October 22, 2024

Python Advanced Course

List 3.

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.

Task group "Letters"

Below are tasks involving the implementation of functions that return lists of numbers natural ones that meet the appropriate conditions. Each task should be performed in three versions: imperative version, foldable list version and functional version:

- in the imperative version we use the while, for in etc. instructions and supplement the resulting list with the append method;
- The folded list version should be in the form of one folded list or nested list comprehensions. In the case of nesting, you can extract sublists e.g. like this:

```
def given_function(n):
        list_temp = [complex list]
        return [foldable_list_containing_temp_list ]
```

- The functional implementation should use functions dedicated to operations on lists (or list generators): filter, range, sum or reduce.
  Here I emphasize that the function should ultimately return a list, not a generator.

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

n	foldable	imperative	functional
10:	0.018	0.07	0.008
20:	0.042	0.016	0.12
30:	0.074	0.024	0.43
40:	0.111	0.032	0.34
50:	0.155	0.040	0.15
...			

Task 1.

Program single-argument functions prime_imperative(n), prime_composite(n) and prime_function(n), which return a list of prime numbers no greater than n, for example

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

Task 2.

Program single-argument functions perfect_imperative(n), perfect_composite(n), and perfect_functional(n) that return a list of perfect numbers no greater than n, for example

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

Task 3.

Program the unary functions imperative_factor(n), compound_factor(n), and functional_factor(n) that compute the prime factorization of n and return a list of pairs [(p1, w1),(p2, w2), . . . ,(pk, wk)] such that and p1, . . . , pk are distinct primes. For example

$n = p_1^{w1}$ ÿp $_2^{w2}$ ÿ. . .ÿp $_k^{wk}$

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

Since this task may require a list of primes, you can implement an auxiliary function that checks the primality of a number or returns a list of primes. In the case of this auxiliary function, the implementation can be anything.

Task 4.

Program single-argument functions friendly_imperative(n), friendly_foldable(n), and friendly_function(n) that return a list of pairs of friendly numbers no larger than n, for example

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

Appropriate definitions can be found, for example, in the Polish Wikipedia.

Task group "Puzzles"

Program a search for solutions to puzzles based on checking all potential solutions (brute force strategy). The function solving the problem should return an iterator, so that all found solutions can be printed using the for-in statement:

```
for solution in task_solving(input_data): display_solution(solution)
```

Additionally, for each task, program a program that clearly displays the solution to the puzzle.

It is important that the implementation:

1. find all solutions (even if it takes a long time); 2. use generators instead of creating lists of potential solutions; ta-
   The lists will be very long, which we want to avoid.

It can be assumed that input data is always correct and does not need additional checking.

Task 5.

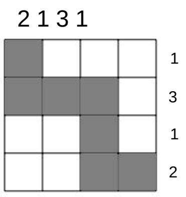
A cryptarithm is a task in which letters must be replaced by numbers so that the correct operation is created. An example of such a cryptarithm is

```
   KYOTO
+ OSAKA
-------
   TOKYO
```

Write a program that solves such cryptarithms. Assume that the input data consists of three words and an operator. We can limit ourselves to the basic four arithmetic operators.

Task 6.

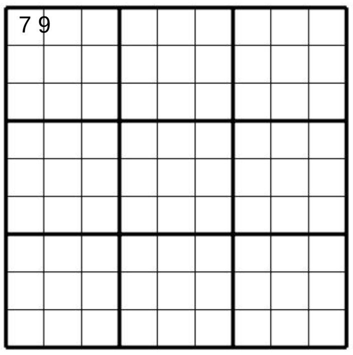
The following task involves reconstructing a two-dimensional image based on a cast shadow. We assume that the image is a rectangle of black and white pixels.
Shadow is two vectors describing how many black pixels there are in a row or column. Below is an example of a 4 × 4 image:



whose shadow is described by two vectors: H = (2, 1, 3, 1), V = (1, 3, 1, 2). For a given shadow, there can be many different images. Program a function that finds all possible reconstructions of the image.

Task 7.

In the popular Sudoku puzzle, the task is to fill a 9 × 9 grid with the numbers 1 through 9 so that no number is repeated in each row and column. Additionally, no number can be repeated in each 3 × 3 subsquare. Below is an example of a correctly filled out 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



Program a function that, for a partially filled diagram, finds all valid fills.

Marcin Mÿotkowski