Programming Assignment 1

ECS 032B FQ 2022

Due: October 16th, 2022

Setup:

Python3.6+ to develop and run your code.

Provided files:

PA1 /

- __init__.py
- pythonreview.py
- arrays.py
- linkedlist.py
- adts.py
- utils.py
  - *This file is very important. Make sure to read and understand the code inside.*
  - *Hint: You may not need to override some of the provided functions.*
- test.py
  - *Run this file to test your code.*
- tests/
  - test_pythonreview.py
  - test_arrays.py
  - test_linkedlist.py
  - test_adts.py

Submission instructions:

Submit all provided files through Gradescope as a zip file.

Do not remove/rename files nor change any naming conventions.

You should receive your grade for correctness automatically.

Rubric:

Correctness = 18 points

TAs will double-check your assignment to ensure that you are adhering the specifications. They may overrule your score if you break the listed restrictions.

Documentation = 2 points

Provide comments in code and good documentation on Gradescope.

A TA will grade this based on correctness.

Extra Credit

+0.2 points for every day you turn in all parts (Python files + Documentation) early. You must submit all parts; the latest day will be counted.

(Maximum: 2 points)

Section 1: Python Review (pythonreview.py)

No additional modules to be imported for this section.

You may use itertools (see: https://docs.python.org/3/library/itertools.html#itertools.combinations)

You may use any built-in Python functionalities.

`def findWithSum(arr:list, val:int, n=2:int) -> dict:`

Given a list of integers `arr` and integer `val`, find `n` integers in `arr` that add up to the `val`. Return a dictionary with keys as indices of the `n` elements and the values as the value of the elements.

Assume that each input would have exactly one solution. If no solutions exist, then return an empty dictionary.

```
arr = []
val = 0
findWithSum(arr,val)
>>> {}

arr = [1,2,3]
val = 2
findWithSum(arr,val,1)
>>> {1:2}

arr = [1,1,1]
val = 3
findWithSum(arr,val,3)
>>> {0:1,1:1,2:1}

arr = [3,2,5]
val = 5
findWithSum(arr,val,1)
>>> {2:5}
```

For Section 2-4:

No additional modules may be imported.

Do not modify the arguments or the naming of the already defined functions.

You may not use built-in Python data structures.

Cannot use any of the following:

https://docs.python.org/3.8/tutorial/datastructures.html

Section 2: Dynamic Array (arrays.py)

In this section, you will implement a dynamic array using the `StaticArray` class

This class extends the given `StaticArray` class (a child class of `Collections`).

Enforce shifting property:

There should not be empty slots (i.e. `None`) between occupied slots.

This property is not enforced in `StaticArray`, hence some modifications must be made.

Assume that `None` cannot be an element in `Collections`

Follow this resizing rule:

Double the size if more than 80% is occupied. Only double upon adding new items.

Halve the size if less than 20% is occupied. Only halve upon removing items.

```
class DynamicArray(StaticArray):
```

```
def __init__(self, size:int, isSet = False:bool):
```
Constructor for collection with `size` empty slots.

To determine whether this collection will enforce the set property, use `isSet` argument.

```
def __getitem__(self, index:int) -> Any:
```
Return the value at the `index` of the collection.

```
def __setitem__(self, index:int, value:Any) -> None:
```
Set the `index` of the collection to the `value`.

```
def __delitem__(self, index:int) -> None:
```
Remove the element at `index` of the collection.

```
def append(self, value:Any) -> None:
```
Add `value` to collection at the next open slot.

```
def extend(self, arr:Collections) -> None:
```
Append the values of `arr` to the collection.

```
def remove(self, value:Any) -> None:
```
Remove the first instance of the `value` from collection.

```
def argwhere(self, value:Any) -> StaticArray:
```
Find the `value` in your collection.

Returns a `StaticArray` with elements containing the indices where the value exists.

```
def __eq__(self, arr:Collections) -> bool:
```
Check whether `arr` is equivalent to this collection

Two collections are equal only if they share all the same values, in the correct order.

Their sizes do not matter.

```
def __len__(self) -> int:
```
Return the number of total slots.

```
def get_size(self) -> int:
```
Return the number of elements in this collection.

```
def __iter__(self) -> Any:
```
Yield the next value (Used for iterable functions)

```
def reallocate(self, size:int) -> None:
```
Reallocate your collection to a new collection with `size`.

This is where you transfer over your existing content.

This function should call `resize`.

```
def resize(self, size:int) -> None:
```
Do not modify this function.

Resizes your data to `size`.

This function is destructive! Remember to back-up your previous data somewhere else.

Section 3: Linked Lists (linkedlist.py)

In this section, you will be implementing a linked list using the `Node` class.

You will mostly follow the same instructions as **Section 2**, but with a linked list implementation.

As this class extends `Collections`, you will have to implement all those functions.

Whenever you return a value from the linked list, return it as a `Node`. Do not return the data of the node.

The purpose of returning the node instead of the data itself is mostly just for testing and making sure that you are adhering to the requirements. In practice, you should just return the data.

When you store values in the linked list, you will accept as the original data type and internally wrap it and store it as `Node`

```
class LinkedList(Collections):
    def __init__(self, isSet = False:bool, isDoubly = False:bool, isCircular =
    False:bool):
```

Constructor for linked list.

To determine whether this collection will enforce the set property, use `isSet` argument.

To determine whether this linked list is a doubly or not, use the `isDoubly` argument.

To determine whether this linked list is a circular or not, use the `isCircular` argument.

Section 4: Stacks and Queue ADTs (adts.py)

In this section, you will be implementing stacks and queues.

Use your `DynamicCollection` and `LinkedList` implementations.

```
class StackOrQueue ():
    def __init__(self, isQueue = False:bool, isLinked = False:bool,):
```

This function will initialize this collection.

Implement this collection is a singly linked list if `isLinked = True`. Otherwise, use a dynamic array.

To determine whether this collection is a queue or stack, use the `isQueue` argument.

```
def peek(self) -> Any:
```

Return the first value in the collection but do not remove the value.

```
def push(self, value: Any) -> None:
```

Add the value into the collection.

Treat this as enqueue if `isQueue` is used.

```
def pop(self) -> Any:
```

Remove and return the first value from the collection.

If collection is empty, return None.

Treat this as dequeue if `isQueue` is used.