Welcome to the CoGrammar Hash Tables Lecture

The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated moderators answering questions.

CoGrammar

Coding Interview Workshop Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly. **(Fundamental British Values: Mutual Respect and Tolerance)**
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: **Questions**

Coding Interview Workshop Housekeeping cont.

- For all **non-academic questions**, please submit a query: www.hyperiondev.com/support

- Report a **safeguarding** incident: www.hyperiondev.com/safeguardreporting

- We would love your **feedback** on lectures: Feedback on Lectures

Skills Bootcamp 8-Week Progression Overview

Fulfil 4 Criteria to Graduation

✅ Criterion 1: Initial Requirements

Timeframe: First 2 Weeks
Guided Learning Hours (GLH): Minimum of 15 hours
Task Completion: First four tasks

Due Date: 24 March 2024

✅ Criterion 2: Mid-Course Progress

60 Guided Learning Hours

Data Science - **13 tasks**
Software Engineering - **13 tasks**
Web Development - **13 tasks**

Due Date: 28 April 2024

CoGrammar

Skills Bootcamp Progression Overview

✅ Criterion 3: Course Progress

Completion: All mandatory tasks, including Build Your Brand and resubmissions by study period end
Interview Invitation: Within 4 weeks post-course
Guided Learning Hours: Minimum of 112 hours by support end date (10.5 hours average, each week)

✅ Criterion 4: Demonstrating Employability

Final Job or Apprenticeship Outcome: Document within 12 weeks post-graduation
Relevance: Progression to employment or related opportunity

CoGrammar

SKILLS FOR LIFE

SKILLS BOOTCAMPS

Department for Education

CoGrammar

Hash Tables

May 2024

Learning Objectives

- ❖ Explain the concept of **hashing** and how it facilitates efficient **data lookup**, **storage**, and **retrieval**.
- ❖ Illustrate how hash tables are **implemented** and the underlying mechanisms that allow for **efficient data access**, including the use of **hash functions** and **handling collisions**.
- ❖ Demonstrate the process of **inserting**, **deleting**, and **retrieving data** from a hash table in Python and JavaScript, emphasizing the importance of **choosing appropriate hash functions**.

CoGrammar

Learning Objectives

- ❖ Analyze the **impact of hash function selection** on the performance of hash tables, including the **handling of collisions** and the importance of **load factor**.
- ❖ Evaluate **real-world applications** of hash tables in software development and data science, identifying scenarios where **hash tables offer significant advantages** over other data structures.

What is the primary purpose of a hash function in a hash table?

A. To encrypt data.
B. To map data to unique hash codes.
C. To sort data in ascending order.
D. To reduce data size.

CoGrammar

Correct Answer: B

- ❖ In a hash table, key-value pairs are stored in an associative array.
- ❖ The position where the pair will be stored is determined using the hash function, which takes in the pair's key.
- ❖ The hash function can be any deterministic function which maps a key to the range of indices but the aim is that the function is efficient, it distributes keys uniformly and multiple keys being mapped to the same index is avoided.

What is a hash collision?

A. A security breach in a hash table.
B. When two different inputs produce the same hash code.
C. A failure in the hash function.
D. A type of data compression.

Correct Answer: B

- ❖ Since we have no way of knowing what values will be stored in a hash table, our hash function may map unique pairs to the same position in the associative array.
- ❖ This is known as a hash collision.
- ❖ We try to avoid this scenario as much as possible, but we have to account for collision resolution in our hash table implementations.

How do Python dictionaries and JavaScript Maps primarily store and access data?

A. Using an array-based structure.
B. Through a tree structure.
C. Using a hash table mechanism.
D. Via direct indexing.

CoGrammar

Correct Answer: C

- Maps in JavaScript (and Objects) and Dictionaries in Python are stored internally using hash tables.
- All the complexities involved in creating an efficient hash table are dealt with by JavaScript and Python.
- This is the easiest way to create hash tables in our code.

Efficient Spell-Checkers

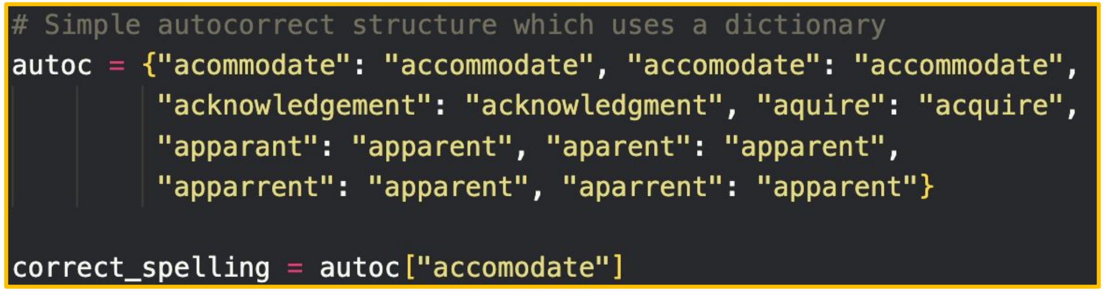
Consider spell-checking tools which are used on your devices. Suggestions and corrections need to be made as you are typing so correct spellings of words need to be looked up very quickly. A data structure containing pairs of common incorrect spelled words and the correct spelling, but this structure would be very big.

- ➢ What data structure can be used that will allow us to **create such a big data structure** efficiently and quickly?
- ➢ How can we store these pairs of words in a way that ensures that our data structure can be searched, **searched, common misspellings looked up** and the correct spellings retrieved very quickly?

CoGrammar

Example: Spell Checkers

The data structure known as a dictionary or map is a data structure that can easily solve our problem. Key-value pairs are stored and values are accessed using the key value.



```python
# Simple autocorrect structure which uses a dictionary
autoc = {"acommodate": "accommodate", "accomodate": "accommodate",
         "acknowledgement": "acknowledgment", "aquire": "acquire",
         "apparant": "apparent", "aparent": "apparent",
         "apparrent": "apparent", "aparrent": "apparent"}

correct_spelling = autoc["accomodate"]
```

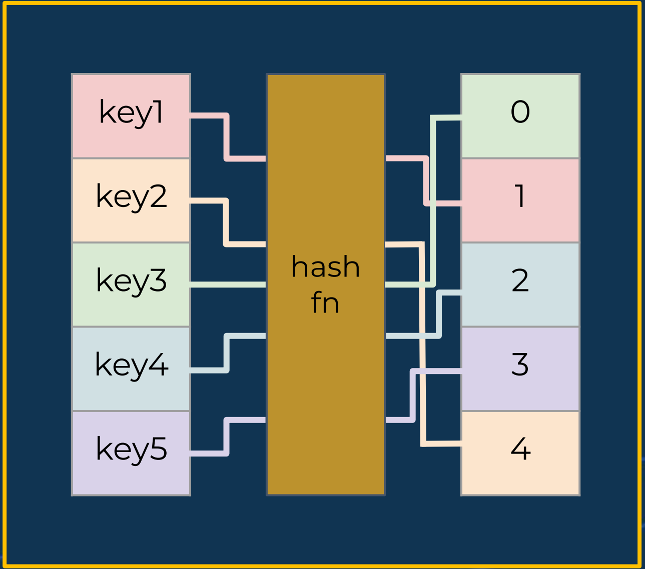
➢ It turns out that this data structure is very efficient, but how does it work? How is it so efficient?

CoGrammar

Hash Tables

A data structure that is used to store key-value pairs which uses a hash function to transform the key into the index of an associated array element where the data will be stored.

- Hash tables allow for **efficient and fast** insertions, deletions and look-ups due to the manner in which values are stored.
- Since the **index where the values is stored is a function of the value**, the position where the value must be stored is always known.



CoGrammar

Click here to visualize this data structure!

- **Hashing** refers to using the hash function to calculate **the hash** which is the index of the array element where the key-value pair will be stored.
- Each array element is called a **bucket or slot** and can store one or more key-value pairs.
- The **hash function** can be any **deterministic** function which maps a key to the range of indices but the aim is that the function is **efficient**, it distributes keys **uniformly** and **multiple keys being mapped to the same index** is avoided.
- The **load factor** is the ratio of the number of stored elements to the total number of buckets and is used to determine whether the table can be downsized to improve performance.

Hash Collisions

When two or more distinct keys hash to the same array index.

- Although we try to avoid hash collisions, they are inevitable so we have to implement a **collision-resolution** process.
- When searching or storing data, a hash collision **increases the time complexity** of the task.

Operation	Average Case	Worst Case
Insert	O(1)	O(n)
Lookup/Search	O(1)	O(n)
Delete	O(1)	O(n)

CoGrammar

Collision-Resolution

- **Chaining**: Linked Lists are used for each bucket, so multiple key-value pairs can be stored in the same bucket.
  - Pros: Simple to implement, dynamic resizing
  - Cons: Memory overhead implications, space inefficiencies
- **Linear Probing**: If there is a collision, place the pair in the next available slot in the hash table (linearly).
  - Pros: Simple to implement, memory-efficient
  - Cons: Primary clustering (large blocks of occupied elements), clustering results in more time inefficiencies

CoGrammar

Applications of Hash Tables

Web Development

- ❖ **Caching**: Hash tables are used in caching mechanisms to store frequently accessed data, improving the performance of web applications by reducing database queries or expensive computations.
- ❖ **URL Routing**: In web frameworks, hash tables can be used to map URL routes to corresponding handlers or controllers, enabling efficient routing and navigation within web applications.

Applications of Hash Tables

Data Science

- **Indexing**: Hash tables are used for indexing large datasets, allowing quick access to specific records or subsets of data based on their hashed keys.
- **Duplicate Detection**: Hash tables help in detecting and eliminating duplicate entries in datasets, streamlining data preprocessing tasks in data science pipelines.

CoGrammar

Applications of Hash Tables

Software Engineering

- ❖ **Symbol Tables**: Hash tables are employed in compilers and interpreters to implement symbol tables, storing identifiers, variables, and other program symbols along with their associated metadata.
- ❖ **Cache Management**: Similar to web development, hash tables are used for caching frequently accessed data or results of expensive computations in software systems to improve performance.

CoGrammar

Built-in Hash Tables

- **Dictionaries** (Python) and **Maps** (JavaScript) are data structures which store key-value pairs.
- They are implemented internally using a **hash table**, using built-in hash functions.
- They are the easiest way to implement a hash table, since all the complexities involved in ensuring efficiency is abstracted away.
- A custom hash table would be primarily used when a **custom hash function** needs to be implemented.
- This is useful in the case where keys are of **custom or non-hashable data types.**

CoGrammar

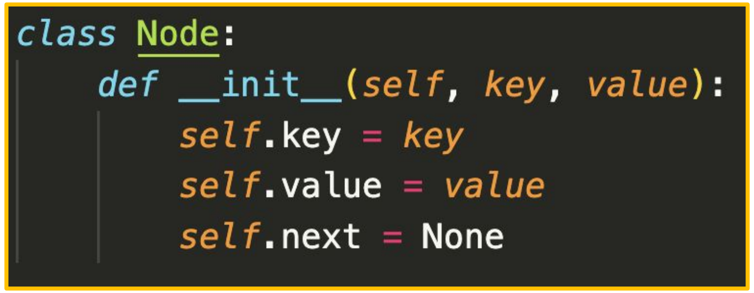
Let’s Breathe!

Let’s take a small break before moving on to the next topic.

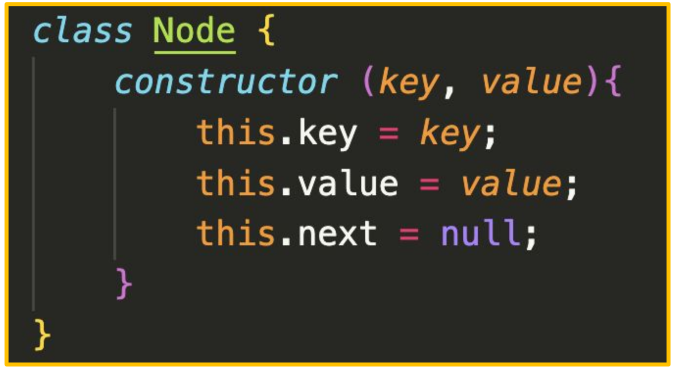
CoGrammar

Hash Table Implementation

For this implementation, we'll use chaining to handle hash collisions. We use an array to store Nodes containing the key-value pairs.



```python
class Node:
    def __init__(self, key, value):
        self.key = key
        self.value = value
        self.next = None
```

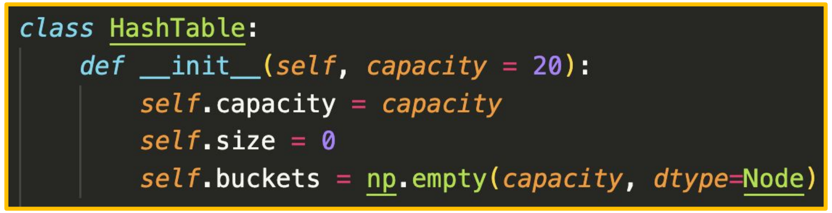


```javascript
class Node {
    constructor (key, value){
        this.key = key;
        this.value = value;
        this.next = null;
    }
}
```

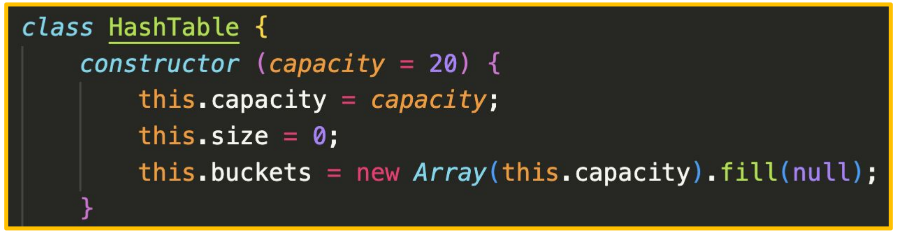
CoGrammar

Hash Table Implementation

For maximum efficiency, we should constantly resize the hash table based on the load but this isn't included in our implementation since we'll be using the hash table for small datasets.



```python
class HashTable:
    def __init__(self, capacity = 20):
        self.capacity = capacity
        self.size = 0
        self.buckets = np.empty(capacity, dtype=Node)
```

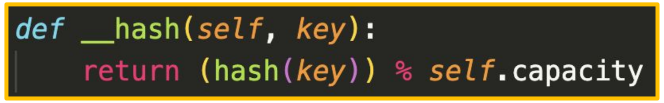


```javascript
class HashTable {
    constructor (capacity = 20) {
        this.capacity = capacity;
        this.size = 0;
        this.buckets = new Array(this.capacity).fill(null);
    }
```

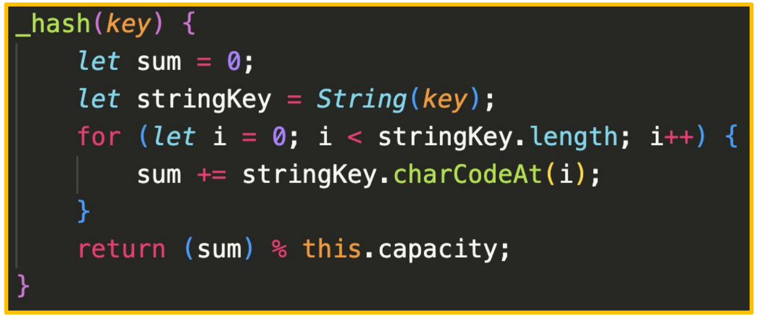
CoGrammar

Hash Table Implementation

In Python, we will be using the built-in hash function which sufficiently spreads our elements and our keys are Hashable. In JavaScript, we implement our own hash function by summing the UTF-16 codes of each character.



```python
def __hash(self, key):
    return (hash(key)) % self.capacity
```

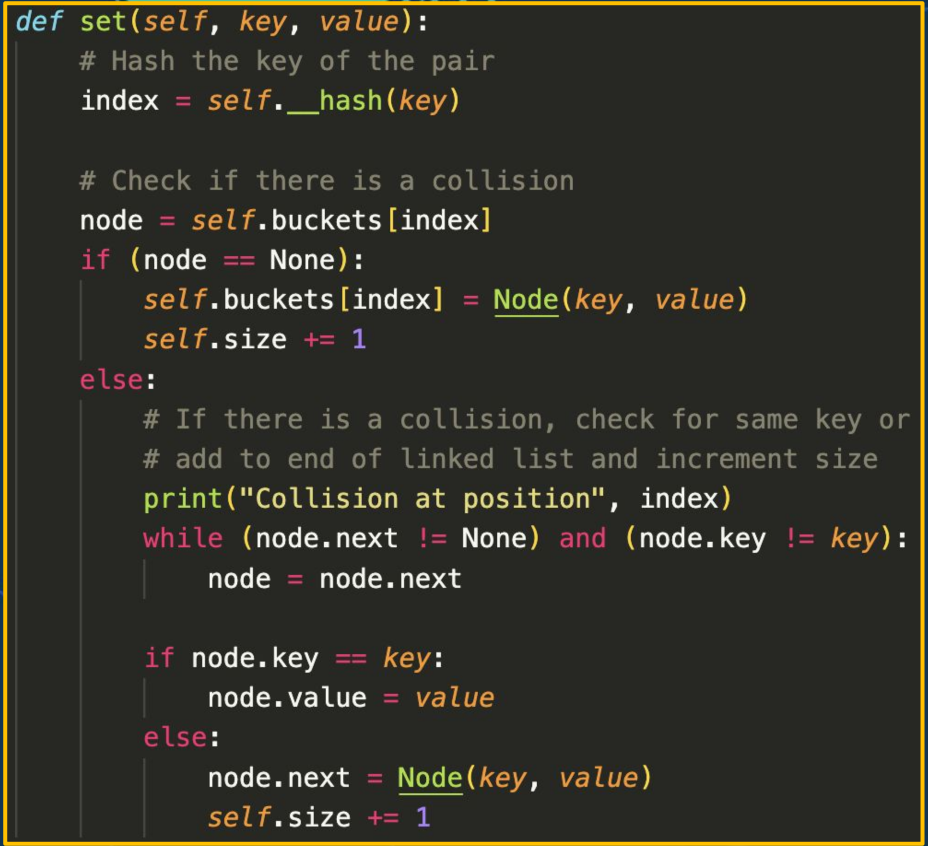


```javascript
_hash(key) {
    let sum = 0;
    let stringKey = String(key);
    for (let i = 0; i < stringKey.length; i++) {
        sum += stringKey.charCodeAt(i);
    }
    return (sum) % this.capacity;
}
```

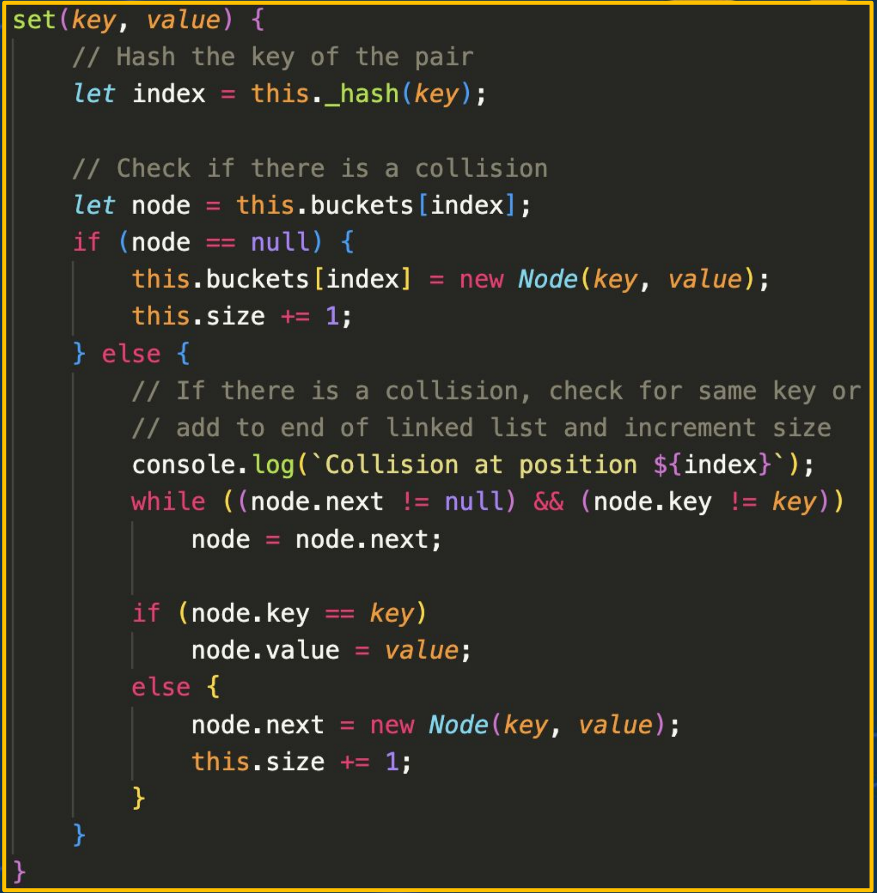
CoGrammar

Hash Table Implementation

We implement three functions: **set**, **get** and **remove**. Other useful functions we might want to implement as well include a **resize** function, which can be used to manage the load factor of our hashtable, and a **print** function, which will allow us to visualise the hashtable

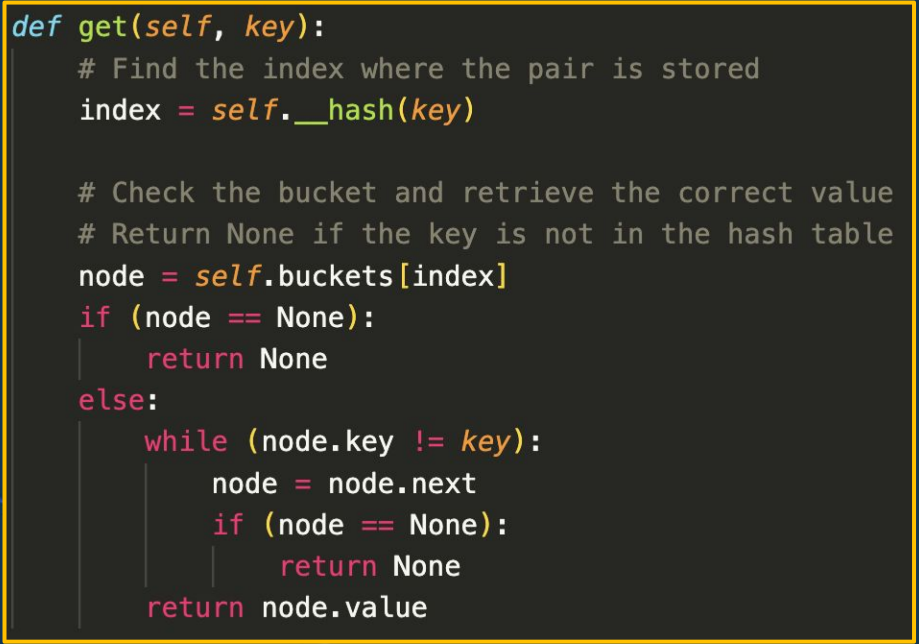


```python
def set(self, key, value):
    # Hash the key of the pair
    index = self.__hash(key)

    # Check if there is a collision
    node = self.buckets[index]
    if (node == None):
        self.buckets[index] = Node(key, value)
        self.size += 1
    else:
        # If there is a collision, check for same key or
        # add to end of linked list and increment size
        print("Collision at position", index)
        while (node.next != None) and (node.key != key):
            node = node.next

        if node.key == key:
            node.value = value
        else:
            node.next = Node(key, value)
            self.size += 1
```

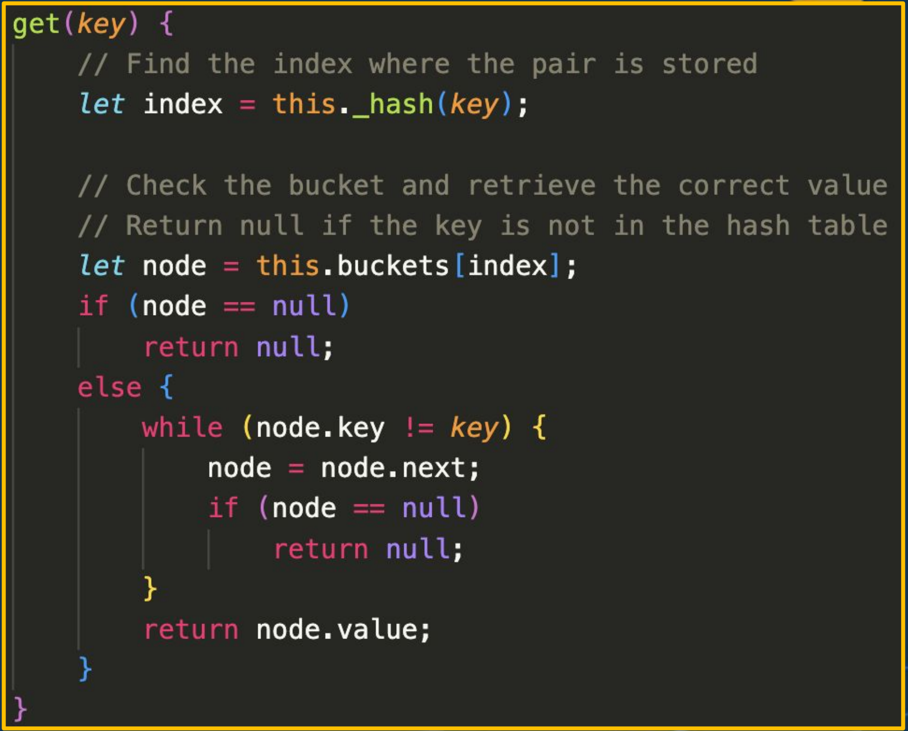


```javascript
set(key, value) {
    // Hash the key of the pair
    let index = this._hash(key);

    // Check if there is a collision
    let node = this.buckets[index];
    if (node == null) {
        this.buckets[index] = new Node(key, value);
        this.size += 1;
    } else {
        // If there is a collision, check for same key or
        // add to end of linked list and increment size
        console.log(`Collision at position ${index}`);
        while ((node.next != null) && (node.key != key))
            node = node.next;

        if (node.key == key)
            node.value = value;
        else {
            node.next = new Node(key, value);
            this.size += 1;
        }
    }
}
```

CoGrammar



```python
def get(self, key):
    # Find the index where the pair is stored
    index = self.__hash(key)

    # Check the bucket and retrieve the correct value
    # Return None if the key is not in the hash table
    node = self.buckets[index]
    if (node == None):
        return None
    else:
        while (node.key != key):
            node = node.next
            if (node == None):
                return None
        return node.value
```

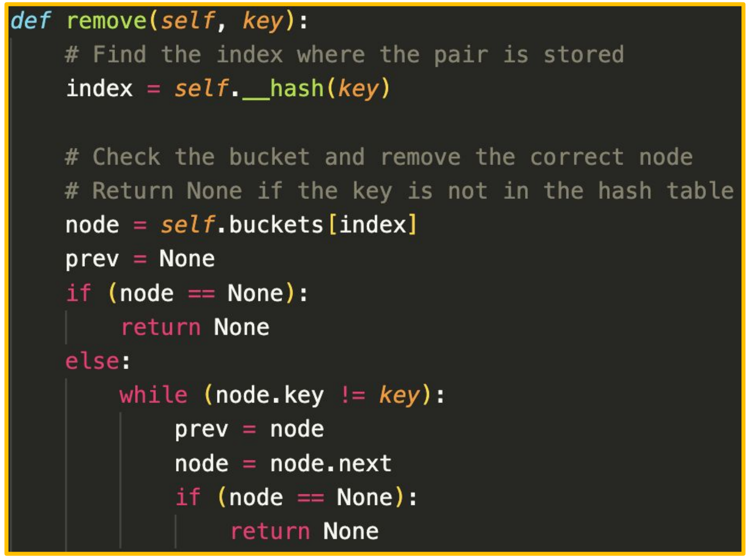


```javascript
get(key) {
    // Find the index where the pair is stored
    let index = this._hash(key);

    // Check the bucket and retrieve the correct value
    // Return null if the key is not in the hash table
    let node = this.buckets[index];
    if (node == null)
        return null;
    else {
        while (node.key != key) {
            node = node.next;
            if (node == null)
                return null;
        }
        return node.value;
    }
}
```

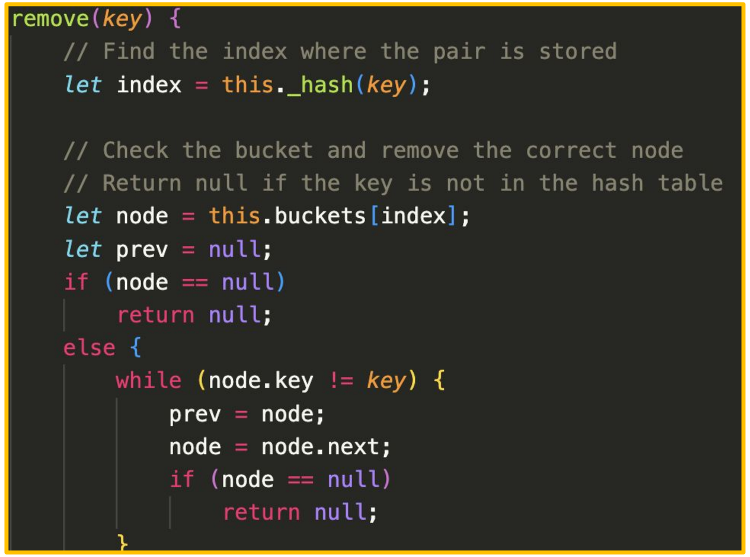
CoGrammar

Hash Table Implementation

To remove elements, first we find the correct element. The code is the same as the get function.



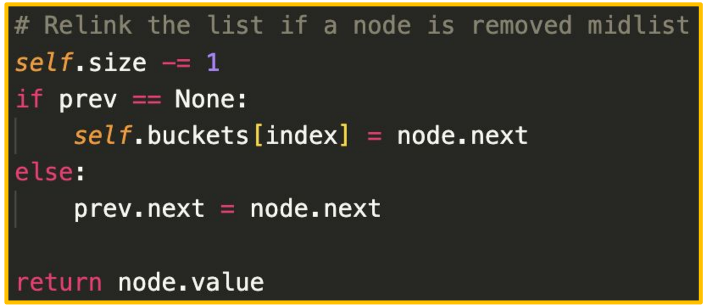
```python
def remove(self, key):
    # Find the index where the pair is stored
    index = self.__hash(key)

    # Check the bucket and remove the correct node
    # Return None if the key is not in the hash table
    node = self.buckets[index]
    prev = None
    if (node == None):
        return None
    else:
        while (node.key != key):
            prev = node
            node = node.next
            if (node == None):
                return None
```



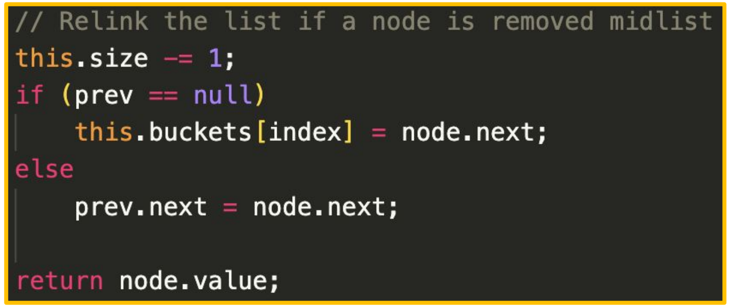
```javascript
remove(key) {
    // Find the index where the pair is stored
    let index = this._hash(key);

    // Check the bucket and remove the correct node
    // Return null if the key is not in the hash table
    let node = this.buckets[index];
    let prev = null;
    if (node == null)
        return null;
    else {
        while (node.key != key) {
            prev = node;
            node = node.next;
            if (node == null)
                return null;
        }
```

Hash Table Implementation

Next, we remove the element and ensure the linked list is entact.



```python
# Relink the list if a node is removed midlist
self.size -= 1
if prev == None:
    self.buckets[index] = node.next
else:
    prev.next = node.next

return node.value
```



```javascript
// Relink the list if a node is removed midlist
this.size -= 1;
if (prev == null)
    this.buckets[index] = node.next;
else
    prev.next = node.next;

return node.value;
```

CoGrammar

What is a common method to resolve hash collisions?

A. Linear probing.
B. Increasing the hash table size.
C. Encrypting the hash codes.
D. Sorting the buckets.

CoGrammar

Correct Answer: A

- ❖ Linear Probing is a Collision-Resolution method which involves looking for an open element in the hash table when a collision occurs.
- ❖ Increasing the hash table size is not an effective collision resolution method since this will increase the load factor of the hash table and affect performance.

When would a custom hash table implementation be preferred over a dictionary or map?

A. For smaller datasets.
B. For greater control over collision handling.
C. When using non-hashable data types.
D. For automatic memory management.

CoGrammar

Correct Answer: C

- ❖ Some keys cannot easily be hashed using simple hash functions.
- ❖ Objects cannot be used as keys in Dictionaries and Maps.
- ❖ In these cases, a custom hash table implementation can be used to create a hash function which better suits the needs of the hash function.

CoGrammar

Portfolio Assignment: SE

Implementing a Hash Table for Symbol Table Management

Objective: Develop a command-line tool in Python for managing symbol tables using a hash table data structure. The tool should allow users to insert, delete, and retrieve symbols, and handle collisions using appropriate techniques.

CoGrammar

Portfolio Assignment: SE

Requirements:

➢ Implement functions for inserting symbols into the hash table, deleting symbols, and retrieving symbols based on their names.

➢ Provide a command-line interface for interacting with the symbol table, including options for inserting, deleting, and retrieving symbols.

➢ Include error handling for cases where symbols cannot be inserted or retrieved due to collisions or other issues.

➢ Provide a README file that explains how to use the tool, including examples of valid commands and their expected outputs.

Portfolio Assignment: DS

Analyzing Movie Ratings using a Hash Table

Objective: Develop a Python script that analyzes movie ratings data using a hash table. The script should read a dataset containing movie ratings, store the ratings in a hash table, and perform various analyses such as finding the average rating per movie and identifying the highest rated movies.

CoGrammar

Portfolio Assignment: DS

Requirements:

- Read a CSV file containing movie ratings data and populate a hash table with this data.
- Implement functions to calculate the average rating per movie and identify the highest rated movies based on the ratings stored in the hash table.
- Provide a command-line interface for running the script and displaying the analysis results.
- Include error handling for cases where the input file is not found or the data is not valid.
- Provide a README file that explains how to run the script and includes a sample input file.

Portfolio Assignment: WD

Implementing a Hash Table for URL Shortening

Objective: Develop a web application that shortens long URLs using a hash table data structure. The application should allow users to input a long URL, generate a shortened URL using the hash table, and redirect to the original URL when the shortened URL is accessed.

CoGrammar

Portfolio Assignment: WD

Requirements:

- ➢ Implement the web application using HTML, CSS, and JS.
- ➢ Implement a hash table data structure in JavaScript to store the mappings between shortened URLs and original URLs.
- ➢ Use a hash function to generate hash codes for the original URLs and handle collisions appropriately.
- ➢ Include error handling for cases where the shortened URL does not exist or when the original URL is not valid.
- ➢ Provide a README file that explains how to run the web application locally. You may use React.js to manage packages, dependencies and creating a server.

Summary

Hash Tables

- ➢ Data structure which stores key-value pairs efficiently using a hash function to determine the index of an array where the pair will be stored
- ➢ Hash functions have to be deterministic and simple to compute

Hash Collisions

- ➢ When the hash function maps a pair to an element where a pair has already been stored, this is known as a hash collision
- ➢ Hash collisions can be handled using linear probing or chaining

Dictionaries and Maps

- ➢ Dictionaries (Python) and Maps (JS) are implemented using a hash table which makes use of a built-in hash function.

CoGrammar

Additional Resources

- Hash Tables - Section in the textbook “Algorithms” by Robert Sedgewick and Kevin Wayne
- Introduction to Hash Tables - Specifically for DS students
- Comprehensive overview of Hash Tables - Goes over all the topics covered as well as providing links to find out more
- Hash Tables in Python - Theory and Implementation of Hash Tables in Python
- Hash Tables in JavaScript - Theory and Implementation of Hash Tables in JavaScript

CoGrammar

CoGrammar

Q & A SECTION

Please use this time to ask any questions relating to the topic, should you have any.

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar