The Art+Logic Programming Challenge

Thanks for your interest in Art+Logic.

This file contains the instructions for the second part of the Art+Logic Programming Challenge. Since you have this file, you should have already received the part 1 document and successfully submitted an application. If you've gotten this document otherwise, please head to our careers page at https://www.artandlogic.com/careers to get started.

Part 2: Weird Text Format-8, Continued

Once again, the scramble/descramble matrix for the protocol looks like this:

Decoded/Raw																															
X	W	V	U	T	S	R	Q		P	N	M	L	K	J	H	G		F	E	D	C	B	A	9	8		7	6	5	4	3	2	1	0
X	P	F	7	W	N	E	6		V	M	D	5	U	L	C	4		T	K	B	3	S	J	A	2		R	H	9	1	Q	G	8	0
Encoded																																	

Refer to the part 1 document for details and examples of the encode/decode process for individual chunks of data.

The task for part 2 of this challenge is to create a small application that is able to:

- convert a string of arbitrary length into the corresponding list of integer values
- convert a list of integer values back into the original string

For any valid string input `s` and your two functions named `encode` and `decode`, we expect to be able to successfully assert `s == decode(encode(s))`.

Your code to perform the encoding/decoding of data should be structured as if it were in a library, not tightly coupled to your demo application. Assume that our reviewers will want to import your code and/or call your code from a pre-existing test harness.

This program should also offer a simple interface. Examples include a command line application that reads from stdin or a file and writes to stdout or a file, or a web page or mobile/desktop app that uses a text edit box to accept input values by typing or pasting (or loading a file) and a button to convert that input into the desired output format.

Your goal here isn't to write code that does the bare minimum of work to produce a correct answer; we're looking to get a taste of what your habits and instincts are for writing production-quality code.

At Art+Logic, production-quality code is expected to include useful tests, documentation that includes at least an explanation of what the project is attempting to accomplish and how to use it, appropriate commenting throughout the code and consistent style.

Example Data

String: `tacocat`
Encoded: `[267487694, 125043731]`

String: `never odd or even`
Encoded: `[267657050, 233917524, 234374596, 250875466, 17830160]`

String: `lager, sir, is regal`
Encoded: `[267394382, 167322264, 66212897, 200937635, 267422503]`

String: `go hang a salami, I'm a lasagna hog`
Encoded: `[200319795, 133178981, 234094669, 267441422, 78666124, 99619077, 267653454, 133178165, 124794470]`

String: `egad, a base tone denotes a bad age`
Encoded: `[267389735, 82841860, 267651166, 250793668, 233835785, 267665210, 99680277, 133170194, 124782119]`

Guidelines, Requirements, and Other Constraints

Your program must be written using one of our preferred programming languages:

- C++
- C#
- JavaScript
- Java
- Objective-C
- Python[1]
- Swift
- Kotlin

[1] We are only accepting Python solutions targeting Python 3.x.

Avoid the use of any third-party libraries or frameworks where possible—remember that the goal here is to help us see how awesome your code can be when you're working on a problem where none of those libraries exist yet.

Another common mistake is to apply 20% of your effort to solving the problem and the rest on building the shell that drives that code. A flashy application won't compensate for a poorly-considered solution to the task at hand.

In production, we also require that code written for Art+Logic conform to the conventions outlined in the Art+Logic Programming Style Guide (at https://styleguide.artandlogic.com). For the purposes of the challenge, we're not expecting *perfect* adherence to our house style, but we're looking for a good-faith effort to write in A+L style.

Your completed submission will be evaluated by a senior Art+Logic developer, who will be looking for several things, but most importantly:

- Does this code work correctly?
- Is it well designed and cleanly implemented?
- Was it completed in a reasonable amount of time?
- If I had to maintain this code, would that make me happy or angry?
- How closely does it follow our Style Guide?
- Did the applicant read the instructions fully?

Submitting Your Solution

Please package your source and any other files accompanying your submission into a ZIP file named after you, formatted as `{firstName}_{lastName}_Part2.zip` and upload it using the form at the URL contained in the email that this challenge document was attached to.

Please do not include any executable files inside your submission.

In the "Notes" field on the submission form, please add the following information:

Time taken to complete part 2: *(in hours)*

Language(s) used in submission:

File version 2023-05-01