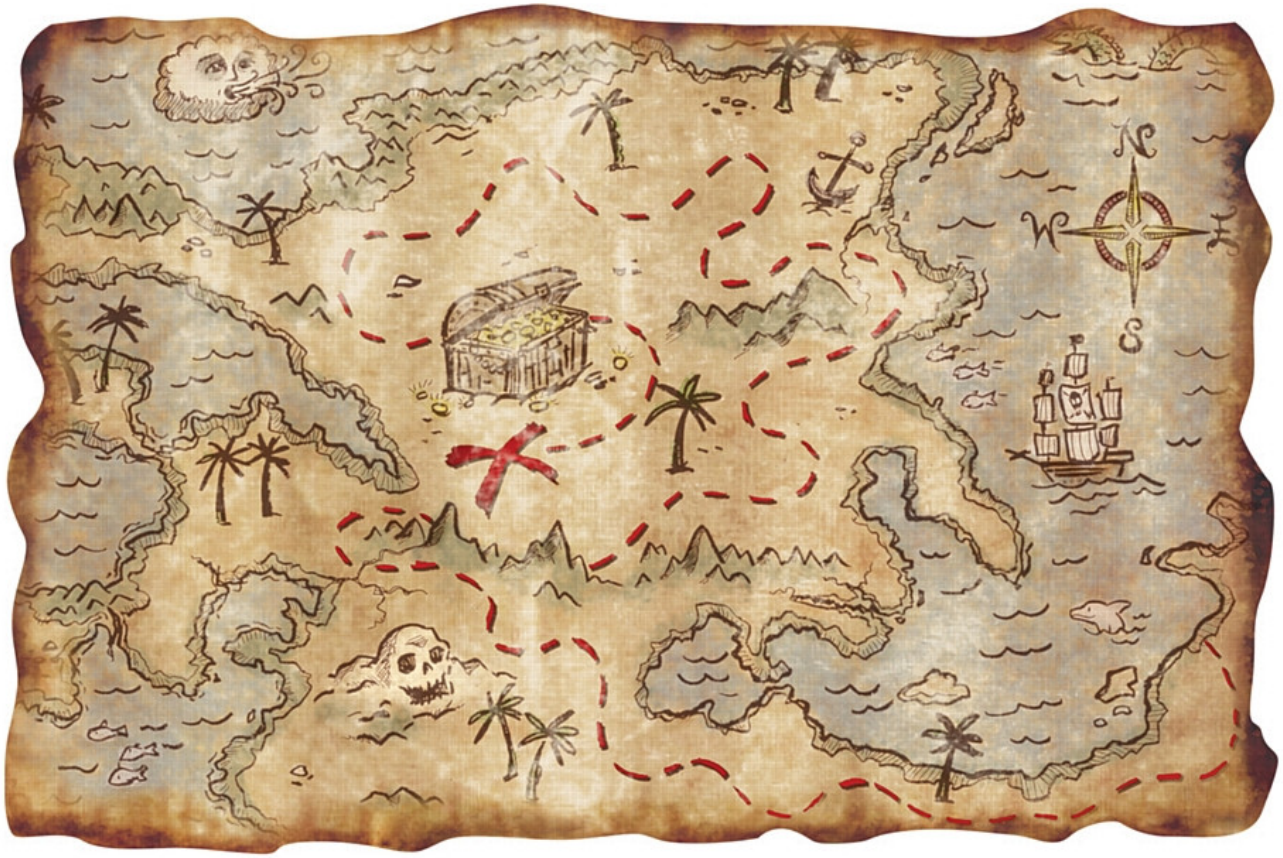
Data Structures

`Map`

(a.k.a. associative array, hash table, dictionary)

I know what a `Map` is!



I DON’T know what a `Map` is!

Remember the `List`?

```
List<String> list = new ArrayList<String>();
list.add(“Amy”);
list.add(“Divya”);
list.add(“Felix”);
list.size(); // 3
```

Remember the `List`?

A `List` is type of **data structure** that we use to hold an *ordered sequence* of stuff (or "elements").

A `List` is also an **object** with **methods** that provide useful ways to do things with the stuff inside.

- `list.add(element)`: adds a new element to the end of the list
- `list.size()`: the number of elements currently in the list
- `list.clear()`: removes all elements from the list
- etc.

What's in a `Map`?

A `Map` is type of **data structure** that we use to hold *key value associations*.

A `Map` is also an **object** with **methods**.

It’s all about your keys and values

What’s a *key value association*?

Some real world examples:

- A word has a definition
- A person has an age
- A city has a zipcode

It's all about your keys and values

In these examples:

- A word has a definition
- A person has an age
- A city has a zipcode

If we have a key, we can get a value.

key ⟶ value

Direction is important!

All hail the `Map`!

A `Map` is type of **data structure** that we use to hold *key value associations*.

A `Map` is also an **object** with **methods**.

- `map.put(key, value)`: associates key with value
- `map.get(key)`: get the value currently associated with key
- `map.size()`: the number of associations currently in the map
- `map.keySet()`: a list* of all keys in the map (in no order)
- `map.values()`: a list* of all values in the map (in no order)

* These technically aren't lists, though you may think of them that way.

No! All hail the `HashMap`!

A `HashMap` is specific kind of `Map`, just like an `ArrayList` is a specific kind of `List`.

Sadly, it has nothing to do with hash browns.

```
List<String> names = new ArrayList<String>();
Map<String, Integer> ages = new HashMap<String, Integer>();
```

Notice that since the `ages` map associates Strings (names) to Integers (ages), we use `<String, Integer>`. I.e. keys are Strings, values are Integers.

How would you create a map to associate words with definitions?

```
Map<String, String> dictionary = new HashMap<String, String>();
```

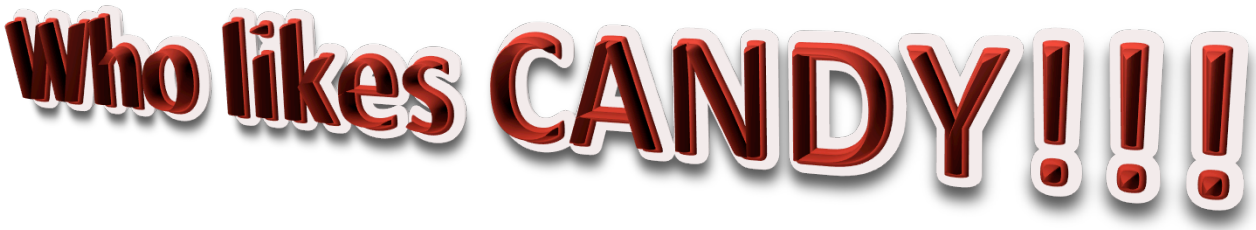
Random fact: some languages call a map a *dictionary*!

The fine print…

```
Map<String, Integer> ages = new HashMap<String, Integer>();
ages.put(“Divya”, 15);
ages.get(“Divya”); // 15

ages.get(“Felix”); // null since “Felix” has no association yet

ages.put(“Felix”, 20);
ages.get(“Felix”); // 20

ages.put(“Felix”, 21); // change “Felix” association from 20 to 21
ages.get(“Felix”); // 21

ages.remove(“Felix”); // remove “Felix” association
ages.get(“Felix”); // null
```

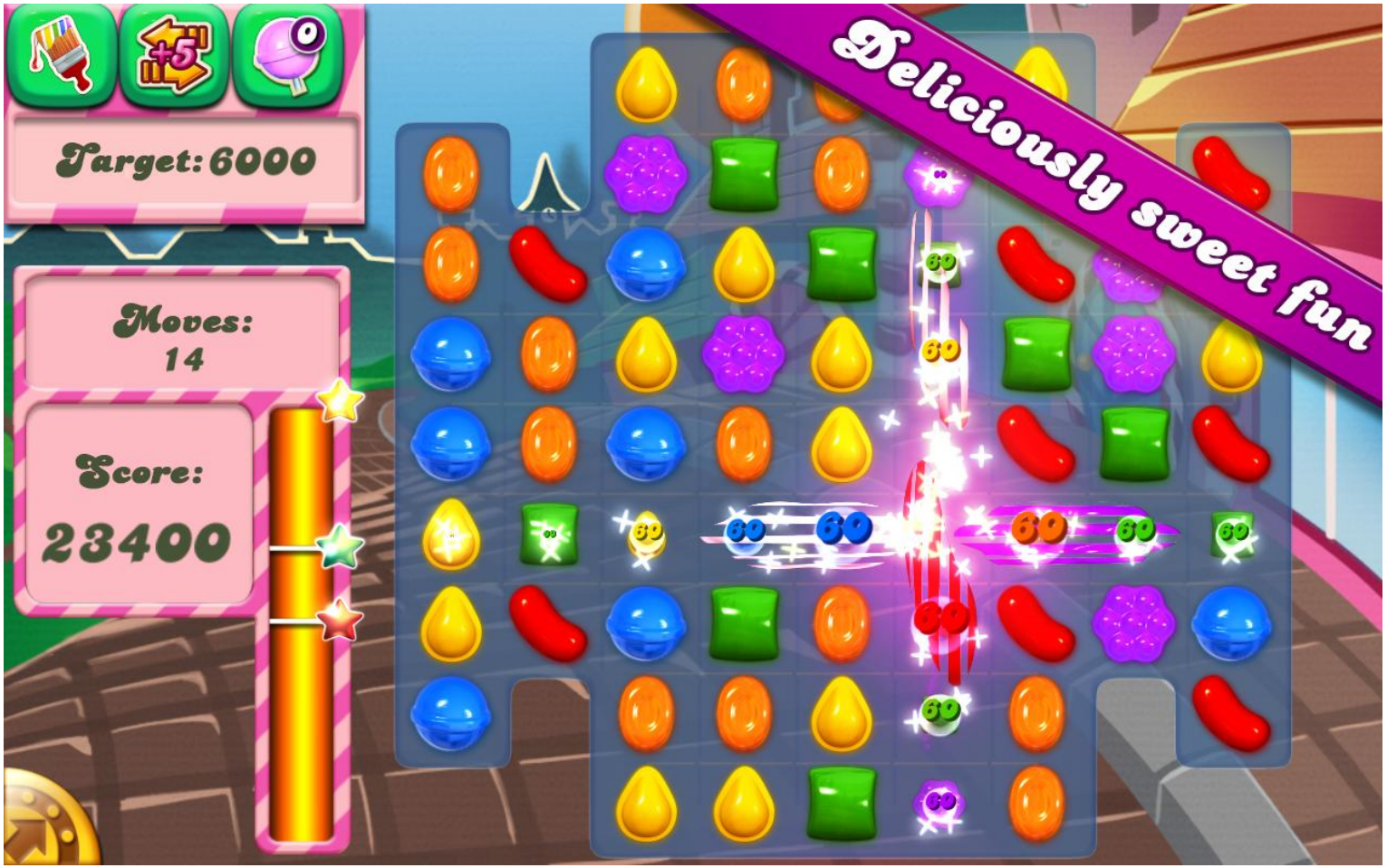
Who cares?

Who needs a `Map` anyway?



But seriously

`Map` is deliciously sweet fun



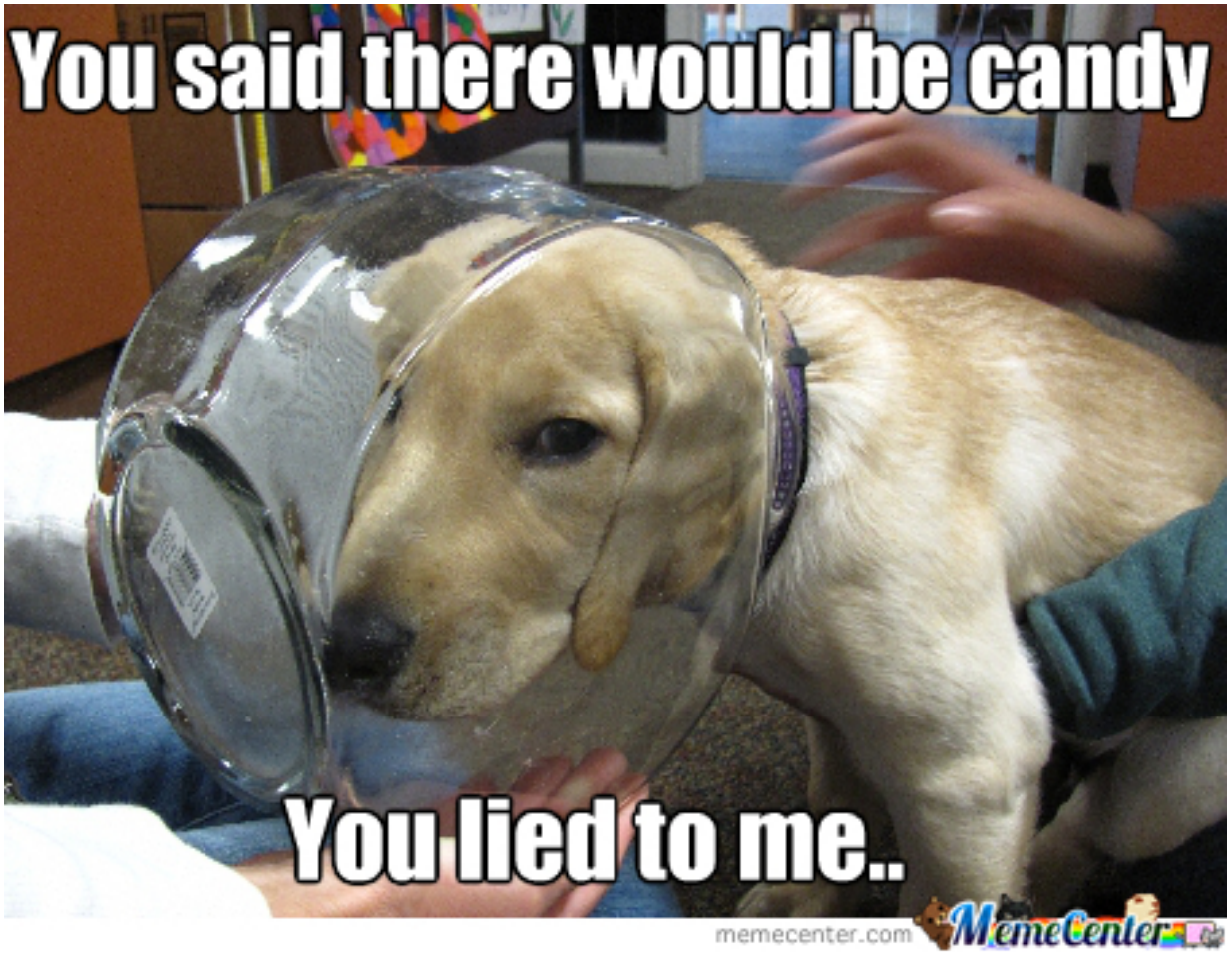
Deliciously sweet fun

```
enum CandyCrushType {
      THREE_IN_A_ROW,
      FOUR_IN_A_ROW,
      FIVE_IN_A_ROW,
      L_SHAPE,
      T_SHAPE
}

/**
 * This function creates a Map that associates a point value with each
 * different kind of crush listed above.
 *
 * We use this map to answer a question like "how many points does a 5-in-a-row score"?
 */
Map<CandyCrushType, Integer> candyCrushPoints() {
      Map<CandyCrushType, Integer> pointMap = new HashMap<CandyCrushType, Integer>();
      pointMap.put(CandyCrushType.THREE_IN_A_ROW, 10);
      pointMap.put(CandyCrushType.FOUR_IN_A_ROW, 20);
      pointMap.put(CandyCrushType.FIVE_IN_A_ROW, 30);
      pointMap.put(CandyCrushType.L_SHAPE, 25);
      pointMap.put(CandyCrushType.T_SHAPE, 25);
      return pointMap;
}
```

Deliciously sweet fun

```
/**
 * Computes the total score received in a game of Candy Crush for a given list of
 * crushes completed during the game.
 */
int computeTotalCandyCrushScore(List<CandyCrushType> crushes) {
     int total = 0;
     for (CandyCrushType crush : crushes) {
          // Get the point value for the current crush and add to total.
          total += candyCrushPoints().get(crush);
     }
     return total;
}
```

Sorry


You said there would be candy
You lied to me..
memecenter.com
MemeCenter

Counting duplicates

Recall

```
/**
 * Counts the number of words which are duplicated in a list.
 */
int countDuplicates(List<String> words) {
    List<String> duplicates = new ArrayList<String>();
    for (int i = 0; i < words.size(); i++) {
        String word = words.get(i);
        if (duplicates.contains(word)) continue; // Already counted.
        // Find words at all *other* indices and look for a duplicate.
        for (int j = 0; j < words.size(); j++) {
            if (i == j) continue; // Skip if we're looking at the same index.
            String otherWord = words.get(j);
            if (word.equals(otherWord)) {
                duplicates.add(word);
                break;
            }
        }
    }
    return duplicates.size();
}
```

Counting duplicates ‘mo ‘betta

```
int countDuplicates(List<String> words) {
     Map<String, Integer> wordCounts = new HashMap<String, Integer>();
     for (String word : words) {
          Integer currCount = wordCounts.get(word);
          if (currCount == null) { // We haven't seen the word before.
               wordCounts.put(word, 1);
          } else {
               wordCounts.put(word, currCount + 1);
          }
     }

     int duplicates = 0;
     for (Integer count : wordCounts.values()) {
          if (count > 1) {
               ++duplicates;
          }
     }
     return duplicates;
}
```

`Map` is 'mo 'betta?

Some situations can benefit from the use of a `Map`, while others benefit from the use of a `List`. Some benefit from both!

So choose the right tool for the job!

But don't forget that the `Map` is one of the most important and widely used data structures in computer science. If you're stuck on a problem it's usually worth your time to think about if a `Map` can help.

Map is your friend

A friend in need (might) need a Map, indeed.