**Chapter 10 test problems**

1. Implement function oddCount() that takes a list of integers as input and returns the number of odd integers in the list. Your implementation must be recursive, without any loops.  
  
>>> oddCount([])  
0  
>>> oddCount([1,2,0])  
1  
>>> oddCount([1,2,-4,-5])  
2

2. Implement function negCount() that takes a list of integers as input and returns the number of negative integers in the list. Your implementation must be recursive, without any loops.  
  
>>> negCount([])  
0  
>>> negCount([3,2,1,0])  
0  
>>> negCount([-3,2,-1,0])  
2

3. Implement recursive function prod() that takes a list of numbers and returns their product. If the input list is empty, the function should return 1. Your implementation must be recursive, without any loops.  
  
>>> prod([])  
1  
>>> prod([4])  
4  
>>> prod([4, 2, 6])  
48

4. Implement recursive function prod2() that takes a list of numbers as input and returns the product of the *non-zero* numbers in the list. If the input list is empty, the function should return 1. Your implementation must be recursive, without any loops.

>>> prod2([])

1

>>> prod2([4, 2, 6])

48

>>> prod2([4, 0, 2, 6, 0])

48

5. Write recursive function interval() that takes, as input, a list lst of number as well as two numbers a and b such that a < b. The function should return a list containing all numbers in lst between a and b. Your implementation must be recursive, without any loops.

>>> lst = [-1, 6, 4, -2, 0, 3]

>>> interval(lst, -2, 5)

[-1, 4, 0, 3]

6. Implement recursive function minmax() that takes a non-empty list of numbers as input and returns the minimum and maximum numbers in the list as a 2-tuple. Your implementation must be recursive, without any loops and without calling the built-in min() or max() functions on a sequence type (such as a list).

>>> minmax([4])

(4, 4)

>>> minmax([4,3])

(3, 4)

>>> minmax([-1, 6, 4, -2, 0, 3])

(-2, 6)

7. Write a recursive method binary() that takes a non-negative integer n and prints the binary representation of integer n.  
  
>>> binary(0)  
0  
>>> binary(1)  
1  
>>> binary(3)  
11  
>>> binary(9)  
1001

8. Implement the recursive function design() that takes a non-negative integer as input and prints a design on the screen as shown below.  
  
>>> design(0)  
>>> design(1)  
\*  
>>> design(2)  
\*  
\*\*  
\*  
>>> design(3)  
\*  
\*\*  
\*  
\*\*\*  
\*  
\*\*  
\*  
>>> design(4)  
\*  
\*\*  
\*  
\*\*\*  
\*  
\*\*  
\*  
\*\*\*\*  
\*  
\*\*  
\*  
\*\*\*  
\*  
\*\*  
\*

9. Implement a function mystery() that takes as input a positive integer n and answers this question: How many times can n be halved (using integer division) before reaching 1? This value should returned. Your implementation must be recursive, without any loops.

>>> mystery(1)

0

>>> mystery(3)

1

>>> mystery(4)

2

>>> mystery(11)

3

10. Write function collatz() that takes a positive integer x as input and prints the Collatz sequence starting at x. A Collatz sequence is obtained by repeatedly applying this rule to the previous number x in the sequence:

x =

Your function should stop when the sequence gets to number 1. Your implementation must be recursive, without any loops.

>>> collatz(1)

1

>>> collatz(10)

10

5

16

8

4

2

1

11. Function recMap() takes a list and a function as input, applies the function to every item in the list, and returns a list containing the obtained values:

def recMap(lst, f):

'returns list [f(lst[0]), f(lst[1]), ..., f(lst[n-1])]'

if len(lst) == 0:

return []

return recMap(lst[:-1], f) + [f(lst[-1])]

Make use of this function in to capitalize the strings in a list of strings. For example, when applied to list lst defined as

>>> lst = ['anne', 'betsy', 'chad', 'david', 'enrico'].

your expression should return ['Anne', 'Betsy', 'Chad', 'David', 'Enrico']. Make your solution as simple and short as you can.

12. Implement function dirSize() that takes a pathname (as a string) as input. If the pathname refers to a regular file, the function should return the size of the file; if the pathname is a folder, the function should return the sum of the sizes of all files contained in the directory, whether directly or indirectly. Notes: Function getsize() in module os.path takes a pathname as input and returns the size of the corresponding file.

>>> dirSize('.')

1376025

>>> dirSize('..')

88808283

13. Function search() is an implementation of binary search on input list lst:

def search(lst, target, i, j):

if i == j:

return

mid = (i+j)//2

if lst[mid] == target:

return mid

if target < lst[mid]:

return search(lst, target, i, mid)

else:

return search(lst, target, mid+1, j)

What are the the values of lst[mid] during the execution of

>>> lst = [-4,-1,0,3,7,9,14,31,53,76,77,85,92,99]

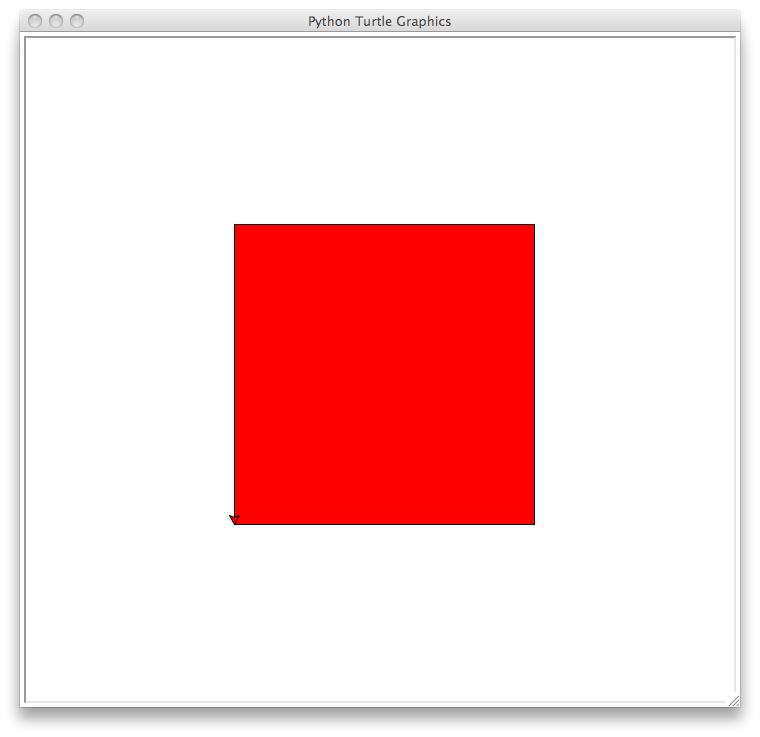
>>> search(lst, 78, 0, len(lst))

14. Implement recursive program pattern() to produce the turtle graphics patterns shown below. To help you, you can assume that you have available a user-defined function square() that takes length len and coordinates x and y as input and draws a red square with a black border of side length len and centered at coordinates (x,y). Here is how to use this function:

>>> from turtle import Screen, Turtle  
>>> s = Screen()  
>>> t = Turtle()  
>>> square(t,0,0,300) # makes turtle t draw square centered at

(0,0) with side length 300 pixels

Executing the above would result in the following display:

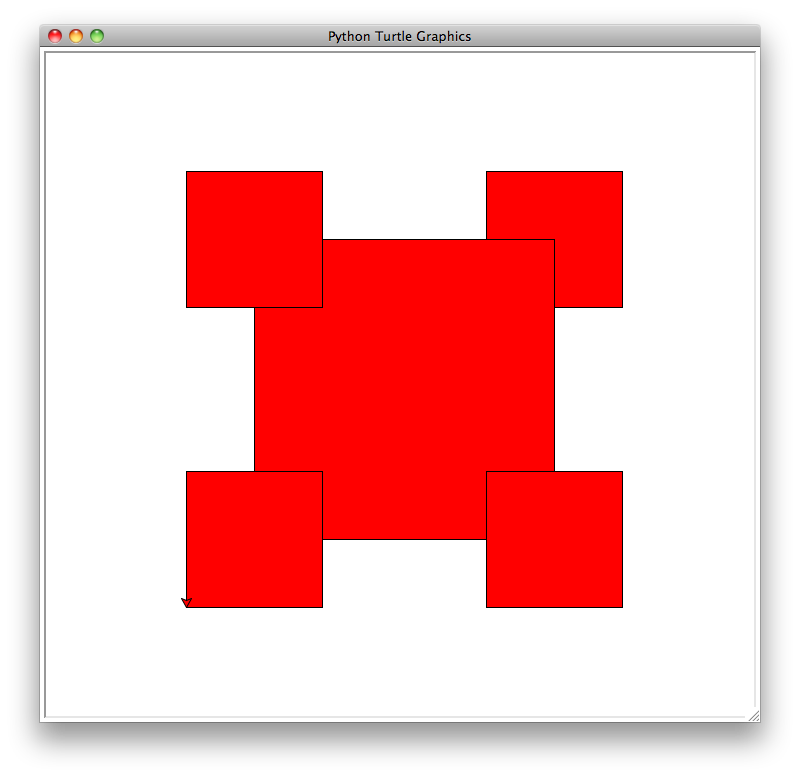


The above happens to also be pattern(1), or more precisely the output of pattern(t,0,0,300,1). The base case, pattern(0) is just nothing... Here is pattern(2):

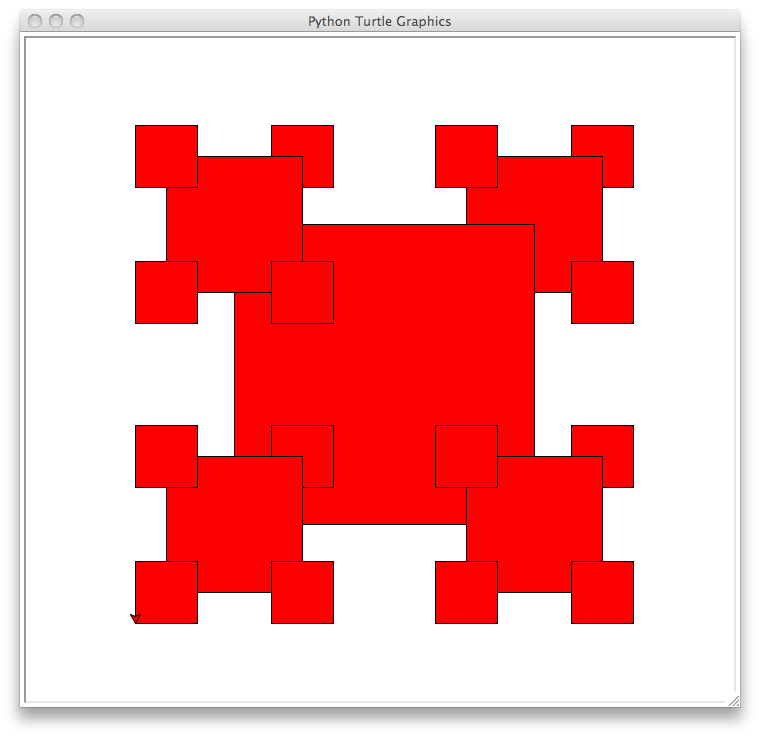
>>> pattern(t,0,0,300,2) # makes turtle t draw pattern 2 with

the main square centered at

(0,0) and with side length 300 pixels



Each of the four small squares is centered at an endpoint of the original square and has length 1/2.2 of the original square. Note how some squares are drawn on top of others: the squares on top are drawn *after* the squares on the bottom. The patterns pattern(3) and pattern(4) are similarly defined, using recursion:

>>> pattern(t,0,0,300,3)  
  
  
>>> pattern(t,0,0,300,4)  
  
