* Hash tables
  + Python has built in hash table (need to implement our own) - dictionary
  + Collection of slots and each slot has an address
    - Ex:
      * 10, cat
      * 20, dog
      * 15, bird
  + Take the key and apply a hashing function that takes the key and produces an integer from it:
    - Table size: m, number of inserted values: n, load factor: λ = n/m
    - Theoretical function (isn’t actually good one just for teaching) – h(k) = k mod tablesize
    - ex:
    - h(k) = 10 mod 6
      * = 4
    - So store 10 cat in location 4
    - h(k) = 20 mod 6
      * = 2
    - Store 20 dog in 2
    - Take the key value pair you’re looking at take the key and hash it
  + Always get the same output for the same input for a hash function – they also require a constant amount of work to calculate— the amount of work to find the hash value is always the same —- so if you can calculate the hash value in a constant amount of time and locate into the array in a constant amount of time then putting things into the hash table requires a constant amount of work
  + Whole key value pair is stored in the index — Idea is to keep the insert as constant time as possible
  + —ex— numpy array of lists of key value pairs where the value is the list of document names
  + Initializing table size — start with a big table - start with one of like 100k slots
    - Take the table size – then use the load factor to determine if you’ve put too many things in the table – want to keep it at around less than .9 or so
    - If it get’s too big make bigger table - go from 100k t o 1million maybe
  + If chains—(when multiple key value pairs exist for one index) don’t exceed a certain length searching is essentially constant time
* 1