Objectives

- To understand how computers can help solve real problems
- To further explore numeric expressions, variables, and assignment To understand the accumulator pattern
- To utilize the math library
- To further explore simple iteration patterns
- To understand simple selection statements
- To use random numbers to approximate an area

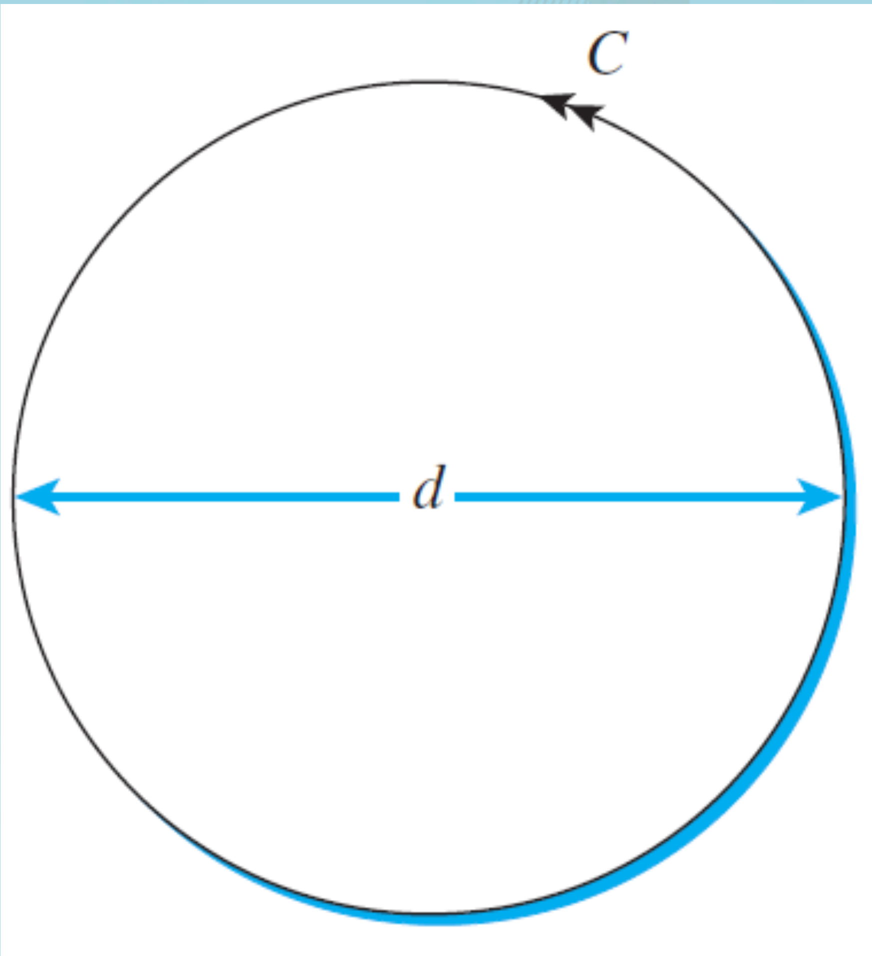
What is PI?

Ratio of circumference to diameter

3.14159265358979323846264338327950288419716939937510...

math.pi from the math module

Figure 2.1



The Archimedes Approach

- Use many sided polygon to approximate circumference of a circle.

- Requires a bit of geometry

Figure 2.2

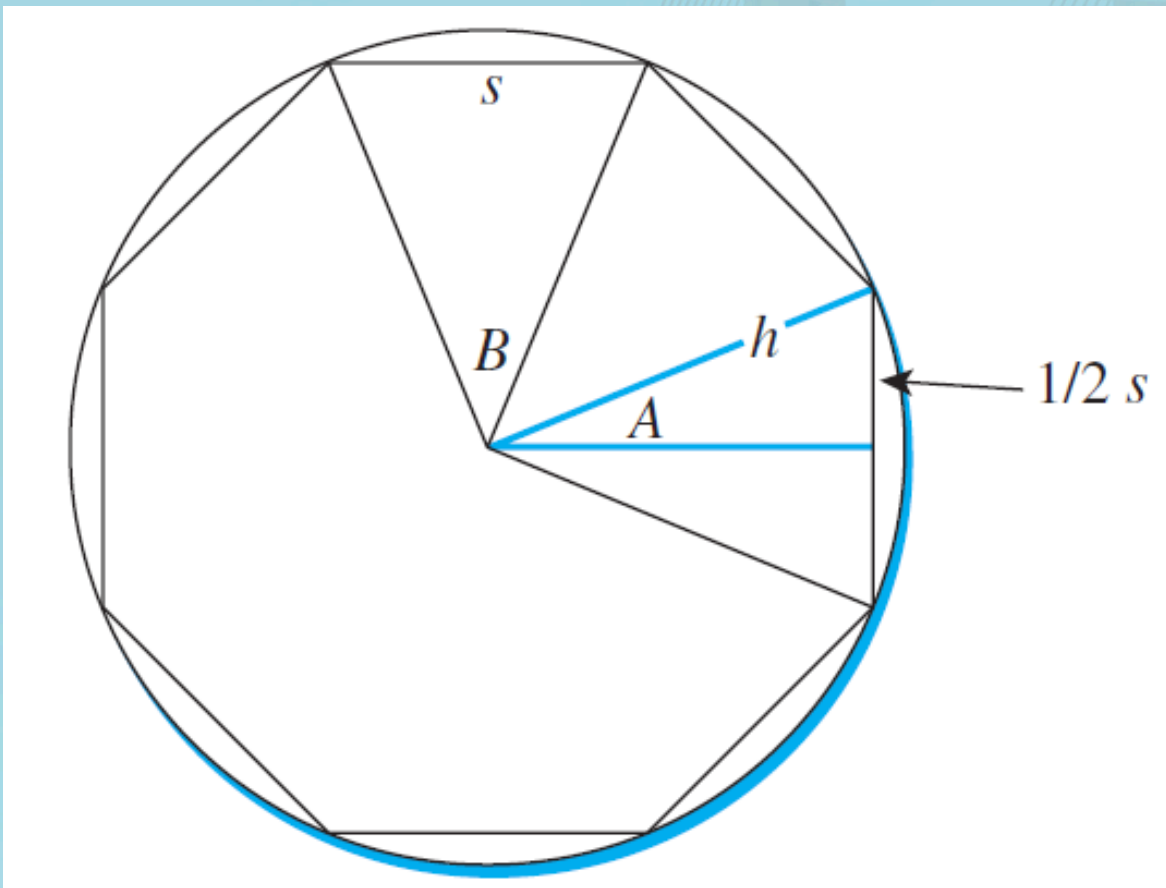
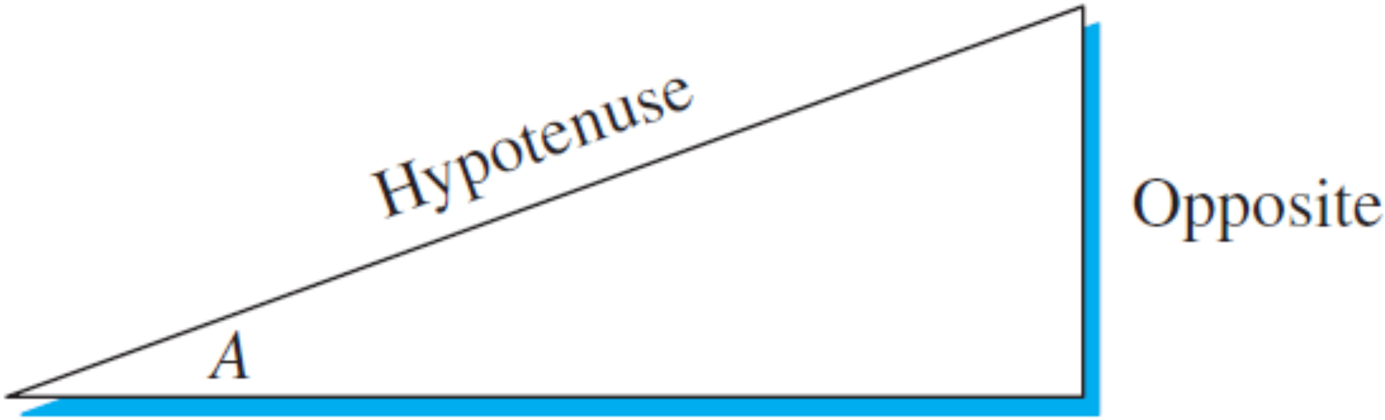


Figure 2.3



$$\sin A = \frac{\text{Opposite}}{\text{Hypotenuse}}$$

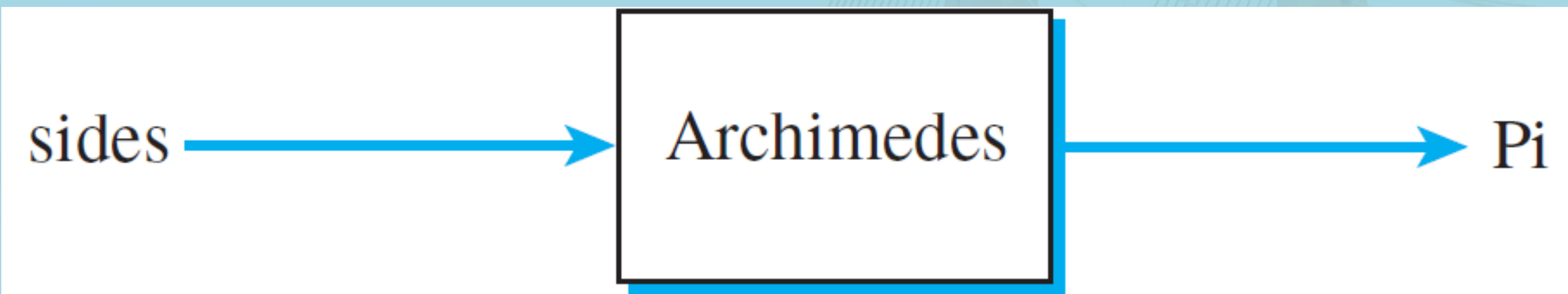
Function

- A name for a sequence of actions
- Can return a value

Listing 2.1

```
def functionName(param1,param2,...):
    statement1
    statement2
    ...
    return expression
```

Figure 2.4



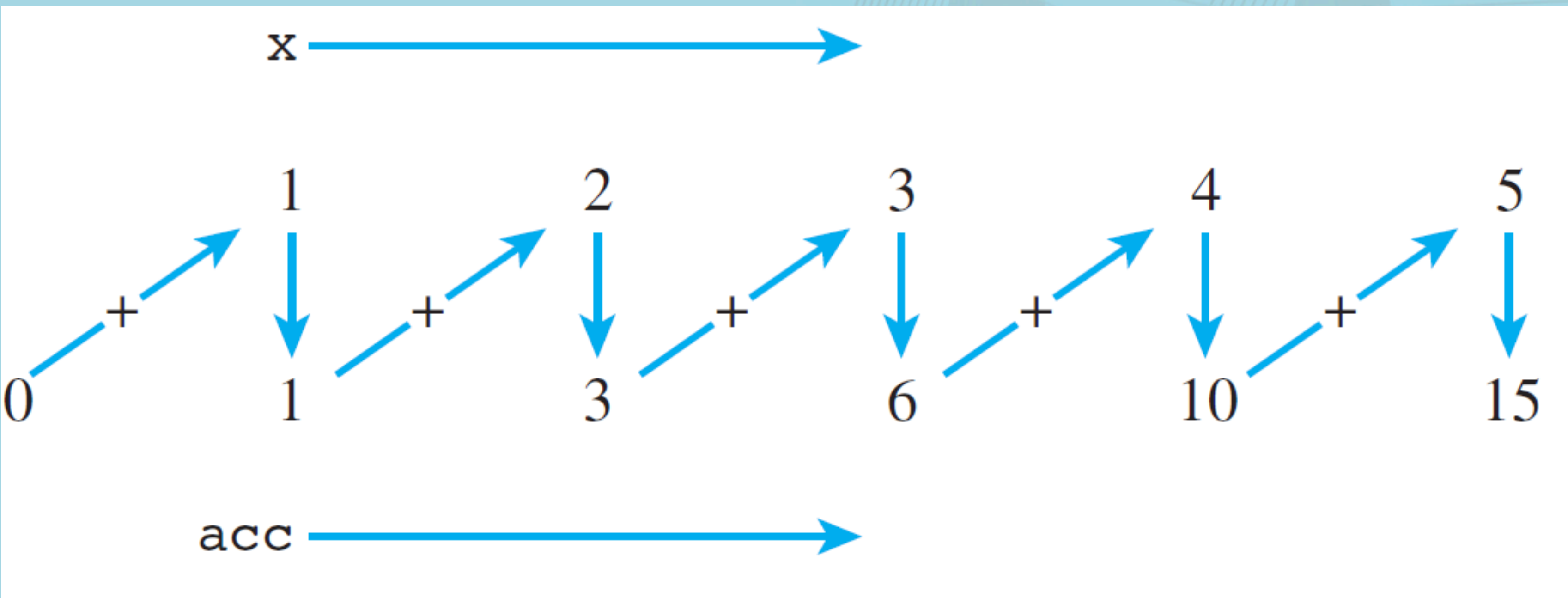
Listing 2.2

```
import math

def archimedes(numSides):

    innerangleB = 360.0/numSides
    halfangleA = innerangleB/2

    onehalfsideS = math.sin(math.radians(halfangleA))

    sideS = onehalfsideS * 2

    polygonCircumference = numSides * sideS
    pi = polygonCircumference/2

    return pi
```

Accumulator Pattern

```
>>> acc=0
>>> for x in range(1,6):
        acc = acc + x
```

Figure 2.5



Leibniz Formula

- Summation of terms
- Use accumulator pattern to add up the terms
- More terms makes the approximation better

Figure 2.6

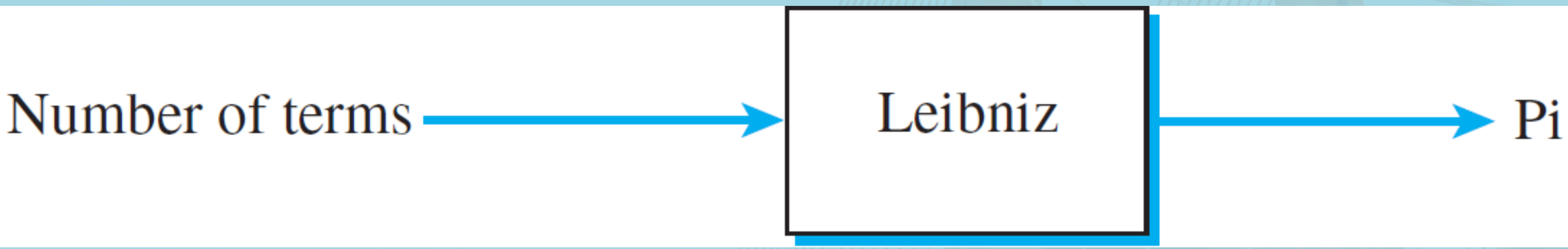
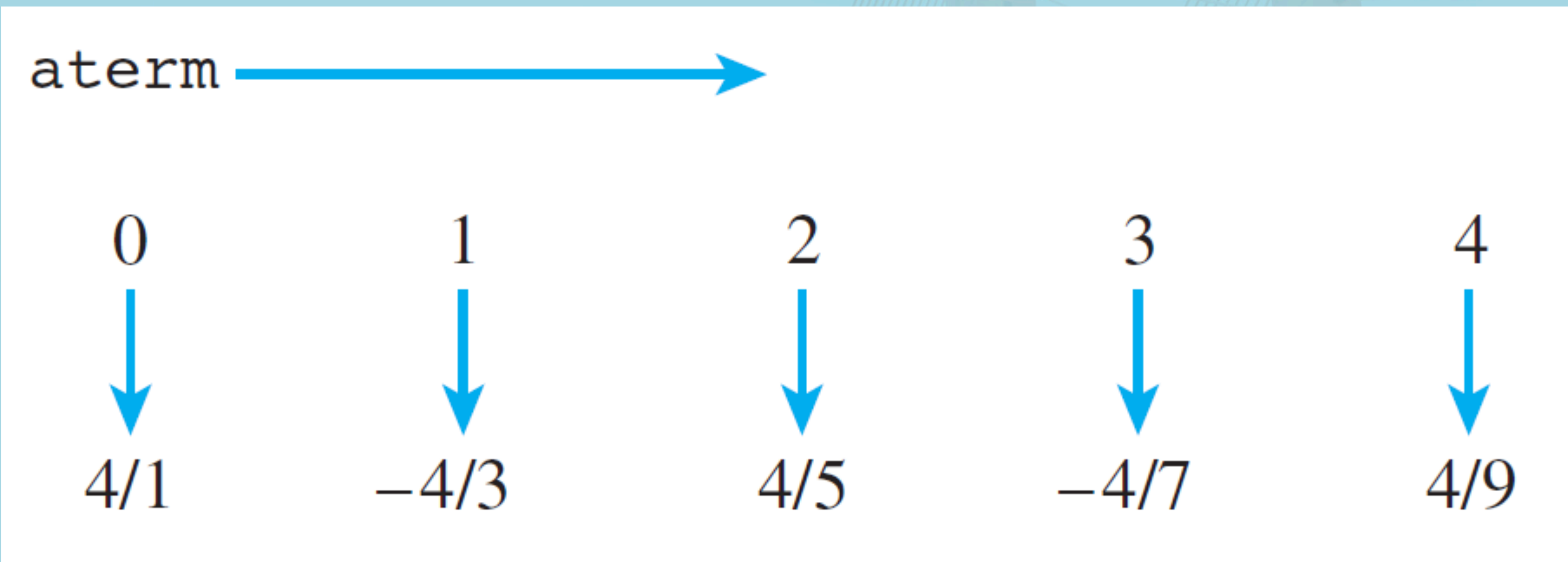


Figure 2.7



Listing 2.3

```
def leibniz(terms):
    acc = 0
    num = 4
    den = 1

    for aterm in range(terms):
        nextterm = num/den * (-1)**aterm

        acc = acc + nextterm

        den = den + 2

    return acc
```

Wallis Formula

- Product of terms
- Use accumulator pattern again
  - This time multiply instead of add
  - Need to initialize with 1 not 0

Figure 2.8

Pair 1	Pair 2	Pair 3
$\frac{2}{1} \times \frac{2}{3}$	$\frac{4}{3} \times \frac{4}{5}$	$\frac{6}{5} \times \frac{6}{7}$

$$\left(\frac{2}{1} \times \frac{2}{3}\right) \times \left(\frac{4}{3} \times \frac{4}{5}\right) \times \left(\frac{6}{5} \times \frac{6}{7}\right) \times \cdots$$

Listing 2.4

```
def wallis(pairs):
  acc = 1
  num = 2
  for apair in range(pairs):
     leftterm = num/(num-1)
     rightterm = num/(num+1)

     acc = acc * leftterm * rightterm

     num = num + 2

  pi = acc * 2
  return pi
```

Monte Carlo Simulation

- Use random numbers to compute an approximation of pi
- Simulation of a special game of darts
- Randomly place darts on the board
- pi can be computed by keeping track of the number of darts that land on the board

Figure 2.9

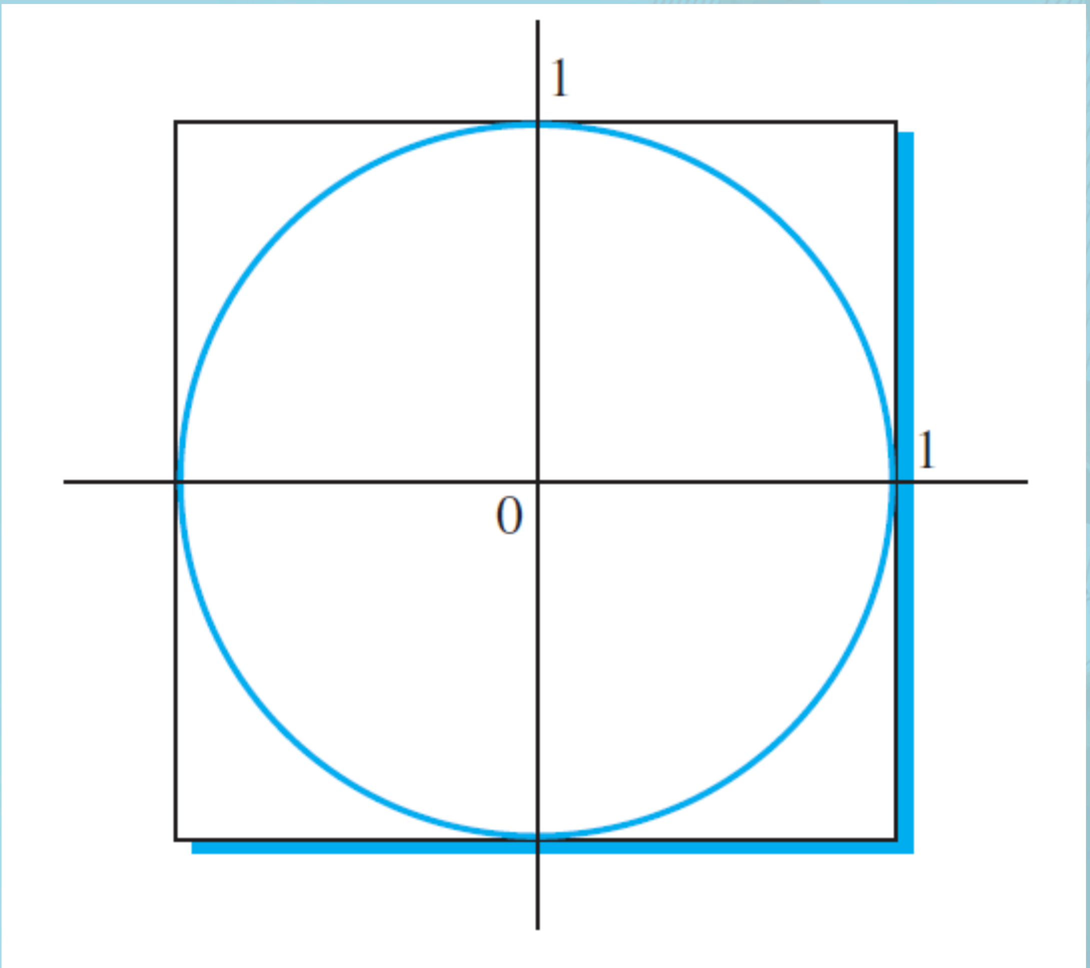
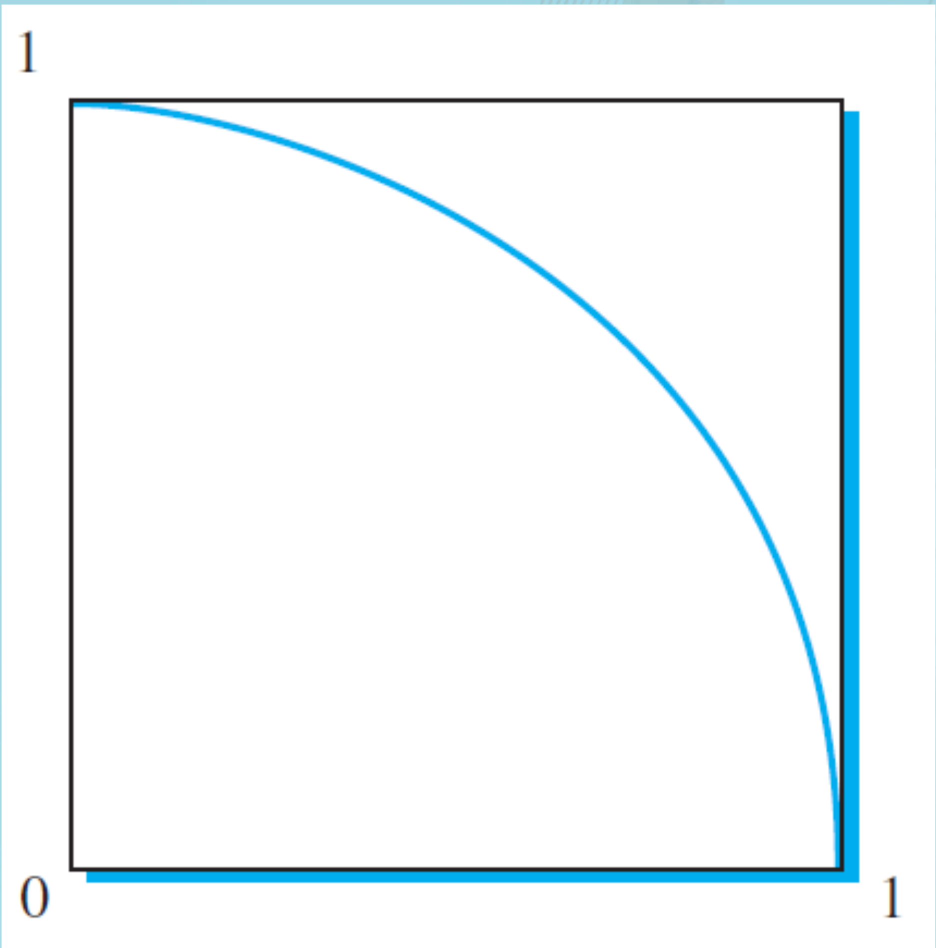


Figure 2.10



Selection Statements

- Ask a question (Boolean Expression)
- Based on the answer, perform a task

Figure 2.11

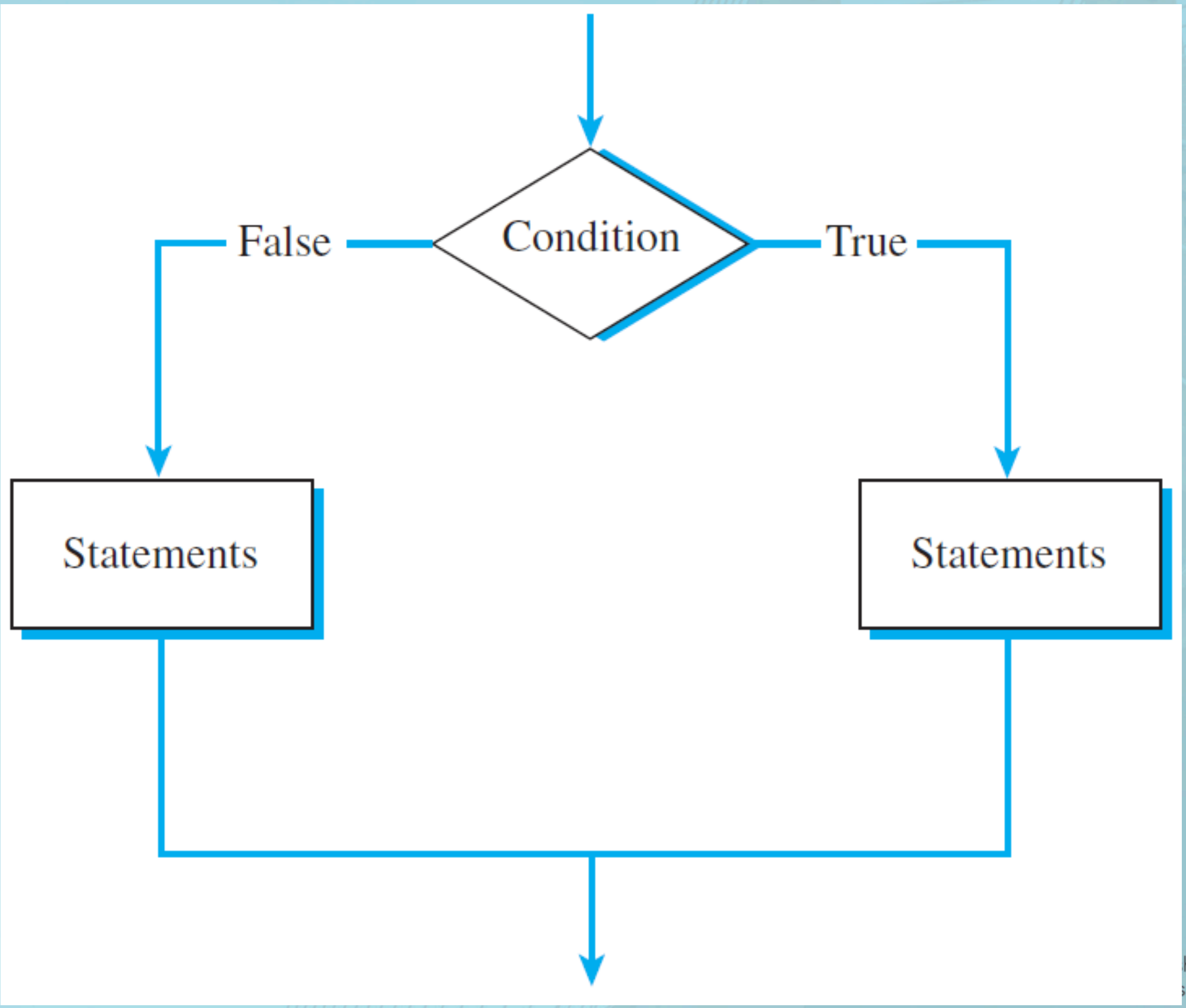
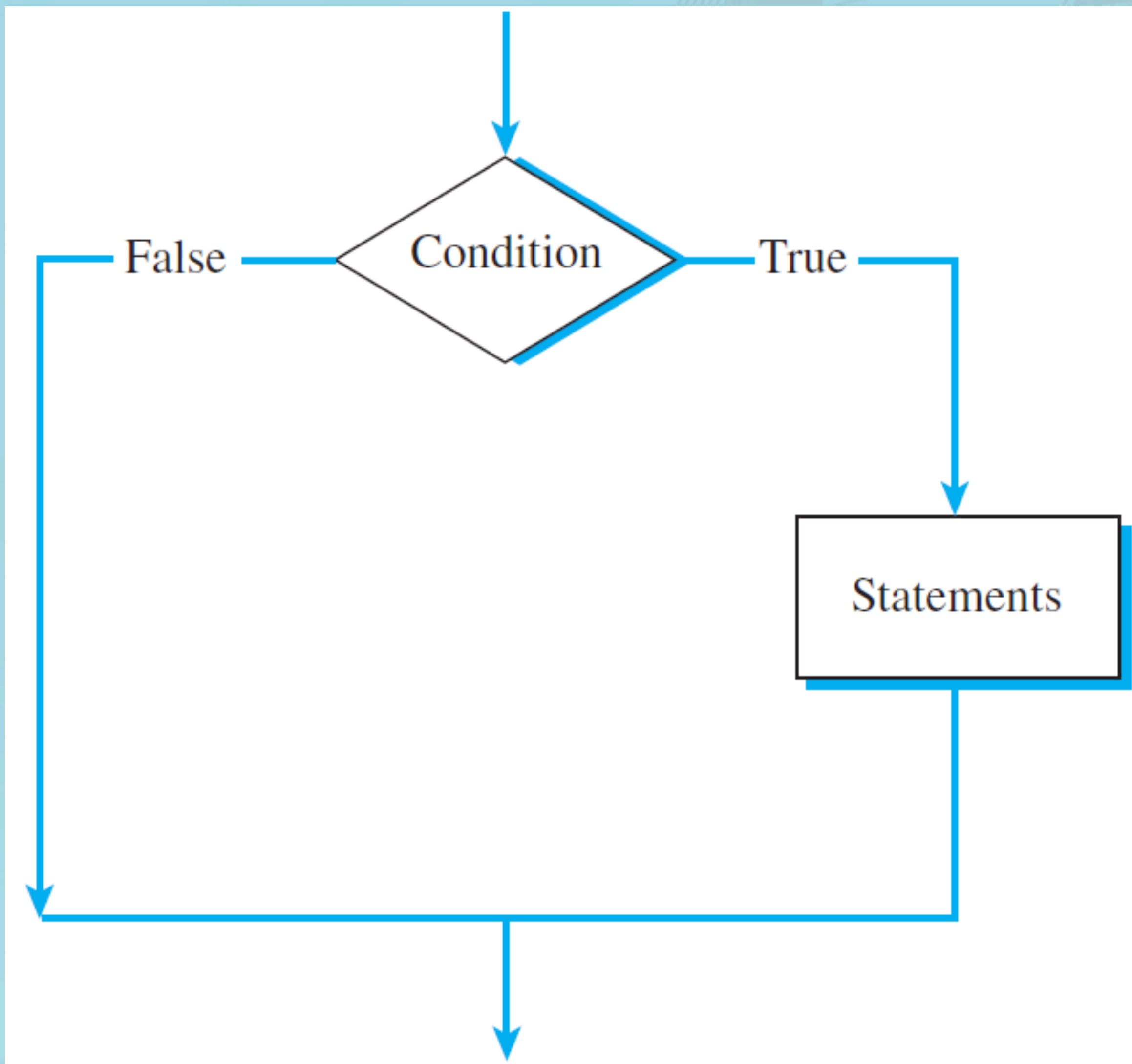


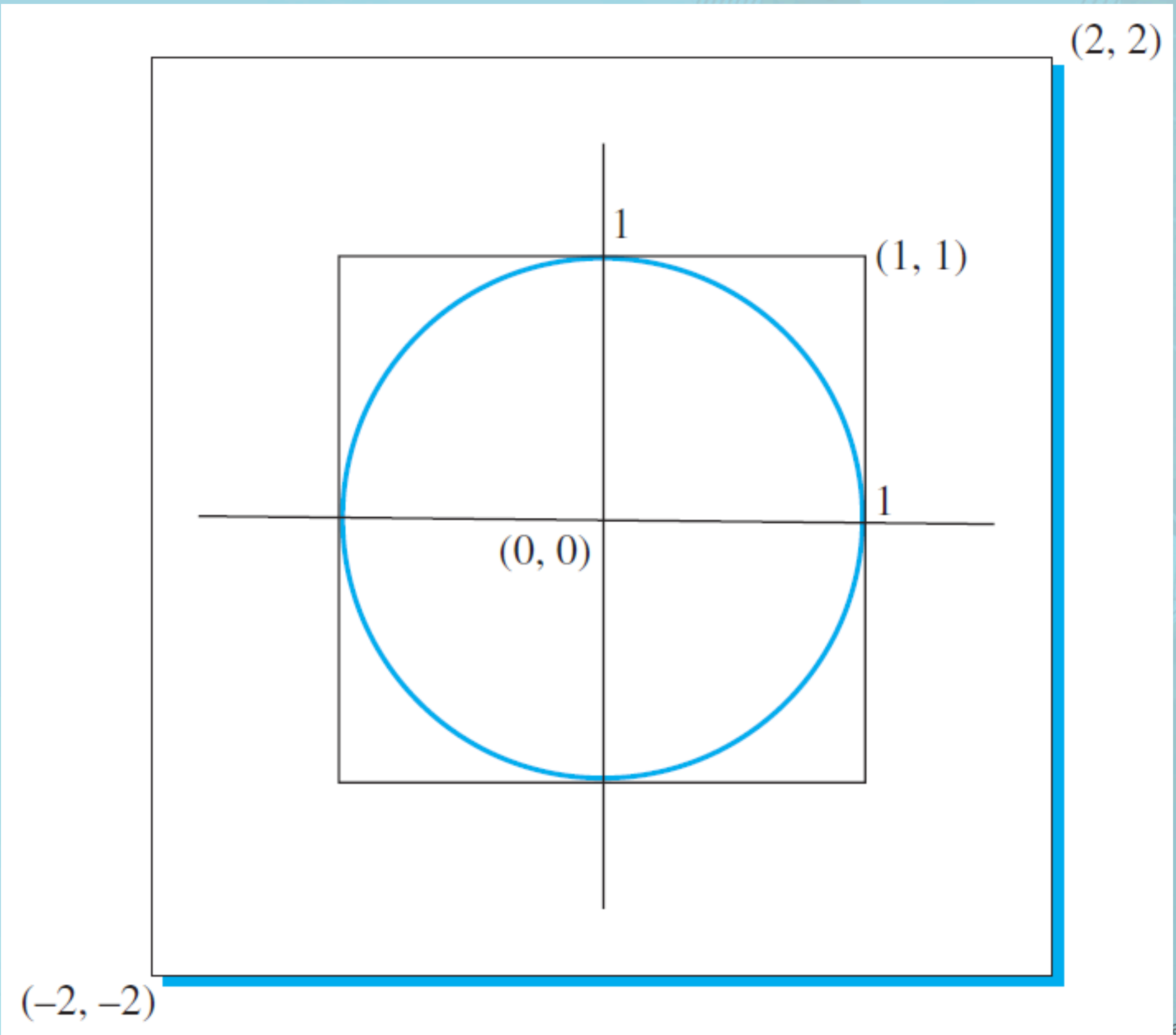
Figure 2.12



Listing 2.5

```
import random
import math

def montePi(numDarts):

   inCircle = 0

   for i in range(numDarts):
      x = random.random()
      y = random.random()

      d = math.sqrt(x**2 + y**2)

      if d <= 1:
        inCircle = inCircle + 1

   pi = inCircle/numDarts * 4

   return pi
```

Figure 2.13



Listing 2.6

```
import random
import math
import turtle

def showMontePi(numDarts):
    wn = turtle.Screen()
    drawingT = turtle.Turtle()

    wn.setworldcoordinates(-2,-2,2,2)

    drawingT.up()
    drawingT.goto(-1,0)
    drawingT.down()
    drawingT.goto(1,0)

    drawingT.up()
    drawingT.goto(0,1)
    drawingT.down()
    drawingT.goto(0,-1)

    circle = 0
    drawingT.up()
```

Listing 2.6 continued

```
for i in range(numDarts):
    x = random.random()
    y = random.random()

    d = math.sqrt(x**2 + y**2)

    drawingT.goto(x,y)

    if d <= 1:
        circle = circle + 1
        drawingT.color("blue")
    else:
        drawingT.color("red")

    drawingT.dot()

pi = circle/numDarts * 4

wn.exitonclick()

return pi
```

Figure 2.14