Algorithms and Data Structures

Set And Map

What is a Set

- A set is an unordered collection with no duplicate elements.
- Basic uses include membership testing and eliminating duplicate entries.
- Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.
- Underlying data structure is dictionary/hash table

Using the `for` Loop, `in`, and `not in` Operators With a Set

- A `for` loop can be used to iterate over elements in a set
  - General format: `for` *`item`* `in` *`set`*`:`
  - The loop iterates once for each element in the set
- The `in` operator can be used to test whether a value exists in a set
  - Similarly, the `not in` operator can be used to test whether a value does not exist in a set

What is a Hash Table

- A **hash table** organizes data so you can quickly look up values for a given key.
- Arrays are pretty similar to hash maps already. Arrays let you quickly look up the value for a given "key" . . . except the keys are called "indices," and we don't get to pick them—they're always sequential integers (0, 1, 2, 3, etc).
- Think of a hash map as a "hack" on top of an array to let us use flexible keys instead of being stuck with sequential integer “indices."
- In Python, this data structure is called Dictionary. You can only use immutable object as key (such as strings, numbers, or tuples).

Dictionaries

- Dictionary: object that stores a collection of data
  - Each element consists of a *key* and a *value*
    - Often referred to as *mapping* of key to value
    - Key must be an immutable object
  - To retrieve a specific value, use the key associated with it
  - Format for creating a dictionary

```
dictionary =
                {key1:val1, key2:val2}
```

Retrieving a Value from a Dictionary

- Elements in dictionary are unsorted
- General format for retrieving value from dictionary: `dictionary[key]`
  - If `key` in the dictionary, associated value is returned, otherwise, `KeyError` exception is raised
- Test whether a key is in a dictionary using the `in` and `not in` operators
  - Helps prevent `KeyError` exceptions

Adding Elements to an Existing Dictionary

- Dictionaries are mutable objects
- To add a new key-value pair:
- ```
  dictionary[key] = value
  ```
  - If key exists in the dictionary, the value associated with it will be changed
  ```

Deleting Elements From an Existing Dictionary

- To delete a key-value pair:
- `del dictionary[key]`
  - If key is not in the dictionary, `KeyError` exception is raised

Getting the Number of Elements and Mixing Data Types

- `len` function: used to obtain number of elements in a dictionary
- Keys must be immutable objects, but associated values can be any type of object
  - One dictionary can include keys of several different immutable types
- Values stored in a single dictionary can be of different types

Creating an Empty Dictionary and Using `for` Loop to Iterate Over a Dictionary

- To create an empty dictionary:
  - Use `{}`
  - Use built-in function `dict()`
  - Elements can be added to the dictionary as program executes
- Use a `for` loop to iterate over a dictionary
  - General format: `for` *`key`* `in` *`dictionary`*`:`

Some Dictionary Methods

- clear method: deletes all the elements in a dictionary, leaving it empty
  - Format: `dictionary.clear()`
- get method: gets a value associated with specified key from the dictionary
  - Format: `dictionary.get(key, default)`
    - default is returned if key is not found
  - Alternative to `[]` operator
    - Cannot raise `KeyError` exception

Some Dictionary Methods (cont'd.)

- items method: returns all the dictionaries keys and associated values
  - Format: `dictionary.items()`
  - Returned as a *dictionary view*
    - Each element in dictionary view is a tuple which contains a key and its associated value
    - Use a `for` loop to iterate over the tuples in the sequence
      - Can use a variable which receives a tuple, or can use two variables which receive key and value

Some Dictionary Methods (cont'd.)

- keys method: returns all the dictionaries keys as a sequence
  - Format: `dictionary.keys()`
- pop method: returns value associated with specified key and removes that key-value pair from the dictionary
  - Format: `dictionary.pop(key, default)`
    - default is returned if key is not found

Some Dictionary Methods (cont'd.)

- popitem method: returns a randomly selected key-value pair and removes that key-value pair from the dictionary
  - Format: `dictionary.popitem()`
  - Key-value pair returned as a tuple
- values method: returns all the dictionaries values as a sequence
  - Format: `dictionary.values()`
  - Use a `for` loop to iterate over the values

Some Dictionary Methods (cont'd.)

Table 9-1 Some of the dictionary methods

Method	Description
`clear`	Clears the contents of a dictionary.
`get`	Gets the value associated with a specified key. If the key is not found, the method does not raise an exception. Instead, it returns a default value.
`items`	Returns all the keys in a dictionary and their associated values as a sequence of tuples.
`keys`	Returns all the keys in a dictionary as a sequence of tuples.
`pop`	Returns the value associated with a specified key and removes that key-value pair from the dictionary. If the key is not found, the method returns a default value.
`popitem`	Returns a randomly selected key-value pair as a tuple from the dictionary and removes that key-value pair from the dictionary.
`values`	Returns all the values in the dictionary as a sequence of tuples.

Exercise

- Create a dictionary that keeps track of the USA's Olympic medal count. Each key of the dictionary should be the type of medal (gold, silver, or bronze) and each key's value should be the number of that type of medal the USA's won. Say, the USA has 33 gold medals, 17 silver, and 12 bronze. Create a dictionary saved in the variable medals that reflects this information.

Exercise

- Create a dictionary contacts for our class:
  - Use full name of students as keys
  - Use email address of students as values
  - practice the functions we learned so far

Exercise

- Update the value for “Phelps” in the dictionary swimmers to include his medals from the Rio Olympics by adding 5 to the current value (Phelps will now have 28 total medals). Do not rewrite the dictionary. Assume swimmers = {'Manuel':4, 'Lochte':12, 'Adrian':7, 'Ledecky':5, 'Dirado':4, 'Phelps':23}

Exercise

- Concatenate following dictionaries to create a new one. Sample Dictionaries:
- d1={1:10, 2:20}
- d2={3:30, 4:40}
- d3={5:50, 6:60}
- Expected Result : {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}
- HINT: You need to use list of dictionaries to loop over the three dictionaries using nested FOR loops. Do not use update or any other dictionary functions that we have not covered in the class.

Exercise

- Concatenate the following dictionaries to creata a new one. If a key repeats across the three dictionaries, you have to add the values corresponding to the key.
- d1={1:10, 2:20}
- d2={1:20, 3:30, 4:40}
- d3={5:50,6:60,3:60}
- Expected result: {1: 30, 2: 20, 3: 90, 4: 40, 5: 50, 6: 60}

Exercise

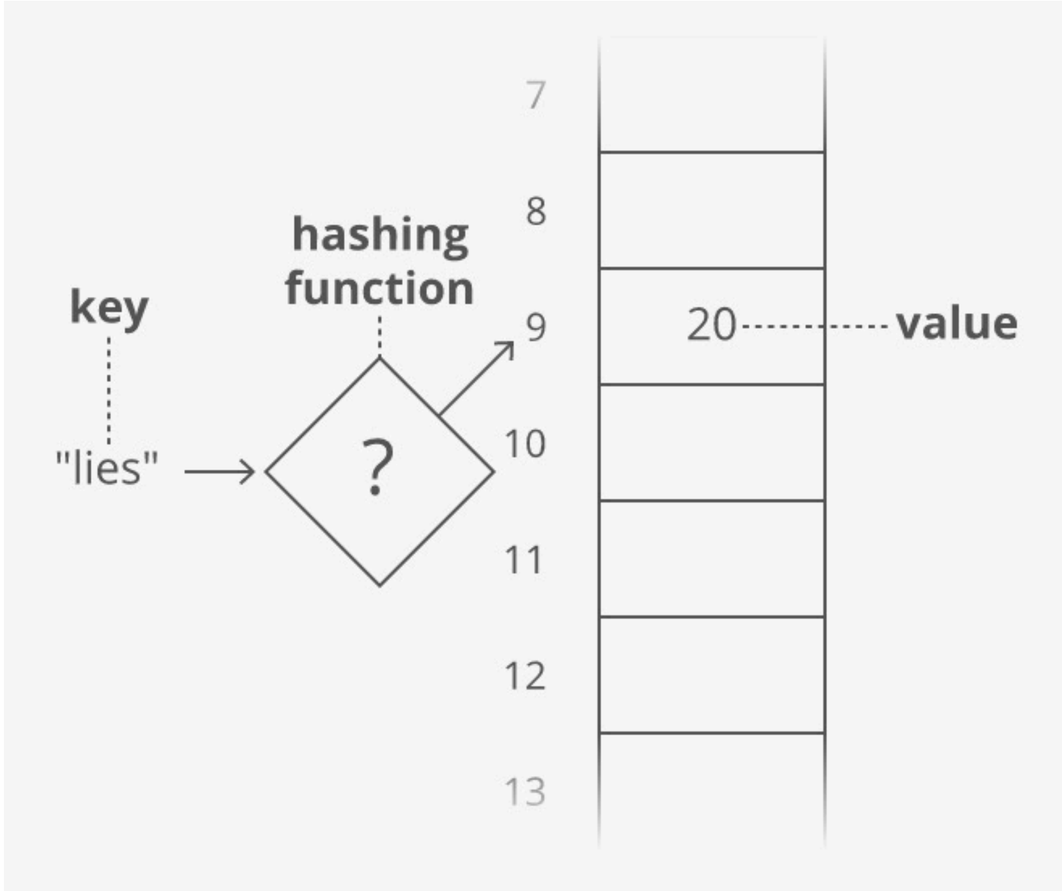
- Write function countLetter that takes a word as a parameter and returns a dictionary that tells us how many times each alphabet appears in the word (no need to account for absent alphabets). Develop your logic using dictionaries only. Save the count result as a dictionary. Display your output in the main() function

Exercise

- Many of you do not like the dummy value -1 we used in Fibonacci Dynamic Programming solution.
- Now we can use a Dictionary cheatsheet to do the job.

Hash Function

- All we need is a function to convert a key into an array index (an integer). That function is called a **hashing function**.



- To look up the value for a given key, we just run the key through our hashing function to get the index to go to in our underlying array to grab the value.

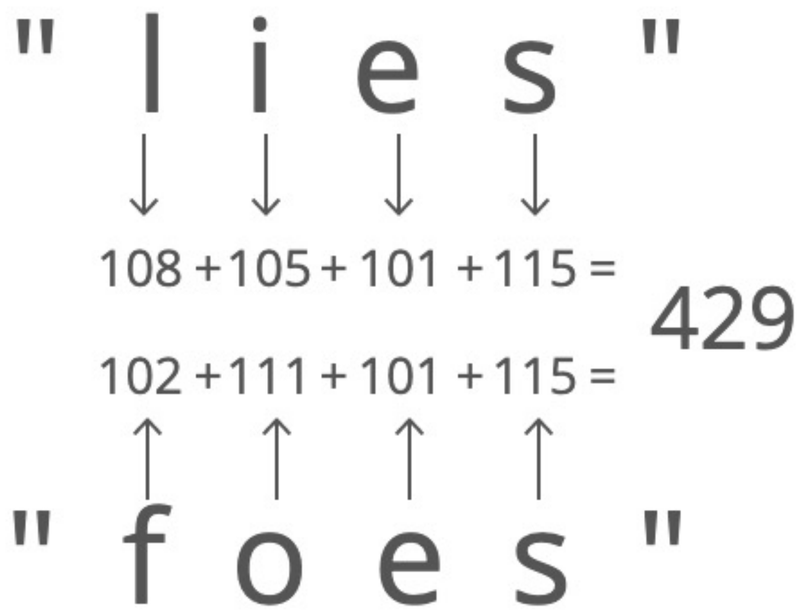
One Hash Method

- Grab the number value for each character and add those up.

- The result is 429. But what if we only have *30* slots in our array? We mod our sum by 30 ensures we get a whole number that's less than 30 (and at least 0):
  - 429 % 30 = 9

Hash collisions

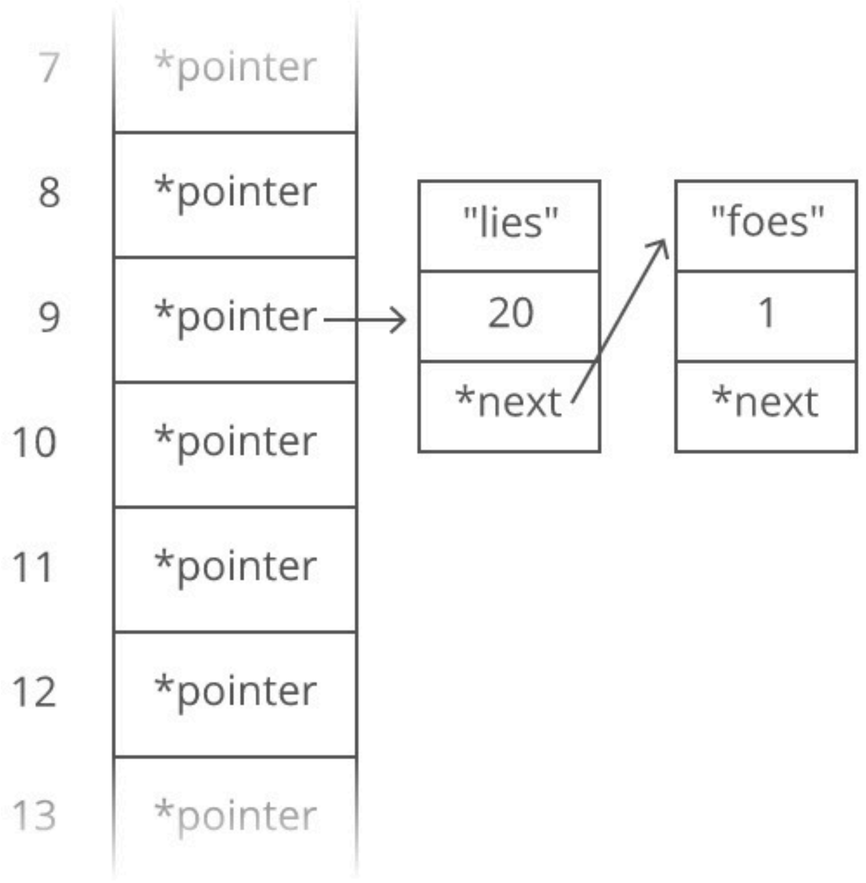
- What if two keys hash to the same index in our array? In our example above, look at "lies" and "foes":



- They both sum up to 429! So of course they'll have the same answer when we mod by 30:
  - 429 % 30 = 9
- This is called a **hash collision**.

Solving Collision

- Instead of storing the actual values in our array, let's have each array slot hold a pointer to a linked list holding the values for all the keys that hash to that index:



Hash Algorithms

- A **hashing algorithm** is a mathematical function that condenses data to a fixed size.
- For example, using “CRC32” we can hash “The Quick Brown Fox Jumps Over The Lazy Dog” to “07606bb6”
- Same input will hash to same output using the same hash algorithm.
- For example. using “MD5” we can hash “cake” to “DF7CE038E2FA96EDF39206F898DF134D” and “cakes” to “0E909116761055 8FDAE6F69BD6716771”.

Cryptographic Hash Algorithms

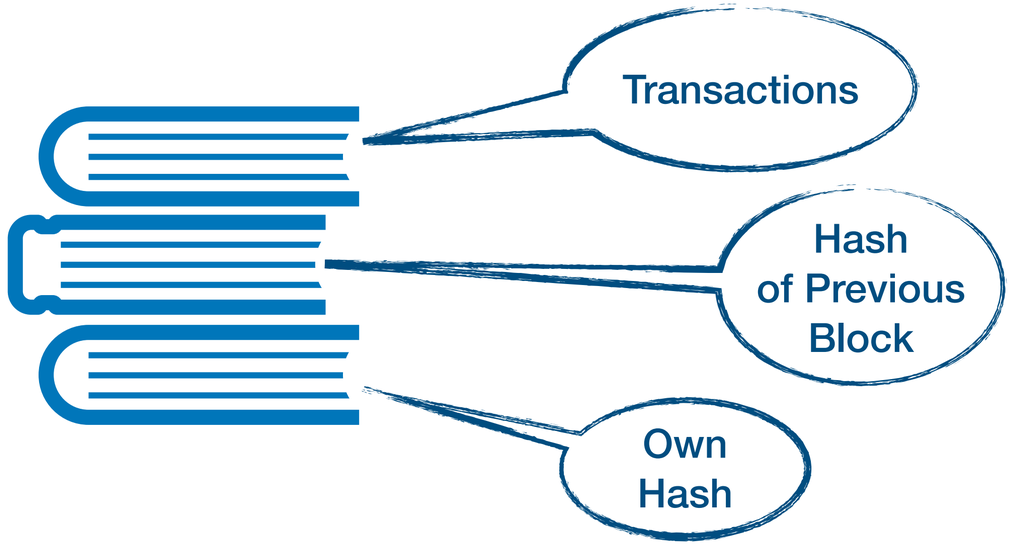
- Hash algorithms used for cryptography purpose follow more requirements:
  - Unique: Two different pieces of data can never produce the same hash
  - Irreversible: If you only had the hash you couldn't use that to figure out what the original piece of data was, therefore allowing the original data to remain secure and unknown.
- Cryptographic hash algorithms are used for digital signatures, secure key exchanges, etc

BlockChain (Simplified)

- Block Chain:
  - Block - Node
  - Chain - List
  - Underlying data structure - Linked List
- The Data part makes BlockChain super powerful
  - Each Node (Block) carries transaction information, user information, self-information etc.
  - Each Node has the hash of previous Node.
  - Each Node has its own hash.
    - Anyone who wants to change any information on one Block, will result the conflict of the next nodes,

Block - Chain

- Blocks carry information about:
  - Transactions like the date, time, and dollar amount of the purchase
  - Who is participating in transactions
  - Information that distinguishes them from other blocks, each block stores a unique code called a “hash” that allows us to tell it apart from every other block.
  - A single block on the blockchain can actually store up to 1 MB of data. Depending on the size of the transactions, that means a single block can house a few thousand transactions under one roof.



BlockChain (More Details)

- Blockchain is a distributed, decentralized, public ledger
  - Each computer in the blockchain network has its own copy of the blockchain, which means that there are thousands, or in the case of Bitcoin, millions of copies of the same blockchain.
  - There isn't a single, definitive account of events that can be manipulated. Instead, a hacker would need to manipulate every copy of the blockchain on the network.
  - Anyone can view the contents of the blockchain.
  - Although transactions on the blockchain are not completely anonymous, personal information about users is limited to their digital signature or username.

Blocks Carry What

- Blocks carry information about:
  - Transactions like the date, time, and dollar amount of the purchase
  - Who is participating in transactions
  - Information that distinguishes them from other blocks, each block stores a unique code called a “hash” that allows us to tell it apart from every other block.
  - A single block on the blockchain can actually store up to 1 MB of data. Depending on the size of the transactions, that means a single block can house a few thousand transactions under one roof.

Block Chain

- Block chain only ADDS new block, never deletes.
- To add a new block to the chain, four things must occur:
  - A transaction must occur.
  - That transaction must be verified.
  - That transaction must be stored in a block.
  - That block must be given a hash.
- Once all of a block's transactions have been verified, it must be hashed. The block is also given the hash of the previous block. Once hashed, the block can be added to the blockchain.

Thank you!