**Code walkthrough**

The ASCII values of each character of the input are summed, it takes the remainder divides by the table size then matches this with the index of an empty array of buckets and so there can be complications and collisions, prime number 13 is used for table size, so the data is evenly distributed.

The program only focuses on insertion and displaying the hash table.

**Hash class**

class HashTable:

def \_\_init\_\_(self, size):

self.table = [[] for \_ in range(size)]

**Hash function**

loops through each character in the key to find the ASCII value of each character, adds these values together, uses the modulo operator (%) with the table size to determine which bucket to places the keys to. Prime number 13 is used as table size to avoid collision.

def hash\_function(self, key):

return sum(ord(char) for char in key) % len(self.table)

**Insert**

To find bucket index and append keys.

def insert(self, key, value):

index = self.hash\_function(key)

self.table[index].append((key, value))

**Display**

To display hash table

def display(self):

for i, bucket in enumerate(self.table):

print(f"Bucket {i}: {bucket}")

**Inserts keys**

hash\_table = HashTable(13)

hash\_table.insert("Dragon", "Blue")

hash\_table.insert("Comic", "Star")

hash\_table.insert("Phone", "Charge")

hash\_table.display()