HashTable and SymbolTable Documentation

HashTable

The **HashTable** class is an implementation of a basic hash table data structure. It allows you to store key-value pairs and provides methods for adding, retrieving, and checking the existence of elements. Here's a detailed explanation of its methods and properties:

def __init__(self, size=100, load_factor=0.7):

- Initializes a new hash table with the specified **size** and **load_factor**.
- **size** (optional): The initial size of the hash table. Defaults to 100.
- **load_factor** (optional): The load factor at which the hash table will be resized. Defaults to 0.7.

Public Methods

resize(self, new_size)

- Resizes the hash table to the specified **new_size**.

hash(self, key)

- Computes the hash value for the given **key**. The hash function is based on the type of the key (int or str).

getSize(self)

- Returns the current size of the hash table.

getHashValue(self, key)

- Returns the hash value for a given **key**.

add(self, key, value=None)

- Adds a key-value pair to the hash table.
- **key**: The key to be added.
- **value** (optional): The value associated with the key. If not provided, the value is set to **None**.

contains(self, key)

- Checks if the hash table contains a key.
- Returns **True** if the key exists in the table; otherwise, **False**.

getPosition(self, key)

- Returns the position (index) of a key in the hash table. If the key does not exist, it returns **(-1, -1)**.

__str__(self)

- Returns a string representation of the hash table, showing its internal structure.

SymbolTable

The **SymbolTable** class is a higher-level wrapper around the **HashTable** class, providing a more specific interface for symbol table operations. A symbol table is often used in programming language compilers to store identifiers and their associated information. Here's the documentation for the **SymbolTable** class:

Constructor

def __init__(self, size=100):

- Initializes a new symbol table with the specified size. Internally, it creates a **HashTable** with the given size.

Public Methods

add(self, key, value=None)

- Adds a key-value pair to the symbol table. This method delegates to the **HashTable**'s **add** method.

has(self, key)

- Checks if a key exists in the symbol table. It delegates to the **HashTable**'s **contains** method and returns **True** if the key exists; otherwise, **False**.

getPosition(self, key)

- Returns the position (index) of a key in the symbol table. If the key does not exist, it returns **(-1, -1)**. This method also delegates to the **HashTable**.

__str__(self)

- Returns a string representation of the symbol table, showing its internal structure. This method delegates to the **HashTable**'s **__str__** method.

Example Usage

The code snippet provided in the documentation demonstrates the usage of the **HashTable** and **SymbolTable** classes. It shows how to add, check for the existence of keys, and retrieve positions for keys within the symbol table. The example illustrates both integer and string keys.

Make sure to import the classes correctly to use them in your code:

from your_module_name import HashTable, SymbolTable

Please replace **your_module_name** with the actual module name where these classes are defined.