Chapter 15 Recursion

1. A function that calls itself is a recursive function. One or more base cases (the simplest case) are used to stop recursion. Every recursive call reduces the original problem, bringing it increasingly close to a base case until it becomes that case.

2. six times. (base case factorial(0))

3. f(n) = 2 if n = 1
   f(n) = 2 * 2^(n-1) for (n > 1)

4. f(n) = x if n = 1
   f(n) = x * x^(n-1) for (n > 1)

5. f(n) = 1 if n = 1
   f(n) = f(n-1) + n for (n > 1)

6. See the text

7. 25 times (Why?

   number of time fib is invoked in fib(0) =
   1

   number of time fib is invoked in fib(1) =
   1

   number of time fib is invoked in fib(2) =
   1+ number of time fib is invoked in fib(1)+number of time fib is invoked in fib(2)
   =1+1+1=3

   number of time fib is invoked in fib(3) =
   1+ number of time fib is invoked in fib(1)+number of time fib is invoked in fib(2) =
   1+1+3=5

   number of time fib is invoked in fib(4) =
   1+ number of time fib is invoked in fib(2)+number of time fib is invoked in fib(3) =
   1+3+5=9

   number of time fib is invoked in fib(5) =
   1+ number of time fib is invoked in fib(3)+number of time fib is invoked in fib(4) =
   1+5+9=15

   number of time fib is invoked in fib(6) =

1+ number of time fib is invoked in fib(4)+number of time fib is invoked in fib(5) = 1+9+15=25

8. (a) The output is 15 (5 + 4 + 3 + 2 + 1 = 15)
   Base case: if (n == 1)
   Recursive call: f(n – 1)
   (b) 7654321
   Base case: if (n > 0)
   Recursive call: f(n / 10)

9. omitted.
10. omitted.
11. (a) The output is 5 4 3 2 1
    (b) The output is 1 2 3 4 5

12. n / 10 should be n // 10.

13. omitted.
14. omitted.

15. Use os.path.isfile to check if a file exists. Use os.path.getsize(file) to return the size of the file. Use os.listdir(directory) to return the files and subdirectory under this directory.

16.

2^5 – 1 = 31 times

17.

- Any recursive functions can be converted into a non-recursive function. (TRUE)
- Recursive function usually takes more time and memory to execute than non-recursive functions. (TRUE)
- Recursive functions are *always* simpler than non-recursive functions. (FALSE)
- There is always a condition statement in a recursive function to check whether a base case is reached. (TRUE)

18.
When a function is invoked, its contents are placed into a stack. If a function is recursively invoked, it is possible that the stack space is exhausted. This causes stack overflow.

19.

A recursive method is said to be *tail recursive* if there are no pending operations to be performed on return from a recursive call.

20.
Tail recursion is desirable, because the function ends when the last recursive call ends. So there is no need to store the intermediate calls in the stack. Some compilers can optimize tail recursion to reduce stack space.

21.
Yes.

22.

```
# Return the Fibonacci number for the specified index
def fib(index):
    return fib1(index, 1, 0)

# Auxiliary tail-recursive function for fib
def fib1(index, next, result):
    if (index == 0):
        return result
    else:
        return fib1(index - 1, next + result, next)
```