2.2 *(What's wrong with this code?)* The following code should read an integer into the variable `rating`:

```
rating = input('Enter an integer rating between 1 and 10')
```

2.3 *(Fill in the missing code)* Replace `***` in the following code with a statement that will print a message like `'Congratulations! Your grade of 91 earns you an A in this course'`. Your statement should print the value stored in the variable `grade`:

```
if grade >= 90:
    ***
```

2.4 *(Arithmetic)* For each of the arithmetic operators `+`, `-`, `*`, `/`, `//` and `**`, display the value of an expression with `27.5` as the left operand and `2` as the right operand.

2.5 *(Circle Area, Diameter and Circumference)* For a circle of radius 2, display the diameter, circumference and area. Use the value 3.14159 for π. Use the following formulas (r is the radius): *diameter* = $2r$, *circumference* = $2\pi r$ and area = πr^2. [In a later chapter, we'll introduce Python's `math` module which contains a higher-precision representation of π.]

2.6 *(Odd or Even)* Use `if` statements to determine whether an integer is odd or even. [*Hint:* Use the remainder operator. An even number is a multiple of 2. Any multiple of 2 leaves a remainder of 0 when divided by 2.]

2.7 *(Multiples)* Use `if` statements to determine whether 1024 is a multiple of 4 and whether 2 is a multiple of 10. (*Hint:* Use the remainder operator.)

2.8 *(Table of Squares and Cubes)* Write a script that calculates the squares and cubes of the numbers from 0 to 5. Print the resulting values in table format, as shown below. Use the tab escape sequence to achieve the three-column output.

```
number  square  cube
0       0       0
1       1       1
2       4       8
3       9       27
4       16      64
5       25      125
```

The next chapter shows how to "right align" numbers. You could try that as an extra challenge here. The output would be:

```
number  square  cube
     0       0     0
     1       1     1
     2       4     8
     3       9    27
     4      16    64
     5      25   125
```

2.9 *(Integer Value of a Character)* Here's a peek ahead. In this chapter, you learned about strings. Each of a string's characters has an integer representation. The set of characters a computer uses together with the characters' integer representations is called that computer's *character set*. You can indicate a character value in a program by enclosing that character in quotes, as in `'A'`. To determine a character's integer value, call the built-in function **`ord`**:

```
In [1]: ord('A')
Out[1]: 65
```

Display the integer equivalents of `B C D b c d 0 1 2 $ * +` and the space character.

2.10 *(Arithmetic, Smallest and Largest)* Write a script that inputs three integers from the user. Display the sum, average, product, smallest and largest of the numbers. Note that each of these is a reduction in functional-style programming.

2.11 *(Separating the Digits in an Integer)* Write a script that inputs a five-digit integer from the user. Separate the number into its individual digits. Print them separated by three spaces each. For example, if the user types in the number `42339`, the script should print

```
4   2   3   3   9
```

Assume that the user enters the correct number of digits. Use both the floor division and remainder operations to "pick off" each digit.

2.12 *(7% Investment Return)* Some investment advisors say that it's reasonable to expect a 7% return over the long term in the stock market. Assuming that you begin with $1000 and leave your money invested, calculate and display how much money you'll have after 10, 20 and 30 years. Use the following formula for determining these amounts:

$$a = p(1 + r)^n$$

where

p is the original amount invested (i.e., the principal of $1000),
r is the annual rate of return (7%),
n is the number of years (10, 20 or 30) and
a is the amount on deposit at the end of the nth year.

2.13 *(How Big Can Python Integers Be?)* We'll answer this question later in the book. For now, use the exponentiation operator `**` with large and very large exponents to produce some huge integers and assign those to the variable `number` to see if Python accepts them. Did you find any integer value that Python won't accept?

2.14 *(Target Heart-Rate Calculator)* While exercising, you can use a heart-rate monitor to see that your heart rate stays within a safe range suggested by your doctors and trainers. According to the American Heart Association (AHA) (`http://bit.ly/AHATargetHeartRates`), the formula for calculating your maximum heart rate in beats per minute is 220 minus your age in years. Your target heart rate is 50–85% of your maximum heart rate. Write a script that prompts for and inputs the user's age and calculates and displays the user's maximum heart rate and the range of the user's target heart rate. [**These formulas are estimates provided by the AHA; maximum and target heart rates may vary based on the health, fitness and gender of the individual. Always consult a physician or qualified healthcare professional before beginning or modifying an exercise program.**]

2.15 *(Sort in Ascending Order)* Write a script that inputs three different floating-point numbers from the user. Display the numbers in increasing order. Recall that an `if` statement's suite can contain more than one statement. Prove that your script works by running it on all six possible orderings of the numbers. Does your script work with duplicate numbers? [This is challenging. In later chapters you'll do this more conveniently and with many more numbers.]

value. Use one counter to keep track of the number of passes, then calculate the number of failures after all the user's inputs have been received.

3.2 *(What's Wrong with This Code?)* What is wrong with the following code?

```
a = b = 7
print('a =', a, '\nb =', b)
```

First, answer the question, then check your work in an IPython session.

3.3 *(What Does This Code Do?)* What does the following program print?

```
for row in range(10):
    for column in range(10):
        print('<' if row % 2 == 1 else '>', end='')
    print()
```

3.4 *(Fill in the Missing Code)* In the code below

```
for ***:
    for ***:
        print('@')
    print()
```

replace the *** so that when you execute the code, it displays two rows, each containing seven @ symbols, as in:

```
@@@@@@@
@@@@@@@
```

3.5 *(`if...else` Statements)* Reimplement the script of Fig. 2.1 using three `if...else` statements rather than six `if` statements. [*Hint*: For example, think of == and != as "opposite" tests.]

3.6 *(Turing Test)* The great British mathematician Alan Turing proposed a simple test to determine whether machines could exhibit intelligent behavior. A user sits at a computer and does the same text chat with a human sitting at a computer and a computer operating by itself. The user doesn't know if the responses are coming back from the human or the independent computer. If the user can't distinguish which responses are coming from the human and which are coming from the computer, then it's reasonable to say that the computer is exhibiting intelligence.

Create a script that plays the part of the independent computer, giving its user a simple medical diagnosis. The script should prompt the user with `'What is your problem?'` When the user answers and presses *Enter*, the script should simply ignore the user's input, then prompt the user again with `'Have you had this problem before (yes or no)?'` If the user enters `'yes'`, print `'Well, you have it again.'` If the user answers `'no'`, print `'Well, you have it now.'`

Would this conversation convince the user that the entity at the other end exhibited intelligent behavior? Why or why not?

3.7 *(Table of Squares and Cubes)* In Exercise 2.8, you wrote a script to calculate the squares and cubes of the numbers from 0 through 5, then printed the resulting values in table format. Reimplement your script using a `for` loop and the f-string capabilities you learned in this chapter to produce the following table with the numbers right aligned in each column.

```
number  square  cube
     0       0     0
     1       1     1
     2       4     8
     3       9    27
     4      16    64
     5      25   125
```

3.8 *(Arithmetic, Smallest and Largest)* In Exercise 2.10, you wrote a script that input three integers, then displayed the sum, average, product, smallest and largest of those values. Reimplement your script with a loop that inputs four integers.

3.9 *(Separating the Digits in an Integer)* In Exercise 2.11, you wrote a script that separated a five-digit integer into its individual digits and displayed them. Reimplement your script to use a loop that in each iteration "picks off" one digit (left to right) using the // and % operators, then displays that digit.

3.10 *(7% Investment Return)* Reimplement Exercise 2.12 to use a loop that calculates and displays the amount of money you'll have each year at the ends of years 1 through 30.

3.11 *(Miles Per Gallon)* Drivers are concerned with the mileage obtained by their automobiles. One driver has kept track of several tankfuls of gasoline by recording miles driven and gallons used for each tankful. Develop a sentinel-controlled-repetition script that prompts the user to input the miles driven and gallons used for each tankful. The script should calculate and display the miles per gallon obtained for each tankful. After processing all input information, the script should calculate and display the combined miles per gallon obtained for all tankfuls (that is, total miles driven divided by total gallons used).

```
Enter the gallons used (-1 to end): 12.8
Enter the miles driven: 287
The miles/gallon for this tank was 22.421875
Enter the gallons used (-1 to end): 10.3
Enter the miles driven: 200
The miles/gallon for this tank was 19.417475
Enter the gallons used (-1 to end): 5
Enter the miles driven: 120
The miles/gallon for this tank was 24.000000
Enter the gallons used (-1 to end): -1
The overall average miles/gallon was 21.601423
```

3.12 *(Palindromes)* A palindrome is a number, word or text phrase that reads the same backwards or forwards. For example, each of the following five-digit integers is a palindrome: 12321, 55555, 45554 and 11611. Write a script that reads in a five-digit integer and determines whether it's a palindrome. [*Hint*: Use the // and % operators to separate the number into its digits.]

3.13 *(Factorials)* Factorial calculations are common in probability. The factorial of a nonnegative integer *n* is written *n*! (pronounced "*n* factorial") and is defined as follows:

$$n! = n \cdot (n - 1) \cdot (n - 2) \cdot \ldots \cdot 1$$

for values of *n* greater than or equal to 1, with 0! defined to be 1. So,

$$5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$$

which is 120. Factorials increase in size very rapidly. Write a script that inputs a nonnegative integer and computes and displays its factorial. Try your script on the integers 10, 20,