DS 5010 Homework 3

Instructions

- Submit your solutions on Canvas by the deadline displayed online.
- Your submission must include a single Python module (file with extension ".py") that includes all of the code necessary to answer the problems. All of your code should run without error.
- Problem numbers must be clearly marked with code comments. Problems must appear in order, but later problems may use functions defined in earlier problems.
- Functions must be documented with a docstring describing at least (1) the function's purpose, (2) any and all parameters, (3) the return value. Code should be commented such that its function is clear.
- All solutions to the given problems must be your own work. If you use third-party code for ancillary tasks, you **must** cite them. (You may use code from class notes without citation.)
- You may use functions from built-in modules (e.g., `math`). You may **NOT** use external modules (e.g., `numpy`, `pandas`, etc.).

In this assignment, you will implement functions, a class, and methods for run-length encoding (RLE), which is a strategy for lossless data compression. RLE compresses "runs" of data, where a "run" is when the same data value occurs multiple times consecutively. The data value for each "run" is stored together with the length of the run. RLE results in efficient data compression when the lengths of the runs are long relative to the total number of runs. It is used widely in areas such as genomics, where the sequences of nucleotides in DNA can be efficiently compressed using RLE. For example, "GGGCCTTTTA" has run values "GCTA" and run lengths (3, 2, 4, 1).

Problem 1 Define a function `rle_encode(x)` satisfying the following criteria:

- Encodes an input iterable `x` using run-length encoding (RLE)
- You may assume `x` is a sequence of numbers or strings/characters.
- Returns a tuple with a `list` of the run values and a `list` of run lengths

Examples:

```
In : rle_encode("aaabbccccaa")
Out: (['a', 'b', 'c', 'a'], [3, 2, 4, 2])

In : rle_encode([1, 1, 1, 2, 2, 0, 2, 2, 2, 2])
Out: ([1, 2, 0, 2], [3, 2, 1, 4])

In : rle_encode(["yes", "yes", "no", "no", "no", "yes"])
Out: (['yes', 'no', 'yes'], [2, 3, 1])
```

Problem 2 Define a function `rle_decode(values, lengths)` satisfying the following criteria:

- Decodes a `list` of run values and a `list` of run lengths
- You may assume the inputs are properly encoded using RLE
- Returns a `list` of the decoded values (always returns a list)

Examples:

```
In : rle_decode(['a', 'b'], [2, 3])
Out: ['a', 'a', 'b', 'b', 'b']

In : rle_decode([1, 2, 0, 2], [3, 2, 1, 4])
Out: [1, 1, 1, 2, 2, 0, 2, 2, 2, 2]

In : rle_decode(['yes', 'no', 'yes'], [2, 3, 1])
Out: ['yes', 'yes', 'no', 'no', 'no', 'yes']
```

Problem 3 Define a class called `Rle` with a method `Rle.__init__(self, values, lengths = None)` satisfying the following criteria:

- An `Rle` instance contains a run-length encoded object
- An `Rle` instance has `list` attributes called `values` and `lengths`
- If `__init__()` parameter `lengths` is `None`, then encode `values` using RLE
- Otherwise, initialize the attributes from the parameters

Examples:

```
In : x = Rle(["hi", "hi", "hi", "lo", "lo", "hi", "lo", "lo", "lo"])

In : x.values
Out: ['hi', 'lo', 'hi', 'lo']

In : x.lengths
Out: [3, 2, 1, 3]

In : y = Rle(["no", "yes", "no"], [3, 3, 1])

In : y.values
Out: ['no', 'yes', 'no']

In : y.lengths
Out: [3, 3, 1]
```

Problem 4 Define the method `Rle.__getitem__(self, i)` satisfying the following criteria:

- Returns the item at offset `i` of the decoded sequence
- Should work for both positive and negative indices
- You do NOT need to implement slicing

Examples:

```
In : y = Rle(["no", "yes", "no"], [3, 3, 1])

In : y[0]
Out: 'no'

In : y[3]
Out: 'yes'

In : y[-1]
Out: 'no'
```

Problem 5 Define the method `Rle.append(self, value)` satisfying the following criteria:

- Appends a new item `value` to the run-length encoded object

- Mutates the `self` instance and returns `None`

Examples:

```
In : y = Rle(["no", "yes", "no"], [3, 3, 1])

In : y.append("yes")

In : y.values
Out: ['no', 'yes', 'no', 'yes']

In : y.lengths
Out: [3, 3, 1, 1]

In : y.append("yes")

In : y.values
Out: ['no', 'yes', 'no', 'yes']

In : y.lengths
Out: [3, 3, 1, 2]
```