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

Lab 2 - ADT Bag

References

Practical Programming, Third Edition

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

Python Standard Library, Sequence Types: Common Sequence Operations

- URL: https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range

This section describes the operations provided by Python's sequence types (lists, tuples, ranges).

Python Standard Library, Sequence Types, Mutable Sequence Types

- URL: https://docs.python.org/3/library/stdtypes.html#mutable-sequence-types

This section describes the operations provided by Python's *mutable* sequence types; e..g., lists.

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 lab2_listbag.py and lab2_test_listbag.py from the *Lab Materials* module in Brightspace and open the files in Wing 101.
4. In lab2_listbag.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. **Important**: if you decide to write a script in lab2_listbag.py, it **must** be placed in an `if __name__ == '__main__':` block, like this:

```
class ListBag:
    # methods not shown

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

Doing this ensures that the script will only be executed when you run lab2_listbag.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 `ListBag` and its Unit Tests

Class `ListBag` in lab2_listbag.py contains the incomplete implementation of ADT Bag that was presented in lectures. It uses a Python `list` as the ADT's underlying data structure.

Seven methods (`__init__`, `__str__`, `__repr__`, `__len__`, `__iter__`, `__contains__` and `add`) have been implemented for you. **Do not modify these methods.**

Experiment with class `ListBag`. 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 `ListBag` object, Python will raise a `NotImplementedError` exception. You'll complete the implementation of these methods in Exercises 1 - 5. **Note: unless otherwise specified, do not use loops when coding these methods.** Instead, make use of the operations (operators and methods) provided by Python's `list` type and the built-in functions that operate on lists.

File lab2_test_listbag.py is an incomplete set of unit tests for class `ListBag`, 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.

Exercise 1: Read the docstring for `count`, delete the `raise` statement, then implement the method. Remember: don't use a loop. 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 lab2_test_listbag.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. Remember: don't use a loop.

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. Remember: don't use a loop. Hint: have a look at the documentation for Python's `random` module: https://docs.python.org/3/library/random.html**.**

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 `ListBag`. Read the docstring for `__add__`, delete the

`raise` statement, then implement the method. You can use a loop, although it's not required.

Note that if parameter `other` refers to an `object` that is not a `ListBag`, the method should raise a `TypeError` exception that displays the message: `"can only concatenate ListBag to ListBag"`. 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 `ListBag`. Read the docstring for `__eq__`, delete the `raise` statement, then implement the method. You can use loops, although they aren't required.

Note that if parameter `other` refers to an `object` that is not a `ListBag`, 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 lab2_listbag.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 lab2_listbag.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 lab2_test_listbag.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.