Introduction to Repetition Structures

- **Often have to write code that performs the same task multiple times**
  - Disadvantages to duplicating code
    - Makes program large
    - Time consuming
    - May need to be corrected in many places
- **<u>Repetition structure</u>: makes computer repeat included code as necessary**
  - Includes condition-controlled loops and count-controlled loops

Repetition structures

- **Repetition structures**
  - cause a statement or set of statements to execute repeatedly
  - are used to perform the same task over and over
  - are commonly called loops

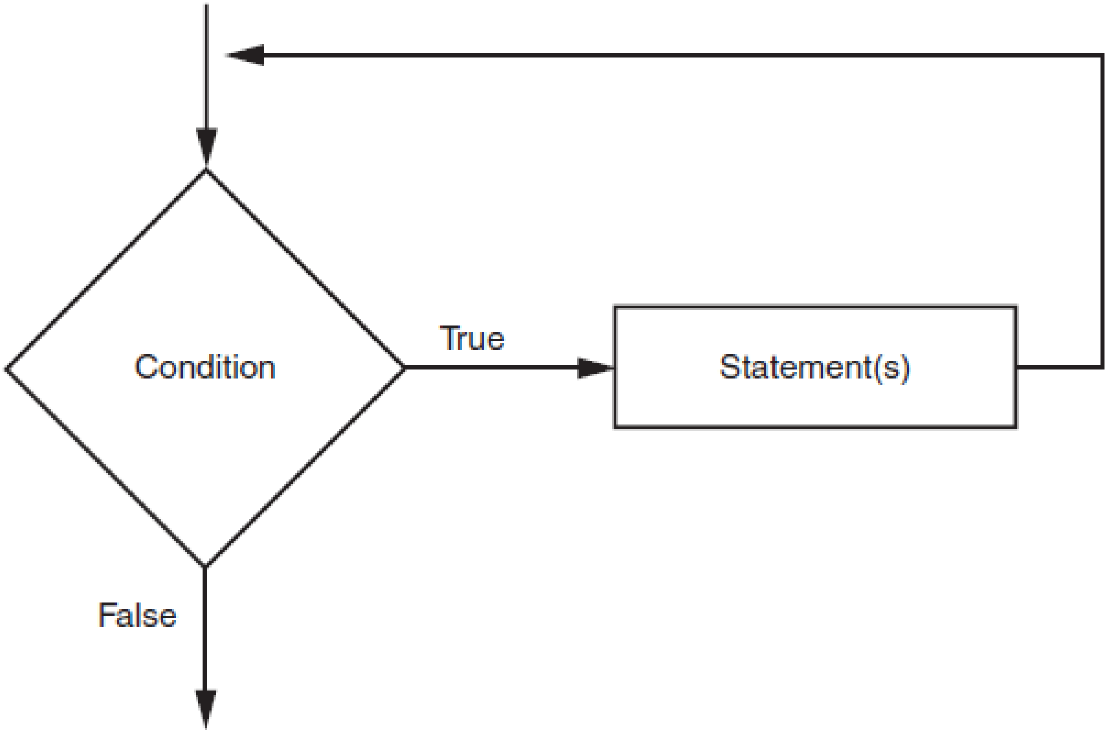
The `while` Loop: a Condition-Controlled Loop

- **<u>`while` loop</u>: while condition is true, do something**
  - Two parts:
    - Condition tested for true or false value
    - Statements repeated as long as condition is true
  - In flow chart, line goes back to previous part
  - General format:

```
while condition:
    statements
```

The `while` Loop: a Condition-Controlled Loop (cont'd.)

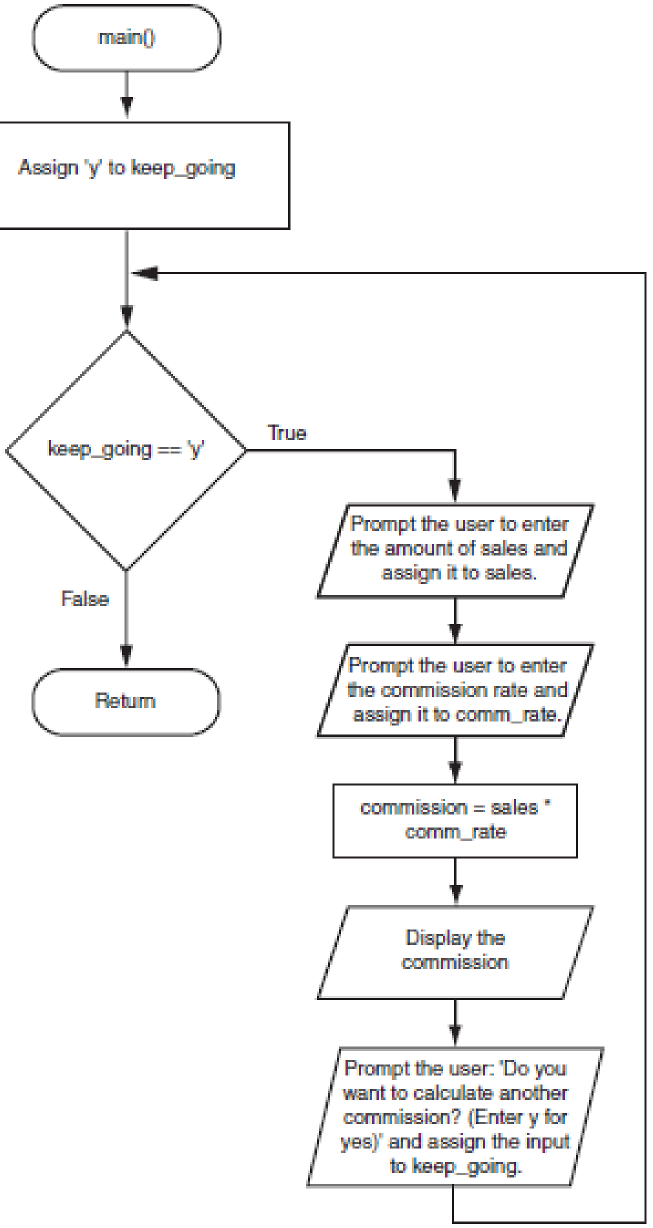
Figure 5-1 The logic of a `while` loop



The `while` Loop: a Condition-Controlled Loop (cont'd.)

- **In order for a loop to stop executing, something has to happen inside the loop to make the condition false**
- **<u>Iteration</u>: one execution of the body of a loop**
- **`while` loop is known as a *pretest* loop**
  - Tests condition before performing an iteration
    - Will never execute if condition is false to start with
    - Requires performing some steps prior to the loop

Figure 5-3 Flowchart for Program 5-1


main()
Assign 'y' to keep_going
keep_going == 'y'
True
False
Return
Prompt the user to enter the amount of sales and assign it to sales.
Prompt the user to enter the commission rate and assign it to comm_rate.
commission = sales * comm_rate
Display the commission
Prompt the user: 'Do you want to calculate another commission? (Enter y for yes)' and assign the input to keep_going.

The `while` Loop: a Condition-Controlled Loop (cont'd.)

- **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
```

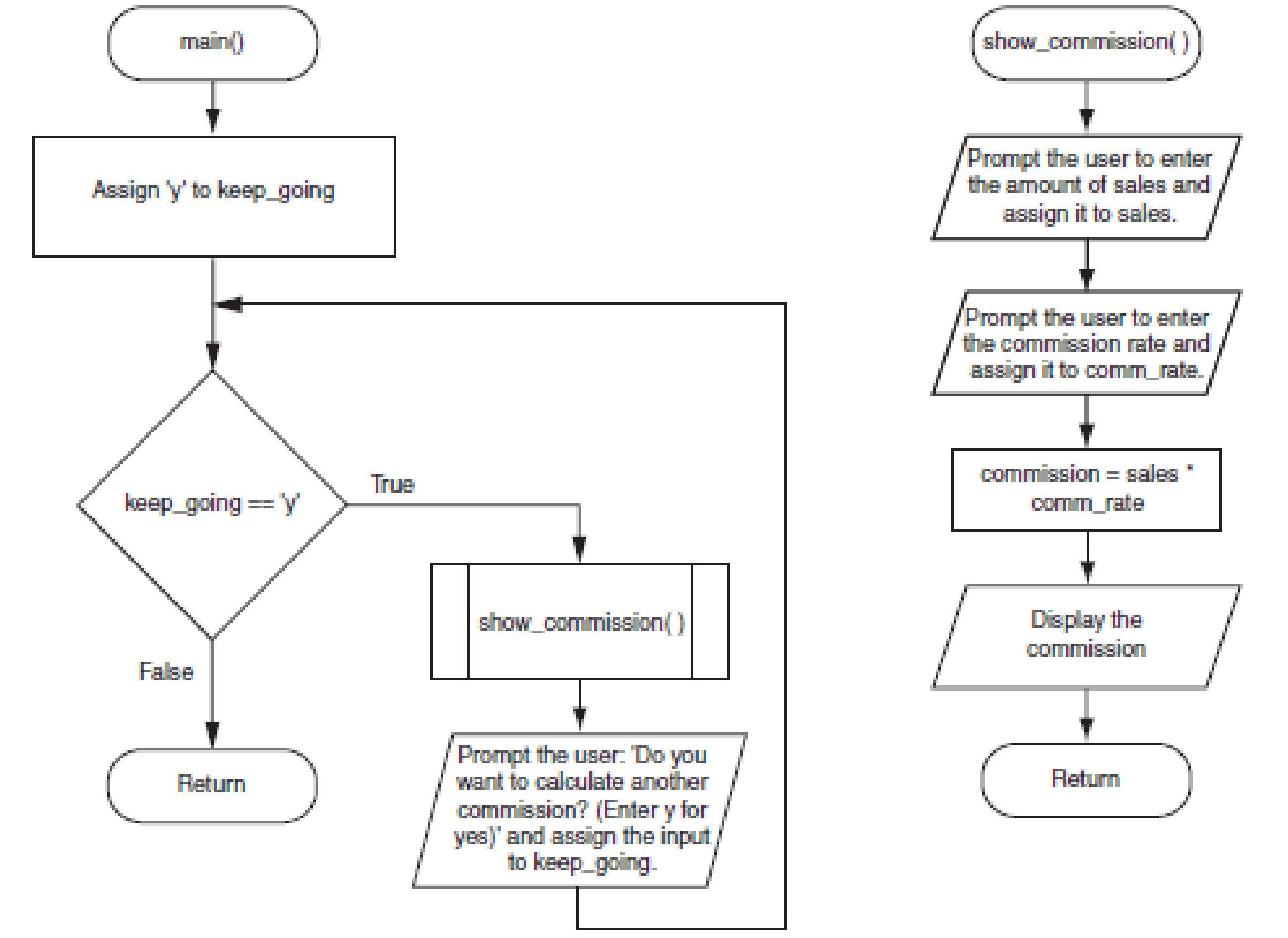
Infinite Loops

- **Loops must contain within themselves a way to terminate**
  - Something inside a `while` loop must eventually make the condition false
- **<u>Infinite loop</u>: loop that does not have a way of stopping**
  - Repeats until program is interrupted
  - Occurs when programmer forgets to include stopping code in the loop

Calling Functions in a Loop

- **Functions can be called from statements in the body of a loop**
  - Often improves the design
  - Example:
    - Write a function to calculate the display the commission for a sales amount
    - Call the function inside a loop

Figure 5-4 Flowcharts for the `main` and `show_commission` functions

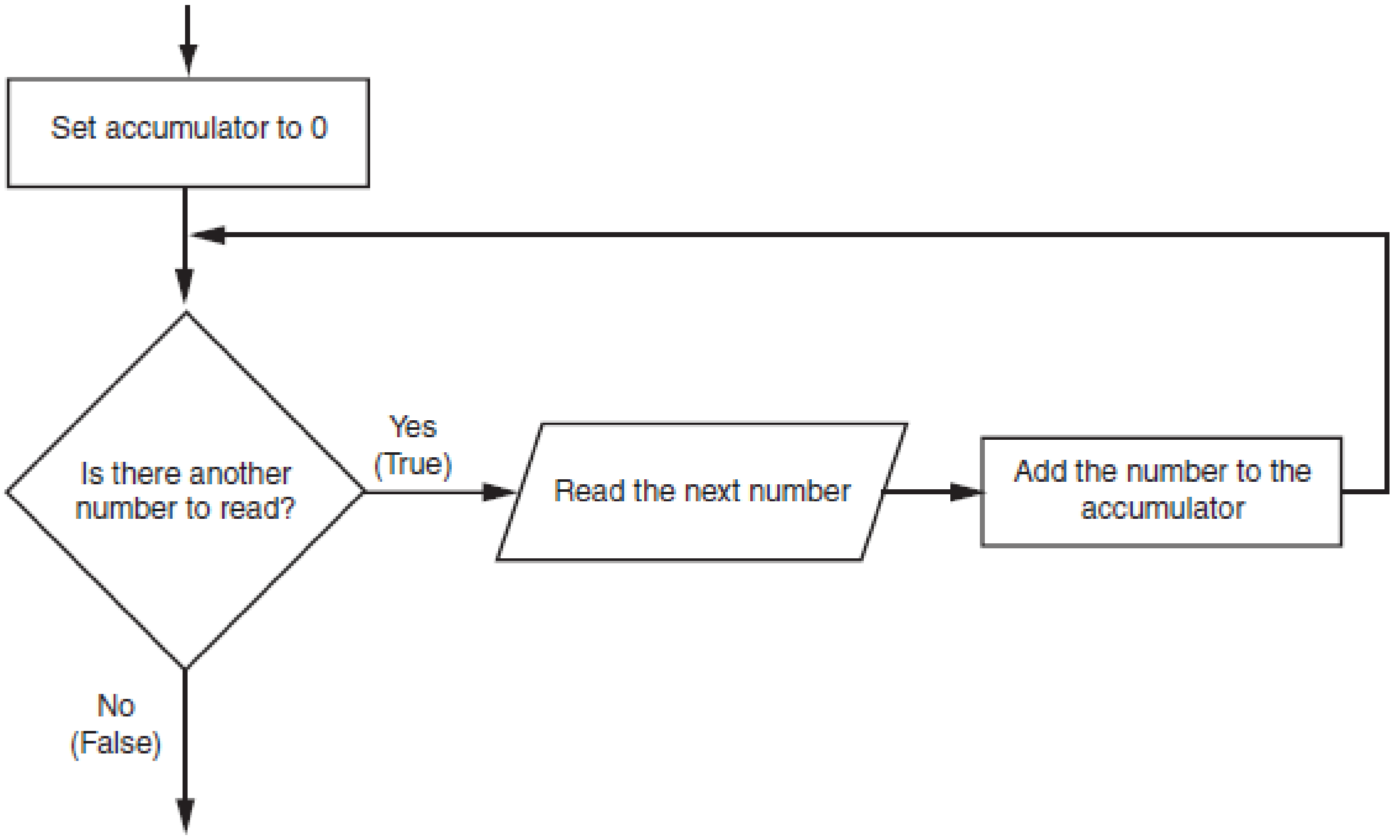

main()
Assign 'y' to keep_going
keep_going == 'y'
True
False
show_commission( )
Prompt the user: 'Do you want to calculate another commission? (Enter y for yes)' and assign the input to keep_going.
Return
show_commission( )
Prompt the user to enter the amount of sales and assign it to sales.
Prompt the user to enter the commission rate and assign it to comm_rate.
commission = sales * comm_rate
Display the commission
Return

Calculating a Running Total

- **Programs often need to calculate a total of a series of numbers**
  - Typically include two elements:
    - A loop that reads each number in series
    - An *accumulator* variable
  - Known as program that keeps a running total: accumulates total and reads in series
  - At end of loop, accumulator will reference the total

Calculating a Running Total (cont'd.)

Figure 5-7 Logic for calculating a running total


Set accumulator to 0
Is there another number to read?
Yes (True)
Read the next number
Add the number to the accumulator
No (False)

The Augmented Assignment Operators

- **In many assignment statements, the variable on the left side of the = operator also appears on the right side of the = operator**
- **<u>Augmented assignment operators</u>: special set of operators designed for this type of job**
  - Shorthand operators

The Augmented Assignment Operators (cont'd.)

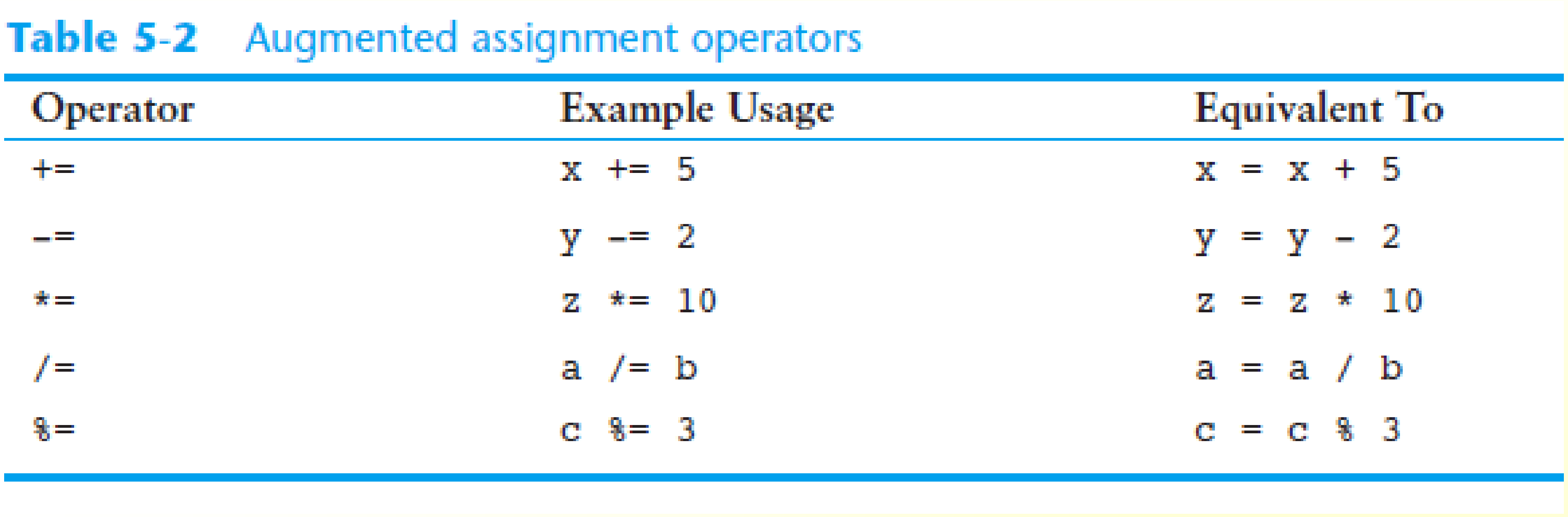


Table 5-2 Augmented assignment operators

Operator	Example Usage	Equivalent To
`+=`	`x += 5`	`x = x + 5`
`-=`	`y -= 2`	`y = y - 2`
`*=`	`z *= 10`	`z = z * 10`
`/=`	`a /= b`	`a = a / b`
`%=`	`c %= 3`	`c = c % 3`

Exercise 1

```
def main():
    endProgram = 'no'
    while endProgram == 'no':
        print('Hello. You are running this simple program')
        endProgram = raw_input('Do you want to end this 
program? Enter yes or no: ')

main()
```

Exercise 2

```
def main():
    endProgram = 'no'
    while endProgram =='no':
        sales = float(input('Enter sales: '))
        print('You sold: ' , sales)

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

main()
```

Exercise 3

```
def main():
    endProgram = 'no'
    totalSales = 0
    while endProgram =='no':
        sales = float(input('Enter sales: '))
        print('You sold: ' , sales)
        totalSales = totalSales + sales
        endProgram = raw_input('Do you want to end this program? Enter yes or no: ')
    print('Total sales: ' , totalSales)
main()
```