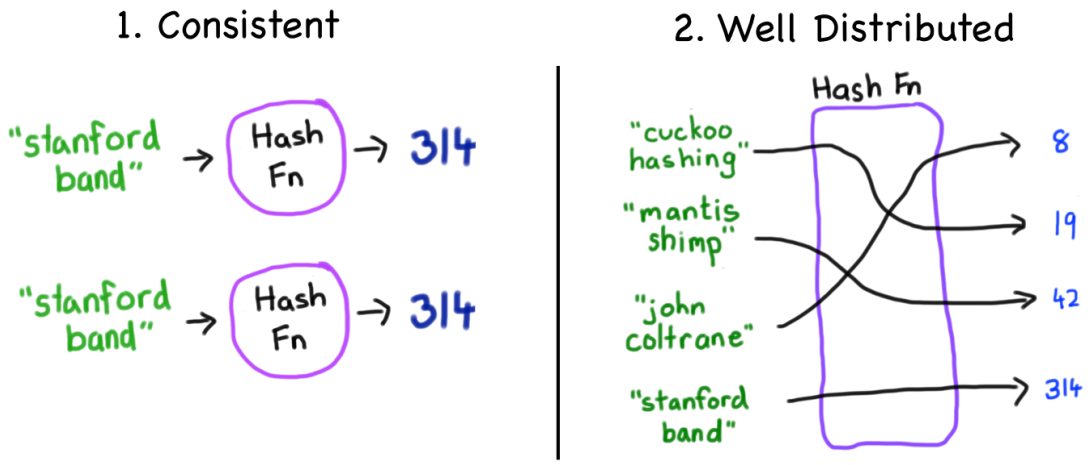
Hashing

At its core a hash function is relatively simple. It's a function which takes in a key and a max value and based on the value of the key, returns a number between 0 and maxValue. All hash functions must be consistent, and we desire that there are well distributed.



Consistent: Equal inputs *always* yield equal outputs.
Few collisions: Unequal inputs yield unequal outputs as often as possible.

For example, if a hash function returned a random number, it would not be consistent because passing in the same key would often result in unequal outputs. Hash functions are one of the great ideas of computer science. And they are used in all disciplines. Some famous examples are: preserving privacy and security online, and as a way to create hash maps that can store large amounts of data with O(1) `get` and `put` complexity.

String Hashes

There are many ways to write a hash function that is consistent and has few collisions. A very common motif is to use the key to generate a preHash number which lies in the range 0 to infinity and then to mod the prehash by the max to project it to the range [0, max). In general we want hash functions to fufill the following two properties:

You could define a hash function for any type of data. Strings hashes are both prolific and enlightening (in that they provide a good base from which to understand other hash functions). The prototype of a string has function is:

```
int hash(string key, int max);
```

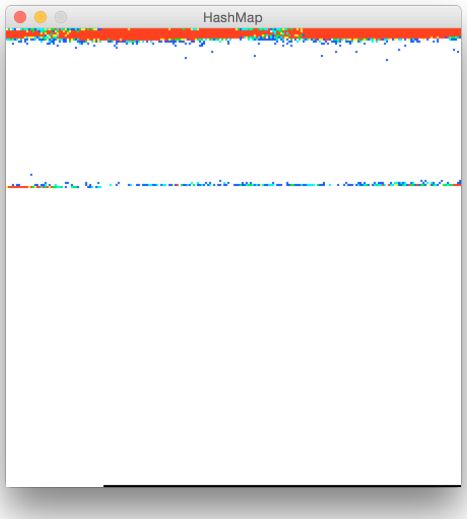
And it should return a value that is in the range [0, max). Sometimes max is defined in a scope visible to the function (and as such does not have to be passed as an argument). A simple consistent hash function is one which returns the sum of all the characters.

```
int hashA(string key, int max) {
   unsigned int preHash = 0;
   for(int i = 0; i < key.length(); i++) {
      preHash += key[i];
   }
   return preHash % max;
}
```

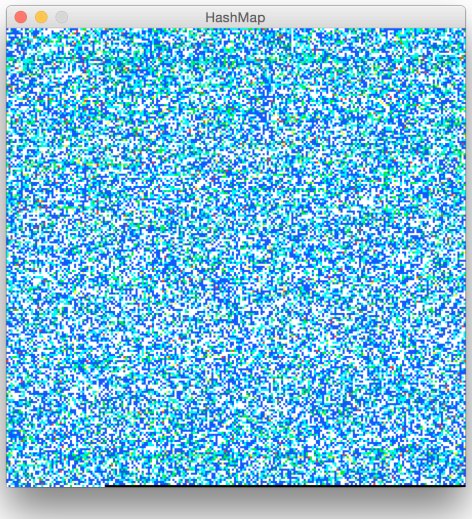
We can do better! The string hash function bellow is the one used by Java. It's by no means perfect, but its great for hash maps. It's very simmilar to `hashA`, but now each char is multiplied by 31^i:

```
int hashB(string key, int max) {
   unsigned int preHash = 0;
   for(int i = 0; i < key.length(); i++) {
      preHash += pow(31, i) * key[i];
   }
   return preHash % max;
}
```

Which one is better? Here is the distribution of bucket use for 50,000 wikipedia articles into 50,000 buckets. Each pixel is a bucket. Red means that there were more than 5 collisions, white means the bucket is not used (blue is 1, cyan 2, green 3 and yellow 4).



(a)



(b)

Clearly the Java hash function is better choice for this dataset! It uses all 50,000 buckets whereas `hashA` (whose name is lose-lose) does not do so well.