Word histogram

To compare different authors, or to identify a good match in a web search, we can use a histogram of a document. It contains all the words used, and for each word how often it was used.

We want to compute a mapping:

$$words \to \mathbb{N}$$

that maps a word w to the number of times it was used.

Map data type

We need a container to store pairs of (word, count), that is `Pair<String, Int>`.

It should support the following operations:

- insert a new pair (given word and count),
- given a word, find the current count,
- update the count for a word,
- enumerate all the pairs in the container.

This data type is called a map (or dictionary).
A map implements a mapping from some key type to some value type.

Creating a map

We can think of a map `Map<K,V>` as a container for `Pair<K,V>` pairs.

```
>>> val m1 = mapOf(Pair("A", 3), Pair("B", 7))
>>> m1
{A=3, B=7}
```

However, Kotlin provides a nicer syntax to express the mapping:

```
>>> 23 to 19
(23, 19)
>>> "CS109" to "Otfried"
(CS109, Otfried)
>>> val m = mapOf("A" to 7, "B" to 13)
>>> m
{A=7, B=13}
```


Querying maps

```
>>> m["A"]
7
>>> m["B"]
13
>>> m["C"]
null
```

Return type is actually `Int?`.

Which means we have to check for `null` before doing anything with the value.

Or use the `getOrElse` method:

```
>>> m.getOrElse("A") { 99 }
7
>>> m.getOrElse("C") { 99 }
99
```

Map methods

Check if key is in map:

```
>>> "A" in m
true
>>> "C" in m
false
>>> "C" !in m
true
```

Size of the map and emptiness:

```
>>> m.size
2
>>> m.isEmpty()
false
>>> m.isNotEmpty()
true
```


Looping over elements of the map

We can use a `for` loop like for lists and arrays, but with two variables:

```
>>> fun printMap(m: Map<String, Int>) {
...   for ( (k,v) in m)
...     println("$k --> $v")
... }
>>> printMap(m)
A --> 7
B --> 13
```


Mutable maps

We can also use mutable maps:

```
>>> val m = mutableMapOf("A" to 7, "B" to 13)
>>> println(m)
{A=7, B=13}
>>> m["C"] = 99
>>> println(m)
{A=7, B=13, C=99}
>>> m.remove("A")
7
>>> println(m)
{B=13, C=99}
>>> m["B"] = 42
>>> println(m)
{B=42, C=99}
```

A useful method: `getOrPut`

```
>>> m.getOrPut("B") { 99 }
42
>>> println(m)
{B=42, C=99}
>>> m.getOrPut("D") { 99 }
99
>>> println(m)
{B=42, C=99, D=99}
```


Word histogram

```
fun histogram(fname: String): Map<String, Int> {
  val file = java.io.File(fname)
  val hist = mutableMapOf<String, Int>()
  file.forEachLine {
    if (it != "") {
      val words = it.split(Regex("[ ,:;.?!<>()-]+")
      for (word in words) {
        if (word == "") continue
        val upword = word.toUpperCase()
        hist[upword] =
                hist.getOrElse(upword) { 0 } + 1
      }
    }
  }
  return hist
}
```

Printing the map

Iterating over the pairs in a map:

```
for ((word, count) in h)
  println("%20s: %d".format(word, count))
```

Words show up in a rather random order. We can fix this by converting the map to a sorted map:

```
val s = h.toSortedMap()
for ((word, count) in s)
  println("%20s: %d".format(word, count))
```

Maps are implemented using a hash table, which allows extremely fast insertion, removal, and search, but does not maintain any ordering on the keys. (Come to CS206 to learn about hash tables.)

Pronounciation dictionary

Let's build a real "dictionary", mapping English words to their pronounciation.

We use data from `cmudict.txt`:

```
## Date:  9-7-94
##
...
ADHERES  AH0 D HH IH1 R Z
ADHERING  AH0 D HH IH1 R IH0 NG
ADHESIVE  AE0 D HH IY1 S IH0 V
ADHESIVE(2)  AH0 D HH IY1 S IH0 V
...
```


Reading the file

Reading the dictionary file:

```
fun readPronounciations(): Map<String,String> {
  val file = java.io.File("cmudict.txt")
  var m = mutableMapOf<String, String>()
  file.forEachLine {
    l ->
      if (l[0].isLetter()) {
        val p = l.trim().split(Regex("\\s+"), 2)
        val word = p[0].toLowerCase()
        if (!("(" in word))
          m[word] = p[1]
      }
  }
  return m
}
```


Finding homophones

English has many words that are homophones: they sound the same, like "be" and "bee", or "sewing" and "sowing".

Create a dictionary mapping pronounciations to words:

```
fun reverseMap(m: Map<String, String>):
      Map<String, Set<String>> {
  var r = mutableMapOf<String,MutableSet<String>>()
  for ((word, pro) in m) {
    val s = r.getOrElse(pro) {
              mutableSetOf<String>() }
    s.add(word)
    r[pro] = s
  }
  return r
}
```

A word puzzle

There are words in English that sound the same if you remove the first letter: 'knight' and 'night' is an example.

```
fun findWords() {
  val m = readPronounciations()
  for ((word, pro) in m) {
    val ord = word.substring(1)
    if (pro == m[ord])
      println(word)
  }
```

Is there a word where you can remove both the first or the second letter, and it will still sound the same?