Third Collection of Python Problems

Ilkka Kokkarinen

Chang School of Continuing Education
Toronto Metropolitan University

Version of August 17, 2024

This document contains the specifications for a third collection of Python problems created by Ilkka Kokkarinen starting from March 2023, to augment the original set of 109 Python Problems designed for the course **CCPS 109 Computer Science I** for Chang School of Continuing Education, Toronto Metropolitan University, Canada (formerly Ryerson University). .

Same as in the original problem collection, these problems are listed roughly in their order of increasing difficulty. The complexity of the solutions for these additional problems ranges from straightforward loops up to convoluted branching recursive backtracking searches that require all sorts of clever optimizations to prune the branches of the search sufficiently to keep the total running time of the test suite within the twenty second time limit.

The rules for solving and testing these problems are exactly the same as they were for the original 109 problems. You must implement all your functions in a single source code file that must be named `labs109.py`, the exact same way as you wrote your solutions to the original 109 problems. The automated test suite to check that your solutions are correct, along with all associated files, is available in the GitHub repository `ikokkari/PythonProblems`.

Python coders of all levels will hopefully find something interesting in this collection. The author believes that there is room on Earth for all races to live, prosper and get strong at coding while having a good time by solving these problems. Same as with the original 109 problems, if some problem doesn't spark joy to you, please don't get stuck with that problem, but just skip it and move on to the next problem that interests you. Life is too short to be wasted on solving stupid problems.

Table of Contents

.

Multiply and sort

```
def multiply_and_sort(n, multiplier):
```

The following cute problem comes from the post "Double (or triple) and sort" in Michael Lugo's blog "God Plays Dice". Starting from the given positive integer `n`, multiply the current number by the given `multiplier`, and sort the digits of the result in ascending order to get the next number in the sequence. Note that this operation can make the number smaller, since all zero digits will effectively disappear from the front. For example, given `n` of 513 and `multiplier` of 2, the product 1026 with its digits sorted becomes 0126, which equals 126. Rinse and repeat until you come to an integer that you have seen before, and return the largest integer that you encountered along the way in this sequence.

`n`	`multiplier`	Expected result
`2`	`5`	`4456`
`3`	`4`	`2667`
`5`	`5`	`4456`
`20`	`7`	`15556688`
`31`	`7`	`12345678888888888889`

As one commenter of the original post pointed out, this process seems to always terminate up to multipliers up to 20, but then freezes for some multipliers from 21 onwards and behaves well for the others. Interested readers might want to further explore this behaviour and prove some necessary and sufficient conditions for termination. The automated tester will, of course, give your function only numbers for which this sequence terminates in a reasonable time.

Split at None

```
def split_at_none(items):
```

The original Fizzbuzz problem to quickly screen out the utterly hopeless candidates for programming jobs will soon be almost two decades old and too well known, so our present time requires more sophisticated tests for this purpose. Modern problems require modern solutions, after all.

A recent tweet by Hasen Judi offers another interview screening problem that is more fitting the power of modern scripting languages, yet requires no special techniques past the level of basic imperative programming taught in any introductory programming course. Given a list of `items`, some of which equal `None`, return a list that contains as its elements the sublists of consecutive `items` split around the `None` elements serving as separator marks, each consecutive block of elements a separate list in the result. Note the correct expected behaviour whenever these `None` separators are located at the beginning and the end of the list, and the correct expected behaviour whenever multiple `None` values are consecutive `items`.

`items`	Expected result
`[-4, 8, None, 5]`	`[[-4, 8], [5]]`
`[None, None, 12, None, 3]`	`[[], [], [12], [3]]`
`[None, None, None, None, 5, None]`	`[[], [], [], [], [5], []]`
`[None, -36, 25, 28, None, -27, -24, 34, -28, 24, 44, 6, 33, 20, None]`	`[[], [-36, 25, 28], [-27, -24, 34, -28, 24, 44, 6, 33, 20], []]`
`[53, 20, -49, 34, None, 13, -43, -14, 53, -6, None, None, None, -26, None, 14, None, None, -11]`	`[[53, 20, -49, 34], [13, -43, -14, 53, -6], [], [], [-26], [14], [], [-11]]`

(Cue the haughty reply tweets of “I have no idea how to solve this totes stupid and meaningless problem that has nothing to do with real work! It's just that those stupid employers don't see what a smart and productive programmer I am in practice, but I just always panic and freeze when I am given simple tests” in three, two, one...)