Carleton University
Department of Systems and Computer Engineering
SYSC 2100 — Algorithms and Data Structures — Winter 2023

Lab 3 - ArrayBag: an array-based implementation of ADT Bag

References

Practical Programming, Third Edition

- Chapter 7, *Using Methods*
- Chapter 14, *Object-Oriented Programming*
- Chapter 15, *Testing and Debugging*

Getting Started

1. Review *Important Considerations When Submitting Files to Brightspace*, which can be found in the course outline.

2. Launch Wing 101 and configure Wing's code reformatting feature. Instructions can be found in Appendix A of document, *SYSC 2100 - Style Guide for Python Code*.

   All code you submit for grading must be formatted. If you decide to disable automatic reformatting, make sure you manually reformat your code (Appendix A.2). At a minimum, we recommend that you reformat your file as you finish each exercise.

3. Download lab3_arraybag.py and lab3_test_arraybag.py from the *Lab Materials* module in Brightspace and open the files in Wing 101.

4. In lab3_arraybag.py, locate these assignment statements at the top of the file:

   ```
   __author__ = ''
   __student_number__ = ''
   ```

   Replace the empty strings with strings containing your name and student number. (Don't modify the variable names.)

5. Your solutions to the exercises must conform to the coding conventions described in *SYSC 2100 - Style Guide for Python Code*.

6. The methods you write in class `ArrayBag` are not permitted to use instances of Python's container types (`list`, `set`, etc.). The test classes in lab3_test_arraybag.py are permitted to do this.

7. **Important**: if you decide to write a script in lab3_arraybag.py, it **must** be placed in an `if __name__ == '__main__':` block, like this:

```
class ArrayBag:
    # methods not shown

if __name__ == '__main__':
    empty_bag = ArrayBag()
    bag = ArrayBag([1, 2, 3, 2, 1])
        ...
```

   Doing this ensures that the script will only be executed when you run lab3_arraybag.py as the "main" module, and won't be executed when the file is imported into another module; for example, the testing/grading program provided to the TAs.

Overview of Class `ArrayBag` and its Unit Tests

In Lab 2, you worked on class `ListBag`, an implementation of ADT Bag that uses an instance of Python's built-in `list` type as its underlying data structure. The `list` object did most of the "hard work": the `ListBag` methods were implemented in terms of the operations supported by type `list`.

In this lab, you'll reimplement ADT Bag. Class `ArrayBag` in lab3_arraybag.py contains an incomplete implementation of ADT Bag that uses a fixed-capacity array as its data structure.

Eight methods (`__init__`, `__str__`, `__repr__`, `__len__`, `__iter__`, `__contains__`, `add` and `_resize`) have been implemented for you. Most of the code is very similar to the methods in class `ArrayList`, which was presented in lectures. **Do not modify these methods.**

You can view nested class `_ArrayBagIterator` and function `_new_array` as abstract "black boxes". You don't need to understand the implementation of `_ArrayBagIterator` to be able to iterate over a bag; similarly, you don't need to understand the implementation `_new_array` to be able to call the function to create arrays.

Experiment with class `ArrayBag`. Read the docstrings for the implemented methods and use the Python shell to try the docstring examples.

The class also has "stub" implementations of five methods that provide additional operations on bags. If you call any of these methods on a `ArrayBag` object, Python will raise a `NotImplementedError` exception. You'll complete the implementation of these methods in Exercises 1 - 5.

File lab3_test_arraybag.py is an incomplete set of unit tests for class `ArrayBag`, implemented using the PyUnit (`unittest`) framework. Read the test classes. The script at the

bottom of the file calls `unittest.main`, which will run all the test cases (the methods with names that start with `test`). Run the script and review the output in the shell window.

When you implement the test classes for Exercises 1 - 5, you should be able to reuse most of the code in the test methods you wrote for Lab 2. Of course, you'll have to modify the methods to create `ArrayBag` objects instead of `ListBag` objects. If you have to rewrite other code in a test method, perhaps the method "knows too much" about how the bag is implemented. Does it access the bag's data structure directly? Can you rewrite the method to remove this dependancy?

Exercise 1: Read the docstring for `count`, delete the `raise` statement, then implement the method. Use the Python shell to run a few tests on `count`.

Think about the test cases that would be required to thoroughly test `count`. Implement these test cases as methods in class `CountTestCase` (in lab3_test_arraybag.py). Run the test script and review the output in the shell window. If necessary, edit `count` and rerun the test script until all test cases pass.

Exercise 2: Read the docstring for `remove`, delete the `raise` statement, then implement the method. This method should reduce the bag's capacity when two-thirds or more of the backing array is not used. (Hint: see class `ArrayList`.)

Note that:

- When the bag is empty, the method should raise a `ValueError` exception that displays the message, "`bag.remove(x): remove from empty bag`". The statement that does this is: `raise ValueError("bag.remove(x): remove from empty bag")`
- When the bag has no occurrences of the item we want to remove, the method should raise a `ValueError` exception that displays the message, "`bag.remove(x): x not in bag`".

Use the Python shell to run a few tests on `remove`.

Think about the test cases that would be required to thoroughly test `remove`. Implement these test cases as methods in class `RemoveTestCase`. (Don't delete `test_remove1` and `test_remove2`.) Run the test script and review the output in the shell window. If necessary, edit `remove` and rerun the test script until all test cases pass.

Exercise 3: Read the docstring for `grab`, delete the `raise` statement, then implement the method. Hint: have a look at the documentation for Python's `random` module: https://docs.python.org/3/library/random.html.

This method should reduce the bag's capacity when two-thirds or more of the backing array is not used. (Hint: see class `ArrayList`.)

Note that when the bag is empty, the method should raise a `ValueError` exception that displays the message, "`bag.grab(): grab from empty bag`".

Use the Python shell to run a few tests on `grab`.

Think about the test cases that would be required to thoroughly test `grab`. Implement these test cases as methods in class `GrabTestCase`. Don't delete `test_grab1`.) Run the test script and review the output in the shell window. If necessary, edit `grab` and rerun the test script until all test cases pass.

Exercise 4: In order to be able to use Python's + operator to concatenate two bags, we need to define a method named `__add__` in `ArrayBag`. Read the docstring for `__add__`, delete the `raise` statement, then implement the method.

Note that if parameter `other` refers to an `object` that is not an `ArrayBag`, the method should raise a `TypeError` exception that displays the message: "`can only concatenate ArrayBag to ArrayBag`". Hint: built-in function `isinstance` determines if an object is an instance of a specified class. See section *Function isinstance, Class object and Class Book* in *Practical Programming,* Chapter 14.

Use the Python shell to run a few tests on `__add__`.

Think about the test cases that would be required to thoroughly test `__add__`. Implement these test cases as methods in class `DunderAddTestCase`. Run the test script and review the output in the shell window. If necessary, edit `__add__` and rerun the test script until all test cases pass.

Exercise 5: In order to be able to use Python's == operator to determine if two bags are equal, we need to define a method named `__eq__` in `ArrayBag`. Read the docstring for `__eq__`, delete the `raise` statement, then implement the method.

Note that if parameter `other` refers to an `object` that is not a `ArrayBag`, the method should return `False`.

Use the Python shell to run a few tests on `__eq__`.

Think about the test cases that would be required to thoroughly test `__eq__`. Implement these test cases as methods in class `EqTestCase`. Run the test script and review the output in the shell window. If necessary, edit `__eq__` and rerun the test script until all test cases pass.

Wrap-up

- Before submitting lab3_arraybag.py, review your code. Does it conform to the coding conventions, as specified in the *Getting Started* section? Has it been formatted? Did you edit the `__author__` and `__student_number__` variables?

- Submit lab3_arraybag.py to Brightspace. Make sure you submit the file that contains your solutions, not the unmodified file you downloaded from Brightspace! You are permitted to make changes to your solutions and resubmit the files as many times as you want, up to the submission due date. Only the most recent submission will be saved by Brightspace.
- Don't submit lab3_test_arraybag.py.

  Solutions that are emailed to your instructor or a TA will not be graded, even if they are emailed before the submission due date.