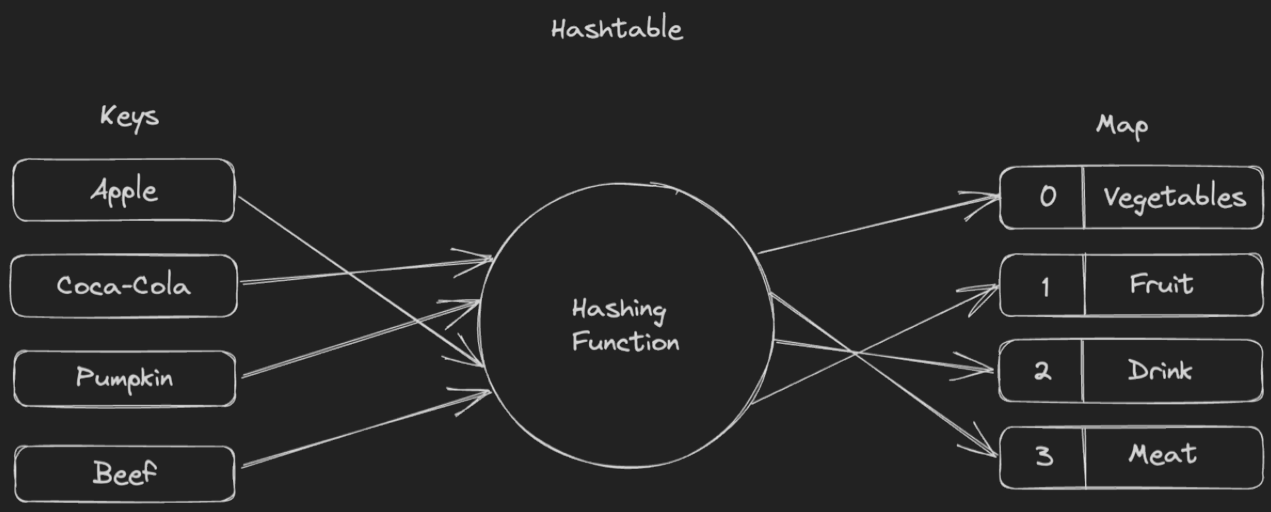
Operation	Maps	If Many Collision
Insertion	O(1)	O(n)
Retrieval	O(1)	O(n)
Removal	O(1)	O(n)

Definition

A hashmap is a data structure that provides efficient lookup, insertion, and deletion operations. It uses a technique called hashing to store and retrieve values based on a unique key.



Terminology

Load Factor	The amount of data points vs the amount of storage `data.len / storage.capacity`
Key	A value that is hashable and is used to look up data. It should not be updated
Value	A value that is associated to a key
Collision	When 2 or more keys map to the same cell

Hashing

Hashing is the process of converting a key into an index within a hash table using a hash function. The hash function takes the key as input and computes a numeric value called the hash code. The hash code is then used to determine the index where the value should be stored or retrieved from.

Key-Value Pairs

Hashtables store data in key-value pairs. The key is used to retrieve the associated value from the hashtable. When storing a value, the key is hashed to determine the index, and the value is stored at that index. To retrieve the value, the key is hashed again to find the corresponding index and retrieve the value stored at that index.

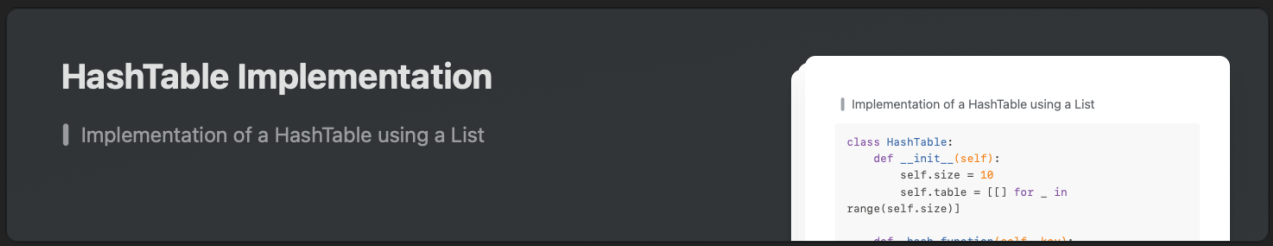
Collision Handling

Collisions can occur when two different keys hash to the same index. There are several techniques to handle collisions:

1. **Separate Chaining**: Each index in the hashtable contains a linked list of key-value pairs. Collisions are resolved by appending new pairs to the linked list at the corresponding index.
2. **Open Addressing**: When a collision occurs, the hashtable probes for the next available index to store the value. Various strategies like linear probing, quadratic probing, or double hashing can be used to find the next index.
    1. **Linear Probing:** Search linearly—$\text{hash} = \text{hash} + i$ for the next available index, can lead to *clustering*
    2. **Quadratic Probing:** Search quadratically—$\text{hash} = \text{hash} + i^2$ for the next available index
3. **Handling Load Factor:** By maintaining a threshold for when your load factor is too high, and when the map should resize we can avoid creating collisions all together. Generally $\lambda \approx .70$

Use Cases

- In caching systems to store recently accessed data for quick retrieval.
- Implementing data structures, where words or keys are mapped to their corresponding definitions or values.
- In compilers and interpreters to store variables, functions, and their associated metadata.
- Count occurrences of elements in a dataset or analyze the frequency of different items.



↑ Map

HashTable Implementation

Implementation of a HashTable using a List

```
class HashTable:
    def __init__(self):
        self.size = 10
        self.table = [[] for _ in range(self.size)]

    def _hash_function(self, key):
        # Compute the hash code using a suitable hash function
        # Map the hash code to an index within the table size
        return hash(key) % self.size

    def insert(self, key, value):
        index = self._hash_function(key)
        for pair in self.table[index]:
            if pair[0] == key:
                pair[1] = value
                return
        self.table[index].append([key, value])

    def get(self, key):
        index = self._hash_function(key)
        for pair in self.table[index]:
            if pair[0] == key:
                return pair[1]
        return None

    def remove(self, key):
        index = self._hash_function(key)
        for i, pair in enumerate(self.table[index]):
            if pair[0] == key:
                del self.table[index][i]
                return
```