SODV1101 – Programming Fundamentals
ILA 5.1 – Repetition Structures

Name: ____________________________

1–Repetition Structures Pseudocode: Condition Controlled Loops

Critical Review

A repetition structure causes a statement or set of statements to execute repeatedly.

Repetition structures are used to perform the same task over and over.

Repetition structures are commonly called loops

A condition-controlled loop uses a true/false condition to control the number of times that it repeats.

The general structure of a While loop with a condition-controlled statement is:

```
//Declare loop control variable
While condition
     Statement
     Statement
     Etc.
     //Ask Question that changes the loop control variable
End While
```

The general structure of a Do While loop with a condition-controlled statement is:

```
//Declare loop control variable
Do
     Statement
     Statement
     Etc.
     //Ask Question that changes the loop control variable
While Condition
```

SODV1101 – Programming Fundamentals
ILA 5.1 – Repetition Structures

This lab requires you to implement a condition controlled loop.

Step 1: Examine the following main Module from Lab 4.2. Loops are commonly used to call modules multiple times. The best design is to use a loop around the module calls in Main.

```
Module main ()
     //Declare local variables
     Declare Real monthlySales
     Declare Real storeAmount
     Declare Real empAmount
     Declare Real salesIncrease

     //Function calls
     Call getSales(monthlySales)
     Call getIncrease(salesIncrease)
     Call storeBonus(monthlySales, storeAmount)
     Call empBonus(salesIncrease, empAmount)
     Call printBonus(storeAmount, empAmount)
End Module
```

Step 2: In the space provided, create a loop control variable named keepGoing of the data type string. Initialize this variable to "y". (Reference: Modularizing the Code in the Body of a Loop, page 172).

Step 3: In the space provided, write a while statement.

```
Module main ()
     //Declare local variables
     Declare Real monthlySales
     Declare Real storeAmount
     Declare Real empAmount
     Declare Real salesIncrease
    _____________________________________________________

     //Function calls
     While _________________________________
          Call getSales(monthlySales)
          Call getIncrease(salesIncrease)
          Call storeBonus(monthlySales, storeAmount)
          Call empBonus(salesIncrease, empAmount)
          Call printBonus(storeAmount, empAmount)
          Display "Do you want to run the program again?
          (Enter y for yes)."
          Input _________________________________
     End While
End Module
```

Step 4: In the space provided, create a loop control variable named keepGoing of the data type string. Initialize this variable to “y”. (Reference: Writing a Do-While Loop, page 175).

Step 5: In the space provided, write a do while statement.

```
Module main ()
     //Declare local variables
     Declare Real monthlySales
     Declare Real storeAmount
     Declare Real empAmount
     Declare Real salesIncrease
     ____________________________________________________

     //Function calls
     Do
          Call getSales(monthlySales)
          Call getIncrease(salesIncrease)
          Call storeBonus(monthlySales, storeAmount)
          Call empBonus(salesIncrease, empAmount)
          Call printBonus(storeAmount, empAmount)
          Display "Do you want to run the program again?
          (Enter y for yes)."
          Input _________________________________
     While ___________________________________________
End Module
```

2 –Repetition Structures Pseudocode: Counter Controlled Loops

Critical Review

A count-controlled loop repeats a specific number of times.

The loop keeps a count of the number of times that it iterates, and when the count reaches a specified amount the loop stops.

A variable, known as a counter variable, is used to store the number of iterations that it has performed.

The three actions that take place are initialization, test, and increment.

- Initialization: Before the loop begins, the counter variable is initialized to a starting value.
- Test: The loop tests the counter variable by comparing it to a maximum value.
- Increment: To increment a variable means to increase its value. This is done by adding one to the loop control variable.

Any loop can be used with a count-controlled loop.

A running total is a sum of numbers that accumulates with each iteration of a loop. The variable used to keep the running total is called an accumulator.

This lab requires you to write a complete program using a condition controlled loop, a counter controlled loop, and an accumulator. The program is a follows:

```
Write a program that will allow a grocery store to keep
track of the total number of bottles collected for seven
days.  The program should allow the user to enter the total
number of bottles returned for seven days.  The program
will calculate the total number of bottles returned for the
week and the amount paid out (the total returned times .10
cents).  The output of the program should include the total
number of bottles returned and the total paid out.
```

Step 1: In the pseudocode below, declare the following variables under the documentation for Step 1.

- A variable called totalBottles that is initialized to 0
  - This variable will store the accumulated bottle values
- A variable called counter and that is initialized to 1
  - This variable will control the loop
- A variable called todayBottles that is initialized to 0
  - This variable will store the number of bottles returned on a day
- A variable called totalPayout that is initialized to 0
  - This variable will store the calculated value of totalBottles times .10

- A variable called keepGoing that is initialized to "y"
    - This variable will be used to run the program again

Step 2: In the pseudocode below, make calls to the following functions under the documentation for Step 2.

- A function call to getBottles that passes totalBottles, todayBottles, and counter.
- A function called calcPayout that passes totalPayout and totalBottles.
- A function called printInfo that passes totalBottles and totalPayout

Step 3: In the pseudocode below, write a condition controlled while loop around your function calls using the keepGoing variable under the documentation for Step 3.

Complete Steps 1-3 below:

```
Module main ()

    //Step 1: Declare variables below
    ___________________________________________________
    ___________________________________________________
    ___________________________________________________
    ___________________________________________________
    ___________________________________________________

    //Step 3: Loop to run program again
    While _________________________________
         //Step 2: Call functions
         _____________________________________________
         _____________________________________________
         _____________________________________________

         Display "Do you want to run the program again?
         (Enter y for yes)."
         Input _________________________________
    End While
End Module
```

Step 4: In the pseudocode below, write the missing lines, including:

a. The missing parameter list
b. The missing condition (Hint: should run seven iterations)
c. The missing input variable
d. The missing accumulator
e. The increment statement for the counter

```
//getBottles module
Module getBottles(a.________________________________________)
     While b.________________________________
          Display "Enter number of bottles returned for the
          day:"
          Input c.______________________________
          d. ___________________________________
          e. _____________________________________

     End While
End Module
```

Step 5: In the pseudocode below, write the missing lines, including:

a. The missing parameter list
b. The missing calculation

```
//getBottles module
Module calcPayout(a.________________________________________)
     totalPayout = 0 //resets to 0 for multiple runs
     b. _______________________________________________
End Module
```

Step 6: In the pseudocode below, write the missing lines, including:

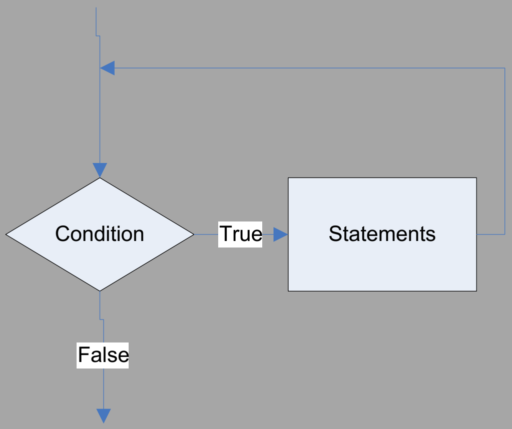
a. The missing parameter list
b. The missing display statement
c. The missing display statement

```
//printInfo module
Module printInfo(a.________________________________________)
     b. _______________________________________________
     c. _______________________________________________
End Module
```

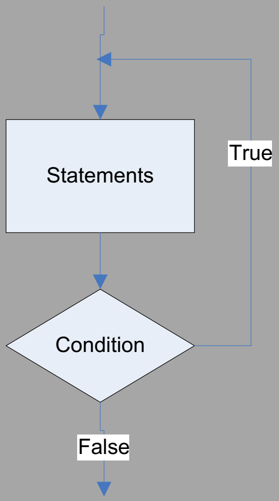
3 – Flowcharts

Critical Review

In a while loop, the question is asked first. After the statements process, the control goes back above the condition.



In a do-while loop, the question is asked last. The statements always process at least one time.



In Raptor, you can place the modules before or after the condition depending on whether you want to use a do-while or a while loop.

This lab requires you to convert your pseudocode in Lab 5.2 to a flowchart. Use an application such as Raptor or Visio.

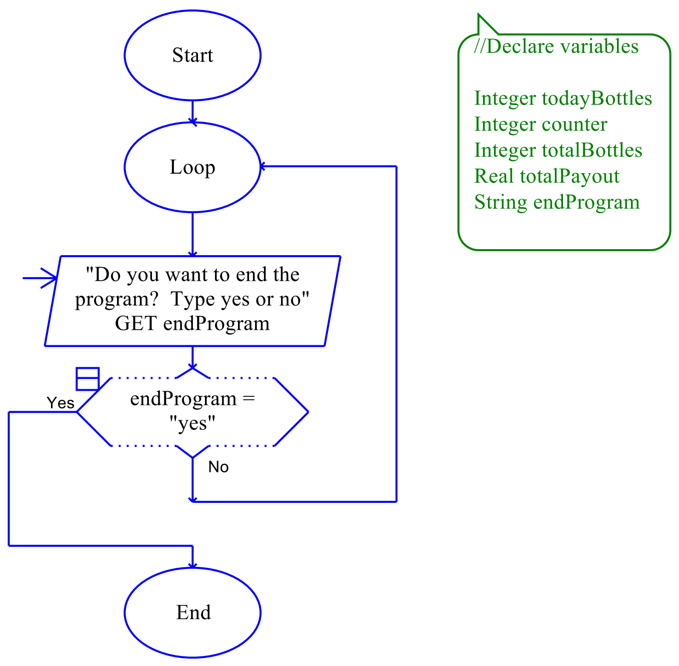
Step 1: Start Raptor and save your document as *Lab 5-3*. The *.rap* file extension will be added automatically. Start by adding a Comment box that declares your variables. The only variable from Lab 5.2 that is different is the keepGoing variable. Name this endProgram instead.

Step 2: Click the Loop symbol and drag and drop it between the Start and the End symbol.

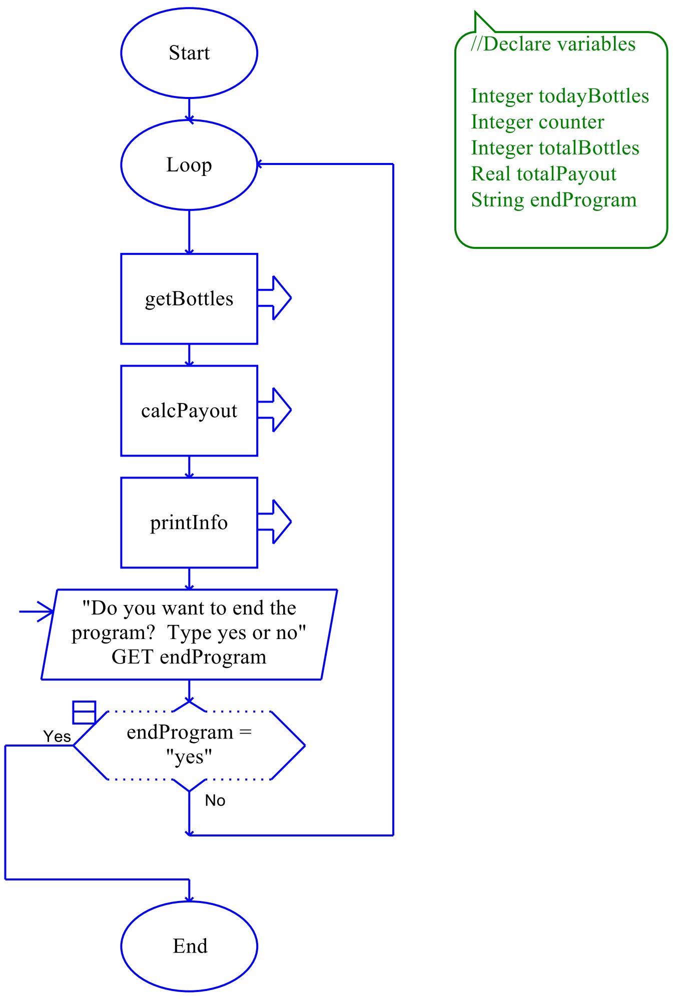
Step 3: Click the Input symbol and drag and drop it between the Loop symbol and the Diamond symbol.

Step 4: Double click the Input symbol and ask the question "Do you want to end the program? Enter yes or no:". Store the answer in the endProgram variable.

Step 5: Double click the Diamond symbol, and type endProgram = "yes" as the condition. When the program executes, the user must type "yes" exactly, in order for the program to end. Now, main should look as the following:



Step 6: The next step in your flowchart should be to call your methods. Add your modules under the Loop oval. Be sure to click yes to add new tabs for each module. Now, main should look as the following:



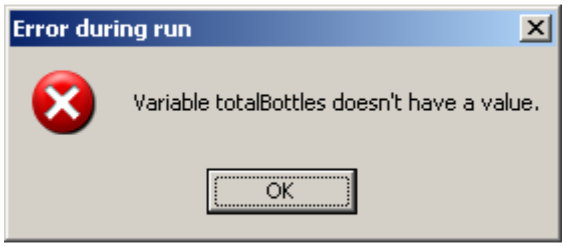
Step 7: Click on the getBottles tab. Add a Loop symbol between the Start and End symbols. Double click the Diamond symbol and enter the condition as counter >7.

Step 8: Add an Input symbol and add the code "Enter the number of bottles returned for today:". Store the value in the todayBottles variable.

Step 9: Add an Assignment symbol next and set totalBottles to totalBottles + todayBottles.

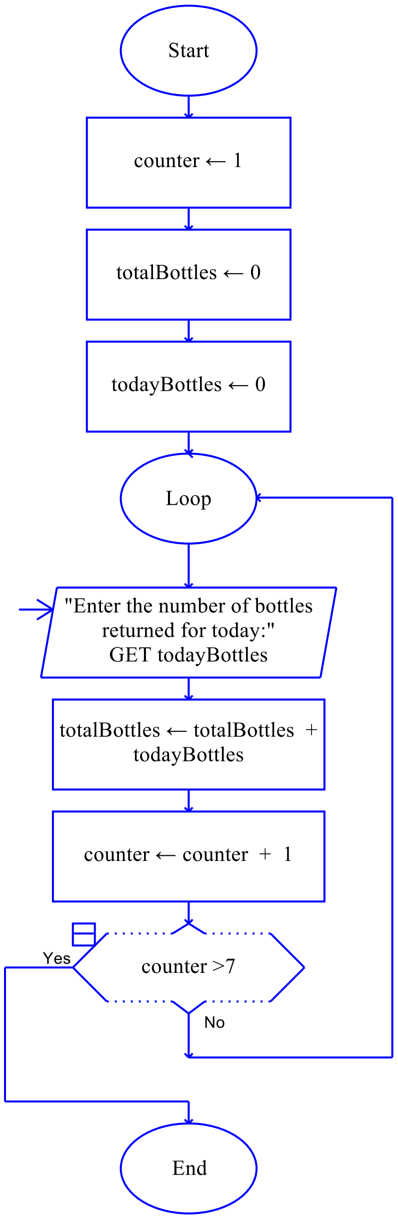
Step 10: Add another Assignment symbol next and set counter to counter + 1.

Step 11: Save your program and try running it. You'll notice an error occur when the loop starts processing in the getBottles module. This is because totalBottles does not have a starting value.



Variable totalBottles doesn't have a value.

Step 12: To fix the error, to set the counter to 1, and to reset the todayBottles back to 0 for multiple repetitions, add three Assignment symbols above the Loop symbol. In one symbol, set counter to 1. In the other, set totalBottles to 0. In the other, set todayBottles to 0. Your getBottles module should look as follows:


Start
counter ← 1
totalBottles ← 0
todayBottles ← 0
Loop
"Enter the number of bottles returned for today:" GET todayBottles
totalBottles ← totalBottles + todayBottles
counter ← counter + 1
counter >7
Yes
No
End

Step 13: Click the calcPayout module and add an Assignment symbol. Set totalPayout to totalBottles times .10.

Step 14: Click the printInfo module and add two Output symbols that print the total bottles returned and the total amount paid out.

Step 15: Test your program against the following values. If there is an error, go back through the steps to locate the problem.

Input Values	Expected Output
Seven days of bottles: 346 238 638 890 1035 899 536	The total number of bottles collected were: 4582 The total amount paid out is $458.2000

Step 16: The final step is to insert your finished flowchart in the space below.

PASTE FLOWCHART HERE

4 – Python Code

Critical Review

In Python, you use the while statement to write a condition-controlled loop. The loop has two parts: (1) a condition that is tested for a true or false value, and (2) a statement or set of statements that is repeated as long as the condition is true.

A while loop can also be used for count-controlled loops.

Here is the general format of the `while` loop in Python:

```
while condition:
        statement
        statement
        etc.
```

Since the while loop is a pre-test, it is important to initialize your loop control variable to a starting value so that the first iteration will be true.

As with all loops, be sure to change the loop control variable either by incrementing or asking a question.

The goal of this lab is to convert the Bottle Return program to Python code.

Step 1: Start the IDLE Environment for Python. Prior to entering code, save your file by clicking on File and then Save. Select your location and save this file as *Lab5-4.py*. Be sure to include the .py extension.

Step 2: Document the first few lines of your program to include your name, the date, and a brief description of what the program does.

Step 3: Start your program with the following code for main:

```
#Lab 5-4 The Bottle Return Program

#the main function
def main():

#calls main
main()
```

Step 4: Python only supports While loops, so endProgram must be initialized to 'no'. Under `def main():`, create a variable called endProgram and set it to 'no' such as:

```
        endProgram = 'no'
```

Step 5: The next step is to write a while statement with the condition of endProgram == 'no':. The statement should be aligned with the statement in Step 4. The code should be as follows:

```
    while endProgram == 'no':
```

Step 6: The code inside of the while statement should be tabbed over and include your function calls. The function getBottles() will return totalBottles so the call should be set to that variable. The function calcPayout should pass totalBottles as an argument and will return totalPayout from the function. The function printInfo should pass totalBottles and totalPayout as arguments. The code should look as follows:

```
    totalBottles = getBottles()
    totalPayout = calcPayout(totalBottles)
    printInfo(totalBottles, totalPayout)
```

Step 7: The next step is to modify the loop control variable. This is done with a simple raw_input statement such as:

```
    endProgram = raw_input('Do you want to end the program?
    (Enter yes or no): ')
```

Step 8: The next function to code is getBottles. Write a definition for getBottles that accepts no arguments. The code should look as follows:

```
    #this function will get the number of bottles returned
    def getBottles():
```

Step 9: The first step in your function should be to set your variables to values. In Python and most programming languages, in order for variables to be used, they need to have a starting value. This also allows for a reset of the variables when the program runs again. Set totalBottles and todayBottles to 0 and counter to 1. Your code should look as follows:

```
   totalBottles = 0
   todayBottles = 0
   counter = 1
```

Step 10: Write a while loop with the condition of counter <= 7. This code should look as follows:

```
    while counter <= 7:
```

Step 11: Inside the while loop, write a statement that allows the user to enter the number of bottles for today. This code should look as follows:

```
todayBottles = input('Enter number of bottles for today: ')
```

Step 12: Next, write the accumulator statement. This code should look as follows:

```
totalBottles = totalBottles + todayBottles
```

Step 13: The last statement inside the loop should increment counter by one so the loop will end after seven iterations. This code should look as follows:

```
counter = counter + 1
```

Step 14: The final statement in the getBottles function is to return totalBottles back to main. This code should look as follows:

```
return totalBottles
```

Step 15: Create a function definition for calcPayment that accepts totalBottles in the parameter list. This function should first reset totalPayout to 0. This is done so that on multiple iterations of the program, totalPayout is reset to 0. The second step in this function is to calculate totalPayout as totalBottles times .10. The last step is to return totalPayout. Your code should look as follows:

```
#this function will calculate the payout
def calcPayout(totalBottles):
    totalPayout = 0
    totalPayout = totalBottles * .10
    return totalPayout
```

Step 16: The final function in this program is printInfo. This function accepts two variables in the parameter list so that it can display the total number of bottles returned and the total amount paid out. Your code should look as follows:

```
#this function will display the information
def printInfo(totalBottles, totalPayout):
    print 'The total number of bottles collected is',
    totalBottles
    print 'The total paid out is $', totalPayout
```

Step 17: Click Run and Run Module to see how your program processes. Test the following values to verify the expected output.

```
>>>

Enter number of bottles for today: 346
Enter number of bottles for today: 238
Enter number of bottles for today: 638
Enter number of bottles for today: 890
Enter number of bottles for today: 1035
Enter number of bottles for today: 899
Enter number of bottles for today: 536
The total number of bottles collected is 4582
The total paid out is $ 458.2

Do you want to end the program? (Enter yes or no): no

Enter number of bottles for today: 425
Enter number of bottles for today: 342
Enter number of bottles for today: 235
Enter number of bottles for today: 539
Enter number of bottles for today: 485
Enter number of bottles for today: 321
Enter number of bottles for today: 128
The total number of bottles collected is 2475
The total paid out is $ 247.5

Do you want to end the program? (Enter yes or no): yes
>>>
```

Step 18: Execute your program so that it works and paste the final code below

PASTE CODE HERE

Lab 5.5 – Programming Challenge 1 – Yum Yum Burger Joint

Write the Flowchart and Python code for the following programming problem and the pseudocode below.

Write a program that will calculate the cost of purchasing a meal. This program will include decisions and loops. Details of the program are as follows:

- Your menu items only include the following food with accompanied price:
    - Yum Yum Burger = .99
    - Grease Yum Fries = .79
    - Soda Yum = 1.09
- Allow the user of the program to purchase any quantity of these items on one order.
- Allow the user of the program to purchase one or more types of these items on one order.
- After the order is placed, calculate total and add a 6% sales tax.
- Print to the screen a receipt showing the total purchase price.

Your sample output might look as follows:

```
Enter 1 for Yum Yum Burger
Enter 2 for Grease Yum Fries
Enter 3 for Soda Yum
Enter now ->1
Enter the number of burgers you want 3
Do you want to end your order? (Enter yes or no): no

Enter 1 for Yum Yum Burger
Enter 2 for Grease Yum Fries
Enter 3 for Soda Yum
Enter now ->3
Enter the number of sodas you want 2
Do you want to end your order? (Enter yes or no): no

Enter 1 for Yum Yum Burger
Enter 2 for Grease Yum Fries
Enter 3 for Soda Yum
Enter now ->1
Enter the number of burgers you want 1
Do you want to end your order? (Enter yes or no): no

Enter 1 for Yum Yum Burger
Enter 2 for Grease Yum Fries
Enter 3 for Soda Yum
Enter now ->2
Enter the number of fries you want 2
Do you want to end your order? (Enter yes or no): yes
```

```
The total price is $ 8.1832
Do you want to end program? (Enter no to process a new
order): no

Enter 1 for Yum Yum Burger
Enter 2 for Grease Yum Fries
Enter 3 for Soda Yum
Enter now ->2
Enter the number of fries you want 2
Do you want to end your order? (Enter yes or no): no

Enter 1 for Yum Yum Burger
Enter 2 for Grease Yum Fries
Enter 3 for Soda Yum
Enter now ->3
Enter the number of sodas you want 2
Do you want to end your order? (Enter yes or no): yes

The total price is $ 3.9856
Do you want to end program? (Enter no to process a new
order): yes
```

The Pseudocode

```
Module main()

    Call declareVariables(endProgram, endOrder, totalBurger, totalFry, totalSoda,
    total, tax, subtotal, option, burgerCount, fryCount, sodaCount)

    //Loop to run program again
    While endProgram == “no”

        Call resetVariables(totalBurger, totalFry, totalSoda, total, tax, subtotal)

        //Loop to take in order
        While endOrder == “no”
            Display “Enter 1 for Yum Yum Burger”
            Display “Enter 2 for Grease Yum Fries”
            Display “Enter 3 for Soda Yum”
            Input option
            If option == 1 Then
                Call getBurger(totalBurger, burgerCount)
            Else If option == 2 Then
                Call getFry(totalFry, fryCount)
            Else If option == 3 Then
                Call getSoda(totalSoda, sodaCount)
            End If
```

```
                Display "Do you want to end your order? (Enter no to add more
                items: )"
                Input endOrder
            End While

        Call calcTotal(burgerTotal, fryTotal, sodaTotal, total, subtotal, tax)
        Call printReceipt(total)

        Display "Do you want to end the program? (Enter no to process a new order)"
        Input endProgram
        End While

End Module

Module declareVariables(String Ref endProgram, String Ref endOrder, Real Ref
totalBurger, Real Ref totalFry, Real Ref totalSoda, Real Ref  total, Real Ref tax, Real Ref
subtotal, Real Ref option, Real Ref burgerCount, Real Ref fryCount, Real Ref
sodaCount)
        Declare String endProgram = "no"
        Declare String endOrder = "no"
        Declare Real totalBurger = 0
        Declare Real totalFry = 0
        Declare Real totalSoda = 0
        Declare Real total = 0
        Declare Real tax = 0
        Declare Real subtotal = 0
        Declare Integer option = 0
        Declare Integer burgerCount = 0
        Declare Integer fryCount = 0
        Declare Integer sodaCount = 0
End Module

Module resetVariables (Real Ref totalBurger, Real Ref totalFry, Real Ref totalSoda, Real
Ref total, Real Ref tax, Real Ref subtotal)

        //reset variables
        totalBurger = 0
        totalFry = 0
        totalSoda = 0
        total = 0
        tax = 0
        subtotal = 0
End Module

Module getBurger(Real Ref totalBurger, Integer burgerCount)
        Display "Enter the number of burgers you want"
        Input burgerCount
        Set totalBurger = totalBurger + burgerCount * .99
End Module

Module getFry(Real Ref totalFry, Integer fryCount)
        Display "Enter the number of fries you want"
        Input fryCount
        Set totalFry = totalFry + fryCount * .79
End Module

Module getSoda(Real Ref totalSoda, Integer sodaCount)
        Display "Enter the number of sodas you want"
        Input sodaCount
        Set totalSoda = totalSoda + sodaCount * 1.09
End Module

Module calcTotal(Real totalBurger, Real totalFry, Real totalSoda, Real Ref total, Real
subtotal, Real tax)
        Set subtotal = totalBurger + totalFry + totalSoda
        Set tax = subtotal * .06
        Set total = subtotal + tax
End Module

Module printReceipt(Real total)
        Display "Your total is $", total
End Module
```

The Flowchart

PASTE FLOWCHART HERE

The Python Code

PASTE CODE HERE