Lab 8

In this lab, you will implement a hash table that uses separate chaining for its collision resolution strategy.

1 Hash Table

1.1 Our Implementation

In this lab, your "chains" will be Python lists. So, the `HashTable` class will be a Python list (the table) of Python lists (the separate chains).

The hash table will be initialized with two parameters, the initial capacity of the hash table, and a hash function to use for the keys. By default, we will use Python's builtin `hash` function, but we want to support creating a hash table that uses a different hashing function.

To create a list of empty lists, we will do something like:

```
table = [[] for _ in range(10)]
```

We can then add to a given chain by appending to the relevant list. So, if we want to insert the key-value pair `(16, 'cat')` into the table at index 6, we would do:

```
table[6].append((16, 'cat'))
```

And then the table would look like:

```
[[], [], [], [], [], [], [(16, 'cat')], [], [], []]
```

1.2 Operations

You will implement the following operations for a hash table:

- `insert` This function takes a hash table, a key, and a value as arguments. The function will insert the key-value pair into the hash table based on the hash value of the key mod the table size.

  If they key being inserted is already present in the hash table, *the old value will be replaced* by the new given value. We should not end up with duplicate keys in the table.

  If the insert would cause the load factor of the hash table to exceed 1.0, your hash table should double in capacity and reinsert all of the values from the old table into the new table.

- `get_item` This function takes a hash table and a key as arguments. The function will retrieve the value for the given key, and will return the value.

  If the given key is not present in the hash table, this function will raise a `KeyError`.

- `contains` This function takes a hash table and a key as arguments. The function will return `True` if the key is present in the hash table, and `False` otherwise.

- `remove` This function takes a hash table and a key as arguments. The function will remove the key-value pair from the hash table and return the pair (as a tuple).

  If the given key is not present in the hash table, this function will raise a `KeyError`.

- `size` This function takes a hash table as an argument and returns the number of key-value pairs that have been inserted into the hash table.
- `keys` This function takes a hash table as an argument and returns a list of all of the keys in the table.
- `load_factor` This function takes a hash table as an argument and returns the load factor of the hash table.
- `_contents` This function takes a hash table as an argument and returns the internal hash table array. You may find this useful for testing.

  Note that the leading underscore is not a typo. It's generally bad practice for a function like this to exist, and so we're using the Python standard practice of a leading underscore to mean that it's not meant to be used directly by "users" of your `HashTable` structure. You should only be using this for testing.

2 Testing

As mentioned in the syllabus, your code is periodically graded prior to the deadline and you will receive automated feedback based on the results of my tests. *But*, in order to receive *any* feedback, your tests much provide 100% test coverage of the code you are submitting. If you do not have 100% coverage, this is the only feedback you will receive.

100% test coverage means that every line of your code is run at some point in some test. Imagine if that weren't the case. That means that you have a line of code (or multiple lines of code) that could do anything, and you'd never know. You never tested them!

Your tests will go in files named `hash_table_tests.py`.

3 GitHub Submission

Push your finished code back to GitHub. Refer to Lab 0, as needed, to remember how to push your local code.