

# 파이썬으로 배우는 데이터 구조



한동대학교  
전산전자공학부  
김영섭 교수



# 학습 목표

---

해시(Hash) 테이블과 해시 함수를  
이해하고 특징들을 학습한다

# **Data Structures in Python**

## **Chapter 6**

- Hash Table
- Collision Resolution
- Double Hashing & Rehashing
- HashMap Coding

# Agenda & Readings

---

- Agenda
  - Hashing
  - Hash Table
  - Hash Function
- Reference:
  - Problem Solving with Algorithms and Data Structures
  - Chapter 5 - Hashing

# Overview

- Hashing or Hash Table Data Structure:

- Data structures so far

Array of size n	unsorted list	sorted array	Trees BST – average AVL – worst	Heap, Priority Queue	<b>Hashing</b>
insert	find+O(1)	O(n)	O(log n)	O(log n)	
find	O(n)	O(log n)	O(log n)	O(log n)	
remove	find+O(1)	O(n)	O(log n)	O(log n)	

# Overview

- Hashing or Hash Table Data Structure:  
supports insertion, deletion and search in average case constant time  **$O(1)$** .

- Data structures so far

Array of size $n$	unsorted list	sorted array	Trees BST – average AVL – worst	Heap, Priority Queue	<b>Hashing</b>
insert	find+ $O(1)$	$O(n)$	$O(\log n)$	$O(\log n)$	$O(1)$
find	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(1)$
remove	find+ $O(1)$	$O(n)$	$O(\log n)$	$O(\log n)$	$O(1)$

# Overview

---

- Hashing or Hash Table Data Structure:  
supports insertion, deletion and search in average case constant time  **$O(1)$** .
- **Hash table**
  - It is data structure that stores **key-value pairs**.
  - The key is sent to a **hash function** that performs arithmetic operations on it.
  - The result is called **hash value** that is the **index of the key-value pair** in the **hash table**.
- **Hash function**
  - `hash(key) → integer value`
  - `hash("string key") → integer value`

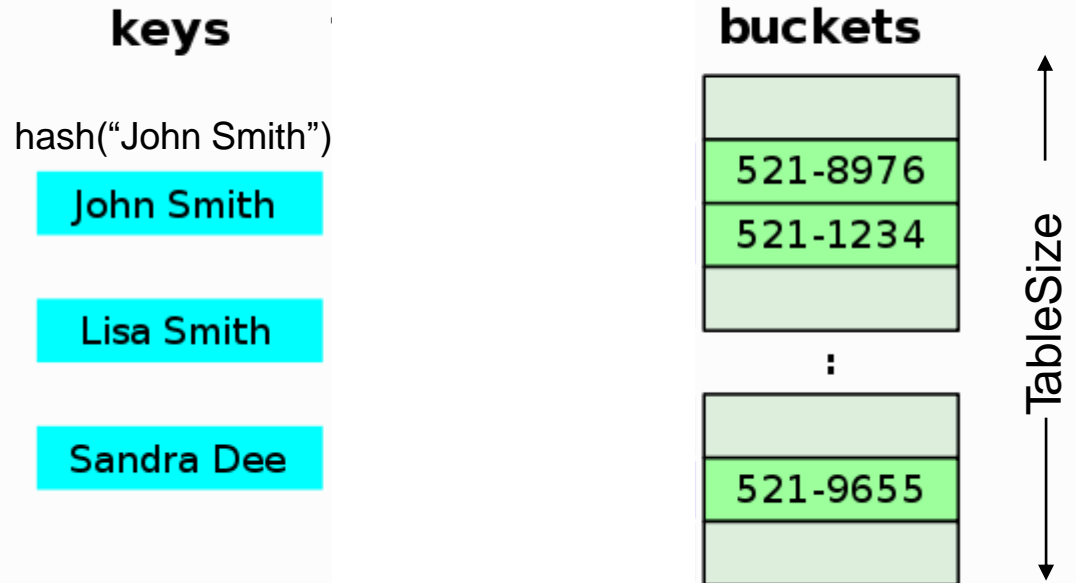
# Hash Table

- Hash table is an array of fixed size elements

Let us suppose that there are one billion of names and numbers.

- Find, insert, and remove a number by a given name in  **$O(1)$** .

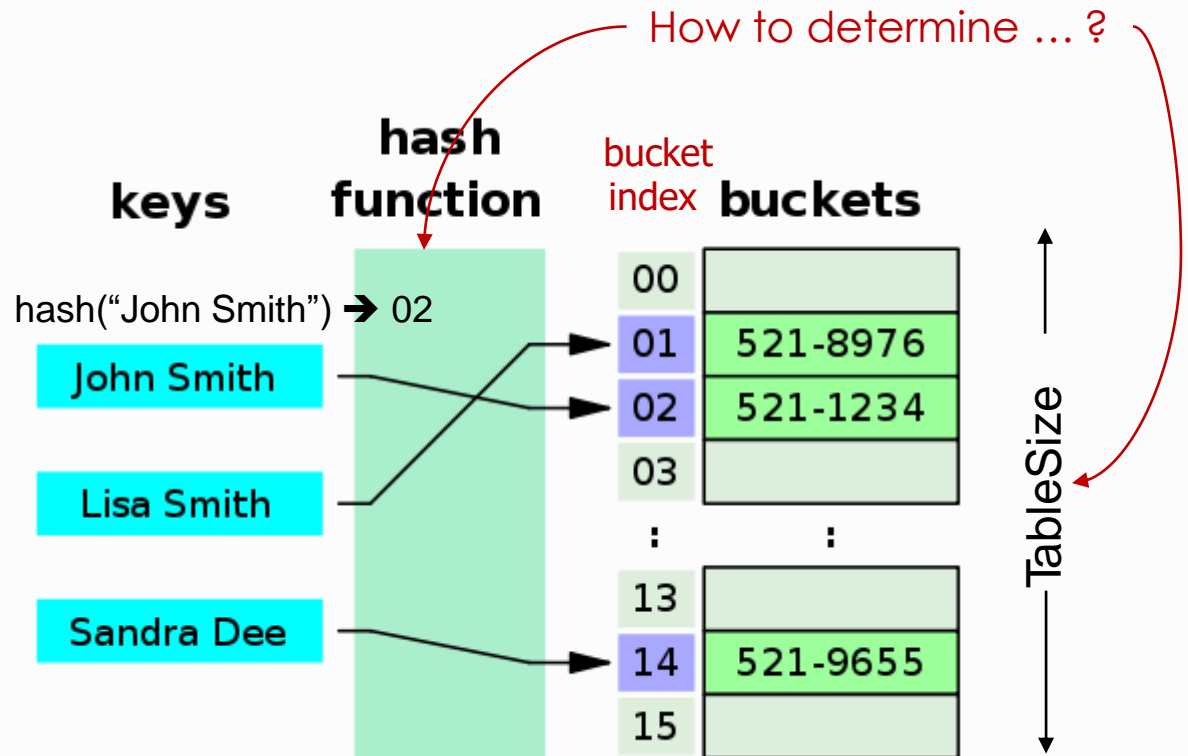
Time Complexity





# Hash Table

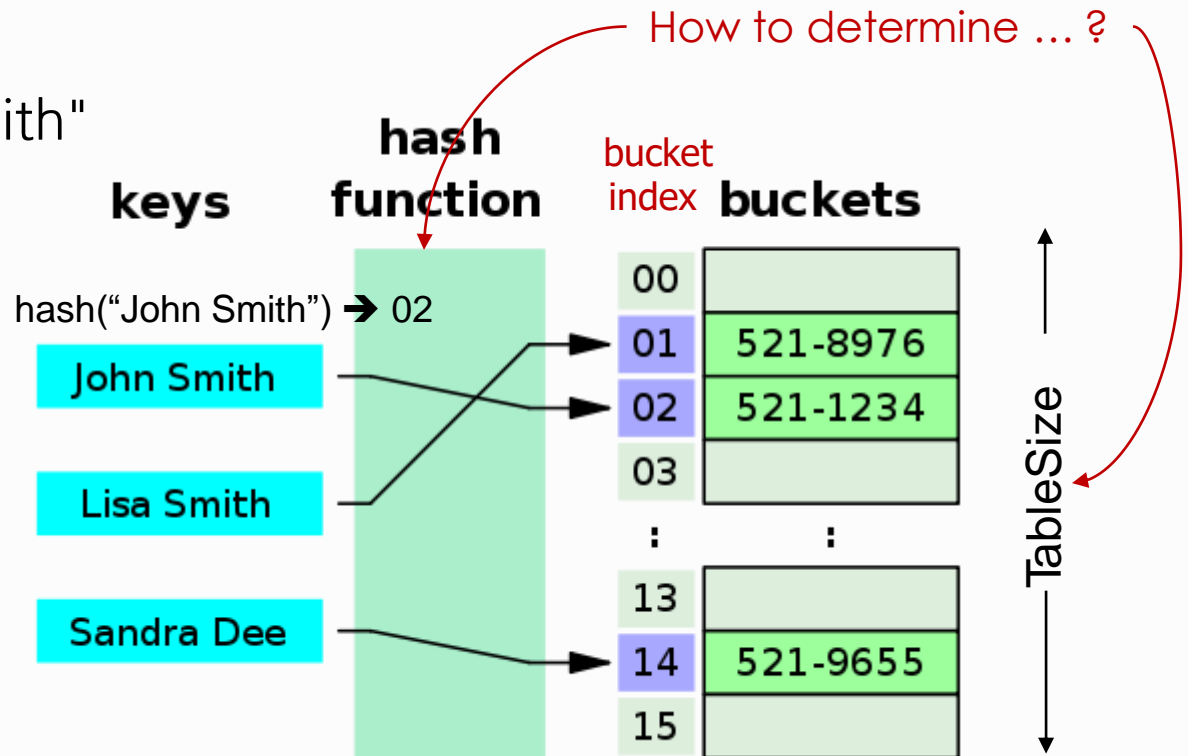
- Hash table is an array of fixed size elements
- Array elements indexed by a key mapped to a bucket index[0 .. TableSize-1]
- Mapping from key to index using hash(), hash function
  - e.g., hash("John Smith") → 02



# Hash Table

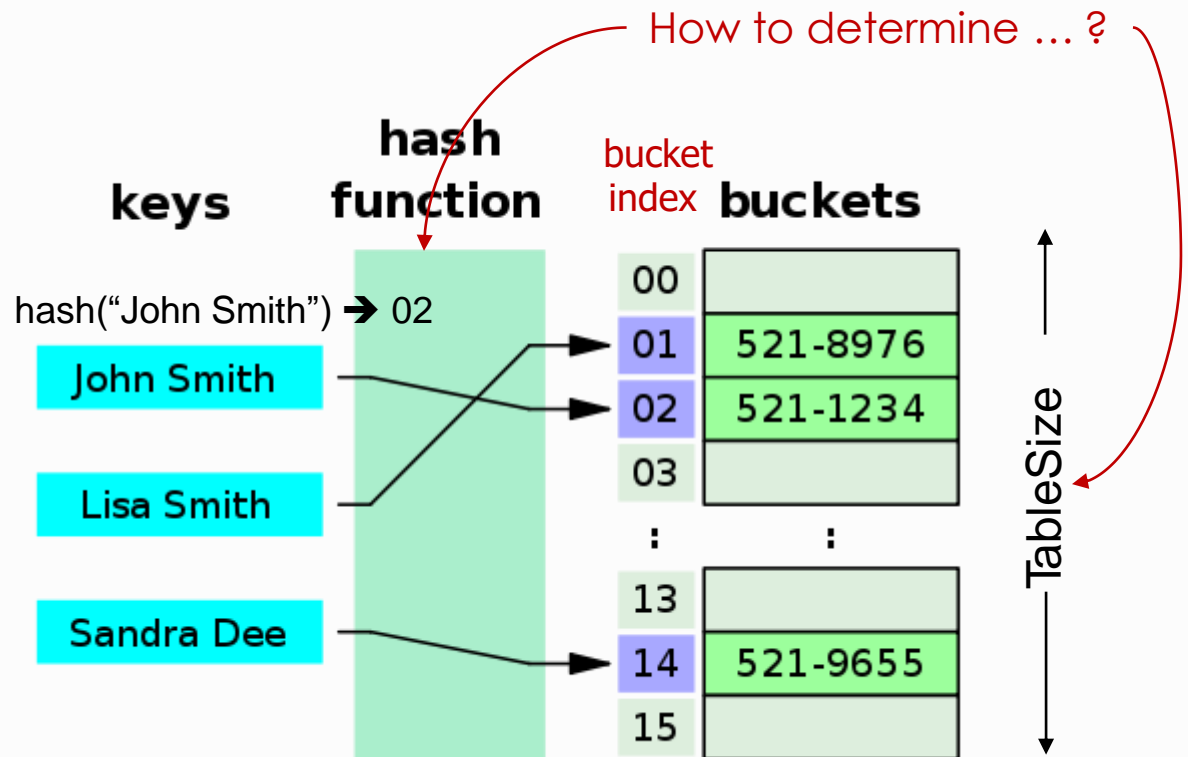
- insert
  - `HashTable[hash("John Smith")] = <"John Smith", 521-1234>`
- remove
  - `HashTable[hash("John Smith")] = None`
- find
  - `HashTable[hash("John Smith")]` returns the element hashed for "John Smith"

What happens  
if `hash("John Smith") == hash("Joe Blow")`?  
"Collision"



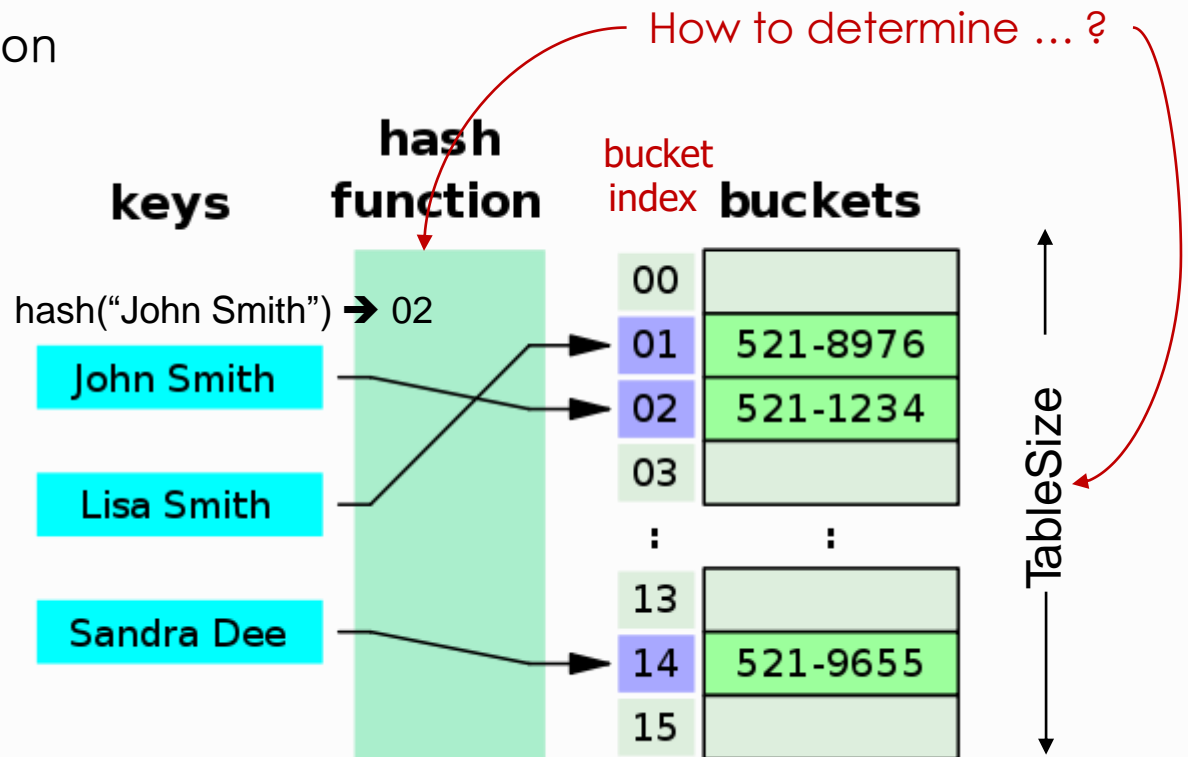
# Hash Table

- Factors affecting Hash Table Design
  - Hash function
  - Table size - Usually fixed at the start
  - Collision handling schemes - Array or Linked List



# Hash Function

- It maps an element's key into a valid hash table index
  - $\text{hash}(\text{key}) \rightarrow \text{hash table index}$
- Note that this is (slightly) different from saying:
  - $\text{hash}(\text{string}) \rightarrow \text{int}$
  - Because the key can be of any type
    - e.g., " $\text{hash}(\text{int}) \rightarrow \text{int}$ " is also a hash function



# Hash Function Properties

---

- It maps an element's key into a valid hash table index
  - $\text{hash}(\text{key}) \rightarrow \text{hash table index}$
- It maps key to integer
  - Constraint: Integer should be between **[0, TableSize-1]**
- A hash function can result in a many-to-one mapping (causing collision)
  - Collision occurs when hash function maps two or more keys to same array index
- Collisions **cannot** be avoided but its chances can be reduced using a "good" hash function

## Hash Function - Effective use of table size

---

- Simple hash function (assume integer keys)
  - $\text{hash}(\text{Key}) = \text{Key} \% \text{TableSize}$
- For random keys,  $\text{hash}()$  distributes keys evenly over table
  - What if `TableSize = 100` and keys are ALL multiples of 10?
  - Better if `TableSize` is a **prime number**


## Hash Function Example: String Keys

---

- Using a very simple function to map strings to integers:
  - Add up character ASCII values (0-255) to produce integer keys
    - e.g., "abcd" = 97 + 98 + 99 + 100 = 394
    - $\text{hash}(\text{"abcd"}) = 394 \% \text{TableSize}$
- Potential problems:
  - Anagrams will map to the same index
    - $\text{hash}(\text{"abcd"}) = \text{hash}(\text{"dbac"})$
  - Small strings may not use all of table
    - $\text{strlen}(s) * 255 < \text{TableSize}$
  - Time proportional to length of the string

## Hash Function Example: String Keys

---

- Another approach:
  - Treat first 3 characters of string as base-27 integer (26 letters plus space)
    - e.g.,  $\text{Key} = s[0] + (27^1 * s[1]) + (27^2 * s[2])$
    - Better than previous approach because ...
- But, potential problems:
- - Apple
  - Apply
  - Appointment
  - Apricot collision



# Hash Function Example: String Keys

- Last approach:
  - Use all N characters of string as an N-digit and base-K number
  - Choose K to be prime number larger than number of different digits (characters)
    - i.e.,  $K = 29, 31, 37$
    - If  $L = \text{Length of string } S$ , then

$$\text{hash}(S) = \sum_{i=0}^{L-1} S[L-1-i] * 37^i \% \text{TableSize} \quad (1)$$

- Use **Horner's rule** to compute  $\text{hash}(S)$ .
  - Limit L for long strings
- Potential problems
  - Overflow
  - Larger runtime

```
# a hash function for strings
def hash(key, tablesize)
    code = 0
    for x in key: code = code * 37 + x
    code %= tablesize
    if code < 0: code += tablesize
    return code
```

## Summary

---

- Using a hash table we can, on average (if table large enough and hash function suitable), insert, delete and search for items in constant time -  **$O(1)$** .
- The **hash function** is the mapping between an item and the slot where the item is stored.
- A **collision** occurs when an item is mapped to an occupied slot.
- A **perfect hash function** is able to map  $m$  items into a table of size  $m$  with no collisions. Perfect hash functions are hard to come by.
- Handling collisions systematically is required - **collision resolution**.

# 학습 정리

- 1) 해시 테이블을 이용하면, 삽입/삭제/검색 작업을  $O(1)$ 으로 해결할 수 있다
- 2) 해시 함수가 반환하는 해시 값(Value)은 항상 정수이다
- 3) 해시 테이블을 구현할 때, 충돌을 최소화할 수 있는 해시 함수를 사용해야 한다

# 파이썬으로 배우는 데이터 구조

수고했습니다  
곧 다음 시간에  
다시 뵙겠습니다

