15-110: Principles of Computing, Spring 2018

Programming Assignment 5

Due: Tuesday, February 27 by 9PM

Note: You are **responsible for protecting your solutions** to the following problems from being seen by other students both physically (e.g., by looking over your shoulder or verbal discussion) and electronically. In particular, since the lab machines use the Andrew File System (AFS) to share files worldwide, you need to be careful that you do not put files in a place that is publicly accessible.

If you are doing the assignment on the Gates-Hillman Cluster machines we use in the lab or on `unix.andrew.cmu.edu`, please remember to have your solutions inside a `private` folder (which is under your home directory). Our recommendation is that you create a `pa5` folder under `~/private/15110` for this assignment. That is, the new directory `pa5` is inside the directory named `15110`, which is inside the `private` directory.

Overview

For this assignment, you will create a Python file for each of the problems below. You should save all of these files in a folder named `pa5`. Once you have every file, you should zip up the `pa5` folder and submit the zipped file on Autolab.

This assignment will make you practice with recursion and make you learn how to write the corresponding Python code to a given algorithm. You will also experiment with stacks and implement a simple RPN (Reverse Polish Notation) calculator.

Problems

1. [2 points] Computing a 2^n recursively.

   Recall that n! (n factorial) is defined recursively as follows for non-negative integer n:

   $n! = n \times (n-1)!$ for $n > 0$ and $n! = 1$ for $n = 0$

   Now consider computing 2^n recursively, $n \geq 0$.

   $2^n = 2 \times 2^{n-1}$ for $n > 0$ and $2^n = 1$ for $n = 0$.

In the file `pow2.py`, write a **recursive** function `pow2(n)` for n ≥ 0 that computes 2^n for any integer n greater than or equal to 0. **IMPORTANT: You may NOT use the ** operator or the `math.pow` function, you may not use a list holding powers of 2, and you may not use a loop. This function should be recursive, so your `pow2` function should have a call to `pow2` inside of it.**

Hint: Your function will look very similar to the factorial function discussed in class!

Example usage:

```
>> python3 -i pow2.py
>>> pow2(0)
1
>>> pow2(4)
16
>>> pow2(6)
64
>>> pow2(7)
128
>>> pow2(10)
1024
>>> pow2(20)
1048576
```

2. [2 points] Printing select strings from a list

In `show_strings.py`, define a Python function `show_strings(stringlist)` that takes a list of strings called *stringlist* and prints out each string that has at least 5 characters. Use the following recursive algorithm:

1. Return None if *stringlist* is empty.
2. Print the first string in *stringlist* if that first string has at least 5 characters.
3. Using a recursive call, show the strings that have at least 5 characters in the rest of *stringlist*. (See List Hint below.)

String Hint: Just like lists, strings have a `len` function.

List Hint: The rest of the list is the list containing everything except the first element of the original list, in the same order.

Example usage: [**PRINT STRINGS ONE PER LINE**]

```
python3 -i show_strings.py
>>> cars = ["buick", "gmc", "chevrolet", "cadillac", "ford",
"bmw", "kia", "lincoln"]
>>> show_strings(cars)
buick
chevrolet
cadillac
lincoln
>>> airports = ["jfk", "lga", "ewr", "isp"]
>>> show_strings(airports)    # no output for this one
>>> teams = ["penguins", "steelers", "pirates"]
>>> show_strings(teams)
penguins
steelers
pirates
```

3. [2 points] Towers of Hanoi Revisited

Recall the solution to the Towers of Hanoi problem given in class, which prints out all of the instructions for the required moves for a given number of discs, n.

We can write a function that computes the total number of moves made in the problem, recursively. Complete the following function, so that it returns the total number of moves made. Assume that n is always greater than or equal to 0. Store your function in `towers.py`. **Your solution must be recursive. Do not use exponentiation (** or math.pow).**

```
def towers(n):
# returns the total number of moves required for
# Towers of Hanoi with n discs, n >= 0
    if n == 0:
        return _________________________
    else:
        # THINK RECURSIVELY!
        # task 1: move n-1 discs from starting peg to extra peg
        nummoves1 = ____________________________________
        # task 2: move 1 disc from starting peg to ending peg
        nummoves2 = ____________________________________
        # task 3: move n-1 discs from extra peg to ending peg
        nummoves3 = ____________________________________
        totalmoves = nummoves1 + nummoves2 + nummoves3
        return totalmoves
```

Example Usage:

```
>> python3 -i towers.py
>>> towers(0)
0
>>> towers(1)
1
>>> towers(4)
15
>>> towers(5)
31
>>> towers(15)
32767
```

4. [2 points] Reversing lists

In `reverse.py`, you will write a Python function `reverse(datalist)` that takes a list *datalist* as its parameter and returns a list with the same data in reverse order.

Write `reverse(datalist)` as a recursive function. Use this algorithm:

I. If *datalist* is empty, then return the appropriate result.
(If a list is empty, what is its reverse?)
II. Otherwise, do the following:
   A. Store the first element from *datalist* in *temp* and then remove it from *datalist*.
   B. Initialize *newlist* to be the reversal of the current *datalist* using recursion. (Hint: this means you need to call this same `reverse` function on this step.)
   C. Append on to *newlist* the original first element from *temp*.
   D. Return *newlist* as the resulting reversed list.

Do not use a built-in reverse function for lists. You should implement the algorithm as specified above.

Example Usage:

```
>> python3 -i reverse.py
>>> reverse([])
[]
>>> reverse([1,2,3,4,5])
[5, 4, 3, 2, 1]
>>> reverse([15, 110])
[110, 15]
>>> reverse([2, 3, 5, 8, 13, 21, 34, 55])
[55, 34, 21, 13, 8, 5, 3, 2]
```

5. [2 points] RPN (Reverse Polish Notation) Calculator

In Python, you can test whether a value is an integer (`int`) using the `isinstance` function. Here's an example:

```
>>> a = 4
>>> b = "+"
>>> isinstance(a, int)
True
>>> isinstance(b, int)
False
```

In this problem, we will implement the algorithm for an RPN Calculator that was given in class. **This algorithm is NOT recursive.** The algorithm starts with a list that contains a valid RPN expression consisting of integers and operators (+, -, *, /). For example:

```
[ 23, 3, "-", 4, 6, "+", "/"]
```

To compute its value, follow this algorithm for the valid RPN *expression*:

I. Set stack equal to the empty list.
II. For each element *x* in *expression*, do the following:
    A. If *x* is an integer, then push (append) x on to the *stack*.
    B. Otherwise it must be an operator, so do the following:
        1. Pop the last element from stack and store it in b.
        2. Pop the last element from stack and store it in a.
        3. If *x* is "+", push (append) a+b on to the stack.
        4. If *x* is "-", push (append) a-b on to the stack.
        5. If *x* is "*", push (append) a*b on to the stack.
        6. If *x* is "/", push (append) a//b on to the stack. (use integer division).
III. Once all of the elements of the expression are processed, there should be one value left in the stack, so pop it and return it as the final answer.

In `rpn.py`, write a function `rpn(expression)` that returns the integer value of the valid RPN *expression*, represented as a list of integers and operators.

Example Usage:

```
python3 -i rpn.py
>>> expression1 = [23, 3, "-", 4, 6, "+", "/"]
>>> rpn(expression1)
2
>>> expression2 = [3, 4, "+", 5, "*"]
>>> rpn(expression2)
35
```

Submission

You should now have the `pa5` folder that contains the following five Python files:

a. `pow2.py`
b. `show_strings.py`
c. `towers.py`
d. `reverse.py`
e. `rpn.py`

Zip up the folder and submit the zipped file named as `pa5.zip` on Autolab.
Be sure to check your submission to see that you submitted the correct code.

NOTE: Although Autolab will score your results based on the output your functions produce, your TAs will review your code to make sure you used the correct algorithms and used recursion correctly and may deduct points if you do not follow the given instructions.