Recursion Practice

1. Write a function f1(list) that returns the sum of the elements in the list.

```
>>> f1([1,2,3,4])
10

>>> f1([])
0
```

2. Consider the following function:

```
f(n) = {  n//2     if n is even
          3n+1     if n is odd   }
```

Write a function f2(n) that returns the number of steps of the function f(n) until it reaches 1 (this is also known as the Collatz Conjecture).
For example, consider f(6):6→3→10→5→16→8→4→2→1, since there are a total of 9 steps, f2(6) evaluates to 9.

```
>>> f2(1)
1

>>> f2(6)
9

>>> f2(11)
15

>>> f2(637228127)
276
```

3. Write a function f3(list) that prints out the elements in the list in reverse order.

```
>>> f3([1,2,3])
3
2
1

>>> f3([])

>>> f3([3,2,1])
1
2
3
```

4. Write a function f4(list) that multiplies all of the odd elements in the list by 3 and prints out each tripled element.

```
>>> f4([1,2,3,4])
3
9

>>> f4([2,4])

>>> f4([11,42,63,15])
33
189
45
```

5. Write a function f5(list) that multiplies all of the odd elements in the list by 3 and prints out each element of the modified list in reverse order.

```
>>> f5([1,2,3,4])
4
9
2
3

>>> f5([2,4])
4
2

>>> f5([11,42,64,15])
45
64
42
33
```

6. Write a function f6(lst) that takes any multidimensional list and returns a one dimensional list with the same values. This is also known as flattening a list. Remember that you can use type([1,2,3]) == list to determine if something is a list. There should be one base case and two recursive cases.

```
>>> f6(['baa'])
['baa']

>>> f6(['baa', [4, True, [10, 5], [1, 2, ['moo']]], ["chirp"]])
['baa', 4, True, 10, 5, 1, 2, 'moo', "chirp"]

>>> f6([])
[]

>>> f6([[[[[[[[[[[[[[23]]]]]]]]]]]]]])
[23]
```

7. Consider a function Ln:

Ln= 2 if n=0;
1 if n=1;
Ln−1+Ln−2 if n>1;

Write a function f7(n) that calculates Ln

```
>>> f7(3)
4

>>> f7(14)
843

>>> f7(0)
2

>>> f7(22)
39603
```

8. Write a function f8(s) that returns True if s is a palindrome, and False otherwise.

```
>>> f8("")
True

>>> f8("kayak")
True

>>> f8("penguin")
False

>>> f8("a")
True
```

9. Write a function f9(n) that returns n!

```
>>> f9(0)
1

>>> f9(1)
1

>>> f9(2)
2

>>> f9(3)
6
```

10. Write a function f10(list) that returns len(list).

```
>>> f10([1,2,3])
3

>>> f10([])
0

>>> f10([2])
1
```

11. Write a function f11(list) that returns the last element in the list.

```
>>> f11([1,2,3])
3

>>> f11([])

>>> f11([1])
1
```

12. Write a function f12(n) that prints the numbers n through 1 in descending order.

```
>>> f12(3)
3
2
1

>>> f12(0)

>>> f12(1)
1
```

13. Write a function f13(n) that returns the number of digits in n. You may assume n is a positive integer.

```
>>> f13(9175)
4

>>> f13(34)
2

>>> f13(268)
3

>>> f13(0)
1
```

14. Write a function f14(list) that returns the first odd number in the list, and None if there are no odd numbers in the list.

```
>>> f14([1,2,3])
1

>>> f14([2,4])

>>> f14([2,4,6,8,10,3])
3
```

15. Write a function f15(list) that returns the sum of all the odd numbers in the list.

```
>>> f15([1,2,3])
4

>>> f15([2,4])
0

>>> f15([1,3,6,9])
13
```

16. Write a function f16(list) that returns a list of all the odd numbers in the list.

```
>>> f16([1,3,5,7])
[1, 3, 5, 7]

>>> f16([2,4])
[]

>>> f16([1,2,3,4,5])
[1, 3, 5]
```

17. Write a function f17(list) that returns the second to last element in the list. Assume len(list) > 1.

```
>>> f17([1,2])
1

>>> f17([1,2,3,4])
3

>>> f17([1,2,3])
2
```

18. Write a function f18(a,b) that returns the greatest common divisor of a and b.

```
>>> f18(5,4)
1

>>> f18(40,60)
20

>>> f18(9,3)
3
```

19. Write a function f19(list1, list2) that merges list1 and list2 in ascending order. Assume list1 and list2 are already sorted.

```
>>> f19([1,2,3],[4,5])
[1, 2, 3, 4, 5]

>>> f19([4,5],[1,2,3])
[1, 2, 3, 4, 5]

>>> f19([],[1,2,3])
[1, 2, 3]

>>> f19([1,2,3],[])
[1, 2, 3]

>>> f19([], [])
[]
```

20. Write a function f20(list) that mergesorts the list. Consider using f19(list1, list2) for the merging step.

```
>>> f20([3,2,1])
[1, 2, 3]

>>> f20([])
[]

>>> f20([5,3,1,2,4,6])
[1, 2, 3, 4, 5, 6]
```

21. Write a function f21(tree) that returns the height of the tree. The tree has the structure [value, left subtree, right subtree].

```
>>> f21([])
0

>>> f21([1,[],[]])
1

>>> f21([1,[1,[],[]],[]])
2
```

22. Write a function f22(tree) that returns the number of nodes in the tree. The tree has the structure [value, left subtree, right subtree].

```
>>> f22([])
0

>>> f22([1,[],[]])
1

>>> f22([1,[1,[],[]],[1,[],[]])
3
```

23. Write a function f23(tree) that returns the sum of the nodes in the tree. The tree has the structure [value, left subtree, right subtree].

```
>>> f23([])
0

>>> f23([1,[],[]])
1

>>> f23([1,[2,[],[]],[3,[],[]]])
6
```

24. Write a function f24(tree) that prints out the values of the tree in ascending order. The tree has the structure [value, left subtree, right subtree] and is a binary search tree.

```
>>> f24([])

>>> f24([1,[],[]])
1

>>> f24([2,[1,[],[]],[3,[],[4,[],[]]]])
1
2
3
4
```

25. Write a function f25(tree) that returns the smallest element in the tree. The tree has the structure [value, left subtree, right subtree] and is a binary search tree. Return -1 if the tree is empty.

```
>>> f25([])
-1

>>> f25([1,[],[]])
1

>>> f25([2,[1,[],[]],[3,[],[4,[],[]]]])
1
```