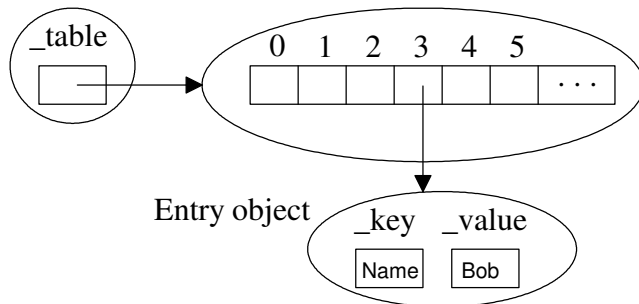


1. The Map/Dictionary abstract data type (ADT) stores key-value pairs. The key is used to look up the data value.

Method call	Class Name	Description
<code>d = ListDict()</code>	<code>__init__(self)</code>	Constructs an empty dictionary
<code>d["Name"] = "Bob"</code>	<code>__setitem__(self, key, value)</code>	Inserts a key-value entry if <code>key</code> does not exist or replaces the old value with <code>value</code> if <code>key</code> exists.
<code>temp = d["Name"]</code>	<code>__getitem__(self, key)</code>	Given a <code>key</code> return its value or <code>None</code> if <code>key</code> is not in the dictionary
<code>del d["Name"]</code>	<code>__delitem__(self, key)</code>	Removes the entry associated with <code>key</code>
<code>if "Name" in d:</code>	<code>__contains__(self, key)</code>	Return <code>True</code> if <code>key</code> is in the dictionary; return <code>False</code> otherwise
<code>for k in d:</code>	<code>__iter__(self)</code>	Iterates over the keys in the dictionary
<code>len(d)</code>	<code>__len__(self)</code>	Returns the number of items in the dictionary
<code>str(d)</code>	<code>__str__(self)</code>	Returns a string representation of the dictionary

ListDict
object

Python list object



```

from entry import Entry

class ListDict(object):
    """Dictionary implemented with a Python list."""

    def __init__(self):
        self._table = []

    def __getitem__(self, key):
        """Returns the value associated with key or
        returns None if key does not exist."""
        entry = Entry(key, None)
        try:
            index = self._table.index(entry)
            return self._table[index].getValue()
        except:
            return None

    def __delitem__(self, key):
        """Removes the entry associated with key."""
        entry = Entry(key, None)
        try:
            index = self._table.index(entry)
            self._table.pop(index)
        except:
            return

    def __str__(self):
        """Returns string repr. of the dictionary"""
        resultStr = "{"
        for item in self._table:
            resultStr = resultStr + " " + str(item)
        return resultStr + "}"

    def __iter__(self):
        """Iterates over keys of the dictionary"""
        for item in self._table:
            yield item.getKey()
        raise StopIteration
  
```

```

class Entry(object):
    """A key/value pair."""

    def __init__(self, key, value):
        self._key = key
        self._value = value

    def getKey(self):
        return self._key

    def getValue(self):
        return self._value

    def setValue(self, newValue):
        self._value = newValue

    def __eq__(self, other):
        if not isinstance(other, Entry):
            return False
        return self._key == other._key

    def __str__(self):
        return str(self._key) + ":" + str(self._value)
  
```

a) Complete the code for the `__contains__` method.

```
def __contains__(self, key):
```

b) Complete the code for the `__setitem__` method.

```
def __setitem__(self, key, value):
```

2. Dictionary implementation using hashing with chaining -- an UnorderedList object at each slot in the hash table.

```

from entry import Entry
from unordered_linked_list import UnorderedList

class ChainingDict(object):
    """Dictionary implemented using hashing with chaining."""

    def __init__(self, capacity = 8):
        self._capacity = capacity
        self._table = []
        for index in range(self._capacity):
            self._table.append(UnorderedList())
        self._size = 0
        self._index = None

    def __contains__(self, key):
        """Returns True if key is in the dictionary or
        False otherwise."""
        self._index = abs(hash(key)) % self._capacity
        entry = Entry(key, None)

        return self._table[self._index].search(entry)

    def __getitem__(self, key):
        """Returns the value associated with key or
        returns None if key does not exist."""
        if key in self:
            entry = Entry(key, None)
            entry = self._table[self._index].remove(entry)
            self._table[self._index].add(entry)
            return entry.getValue()
        else:
            return None

    def __delitem__(self, key):
        """Removes the entry associated with key."""
        entry = Entry(key, None)
        if key in self:
            entry = Entry(key, None)
            entry = self._table[self._index].remove(entry)
            self._size -= 1

    def __setitem__(self, key, value):
        """Inserts an entry with key/value if key
        does not exist or replaces the existing value
        with value if key exists."""
        entry = Entry(key, value)
        if key in self:
            entry = self._table[self._index].remove(entry)
            entry.setValue(value)
        else:
            self._size += 1
            self._table[self._index].add(entry)

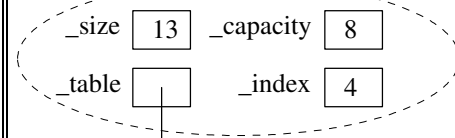
    def __len__(self):
        return self._size

    def __str__(self):
        result = "HashDict: capacity = " + \
            str(self._capacity) + ", load factor = " + \
            str(len(self) / self._capacity)
        for i in range(self._capacity):
            result += "\nRow " + str(i) + ": " + str(self._table[i])
        return result

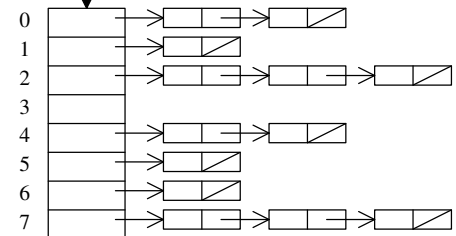
    def __iter__(self):
        """Iterates over the keys of the dictionary"""

```

ChainingDict Object



Python list of UnorderList objects containing Entries



- a) In `__getitem__`, why is the `entry = Entry(key, None)` object created?
- b) In `__getitem__`, where does `self._index` receive its value?
- c) What single modification was needed to the UnorderedList's `remove` method?
- d) Complete the `__iter__` method.