CPE111: Programming with Data Structures
Week 7: Map /Hash Table

Practice I) Map
Mission: Write a program to load student list and student's ID from student.xlsx and add to the map by using student's ID as **key** and student's name as **value**

List 1: Map.py

```
1  from collections.abc import MutableMapping
2  from random import randrange
3  class Map(MutableMapping):
4     """ Our own abstract base class that includes a nonpublic_Item class"""
5     #-------- nested _Item class ------------
6     class _Item:
7      """ Lightweight composite to store key-value pairs as map items"""
8       __slots__ = '_key','_value'
9       def __init__(self,k,v):
10          self._key = k
11          self._value = v
12
13      def __eq__(self,other):
14          return self._key == other._key # compare items based on their keys
15
16      def __ne__(self,other):
17         return not (self == other) # opposite of __eq__
18
19      def __lt__(self,other):
20          return self._key < other._key # compare items based on their keys
21
22 class TableMap(Map):
23    """Map implementation using an unordered list"""
24    def __init__(self):
25       """Create an empty map"""
26       self._table = []
27
28    def __getitem__(self,k):
29       """Return value associated with key (raise Key Error if not found)"""
30       for item in self._table:
31          if k == item._key:
32             return item._value
33       raise KeyError('Key Error: '+repr(k))
34
35    def __setitem__(self,k,v):
36       """ Assign value v to key k, overwriting existing value if present"""
37       for item in self._table:
38          if k == item._key:  #Found a match:
39             item._value = v #reassign value
40             return        # and quit
41       # did not find match for key
42       self._table.append(self._Item(k,v))
```

```
43
44   def __delitem__(self,k):
45       """ Remove item associated with key k (raise KeyError if not found)"""
46       for j in range(len(self._table)):
47           if k == self._table[j]._key:
48               self._table.pop(j)
49               return
50       raise KeyError('Key Error: ' +repr(k))
51
52   def __len__(self):
53       """Return number of items in the map"""
54       return len(self._table)
55
56   def __iter__(self):
57       """Generate iteration of the map's keys"""
58       for item in self._table:
59           yield item._key
```

Practice II) Hash Table and Separate chaining

Mission: Write a program to load student list and student's ID from student.xlsx and add to the Chain Hash Map, and see the differences

List2: ChainHashMap.py

```
1  class ChainHashMap(Map):
2    """Abstract base class for map using has-table with MAD compression"""
3    def __init__(self, cap = 11, p = 109345121):
4        """Create an empty hash-table map"""
5        self._table = cap * [None]
6        self._n = 0 # number of entries in the map
7        self._prime = p # prime for MAD compression
8        self._scale = 1 + randrange(p-1) # scale for 1 to p-1 for MAD
9        self._shift = randrange(p)  # shift from 0 to p-1 for MAD
10   def _hash_function(self,k):
11       return (hash(k)*self._scale + self._shift) % self._prime % len(self._table)
12
13   def __len__(self):
14       return self._n
15
16   def __getitem__(self,k):
17       j = self._hash_function(k)
18       return self._bucket_getitem(j,k)
19
20   def __setitem__(self,k,v):
21       j = self._hash_function(k)
22       self._bucket_setitem(j,k,v) # subroutime maintains self._n
23       if self._n > len(self._table) // 2: # keep load factor <= 0.5
24           self._resize(2 * len(self._table) -1)   # number 2^x -1 is often prime
```

```
25
26  def __delitem__(self,k):
27      j = self._hash_function(k)
28      self._bucket_delitem(j,k)
29      self._n -= 1
30
31  def _resize(self,c): # resize bucket array to capacity c
32      old = list(self.items()) # use iteration to record existing items
33      self._table = c * [None] #
34      self._n = 0
35      for (k,v) in old:
36          self[k] = v
37
38  def _bucket_getitem(self,j,k):
39      bucket = self._table[j]
40      if bucket is None:
41          raise KeyError('Key Error: '+ repr(k)) # no match found
42      return bucket[k]
43
44  def _bucket_setitem(self,j,k,v):
45      if self._table[j] is None:
46          self._table[j] = TableMap() # sent New table
47      oldsize = len(self._table[j])
48      self._table[j][k] = v
49      if len(self._table[j]) > oldsize: # key was new to the table
50            self._n += 1 # increase overall map size
51  def _bucket_delitem(self,j,k):
52      bucket = self._table[j]
53      if bucket is None:
54          raise KeyError('Key Error: '+repr(k))
55      del bucket[k]
56
57  def __iter__(self):
58      for bucket in self._table:
58          if bucket is not None:
59              for key in bucket:
60                  yield key
```