

COSC 350: Data Structures

Practice Problems for Recursion

These problems will not be collected or graded. They are provided simply to give you practice. Most are of a difficulty such that you should be able to solve one on a class quiz. In each case you should write a recursive function to complete the

1. Addition can be implemented using just an increment and a decrement function, functions that add and subtract one. You increment one number while decrementing another until the decremented number is zero.
2. Multiplication can be implemented using repeated additions.
3. Division can be implemented using repeated subtractions.
4. Exponentiation can be implemented using repeated multiplications.
5. Any sort of while-loop can be rewritten as a recursive function. Find examples of while-loops and practice rewriting them.
6. Any sort of for-loop can be rewritten as a recursive function. Find examples of for-loops and practice rewriting them.
7. As the course progresses, we will see that lists can be thought of as recursive structures. But you already know enough to write some list operations recursively using Python's built-in lists. Write a function to count the number of items in a list.
8. Write a function to add one to each item in a list of numbers.
9. Write a function to multiply by two each item in a list of numbers.
10. Write a function to map a function over items in a list, i.e., apply the function to each item in the list. E.g., `map(abs, [1, -2, 3, -4, -5])` would return `[1, 2, 3, 4, 5]`.
11. Write a function to print the elements in a list in order. Then change the function so that the list is printed in reverse order. You should only need to make one change in the order of the statements in the function.
12. Rewrite the previous function using strings.
13. Write a function that returns the reverse of a list ... of a string.

14. Write a function that takes two lists of numbers of equal length and returns a new list that is the pair-wise sum of the numbers in the original lists.
15. Write a function that concatenates two lists.
16. Write a function to see if an item is on a list.
17. Write a function to find the location (index) of the first occurrence an item on a list.
18. Write a function to count the number of times an item occurs on a list.
19. Write a function to generate a list of items give the item and a count. E.g.,
`make(3, 'cats')` would return `['cats', 'cats', 'cats']`.
20. Write a function to return the largest item in a list. ... the smallest item.
21. Write a function to return the average of a list.
22. Write a function to return the median of a sorted list.
23. Write a function to remove duplicate entries in a list.
24. *Write a function to return the cross product of two list. E.g.,
`cross([1,2,3],[4,5])` would return
`[[1,4],[1,5],[2,4],[2,5],[3,4],[3,5]]`.
25. *Write a function to flatten a nested list. E.g.,
`flatten([[1],[2,3,[4]],[],[[[5]]]])` would return `[1,2,3,4,5]`.

While writing these functions, you may want to ask yourself these questions:

1. What is the complexity of the function?
2. Is the function linear- or tree-recursive?
3. Can the function be written to be tail-recursive?
4. Do you need to helper functions? ... for example, to check to see if the data is correctly formatted or valid?