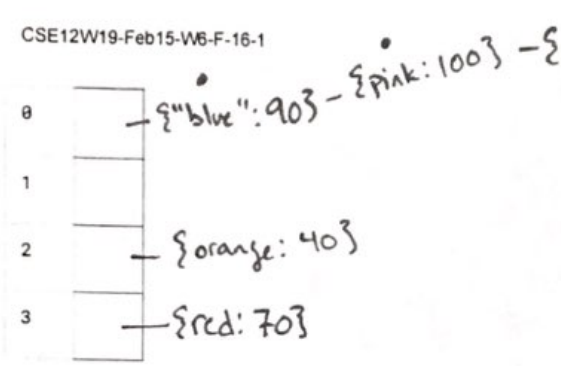
CSE12W19-Feb15-W6-F-16-1



Example:
Start buckets array with size 4
Use string length as the hash function

```
add("red", 70)
add("blue", 90)
add("pink", 100)
add("orange", 40)
add("purplish", 30)
```

→ How many elements in bucket 0?
A: 0 B: 1 C: 2 (D: 3) E: more than 3

→ How many elements in bucket 1?
(A: 0) B: 1 C: 2 D: 3 E: more than 3

→ How many elements in bucket 2?
A: 0 (B: 1) C: 2 D: 3 E: more than 3

→ How many **entries** are checked for get("purplish")?
A: 1 B: 2 C: 2 (D: 3) E: more than 3

A HashTable<Key, Value> using Separate Chaining has:
- size: an int
- buckets: an array of lists of Entries
- hash: a hash function for the Key type

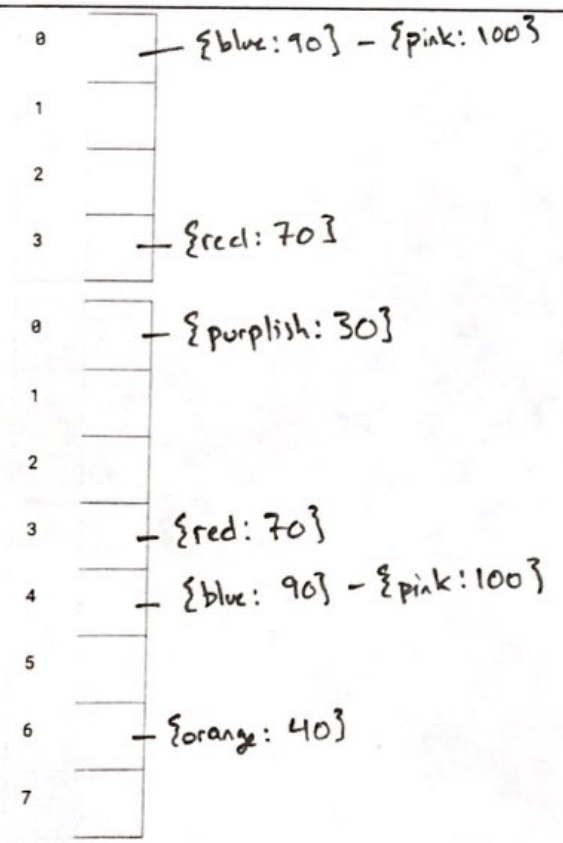
An Entry is a single {key: value} pair.

add
```
void set(key, value):

  hashed = hash(key)
  index = hashed % this.buckets.length
  if this.buckets[index] contains an Entry with key:
    update that Entry to contain value
  else:
    increment size
    bucket = buckets[index]
    add {key: value} to end of bucket


Value get(key):
  hashed = hash(key)
  index = hashed % this.buckets.length
  if this.buckets[index] contains an Entry with key:
    return the value of that entry
  else:
    return null/report an error
```

LoadFactor = $\frac{\#entries}{\#buckets}$ Java: 0.75 Python: 2/3




```
void set(key, value):
  if LoadFactor ≥ 0.75: expandCapacity()

  ... as before ...


void expandCapacity():
  newBuckets = new List[this.buckets.length * 2];
  oldBuckets = this.buckets
  this.buckets = newBuckets
  this.size = 0
  for each list of entries in oldBuckets:
    for each {k: v} in the list:
      this.set(k, v)
```

← first line

Example:
Start buckets array with size 4
Use string length as the hash function

A add("red", 70) 1/4
B add("blue", 90) 1/2
C add("pink", 100) 3/4
D add("orange", 40) 1/2
E add("purplish", 30)

When does expandCapacity happen?

CSE12W19-Feb15-W6-F-16-2

```
int hash1(String s) {
  return s.length();
}


int hash2(String s) {
  int hash = 0;
  for(int i = 0; i < s.length(); i += 1) {
    hash += Character.codePointAt(s, i);
  }
  return hash;
}



public int hash3(String s) {
  int h = 0;
  for (int i = 0; i < s.length(); i++) {
    h = 31 * h + Character.codePointAt(s, i);
  }
  return h;
}
```

4 15 7 = 26 (11 / "dog")

((4 * 31) + 15) * 31 + 7

4 9 13 = 26 (11 / "dim")

((4 * 31) + 9) * 31 + 13

"dog" "log"

"ok" ← hash2 → "ALM"

"unheavenly" "hypoplankton"

All strings (arbitrarily many!): "" ... "a" "b" ... "aa" "ab" ... "zz" ... "aaa" "aab" ... "zzzzzzzz..." ...

Hashed to 2^32 ints: -2147483648 -2147483647 ... -1001 -1000 ... -2 -1 0 1 2 ... 1000 1001 ... 2147483647

distribute evenly

Made into array indices by %: 0 1 2 3 4 5 6 7 8 9 10 ...

- ~~Unique~~ Even distribution
- Fast
- Deterministic