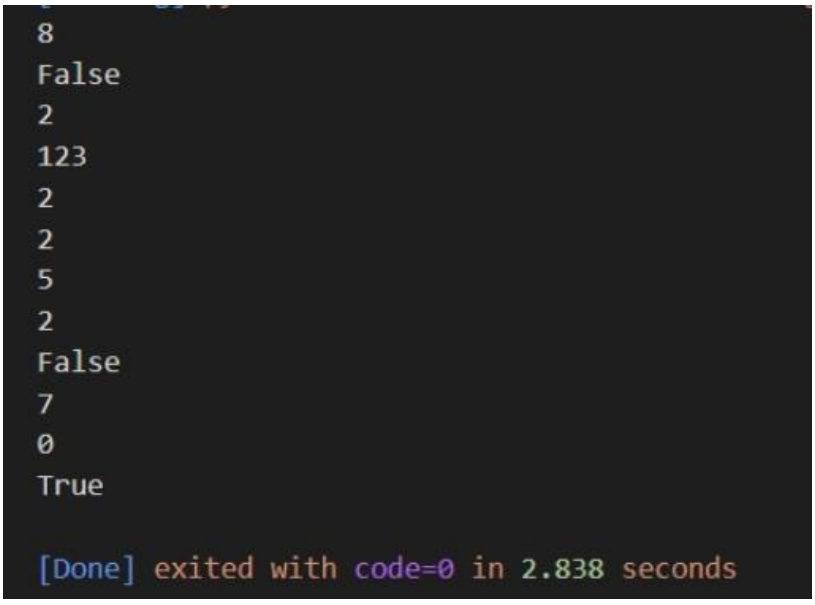
Data Structures And Algorithms - II

PRACTICAL-4

```
class
    HashTable(object):
    MINIMUM_BUCKETS = 4
    BUCKET_SIZE = 5
    def__init__(self, capacity=MINIMUM_BUCKETS*BUCKET_SIZE):
        self.size = 0
        self.threshold = capacity
        self.buckets = [[] for _ in range(capacity//self.BUCKET_SIZE)]

    def insert(self, key, value):
        bucket = self.hash(key)
        for n, element in enumerate(self.buckets[bucket]):
            if element['key'] == key:
                element['value'] = value
                self.buckets[bucket][n] = element
                return
            else:
                self.buckets[bucket].append({'key': key, 'value': value})
                self.size += 1
                if self.size == self.threshold:
                    self.resize()

    def get(self, key):
        bucket = self.hash(key)
        for element in self.buckets[bucket]:
            if element['key'] == key:
                return element['value']
                raise KeyError("No such key '{0}'!".format(key))

    def erase(self, key):
        bucket = self.hash(key)
        for n, element in enumerate(self.buckets[bucket]):
            if element['key'] == key:
                del self.buckets[bucket][n]
                self.size -= 1
                return
                raise KeyError("No such key '{0}'!".format(key))


    def hash(self, key):
        return hash(key) % len(self.buckets)

    def contains(self, key):
        bucket = self.hash(key)
        for element in self.buckets[bucket]:
            if element['key'] == key:
                return True
                return False

    def___getitem_(self, key):
        return self.get(key)

    def___setitem_(self, key, value):
        return self.insert(key, value)
    def___len_(self):
        return self.size
```

```
    def is_empty(self):
        return self.size == 0

    def resize(self):
        capacity = self.size / self.BUCKET_SIZE * 2
        if capacity >= self.MINIMUM_BUCKETS:
            old = self.buckets
            self.buckets = [[] for _ in range(capacity)]
            for n in range(len(self.buckets)):
                self.buckets[n] = old[n]
                self.rehash()

    def resize(self):
        capacity = self.size / self.BUCKET_SIZE * 2
        if capacity >= self.MINIMUM_BUCKETS:
            old = self.buckets
            self.buckets = [[] for _ in range(capacity)]
            for n in range(len(self.buckets)):
                self.buckets[n] = old[n]
                self.rehash()
    def main():
        table = HashTable()
        table["one"] = 1
        table["two"] = 2
        table.insert("three", 3)
        table["four"] = 4
        table["four"] = 4
        table["five"] = 5
        table["six"] = 6
        table["seven"] = 7
        table["eight"] = 8
        print(len(table))
        print(table.is_empty())
        print(table["two"])
        table["one"] = 123
        print(table["one"])
        print(table["two"])
        table.erase("three")
        print(table["two"])
        table.erase("four")
        table.erase("five")
        print(len(table))
        print(table["two"])
        table.erase("one")
        table.erase("two")
        table.erase("six")
        print(table.is_empty())
        print(table["seven"])
        table.erase("seven")
        table.erase("eight")
        print(len(table))
        print(table.is_empty())
if_name__ == '_main_': main()
```

OUTPUT::


8
False
2
123
2
2
5
2
False
7
0
True
[Done] exited with code=0 in 2.838 seconds