Carleton University
Department of Systems and Computer Engineering
SYSC 2100 — Algorithms and Data Structures — Winter 2021

Lab 3 -ADT Bag

Submitting Lab Work for Grading

Remember, you don't have to finish the lab by the end of your lab period. For this lab, the deadline for submitting your solutions to cuLearn for grading is 11:55 pm (Ottawa time) t**wo days** after your scheduled lab. Solutions that are emailed to your instructor or a TA will not be graded, even if they are emailed before the deadline.

Please read *Important Considerations When Submitting Files to cuLearn*, on the last page of the course outline.

References

- *Problem Solving with Algorithms and Data Structures using Python, Third Edition,* Section 3.5, *Performance of Python Data Structures*, and Section 3.6, *Lists*. For this lab, you don't need to know how to build experiments using Python's `timeit` module. You should understand the Big-Oh characterization of the operations (operators, functions and methods) provided by Python's built-in `list` type.

- *Problem Solving with Algorithms and Data Structures using Python, Third Edition,* Sections 4.1, *Objectives* to 4.4, *Implementing a Stack in Python*, inclusive).

- *Python Standard Library, Sequence Types: Common Sequence Operations*
    - URL: https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range

- *Python Standard Library, Sequence Types, Mutable Sequence Types*
    - URL: https://docs.python.org/3/library/stdtypes.html#mutable-sequence-types

ADT Bag - Specification

A *bag* is a simple container that stores a collection of items. Duplicate items are permitted. The ordering of the items isn't specified (it depends on the bag's implementation). The items must be comparable. (In this lab, all the items will be values of type `int`).

ADT Bag provides the following operations:

- create a new, empty bag.
- to-string: return a printable string representation of the bag's contents.
- add: place the specified item in the bag.
- length: return the number of items stored in the bag.
- contains: determine if a specified item is in the bag and return the appropriate boolean

value.

- count: return the total number of occurrences of a specified item (0 if the item isn't in the bag).
- remove: remove and return one occurrence of the specified item from the bag. Raises an exception if the bag is empty or the item isn't in the bag.
- grab: remove and return a randomly selected item from the bag. Raises an exception if the bag is empty.
- iterator: create and return an iterator that can be used to iterate over the collection.

ADT Bag - Implementation

Many data structures could be used to implement ADT Bag. For this lab, you'll use a Python `list` as the ADT's underlying data structure (we'll investigate other implementations later in the course). A `Bag` object will have one instance variable, `_items`, which will refer to an instance of class `list`. The bag's contents will be stored in this list.

While working on the exercises, refer to the two sections in the Python Standard Library documentation listed earlier. One section describes the operations provided by Python's sequence types (lists, tuples, ranges). The other section describes the operations provided by Python's mutable sequence types ; e..g., lists.

You can also refer to the first few sections of Chapter 4 in the course textbook, which describes how to implement another collection (ADT Stack) using a Python `list`.

Log on to cuLearn and download listbag.py from the *Lab Materials* section of the main course page. Open this file in Wing 101. It contains an incomplete implementation of class `Bag`. You've been provided with two methods:

- `__str__` returns a string representation of the bag. This method is simple:

  ```
  return str(self._items)
  ```

  Instance variable `_items` refers to the `list` that stores the bag's contents. The method simply returns the string returned by the underlying list's `__str__` method.

- `__iter__` returns an object that can be used to iterate over the bag. In other words, if `shopping_bag` refers to an instance of `Bag`, you can write:

  ```
  for item in shopping_bag:
      # Print the bag's contents one-by-one.
      print(item)
  ```

  This method requests the underlying list to create an iterator, and returns that object:

  ```
  return iter(self._items)
  ```

"Stub" implementations have been provided for the other methods. If you call any of these

methods on a `Bag` object, Python will throw a `NotImplementedError` exception.

Exercise 1: Read the docstring for `__init__`. A `Bag` object should have one instance variable, names `_items`, which is initialized to an empty instance of Python's `list` type. Replace the `raise` statement with a correct implementation of the method. Use this test to check if your method is correct:

```
>>> bag = Bag()
>>> bag._items
[]  # Shows that _items refers to an empty list.
```

Exercise 2: Read the docstring for `__repr__`. Replace the `raise` statement with a correct implementation of the method. Use the shell to test `__repr__`. (You'll only be able to test if `__repr__` works with an empty bag, because we don't yet have a way to put items in a bag.)

Exercise 3: Read the docstring for `add`. Replace the `raise` statement with a correct implementation of the method. Use the shell to test `add`. Now that you can put items in a bag, you can also verify that `__repr__` works with a bag that contains one or more items. You should also run the example in the docstring for `__iter__`, to convince yourself that we can iterate over the items in a bag.

Exercise 4: Try this experiment:

```
>>> bag = Bag()
>>> len(bag)
```

In order for Python's built-in `len` function to work with `Bag` objects, we need to define a `__len__` method in the class. Read the docstring for `__len__`. Replace the `raise` statement with a correct implementation of the method. Use the shell to test `__len__`.

Exercise 5: Try this experiment:

```
>>> bag = Bag()
>>> 2 in bag
```

In order for Python's `in` operator to work with `Bag` objects, we need to define a `__contains__` method in the class. Read the docstring for `__contains__`. Replace the `raise` statement with a correct implementation of the method. Use the shell to test `__contains__`.

Exercise 6: Read the docstring for `count`. Replace the `raise` statement with a correct implementation of the method. Use the shell to test `count`.

Exercise 7: Read the docstring for `remove`. Replace the `raise` statement with a correct implementation of the method.

- When the bag is empty, the method should raise a `KeyError` exception that displays the message, "`bag.remove(x): remove from empty bag`".

- When the bag has no instances 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 shell to test `remove`.

Exercise 8: Read the docstring for `grab`. Replace the `raise` statement with a correct implementation of the method. Hint: have a look at the documentation for Python's `random` module: https://docs.python.org/3/library/random.html

- When the bag is empty, the method should raise a `KeyError` exception that displays the message, "`bag.grab(): grab from empty bag`".

Use the shell to test `grab`.

Wrap Up

The submission deadlines for this lab are:

Lab Section	Lab Date/Time	Submission Deadline (Ottawa Time)
L5	Tuesday, 11:35 - 13:25	Thursday, Jan. 28, 23:55
L2	Thursday, 9:35 - 11:25	Saturday, Jan. 30, 23:55
L4	Thursday, 12:35 - 14:25	Saturday, Jan. 30, 23:55
L3	Friday, 9:35 - 11:25	Sunday, Jan. 31, 23:55
L1	Friday, 14:35 - 16:25	Sunday, Jan. 31, 23:55

To submit your lab work, go to the cuLearn page **for your lab section** (not the main course page). Submit listbag.py. Ensure you submit the version of the file that contains your solutions, and not the unmodified file you downloaded from cuLearn! You are permitted to make changes to your solutions and resubmit the file as many times as you want, up to the deadline. Only the most recent submission is saved by cuLearn.

Last edited: Jan. 24, 2021