https://github.com/fiamardar/FLCD -> GitHub Repository Link

Lab 2

Statement: Implement the Symbol Table (ST) as the specified data structure, with the corresponding operations

Deliverables: class ST(source code) + documentation.

I chose to implement the SymbolTable as a **HashTable**.

It will contain a *hash function* and a *dictionary of elements.*

The *hash function* will calculate the hash code for each element. This hash code will be calculated as the sum of the ASCII codes of each character in the element.

```
@staticmethod
def hash_code(object_to_hash):
    return sum(bytearray(object_to_hash))
```

The *elements dictionary* has the hash code as the key, and as a value it can have either a single element associated with this hash code, or a list of elements associated with it.

As operations, I implemented add, search and delete.

For **add**, we first check if the hash code associated with the element we want to add already exists as a key in the dictionary. If not, we simply add the hash code as a key and the element as a value. If it already exists, there are two possibilities: the value associated with the key is an element or a list of elements. Here is how we will manage the collisions: if more elements have the same hash code, we will store them in a list and each of them will be found at the position of the tuple (hash_code, index_in_list). And then we will return the hash code or the tuples, depending on the case.

For **search** function, the algorithm is similar, in case we have more elements with the same hash code; we start searching in the list of elements associated with it and return the tuple (hash_code, index_in_list). In case the element is not found, we return False.

For **delete** function, we search for the element and in case we find it, we simply delete it and return the position or the tuple, depending on the case. If the element is not found, return None.