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. c
9. a
10. b
11. d
12. b
13. b
14. b
15. a

16. d
17. d
18. b
19. c
20. 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
```