

## CHAPTER 4

# Repetition Structures

starting out with >>>

# PYTHON®

FOURTH EDITION



TONY GADDIS



Pearson

Copyright © 2018 Pearson Education, Inc.

# 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**
  - Disadvantages to duplicating code
    - Makes program large
    - Time consuming
    - May need to be corrected in many places
- **Repetition structure: makes computer repeat included code as necessary**
  - Includes condition-controlled loops and count-controlled loops



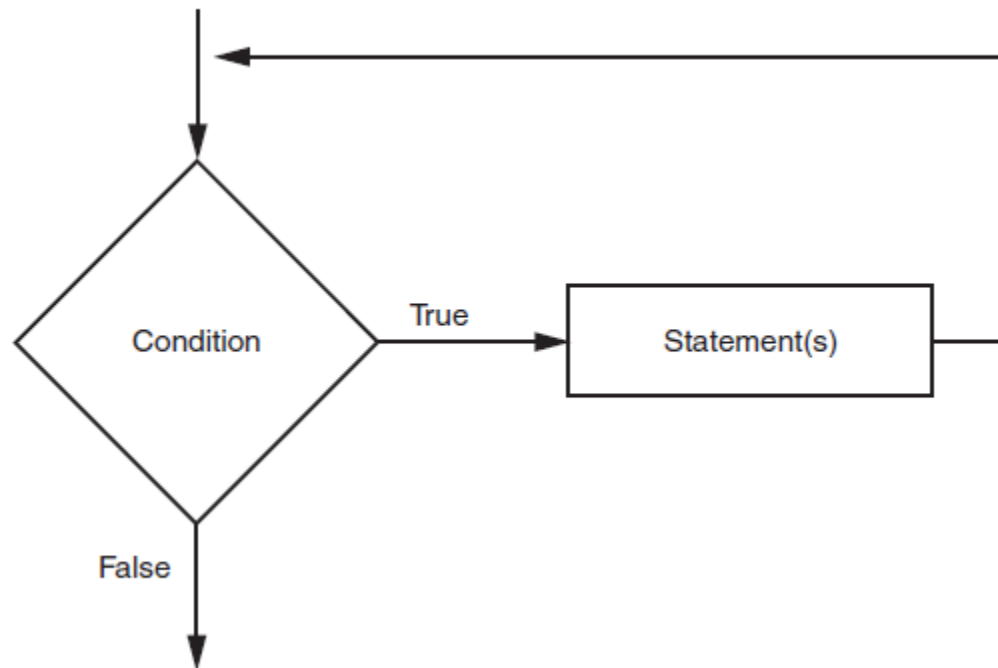
# The `while` Loop: a Condition-Controlled Loop

- `while` loop: 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.)

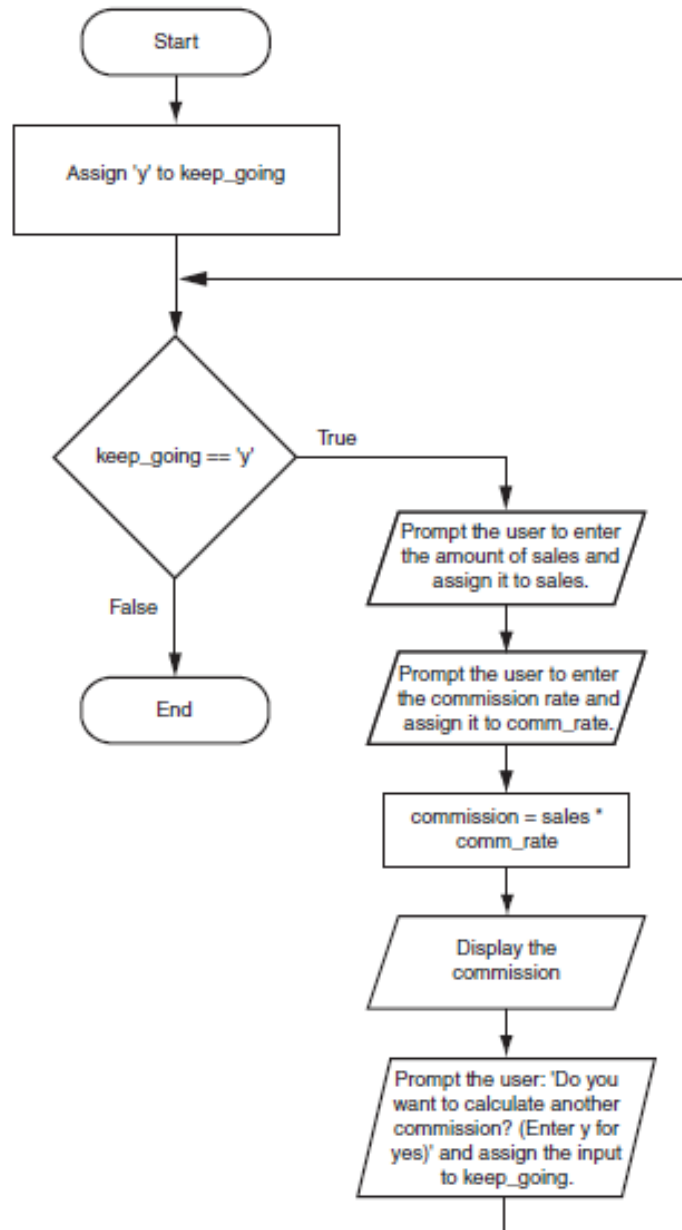
**Figure 4-1** The logic of a while loop



# 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
- 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

Figure 4-3 Flowchart for Program 4-1



# 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



# The `for` Loop: a Count-Controlled Loop

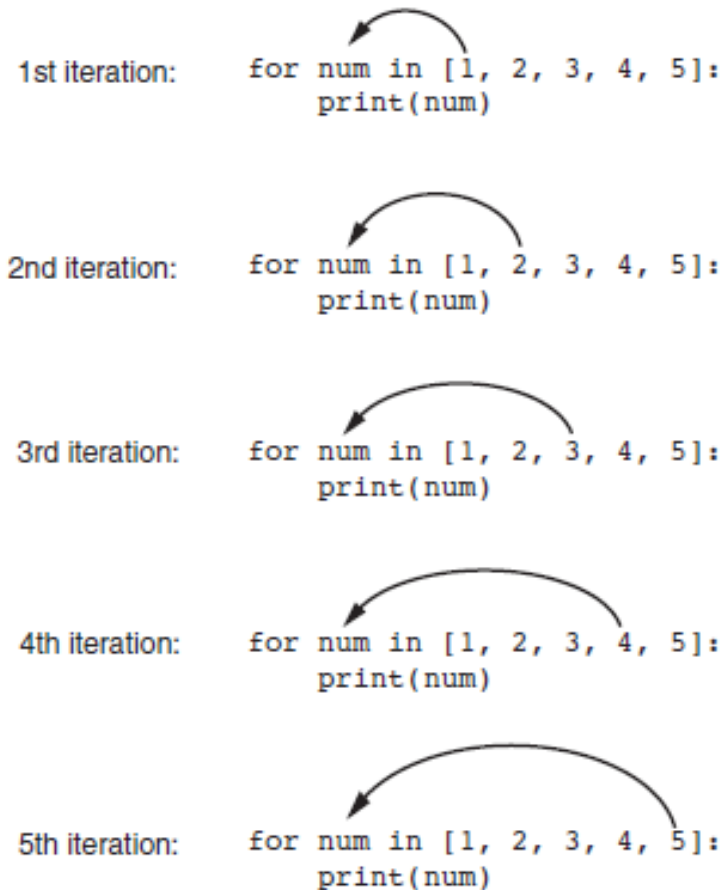
- **Count-Controlled loop**: iterates a specific number of times
  - Use a `for` statement to write count-controlled loop
    - Designed to work with sequence of data items
      - Iterates once for each item in the sequence
    - General format:

```
for variable in [val1, val2, etc]:  
    statements
```
    - **Target variable**: the variable which is the target of the assignment at the beginning of each iteration



**Figure 4-4** The for loop

---



# Using the range Function with the for Loop

- **The range function simplifies the process of writing a for loop**
  - `range` returns an iterable object
    - Iterable: contains a sequence of values that can be iterated over
- **range characteristics:**
  - One argument: used as ending limit
  - Two arguments: starting value and ending limit
  - Three arguments: third argument is step value



# 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: calculate square root of each number in a range

# 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

# Generating an Iterable Sequence that Ranges from Highest to Lowest

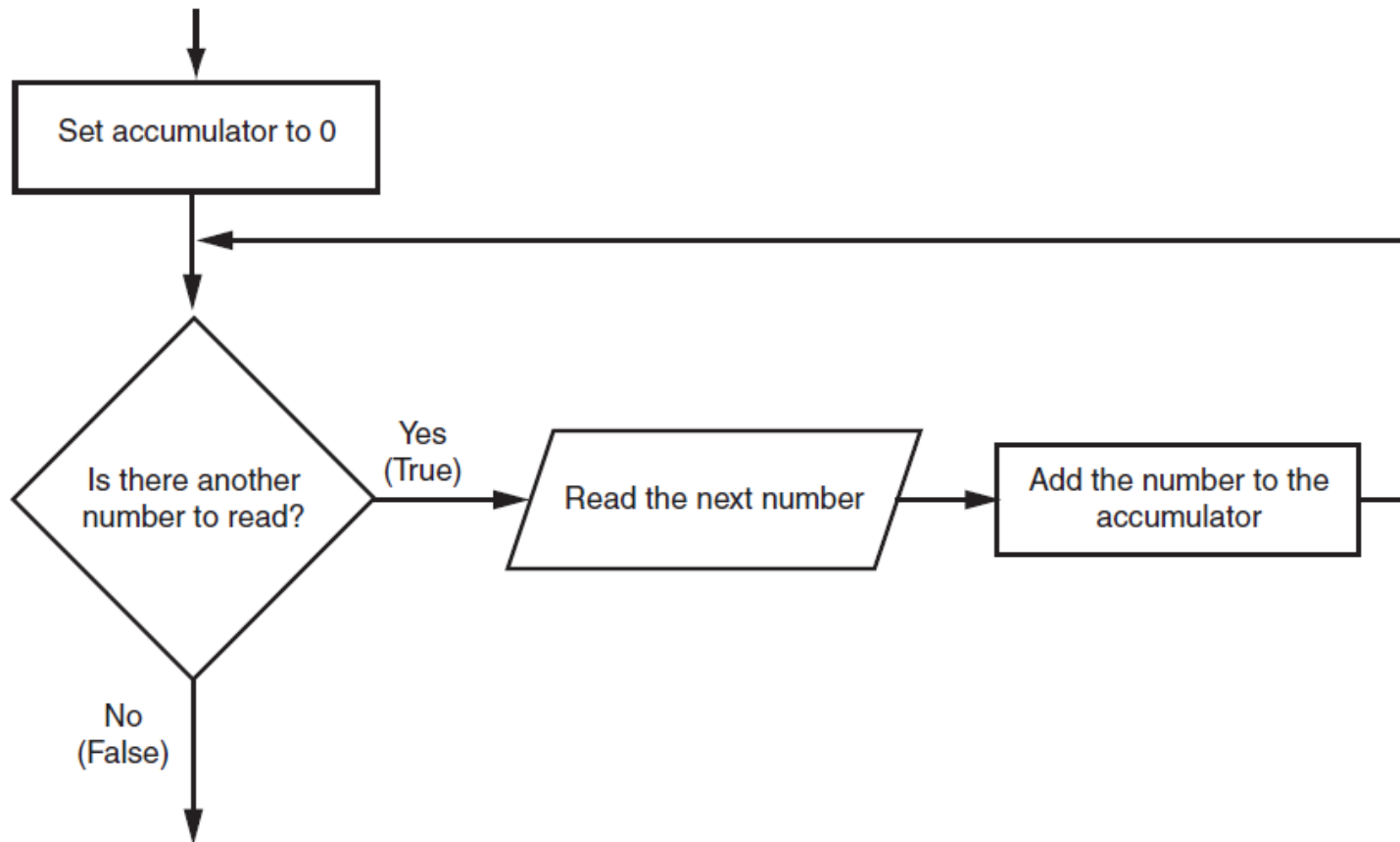
- The `range` function can be used to generate a sequence with numbers in descending order
  - Make sure starting number is larger than end limit, and step value is negative
  - Example: `range (10, 0, -1)`

# 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 4-6** Logic for calculating a running total





# 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
- Augmented assignment operators: special set of operators designed for this type of job
  - Shorthand operators

# The Augmented Assignment Operators (cont'd.)

**Table 4-2** Augmented assignment operators

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

# Sentinels

- **Sentinel: special value that marks the end of a sequence of items**
  - When program reaches a sentinel, it knows that the end of the sequence of items was reached, and the loop terminates
  - Must be distinctive enough so as not to be mistaken for a regular value in the sequence
  - Example: when reading an input file, empty line can be used as a sentinel

# Input Validation Loops

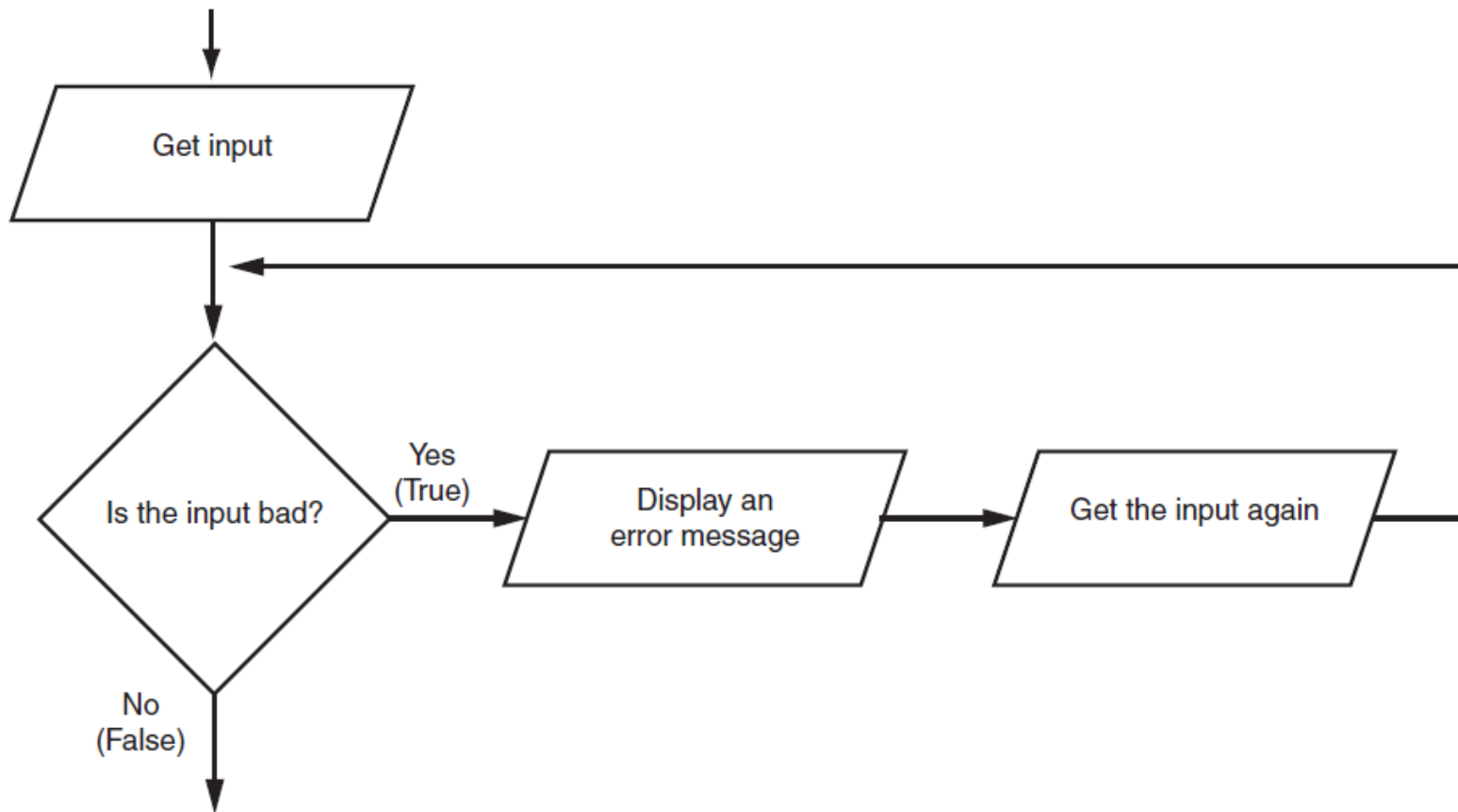
- **Computer cannot tell the difference between good data and bad data**
  - If user provides bad input, program will produce bad output
  - GIGO: garbage in, garbage out
  - It is important to design program such that bad input is never accepted

# Input Validation Loops (cont'd.)

- **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.)

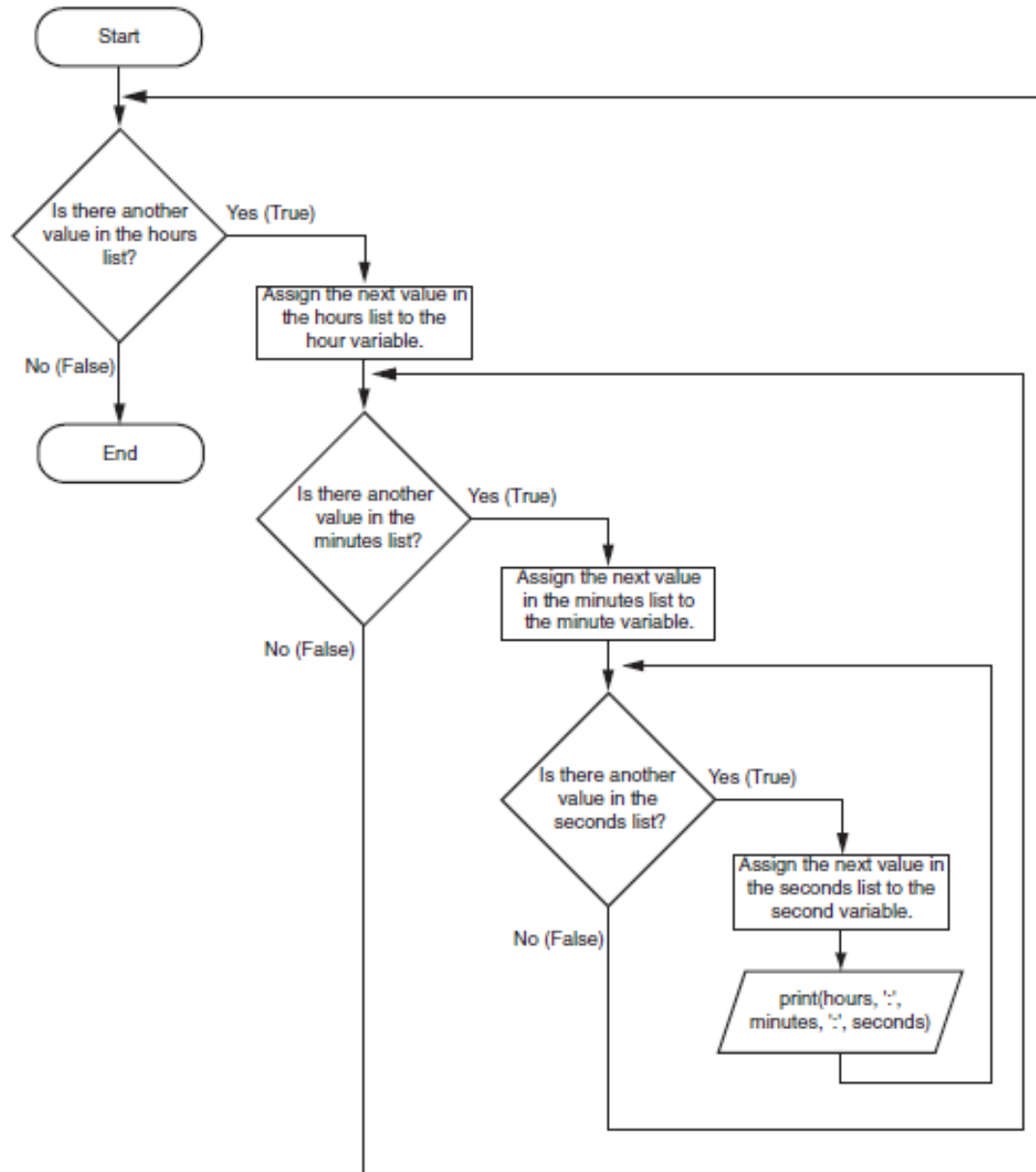
**Figure 4-7** Logic containing an input validation loop



# Nested Loops

- **Nested loop: loop that is contained inside another loop**
  - Example: analog clock works like a nested loop
    - 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”

**Figure 4-8** Flowchart for a clock simulator





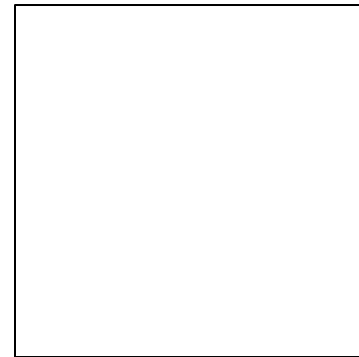
# Nested Loops (cont'd.)

- **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:  
$$\text{number\_iterations\_inner} \times \text{number\_iterations\_outer}$$

# Turtle Graphics: Using Loops to Draw Designs

- You can use loops with the turtle to draw both simple shapes and elaborate designs. For example, the following for loop iterates four times to draw a square that is 100 pixels wide:

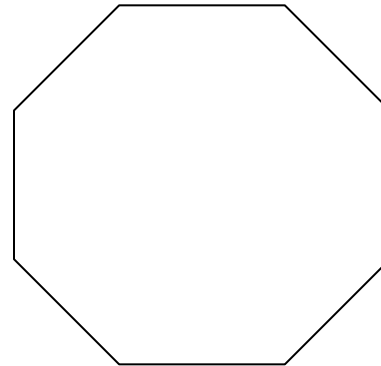
```
for x in range(4):  
    turtle.forward(100)  
    turtle.right(90)
```



# Turtle Graphics: Using Loops to Draw Designs

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

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

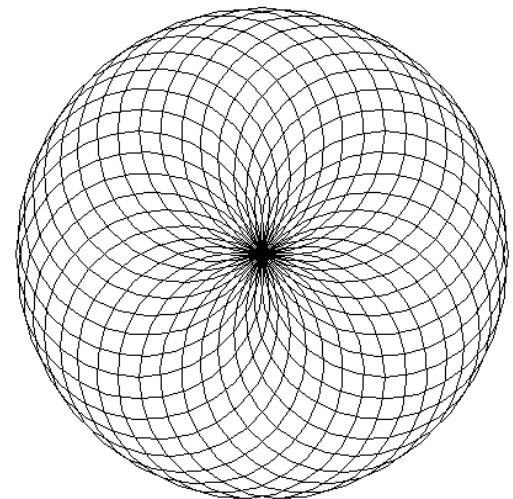


# Turtle Graphics: Using Loops to Draw Designs

- 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.

```
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)
```



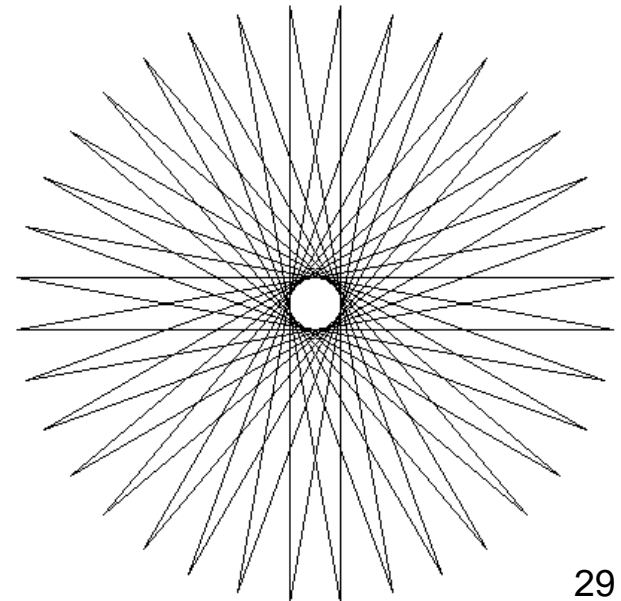
# Turtle Graphics: Using Loops to Draw Designs

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

```
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)
```



# 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