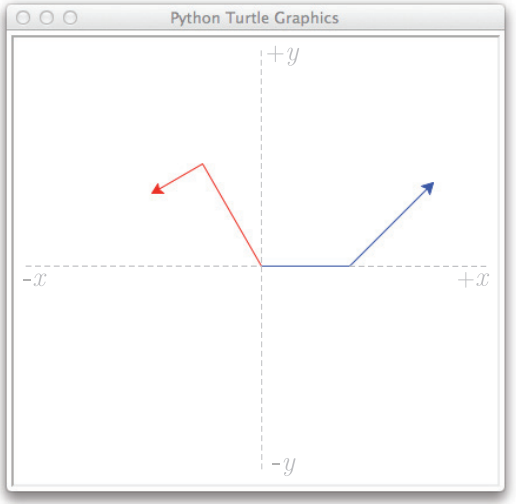

Figure 3.1 A turtle graphics window containing two turtles. The blue turtle moved forward, turned left 45°, and then moved forward again. The red turtle turned left 120°, moved forward, turned left again 90°, and then moved forward again.

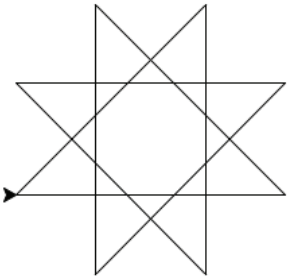

Figure 3.2 A simple geometric "flower" drawn with turtle graphics.

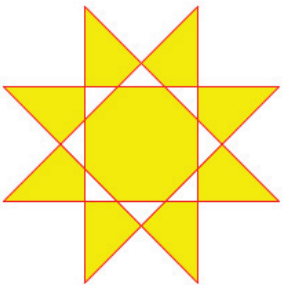

Figure 3.3 A simple geometric "flower," outlined in red and filled in yellow.

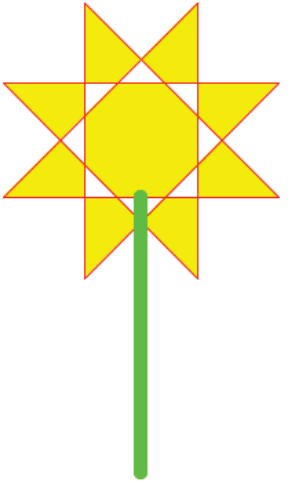

Figure 3.4 A simple geometric "flower" with a stem.

```
import turtle

def bloom(tortoise, fcolor, length):
    tortoise.pencolor('red')
    tortoise.fillcolor(fcolor)
    tortoise.begin_fill()
    for segment in range(8):
        tortoise.forward(length)
        tortoise.left(135)
    tortoise.end_fill()

def stem(tortoise, length):
    tortoise.pencolor('green')
    tortoise.pensize(length / 20)
    tortoise.up()
    tortoise.forward(length / 2)
    tortoise.down()
    tortoise.right(90)
    tortoise.forward(length)

def flower(tortoise, fcolor, length):
    bloom(tortoise, fcolor, length)
    stem(tortoise, length)

george = turtle.Turtle()
george.hideturtle()
george.speed(6)
flower(george, 'yellow', 200)
screen = george.getscreen()
screen.exitonclick()
```

Figure 3.5 The flower program before the flower planting code.

```
import turtle
import random

def bloom(tortoise, fcolor, length):
    tortoise.pencolor('red')
    tortoise.fillcolor(fcolor)
    tortoise.begin_fill()
    for segment in range(8):
        tortoise.forward(length)
        tortoise.left(135)
    tortoise.end_fill()

def stem(tortoise, length):
    tortoise.pencolor('green')
    tortoise.pensize(length / 20)
    tortoise.up()
    tortoise.forward(length / 2)
    tortoise.down()
    tortoise.right(90)
    tortoise.forward(length)

def flower(tortoise, fcolor, length):
    bloom(tortoise, fcolor, length)
    stem(tortoise, length)

def growFlower(x, y):
    span = random.randrange(20, 200)
    fill = random.choice(['yellow',
           'pink', 'red', 'purple'])
    tortoise = turtle.Turtle()
    tortoise.hideturtle()
    tortoise.speed(6)
    tortoise.up()
    tortoise.goto(x, y)
    tortoise.setheading(0)
    tortoise.pensize(1)
    tortoise.down()
    flower(tortoise, fill, span)

george = turtle.Turtle()
george.hideturtle()
screen = george.getscreen()
screen.onclick(growFlower)
screen.mainloop()
```

Figure 3.6 The final flower program with the flower planting code.

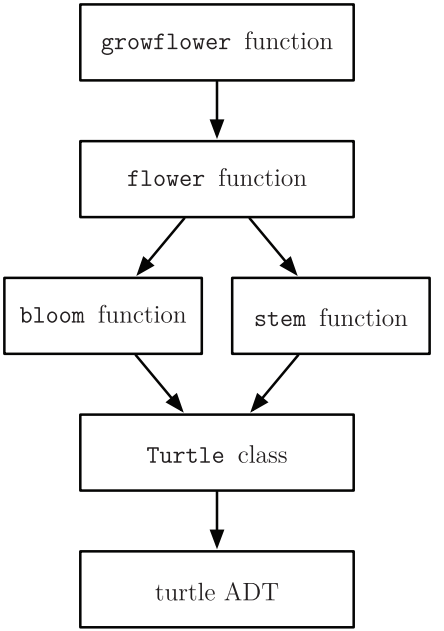

Figure 3.7 Layers of abstraction in the flower-drawing program.

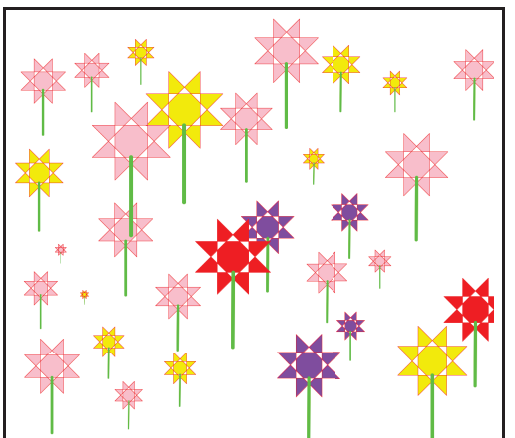

Figure 3.8 A “garden” of random flowers.

003x008.eps

program docstring

```
"""
Purpose: Draw a flower
Author: Ima Student
Date: September 15, 2020
CS 111, Fall 2020
"""
```

import statements

```
import turtle
```

function definitions

```
def bloom(tortoise, fcolor, length):
    """Draws a geometric flower bloom.

    Parameters:
        tortoise: a Turtle object with which to draw the bloom.
        fcolor: a color string to use to fill the bloom.
        length: the length of each segment of the bloom.

    Return value:
        None
    """

    tortoise.pencolor('red')     # set tortoise's pen color to red
    tortoise.fillcolor(fcolor)   # and fill color to fcolor
    tortoise.begin_fill()
    for segment in range(8):     # draw a filled 8-sided
        tortoise.forward(length) #   geometric flower bloom
        tortoise.left(135)
    tortoise.end_fill()

# other functions omitted ...
```

main function

```
def main():
    """Draws a yellow flower with segment length 200, and
       waits for a mouse click to exit.
    """

    george = turtle.Turtle()
    george.hideturtle()
    george.speed(6)
    flower(george, 'yellow', 200)
    screen = george.getscreen()
    screen.exitonclick()
```

main function call

```
main()
```

Figure 3.9 An overview of a program's structure.

```
def windChill(temp, wind):
    """ (docstring omitted) """

    print('Local namespace at the start of windChill:', locals())
    chill = 13.12 + 0.6215*temp + (0.3965*temp - 11.37) * wind**0.16
    temp = round(chill)
    print('Local namespace at the end of windChill:', locals())
    return temp

def main():
    t = -3
    w = 13
    print('Namespace of main before calling windChill:', locals())
    chilly = windChill(t, w)
    print('The wind chill is', chilly)
    print('Local namespace in main after calling windChill:', locals())

main()
```

Figure 3.10 The complete wind chill program, with calls to the locals function.