**Module 6 Lab: Repetition Structures**

This lab accompanies Chapter 5 of *Starting Out with Programming Logic & Design*.

**Module 6 Lab Part 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 |

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 226).

**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

Declare Real keepGoing = “y”

// Function calls

While keepGoing == “y”

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 keepGoing

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 in Pseudocode, page 229).

**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

Declare Real keepGoing = “y”

// 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 keepGoing

While keepGoing == “y”

End Module

**Module 6 Lab Part 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 as 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**

Declare Real totalBottles = 0

Declare Real counter = 1

Declare Real todayBottles = 0

Declare Real totalPayout = 0

Declare Real keepGoing = “y”

**// Step 3: Loop to run program again**

While keepGoing == “y”

**// Step 2: Call functions**

Call getBottles(totalBottles, todayBottles, counter)

Call calcPayout(totalPayout, totalBottles)

Call printInfo(totalBottles, totalPayout)

Display "Do you want to run the program again? (Enter y for yes)."

Input keepGoing

End While

End Module

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

1. The missing parameter list
2. The missing condition (Hint: should run seven iterations)
3. The missing input variable
4. The missing accumulator
5. The increment statement for the counter

// getBottles module

Module getBottles(**a.** totalBottles, todayBottles, counter)

While **b.** counter <= 7

Display "Enter number of bottles returned for the day:"

Input **c.** todayBottles

**d.** totalBottles += todayBottles

**e.** counter ++

End While

End Module

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

1. The missing parameter list
2. The missing calculation

// calcPayout 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:

1. The missing parameter list
2. The missing display statement
3. The missing display statement

//printInfo module

Module printInfo(**a.**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

1. **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**
2. **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

End Module

**Module 6 Lab Part 3 – Python Code**

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

**Step 1:** Start Visual Studio Code. Prior to entering code, save your file by clicking on File and then Save. Select your location and save this file as *Lab6.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 6 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 input statement such as:

endProgram = 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: Submit this completed word document and .py source file to D2L.**