Implementation of a Hash Table

In this lecture we will be implementing our own Hash Table to complete our understanding of Hash Tables and Hash Functions! Make sure to review the video lecture before this to fully understand this implementation!

Keep in mind that Python already has a built-in dictionary object that serves as a Hash Table, you would never actually need to implement your own hash table in Python.

Map

The idea of a dictionary used as a hash table to get and retrieve items using **keys** is often referred to as a mapping. In our implementation we will have the following methods:

- **HashTable()** Create a new, empty map. It returns an empty map collection.
- **put(key,val)** Add a new key-value pair to the map. If the key is already in the map then replace the old value with the new value.
- **get(key)** Given a key, return the value stored in the map or None otherwise.
- **del** Delete the key-value pair from the map using a statement of the form del map[key].
- **len()** Return the number of key-value pairs stored
- **in** the map in Return True for a statement of the form **key in map**, if the given key is in the map, False otherwise.

In [1]:
```
class HashTable(object):

    def __init__(self,size):

        # Set up size and slots and data
        self.size = size
        self.slots = [None] * self.size
        self.data = [None] * self.size

    def put(self,key,data):
        #Note, we'll only use integer keys for ease of use with the Hash Function

        # Get the hash value
        hashvalue = self.hashfunction(key,len(self.slots))

        # If Slot is Empty
        if self.slots[hashvalue] == None:
            self.slots[hashvalue] = key
            self.data[hashvalue] = data

        else:

            # If key already exists, replace old value
            if self.slots[hashvalue] == key:
                self.data[hashvalue] = data

            # Otherwise, find the next available slot
            else:
```

```
                nextslot = self.rehash(hashvalue,len(self.slots))

                # Get to the next slot
                while self.slots[nextslot] != None and self.slots[nextslot] != key:
                    nextslot = self.rehash(nextslot,len(self.slots))

                # Set new key, if NONE
                if self.slots[nextslot] == None:
                    self.slots[nextslot]=key
                    self.data[nextslot]=data

                # Otherwise replace old value
                else:
                    self.data[nextslot] = data

    def hashfunction(self,key,size):
        # Remainder Method
        return key%size

    def rehash(self,oldhash,size):
        # For finding next possible positions
        return (oldhash+1)%size


    def get(self,key):

        # Getting items given a key

        # Set up variables for our search
        startslot = self.hashfunction(key,len(self.slots))
        data = None
        stop = False
        found = False
        position = startslot

        # Until we discern that its not empty or found (and haven't stopped yet)
        while self.slots[position] != None and not found and not stop:

            if self.slots[position] == key:
                found = True
                data = self.data[position]

            else:
                position=self.rehash(position,len(self.slots))
                if position == startslot:

                    stop = True
        return data

    # Special Methods for use with Python indexing
    def __getitem__(self,key):
        return self.get(key)

    def __setitem__(self,key,data):
        self.put(key,data)
```

Let's see it in action!

In [2]:
```
h = HashTable(5)
```

In [3]:
```
# Put our first key in
h[1] = 'one'
```

In [4]:
```
h[2] = 'two'
```

In [5]:
```
h[3] = 'three'
```

In [6]:
```
h[1]
```

Out[6]:
```
'one'
```

In [7]:
```
h[1] = 'new_one'
```

In [8]:
```
h[1]
```

Out[8]:
```
'new_one'
```

In [10]:
```
print(h[4])
```

```
None
```

Great Job!

That's it for this rudimentary implementation, try implementing a different hash function for practice!