better solution for some special case. Don't be afraid to step back and think about the overarching problem. Similarly, when designing programs, you should always have an eye toward making your program more generally useful. If the max of n program is just as easy to write as max of three, you may as well write the more general program because it is more likely to be useful in other situations. That way you get the maximum utility from your programming effort.

- Don't reinvent the wheel. Our fourth solution was to use Python's `max` function. You may think that was cheating, but this example illustrates an important point. A lot of very smart programmers have designed countless good algorithms and programs. If the problem you are trying to solve seems to be one that lots of others must have encountered, you might begin by finding out if the problem has already been solved for you. As you are learning to program, designing from scratch is great experience. Truly expert programmers, however, know when to borrow.

7.6 Chapter Summary

This chapter has laid out the basic control structures for making decisions. Here are the key points.

- Decision structures are control structures that allow a program to execute different sequences of instructions for different cases.

- Decisions are implemented in Python with `if` statements. Simple decisions are implemented with a plain `if`. Two-way decisions generally use an `if-else`. Multi-way decisions are implemented with `if-elif-else`.

- Decisions are based on the evaluation of conditions, which are simple Boolean expressions. A Boolean expression is either true or false. Python has a dedicated `bool` data type with literals `True` and `False`. Conditions are formed using the relational operators: `<`, `<=`, `!=`, `==`, `>`, and `>=`.

- Some programming languages provide exception handling mechanisms which help to make programs more "bulletproof." Python provides a `try-except` statement for exception handling.

- Algorithms that incorporate decisions can become quite complicated as decision structures are nested. Usually a number of solutions are possible,

and careful thought should be given to produce a correct, efficient, and understandable program.

7.7 Exercises

Review Questions

True/False

1. A simple decision can be implemented with an `if` statement.
2. In Python conditions, $\neq$ is written as `/=`.
3. Strings are compared by lexicographic ordering.
4. A two-way decision is implemented using an `if-elif` statement.
5. The `math.sqrt` function cannot compute the square root of a negative number.
6. A single `try` statement can catch multiple kinds of errors.
7. Multi-way decisions must be handled by nesting multiple `if-else` statements.
8. There is usually only one correct solution to a problem involving decision structures.
9. The condition `x <= y <= z` is allowed in Python.
10. Input validation means prompting a user when input is required.

Multiple Choice

1. A statement that controls the execution of other statements is called a
   a) boss structure b) super structure
   c) control structure d) branch

2. The best structure for implementing a multi-way decision in Python is
   a) `if` b) `if-else` c) `if-elif-else` d) `try`

3. An expression that evaluates to either true or false is called
   a) operational b) Boolean c) simple d) compound

4. When a program is being run directly (not imported), the value of `__name__` is
   a) `script` b) `main` c) `__main__` d) `True`

5. The literals for type `bool` are
   a) `T`, `F` b) `True`, `False` c) `true`, `false` d) `1`, `0`

6. Placing a decision inside of another decision is an example of
   a) cloning b) spooning c) nesting d) procrastination

7. In Python, the body of a decision is indicated by
   a) indentation b) parentheses c) curly braces d) a colon

8. A structure in which one decision leads to another set of decisions, which leads to another set of decisions, etc., is called a decision
   a) network b) web c) tree d) trap

9. Taking the square root of a negative value with `math.sqrt` produces a(n)
   a) ValueError b) imaginary number
   c) program crash d) stomachache

10. A multiple choice question is most similar to
    a) simple decision b) two-way decision
    c) multi-way decisions d) an exception handler

Discussion

1. Explain the following patterns in your own words:

   a) simple decision

   b) two-way decision

   c) multi-way decision

2. How is exception handling using `try/except` similar to and different from handling exceptional cases using ordinary decision structures (variations on `if`)?

3. The following is a (silly) decision structure:

```
a, b, c = eval(input('Enter three numbers: '))
```

```
if a > b:
    if b > c:
        print("Spam Please!")
    else:
        print("It's a late parrot!")
elif b > c:
    print("Cheese Shoppe")
    if a >= c:
        print("Cheddar")
    elif a < b:
        print("Gouda")
    elif c == b:
        print("Swiss")
else:
    print("Trees")
    if a == b:
        print("Chestnut")
    else:
        print("Larch")
print("Done")
```

Show the output that would result from each of the following possible inputs:

a) 3, 4, 5

b) 3, 3, 3

c) 5, 4, 3

d) 3, 5, 2

e) 5, 4, 7

f) 3, 3, 2

Programming Exercises

1. Many companies pay time-and-a-half for any hours worked above 40 in a given week. Write a program to input the number of hours worked and the hourly rate and calculate the total wages for the week.

2. A certain CS professor gives five-point quizzes that are graded on the scale 5-A, 4-B, 3-C, 2-D, 1-F, 0-F. Write a program that accepts a quiz score as an input and uses a decision structure to calculate the corresponding grade.

3. A certain CS professor gives 100-point exams that are graded on the scale 90–100:A, 80–89:B, 70–79:C, 60–69:D, <60:F. Write a program that accepts an exam score as input and uses a decision structure to calculate the corresponding grade.

4. A certain college classifies students according to credits earned. A student with less than 7 credits is a Freshman. At least 7 credits are required to be a Sophomore, 16 to be a Junior and 26 to be classified as a Senior. Write a program that calculates class standing from the number of credits earned.

5. The body mass index (BMI) is calculated as a person's weight (in pounds) times 720, divided by the square of the person's height (in inches). A BMI in the range 19–25, inclusive, is considered healthy. Write a program that calculates a person's BMI and prints a message telling whether they are above, within, or below the healthy range.

6. The speeding ticket fine policy in Podunksville is $50 plus $5 for each mph over the limit plus a penalty of $200 for any speed over 90 mph. Write a program that accepts a speed limit and a clocked speed and either prints a message indicating the speed was legal or prints the amount of the fine, if the speed is illegal.

7. A babysitter charges $2.50 an hour until 9:00 PM when the rate drops to $1.75 an hour (the children are in bed). Write a program that accepts a starting time and ending time in hours and minutes and calculates the total babysitting bill. You may assume that the starting and ending times are in a single 24-hour period. Partial hours should be appropriately prorated.

8. A person is eligible to be a US senator if they are at least 30 years old and have been a US citizen for at least 9 years. To be a US representative these numbers are 25 and 7, respectively. Write a program that accepts a person's age and years of citizenship as input and outputs their eligibility for the Senate and House.

9. A formula for computing Easter in the years 1982–2048, inclusive, is as follows: let $a = year\%19$, $b = year\%4$, $c = year\%7$, $d = (19a + 24)\%30$, $e = (2b + 4c + 6d + 5)\%7$. The date of Easter is March $22 + d + e$ (which

could be in April). Write a program that inputs a year, verifies that it is in the proper range, and then prints out the date of Easter that year.

10. The formula for Easter in the previous problem works for every year in the range 1900–2099 except for 1954, 1981, 2049, and 2076. For these 4 years it produces a date that is one week too late. Modify the above program to work for the entire range 1900–2099.

11. A year is a leap year if it is divisible by 4, unless it is a century year that is not divisible by 400. (1800 and 1900 are *not* leap years while 1600 and 2000 *are*.) Write a program that calculates whether a year is a leap year.

12. Write a program that accepts a date in the form month/day/year and outputs whether or not the date is valid. For example 5/24/1962 is valid, but 9/31/2000 is not. (September has only 30 days.)

13. The days of the year are often numbered from 1 through 365 (or 366). This number can be computed in three steps using int arithmetic:

    (a) $\texttt{dayNum} = 31(\texttt{month} - 1) + \texttt{day}$

    (b) if the month is after February subtract $(4(\texttt{month}) + 23)//10$

    (c) if it's a leap year and after February 29, add 1

    Write a program that accepts a date as month/day/year, verifies that it is a valid date (see previous problem), and then calculates the corresponding day number.

14. Do Programming Exercise 7 from Chapter 4, but add a decision to handle the case where the line does not intersect the circle.

15. Do Programming Exercise 8 from Chapter 4, but add a decision to prevent the program from dividing by zero if the line is vertical.

16. Archery Scorer. Write a program that draws an archery target (see Programming Exercise 2 from Chapter 4) and allows the user to click five times to represent arrows shot at the target. Using five-band scoring, a bulls-eye (yellow) is worth 9 points and each successive ring is worth 2 fewer points down to 1 for white. The program should output a score for each click and keep track of a running sum for the entire series.

17. Write a program to animate a circle bouncing around a window. The basic idea is to start the circle somewhere in the interior of the window. Use variables `dx` and `dy` (both initialized to 1) to control the movement of the circle. Use a large counted loop (say 10000 iterations), and each time through the loop move the circle using `dx` and `dy`. When the x-value of the center of the circle gets too high (it hits the edge), change `dx` to -1. When it gets too low, change `dx` back to 1. Use a similar approach for `dy`.

    Note: Your animation will probably run too fast. You can slow it down by using `update` from the graphics library with a rate parameters. For example, this loop will be limited to going around at a rate of 30 times per second:

    ```
    for i in range(10000):
        ...
        update(30) # pause so rate is not more than 30 times a second
    ```

18. Take a favorite programming problem from a previous chapter and add decisions and/or exception handling as required to make it truly robust (will not crash on any inputs). Trade your program with a friend and have a contest to see who can "break" the other's program.