\
                  str(format(totalHours, ',.2f'))+'\t$'+\
                  str(format(totalPay, ',.2f')) + '\n'
            finalHours += totalHours
            finalPay += totalPay
            rowSkip += FULL_ROW

        dataPrint += '\n'\
        'Total Billable Due:\t$'+str(format(finalPay, ',.2f'))+'\n'\
        'Total Billable Hours:\t'+str(format(finalHours, ',.2f'))+'\n'\
        'Average Billable Hours:\t'+ \
            str(format(finalHours/(rowSkip/FULL_ROW), ',.2f'))

    except FileNotFoundError:
        dataPrint += 'No employees on file'
#output
    print(dataPrint)
```

billingModule.py

```
def formMyList():

    infile = open("billing.txt", "r")

    myList = infile.readlines()

    infile.close()

    index = 0
    while index < len(myList):
        myList[index] = myList[index].rstrip("\n")
        index += 1

    return myList

def overtimeCalc(rate, totalHours):
    OVERTIME_CUTOFF = 160.00
    if totalHours > OVERTIME_CUTOFF:
        totalPay = totalPayYesOT(rate, totalHours, OVERTIME_CUTOFF)
    else:
        totalPay = totalPayNoOT(rate, totalHours)

    return totalPay

def totalPayYesOT(rate, totalHours, cutoff):
    RATE_INCREASE = 1.05
    overtimeRate = round(rate * RATE_INCREASE, 2)
    overtime = totalHours - cutoff
    overtimeAmount = round(overtime * overtimeRate, 2)
    totalPay = (cutoff * rate) + overtimeAmount

    return totalPay

def totalPayNoOT(rate, totalHours):
    totalPay = totalHours * rate

    return totalPay

def writeBillingFile(employee, rate, week1, week2, week3, week4):
    outfile = open("billing.txt", "a")
    outfile.write(employee+"\n"+str(rate)+"\n"+
                  str(week1)+"\n"+str(week2)+"\n"+
                  str(week3)+"\n"+str(week4)+"\n")
    outfile.close

def resetBillingFile():
    outfile = open("billing.txt","w").close

def readWeeklyHours(prompt):
    again = True

    while again:
        try:
            weekHours = float(input(prompt))
            if weekHours <35 or weekHours >80:
                print("Invalid number of hours, must be between 35 and 80.\n")
            else:
                again = False
        except ValueError:
            print("Error: Value must be numeric.\n")

    return float(weekHours)

def readHourlyRate(prompt):
    again = True

    while again:
        try:
            rate = float(input(prompt))
            if rate < 20:
                print("Invalid hourly rate, must be at least $20/hour.\n")
            else:
                again = False
        except ValueError:
            print("Error: Value must be numeric.\n")

    return float(rate)
```

menuModule.py

```
import program5, program4, billingModule

def main():
    loopControl = True
    while loopControl:
        try:
            print('\nBilling System Menu:\n')
            print('\t0 - End')
            print('\t1 - Enter billing data')
            print('\t2 - Display ad-hoc billing report')

            option = int(input('\nOption ==> '))

            if option == 0:
                loopControl = False
            elif option == 1:
                program4.main()
            elif option == 2:
                program5.main()
            else:
                print('Please enter an available option.\n')

        except ValueError:
            print('Please enter an available option.\n')

    print('\nMenu ended successfully')

main()
```

Appendix 2: IPO Charts

readWeeklyHours()		
Input	Processing	Output
Input prompt (String) presented to the user when soliciting user input	Allow the user to input hours worked for one week. All previous validity checking is to be included in the function.	Weekly hours (Float) worked for one of the four weeks

readHourlyrate()		
Input	Processing	Output
Input prompt (String) presented to the user when soliciting user input	Allow the user to input the hourly billing rate for the current employee. All previous validity checking is to be included in the function.	Billing rate (Float) for the current employee

readEmployeeName()		
Input	Processing	Output
Input prompt (String) presented to the user when soliciting user input	Allow the user to input the Employee Name for the current employee. All previous validity checking is to be included in the function.	Return the Employee Name as a String

resetBillingFile()		
Input	Processing	Output
No Input	Opens the billing.txt file in the write mode, and closes it immediately	No Output

writeBillingFile()		
Input	Processing	Output
Takes the inputs stored as employee, rate, and weeks 1-4	Opens the billing.txt file in the apprehend mode, and writes employee data to that file	No Output

totalPayNoOT()		
Input	Processing	Output
Takes the input stored as rate, and the calculator of totalHours	Calculates the totalPay by multiplying totalHours * rate	Returns totalPay for the current employee

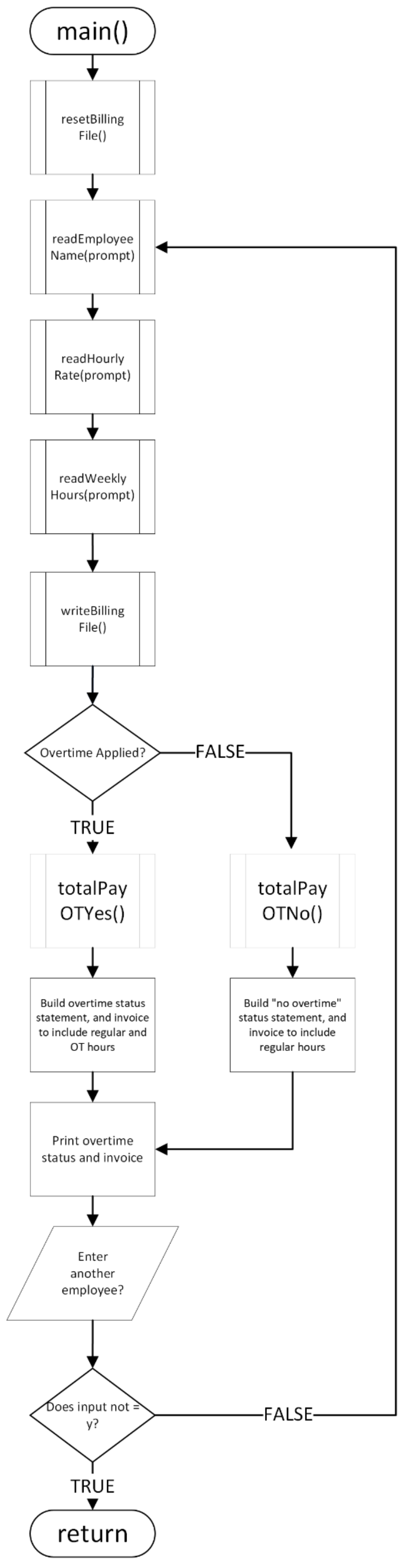
totalPayYesOT()		
Input	Processing	Output
Takes the input stored as rate, totalHours, and cutoff	Performs the overtime calculations for the employee	Returns totalPay for the current employee

overtimeCalc()		
Input	Processing	Output
Takes the input stored as rate and totalHours	Determines if overtime is worked. If overtime is worked, total pay is set to the function totalPayYesOT(). If overtime is not worked, totalPay is set the totalPayNoOT()	Returns totalPay for the current employee

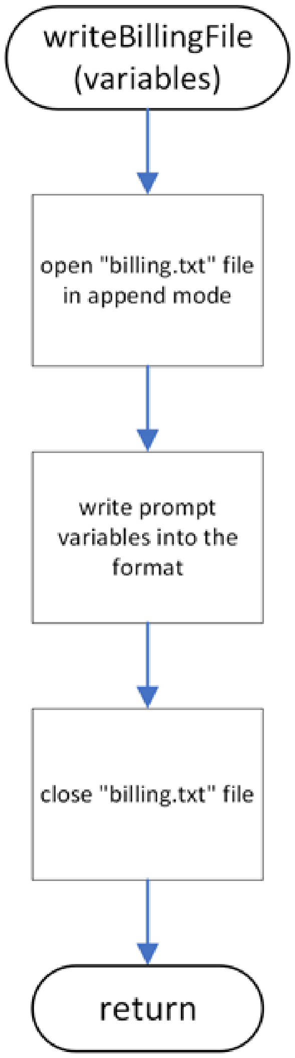
formMyList()		
Input	Processing	Output
No Input	Opens the billing.txt file in the read mode, then reads all lines into myList. myList then gets all the newlines stripped.	Returns myList for all the employee records

Appendix 3: Flowcharts

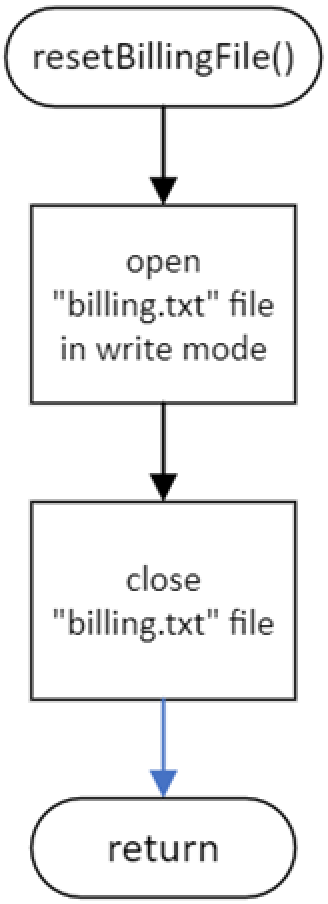
program4.main()


main()
resetBilling File()
readEmployee Name(prompt)
readHourly Rate(prompt)
readWeekly Hours(prompt)
writeBilling File()
Overtime Applied?
FALSE
TRUE
totalPay OTYes()
totalPay OTNo()
Build overtime status statement, and invoice to include regular and OT hours
Build "no overtime" status statement, and invoice to include regular hours
Print overtime status and invoice
Enter another employee?
Does input not = y?
FALSE
TRUE
return

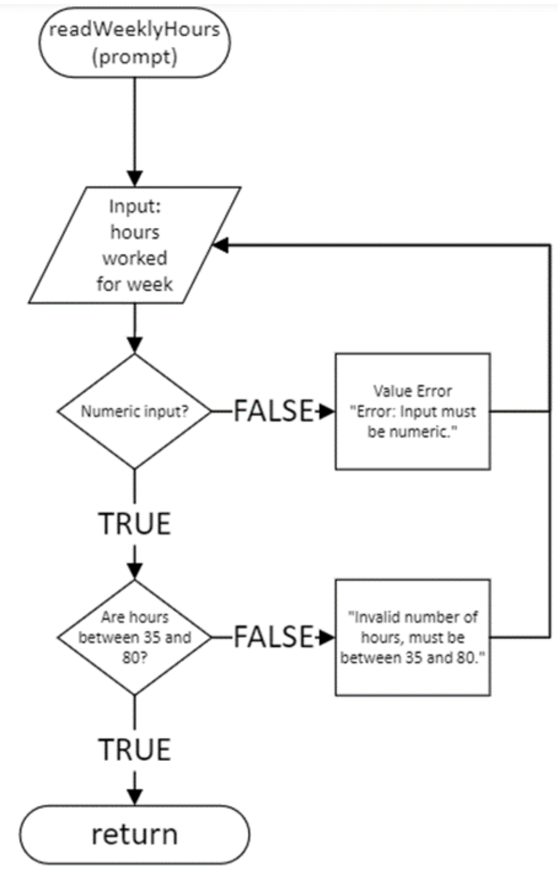
writeBillingFile()


writeBillingFile (variables)
open "billing.txt" file in append mode
write prompt variables into the format
close "billing.txt" file
return

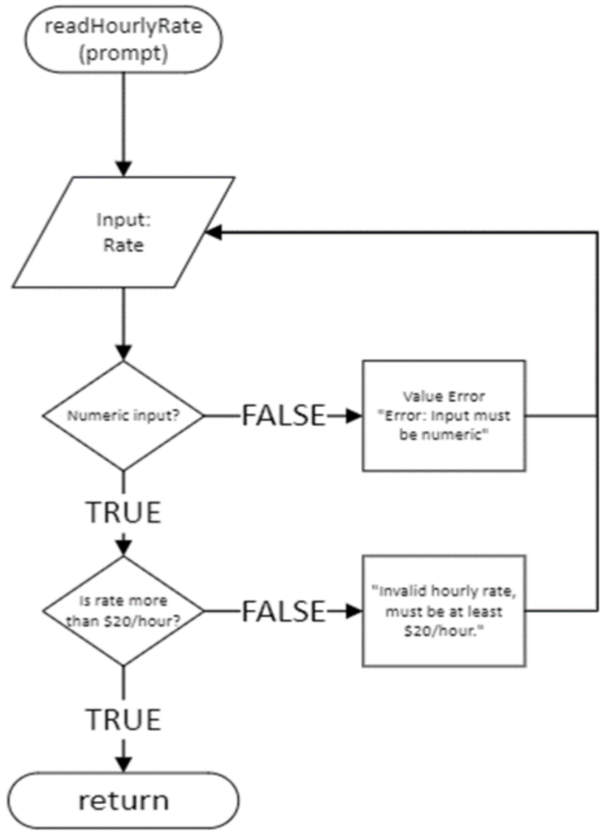
resetBillingFile



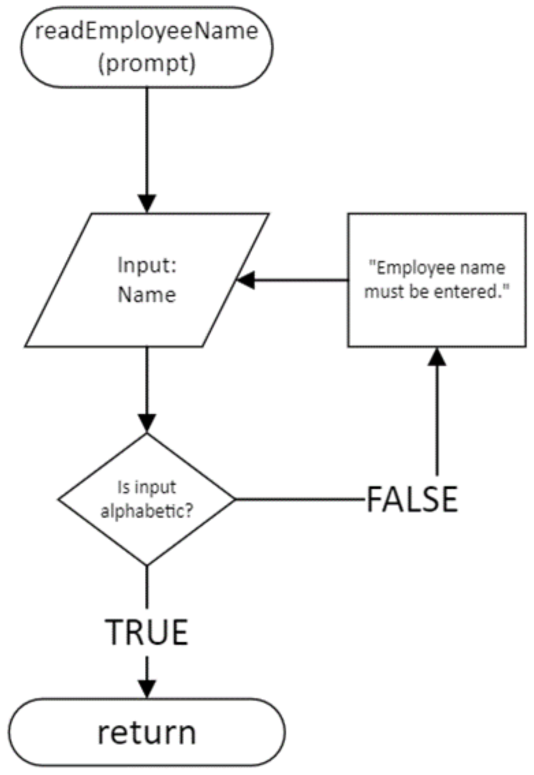
readWeeklyHours()


readWeeklyHours (prompt)
Input: hours worked for week
Numeric input?
FALSE
Value Error "Error: Input must be numeric."
TRUE
Are hours between 35 and 80?
FALSE
"Invalid number of hours, must be between 35 and 80."
TRUE
return

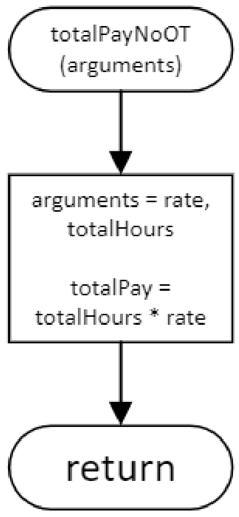
readHourlyRate()


readHourlyRate (prompt)
Input: Rate
Numeric input?
FALSE
Value Error "Error: Input must be numeric"
TRUE
Is rate more than $20/hour?
FALSE
"Invalid hourly rate, must be at least $20/hour."
TRUE
return

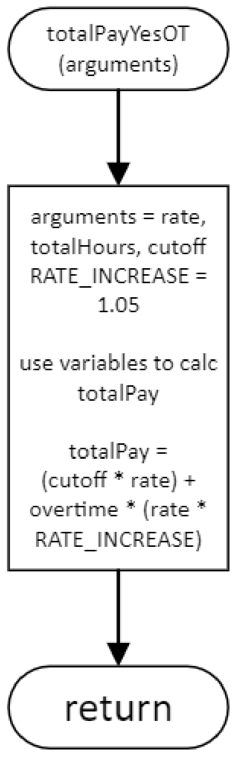
readEmployeeName()



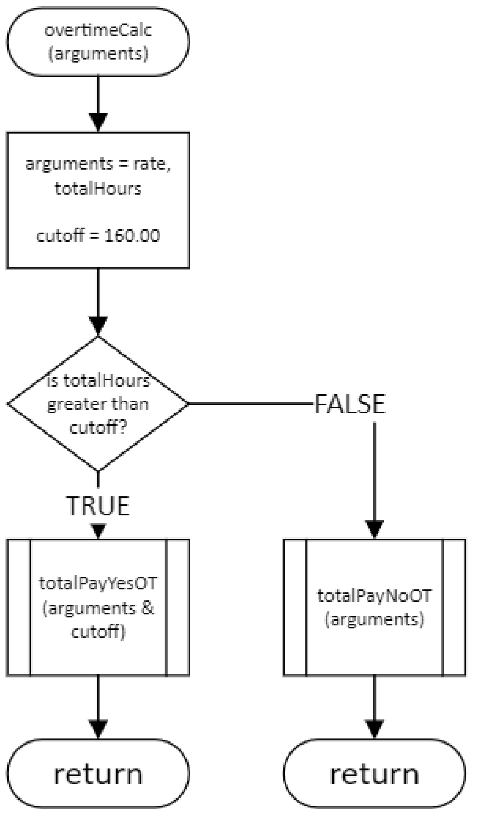
totalPayNoOT()



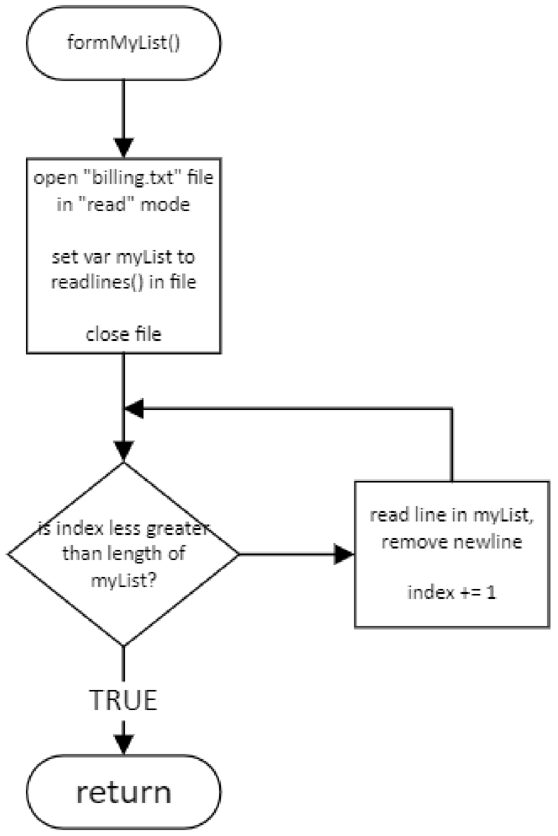
totalPayYesOT()


totalPayYesOT (arguments)
arguments = rate, totalHours, cutoff RATE_INCREASE = 1.05
use variables to calc totalPay
totalPay = (cutoff * rate) + overtime * (rate * RATE_INCREASE)
return

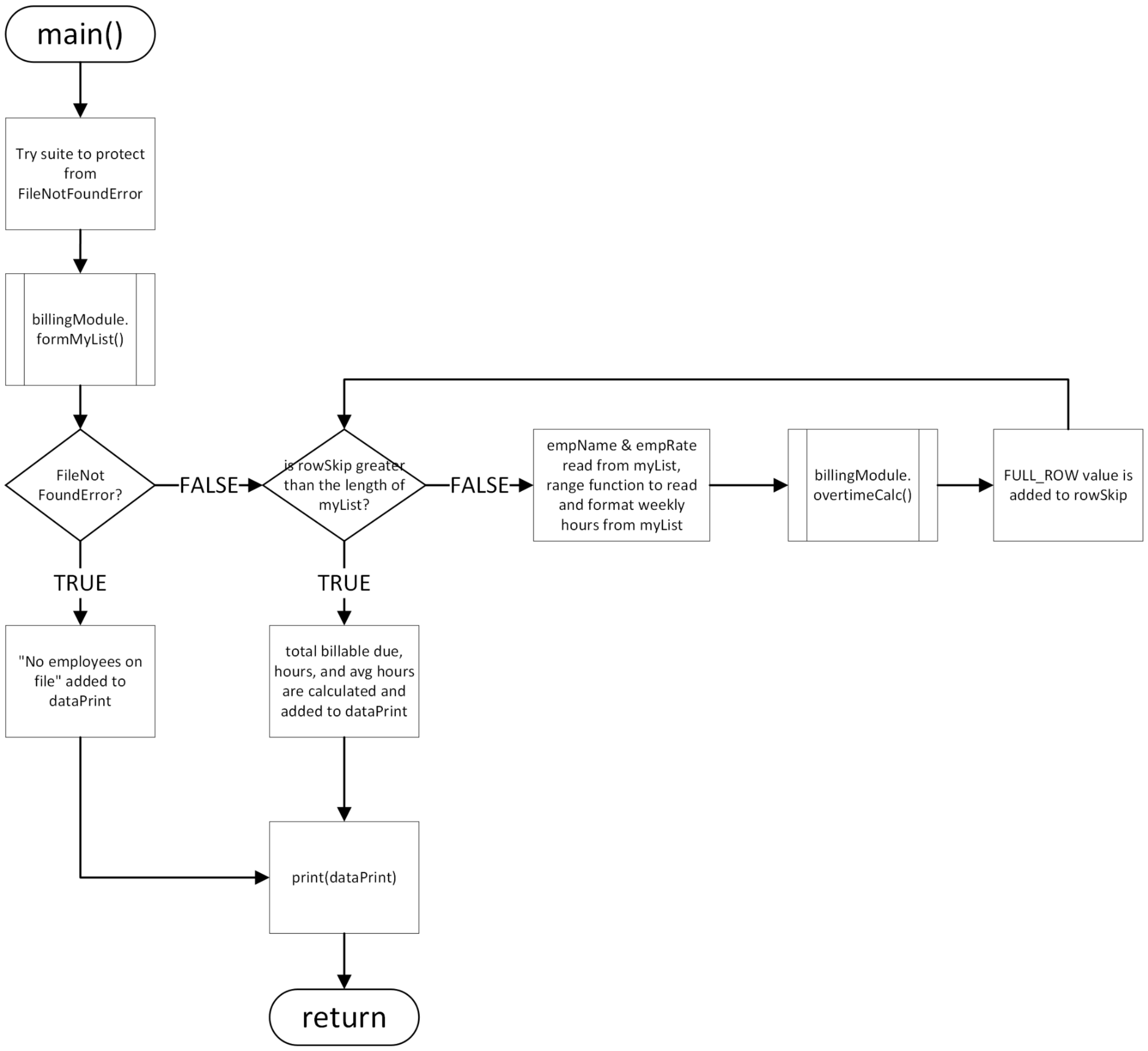
overtimeCalc()


overtimeCalc (arguments)
arguments = rate, totalHours
cutoff = 160.00
is totalHours greater than cutoff?
FALSE
TRUE
totalPayYesOT (arguments & cutoff)
totalPayNoOT (arguments)
return
return

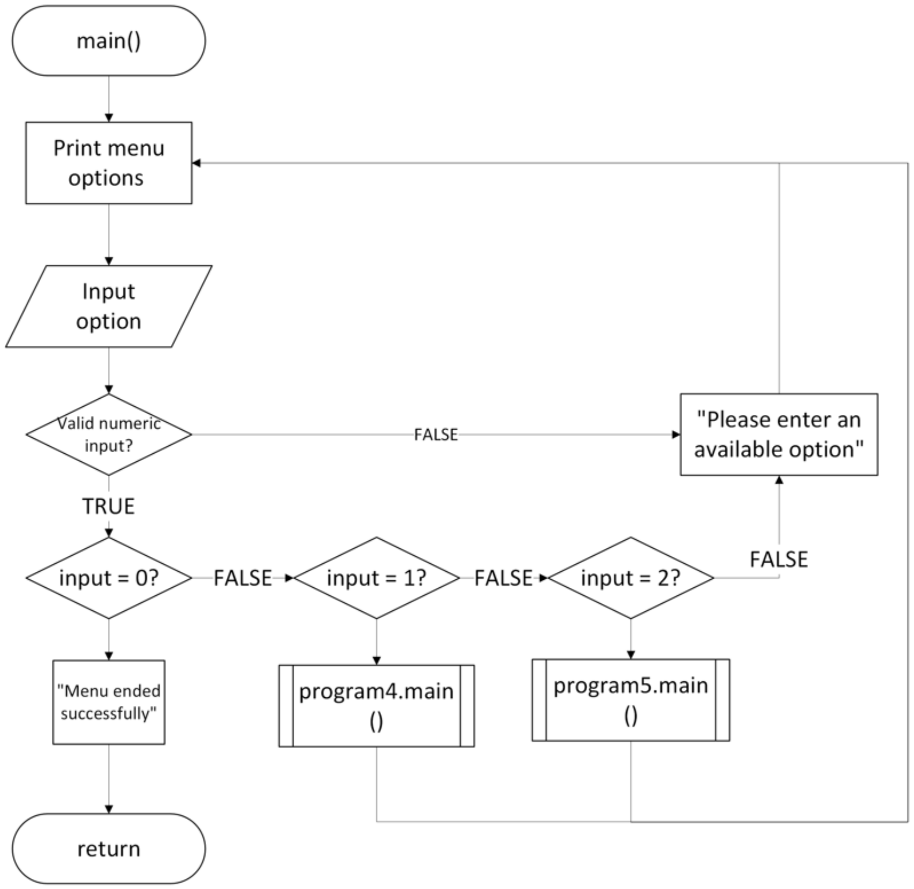
formMyList()


formMyList()
open "billing.txt" file in "read" mode
set var myList to readlines() in file
close file
is index less greater than length of myList?
read line in myList, remove newline
index += 1
TRUE
return

program5.main()


main()
Try suite to protect from FileNotFoundError
billingModule. formMyList()
FileNot FoundError?
FALSE
TRUE
"No employees on file" added to dataPrint
is rowSkip greater than the length of myList?
FALSE
TRUE
empName & empRate read from myList, range function to read and format weekly hours from myList
billingModule. overtimeCalc()
FULL_ROW value is added to rowSkip
total billable due, hours, and avg hours are calculated and added to dataPrint
print(dataPrint)
return

menuModule.main()



Appendix 4: Test Cases

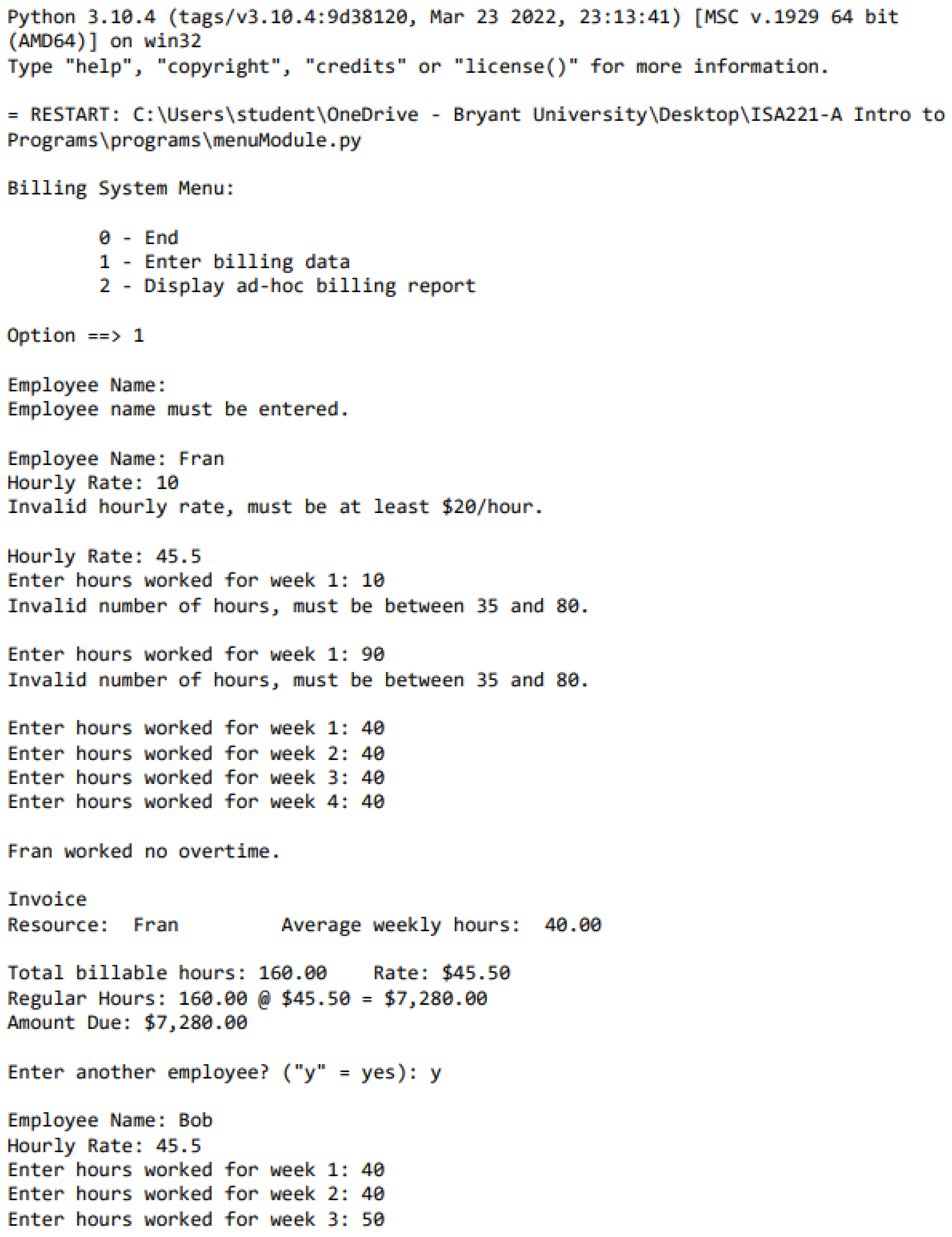
TEST CASES	DESCRIPTION	EXPECTED RESULT	ACHIEVED?
1	resetBillingFile() is ran at start of program, before while loop	.txt file data is opened in write and closed, wiping data in file	YES – Exhibit 6
2	Employee name input is entered as blank or a numeric value	Input validation loop – prints error message, prompts again	YES – Exhibit 1
3	Hourly rate is entered as blank or an alphabetic value	ValueError exception – prints error message, prompts again	YES – Exhibit 1
4	Hourly rate is entered as a numeric value below 20	Input validation loop – prints error message, prompts again	YES – Exhibit 1
5	Hours worked is entered as blank or an alphabetic value	ValueError exception – prints error message, prompts again	YES – Exhibit 1
6	Hours worked is entered as a numeric value outside of 35-80	Input validation loop – prints error message, prompts again	YES – Exhibit 1

7	Exhibit 1 - “Fran” Test: Employee Name: Fran Rate: $45.5 Weekly Hours: 40, 40, 40, 40	Total hours = 160 Avg. Hours = 40.00 OT? No Reg. Hours = 160 * 45.5 Amount Due $7,280.00	YES – Exhibit 1
8	Exhibit 2 - “Bob” Test: Employee Name: Bob Rate: $45.5 Weekly Hours: 40, 40, 50, 50.5	Total hours = 180.5 Avg. Hours = 45.12 OT? Yes - Hours: 20.5 OT Rate $47.77 OT Pay = $979.29 Reg. Hours = $7,280.00 Amount Due $8,259.29	YES – Exhibit 1
9	Exhibit 3 - “Joe” Test: Employee Name: Joe Rate: $45.5 Weekly Hours: 35, 40, 40, 40	Total hours = 155 Avg. Hours = 38.75 OT? No Reg. Hours = 155 * 45.5 Amount Due $7,052.50	YES – Exhibit 1

10	writeBillingFile() variables entered are name, rate, weeks 1-4	Variables write to .txt file, new line for each piece of data	YES – Exhibit 5
11	Enter another employee prompt input is NOT “y”	Program ends	YES – Exhibit 1
12	Enter another employee prompt is entered as “y”	Program loops back to readEmployeeName()	YES – Exhibit 1
13	User inputs second employee data (name, rate, weekly hours)	Variables are rewritten for new employee and calculations use new employee data	YES – Exhibit 1
14	User inputs data for multiple employees and is written to .txt	Data for all employees is written in .txt file, new line for each input for each employee	YES – Exhibit 5
15	User inputs ‘0’ in the menu	“Menu ended successfully”	YES - Exhibit 3
16	User inputs ‘1’ in the menu	program4.main() is run	YES - Exhibit 1
17	User inputs ‘2’ in the menu	program5.main() is run	YES - Exhibit 2

18	User inputs alphabetic or non-valid numeric input	“Please enter an available option” error message prints	YES - Exhibit 4
20	Ad-Hoc “Fran” Test - Prints Fran, $45.50, 40.00, 40.00, 40.00, 40.00, 160.00, and $7,280.00 in format	Fran, $45.50, 40.00, 40.00, 40.00, 40.00, 160.00, and $7,280.00	YES - Exhibit 2
21	Ad-Hoc “Bob” Test - Prints Bob, $45.50, 40.00, 40.00, 50.00, 50.50, 180.50, and $8,259.29 in format	Bob, $45.50, 40.00, 40.00, 50.00, 50.50, 180.50, and $8,259.29	YES - Exhibit 2
22	Ad-Hoc “Joe” Test - Prints Joe, $45.50, 35.00, 40.00, 40.00, 40.00, 155.00, and $7,042.50 in format	Joe, $45.50, 35.00, 40.00, 40.00, 40.00, 155.00, and $7,042.50	YES - Exhibit 2
23	Ad-Hoc statement prints proper calculation for all employees in the proper format	Total Billable Due: $22,591.79 Total Billable Hours: 495.50 Average Billable Hours: 165.17	YES - Exhibit 2
24	Ad-Hoc when there is no data written to the billing.txt file	“No employees on file”	YES - Exhibit 4

Exhibit 1:



```
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:\Users\student\OneDrive - Bryant University\Desktop\ISA221-A Intro to
Programs\programs\menuModule.py

Billing System Menu:

        0 - End
        1 - Enter billing data
        2 - Display ad-hoc billing report

Option ==> 1

Employee Name:
Employee name must be entered.

Employee Name: Fran
Hourly Rate: 10
Invalid hourly rate, must be at least $20/hour.

Hourly Rate: 45.5
Enter hours worked for week 1: 10
Invalid number of hours, must be between 35 and 80.

Enter hours worked for week 1: 90
Invalid number of hours, must be between 35 and 80.

Enter hours worked for week 1: 40
Enter hours worked for week 2: 40
Enter hours worked for week 3: 40
Enter hours worked for week 4: 40

Fran worked no overtime.

Invoice
Resource:  Fran         Average weekly hours:  40.00

Total billable hours: 160.00    Rate: $45.50
Regular Hours: 160.00 @ $45.50 = $7,280.00
Amount Due: $7,280.00

Enter another employee? ("y" = yes): y

Employee Name: Bob
Hourly Rate: 45.5
Enter hours worked for week 1: 40
Enter hours worked for week 2: 40
Enter hours worked for week 3: 50
Enter hours worked for week 4: 50.5

Bob worked 20.50 hours of overtime.

Invoice
Resource:  Bob  Average weekly hours:  45.12

Total billable hours: 180.50    Rate: $45.50
Overtime Hours: 20.50 @ $47.77 = $979.29
Regular Hours: 160.00 @ $45.50 = $7,280.00
Amount Due: $8,259.29

Enter another employee? ("y" = yes): y

Employee Name: Joe
Hourly Rate: 45.5
Enter hours worked for week 1: 35
Enter hours worked for week 2: 40
Enter hours worked for week 3: 40
Enter hours worked for week 4: 40

Joe worked no overtime.

Invoice
Resource:  Joe  Average weekly hours:  38.75

Total billable hours: 155.00    Rate: $45.50
Regular Hours: 155.00 @ $45.50 = $7,052.50
Amount Due: $7,052.50

Enter another employee? ("y" = yes): n

Billing System Menu:

        0 - End
        1 - Enter billing data
        2 - Display ad-hoc billing report

Option ==>
```

Exhibit 2:

```
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:\Users\student\OneDrive - Bryant University\Desktop\ISA221-A Intro to
Programs\programs\menuModule.py

Billing System Menu:

        0 - End
        1 - Enter billing data
        2 - Display ad-hoc billing report

Option ==> 2

Employee        Rate    Week 1  Week 2  Week 3  Week 4  Hours   Total
Fran            $45.50  40.00   40.00   40.00   40.00   160.00  $7,280.00
Bob             $45.50  40.00   40.00   50.00   50.50   180.50  $8,259.29
Joe             $45.50  35.00   40.00   40.00   40.00   155.00  $7,052.50

Total Billable Due:     $22,591.79
Total Billable Hours:   495.50
Average Billable Hours: 165.17

Billing System Menu:

        0 - End
        1 - Enter billing data
        2 - Display ad-hoc billing report

Option ==>
```

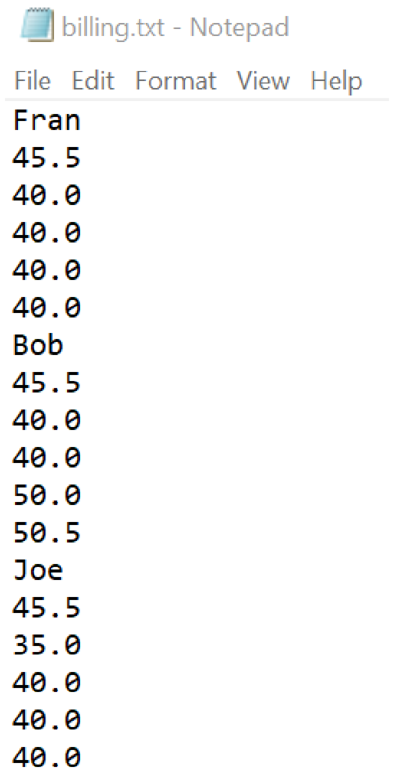
Exhibit 3:

```
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:\Users\student\OneDrive - Bryant University\Desktop\ISA221-A Intro to
Programs\programs\menuModule.py

Billing System Menu:

        0 - End
        1 - Enter billing data
        2 - Display ad-hoc billing report

Option ==> 0

Menu ended successfully
```

Exhibit 4:

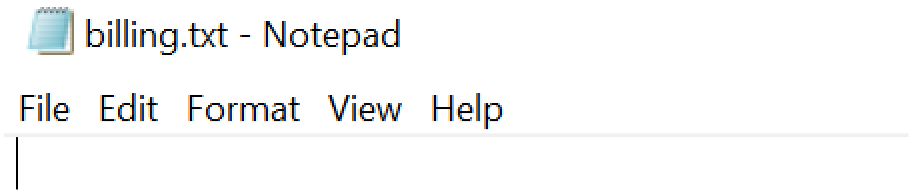
```
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MSC v.1929 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:\Users\student\OneDrive - Bryant University\Desktop\ISA221-A Intro to
Programs\programs\menuModule.py

Billing System Menu:

        0 - End
        1 - Enter billing data
        2 - Display ad-hoc billing report

Option ==> 3
Please enter an available option.


Billing System Menu:

        0 - End
        1 - Enter billing data
        2 - Display ad-hoc billing report

Option ==> 2

Employee        Rate    Week 1  Week 2  Week 3  Week 4  Hours   Total
No employees on file

Billing System Menu:

        0 - End
        1 - Enter billing data
        2 - Display ad-hoc billing report

Option ==>
```

Exhibit 5:

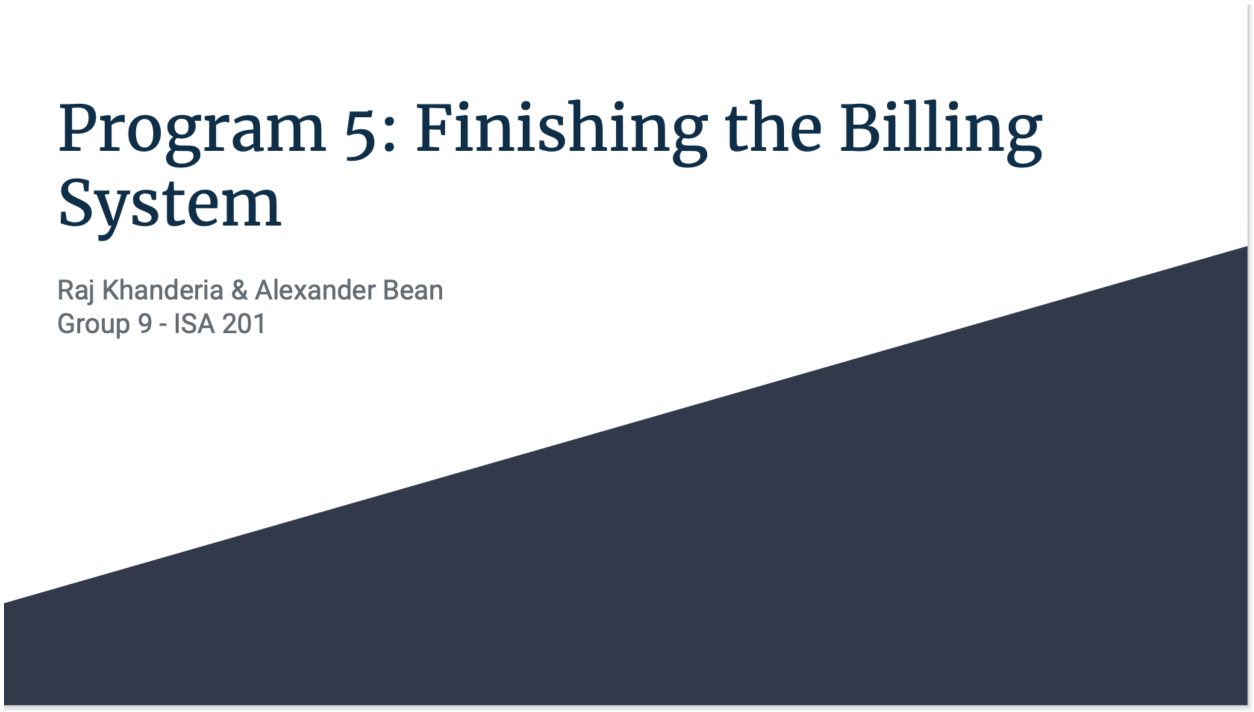


```
Fran
45.5
40.0
40.0
40.0
40.0
Bob
45.5
40.0
40.0
50.0
50.5
Joe
45.5
35.0
40.0
40.0
40.0
```

Exhibit 6:

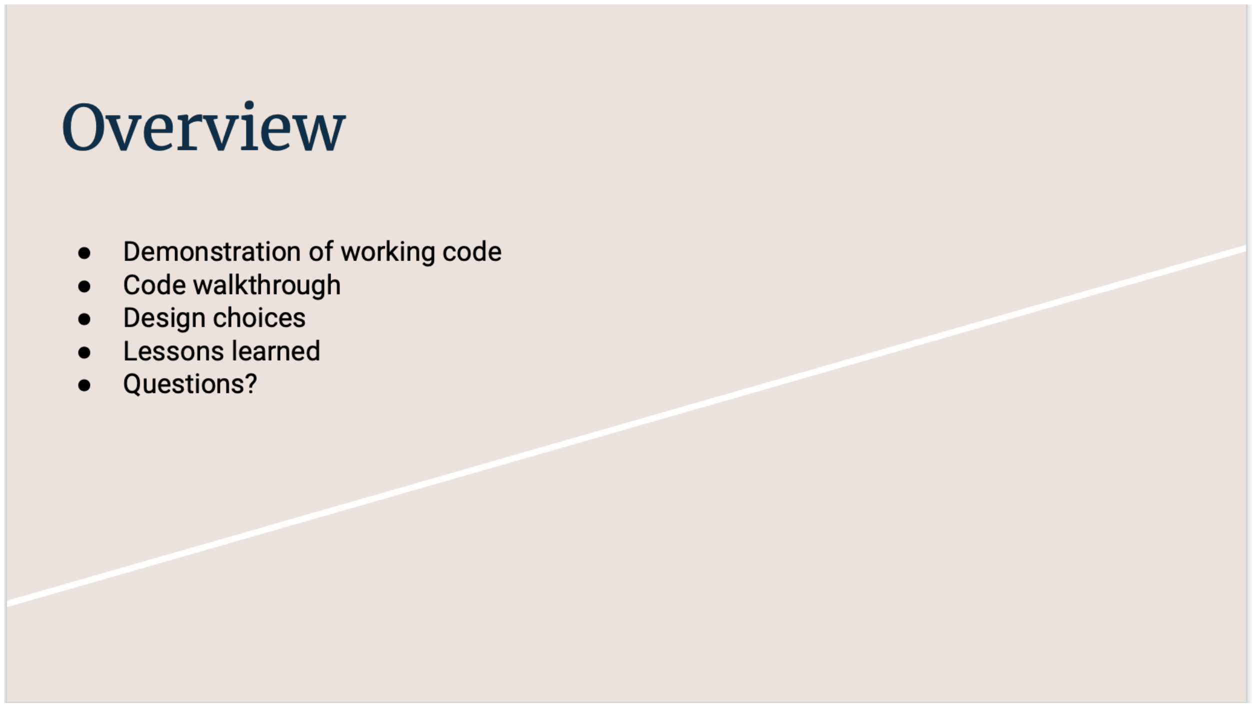


<u>Appendix 5: Presentation Slides</u>



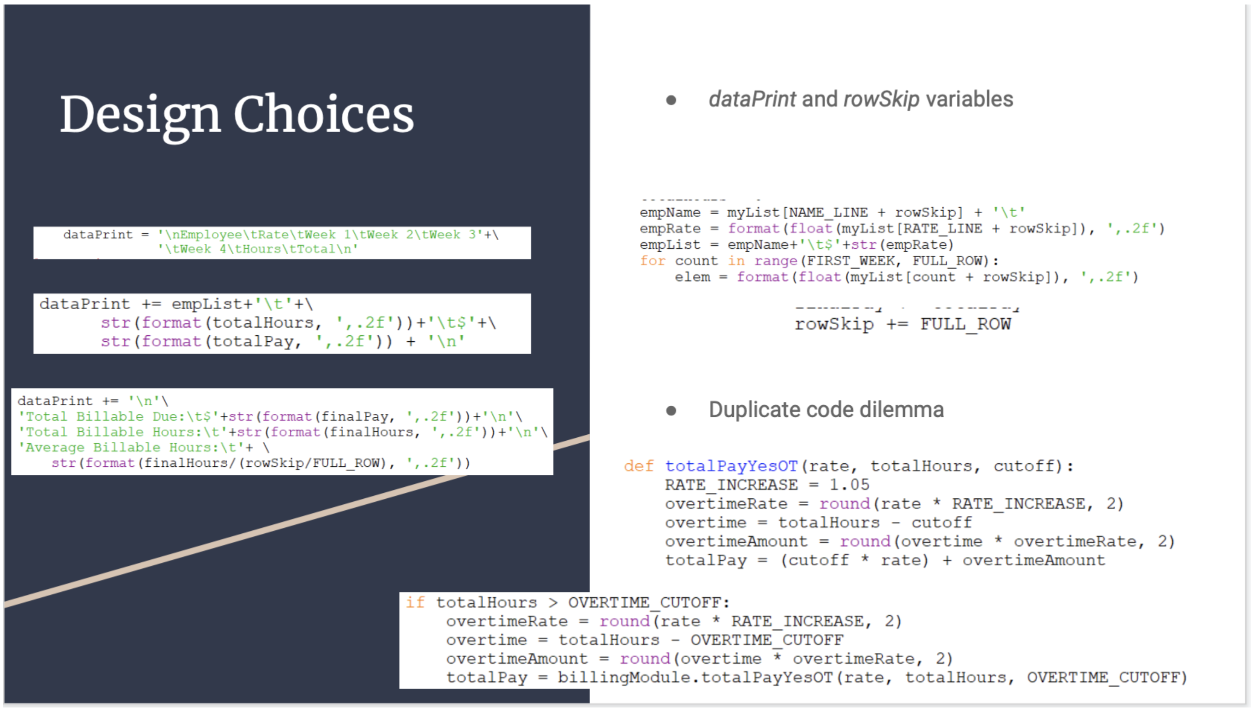
Program 5: Finishing the Billing System

Raj Khanderia & Alexander Bean
Group 9 - ISA 201



Overview

- Demonstration of working code
- Code walkthrough
- Design choices
- Lessons learned
- Questions?



Design Choices

```
dataPrint = '\nEmployee\tRate\tWeek 1\tWeek 2\tWeek 3'+\
            '\tWeek 4\tHours\tTotal\n'
```

```
dataPrint += empList+'\t'+\
        str(format(totalHours, ',.2f'))+'\t$'+\
        str(format(totalPay, ',.2f')) + '\n'
```

```
dataPrint += '\n'\
'Total Billable Due:\t$'+str(format(finalPay, ',.2f'))+'\n'\
'Total Billable Hours:\t'+str(format(finalHours, ',.2f'))+'\n'\
'Average Billable Hours:\t'+ \
    str(format(finalHours/(rowSkip/FULL_ROW), ',.2f'))
```

- *dataPrint* and *rowSkip* variables

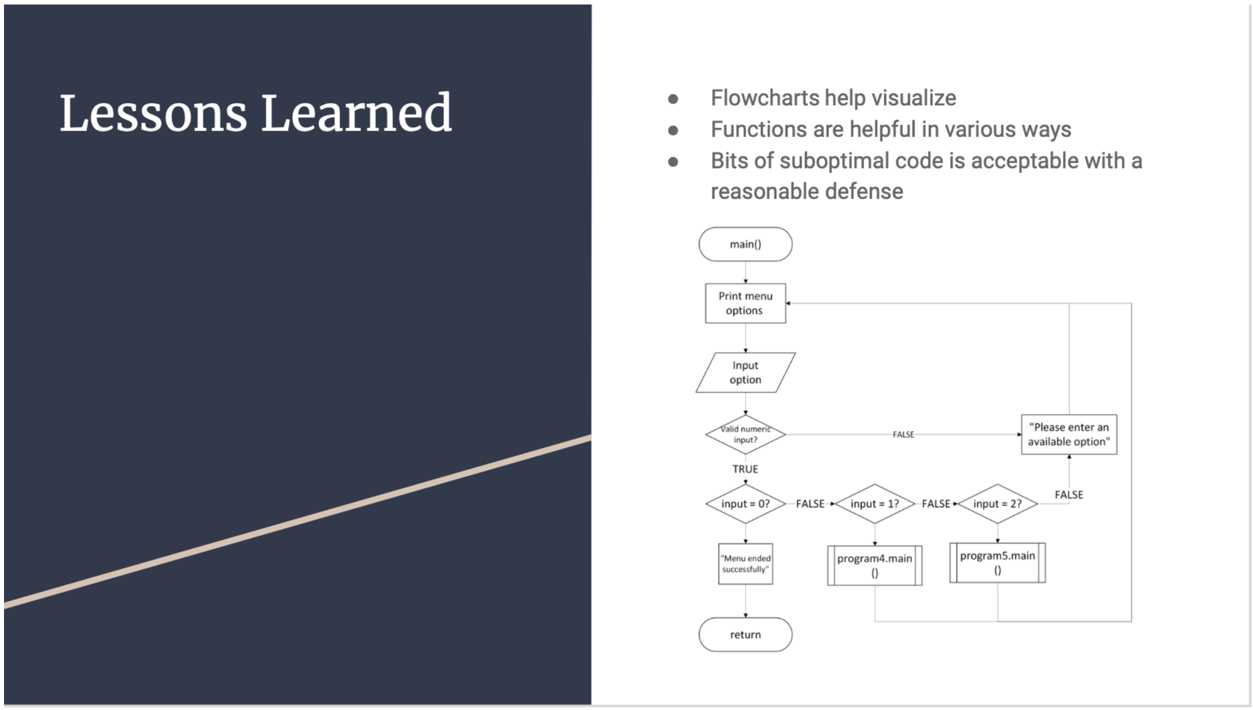
```
empName = myList[NAME_LINE + rowSkip] + '\t'
empRate = format(float(myList[RATE_LINE + rowSkip]), ',.2f')
empList = empName+'\t$'+str(empRate)
for count in range(FIRST_WEEK, FULL_ROW):
    elem = format(float(myList[count + rowSkip]), ',.2f')
```

```
rowSkip += FULL_ROW
```

- Duplicate code dilemma

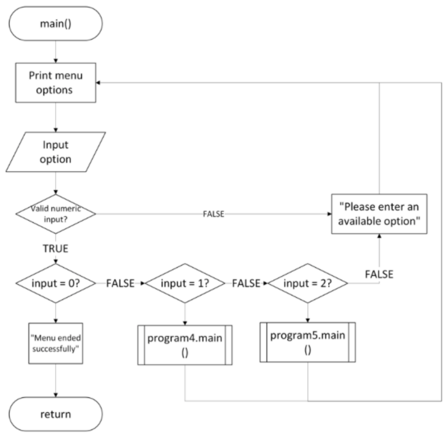
```
def totalPayYesOT(rate, totalHours, cutoff):
    RATE_INCREASE = 1.05
    overtimeRate = round(rate * RATE_INCREASE, 2)
    overtime = totalHours - cutoff
    overtimeAmount = round(overtime * overtimeRate, 2)
    totalPay = (cutoff * rate) + overtimeAmount
```

```
if totalHours > OVERTIME_CUTOFF:
    overtimeRate = round(rate * RATE_INCREASE, 2)
    overtime = totalHours - OVERTIME_CUTOFF
    overtimeAmount = round(overtime * overtimeRate, 2)
    totalPay = billingModule.totalPayYesOT(rate, totalHours, OVERTIME_CUTOFF)
```



Lessons Learned

- Flowcharts help visualize
- Functions are helpful in various ways
- Bits of suboptimal code is acceptable with a reasonable defense



Questions?

Appendix 6: Verification of Billing entry program

View Appendix 4: Test Cases, Exhibit 1

Appendix 7: Verification of Ad-Hoc report program

View Appendix 4: Test Cases, Exhibit 2