Recursion

Review: Recursion

A recursive function is one which makes a call to itself in its body, with a different input. That inner function call might then call itself again, with a third input. This repeats until no self-calls are made; this is known as a *base case* (without a base case, the function will call itself endlessly).

A recursive function on a list will often use a small list (length 0 or 1) as a base case, and the recursive case will often call itself with a list slice that excludes one element (but not always). For example, the following function computes the sum of a list of numbers:

```
def add(list) :
    if list == [] :
        return 0
    else :
        return list[0] + add(list[1:])
```

In the recursive case of this function, the `add` function calls itself with element 0 excluded (`list[1:]`). It then combines the result of the recursive call with the excluded element (`list[0]`).

You may find this YouTube video helpful in trying to understand recursion.

Minimums

Write a recursive function which takes a list of numbers and returns the smallest element in the list. If the list is empty, raise a `ValueError`.

Debugging Recursive Functions

The following function aims to find the sum of a list, but it does not always work. Determine why and modify the function definition to fix this error.

```
def add2(list) :
    if list == [] :
        return 0
    else :
        mid = len(list) / 2
        return add2(list[:mid]) + add2(list[mid:])
```

Family Trees

One of the common uses of recursive functions is with recursive data structures, like trees. Consider a family tree: a person can have children, and their children can have children, and their children can have children etc. This shows that this family tree is a recursive data structure and we will now write some recursive functions to take advantage of this structure.

For this task, use the provided Person class (`week11_person.py`) which models a family tree. Write 2 recursive methods for this class:

1. `num_descendants`: which takes no arguments and returns the total number of descendants that the person has (ie. number of children + number of grandchildren + number of great-grandchildren etc.)
2. `has_descendant`: which takes another Person object as a parameter and returns True if the provided Person object is a descendant of this Person and False otherwise (ie. if the person is a child, or grandchild etc.)

Files and Folders

In a computer's file system, directories (i.e. folders) can contain files, as well as other directories. This self-similarity (directories contain other directories) suggests that we should use recursive functions to traverse the file system.

For this task, write a function `list_files` which takes the name of a directory as input, and returns a list of all filenames that can be accessed from that directory (i.e. all the files in that directory, and its sub-directories, and its sub-sub-directories, and so on). Identify the base case of your function.

An example of using this function follows (obviously, the output will depend on what is in the directory you use as input; this is a fabricated example).

```
>>> list_files('H:/csse1001/')
['H:/csse1001/assign1.py',
 'H:/csse1001/assign1soln.py',
 'H:/csse1001/assign2.py',
 'H:/csse1001/lectures/recursion.py',
 'H:/csse1001/lectures/notes/add.py',
 'H:/csse1001/lectures/notes/search_tree.py',
 'H:/csse1001/tutorials/week2.py',
 'H:/csse1001/tutorials/week3.py',
 'H:/csse1001/tutorials/week11.py',
 'H:/csse1001/mypytutor/MyPyTutor.py',
 'H:/csse1001/mypytutor/MyPyTutor.pyw',
 'H:/csse1001/mypytutor/CSSE1001Answers/Introduction.py']
```

Hint:

To interact with files and folders, Python provides a library called `os`. The functions `os.listdir`, `os.path.join`, and `os.path.isdir` may be useful; see their `help` documentation, and the following examples.

```
>>> import os
>>> directory = os.getcwd()   # Get the name of the directory we are working
in.
>>> directory
'H:/csse1001'
>>> os.listdir(directory)
['assign1.py', 'assign1soln.py', 'assign2.py', 'lectures', 'tutorials',
'mypytutor']
```

```
>>> os.path.join(directory, 'assign1.py')
'H:/csse1001/assign1.py'
>>> os.path.isdir('H:/csse1001/assign1.py')
False
>>> os.path.isdir('H:/csse1001/lectures')
True
```

It may help to use the `list.extend` method; if `a` and `b` are lists, `a.extend(b)` will append every element from `b` into `a`:

```
>>> a = [1, 2]
>>> b = [3, 4]
>>> a.extend(b)
>>> a
[1, 2, 3, 4]
```

Tower of Hanoi

Tower of Hanoi is a simple game. There are 3 pegs and N number of disks. The disks start on one peg and are arranged in decreasing size from bottom to top. The objective of the game is to move the disks from the first peg to the last peg. The rules are that you cannot place a larger disk on top of a smaller disk, and that you can only move one disk at a time. See this YouTube video for details of the game and the algorithm to solve the game.

Write a recursive function that outputs all of the moves required to win the game. The function should have a parameter that indicates the number of disks. The function may need other parameters.

Challenge: Prove it!

1. See the `add` function defined in the Review section above. Use mathematical induction to prove that this function always returns the correct value (the sum of `list`).
2. Use strong mathematical induction to prove that your modified `add2` function always returns the correct value. Identify where the proof fails for the original `add2` function.