Topics for this week:
- any project related issues
- dictionary data structure
  - Python built-in
  - design our own
- hashing
  - hash function, hash code
  - collision handling
Reminders:
- Part 2 due THIS Wednesday
- every project submission: 10% of your course grade
- questions/problems/issues with your implementation?
Problem: Word Counting

We have a text document and we want to count the number of occurrences for every word.

Why?

for example search engines

Example:

Each player gets dealt six cards. The player pieces start at their home bases and their task is to step on the tiles with all six treasures and then come back to their home bases. Whoever does this first wins.

```
each 1
player 2
the 2
their 3
```
Problem: Word Counting

Example:
Each player gets dealt six cards. The player pieces start at their home bases and their task is to step on the tiles with all six treasures and then come back to their home bases. Whoever does this first wins.

Naive approach:

We want to support the following operation:
- lookup / find / search
  - search for a given word
- insert
  - insert a new word

Problem:
- slow lookup - in the worst case
  when the word is not in the list
  \( \Rightarrow O(n) \) run-time

A possible fix:
- keep the list of words sorted
  \( \Rightarrow \) lookup - fast: binary search
  \( O(\log n) \)

Problem:
- insert - slow, need to move
  \( O(n) \) elements in the worst case

Another possibility:
Sorted linked list of words

Problem: lookup - slow in \( O(n) \)
beac. cannot use bin. search, have to "hop" to find middle
Problem: Word Counting

Example:
Each player gets dealt six cards. The player pieces start at their home bases and their task is to step on the tiles with all six treasures and then come back to their home bases. Whoever does this first wins.

Built-in Python dictionary:
- stores key/value pairs

```python
myWordDict = dict()
myWordDict["hello"] = 5  # means that we are setting the number of occurrences of "hello" to 5

# check if a string is already in:
if "hello" in myWordDict:
    pass

# Pseudo-code:
myWordDict = dict()
for every word in the text document:
    if word in myWordDict:
        myWordDict[word] = myWordDict[word] + 1
    else:
        myWordDict[word] = 1
```
Problem: Word Counting

Dictionary Data Structure - how does it work internally?

- hash function (and hash code):
  
  For example: \( h(\text{"hello"}) = \text{\textasciitilde h} \) - we will want to return a number e.g. \( h(\text{"hello"}) = \text{ASCII of } \text{\textasciitilde h} \)

  Suggestion: look at first letter, then have a linked list of all words starting with the letter

  a hash function: any function that assigns a value to an object, for example \( h \) above assigns values to strings

- hash table/array - what do we store there? (and where?)

  ![Diagram of hash table]

  We store "hello" in the bin specified by \( h(\text{"hello"}) \)

  If \( h(\text{"hello"}) \) is bigger than the size of the table, then take modulo size of the table

  The bin is:

  \( h(\text{"hello"}) \mod \text{size of table} \)
Dictionary Data Structure - how does it work internally?

- supports operations/functions:

  - lookup(key, hashFnc, hashTable):
    - bin = hashFnc(key) % size hashTable
    - traverse through the items (list) in hashTable[bin]:
      - if finds key, return the key (and value if included)
      - return None / not found

  - insert(key, value, hashFnc, HashTable):
    - bin = hashFnc(key) % size hashTable
    - go through the list at hashTable[bin]:
      - if finds key, update the value and return
      - append the key and value pair to hashTable[bin]

- time complexity (amortized)?

  \[ O(1) \] in the best case,
  \[ O(n) \] in the worst case if all elements hashed to the same bin

want: constant time, \( O(1) \) if no or few collisions
Problem: Word Counting

Good hash functions?

How do we deal with collisions?

Good table size?