Due Sunday Jan 30 11:59 PM EST

CSE 4256

Dr. Alan Weide

Spring Term 2022

Homework 2

Assignment

This assignment should be completed in a single Python source (`.py`) file.

(1) List your answers to the "Comprehension Check: Slices" slide from the second lecture (slide #12). As a reminder, here are the questions, where `s = "PYTHON"`:

  (a) Write two equivalent expressions to retrieve the character `"N"` from `s`.

  (b) What is the value of the expression `s[0:6]`?

  (c) What is the value of the expression `s[:3]`?

  (d) What is the value of the expression `s[::2]`?

  (e) Write a slice of `s` that is equal to `"PYTH"`.

  (f) Write a slice of `s` that is equal to `"HN"`.

  (g) Write a slice of `s` that is equal to `"NHY"`.

(2) Write a function called `fiblist(n)` that, given a positive integer `n`, produces a list containing the first `n` terms of the Fibonacci Sequence. You may assume that `n` is a positive integer.

(3) Write a function called `ispartitionable(s)` that, given a list of integers, return `True` if the list can be partitioned into two contiguous slices such that the sum of the elements of one slice is equal to the sum of the elements of the other slice, `False` otherwise. You may assume the list `s` contains only integers. Formally, the function should return `True` if the following holds, and `False` otherwise:

$$\exists i : \sum_{j=0}^{i-1} \texttt{s[j]} = \sum_{k=i}^{\texttt{len(s)}-1} \texttt{s[k]}$$

Hint: the built-in function **`sum`**`(s)` is probably useful here.

(4) The list comprehension given on slide #15 of the second lecture to generate the n-row identity matrix uses a *nested list comprehension*. Write an alternate comprehension that is *not* nested to produce the same result, and store that list in a variable called `m`. **Hint**: the sequence repetition ($\star$) and concatenation (+) operators will come in handy.

(5) Write a list comprehension that produces a list containing the sums of the rows in a matrix `m`, and store that list in a variable named `sums`. You may assume `m` is previously defined.

(6) Write a function called `vowelcount(s)` that, given a string, returns the number of characters in the string that are English vowels (*i.e.*, one of the characters 'a', 'e', 'i', 'o', or 'u'). The function should be case-agnostic meaning that, *e.g.*, the character 'A' counts as a vowel.

(7) Write a function called `listfromcsv(s)` that, given a string containing several lines of comma-separated values (*e.g.*, `s` may be `"5,8,hello,2\n9,14,world,1344"`), produces a (2-dimensional) list of those values separated by line (*e.g.*, for the example given, the result should be the list `[['5', '8', 'hello', '2'], ['9', '14', 'world', '1344']]`). **Hint**: the string functions **`str`**`.split(sep)` and **`str`**`.splitlines()` will probably come in handy. This function can be written in one line with a list comprehension.

Challenge Activities

Some homeworks (such as this one) will have additional challenge activities. These activities **do not contribute to your grade**, but they are problems that I find interesting or challenging.

(1) Write a function called `partition(s)` that, given a "partitionable" list of integers, returns a tuple consisting of two subsequences of `s` that have identical sums. You may assume the function `ispartitionable(s)` returns `True`. **Hint**: using the **`enumerate`**`(s)` function keeps track of both the index and item of `s`.

(2) Write a function called `triangles(n)` that, given a positive integer `n`, produces a list of the first `n` triangle numbers. You may assume that `n` is a positive integer.

(3) Write a function called `ctriangles(n)` that produces a list identical to `triangles(n)`, but does so using a single-line list comprehension. **Hint**: the built-in function **`sum`**`(s)` is probably useful here.

(4) Think about the relative efficiency of `triangles(n)` and `ctriangles(n)`. How do they compare? Which is faster? Why?

(5) Write a function called `pascal(r)` that, given a positive integer `r`, produces a list containing the first `r` rows of Pascal's Triangle (more precisely, the function should only generate all *non-zero* entries in the triangle). You may assume that `r` is a positive integer.

Recall: The kth entry in the nth row of Pascal's Triangle has the value $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. It has the nice property that $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$.

(6) If you used any modules to write `pascal(r)`, write a function called `pascal2(r)` to generate the first `r` rows of Pascal's Triangle *without* using any additional modules. Alternatively, if your previous solution did *not* use any additional modules, write a function called `pascal2(r)` that takes advantage of the `math` module (useful functions may include `math.factorial(n)` to compute $n!$ or `math.comb(n, k)` to compute $\binom{n}{k}$) to generate the first `r` rows of Pascal's Triangle.[1]

(7) Write a function called `csvfromlist(s)` that takes as input a 2-dimensional list and generates a string with values across rows seprated by `','` and rows separated by `'\n'`. In essence, `csvfromlist(s)` should "undo" the work that `listfromcsv(s)` did.

Submission

To submit this assignment, upload your `.py` file containing all of your code to the "Homework 2" assignment on Carmen. As always, be sure to note all group members who contributed to the assignment and what those contributions were.

[1]Obviously, `pascal2(r)` should not simply make a call to `pascal(r)`. The two functions should use different algorithms altogether.