Advanced Python: Homework set 4

2023/2024

There are two types of tasks in the list below. Please select one task from each each group and program them. Each of these tasks is worth 4 points.

1 Lists

Below are tasks involving the implementation of functions that return lists of natural numbers that meet appropriate conditions. Each task must be performed in three versions: an imperative version, a comprehension version and a functional version:

1. in the imperative version we use while, for in etc. statements. and completing the resulting list with the append method;
2. The comprehension version should be in the form of one list comprehensiom or nested comprehension lists. In the case of nesting, you can separate sublists, e.g.:

```
def given_function(n):
    list_tymcz =[ list folded]
return [ fold_list containing temp_list ]

def given_function(n):
    tim_list =[ comprehension ]
    return [ fold_list containing temp_list ]
```

3. The functional implementation should use functions dedicated to operations on lists (or list generators): **filter**, **range**, **sum** or **reduce**. I would like to emphasize here that the function is supposed to ultimately return a list, not a generator.

Using the timeit module, check the operating time of individual functions for various data. Format the time measurements in the form of a readable table like:

n	comprehension	imperative
10:	0.018	0.008
20:	0.042	0.016
30:	0.074	0.024
40:	0.111	0.032
50:	0.155	0.040
60:	0.204	0.048
70:	0.261	0.057
80:	0.326	0.065
90:	0.394	0.073

Problem 1

Program unary functions prime_imperative(n), prime_comprehension(n), and prime_functional(n) that return a list of prime numbers no greater than n, for example

```
    >>> prime(20)
[2, 3, 5, 7, 11, 13, 17, 19]
```

Problem 2

Program unary functions `perfect_imperative(n)`, `perfect_comprehension(n)`, and `perfect_functional(n)` that return a list of perfect numbers no larger than n, for example

```
    >>> perfect(10000)
[6, 28, 496, 8128]
```

Problem 3

Program unary functions `imperative_distribution(n)`, `comp_distribution(n)` and `functional_distribution(n)` that compute the prime factorization of n and return a list of pairs $[(p_1, w_1), (p_2, w_2), \ldots, (p_k, w_k)]$ such that $n = p_1^{w_1} * p_2^{w_2} * \ldots * p_k^{w_k}$ and $p_1, \ldots, p_k$ are different prime numbers. For example

```
    >>> distribution(756)
[(2, 2), (3, 3), (7, 1)]
```

Since this task may require a list of prime numbers, you can implement a helper function that checks the primality of a number or returns a list of prime numbers. For this helper function, the implementation can be anything you like.

Problem 4

Program the unary functions `amicable_imperative(n)`, `amicable_comprehension(n)`, and `amicable_functional(n)` that return a list of pairs of friendly numbers no larger than n, for example

```
    >>> zaprzyjaznione(1300)
[(220, 284), (1184, 1210)]
```

Appropriate definitions can be found, for example in Wikipedia. Choose two of the given tasks. Each task is worth 4 points.

2 Problem solving

Program a solution search based on checking all potential solutions (brute force). A function solving a task should return an iterator so that all found solutions can be listed using the for-in statement:

```
for solution in problem_solving(input):
    print(solution)
for solution in problem_solving(input):
    print (solution)
```

Take care not to generate lists unnecessarily (e.g. with permutations). You can assume that the input data is always correct and does not need to be checked further.

Problem 5

Cryptarithm is a task in which letters must be replaced with numbers to produce the correct function. An example of such a cryptorhythm is

$$\begin{array}{r} \text{KYOTO} \\ +\ \text{OSAKA} \\ \hline \text{TOKYO} \end{array}$$

Write a program to solve such cryptorithms. Assume that the input contains three words and an operator.

Problem 6

The following task involves reconstructing a two-dimensional image based on a cast shadow. We assume that the image is a rectangle of black and white pixels. Shadow are two vectors that describe how many blackened pixels there are in a row or column. On the Polish version of the list there is an example whose shadow is described by two vectors: $H = (2, 1, 3, 1), V = (1, 3, 1, 2)$. There can be many different images for a given shadow.

Problem 7

In the popular Sudoku puzzle, the task is to fill the 9×9 diagram with digits from 1 to 9 so that no digit is repeated in each row and each column. Additionally, no digits may be repeated in each 3×3 subsquare. Below is an example of a correctly completed diagram:

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Program the `sudoku_solution(s)` function which, for a partially completed s diagram, returns its correct filling (or `None` if there is no solution). The diagram representation is arbitrary.

Also program a function that will display a clear diagram.