Objetivo. El objetivo de esta sesión es implementar en Python 3.3 la estructura de datos `hashtable`. Al final de la sesión de hoy, cada estudiante debe contar con una implementación correcta de `hashtable`.

Material bibliográfico. Se utilizará como material de referencia el material de la primera sesión de la semana (lecturas, videos, etc.). Complementariamente, se utilizará como punto de partida una plantilla disponible en la página del curso.

Instrucciones. Realice las siguientes actividades:

1. Inspeccione el archivo `hashtable.py`: determine qué métodos hay, cuáles están implementados y cuáles están aún por implementar. Cargue la plantilla en Python y obtenga la documentación de cada uno de los métodos con el comando:

   ```
   help(hashtable)
   ```

2. Implemente cada uno de los métodos privados que utilizan la convención de nombres de Python para métodos especiales que no han sido implementados. Pruebe que el código que ha implementado funciona para:

   - crear una instancia de `hashtable` (método `__init__`)
   - visualizar el contenido de un `hashtable` vacío (método `__str__`)

3. Implemente el método `_hash` que computa la función de *hash* para una llave dada k.

   Escriba la documentación asociada a este método. Luego, procesa a hacer la implementación teniendo en cuenta las siguientes condiciones:

   Procesa a hacer la implementación de la siguiente manera:

   (a) convierta la llave k a una cadena en representación de *bytes*, digamos k_1.

   (b) usando la libreria `hashlib` obtenga la representación *md5* de k_1, digamos k_2; note que k_1 es una cadena que representa un número en base 16

   (c) convierta k_2 a base 10 obteniendo un número entero k_h; durante el proceso de conversión haga las operaciones de suma módulo el número primo 9876103 (como una nota al margen: 9876103 es el primo de 7 dígitos más grande que no repite dígitos)

   El número k_h es el *código de hash* correspondiente a k.

4. Implemente el método `_compress` que implementa la función de compresión para un código de *hash* k_h.

   Escriba la documentación asociada a este método. Luego, procesa a hacer la implementación teniendo en cuenta las siguientes condiciones:

   Procesa a hacer la implementación de la siguiente manera:

   (a) compute el valor $(((a \cdot k_h + b) \mod p) \mod N)$ en donde

   - $a = 123456789$
   - $b = 987654321$
   - $p = 4093082899$
   - N es el tamaño reservado para la tabla de *hash*, i.e., $N =$ `len(self._list)`.

   El valor obtenido es un número entero, digamos k_c. Si k_h es el código de *hash* de una llave k, entonces k_c es tael código de compresión de la llave k.

5. Implemente los métodos privados `_enlarge` y `_shrink` que, respectivamente, reducen y aumentan el espacio reservado para el *hashtable*. Tenga en cuenta que el factor de ocupación α está definido como $\alpha = \frac{n}{N}$, en donde n es la cantidad de elementos almacenados en el *hashtable* y N es el tamaño reservado para el *hashtable*.

   Escriba la documentación asociada a cada uno de estos dos métodos. Luego, procesa a hacer la implementación teniendo en cuenta las siguientes condiciones:

   - es necesario reducir el espacio reservado únicamente cuando $\alpha < 0.25$ y $N > 100$; el factor de reducción es 2, es decir, el espacio reservado se reduce a la mitad.
   - es necesario aumentar el espacio reservado únicamente cuando $\alpha > 0.75$; el factor de aumento es 2, es decir, el espacio reservado se aumenta al doble.

6. Implemente los métodos públicos `put`, `get` y `remove`.