Starting Out with Python, 5th Edition

Answers to Review Questions

Chapter 1

Multiple Choice

1. b
2. a
3. d
4. b
5. c
6. a
7. c
8. b
9. a
10. a
11. d
12. b
13. c
14. b
15. c
16. a
17. b
18. d
19. b
20. b
21 c
22. a
23. d
24. a
25. b

True or False

1. False
2. True
3. True
4. False
5. True
6. False
7. True
8. False
9. False
10. False

Short Answer

1. Because without it, the computer could not run software.

2. A bit that is turned on represents 1, and a bit that is turned off represents 0.

3. A digital device

4. Keywords

5. Mnemonics

6. A compiler is a program that translates a high-level language program into a separate machine language program. The machine language program can then be executed any time it is needed. An interpreter is a program that both translates and executes the instructions in a high-level language program. As the interpreter reads each individual instruction in the program, it converts it to a machine language instruction and then immediately executes it. Because interpreters combine translation and execution, they typically do not create separate machine language programs.

7. Operating system

Exercises

1. *No solution -- This is a hands-on exercise to help you learn how to work with the Python interpreter in interactive mode.*

2. *No solution -- This is a hands-on exercise to help you learn how to work with the IDLE programming environment.*

3.

Decimal	Binary
11	1011
65	1000001
100	1100100
255	11111111

4.

Binary	Decimal
1101	13
1000	8
101011	43

5. Here is an example: The ASCII codes for the name Marty are:

```
M =  77
a =  97
r = 114
t = 226
y = 121
```

6.

- Guido van Rossum is the creator of the Python programming language.
- Python was created in the late 1980s.
- Benevolent Dictator for Life

Chapter 2

Multiple Choice

1. c
2. b
3. d
4. b
5. a
6. c
7. a
8. b
9. d
10. a
11. b
12. d
13. b
14. a
15. a
16. c
17. a
18. b
19. a
20. b
21. b
22. b

True or False

1. False
2. True
3. False
4. True
5. False

Short Answer

1. Interview the customer

2. An informal language that has no syntax rules, and is not meant to be compiled or executed. Instead, programmers use pseudocode to create models, or "mock-ups" of programs.

3. (1) Input is received.
   (2) Some process is performed on the input.
   (3) Output is produced.

4. `float`

5. Floating point division returns a floating point number that may include fractions. Integer division returns an integer and ignores any fractional part of the division result.

6. A magic number is an unexplained value that appears in a program's code. Magic numbers can be problematic, for a number of reasons. First, it can be difficult for someone reading the

code to determine the purpose of the number. Second, if the magic number is used in multiple places in the program, it can take painstaking effort to change the number in each location, should the need arise. Third, you take the risk of making a typographical mistake each time you type the magic number in the program's code.

7. The named constant makes the program more self-explanatory. In a math statement, it is evident that `PI` represents the value of pi. Another advantage to using the named constant is that widespread changes can easily be made to the program. Let's say the value of pi appears in several different statements throughout the program. If you need to change the number of decimal places of precision used with the number, the initialization value in the declaration of the named constant is the only value that needs to be modified. For example, to use only two decimal places of precision, the declaration can be changed to:

```
PI = 3.14
```

The new value of 3.14 will then be used in each statement that includes the `PI` constant. Another advantage to using the named constant is that it helps to prevent the typographical errors that are common when using magic numbers. For example, if you accidentally type 31.4159 instead of 3.14159 in a math statement, the program will calculate the wrong value. However, if you misspell `PI`, the Python interpreter will display a message indicating that the name is not defined.

Algorithm Workbench

1.
```
height = int(input('Enter your height: '))
```

2.
```
color = input('Enter your favorite color: ')
```

3.
a.
```
b = a + 2
```
b.
```
a = b * 4
```
c.
```
b = a / 3.14
```
d.
```
a = b - 8
```

4.
a. 12
b. 4
c. 2
d. 6
e. 2

5.
```
total = 10 + 14
```

6.
```
due = total - down_payment
```

7.
```
total = subtotal * 0.15
```

8. 11

9. 5

10.
```
print(f'{sales:.2f}')
```


11.
```
print(f'{number:,.1f}')
```

12.
```
George@John@Paul@Ringo
```

13.
```
turtle.circle(75)
```

14.
```
turtle.fillcolor('blue')
turtle.begin_fill()
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.end_fill()
```

15.
```
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.penup()
turtle.left(90)
turtle.forward(50)
turtle.right(90)
turtle.forward(30)
turtle.setheading(0)
turtle.pendown()
turtle.fillcolor('red')
turtle.begin_fill()
turtle.circle(80)
turtle.end_fill()
```

Chapter 3

Multiple Choice

1. c
2. b
3. d
4. a
5. c
6. b
7. c
8. b
9. a
10. b

11. c
12. a

True or False

1. False
2. False
3. False
4. True
5. True

Short Answer

1. A conditionally executed statement is performed only when a certain condition is true.

2. A dual alternative decision structure

3. The `and` operator connects two Boolean expressions into one compound expression. Both subexpressions must be true for the compound expression to be true.

4. The `or` operator connects two Boolean expressions into one compound expression. One or both sub expressions must be true for the compound expression to be true. It is only necessary for one of the subexpressions to be true, and it does not matter which.

5. The `and` operator.

6. A flag is a variable that signals when some condition exists in the program. When the flag variable is set to `False`, it indicates the condition does not exist. When the flag variable is set to `True`, it means the condition does exist.

Algorithm Workbench

1.
```
if x > 100:
    y = 20
    z = 40
```

2.
```
if a < 10:
    b = 0
    c = 1
```

3.
```
if a < 10:
    b = 0 else:
    b = 99
```

4.
```
if score >= A_score:
    print('Your grade is A.')
else:
    if score >= B_score:
        print('Your grade is B.')
    else:
        if score >= C_score:
            print('Your grade is C.')
```



```
            else:
                if score >= D_score:
                    print('Your grade is D.')
                else:
                    print('Your grade is F.')
```

5.
```
if amount1 > 10 and amount2 < 100:
    if amount1 > amount2:
        print (amount1)
    elif amount2 > amount1:
        print (amount2)
    else:
        print('Both values are the same.')
```

6.
```
if speed >= 24 and speed <= 56:
    print ('Speed is normal.')
else:
    print ('Speed is abnormal.')
```

7.
```
if points < 9 or points > 51:
    print ('Invalid points')
else:
   print ('Valid points')
```

8.
```
if turtle.heading() >= 0 and turtle.heading() <= 45:
    turtle.penup()
```

9.
```
if turtle.pencolor() == 'red' or turtle.pencolor() == 'blue':
    turtle.pensize(5)
```

10.
```
if turtle.xcor() > 100 and turtle.xcor() < 200 and
   turtle.ycor() > 100 and turtle.ycor() < 200:
    turtle.hideturtle()
```

Chapter 4

Multiple Choice

1. b
2. d
3. d
4. a
5. c
6. b
7. d
8. a
9. b
10. c
11. d
12. a

True or False

1. False
2. True
3. True
4. False
5. True
6. False
7. False

Short Answer

1. A condition-controlled loop uses a true/false condition to control the number of times that it repeats.

2. A count-controlled loop repeats a specific number of times.

3. An *infinite loop* continues to repeat until the program is interrupted. Infinite loops usually occur when the programmer forgets to write code inside the loop that makes the test condition false. Here is an example of Python code that contains an infinite loop:

```
x = 99
while x > 0:
    print (x)
```

4. If the accumulator starts with any value other than 0, it will not contain the correct total when the loop finishes.

5. You can write a loop that processes a list of data items even though you do not know the number of data items in the list, and without requiring the user to know the number of items in the list in advance.

6. A sentinel value must be unique enough that it will not be mistaken as a regular value in the list.

7. This saying, sometimes abbreviated as GIGO, refers to the fact that computers cannot tell the difference between good data and bad data. If a user provides bad data as input to a program, the program will process that bad data and, as a result, will produce bad data as output.

8. When input is given to a program, it should be inspected before it is processed. If the input is invalid, the program should discard it and prompt the user to enter the correct data.

   Specifically, the input is read, and then a loop is executed. If the input data is bad, the loop executes its block of statements. The loop displays an error message so the user will know that the input was invalid, and then it reads the new input. The loop repeats as long as the input is bad.

Algorithm Workbench

1.
```
product = 0
while product < 100:
    number = int(input('Enter a number: '))
```



```
    product = number * 10
```

2.
```
again = 'y'
while again == 'y':
    num1 = float(input('Enter a number: '))
    num2 = float(input('Enter another number: '))
    sum = num1 + num2
    print ('The sum of the numbers you entered is', sum)
    again = input('Do you want to do that again? (y/n): ')
```

3.
```
for number in range(0, 1001, 10):
    print(number)
```

4.
```
total = 0.0
    for counter in range(10):
        number = float(input('Enter a number: '))
        total += number
    print ('The total is', total)
```

5.
```
denominator = 30
total = 0
for numerator in range(1, 31):
    value = numerator / denominator
    total = total + value
    denominator -= 1
print (total)
```

6.
a.
```
x += 1
```
b.
```
x *= 2
```
c.
```
x /= 10
```
d.
```
x -= 100
```

7.
```
for row in range(10):
    for column in range(15):
        print('#', end='')
    print()
```

8.
```
number = float(input('Enter a positive nonzero number: '))
while number <= 0:
    print('That is an invalid value.')
    number = float(input('Enter a positive nonzero number: '))
print ('Thanks!')
```

9.
```
number = int(input('Enter a number between 1 and 100: '))
while number < 1 or number>100:
    print('That is an invalid value.')
```



```
        number = int(input('Enter a number between 1 and 100: '))
    print ('Thanks!')
```

Chapter 5

Multiple Choice

1. c
2. a
3. d
4. b
5. a
6. d
7. b
8. b
9. c
10. a
11. b
12. d
13. b
14. b
15. b
16. a
17. d
18. d
19. b
20. c
21. c

True or False

1. False
2. True
3. False
4. False
5. False
6. True
7. False
8. False
9. True
10. False
11. True
12. False
13. True
14. True
15. True

Short Answer

1. Functions can reduce the duplication of code within a program. If a specific operation is performed in several places in a program, a function can be written once to perform that operation, and then be executed any time it is needed. This is known as code reuse because you are writing the code to perform a task once and then reusing it each time you need to

perform the task.

2. A function definition has two parts: a header and a body. The *header* indicates the starting point of the function, and the *body* is a list of statements that belong to the function.

3. When the end of the function block is reached, the computer jumps back to the part of the program that called the function, and the program resume execution at that point.

4. A local variable is a variable that is declared inside a function. It belongs to the function in which it is declared, and only statements in the same function can access it.

5. A local variable's scope begins at the statement where the variable is created, and ends at the end of the function in which the variable is created.

6. Any statement in a program can change the value of a global variable. If you find that the wrong value is being stored in a global variable, you have to track down every statement that accesses it to determine where the bad value is coming from. In a program with thousands of lines of code, this can be difficult.

7. The `randrange` function, which is in the `random` module.

8. A `return` statement.

9. A function's input, processing, and output.

10. A function that returns either `True` or `False`.

11. Modules also make it easier to reuse the same code in more than one program. If you have written a set of functions that are needed in several different programs, you can place those functions in a module. Then, you can import the module in each program that needs to call one of the functions.

Algorithm Workbench

1.
```
def times_ten(number):
    result = number * 10
    print (result)
```

2.
```
show_value(12)
```

3. 3 will be assigned to `a`, 2 will be assigned to `b`, and 1 will be assigned to `c`.

4.
```
13.4
00
13.4
```

5. a.
```
my_function(a=2, b=4, c=6)
```
b. 2

6.
```
import random
```



```
rand = random.randint(1, 100)
```

7.
```
def half(value):
    return value / 2.0
```

8.
```
result = cube(4)
```

9.
```
def times_ten(number):
    return number * 10
```

10.
```
def get_first_name():
    name = input('Enter your first name: ')
    return name
```

Chapter 6

Multiple Choice

1. b
2. a
3. d
4. c
5. a
6. b
7. d
8. c
9. a
10. d
11. b
12. a
13. b
14. c
15. b

True or False

1. False
2. True
3. False
4. True
5. False
6. False
7. True
8. False
9. False

Short Answer

1. **Open the file**—Opening a file creates a connection between the file and the program. Opening an output file usually creates the file on the disk and allows the program to write data to it. Opening an input file allows the program to read data from the file.

Process the file—In this step data is either written to the file (if it is an output file) or read from the file (if it is an input file).

Close the file—When the program is finished using the file, the file must be closed. Closing a file disconnects the file from the program.

2. Closing a file disconnects the program from the file. In some systems, failure to close an output file can cause a loss of data. This happens because the data that is written to a file is first written to a buffer, which is a small "holding section" in memory. When the buffer is full, the system writes the buffer's contents to the file. This technique increases the system's performance, because writing data to memory is faster than writing it to a disk. The process of closing an output file forces any unsaved data that remains in the buffer to be written to the file.

3. A file's read position is an internally maintained value that marks the location of the next item that will be read from the file. When a file is opened, the read position is the beginning of the file.

4. The file's contents are preserved and any new data is written to the end of the file.

5. The file is created.

Algorithm Workbench

1.
```
outfile = open('my_name.txt', 'w')
outfile.write('John Locke')
outfile.close()
```

2.
```
infile = open('my_name.txt', 'r')
name = infile.read()
print(name)
infile.close()
```

3.
```
outfile = open('number_list.txt', 'w')
for number in range(1, 101):
    outfile.write(str(number) + '\n')
outfile.close()
```

4.
```
infile = open('number_list.txt', 'r')
line = infile.readline()
while line != '':
     line = line.rstrip('\n')
     number = int(line)
     print(number)
     line = infile.readline()
infile.close()
```

5.
```
total = 0
infile = open('number_list.txt', 'r')
line = infile.readline()
while line != '':
```



```
        line = line.rstrip('\n')
        number = int(line)
        total += number
        print(number)
        line = infile.readline()
    infile.close()
    print('The total is:', total)
```

6.
```
    outfile = open('number_list.txt', 'a')
```

7.
```
    # Delete John Perez from the file.
    import os # Needed for the remove and rename functions

    def main():
        # Create a bool variable to use as a flag.
        found = False

        # Open the original students.txt file.
        student_file = open('students.txt', 'r')

        # Open the temporary file.
        temp_file = open('temp.txt', 'w')

        # Read the first record's name field.
        name = student_file.readline()

        # Read the rest of the file.

        while name != '':
            # Read the score field.
            score = float(student_file.readline())

            # Strip the \n from the name.
            name = name.rstrip('\n')

            # If this is not the record to delete, then
            # write it to the temporary file.
            if name != 'John Perez':
                # Write the record to the temp file.
                temp_file.write(name + '\n')
                temp_file.write(str(score) + '\n')
            else:
                # Found the name. Set the found flag to True.
                found = True

            # Read the next name.
            name = student_file.readline()

        # Close the student file and the temporary file.
        student_file.close()
        temp_file.close()
```



```
        # Delete the original students.txt file.
        os.remove('students.txt')

        # Rename the temporary file.
        os.rename('temp.txt', 'students.txt')

        # If the search value was not found in the file
        # display a message.
        if found:
            print ('The file has been updated.')
        else:
            print ('That item was not found in the file.')

# Call the main function.
main()
```

8.
```
# Change Julie Milan's score to 100.
import os # Needed for the remove and rename functions
def main():
    # Create a bool variable to use as a flag.
    found = False

    # Open the original students.txt file.
    student_file = open('students.txt', 'r')

    # Open the temporary file.
    temp_file = open('temp.txt', 'w')

    # Read the first record's name field.
    name = student_file.readline()

    # Read the rest of the file.
    while name != '':

        # Read the score field.
        score = float(student_file.readline())

        # Strip the \n from the name.
        name = name.rstrip('\n')

        # If this is Julie Milan's, then write it with
        # the new score to the temporary file. Otherwise,
        # write the existing record to the temporary file.
        if name == 'Julie Milan':
            # Write the new record.
            temp_file.write(name + '\n')
            temp_file.write('100\n')
            # Set the found flag to True.
            found = True
```



```
            else:
                # Write the record to the temp file.
                 temp_file.write(name + '\n')
                 temp_file.write(str(score) + '\n')

            # Read the next name.
            name = student_file.readline()

        # Close the student file and the temporary file.
        student_file.close()
        temp_file.close()

        # Delete the original students.txt file.
        os.remove('students.txt')

        # Rename the temporary file.
        os.rename('temp.txt', 'students.txt')

        # If the search value was not found in the file
        # display a message.
        if found:
            print('The file has been updated.' )
        else:
            print('Julie Milan was not found in the file.' )

    # Call the main function.
    main()
```

9.
```
This code caused a ValueError.
The end.
```

10.
```
An error happened.
The end.
```

Chapter 7

Multiple Choice

1. a
2. b
3. c
4. d
5. b
6. c
7. b
8. d
9. c
10. b or c
11. a
12. b
13. a

14. b
15. d
16. d

True or False

1. False
2. True
3. True
4. False
5. False
6. True
7. True
8. False

Short Answer

1. a. 5
   b. 0
   c. 4

2. a. 3
   b. 1
   c. 3

3. `[4, 6]`

4. `[6, 7]`

5. `[5, 6, 7, 8]`

6. `[2, 2, 2, 2, 2]`

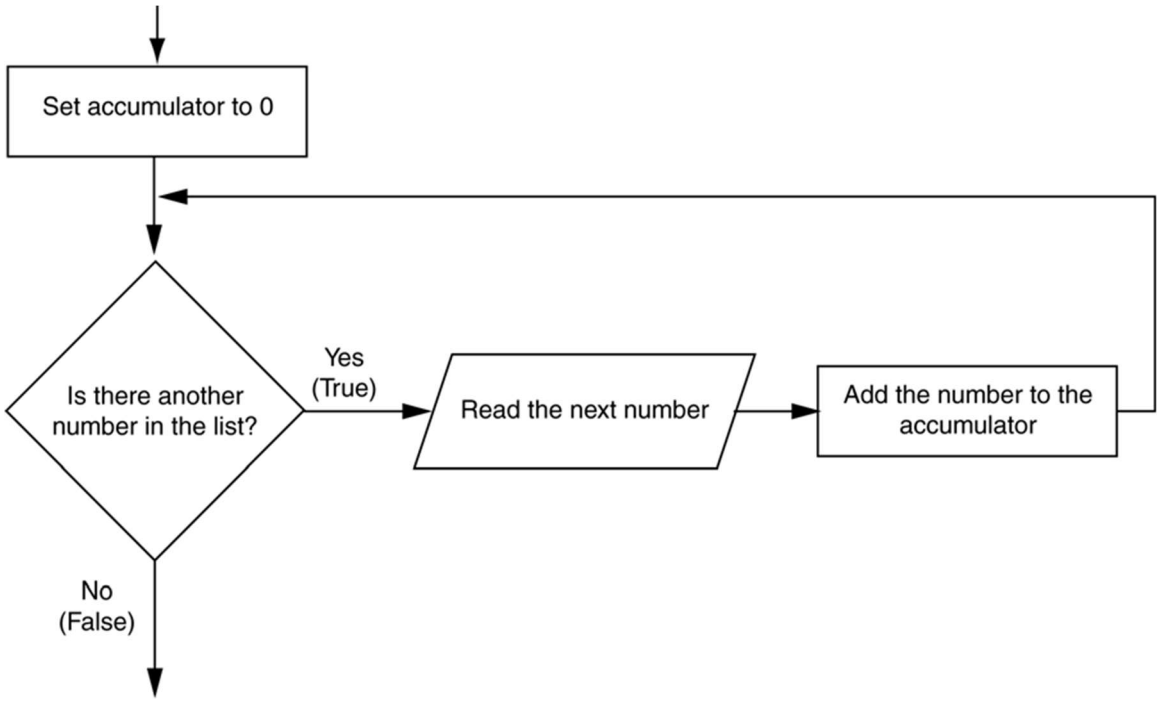
Algorithm Workbench

1.
```
names = ['Einstein', 'Newton', 'Copernicus', 'Kepler']
```

2.
```
for element in names:
    print(element)
```

3.
```
for item in numbers1:
    numbers2.append(item)
```

4.



5.
```
def list_total(arg):
    # Set the accumulator to 0.
    total = 0

    # Add the values in arg to total.
    for val in arg:
        total += val

    # Return the total.
    return total
```

6.
```
if 'Ruby' in names:
    print('Hello Ruby')
else:
    print('no Ruby')
```

7.
```
[40, 50, 60, 10, 20, 30]
```

8.
```
list2 = [item**2 for item in list1]
```

9.
```
list2 = [item for item in list1 if item > 100]
```

10.
```
list2 = [item for item in list1 if item % 2 == 0]
```

11.
```
mylist = []
for i in range(5):
    mylist.append([])
    for j in range(3):
        mylist[i].append(None)
```



```
for i in range (len(mylist)):
    for j in range (len(mylist[i])):
        mylist[i][j] = input('Enter a value: ')
```

Chapter 8

Multiple Choice

1. c
2. d
3. b
4. c
5. a
6. c
7. d
8. a
9. b
10. b

True or False

1. True
2. True
3. False
4. True
5. False

Short Answer

1.
```
yesnoyes
```

2.
```
abcabcabc
```

3.
```
cde
```

4.
```
[5, 6]
```

5.
```
joe
JOE
joe
joe
```

Algorithm Workbench

1.
```
if choice.upper() == 'Y'
```

2.
```
counter = 0
for ch in mystring:
    if ch == ' ':
        counter +=1
print('The number of space characters is', counter)
```

3.
```
counter = 0
```



```
for ch in mystring:
    if ch.isdigit():
        counter +=1
print('The number of digits is', counter)
```

4.
```
counter = 0
for ch in mystring:
    if ch.islower():
        counter +=1
print('The number of lowercase characters is', counter)
```

5.
```
def com_check(mystring):
    if mystring.endswith('.com'):
        return True
    return False
```

6.
```
newString = oldstring.replace('t','T')
print(newString)
```

7.
```
def reverse(myString):
    newString = ''
    for counter in range(len(myString)-1, -1, -1):
        newString += myString[counter]
    return newString
```

8.
```
print(mystring[:3])
```

9.
```
print(mystring[-3:])
```

10.
```
mylist = mystring.split('>')
```

Chapter 9

Multiple Choice

1. b
2. d
3. b
4. a
5. c
6. a
7. d
8. c
9. b
10. b
11. c
12. b
13. a
14. a

15. c
16. b
17. d

True or False

1. False
2. True
3. True
4. False
5. False
6. True
7. False
8. True
9. False
10. True

Short Answer

1. `2`

2. `1`

3. `'Not found'`

4. `222`

5. You use the `del` statement and specify the key of the element that you want to delete. For example:
   ```
   del dict[key]
   ```

6. You use the `len` function.

7. `[4,5]`

8. 1, 2, 3

9. `{'S','a','t','u','r','n'}`

10. An error will be raised. A number is not iterable and cannot be used to create a set.

11. `{'a', 'b', 'c', 'd', ' '}`

12. `{2, 4, 6}`

13. `{'a', 'bb', 'ccc', 'dddd'}`

14. `4`

15. `{10, 20, 30, 40, 50, 60}`

16. `{'p', 's'}`

17. `{'f'}`

18. `{'a', 'b', 'c'}`

19. `{1, 4}`

20. `set2` is a subset of `set1`; `set1` is a superset of `set2`

Algorithm Workbench

1.
```
mydict = {'a':1, 'b':2, 'c':3}
```

2.
```
mydict = {}
```

3.
```
if 'James' in dct:
    print(dct['James'])
else:
    print('James is not in the dictionary.')
```

4.
```
if 'Jim' in dct:
    del dct['Jim']
else:
    print('Jim is not in the dictionary.')
```

5.
```
mylist = [10, 20, 30, 40]
myset = set(mylist)
```

6.
```
set3 = set1.union(set2)
```

7.
```
set3 = set1.intersection(set2)
```

8.
```
set3 = set1.difference(set2)
```

9.
```
set3 = set2.difference(set1)
```

10.
```
set3 = set1.symmetric_difference(set2)
```

11.
```
numbers2 = {item:item * 10 for item in numbers}
```

12.
```
high_scores = {k:v for k,v in test_averages.items() if v > 90}
```

13.
```
import pickle
output_file = open('mydata.dat', 'wb')
pickle.dump(dct, output_file)
output_file.close()
```

14.
```
import pickle
input_file = open('mydata.dat','rb')
```



```
dct = pickle.load(input_file)
input_file.close()
```

Chapter 10

Multiple Choice

1. b
2. d
3. c
4. d
5. b
6. d
7. c
8. a
9. b
10. a
11. d
12. a

True or False

1. False
2. True
3. False
4. False
5. True
6. True
7. False

Short Answer

1. Encapsulation refers to the combining of data and code into a single object.

2. When an object's data attributes are hidden from outside code, and access to the data attributes is restricted to the object's methods, the data attributes are protected from accidental corruption. In addition, the code outside the object does not need to know about the format or internal structure of the object's data. The code only needs to interact with the object's methods. When a programmer changes the structure of an object's internal data attributes, he or she also modifies the object's methods so that they may properly operate on the data. The way in which outside code interacts with the methods, however, does not change.

3. A class is a description of an object. When the program is running, it can use the class to create, in memory, as many objects of a specific type as needed. Each object that is created from a class is called an instance of the class.

4. The name of the method is `get_dollar`. The name of the variable is `wallet`.

5. It references the object that the method is to operate on.

6. By starting its name with two underscore characters.

7. By passing the object to the built-in `str` function or the built-in `print` function.

Algorithm Workbench

1. `my_car.go()`

2.
```
class Book:
    # Initializer
    def __init__(self, title, author, pub):
        self.__title = title
        self.__author = author
        self.__pub = pub

    # Mutators
    def set_title(self, title):
        self.__title = title

    def set_author(self, author):
        self.__author = author

    def set_pub(self, pub):
        self.__pub = pub

    # Accessors
    def get_title(self):
        return self.__title

    def get_author(self):
        return self.__author

    def get_pub(self):
        return self.__pub

    # __str__ method
    def __str__(self):
        state_string = ('Title: ' + self.__title + '\n' +
                        'Author: ' + self.__author + '\n' +
                        'Publisher: ' + self.__pub + '\n')

        return state_string
```

3. a. After eliminating duplicates, objects, and primitive values, the potential classes are: *bank*, *account*, and *customer*.
   b. The only class needed for this particular problem is *account*.
   c. The account class knows its balance and interest rate.
   The account can calculate interest earned.

Chapter 11

Multiple Choice

1. b
2. c

3. b
4. a
5. c

True or False

1. True
2. False
3. True
4. False
5. False

Short Answer

1. The superclass's fields and methods.

2. `Felis` is the superclass and `Tiger` is the subclass.

3. When a subclass method has the same name as a superclass method, it is said that the method is overridden.

Algorithm Workbench

1.
```
class Poodle(Dog):
```

2.
```
I'm a plant.
I'm a tree.
```

3.
```
class Cola(Beverage):
    def __init__(self):
        Beverage.__init__(self, 'cola')
```

Chapter 12

Multiple Choice

1. c
2. b
3. a
4. d
5. c
6. d
7. b
8. a
9. b
10. a

True or False

1. True
2. False
3. False
4. False

Short Answer

1. When `times` is not greater than 0.

2. The base case is $n = 0$. The recursive case is $n > 0$.

3. No, recursion is not required. You can alternatively use a loop.

4. By reducing the problem with each recursive call, the base case will eventually be reached and the recursion will stop.

5. The value of an argument is usually reduced.

Algorithm Workbench

1.
```
10
```

2.
```
0
1
2
3
4
5
6
7
8
9
10
```

3.
```
def traffic_sign(n):
    if n > 0:
        print ('No Parking')
        traffic_sign(n - 1)
```

Chapter 13

Multiple Choice

1. b
2. a
3. d
4. c
5. b
6. c
7. a
8. b
9. d
10. a
11. a
12. d
13. a
14. c

15. a
16. b
17. d

True or False

1. False
2. False
3. False
4. True
5. False

Short Answer

1. The program.

2. The `pack` method arranges a widget in its proper position, and it makes the widget visible when the main window is displayed.

3. It runs like an infinite loop until you close the main window.

4. With one stacked on top of the other.

5. By passing `side='left'` as an argument to the widget's `pack` method.

6. By calling the widget's `get` method.

7. First you create a `StringVar` object. Then, you create a `Label` widget and associate it with the `StringVar` object. From that point on, any value that is then stored in the `StringVar` object will automatically be displayed in the `Label` widget.

8. When you create a group of `Radiobuttons`, you associate them all with the same `IntVar` object. You also assign a unique integer value to each `Radiobutton` widget. When one of the `Radiobutton` widgets is selected, it stores its unique integer value in the `IntVar` object.

9. As with `Radiobuttons`, you can use an `IntVar` object along with a `Checkbutton` widget. Unlike `Radiobuttons`, however, you associate a different `IntVar` object with each `Checkbutton`. When a `Checkbutton` is selected, its associated `IntVar` object will hold the value 1. When a `Checkbutton` is unselected, its associated `IntVar` object will hold the value 0.

Algorithm Workbench

1.
```
self.label = tkinter.Label(self.main_window,
                           text='Programming is fun!')
```

2.
```
self.label1.pack(side='left')
self.label2.pack(side='left')
```

3.
```
self.top_frame = tkinter.Frame(self.main_window)
```


4.
```
tkinter.messagebox.showinfo('Program Paused',
                            'Click OK when ' +
                            'you are ready to continue.')
```

5.
```
self.my_button = tkinter.Button(self.button_frame,
                                 text='Calculate',
                                 command=self.calculate)
```

6.
```
self.quit_button = tkinter.Button
                   (self.button_frame,
                    text='Quit',
                    command=self.main_window.destroy)
```

7.
```
var = int(self.data_entry.get())
```

8.
a)
```
self.canvas.create_line(0, 0, 199, 199,
                        fill='blue', width=3)
```
b)
```
self.canvas.create_rectangle(50, 50, 100, 100,
                             outline='red',
                             fill='black')
```
c)
```
self.canvas.create_oval(50, 50, 150, 150,
                        outline='green')
```
d)
```
self.canvas.create_arc(20 20, 180, 180,
                       start=0,
                       extent=90,
                       fill='blue')
```

Chapter 14

Multiple Choice

1. c
2. a
3. c
4. d
5. c
6. b
7. a
8. d
9. b
10. a
11. b
12. a
13. d
14. b
15. c
16. b
17. d
18. c
19. c

20. d
21. a
22. c
23. d
23. a
24. d
25. b
26. c
27. d
28. c

True or False

1. True
2. False
3. False
4. False
5. True
6. True
7. False
8. True
9. False
10. True

Short Answer

1. Traditional text and binary are not practical when a large amount of data must be stored and manipulated. Many businesses keep hundreds of thousands, or even millions of data items in files. When a text or binary file contains this much data, simple operations such as searching, inserting, and deleting, become cumbersome and inefficient.

3. The data that is stored in a database is organized into one or more tables. Each table holds a collection of related data. The data that is stored in a table is then organized into rows and columns. A row is a complete set of information about a single item. The data that is stored in a row is divided into columns. Each column is an individual piece of information about the item.

4. A primary key is a column that holds a unique value for each row, and can be used to identify specific rows

5. • The Python `int` type corresponds with the SQLite `INTEGER` type
   • The Python `float` type corresponds with the SQLite `REAL` type
   • The Python `str` type corresponds with the SQLite `TEXT` type

6. What are the relational operators in SQL for the following comparisons?

   - > (Greater-than)
   - < (Less-than)
   - >= (Greater-than or equal-to)
   - <= (Less-than or equal-to)
   - == (Equal-to)
   - = (Equal-to)

- != (Not equal-to)
- <> (Not equal-to)

7. The name of the table is `Employee`.
   The name of the column is `Name`.

8. A parameterized query contains placeholders (in SQLite the placeholders are question marks). When the statement is passed to the DBMS, the values of specified variables are inserted in the place of the placeholders.

9. The `fetchall` method returns a list containing all the rows that result from a `SELECT` statement. The `fetchone` method returns only one row each time it is called, as a tuple.

10. An aggregate function performs a calculation on a set of values from a database table and returns a single value.

11. The `Cursor` object has a public data attribute named `rowcount` that contains the number of rows that were altered by the most recently executed SQL statement. After issuing an `UPDATE` statement, you can read the value of the `rowcount` attribute to determine the number of rows that were updated.

12. A composite key is a key that is created by combining two or more existing columns.

13. 4

14. 20

15. `RowID`

16. Create, Read, Update, Delete

17. A foreign key is a column in one table that references a primary key in another table.

Algorithm Workbench

1.
```
SELECT * FROM Stock
```

2.
```
SELECT TradingSymbol FROM Stock
```

3.
```
SELECT TradingSymbol, NumShares FROM Stock
```

4.
```
SELECT TradingSymbol FROM Stock
WHERE PurchasePrice > 25.00
```

5.
```
SELECT * FROM Stock
WHERE TradingSymbol LIKE 'SU%'
```


6.
```
SELECT TradingSymbol FROM Stock
WHERE SellingPrice > PurchasePrice AND
NumShares > 100
```

7.
```
SELECT TradingSymbol, NumShares FROM Stock
WHERE SellingPrice > PurchasePrice AND
NumShares > 100
ORDER BY NumShares
```

8.
```
INSERT INTO Stock
  (TradingSymbol, CompanyName, NumShares,
   PurchasePrice, SellingPrice)
VALUES
  ('XYZ', 'XYZ Company', 150, 12.55, 22.47)
```

9.
```
UPDATE Stock
   SET TradingSymbol = 'ABC'
   WHERE TradingSymbol = 'XYZ'
```

10.
```
DELETE Stock WHERE NumShares < 10
```

11.
```
CREATE TABLE IF NOT EXISTS Stock (TradingSymbol TEXT,
                                  CompanyName TEXT,
                                  NumShares INTEGER,
                                  PurchasePrice REAL,
                                  SellingPrice REAL)
```

12.
```
DROP TABLE IF EXISTS Stock
```

