# File: recursion2.py

# Description:

# Student Name:

# Student UT EID:

# Partner's Name:

# Partner's UT EID:

# Course Name: CS 313E

# Unique Number:

# Date Created:

# Date Last Modified:

# Given an array of ints, is it possible to choose a group of some

# of the ints, such that the group sums to the given target?

# This is a classic backtracking recursion problem. Once you

# understand the recursive backtracking strategy in this problem,

# you can use the same pattern for many problems to search a space

# of choices. Rather than looking at the whole array, our convention

# is to consider the part of the array starting at index start and

# continuing to the end of the array. The caller can specify the

# whole array simply by passing start as 0. No loops are needed --

# the recursive calls progress down the array.

#def groupSum(start, nums, target):

# Given an array of ints, is it possible to choose a group of some

# of the ints, beginning at the start index, such that the group

# sums to the given target? However, with the additional constraint

# that all 6's must be chosen. (No loops needed.)

#def groupSum6(start, nums, target):

# Given an array of ints, is it possible to choose a group of some

# of the ints, such that the group sums to the given target with this

# additional constraint: If a value in the array is chosen to be in

# the group, the value immediately following it in the array must

# not be chosen. (No loops needed.)

#def groupNoAdj(start, nums, target):

# Given an array of ints, is it possible to choose a group

# of some of the ints, such that the group sums to the given

# target with these additional constraints: all multiples of

# 5 in the array must be included in the group. If the value

# immediately following a multiple of 5 is 1, it must not

# be chosen. (No loops needed.)

#def groupSum5(start, nums, target):

# Given an array of ints, is it possible to choose a

# group of some of the ints, such that the group sums

# to the given target, with this additional constraint:

# if there are numbers in the array that are adjacent

# and the identical value, they must either all be chosen,

# or none of them chosen. For example, with the array

# [1, 2, 2, 2, 5, 2], either all three 2's in the middle

# must be chosen or not, all as a group. (one loop can

# be used to find the extent of the identical values).

#def groupSumClump(start, nums, target):

# Given an array of ints, is it possible to divide the

# ints into two groups, so that the sums of the two

# groups are the same. Every int must be in one group

# or the other. Write a recursive helper method that

# takes whatever arguments you like, and make the

# initial call to your recursive helper from splitArray().

# (No loops needed.)

#def splitArray(nums):

#def splitArrayHelper():

# Given an array of ints, is it possible to divide the

# ints into two groups, so that the sum of one group

# is a multiple of 10, and the sum of the other group

# is odd. Every int must be in one group or the other.

# Write a recursive helper method that takes whatever

# arguments you like, and make the initial call to your

# recursive helper from splitOdd10(). (No loops needed.)

#def splitOdd10(nums):

#def splitOdd10Helper():

# Given an array of ints, is it possible to divide the ints

# into two groups, so that the sum of the two groups is the

# same, with these constraints: all the values that are

# multiple of 5 must be in one group, and all the values

# that are a multiple of 3 (and not a multiple of 5)

# must be in the other. (No loops needed.)

#def split53(nums):

#def split53Helper():

#######################################################################################################

#######################################################################################################

# #

# DO NOT MODIFY ANYTHING BELOW THIS LINE !! #

# #

#######################################################################################################

#######################################################################################################

def main(argv):

problems = ["groupSum", "groupSum6", "groupNoAdj", "groupSum5", "groupSumClump", "splitArray", "splitOdd10", "split53"]

if len(argv) == 0:

printHelp()

exit(1)

elif "all" in argv:

argv = problems

for problem in argv:

if not problem in problems:

printHelp()

exit(1)

groupSum\_args = [(0, [2, 4, 8], 10), (0, [2, 4, 8], 14), (0, [2, 4, 8], 9), (0, [2, 4, 8], 8), (1, [2, 4, 8], 8), (1, [2, 4, 8], 2), (0, [1], 1), (0, [9], 1), (1, [9], 0), (0, [], 0), (0, [10, 2, 2, 5], 17), (0, [10, 2, 2, 5], 15), (0, [10, 2, 2, 5], 9)]

groupSum6\_args = [(0, [5, 6, 2], 8), (0, [5, 6, 2], 9), (0, [5, 6, 2], 7), (0, [1], 1), (0, [9], 1), (0, [], 0), (0, [3, 2, 4, 6], 8), (0, [6, 2, 4, 3], 8), (0, [5, 2, 4, 6], 9), (0, [6, 2, 4, 5], 9), (0, [3, 2, 4, 6], 3), (0, [1, 6, 2, 6, 4], 12), (0, [1, 6, 2, 6, 4], 13), (0, [1, 6, 2, 6, 4], 4), (0, [1, 6, 2, 6, 4], 9), (0, [1, 6, 2, 6, 5], 14), (0, [1, 6, 2, 6, 5], 15), (0, [1, 6, 2, 6, 5], 16)]

groupNoAdj\_args = [(0, [2, 5, 10, 4], 12), (0, [2, 5, 10, 4], 14), (0, [2, 5, 10, 4], 7), (0, [2, 5, 10, 4, 2], 7), (0, [2, 5, 10, 4], 9), (0, [10, 2, 2, 3, 3], 15), (0, [10, 2, 2, 3, 3], 7), (0, [], 0), (0, [1], 1), (0, [9], 1), (0, [9], 0), (0, [5, 10, 4, 1], 11)]

groupSum5\_args = [(0, [2, 5, 10, 4], 19), (0, [2, 5, 10, 4], 17), (0, [2, 5, 10, 4], 12), (0, [2, 5, 4, 10], 12), (0, [3, 5, 1], 4), (0, [3, 5, 1], 5), (0, [1, 3, 5], 5), (0, [3, 5, 1], 9), (0, [2, 5, 10, 4], 7), (0, [2, 5, 10, 4], 15), (0, [2, 5, 10, 4], 11), (0, [1], 1), (0, [9], 1), (0, [9], 0), (0, [], 0)]

groupSumClump\_args = [(0, [2, 4, 8], 10), (0, [1, 2, 4, 8, 1], 14), (0, [2, 4, 4, 8], 14), (0, [8, 2, 2, 1], 9), (0, [8, 2, 2, 1], 11), (0, [1], 1), (0, [9], 1)]

splitArray\_args = [([2, 2]), ([2, 3]), ([5, 2, 3]), ([5, 2, 2]), ([1, 1, 1, 1, 1, 1]), ([1, 1, 1, 1, 1]), ([]), ([1]), ([3, 5]), ([5, 3, 2]), ([2,2,10,10,1,1]), ([1,2,2,10,10,1,1]), ([1,2,3,10,10,1,1])]

splitOdd10\_args = [[5, 5, 5], [5, 5, 6], [5, 5, 6, 1], [6, 5, 5, 1], [6, 5, 5, 1, 10], [6, 5, 5, 5, 1], [1], [], [10, 7, 5, 5], [10, 0, 5, 5], [10, 7, 5, 5, 2], [10, 7, 5, 5, 1]]

split53\_args = [[1,1], [1, 1, 1], [2, 4, 2], [2, 2, 2, 1], [3, 3, 5, 1], [3, 5, 8], [2, 4, 6], [3, 5, 6, 10, 3, 3]]

groupSum\_ans = [True, True, False, True, True, False, True, False, True, True, True, True, True]

groupSum6\_ans = [True, False, False, True, False, True, True, True, False, False, False, True, True, False, False, True, True, False]

groupNoAdj\_ans = [True, False, False, True, True, True, False, True, True, False, True, True]

groupSum5\_ans = [True, True, False, False, False, True, True, False, False, True, False, True, False, True, True]

groupSumClump\_ans = [True, True, False, True, False, True, False]

splitArray\_ans = [True, False, True, False, True, False, True, False, False, True, True, False, True]

splitOdd10\_ans = [True, False, True, True, True, False, True, False, True, False, True, False]

split53\_ans = [True, False, True, False, True, False, True, True]

for prob in argv:

correct = 0 # counts number of test cases passed

leftParen = "("

rightParen = ")"

# loop over test cases

for i in range(len(locals()[prob+"\_args"])):

if type(locals()[prob+"\_args"][i]) is tuple:

leftParen = rightParen = ""

if (type(locals()[prob+"\_args"][i]) is str) or (type(locals()[prob+"\_args"][i]) is int) or (type(locals()[prob+"\_args"][i]) is list) or (len(locals()[prob+"\_args"][i]) == 1): # function takes one argument

if globals()[prob](locals()[prob+"\_args"][i]) == locals()[prob+"\_ans"][i]:

print ("\nCorrect!", prob + leftParen + str(locals()[prob+"\_args"][i]) + rightParen + " result:", str(globals()[prob](locals()[prob+"\_args"][i])), " expected:", str(locals()[prob+"\_ans"][i]))

correct += 1

else: # print fail message

print ("\nWrong!", prob + leftParen + str(locals()[prob+"\_args"][i]) + rightParen + " result:", str(globals()[prob](locals()[prob+"\_args"][i])), " expected:", str(locals()[prob+"\_ans"][i]))

elif len(locals()[prob+"\_args"][i]) == 2: # there are two arguments to function

first, second = locals()[prob+"\_args"][i]

if globals()[prob](first, second) == locals()[prob+"\_ans"][i]:

print ("\nCorrect!", prob + leftParen + str(locals()[prob+"\_args"][i]) + rightParen + " result:", str(globals()[prob](first, second)), " expected:", str(locals()[prob+"\_ans"][i]))

correct += 1

else: # print fail message

print ("\nWorng!", prob + leftParen + str(locals()[prob+"\_args"][i]) + rightParen + " result:", str(globals()[prob](first, second)), " expected:", str(locals()[prob+"\_ans"][i]))

else:

first, second, third = locals()[prob+"\_args"][i]

if globals()[prob](first, second, third) == locals()[prob+"\_ans"][i]:

print ("\nCorrect!", prob + leftParen + str(locals()[prob+"\_args"][i]) + rightParen + " result:", str(globals()[prob](first, second, third)), " expected:", str(locals()[prob+"\_ans"][i]))

correct += 1

else: # print fail message

print ("\nWrong!", prob + leftParen + str(locals()[prob+"\_args"][i]) + rightParen + " result:", str(globals()[prob](first, second, third)), " expected:", str(locals()[prob+"\_ans"][i]))

print ("\n" + prob + ": passed", correct, "out of", len(locals()[prob+"\_ans"]), "\n")

def printHelp():

print ("\nInvoke this script with the name of the function you wish to test.")

print ("e.g. python recursion1.py factorial")

print ("Invoke with \"python recursion1.py all\" to run all of the function tests\n")

import sys

main(sys.argv[1:])