CHAPTER 4

Repetition Structures

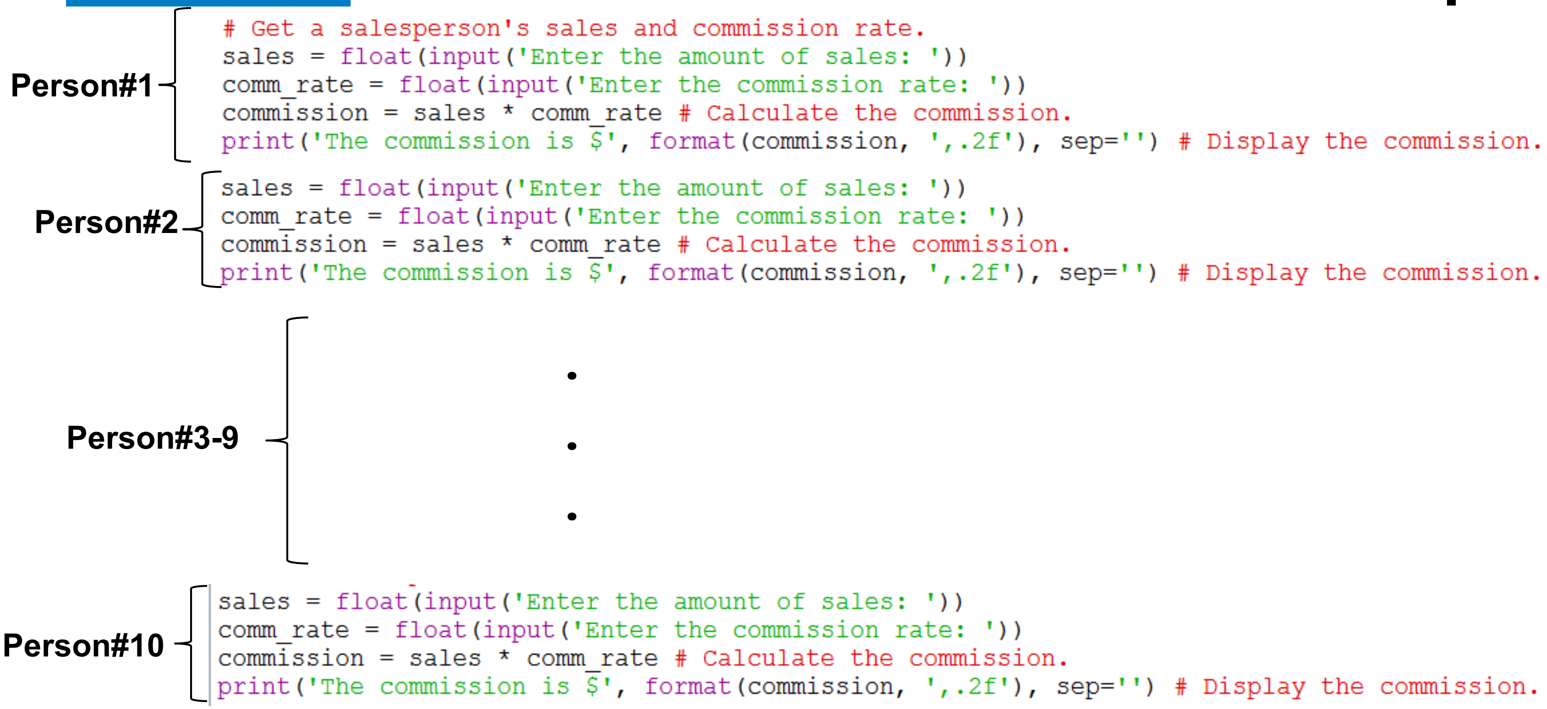
Topics

- Introduction to Repetition Structures
- The `while` Loop: a Condition-Controlled Loop
- The `for` Loop: a Count-Controlled Loop
- Calculating a Running Total
- Sentinels
- Input Validation Loops
- Nested Loops
- Turtle Graphics: Using Loops to Draw Designs

Introduction to Repetition Structures

- **Often have to write code that performs the same task multiple times in a program**

Example: Commission calculator for 10 salesperson



Person#1

```
# Get a salesperson's sales and commission rate.
sales = float(input('Enter the amount of sales: '))
comm_rate = float(input('Enter the commission rate: '))
commission = sales * comm_rate # Calculate the commission.
print('The commission is $', format(commission, ',.2f'), sep='') # Display the commission.
```

Person#2

```
sales = float(input('Enter the amount of sales: '))
comm_rate = float(input('Enter the commission rate: '))
commission = sales * comm_rate # Calculate the commission.
print('The commission is $', format(commission, ',.2f'), sep='') # Display the commission.
```

Person#3-9

.
.
.

Person#10

```
sales = float(input('Enter the amount of sales: '))
comm_rate = float(input('Enter the commission rate: '))
commission = sales * comm_rate # Calculate the commission.
print('The commission is $', format(commission, ',.2f'), sep='') # Display the commission.
```

- Disadvantages to duplicating code
  - Makes program large
  - Time consuming
  - May need to be corrected in many places

Introduction to Repetition Structures

- Instead of writing the same sequence of statements over and over, a better way to repeatedly perform an operation is to write the code for the operation once, then place that code in a structure that makes the computer repeat it as many times as necessary.

- Repetition structure: makes computer repeat certain part of the program/code as many times as necessary

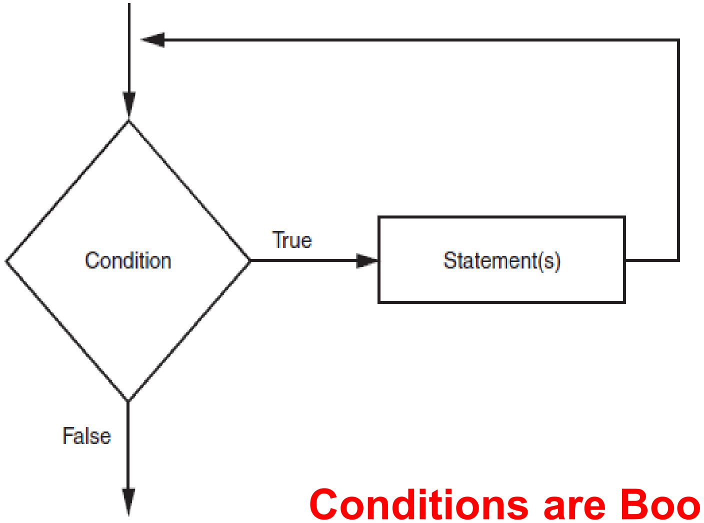
There are two types of Repetition/loops

- Condition-controlled loops – `while` repetitions
- Count-controlled loops – `for` repetitions

The `while` Loop
Condition-Controlled Loop

- **`while` loop: while condition is true, do something**
  - Condition tested for true or false value
  - Statements repeated as long as condition is true
  - In flow chart, line goes back to previous part
  - Something inside a `while` loop must eventually make the condition false

Flow Chart for a While Repetition



General format – Python Syntax:

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

Block of Statements are repeated as long as condition is True

Conditions are Boolean expression that we have learned in Chapter 3.

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

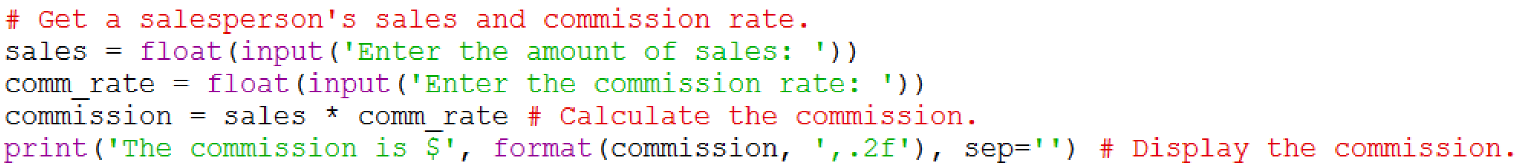
- **In order for a loop to stop executing, something has to happen inside the loop to make the condition false**

- **Iteration: 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

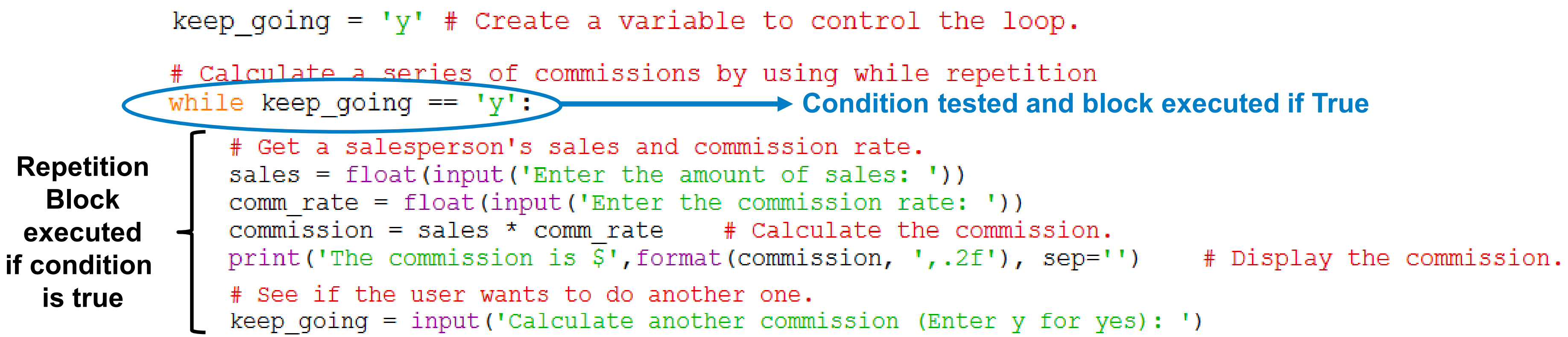
Example: Commission Calculator 1/3

- **Commission Rate Calculation for <u>one Salesperson</u>**



```
# Get a salesperson's sales and commission rate.
sales = float(input('Enter the amount of sales: '))
comm_rate = float(input('Enter the commission rate: '))
commission = sales * comm_rate # Calculate the commission.
print('The commission is $', format(commission, ',.2f'), sep='') # Display the commission.
```

- **We can make it repeated <u>as much as needed</u> by using `while` repetition.** **Page 184**
  - We need a condition to control the repetition.
  - Our condition will be y (yes) n (no) type of condition.



```
keep_going = 'y' # Create a variable to control the loop.

# Calculate a series of commissions by using while repetition
while keep_going == 'y':
    # Get a salesperson's sales and commission rate.
    sales = float(input('Enter the amount of sales: '))
    comm_rate = float(input('Enter the commission rate: '))
    commission = sales * comm_rate    # Calculate the commission.
    print('The commission is $',format(commission, ',.2f'), sep='')    # Display the commission.
    # See if the user wants to do another one.
    keep_going = input('Calculate another commission (Enter y for yes): ')
```

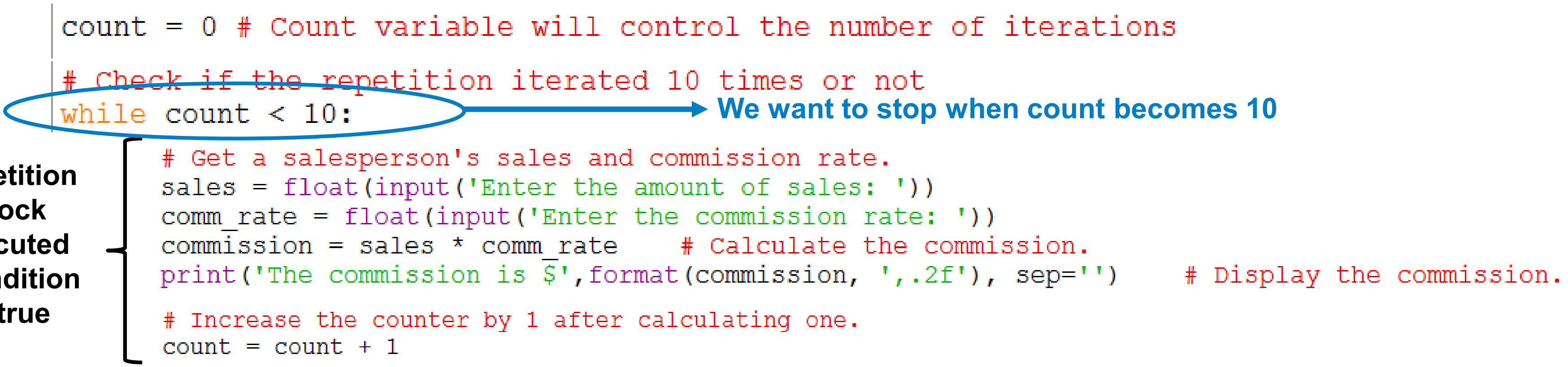
Condition tested and block executed if True

Repetition Block executed if condition is true

If the condition is false, these statements are skipped, and the program exits the loop.

Example: Commission Calculator 2/3

- **Commission Rate Calculation for <u>10 Salesperson</u>**
- **We can make it repeated commission calculation 10 times by using `while` repetition.**
  - Here we need a counter to count the iterations.
  - Rather than y (yes) answer we will use a counter.
  - This counter helps us to stop the repetition after 10 iterations.



```
count = 0 # Count variable will control the number of iterations

# Check if the repetition iterated 10 times or not
while count < 10:
    # Get a salesperson's sales and commission rate.
    sales = float(input('Enter the amount of sales: '))
    comm_rate = float(input('Enter the commission rate: '))
    commission = sales * comm_rate    # Calculate the commission.
    print('The commission is $',format(commission, ',.2f'), sep='')    # Display the commission.

    # Increase the counter by 1 after calculating one.
    count = count + 1
```

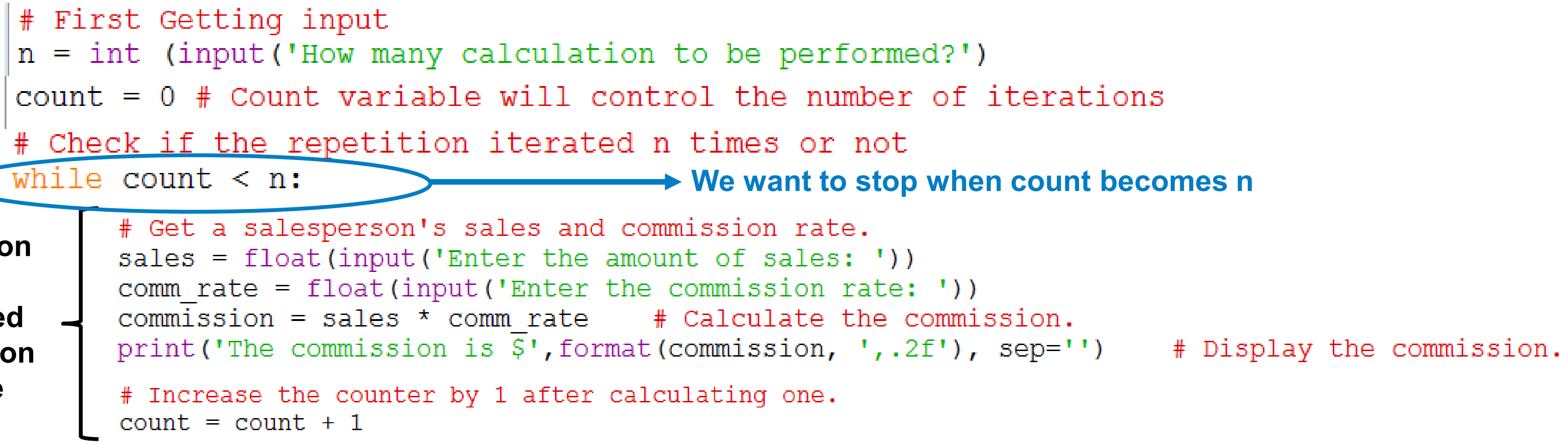
We want to stop when count becomes 10

Repetition Block executed if condition is true

Example: Commission Calculator 3/3

<u>Commission Rate Calculation for a desired-number of Salesperson (a desired-number: </u>user defined number)

- **We can make it repeated commission calculation `n` times by using `while` repetition.**
  - `n` will be inputted by the user
  - Here again we need a counter to count the number of iterations.



```
# First Getting input
n = int (input('How many calculation to be performed?'))
count = 0 # Count variable will control the number of iterations
# Check if the repetition iterated n times or not
while count < n:
    # Get a salesperson's sales and commission rate.
    sales = float(input('Enter the amount of sales: '))
    comm_rate = float(input('Enter the commission rate: '))
    commission = sales * comm_rate    # Calculate the commission.
    print('The commission is $',format(commission, ',.2f'), sep='')    # Display the commission.

    # Increase the counter by 1 after calculating one.
    count = count + 1
```

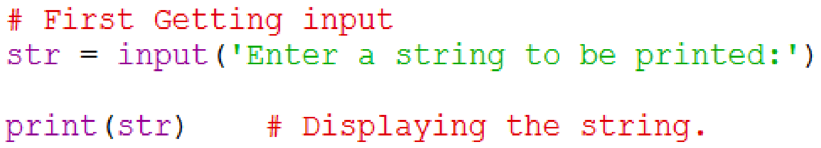
We want to stop when count becomes n

Repetition Block executed if condition is true

Example: Printing Multiple Times

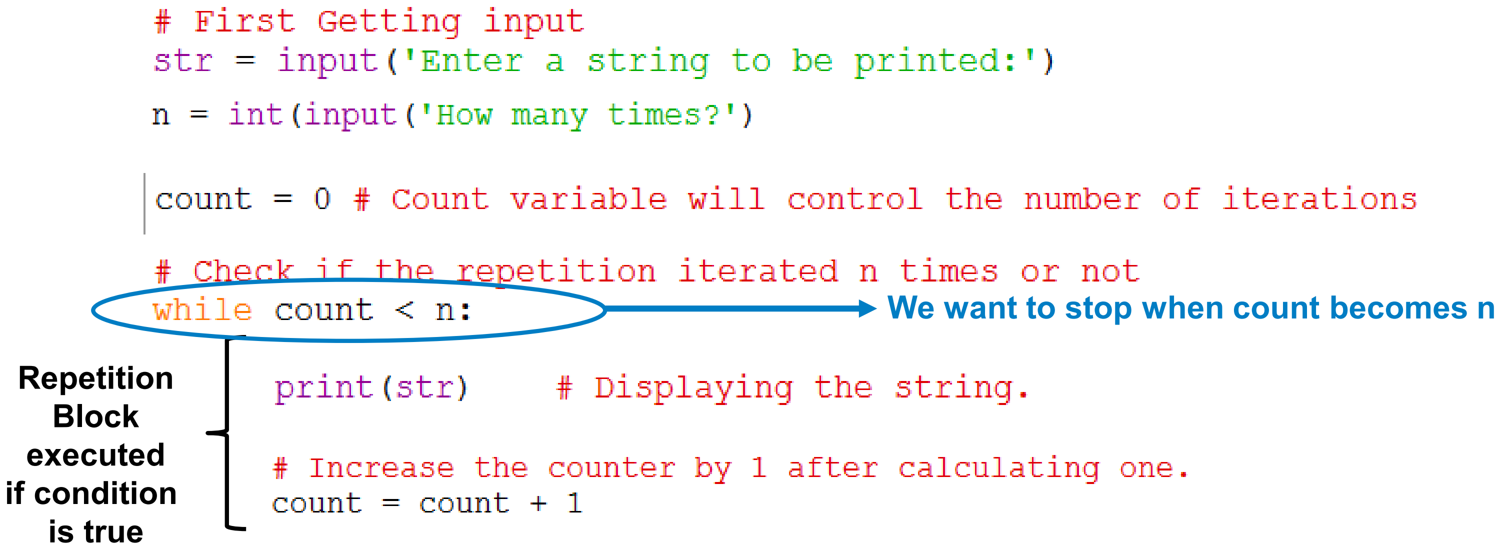
Write a program which prints/display a given string to the screen for a desired number of times.

- **Sequential Structure - printing only one times.**



```
# First Getting input
str = input('Enter a string to be printed:')

print(str)     # Displaying the string.
```

- **Repetition Structure – printing for a desired number of times.**



```
# First Getting input
str = input('Enter a string to be printed:')
n = int(input('How many times?'))

count = 0 # Count variable will control the number of iterations

# Check if the repetition iterated n times or not
while count < n:

    print(str)     # Displaying the string.

    # Increase the counter by 1 after calculating one.
    count = count + 1
```

We want to stop when count becomes n

Repetition Block executed if condition is true

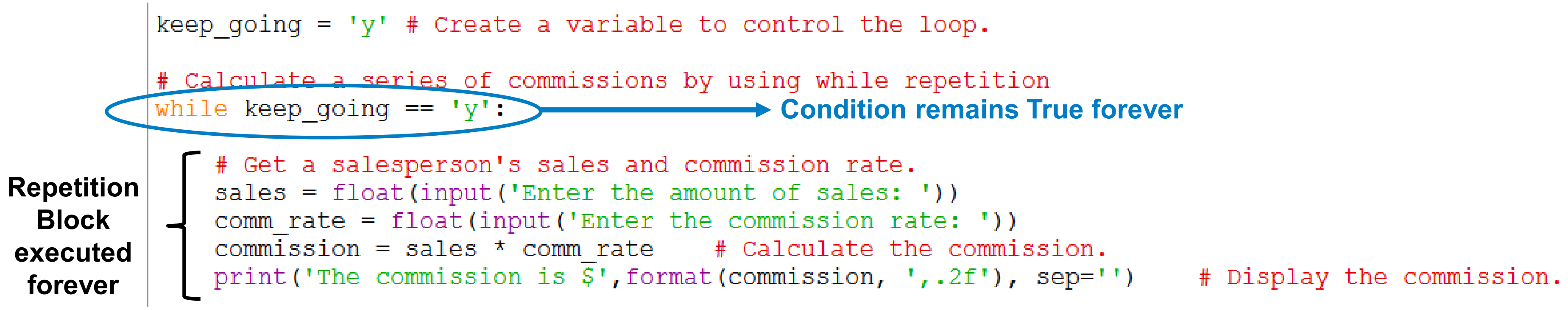
Infinite Loops

- **Loops must contain within themselves a way to terminate**
  - Something inside a `while` loop must eventually make the condition False
- **Infinite loop: 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
  - Sometimes, infinite loop occurs as a part of logical error

Example: Infinite Loop

Page 189

- **Below is infinite loop example for commission calculation program.**



```
keep_going = 'y' # Create a variable to control the loop.

# Calculate a series of commissions by using while repetition
while keep_going == 'y':

    # Get a salesperson's sales and commission rate.
    sales = float(input('Enter the amount of sales: '))
    comm_rate = float(input('Enter the commission rate: '))
    commission = sales * comm_rate     # Calculate the commission.
    print('The commission is $',format(commission, ',.2f'), sep='')     # Display the commission.
```

Condition remains True forever

Repetition Block executed forever

- `keep_going` variable remains as `y` all the time.
- `keep_going == 'y'` condition remains True.
- Repeats until program is interrupted

Checkpoint

Page 190

4.4 What is a loop iteration?

One execution of the body of a loop.

4.5 Does the while loop test its condition before or after it performs an iteration?

Tests condition before performing an iteration. While loop is also known as a pre-test loop.

4.6 How many times will 'Hello World' be printed in the following program?

```
count = 10
while count < 1:
     print('Hello World')
```

It doesn't do iteration because `10 < 1` is False at the beginning.

4.7 What is an infinite loop?

Loop that does not have a way of stopping and repeats until program is interrupted.

The `for` Loop: a Count-Controlled Loop

- **Count-Controlled loop: iterates a specific number of times**
  - A `for` statement more suitable to write count-controlled loop
    - Designed to work with sequence of data items
      - Iterates once for each item in the sequence
    - **General format / Python Syntax:**

```
for variable in [val1, val2, etc]:
        statement
        statement
        etc.
```

- **Target variable**: the variable which is the target of the assignment at the beginning of each iteration

Example: Simple `for` Repetition 1/2 Page 191

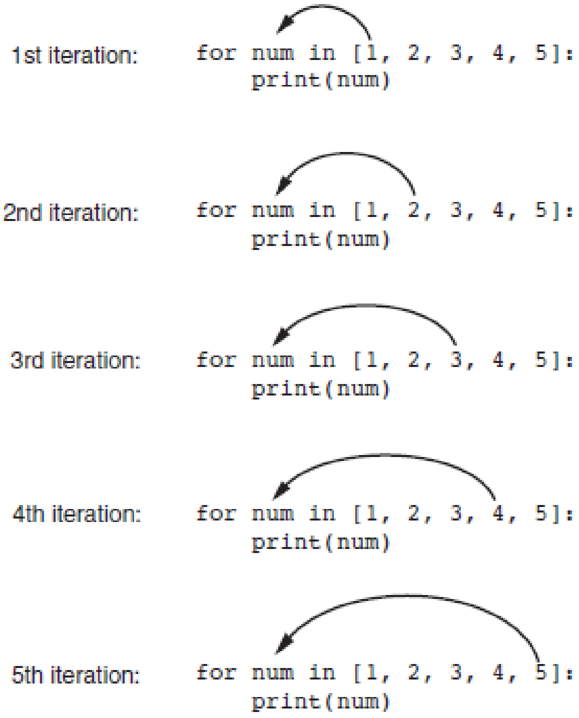
A program that list the integer numbers from 1 through 5.

```
# This program demonstrates a simple for loop
# that uses a list of numbers.

print('I will display the numbers 1 through 5.')
for num in [1, 2, 3, 4, 5]:
    print(num)
```

Program Output

```
I will display the numbers 1 through 5.
1
2
3
4
5
```

Iterations in the Program



1st iteration:

```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

2nd iteration:

```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

3rd iteration:

```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

4th iteration:

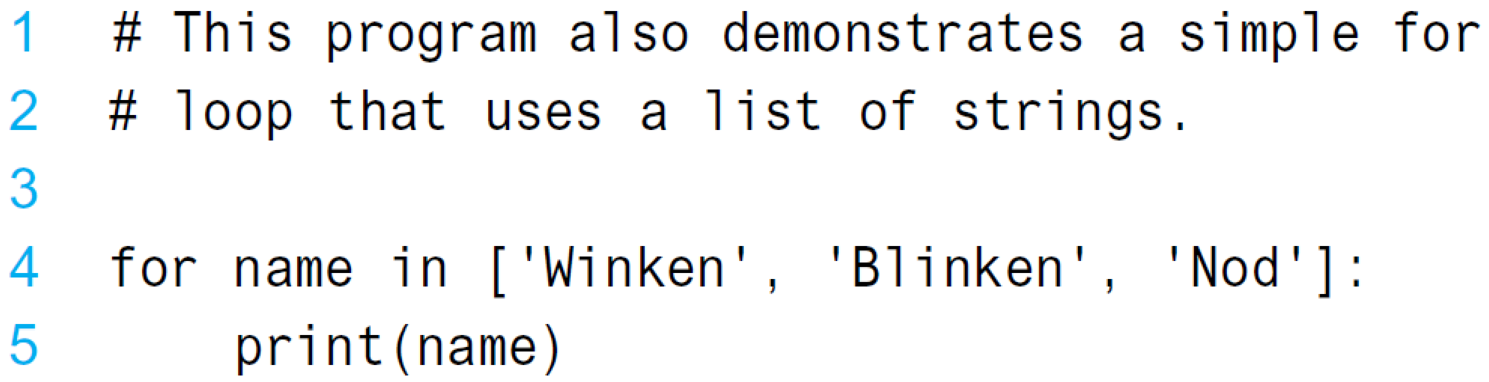
```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

5th iteration:

```
for num in [1, 2, 3, 4, 5]:
    print(num)
```

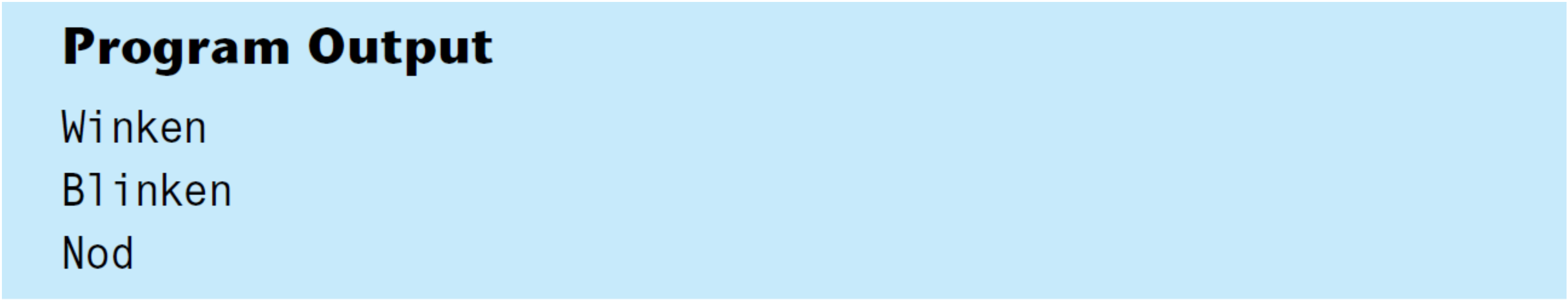
Example: Simple `for` Repetition 2/2 Page 191

A program that list set of names.



```
# This program also demonstrates a simple for
# loop that uses a list of strings.

for name in ['Winken', 'Blinken', 'Nod']:
    print(name)
```

`name` variable iterates three times by running through list of names



Program Output

```
Winken
Blinken
Nod
```

Using the `range` Function with the `for` Loop

- **A built-in `range` function simplifies the process of writing a count-controlled `for` loop**
  - `range` returns an iterable object – sequence of values
    - **Iterable object**: contains a sequence of values that can be iterated over

- **`range` function can be used in different ways:**
  - **One argument:** used as ending limit

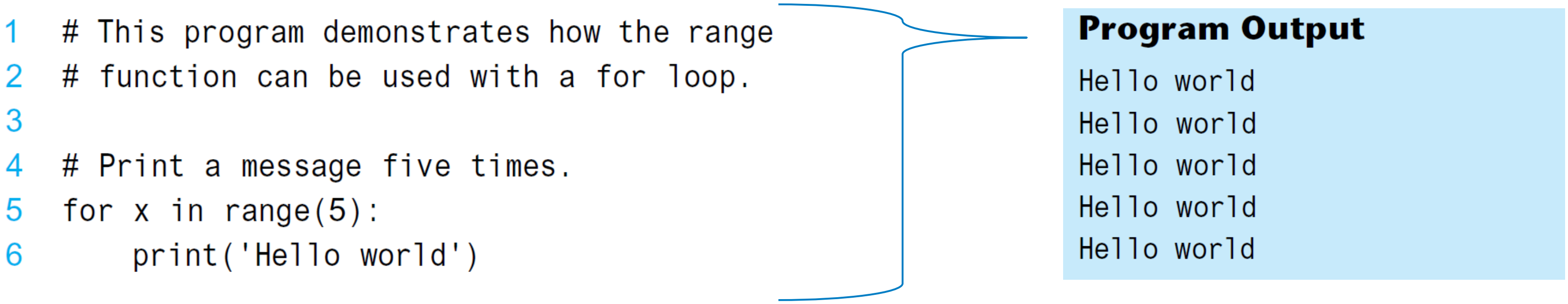
    `range(10)` → [0,1,2,3,4,5,6,7,8,9]
  - **Two arguments:** starting value and ending limit

    `range(5,10)` → [5,6,7,8,9]
  - **Three arguments:** third argument is step value

    `range(24,100,2)` → [24,26,28,……..,94,96,98]

Example: Using `range` function Page 193

- Printing 'Hello World' <u>5 times</u>.

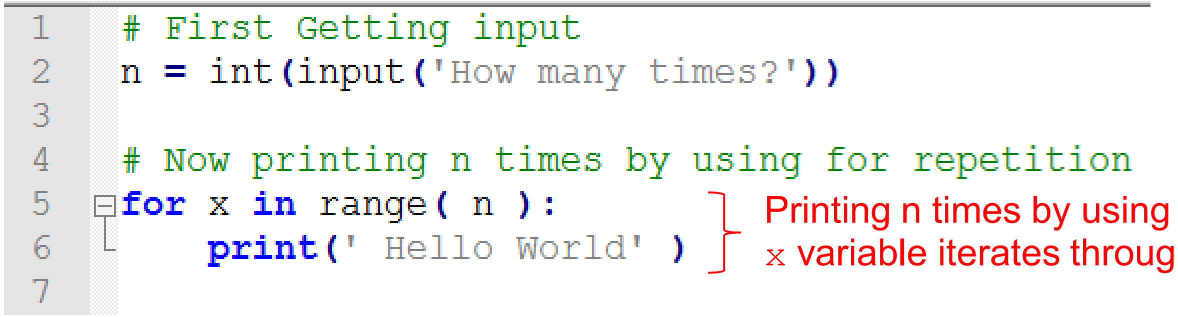


```
# This program demonstrates how the range
# function can be used with a for loop.

# Print a message five times.
for x in range(5):
    print('Hello world')
```

Program Output

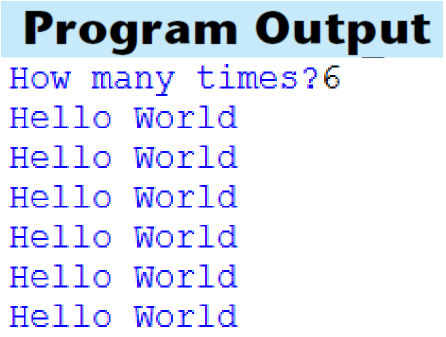
```
Hello world
Hello world
Hello world
Hello world
Hello world
```

- Printing 'Hello World' for <u>a desired-number of times</u>.



```
# First Getting input
n = int(input('How many times?'))

# Now printing n times by using for repetition
for x in range( n ):
    print(' Hello World' )

```

Printing n times by using count-controlled for repetition
`x` variable iterates through `0` through `n-1`



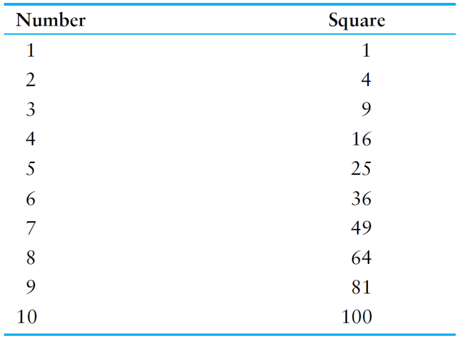
Program Output

```
How many times?6
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

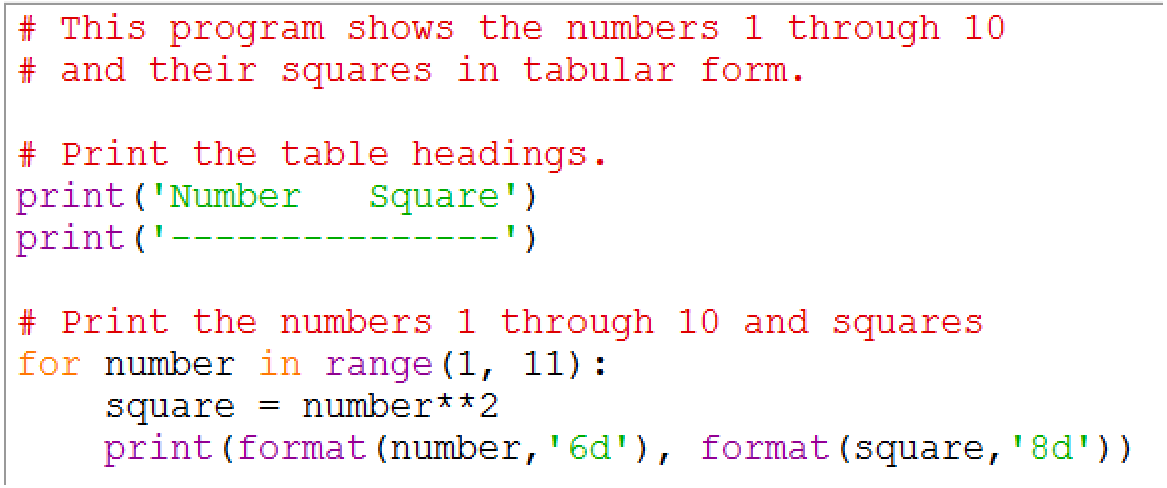
Using the Target Variable Inside the Loop

- **Purpose of target variable is to reference each item in a sequence as the loop iterates**
- **Target variable can be used in calculations or tasks in the body of the loop**

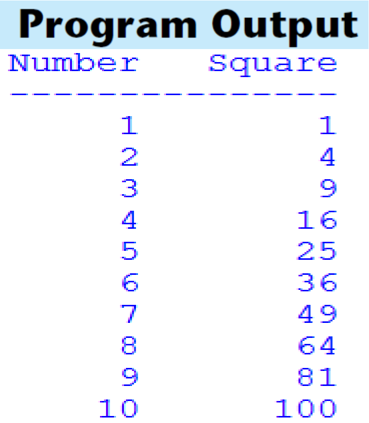
Example: Write a program that displays the numbers 1 through 10 and their respective squares, in a table similar to the following: **Page 194**



Number	Square
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81
10	100



```
# This program shows the numbers 1 through 10
# and their squares in tabular form.

# Print the table headings.
print('Number    Square')
print('----------------')

# Print the numbers 1 through 10 and squares
for number in range(1, 11):
    square = number**2
    print(format(number,'6d'), format(square,'8d'))
```



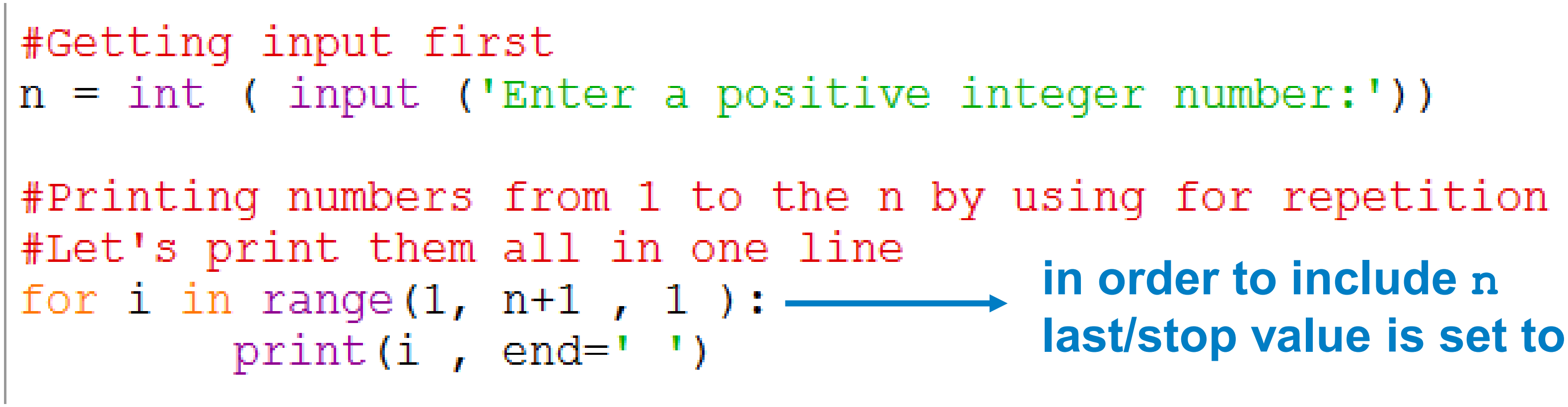
Program Output

```
Number    Square
----------------
     1         1
     2         4
     3         9
     4        16
     5        25
     6        36
     7        49
     8        64
     9        81
    10       100
```

Letting the User Control the Loop Iterations

- **Sometimes the programmer does not know exactly how many times the loop will execute**
- **Can receive range inputs from the user, place them in variables, and call the `range` function in the `for clause` using these variables**
    - Be sure to consider the end cases: `range` does not include the ending limit

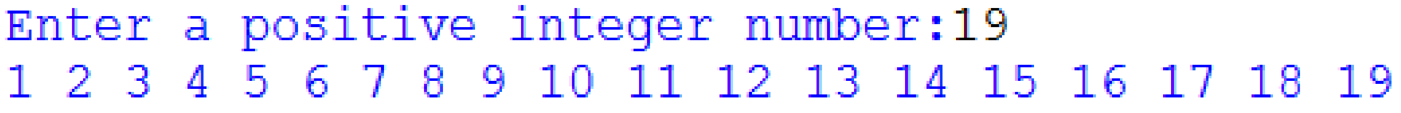
Example: Printing integer from 1 to a desired positive integer number



```
#Getting input first
n = int ( input ('Enter a positive integer number:'))

#Printing numbers from 1 to the n by using for repetition
#Let's print them all in one line
for i in range(1, n+1 , 1 ):
        print(i , end=' ')
```

in order to include `n` last/stop value is set to `n+1`

Example Program Run:



```
Enter a positive integer number:19
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

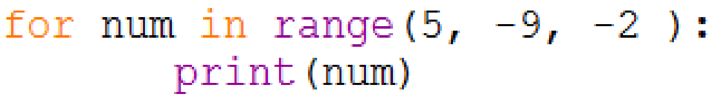
Generating an Iterable Sequence that Ranges from Highest to Lowest

- **The `range` function can be used to generate a sequence with numbers in descending order**
  - Make sure starting number is larger than end limit, and step value is negative

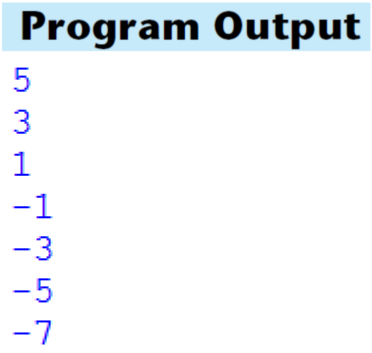
Example:

`range (10, 0, -1)` ⟶ **[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]**

Example: What will be the output of the following program



```
for num in range(5, -9, -2 ):
        print(num)
```



Program Output

```
5
3
1
-1
-3
-5
-7
```

Checkpoint

Page 200

4.8 Rewrite the following code so it calls the range function instead of using the list [0, 1, 2, 3, 4, 5]:

```
for x in [0, 1, 2, 3, 4, 5]:
      print('I love to program!')
```

4.9 What will the following code display?

```
for number in range(6):
      print(number)
```

4.10 What will the following code display?

```
for number in range(2, 6):
      print(number)
```

4.11 What will the following code display?

```
for number in range(0, 501, 100):
      print(number)
```

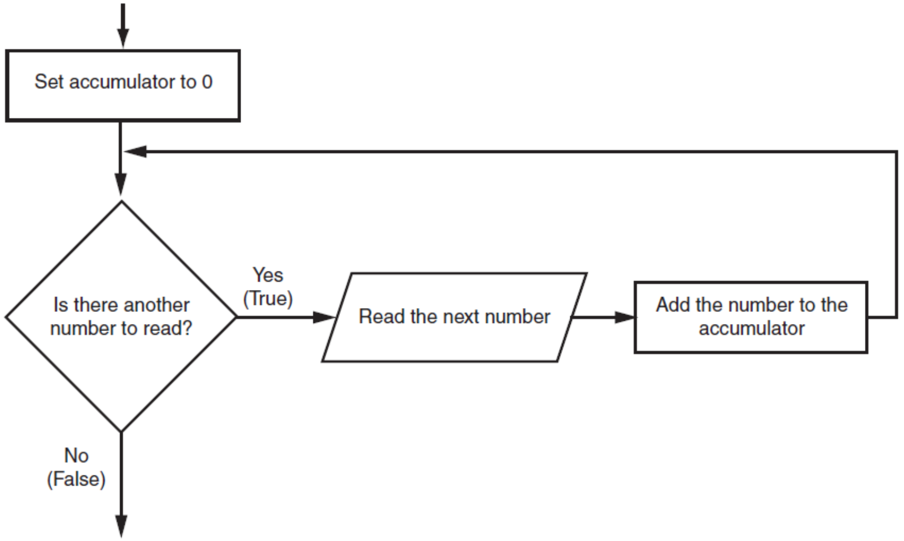
4.12 What will the following code display?

```
for number in range(10, 5, -1):
      print(number)
```

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

Figure 4-6 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)

Example: Processing Sequence of numbers 1/3

Write a Python Program that finds the total of 10 integer numbers entered from keyboard.

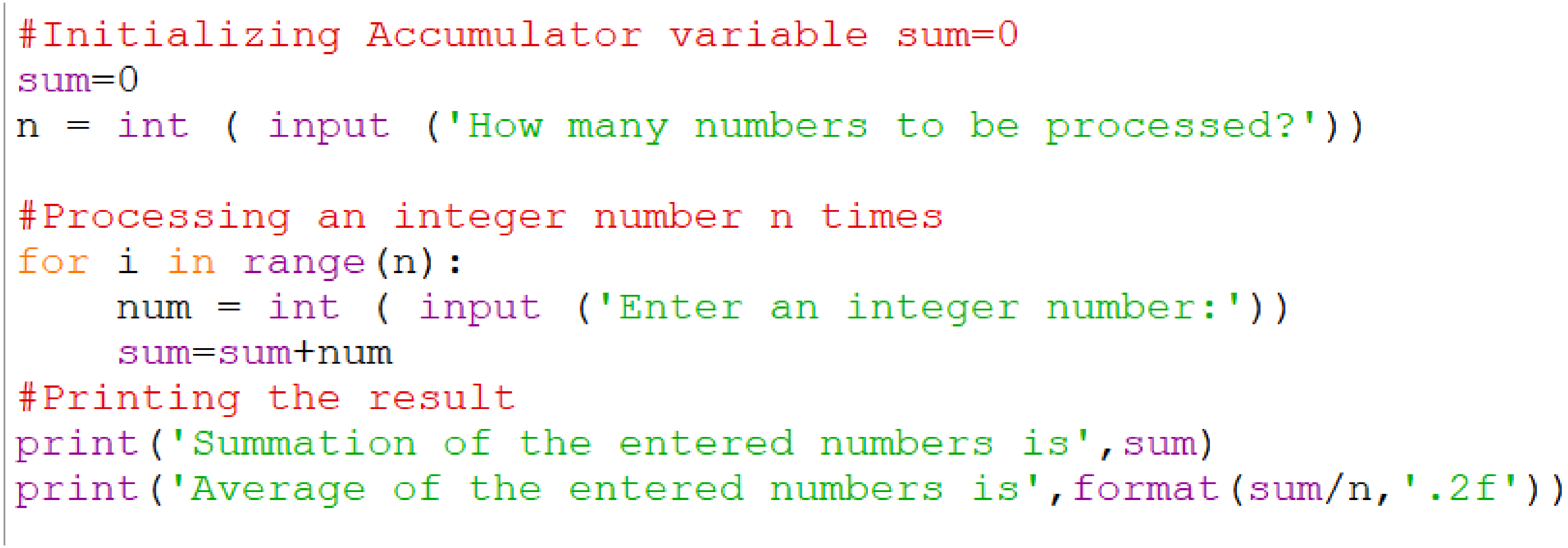
```
#Initializing Accumulator variable sum=0
sum=0

#Processing integer numbers 10 times
for i in range(10):
    num = int ( input ('Enter an integer number:'))
    sum=sum+num
#Printing the result
print('Summation of the entered numbers is',sum)
```

Example Program Run:

```
Enter an integer number:12
Enter an integer number:-4
Enter an integer number:17
Enter an integer number:98
Enter an integer number:35
Enter an integer number:89
Enter an integer number:-100
Enter an integer number:12
Enter an integer number:41
Enter an integer number:-50
Summation of the entered numbers is 150
```

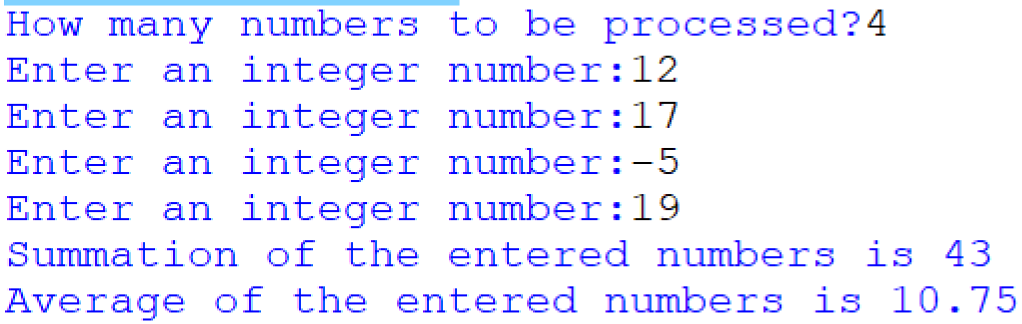
Example: Processing Sequence of numbers 2/3

Write a Python Program that finds the total and average of a desired-number of integer numbers entered from keyboard.



```
#Initializing Accumulator variable sum=0
sum=0
n = int ( input ('How many numbers to be processed?'))

#Processing an integer number n times
for i in range(n):
    num = int ( input ('Enter an integer number:'))
    sum=sum+num
#Printing the result
print('Summation of the entered numbers is',sum)
print('Average of the entered numbers is',format(sum/n,'.2f'))
```

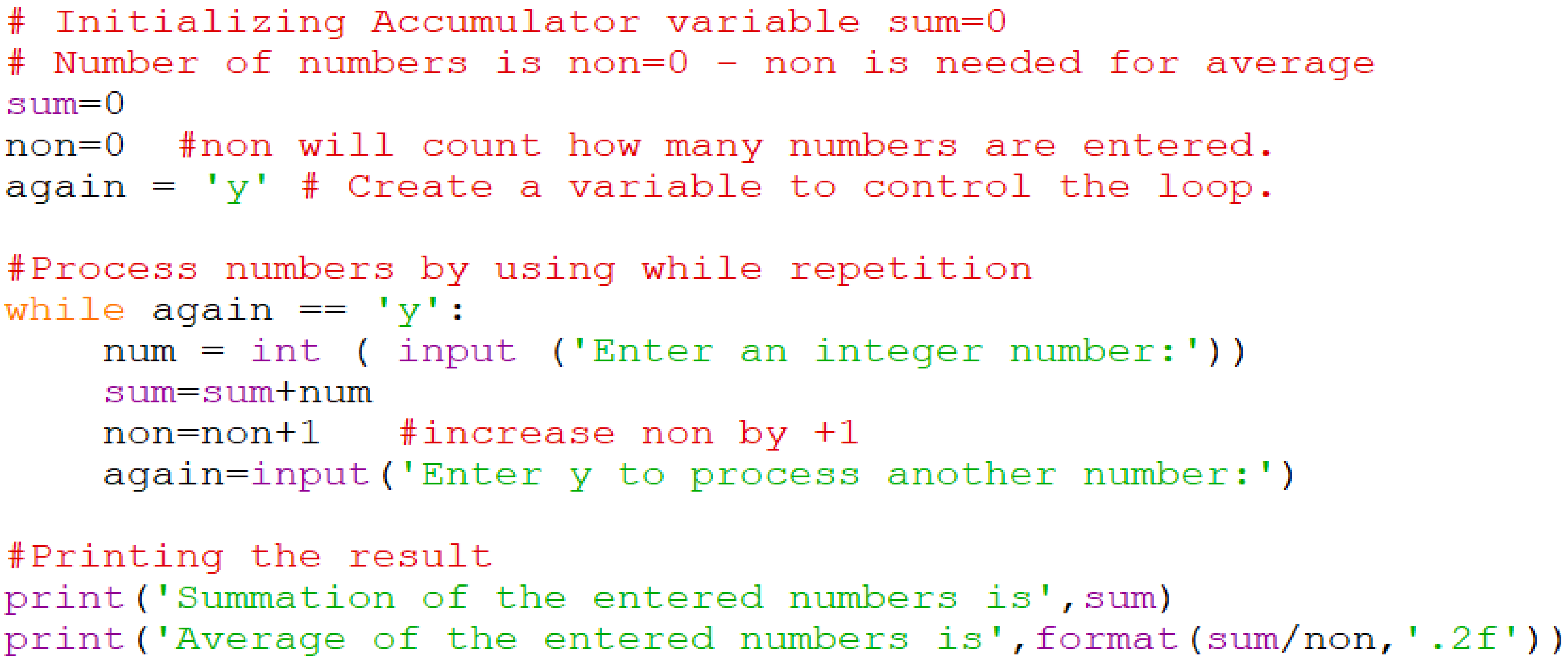
Example Program Run:



```
How many numbers to be processed?4
Enter an integer number:12
Enter an integer number:17
Enter an integer number:-5
Enter an integer number:19
Summation of the entered numbers is 43
Average of the entered numbers is 10.75
```

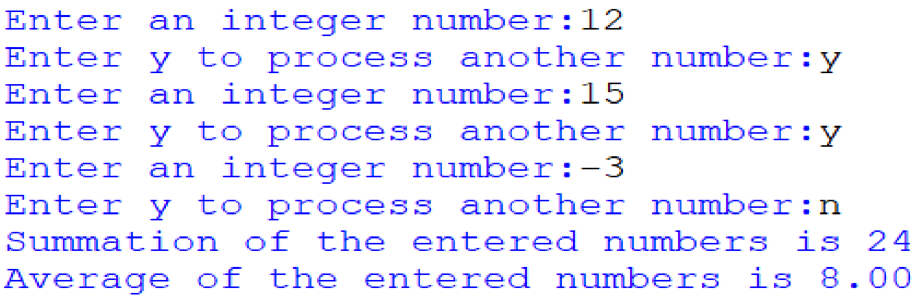
Example: Processing Sequence of numbers 3/3

Write a Python Program that finds the total and average of an unknown number of integer numbers entered from keyboard.



```
# Initializing Accumulator variable sum=0
# Number of numbers is non=0 - non is needed for average
sum=0
non=0  #non will count how many numbers are entered.
again = 'y' # Create a variable to control the loop.

#Process numbers by using while repetition
while again == 'y':
    num = int ( input ('Enter an integer number:'))
    sum=sum+num
    non=non+1   #increase non by +1
    again=input('Enter y to process another number:')

#Printing the result
print('Summation of the entered numbers is',sum)
print('Average of the entered numbers is',format(sum/non,'.2f'))
```

Example Program Run:



```
Enter an integer number:12
Enter y to process another number:y
Enter an integer number:15
Enter y to process another number:y
Enter an integer number:-3
Enter y to process another number:n
Summation of the entered numbers is 24
Average of the entered numbers is 8.00
```

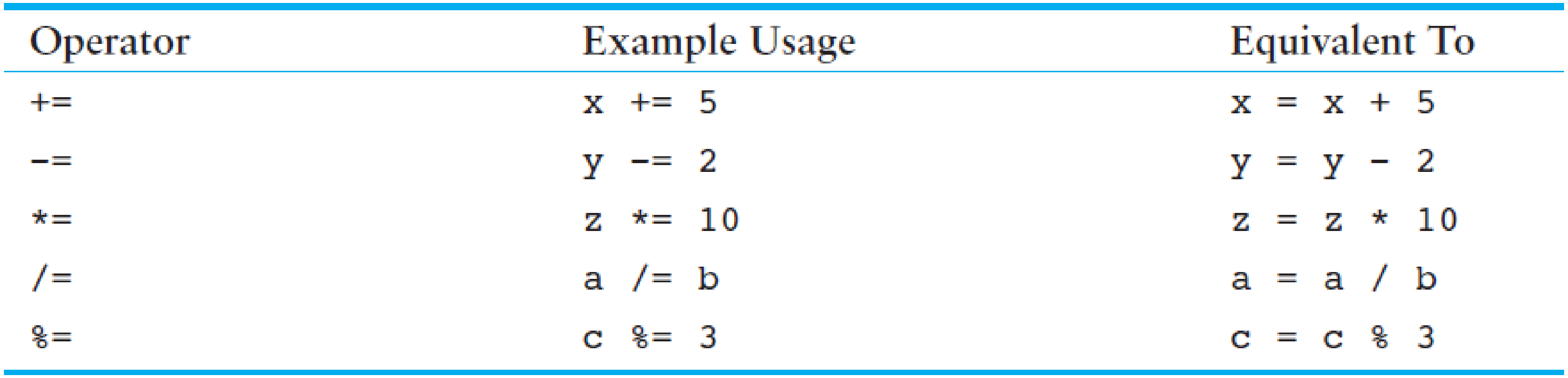
The Augmented Assignment Operators

- **In many assignment statements, the variable on the left side of the = operator also appears on the right side of the = operator**

```
 x=x+1                 total=total+number
mult=mult*2            balance = balance – withdrawal
```

- These can be written in shorthand notation in Python.
- **Augmented assignment operators: special set of operators designed for this type of job - Shorthand operators**

Table 4-2 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`

Checkpoint

Page 204

4.14 Should an accumulator be initialized to any specific value? Why or why not?

4.15 What will the following code display?

```
total = 0
for count in range(1, 6):
      total = total + count
print(total)
```

4.16 What will the following code display?

```
number1 = 10
number2 = 5
number1 = number1 + number 2
print(number1)
print(number2)
```

4.17 Rewrite the following statements using augmented assignment operators:

a) `quantity = quantity + 1`

b) `days_left = days_left - 5`

c) `price = price * 10`

d) `price = price / 2`

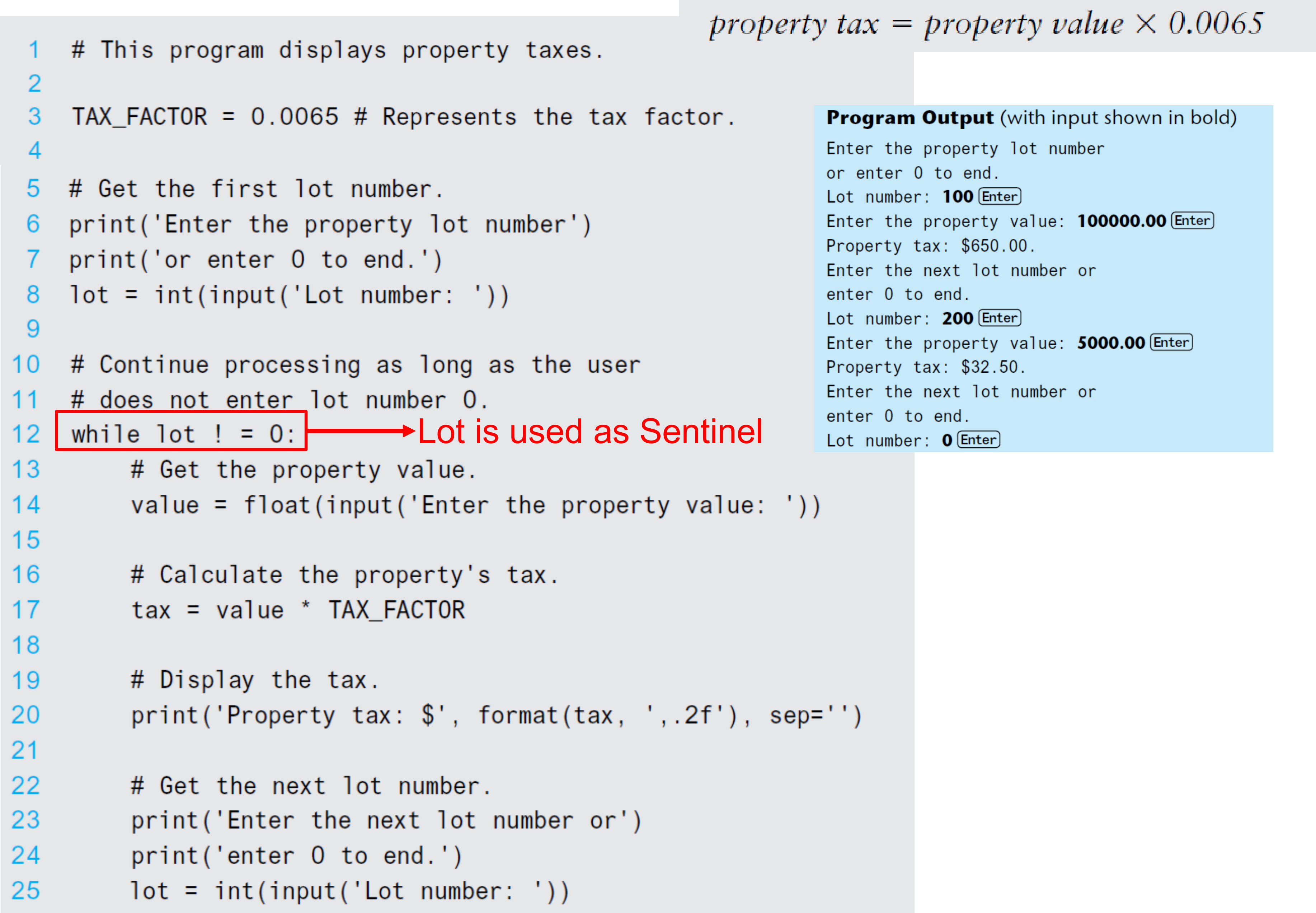
Sentinels

- **Sentinel: special value that marks the end of a sequence of items**
  - Sentinels are used to terminate a condition-controlled repetition.
  - Must be distinctive enough so as not to be mistaken for a regular value in the sequence

Examples:

```
while ( keep_going == 'y' )
while ( age <= 0 )
while ( grade >= 0 or grade <= 100 )
while ( keep_going ! = 999)
while ( product > 100 )
while ( word == '' )
while ( number <=0 )
```

Example: Sentinel Usage – Tax Calc. Page 205



$$\text{property tax} = \text{property value} \times 0.0065$$

```
# This program displays property taxes.

TAX_FACTOR = 0.0065 # Represents the tax factor.

# Get the first lot number.
print('Enter the property lot number')
print('or enter 0 to end.')
lot = int(input('Lot number: '))

# Continue processing as long as the user
# does not enter lot number 0.
while lot ! = 0:
    # Get the property value.
    value = float(input('Enter the property value: '))

    # Calculate the property's tax.
    tax = value * TAX_FACTOR

    # Display the tax.
    print('Property tax: $', format(tax, ',.2f'), sep='')

    # Get the next lot number.
    print('Enter the next lot number or')
    print('enter 0 to end.')
    lot = int(input('Lot number: '))
```

Lot is used as Sentinel

Program Output (with input shown in bold)

```
Enter the property lot number
or enter 0 to end.
Lot number: 100 Enter
Enter the property value: 100000.00 Enter
Property tax: $650.00.
Enter the next lot number or
enter 0 to end.
Lot number: 200 Enter
Enter the property value: 5000.00 Enter
Property tax: $32.50.
Enter the next lot number or
enter 0 to end.
Lot number: 0 Enter
```

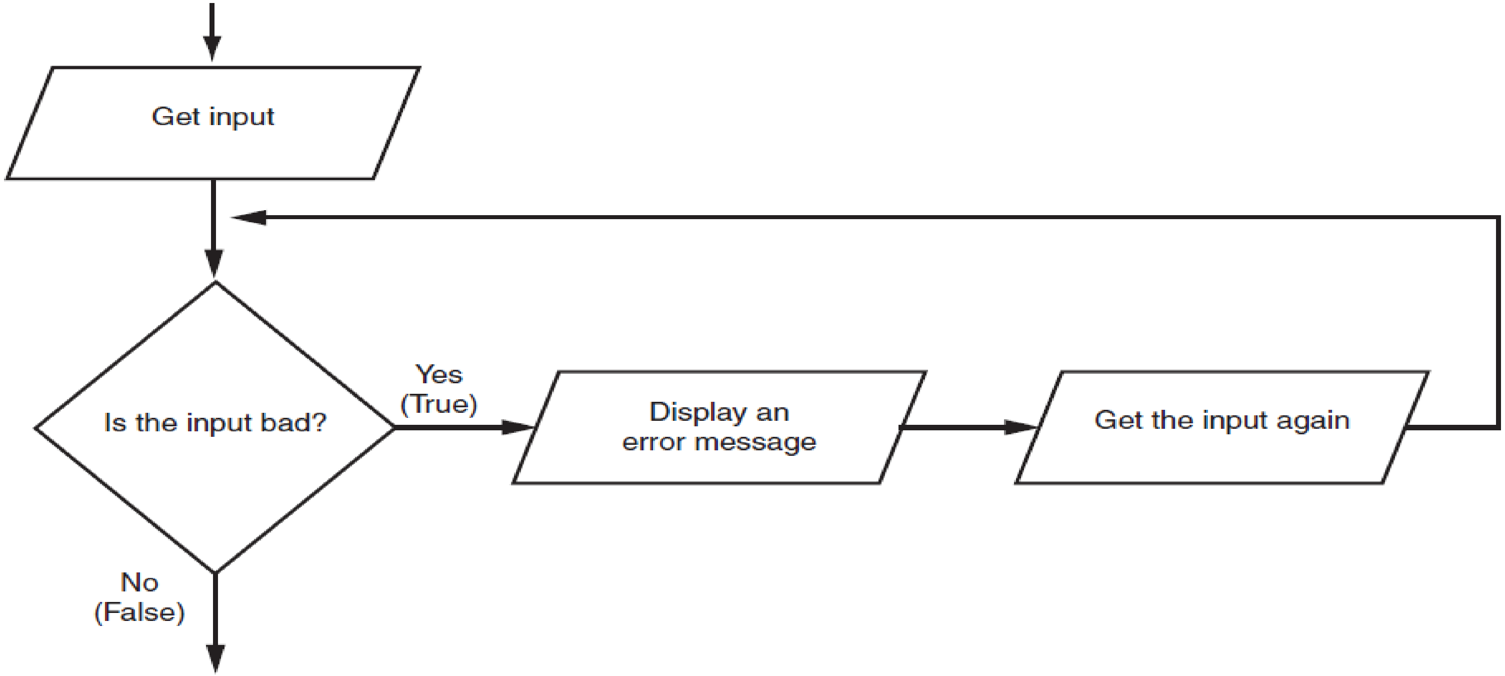
Input Validation Loops

- **Computer cannot tell the difference between good data and bad data**
  - If user provides bad input, program will produce bad output
  - GIGO: garbage in, garbage out
  - It is important to design program such that bad input is never accepted
- **Input validation: inspecting input before it is processed by the program**
  - If input is invalid, prompt user to enter correct data
  - Commonly accomplished using a `while` loop which repeats as long as the input is bad
    - If input is bad, display error message and receive another set of data
    - If input is good, continue to process the input

Input Validation Loops (cont'd.)

- **Input validation: inspecting input before it is processed by the program**

Figure 4-7 Logic containing an input validation loop

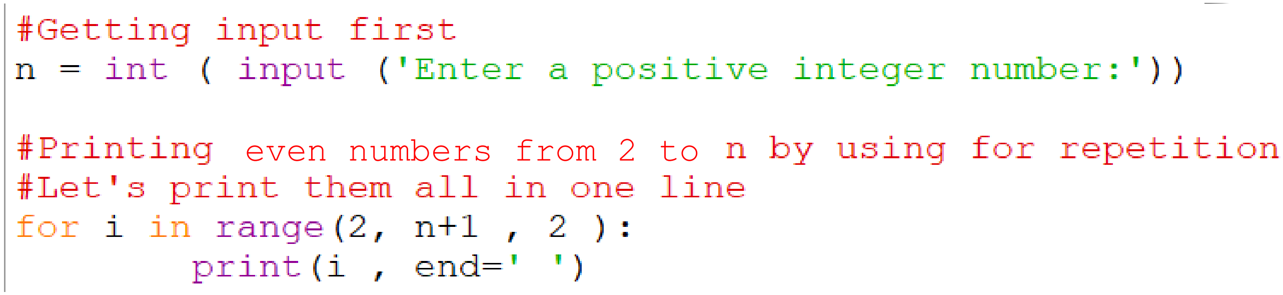


Example: Think about that in a program you ask for a positive integer from user but the user entered a negative number. In this case you can terminate the program rather than GIGO, and by doing Input Validation, you can ask user to reenter a valid data/information.

Example: Input Validation

Let’s consider a program which prints out the positive EVEN integer numbers from 2 to a desired-number.

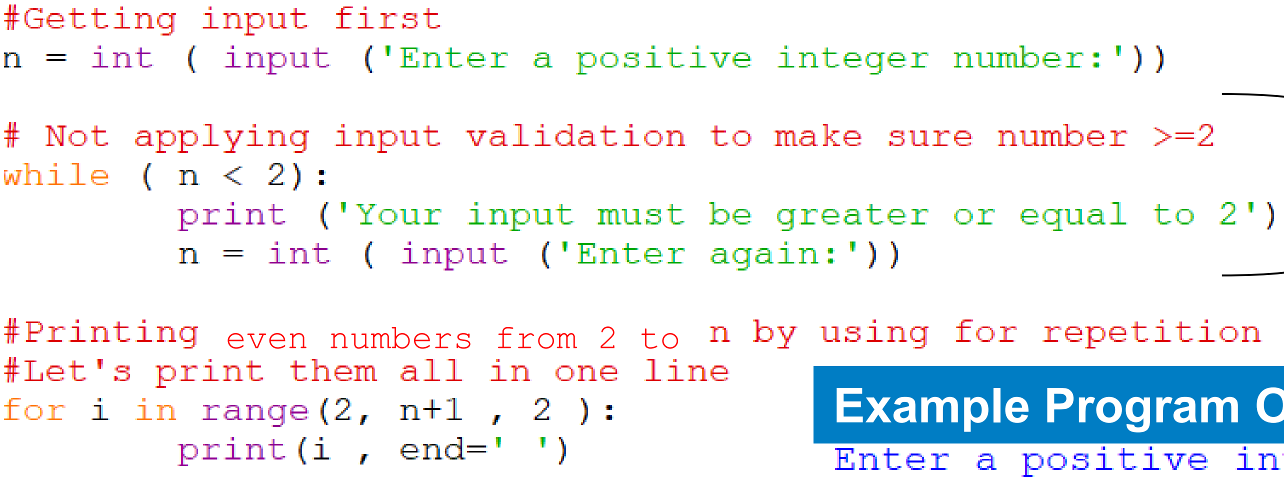
Program without Input Validation



```
#Getting input first
n = int ( input ('Enter a positive integer number:'))

#Printing  even numbers from 2 to n by using for repetition
#Let's print them all in one line
for i in range(2, n+1 , 2 ):
        print(i , end=' ')
```

No input-validation
If the entered number < 2 program doesn’t work.

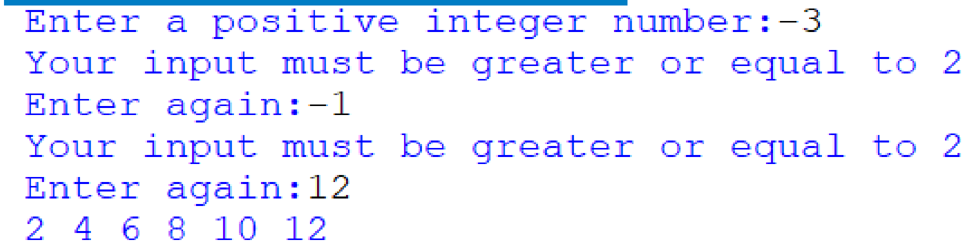
Program with Input Validation



```
#Getting input first
n = int ( input ('Enter a positive integer number:'))

# Not applying input validation to make sure number >=2
while ( n < 2):
        print ('Your input must be greater or equal to 2')
        n = int ( input ('Enter again:'))

#Printing  even numbers from 2 to  n by using for repetition
#Let's print them all in one line
for i in range(2, n+1 , 2 ):
        print(i , end=' ')
```

Applying input-validation
If the entered number < 2, the program ask for input again.

Example Program Output

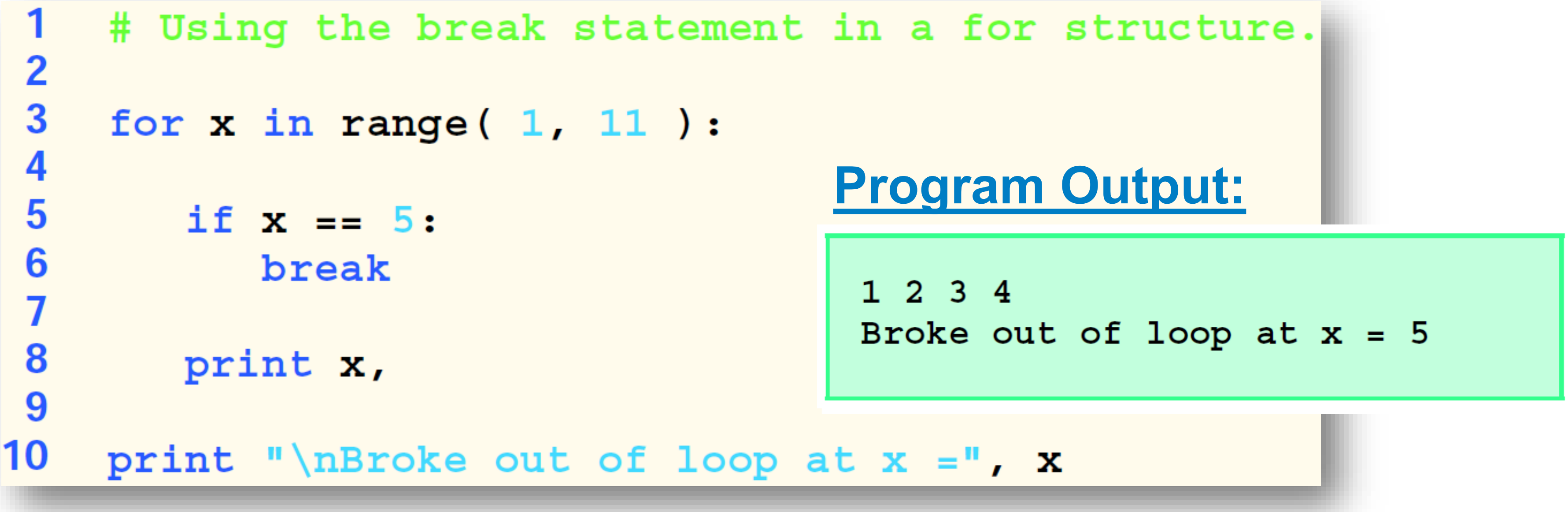


```
Enter a positive integer number:-3
Your input must be greater or equal to 2
Enter again:-1
Your input must be greater or equal to 2
Enter again:12
2 4 6 8 10 12
```

`break` and `continue` Statements 1/3

- Python offers the ***break*** and ***continue*** statements, which alter the flow of control.
- The **break** statement, when executed in a **while** or **for** repetition, causes immediate exit from the repetition.

Example: **the following example illustrate how `break` works:**



```
# Using the break statement in a for structure.

for x in range( 1, 11 ):

    if x == 5:
        break

    print x,

print "\nBroke out of loop at x =", x
```

Program Output:

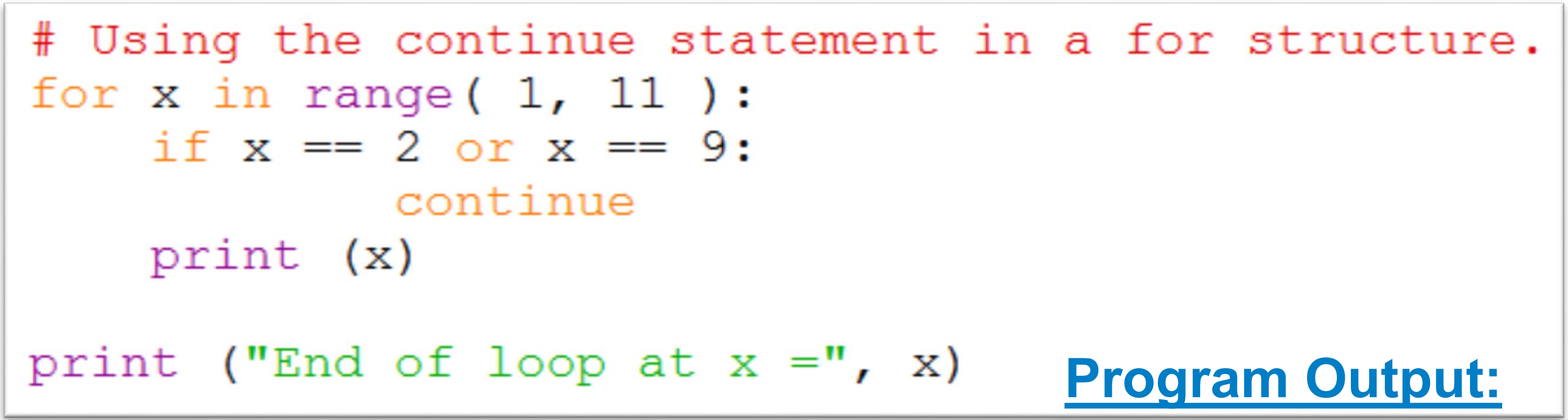
```
1 2 3 4
Broke out of loop at x = 5
```

`At x = 5, iteration/repetition is terminated!`

break and continue Statements 2/3

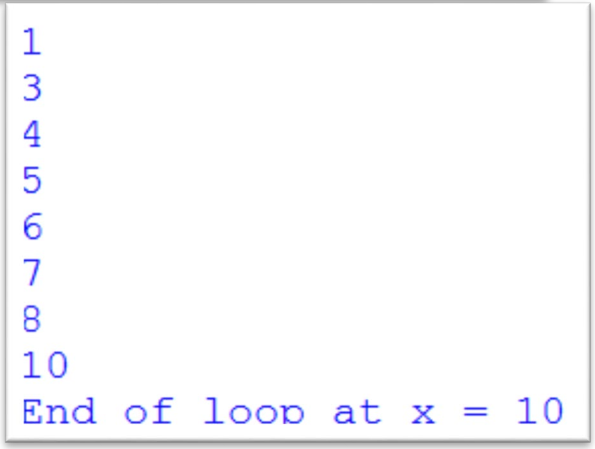
- The ***continue*** statement, when executed in a **while** or a **for** structure, skips the remaining statements in the body of that structure and proceeds with the next iteration of the loop.

Example: **the following example illustrate how `continue` works:**



```
# Using the continue statement in a for structure.
for x in range( 1, 11 ):
    if x == 2 or x == 9:
            continue
    print (x)

print ("End of loop at x =", x)
```

Program Output:



```
1
3
4
5
6
7
8
10
End of loop at x = 10
```

`Values 2 and 9 are skipped!`

break and continue Statements 3/3

Using the break statement for the class-average program while repetition.

```
# Using the break statement class-average program.
# initialization phase
total = 0             # sum of grades
gradeCounter = 0      # number of grades entered

while 1:
   grade = raw_input( "Enter grade, -1 to end: " )
   grade = int( grade )

   # exit loop if user inputs -1
   if grade == -1:
      break

   total += grade
   gradeCounter += 1

# termination phase
if gradeCounter != 0:
   average = float( total ) / gradeCounter
   print "Class average is", average
else:
   print "No grades were entered"
```

Program Output:

```
Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.5
```

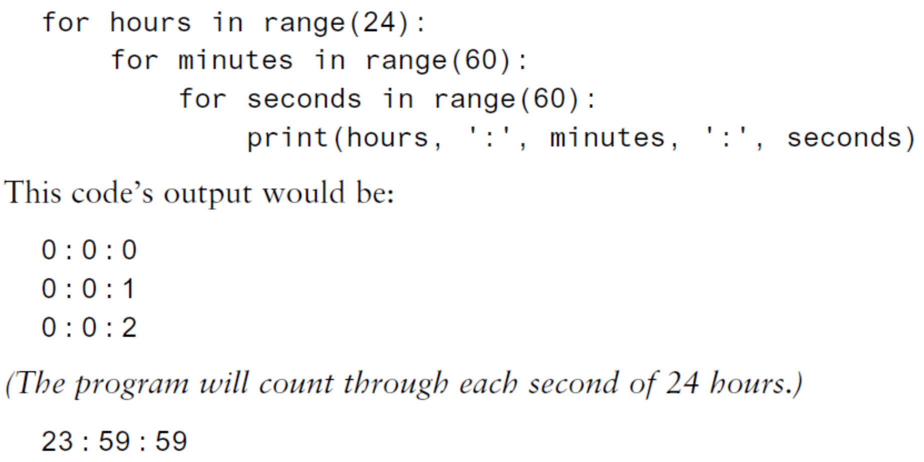
Nested Loops

- **<u>Nested loop</u>: loop that is contained inside another loop**

<u>Example:</u> digital clock works like a nested loop

hh:mm:ss

- Hours hand moves once for every twelve movements of the minutes hand: for each iteration of the "hours," do twelve iterations of "minutes"
- Seconds hand moves 60 times for each movement of the minutes hand: for each iteration of "minutes," do 60 iterations of "seconds"



```
for hours in range(24):
    for minutes in range(60):
        for seconds in range(60):
            print(hours, ':', minutes, ':', seconds)
```

This code's output would be:

```
0 : 0 : 0
0 : 0 : 1
0 : 0 : 2
```

(The program will count through each second of 24 hours.)

```
23 : 59 : 59
```

Review this example Page 212
and
Review Example Program 4-17 Page 214

Nested Loops (cont'd.)

- **Key points about nested loops:**
  - Inner loop goes through all of its iterations for each iteration of outer loop
  - Inner loops complete their iterations faster than outer loops
  - Total number of iterations in nested loop:

```
number_iterations_inner  x number_iterations_outer
```

Example: How many times does the following code print Hello?

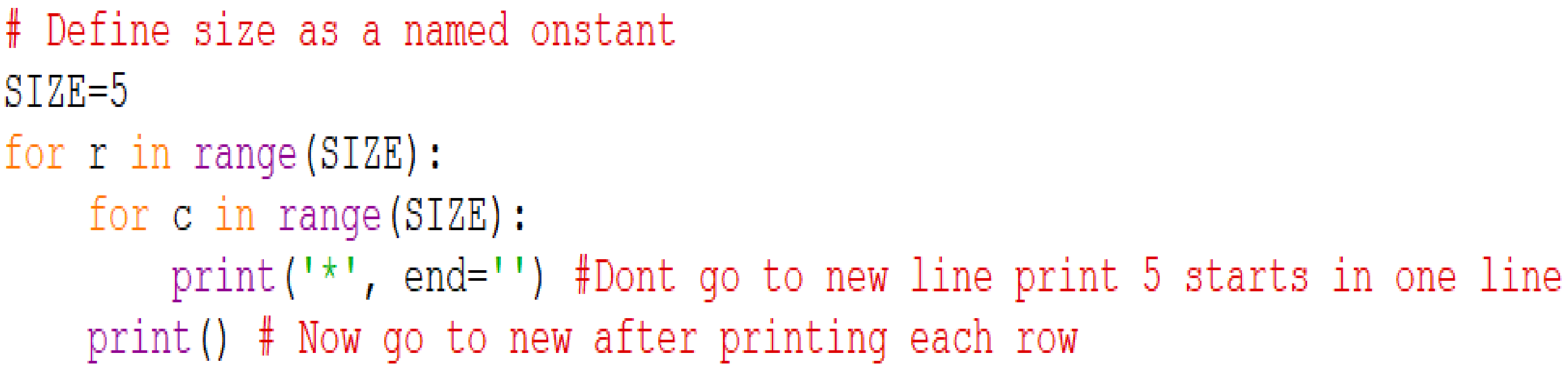
```
for i in range(60):
    for j in range(6):
        print('Hello')
```

Example: Nested Loop 1/3 Page 215

Write a program that displays 5 by 5 square asterisk.

Program Output

```
*****
*****
*****
*****
*****
```



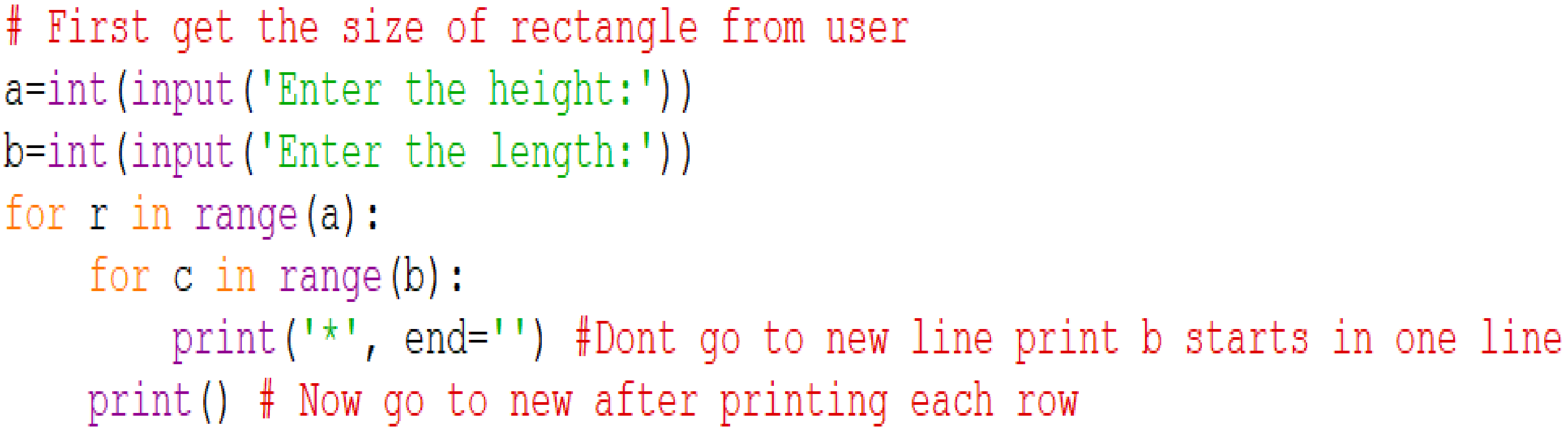
```
# Define size as a named onstant
SIZE=5
for r in range(SIZE):
    for c in range(SIZE):
        print('*', end='') #Dont go to new line print 5 starts in one line
    print() # Now go to new after printing each row
```

Example: Nested Loop 2/3

Write a program that displays a rectangular asterisk with a desired size.

Program Output

```
Enter the height:3
Enter the length:7
*******
*******
*******
```



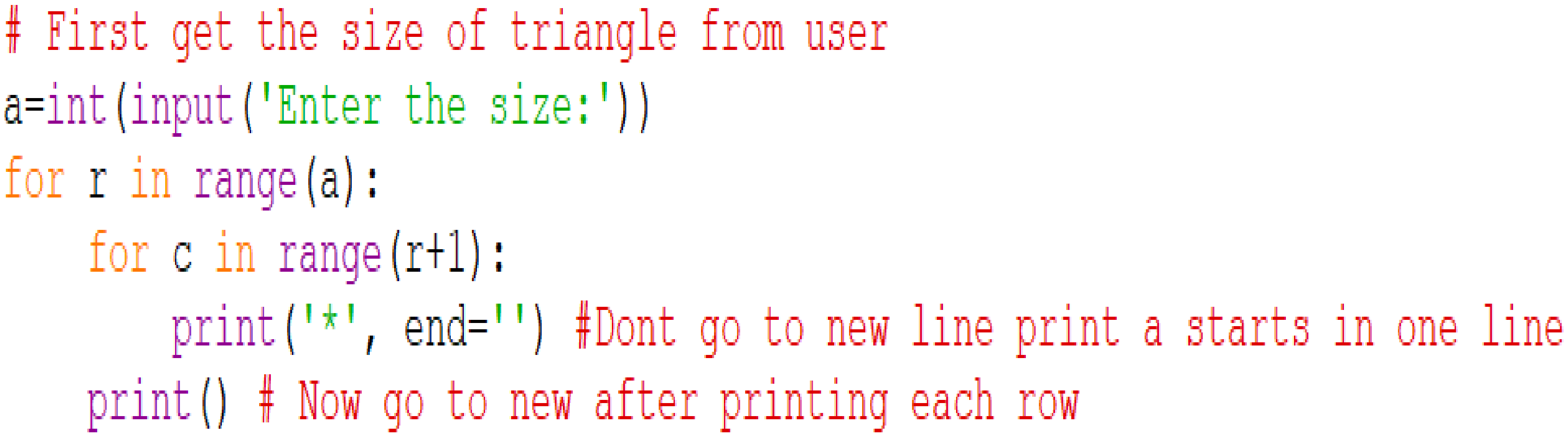
```
# First get the size of rectangle from user
a=int(input('Enter the height:'))
b=int(input('Enter the length:'))
for r in range(a):
    for c in range(b):
        print('*', end='') #Dont go to new line print b starts in one line
    print() # Now go to new after printing each row
```

Example: Nested Loop 3/3

Write a program that displays a right angle triangle asterisk with a desired size.

Program Output

```
Enter the size:4
*
**
***
```



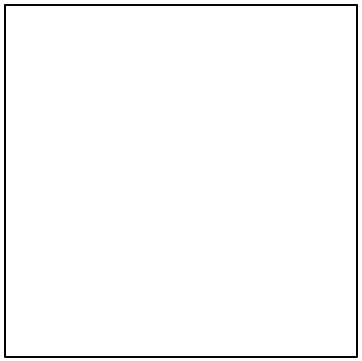
```
# First get the size of triangle from user
a=int(input('Enter the size:'))
for r in range(a):
    for c in range(r+1):
        print('*', end='') #Dont go to new line print a starts in one line
    print() # Now go to new after printing each row
```

Turtle Graphics: Using Loops to Draw Designs

- You can use loops with the turtle to draw both simple shapes and elaborate designs.

Example: the following for loop iterates four times to draw a square that is 100 pixels wide:

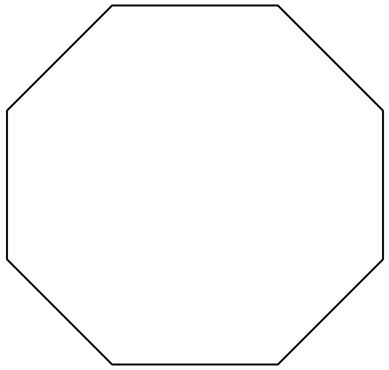
```
import turtle
for x in range(4):
    turtle.forward(100)
    turtle.right(90)
```



Turtle Graphics: Using Loops to Draw Designs

- **This `for` loop iterates eight times to draw the octagon:**

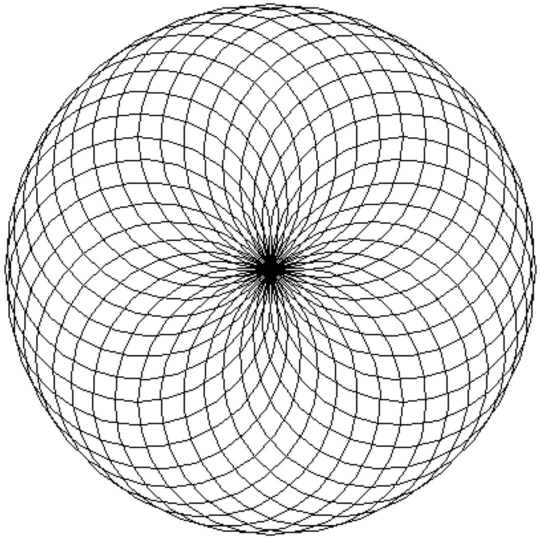
```
import turtle
for x in range(8):
    turtle.forward(100)
    turtle.right(45)
```



Turtle Graphics: Using Loops to Draw Designs

- **You can create interesting designs by repeatedly drawing a simple shape, with the turtle tilted at a slightly different angle each time it draws the shape.**

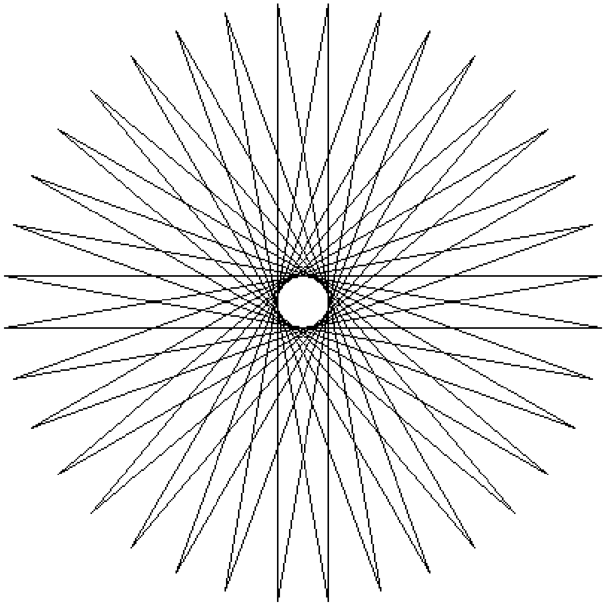
```
import turtle
NUM_CIRCLES = 36     # Number of circles to draw
RADIUS = 100         # Radius of each circle
ANGLE = 10           # Angle to turn

for x in range(NUM_CIRCLES):
    turtle.circle(RADIUS)
    turtle.left(ANGLE)
```



Turtle Graphics: Using Loops to Draw Designs

- **This code draws a sequence of 36 straight lines to make a "starburst" design.**

```
import turtle
START_X = -200          # Starting X coordinate
START_Y = 0             # Starting Y coordinate
NUM_LINES = 36          # Number of lines to draw
LINE_LENGTH = 400       # Length of each line
ANGLE = 170             # Angle to turn

turtle.hideturtle()
turtle.penup()
turtle.goto(START_X, START_Y)
turtle.pendown()

for x in range(NUM_LINES):
    turtle.forward(LINE_LENGTH)
    turtle.left(ANGLE)
```



Review Questions

Page 223

Multiple Choice

5. A(n) __________ loop has no way of ending and repeats until the program is interrupted.
   a. indeterminate
   b. interminable
   [✓ c. infinite]
   d. timeless

6. The -= operator is an example of a(n) __________ operator.
   a. relational
   [✓ b. augmented assignment]
   c. complex assignment
   d. reverse assignment

7. A(n) __________ variable keeps a running total.
   a. sentinel
   b. sum
   c. total
   [✓ d. accumulator]

8. A(n) __________ is a special value that signals when there are no more items from a list of items to be processed. This value cannot be mistaken as an item from the list.
   [✓ a. sentinel]
   b. flag
   c. signal
   d. accumulator

Algorithm Workbench

Page 225

5. Write a loop that calculates the total of the following series of numbers:

$$\frac{1}{30} + \frac{2}{29} + \frac{3}{28} + \ldots \frac{30}{1}$$

```
sum=0.0
for i in range(1,30):
        sum+=(i/(31-i))
```

6. Rewrite the following statements using augmented assignment operators.

```
a. x = x + 1          x=+1
b. x = x * 2          x*=2
c. x = x / 10         x/=10
d. x = x - 100        x-=100
```

7. Write a set of nested loops that displays the first ten values of the multiplication tables from 1 to 10. The code should print 100 lines in total, starting at “1 × 1 = 1” and ending with “10 × 10 = 100”.

```
for i in range(1,10):
      for j in range(1,10):
            print( i,'X',j,'=',i*j)
```

9. Write code that prompts the user to enter a number in the range of 1 through 100 and validates the input.

```
num=(inputint('Enter a number 1-100:'))
while ( num < 1 or num > 100):
        num=int(input('Number must be in the range 1-100:'))
```

Programming Exercises

Page 226

5. Average Rainfall

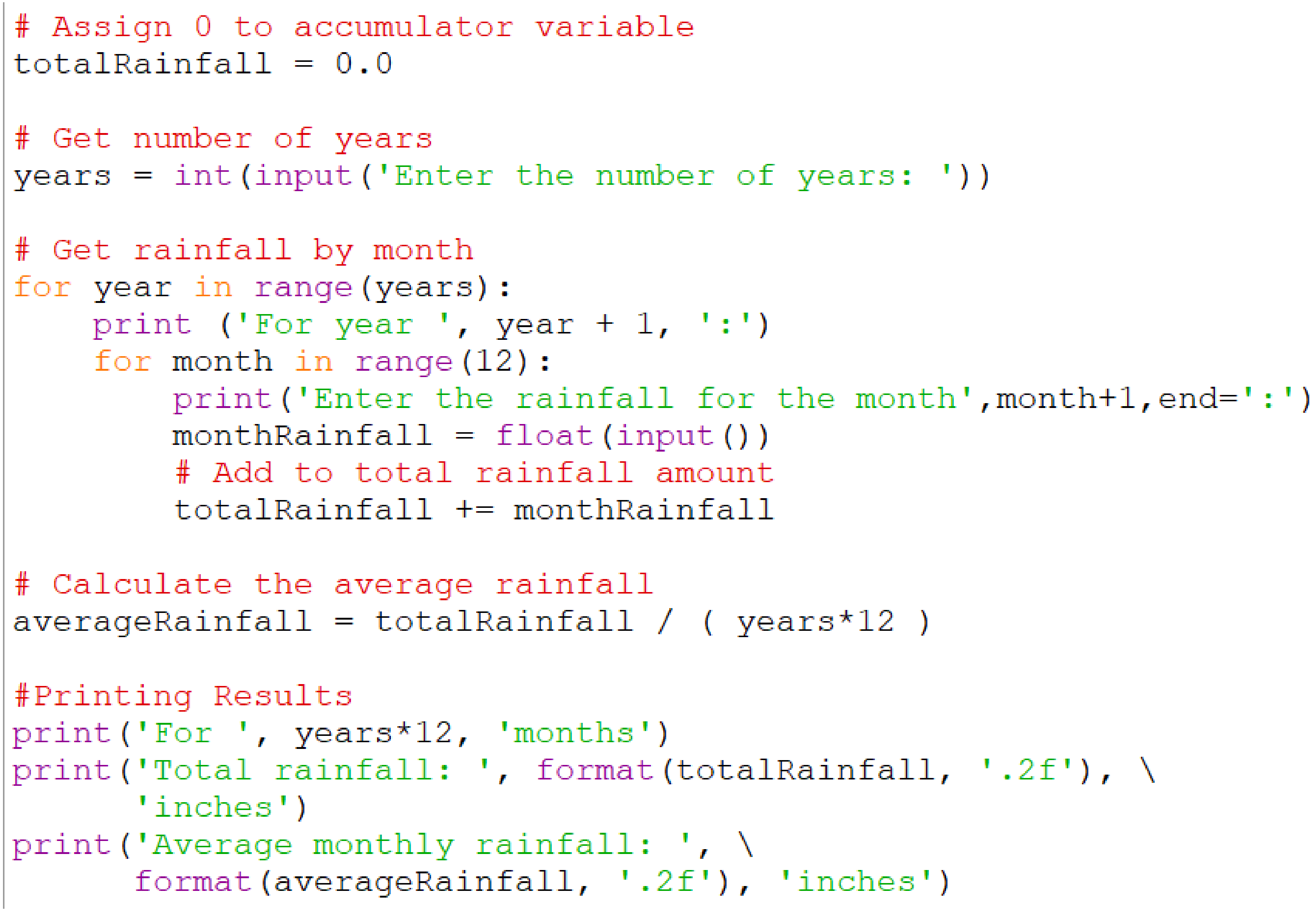
Write a program that uses nested loops to collect data and calculate the average rainfall over a period of years. The program should first ask for the number of years. The outer loop will iterate once for each year. The inner loop will iterate twelve times, once for each month. Each iteration of the inner loop will ask the user for the inches of rainfall for that month. After all iterations, the program should display the number of months, the total inches of rainfall, and the average rainfall per month for the entire period.

Programming Exercises

Program

Page 226

5. Average Rainfall



```
# Assign 0 to accumulator variable
totalRainfall = 0.0

# Get number of years
years = int(input('Enter the number of years: '))

# Get rainfall by month
for year in range(years):
    print ('For year ', year + 1, ':')
    for month in range(12):
        print('Enter the rainfall for the month',month+1,end=':')
        monthRainfall = float(input())
        # Add to total rainfall amount
        totalRainfall += monthRainfall

# Calculate the average rainfall
averageRainfall = totalRainfall / ( years*12 )

#Printing Results
print('For ', years*12, 'months')
print('Total rainfall: ', format(totalRainfall, '.2f'), \
      'inches')
print('Average monthly rainfall: ', \
      format(averageRainfall, '.2f'), 'inches')
```

Programming Exercises

Page 226

8. Average Word Length

Write a program with a loop that repeatedly asks the user to enter a word. The user should enter nothing (press Enter without typing anything) to signal the end of the loop. Once the loop ends, the program should display the average length of the words entered, rounded to the nearest whole number.

Program

```
# Initialise accumulator variables.
word_count = 0
total_length = 0
# Input the first word before entering to the while loop
word = input('Enter a word (enter nothing to quit): ')

# Repeat while loop as long as word is not an empty string.
while word != '':
    # Increment the word count and add to the total length.
    word_count += 1
    total_length += len(word)

    # Get another word.
    word = input('Enter a word (enter nothing to quit): ')

# Display average word length (or a message if no words entered).
# if decision structure needed just in case user didn't enter anything
if word_count == 0:
    print('No words entered!')
else:
    print('Average Word Length:', round(total_length / word_count))
```

Programming Exercises

Page 227

12. Calculating the Factorial of a Number

In mathematics, the notation *n*! represents the factorial of the nonnegative integer *n*. The factorial of *n* is the product of all the nonnegative integers from 1 to *n*. For example,

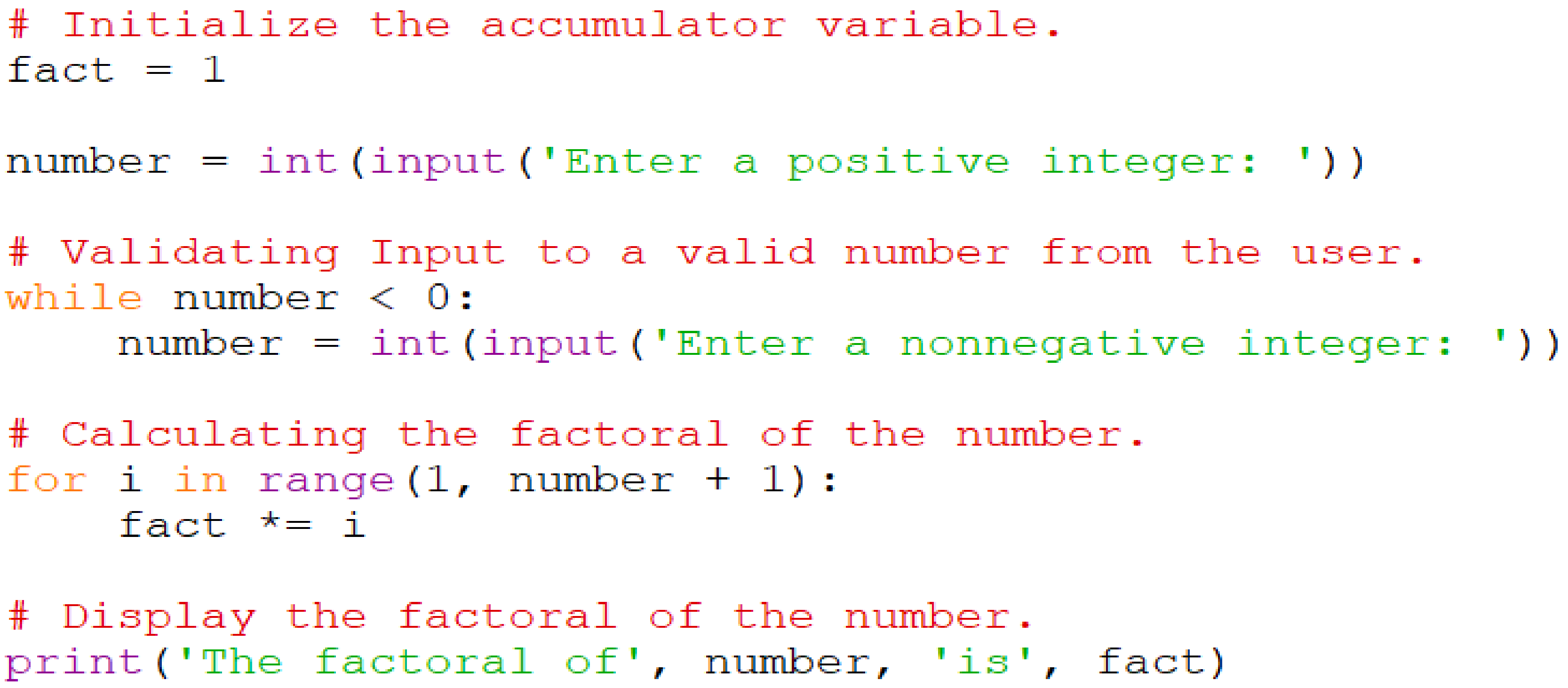
7! = 1 x 2 x 3 x 4 x 5 x 6 x 7 = 5040

and

4! = 1 x 2 x 3 x 4 = 24

Write a program that lets the user enter a nonnegative integer then uses a loop to calculate the factorial of that number. Display the factorial.

Program



```
# Initialize the accumulator variable.
fact = 1

number = int(input('Enter a positive integer: '))

# Validating Input to a valid number from the user.
while number < 0:
    number = int(input('Enter a nonnegative integer: '))

# Calculating the factoral of the number.
for i in range(1, number + 1):
    fact *= i

# Display the factoral of the number.
print('The factoral of', number, 'is', fact)
```

Programming Exercises

Page 227

13. Population

Write a program that predicts the approximate size of a population of organisms. The application should use text boxes to allow the user to enter the starting number of organisms, the average daily population increase (as a percentage), and the number of days the organisms will be left to multiply. **For example**, assume the user enters the following values:

Starting number of organisms: 2

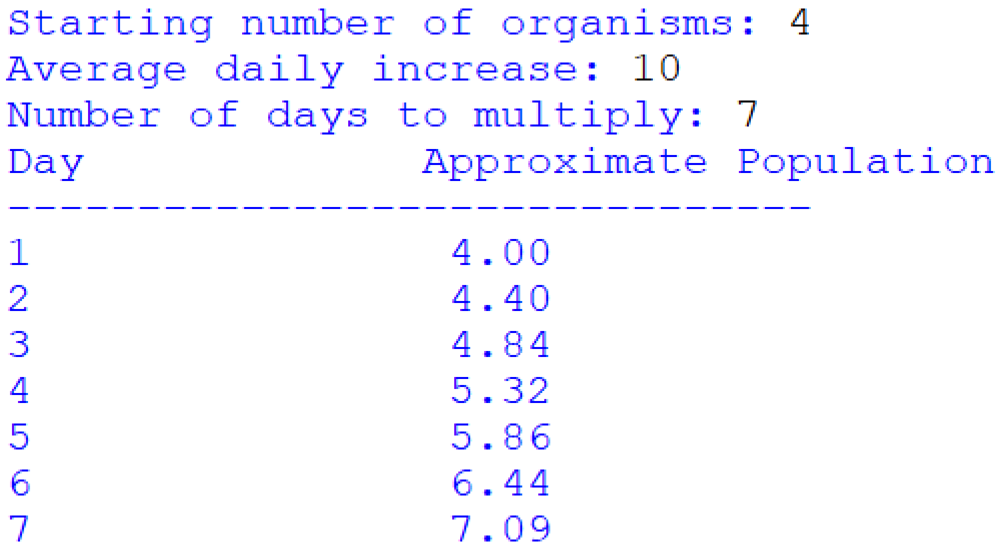
Average daily increase: 30%

Number of days to multiply: 10

The program should display the following table of data:

Day	**Approximate Population**
1	2.00
2	2.60
3	3.38
4	4.39
5	5.71
6	7.43
7	9.65
8	12.50
9	16.31
10	21.21

Example Program Output

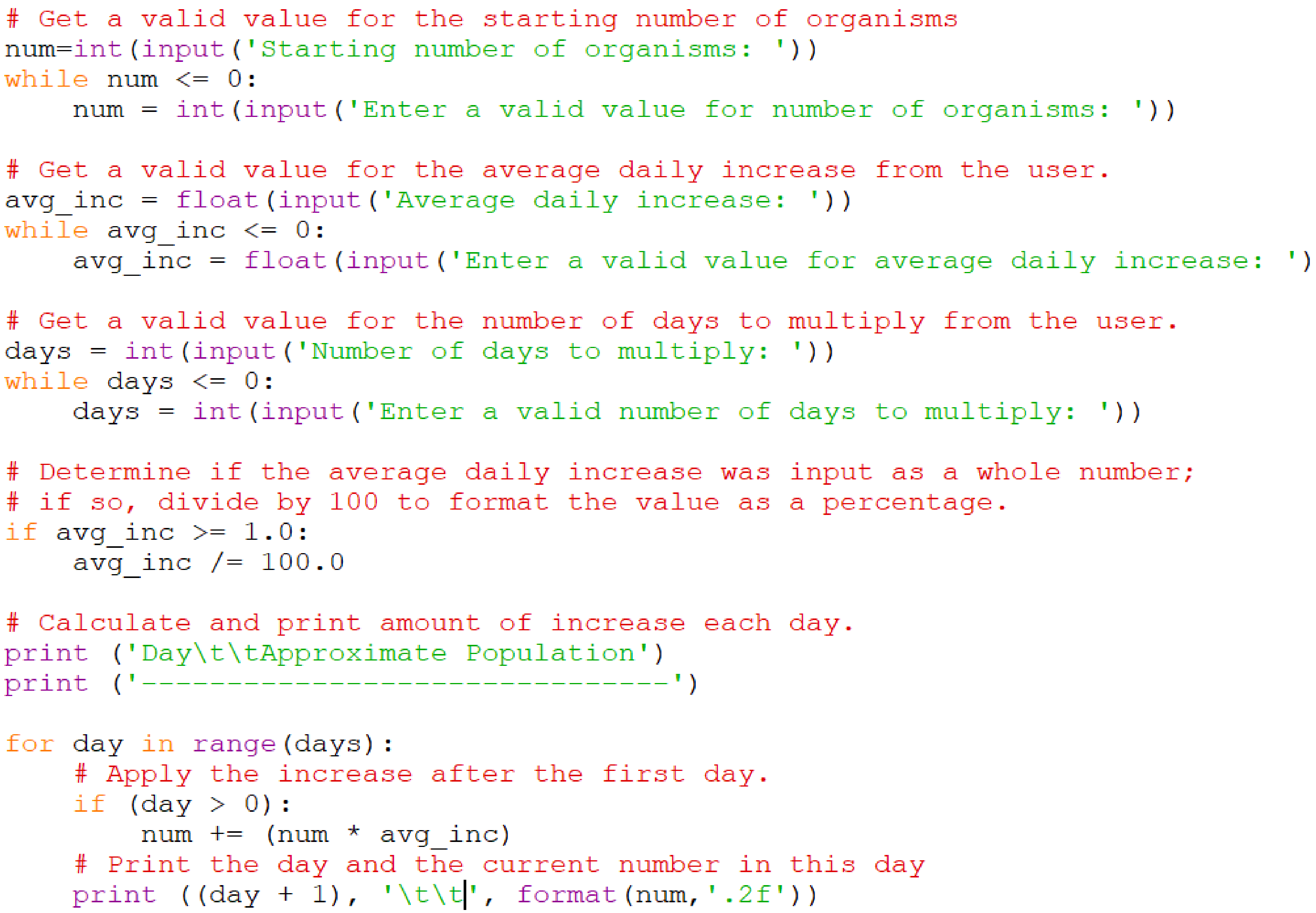


```
Starting number of organisms: 4
Average daily increase: 10
Number of days to multiply: 7
Day             Approximate Population
-------------------------------
1                4.00
2                4.40
3                4.84
4                5.32
5                5.86
6                6.44
7                7.09
```

Programming Exercises

Page 227

13. Population Program



```
# Get a valid value for the starting number of organisms
num=int(input('Starting number of organisms: '))
while num <= 0:
    num = int(input('Enter a valid value for number of organisms: '))

# Get a valid value for the average daily increase from the user.
avg_inc = float(input('Average daily increase: '))
while avg_inc <= 0:
    avg_inc = float(input('Enter a valid value for average daily increase: ')

# Get a valid value for the number of days to multiply from the user.
days = int(input('Number of days to multiply: '))
while days <= 0:
    days = int(input('Enter a valid number of days to multiply: '))

# Determine if the average daily increase was input as a whole number;
# if so, divide by 100 to format the value as a percentage.
if avg_inc >= 1.0:
    avg_inc /= 100.0

# Calculate and print amount of increase each day.
print ('Day\t\tApproximate Population')
print ('-------------------------------')

for day in range(days):
    # Apply the increase after the first day.
    if (day > 0):
        num += (num * avg_inc)
    # Print the day and the current number in this day
    print ((day + 1), '\t\t', format(num,'.2f'))
```

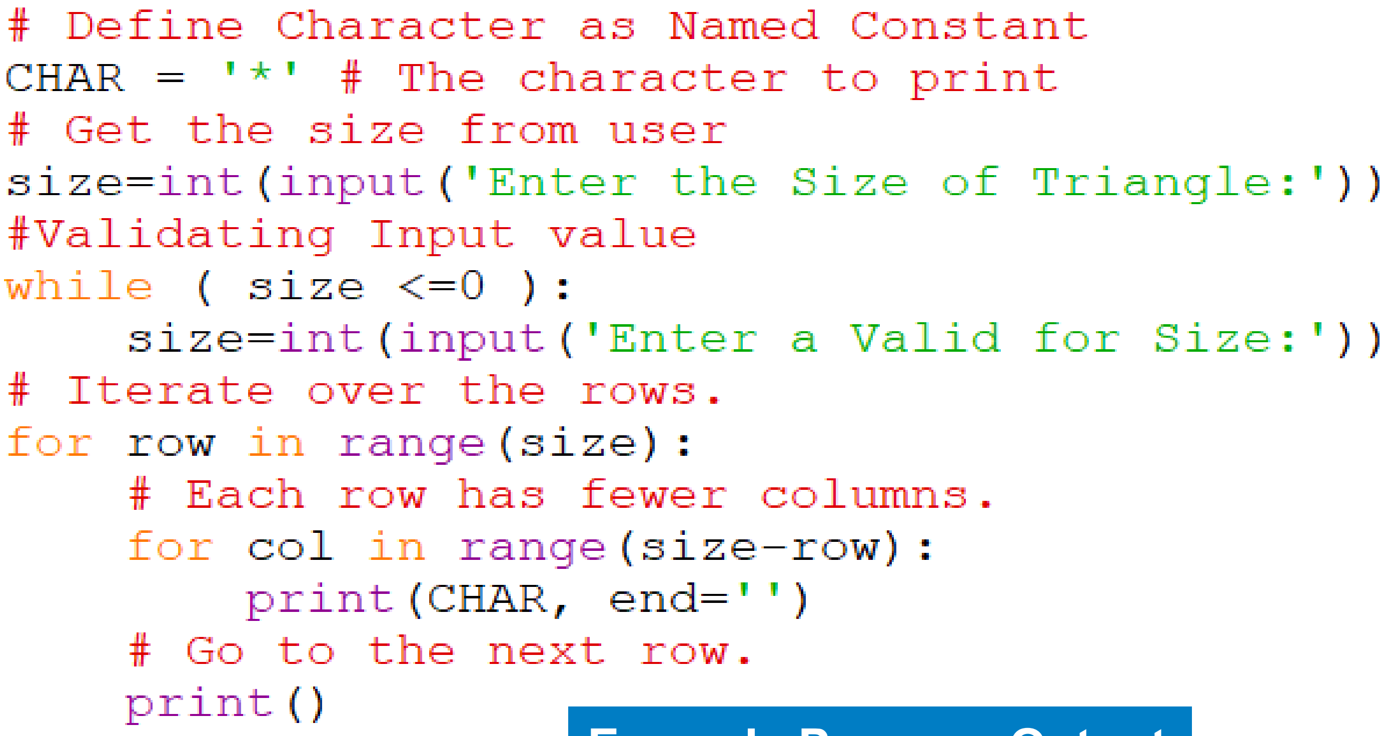
Programming Exercises

Page 228

14. Write a program that uses nested loops to draw this pattern: consider a general program which draw this figure for any size.

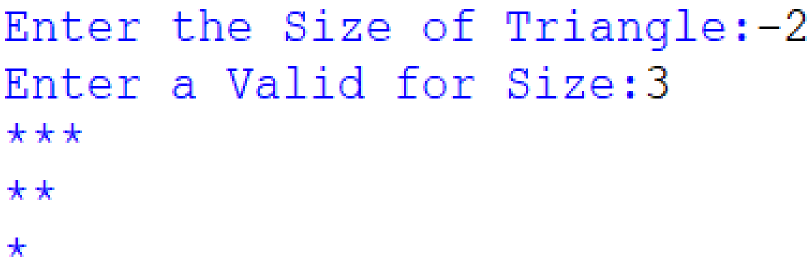
```
*******
******
*****
****
***
**
*
```

Program



```
# Define Character as Named Constant
CHAR = '*' # The character to print
# Get the size from user
size=int(input('Enter the Size of Triangle:'))
#Validating Input value
while ( size <=0 ):
    size=int(input('Enter a Valid for Size:'))
# Iterate over the rows.
for row in range(size):
    # Each row has fewer columns.
    for col in range(size-row):
        print(CHAR, end='')
    # Go to the next row.
    print()
```

Example Program Output



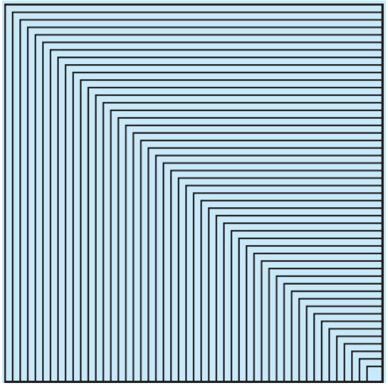
```
Enter the Size of Triangle:-2
Enter a Valid for Size:3
***
**
*
```

Programming Exercises

Page 228

16. Turtle Graphics: Repeating Squares

In this chapter, you saw an example of a loop that draws a square. Write a turtle graphics program that uses nested loops to draw 100 squares, to create the design shown in Figure.



Program

```
import turtle
# Named constants
STARTING_SIZE = 5
NUM_SQUARES = 100
STEP = 5
NUM_SIDES = 4
ANGLE = 90
turtle.speed(0)

# Set the inital x & y coordinates
turtle.penup()
turtle.goto(200, -200)
turtle.pendown()

# Set the initial size of the square
size = STARTING_SIZE
# Draw the pattern.
for count in range(NUM_SQUARES):
    # Draw the square
    for s in range(NUM_SIDES):
        turtle.left(ANGLE)
        turtle.forward(size)
    # Prepare for the next square.
    size = size + STEP

# Hide the turtle.
turtle.hideturtle()
```

Summary

- **This chapter covered:**
  - Repetition structures, including:
    - Condition-controlled loops
    - Count-controlled loops
    - Nested loops
  - Infinite loops and how they can be avoided
  - `range` function as used in `for` loops
  - Calculating a running total and augmented assignment operators
  - Use of sentinels to terminate loops
  - Using loops to draw turtle graphic designs