complex behavior that is captured in method algorithms (e.g., a projectile), or they may be little more than a collection of relevant information about some individual (e.g., a student record).

- Correctly designed classes provide encapsulation. The internal details of an object are hidden inside the class definition so that other portions of the program do not need to know how an object is implemented. This separation of concerns is a programming convention in Python; the instance variables of an object should only be accessed or modified through the interface methods of the class.

- Most GUI systems are built using an object-oriented approach. We can build novel GUI widgets by defining suitable classes. GUI widgets can be used to construct custom dialogs for user interaction.

10.9 Exercises

Review Questions

True/False

1. New objects are created by invoking a constructor.

2. Functions that live in objects are called instance variables.

3. The first parameter of a Python method definition is called `this`.

4. An object may have only one instance variable.

5. In data processing, a collection of information about a person or thing is called a file.

6. In a Python class, the constructor is called `__init__`.

7. A docstring is the same thing as a comment.

8. Instance variables go away once a method terminates.

9. Method names should always begin with one or two underscores.

10. It is considered bad style to directly access an instance variable outside of a class definition.

Multiple Choice

1. What Python reserved word starts a class definition?
   a) `def` b) `class` c) `object` d) `__init__`

2. A method definition with four formal parameters is generally called with how many actual parameters?
   a) three b) four c) five d) it depends

3. A method definition is similar to a(n)
   a) loop b) module c) import statement d) function definition

4. Within a method definition, the instance variable `x` could be accessed via which expression?
   a) `x` b) `self.x` c) `self[x]` d) `self.getX()`

5. A Python convention for defining methods that are "private" to a class is to begin the method name with
   a) "private" b) a pound sign (#)
   c) an underscore (_) d) a hyphen (-)

6. The term applied to hiding details inside class definitions is
   a) obscuring b) subclassing
   c) documentation d) encapsulation

7. A Python string literal can span multiple lines if enclosed with
   a) " b) ' c) """ d) \

8. In a `Button` widget, what is the data type of the instance variable `active`?
   a) bool b) int c) float d) `str`

9. Which of the following methods is *not* part of the `Button` class in this chapter?
   a) `activate` b) `deactivate` c) `setLabel` d) `clicked`

10. Which of the following methods is part of the `DieView` class in this chapter?
    a) `activate` b) `setColor` c) `setValue` d) `clicked`

Discussion

1. Explain the similarities and differences between instance variables and "regular" function variables.

2. Explain the following in terms of actual code that might be found in a class definition:

   a) method

   b) instance variable

   c) constructor

   d) accessor

   e) mutator

3. Show the output that would result from the following nonsense program:

```
class Bozo:

    def __init__(self, value):
        print("Creating a Bozo from:", value)
        self.value = 2 * value

    def clown(self, x):
        print("Clowning:", x)
        print(x * self.value)
        return x + self.value

def main():
    print("Clowning around now.")
    c1 = Bozo(3)
    c2 = Bozo(4)
    print c1.clown(3)
    print c2.clown(c1.clown(2))

main()
```

Programming Exercises

1. Modify the cannonball simulation from the chapter so that it also calculates the maximum height achieved by the cannonball.

2. Use the `Button` class discussed in this chapter to build a GUI for one (or more) of your projects from previous chapters.

3. Write a program to play "Three Button Monte." Your program should draw three buttons labeled "Door 1," "Door 2," and "Door 3" in a window and randomly select one of the buttons (without telling the user which one is selected). The program then prompts the user to click on one of the buttons. A click on the special button is a win, and a click on one of the other two is a loss. You should tell the user whether they won or lost, and in the case of a loss, which was the correct button. Your program should be entirely graphical; that is, all prompts and messages should be displayed in the graphics window.

4. Extend the program from the previous problem by allowing the player to play multiple rounds and displaying the number of wins and losses. Add a "Quit" button for ending the game.

5. Modify the `Student` class from the chapter by adding a mutator method that records a grade for the student. Here is the specification of the new method:

   `addGrade(self, gradePoint, credits)` `gradePoint` is a float that represents a grade (e.g., A = 4.0, A– = 3.7, B+ = 3.3, etc.), and `credits` is a float indicating the number of credit hours for the class. Modify the student object by adding this grade information.

   Use the updated class to implement a simple program for calculating GPA. Your program should create a new student object that has 0 credits and 0 quality points (the name is irrelevant). Your program should then prompt the user to enter course information (`gradePoint` and `credits`) for a series of courses, and then print out the final GPA achieved.

6. Extend the previous exercise by implementing an `addLetterGrade` method. This is similar to `addGrade` except that it accepts a letter grade as a string (instead of `gradePoint`). Use the updated class to improve the GPA calculator by allowing the entry of letter grades.

7. Write a modified `Button` class that creates circular buttons. Call your class `CButton` and implement the exact same methods that are in the existing `Button` class. Your constructor should take the center of the button and its radius as normal parameters. Place your class in a module called `cbutton.py`. Test your class by modifying `roller.py` to use your buttons.

8. Modify the `DieView` class from the chapter by adding a method that allows the color of the pips to be specified.

`setColor(self, color)` Changes the color of the pips to `color`.

Hints: You can change the color by changing the value of the instance variable `foreground`, but you also need to redraw the die after doing this. Modify `setValue` so that it remembers the value of the die in an instance variable. Then `setColor` can call `setValue` and pass the stored value to redraw the die. You can test your new class with the `roller.py` program. Have the dice change to a random color after each roll (you can generate a random color with the `color_rgb` function).

9. Write a class to represent spheres. Your class should implement the following methods:

   `__init__(self, radius)` Creates a sphere having the given `radius`.

   `getRadius(self)` Returns the radius of this sphere.

   `surfaceArea(self)` Returns the surface area of the sphere.

   `volume(self)` Returns the volume of the sphere.

   Use your new class to solve Programming Exercise 1 from Chapter 3.

10. Same as the previous problem, but for a cube. The constructor should accept the length of a side as a parameter.

11. Implement a class to represent a playing card. Your class should have the following methods:

    `__init__(self, rank, suit)` `rank` is an int in the range 1–13 indicating the ranks ace–king, and `suit` is a single character "d," "c," "h," or "s" indicating the suit (diamonds, clubs, hearts, or spades). Create the corresponding card.

    `getRank(self)` Returns the rank of the card.

    `getSuit(self)` Returns the suit of the card.

    `value(self)` Returns the Blackjack value of a card. Ace counts as 1, face cards count as 10.

    `__str__(self)` Returns a string that names the card. For example, `"Ace of Spades"`.

    Note: A method named `__str__` is special in Python. If asked to convert an object into a string, Python uses this method, if it's present. For example,

```
c = Card(1,"s")
print c
```

will print "Ace of Spades."

Test your card class with a program that prints out n randomly generated cards and the associated Blackjack value where n is a number supplied by the user.

12. Extend your card class from the previous problem with a `draw(self, win, center)` method that displays the card in a graphics window. Use your extended class to create and display a hand of five random cards. *Hint*: The easiest way to do this is to search the Internet for a free set of card images and use the `Image` object in the graphics library to display them.

13. Here is a simple class that draws a (grim) face in a graphics window:

```
# face.py
from graphics import *

class Face:

    def __init__(self, window, center, size):
        eyeSize = 0.15 * size
        eyeOff = size / 3.0
        mouthSize = 0.8 * size
        mouthOff = size / 2.0
        self.head = Circle(center, size)
        self.head.draw(window)
        self.leftEye = Circle(center, eyeSize)
        self.leftEye.move(-eyeOff, -eyeOff)
        self.rightEye = Circle(center, eyeSize)
        self.rightEye.move(eyeOff, -eyeOff)
        self.leftEye.draw(window)
        self.rightEye.draw(window)
        p1 = center.clone()
        p1.move(-mouthSize/2, mouthOff)
        p2 = center.clone()
        p2.move(mouthSize/2, mouthOff)
```

```
self.mouth = Line(p1,p2)
self.mouth.draw(window)
```

Add methods to this class that cause the face to change expression. For example you might add methods such as `smile`, `wink`, `frown`, `flinch`, etc. Your class should implement at least three such methods.

Use your class to write a program that draws a face and provides the user with buttons to change the facial expression.

14. Modify the `Face` class from the previous problem to include a `move` method similar to other graphics objects. Using the `move` method, create a program that makes a face bounce around in a window (see Programming Exercise 17 from Chapter 7). Bonus: have the face change expression each time it "hits" the edge of the window.

15. Modify the cannonball animation so that the input dialog window stays on screen at all times.

16. Advanced: Add a `Target` class to the cannonball animation. A target should be a rectangle placed at a random `x` position at the bottom of the window. Allow users to keep firing until they hit the target.

17. Redo the regression problem from Chapter 8 (Programming Exercise 13) using a `Regression` class. Your new class will keep track of the various quantities that are needed to compute a line of regression (the running sums of x, y, x^2, and xy). The `Regression` class should have the following methods:

__init__ Creates a new regression object to which points can be added.

addPoint Adds a point to the regression object.

predict Accepts a value of x as a parameter, and returns the value of the corresponding y on the line of best fit.

Note: Your class might also use some internal helper methods to do such things as compute the slope of the regression line.