Python's built-in hash table is the dictionary

Hash table is a collection of buckets (slots). Give each slot an index/address
Data set: 10, cat
20, dog
15, bird
Apply hashing function to key and product an integer, usually really long, from it
Ex: h(k) = k mod 6 -> 6 is from the table size

0	
1	
2	(20, dog) -> (2, bird)
3	(15, bird)
4	(10, cat)
5	

Table size: m
Number of inserted values: n
λ load factor: n/m (how many slots taken in table)

h(k) -> constant work for any k, can't depend on size of the table or other factors
-regardless of what inserting and how many values have been inserted as long as inserting into empty slot

Say have 2, bird -> 2 mod 6 = 2. Already something there. Have some options
-Not insert (bad)
-look for next open address
-create list of things that map to location
(create hash map as array, but each location can be list)

Table Size
->ex 100k
-> more buckets, higher probability that buckets have less things in them
-> use load factor to determine when too many things in table, want λ < .9
Maybe 100k -> 1 mil -> 5 mil

Say the longest chain is 5 kv pairs (for 100k shortest AVL tree is log2(100k)
"Essentially" constant time

Want hash table with good dispersion -> values that are spread across output space
Want key value pairs that go across space

Ex (making up hash value):
(finance, 7.json) -> 381
(money, 10.json) -> 767
(bank, 15.json) -> 951
(money, 7.json) -> 767

0	
1	(finance, 7.json) (bank, 15.json)
2	
3	
4	
5	
6	
7	(money, [10.json, 7.json])
8	
9	