Project 3 (Guitar Sound Synthesis) Checklist

Prologue

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

Relevant lecture material

- ⇝ Defining Functions
- ⇝ Modules and Clients

Files

- ⇝ `project3.pdf` (project description)
- ⇝ `project3.zip` (starter files for the exercises/problems, `report.txt` file for the project report, and `run_tests.py` file to test your solutions)

Exercises

Exercise 1. (*Sine Function*) Implement the function `sin()` in `sin.py` that calculates the sine of the argument x in radians, using the formula

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots .$$

Hint: to avoid the inaccuracies caused by computing with huge numbers, follow the approach described on page 97 of the IPP text for computing the function e^x.

```
$ python3 sin.py 60
0.8660254037844385
0.8660254037844386
```

Exercises

sin.py

```
import math
import stdio
import sys


# Return sin(x) calculated using the formula:
#   sin(x) = x - x^3/3! + x^5/5! - x^7/7! + ...
def sin(x):
    # Initialize total (sum of the series) to 0.0.
    ...

    # Initialize term (each term in the series) to 1.0, and sign
    # (sign of the term) to 1.
    ...
    ...

    # Initialize i to 1.
    ...

    # Repeat until convergence.
    while total != total + term:
        # Set term to its previous value times x divided by i.
        ...

        # If i is odd, increment total by sign * term, and
        # toggle (negate) sign.
        ...

        # Increment i by 1.
        ...

    # Return the result, total.
    ...
```

Exercises

sin.py

```
# Test client [DO NOT EDIT].
def _main():
    x = math.radians(float(sys.argv[1]))
    stdio.writeln(sin(x))
    stdio.writeln(math.sin(x))


if __name__ == '__main__':
    _main()
```

Exercises

Exercise 2. (*Euclidean Distance*) Implement the function `distance()` in `distance.py` that returns the Euclidean distance between the vectors x and y represented as one-dimensional lists of floats. The Euclidean distance is calculated as the square root of the sums of the squares of the differences between the corresponding entries. You may assume that x and y have the same length.

```
$ python3 distance.py 5
-9 1 10 -1 1
-5 9 6 7 4
13.0
```

Exercises

distance.py

```
import stdio
import sys


# Return the Euclidean distance between x and y, calculated as as
# the square root of the sums of the squares of the differences
# between corresponding entries. You may assume that x and y have
# the same length.
def distance(x, y):
    # Initialize total to 0.0.
    ...

    # Iterate over the lists x and y.
    for u, v in zip(..., ...):
        # Add square of (u - v) to total.
        ...

    # Return the square root of total.
    ...


# Test client [DO NOT EDIT].
def _main():
    n = int(sys.argv[1])
    x = [stdio.readFloat() for i in range(n)]
    y = [stdio.readFloat() for i in range(n)]
    stdio.writeln(distance(x, y))


if __name__ == '__main__':
    _main()
```

Exercises

Exercise 3. (*Palindrome*) Implement the function `is_palindrome()` in `palindrome.py` that returns `True` if the argument s is a palindrome (ie, reads the same forwards and backwards), and `False` otherwise. You may assume that s is all lower case and doesn't contain any whitespace characters.

```
$ python3 palindrome.py bolton
False
$ python3 palindrome.py amanaplanacanalpanama
True
```

Exercises

palindrome.py

```
import stdio
import sys


# Return True if s is a palindrome and False otherwise. You may
# assume that s is all lower case and doesn't any whitespace
# characters.
def is_palindrome(s):
    # Iterate over half of the string s.
    for i in range(...):
        # Compare character at i with the character at
        # len(s) - i - 1. If they are different, s is not a
        # palindrome, so return False.
        ...

    # s is a palindrome, so return True.
    ...


# Test client [DO NOT EDIT].
def _main():
    s = sys.argv[1]
    stdio.writeln(is_palindrome(s))


if __name__ == '__main__':
    _main()
```

Exercises

Exercise 4. (*Reverse*) Implement the function `reverse()` in `reverse.py` that reverses the one-dimensional list a in place, ie, without creating a new list.

```
$ python3 reverse.py
to be or not to be that is the question
<ctrl-d>
question the is that be to not or be to
```

Exercises

reverse.py

```
import stdio
import sys


# Reverse the one-dimensional list a in place, ie, without
# creating a new list.
def reverse(a):
    # Iterate over half of the list a.
    for i in range(...):
        # Exchange element at i in a with the element at
        # len(a) - i - 1.
        ...

# Test client [DO NOT EDIT].
def _main():
    a = stdio.readAllStrings()
    reverse(a)
    for v in a[:-1]:
        stdio.writef('%s ', v)
    stdio.writeln(a[-1])


if __name__ == '__main__':
    _main()
```

Exercises

Exercise 5. (*Transpose*) Implement the function `transpose()` in `transpose.py` that creates and returns a new matrix that is the transpose of the matrix represented by the argument a. Note that a need not have the same number rows and columns. Recall that the transpose of an m-by-n matrix A is an n-by-m matrix B such that $B_{ij} = A_{ji}$, where $0 \leq i < n$ and $0 \leq j < m$.

```
$ python3 transpose.py
2 3
1 2 3
4 5 6
1.0 4.0
2.0 5.0
3.0 6.0
```

Exercises

transpose.py

```
import stdarray
import stdio


# Return a new 2D list representing the transpose of the matrix
# represented by the given 2D list a. Note that the a need not
# have the same number of rows and columns.
def transpose(a):
    # Get the dimensions of matrix a.
    m = ...  # number of rows in a
    n = ...  # number of columns in a

    # Create an n-by-m matrix c with all elements initialized
    # to 0.0.
    ...

    # Fill in the elements of c such that c[i][j] = a[j][i],
    # where 0 <= i < n and 0 <= j < m.
    ...

    # Return c.
    ...


# Test client [DO NOT EDIT].
def _main():
    a = stdarray.readFloat2D()
    c = transpose(a)
    for row in c:
        for v in row[:-1]:
            stdio.write(str(v) + ' ')
        stdio.writeln(row[-1])


if __name__ == '__main__':
    _main()
```

Problems

Student

The guidelines for the project problems that follow will be of help only if you have read the description of the project and have a general understanding of the problems involved. It is assumed that you have done the reading.

Instructor

Please summarize the project description for the students before you walk them through the rest of this checklist document.

Problems

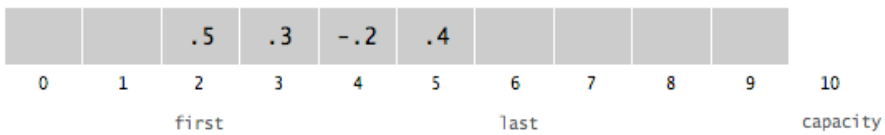
Problem 1. (*Ring Buffer*) Write a module `ring_buffer.py` that implements the following API:

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

Problems

Hints

- ⇝ We represent a ring buffer as a four-element list, in which
  - ⇝ 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`;
  - ⇝ the fourth element (`last`) stores the index one beyond the most recently inserted item
- ⇝ For example, the ring buffer



  is represented as the list

```
[[●, ●, 0.5, 0.3, -0.2, 0.4, ●, ●, ●, ●], 4, 2, 6]
```

- ⇝ Exercise: draw the ring buffer represented by the list

```
[[●, 3, 1, 4, 1, 5, 9], 6, 1, 0]
```

Problems

- `create(capacity)`
  - Create and return a ring buffer with `buff` having the given capacity and all of `buff`'s 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   returns
-------------------------------------------------------------------
create(5)          [[None, None, None, None, None], 0, 0, 0]    rb
enqueue(rb, 'A')   [[ 'A', None, None, None, None], 1, 0, 1]
enqueue(rb, 'B')   [[ 'A',  'B', None, None, None], 2, 0, 2]
enqueue(rb, 'C')   [[ 'A',  'B',  'C', None, None], 3, 0, 3]
dequeue(rb)        [[ 'A',  'B',  'C', None, None], 2, 1, 3]    'A'
enqueue(rb, 'D')   [[ 'A',  'B',  'C',  'D', None], 3, 1, 4]
enqueue(rb, 'E')   [[ 'A',  'B',  'C',  'D',  'E'], 4, 1, 0]
enqueue(rb, 'F')   [[ 'F',  'B',  'C',  'D',  'E'], 5, 1, 1]
peek(rb)           [[ 'F',  'B',  'C',  'D',  'E'], 5, 1, 1]    'B'
enqueue(rb, 'G')   Error: buffer full!
```

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

Problems

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 (use `math.ceil()`) to the nearest integer, and all values initialized to `0.0`
- ⇝ `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

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

Problems

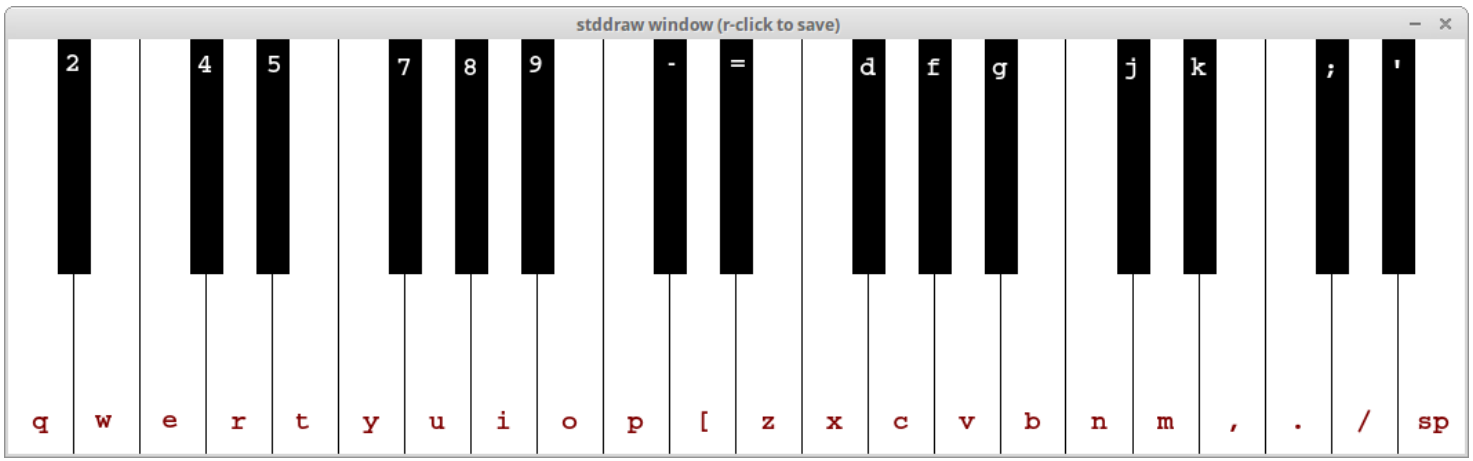
- `pluck(string)`
  - Replace each value (dequeue followed by enqueue) in the given ring buffer with a random number from the interval $[-0.5, 0.5]$
- `tic(string)`
  - Dequeue a value `a` in the given ring buffer and peek at the next value `b`
  - Enqueue the value `0.996 * 0.5 * (a + b)` into the ring buffer
- `sample(string)`
  - Peek and return a value from the given ring buffer

Problems

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

Use the template file `report.txt` to write your report for the project

Your report must include

- ⇝ Time (in hours) spent on the project
- ⇝ Difficulty level (1: very easy; 5: very difficult) of the project
- ⇝ A 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.)

Epilogue

Before you submit your files

- ⇝ Make sure your programs meet the style requirements by running the following command on the terminal

```
$ pycodestyle <program>
```

  where `<program>` is the `.py` file whose style you want to check

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

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

  where the optional argument `<items>` lists the exercises/problems (`Exercise1`, `Problem2`, etc.) you want to test, separated by spaces; all the exercises/problems are tested if no argument is given

- ⇝ Make sure your code is adequately commented, is not sloppy, and meets any project-specific requirements, such as corner cases and running time
- ⇝ Make sure your report uses the given template, isn't too verbose, doesn't contain lines that exceed 80 characters, and doesn't contain spelling mistakes

Epilogue

Files to submit

1. `sin.py`
2. `distance.py`
3. `palindrome.py`
4. `reverse.py`
5. `transpose.py`
6. `ring_buffer.py`
7. `guitar_string.py`
8. `report.txt`