NAME ____ **ICA-15**

NetID ______________________@email.arizona.edu

1. Implement the Dictionary ADT
   - holds key/value pairs
   - provides the following operations:
     - put(key, value)
       - makes an entry for a key/value pair
       - assumes key is not already in the dictionary
     - get(key) looks up key in the dictionary
       - returns the value associated with key (and None if not found)

```
Usage:
>>> d = Dictionary(7)
>>>
>>> d.put('five', 5)
>>> d.put('three', 3)
>>>d.get('three') == 3
Hint:
>>> d._pairs
[['five', 5], ['three', 3], None, None, None, None, None]
```

2. Modify the ADT below to use a hash function to compute the index for a new key/value pair.

   Use the following hash function:

   ```
    def _hash(self, k):
           return len(k) % len(self._pairs)
   ```

```
class Dictionary:
    def __init__(self,capacity):
        # each element will be a key/value pair
        self._pairs = [None] * capacity
        self._nextempty = 0

    def put(self, k, v):
        self._pairs[self._nextempty] = [k,v]
        self._nextempty += 1

    def get(self, k):
        for pair in self._pairs[0:self._nextempty]:
            if pair[0] == k:
                return pair[1]
        return None
```

3. Use open addressing to insert the key 23 into the hash table below. Give the probe sequence. *The hash function is the key % 7.*

0	1	2	3	4	5	6
14	24	2	10		19	

probe sequence: ________________________________

4. Modify the put() method of the ADT below to implement open addressing with linear probing.

```
class Dictionary:
    def __init__(self, capacity):
       # each element will be a key/value pair
        self._pairs = [None] * capacity

    def _hash(self, k):
        return len(k) % len(self._pairs)

    def put(self, k, v):
        self._pairs[self._hash(k)] = [k,v]
```

5. Use double hashing to insert key 23:

key	hash value	probe decrement
10	3	1
2	2	1
19	5	2
14	0	2
24	3	3
23	2	3

0	1	2	3	4	5	6
14		2	10	24	19	