Project 4 (Guitar Sound Synthesis)

Clarifications and Hints

Prologue

Project goal: write a program to simulate the plucking of a guitar string using the *Karplus-Strong* algorithm

The zip file (`https://www.cs.umb.edu/~msolah/cs110_s18/projects/project4.zip`) for the project contains

- project specification (`project4.pdf`)
- starter files (`ring_buffer.py`, `guitar_string.py`)
- test script (`run_tests.py`)
- visualization client (`guitar_sound_synthesis.py`)
- report template (`report.txt`)

This checklist will help only if you have read the writeup for the project and have a general understanding of the problems involved. So, please read the project writeup before you continue with this checklist.

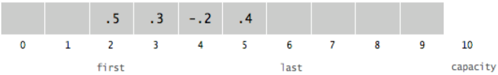
Problems

Problem 1 (*Ring Buffer*) Write a module `ring_buffer.py` that implements the following API:

function	description
`create(capacity)`	create and return a ring buffer, with the given maximum capacity and with all elements initialized to `None`
`capacity(rb)`	capacity of the buffer rb
`size(rb)`	number of items currently in the buffer rb
`is_empty(rb)`	is the buffer rb empty?
`is_full(rb)`	is the buffer rb? full?
`enqueue(rb, x)`	add item x to the end of the buffer rb
`dequeue()`	delete and return item from the front of the buffer rb
`peek(rb)`	return (but do not delete) item from the front of the buffer rb

Hints

- We represent a ring buffer as a four-element list: the first element (`buff`) is a list of floats of a given capacity; the second element (`size`) is the number of items in `buff`, ie, its size; the third element (`first`) stores the index of the item that was least recently inserted into `buff`; and the fourth element (`last`) stores the index one beyond the most recently inserted item — for example, the following ring buffer



is represented as the list `[[•, •, 0.5, 0.3, -0.2, 0.4, •, •, •, •], 4, 2, 6]`

Problems

- `create(capacity)`
    - Create and return a ring buffer with `buff` having the given capacity and all of its items set to `None`, and with `size`, `first`, and `last` initialized to 0
- `capacity(rb)`
    - Return the capacity of the given ring buffer
- `size(rb)`
    - Return the size of the given ring buffer
- `is_empty(rb)`
    - Return `True` if the given ring buffer is empty (ie, its size is 0), and `False` otherwise
- `is_full(rb)`
    - Return `True` if the given ring buffer is full (ie, its size equals its capacity), and `False` otherwise
- `enqueue(rb, x)`
    - Store `x` at `buff[last]` in the given ring buffer
    - If `last + 1` equals capacity, set `last` to 0; otherwise, increment it by 1
    - Increment `size` by 1

Problems

- dequeue()
  - Assign the item buff[first] in the given ring buffer to some variable v
  - If first + 1 equals capacity, set first to 0; otherwise, increment it by 1
  - Decrement size by 1
  - Return v
- peek(rb)
  - Return the item buff[first] in the given ring buffer
- Example (rb for ring buffer, b for buff, s for size, f for first, and l for last)

```
                              b[0]  b[1]  b[2]  b[3]  b[4]   s  f  l

create(5)               rb = [[None, None, None, None, None], 0, 0, 0]
enqueue(rb, ’A’)        rb = [[ ’A’, None, None, None, None], 1, 0, 1]
enqueue(rb, ’B’)        rb = [[ ’A’,  ’B’, None, None, None], 2, 0, 2]
enqueue(rb, ’C’)        rb = [[ ’A’,  ’B’,  ’C’, None, None], 3, 0, 3]
dequeue(rb)             rb = [[ ’A’,  ’B’,  ’C’, None, None], 2, 1, 3] # ’A’
enqueue(rb, ’D’)        rb = [[ ’A’,  ’B’,  ’C’,  ’D’, None], 3, 1, 4]
enqueue(rb, ’E’)        rb = [[ ’A’,  ’B’,  ’C’,  ’D’,  ’E’], 4, 1, 0]
enqueue(rb, ’F’)        rb = [[ ’F’,  ’B’,  ’C’,  ’D’,  ’E’], 5, 1, 1]
peek(rb)                rb = [[ ’F’,  ’B’,  ’C’,  ’D’,  ’E’], 5, 1, 1] # ’B’
enqueue(rb, ’G’)        Error: buffer full!
```

Problems

Problem 2 (*Guitar String*) reate a module `guitar_string.py` to model a vibrating guitar string. The module must implement the following API:

function	description
`create(frequency)`	create and return a guitar string of the given frequency, using a sampling rate given by SPS, a constant in `guitar_string.py`
`create_from_samples(init)`	create and return a guitar string whose size and initial values are given by the list $init$
`pluck(string)`	pluck the given guitar string by replacing the buffer with white noise
`tic(string)`	advance the simulation one time step on the given guitar string by applying the Karplus-Strong update
`sample(string)`	current sample from the given guitar string

Hints

- We represent a guitar string as a ring buffer[1]
- `create(frequency)`
    - Create and return a ring buffer with capacity calculated as the sampling rate (`SPS`) divided by the given frequency and rounded up to the nearest integer, and all values initialized to `0.0`

[1] Make sure you use the API to manipulate a ring buffer, and do **not** access its internals directly

Problems

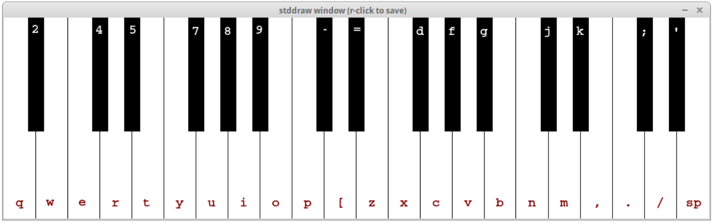
- `create_from_samples(init)`
  - Create a ring buffer whose capacity is same as the size of the given list `init`
  - Populate the ring buffer with values from `init`
  - Return the ring buffer
- `pluck(string)`
  - Replace each value (dequeue followed by enqueue) in the given ring buffer with a random number from the interval $[-0.5, 0.5]$
- `tic(string)`
  - Dequeue a value `a` in the given ring buffer and peek at the next value `b`
  - Enqueue the value `0.996 * 0.5 * (a + b)` into the ring buffer
- `sample(string)`
  - Peek and return a value from the given ring buffer

Epilogue

The program guitar_sound_synthesis.py is a visual client that uses your guitar_string.py (and ring_buffer.py) modules to play a guitar in real-time, using the keyboard to input notes; when the user types the appropriate characters, the program plucks the corresponding string

The keyboard arrangement imitates a piano keyboard: the "white keys" are on the qwerty and zxcv rows and the "black keys" on the 12345 and asdf rows of the keyboard

```
$ python3 guitar_sound_synthesis.py
```



Epilogue

Your project report (use the given template, `report.txt`) must include

- time (in hours) spent on the project
- short description of how you approached each problem, issues you encountered, and how you resolved those issues
- acknowledgement of any help you received
- other comments (what you learned from the project, whether or not you enjoyed working on it, etc.)

Before you submit your files

- make sure your programs meet the input and output specifications by running the following command on the terminal

  ```
  $ python3 run_tests.py -v [<problems>]
  ```

  where the optional argument `<problems>` lists the problems (`Problem1`, `Problem2`, etc.) you want to test, separated by spaces; all the problems are tested if no argument is given
- make sure your programs meet the style requirements by running the following command on the terminal

  ```
  $ pycodestyle <program>
  ```

- make sure your report isn't too verbose, doesn't contain lines that exceed 80 characters, and doesn't contain spelling/grammatical mistakes