Abstract Data Type Map

Another fundamental abstract data type is the map (also called dictionary, in particular in Python).
A map implements a mapping from some key type to some value type.

Typical example: Imagine a student database. Each entry represents information about one student, like name, department, birthday, scores, etc.

Each student is identified with a unique student id.

The data base is a map from student ids to student entries.

Other examples: map country code to country name, stock symbol to company name, IP address to country.

Map ADT

The most important map methods are:

- `dict()` Create new map.
- `len(d)` Return number of items in the map.
- `d[k]` Return value of item with key `k`,
  raise error if it does not exist.
- `d.get(k, v0)` Return value of item with key `k`
  if it exists, otherwise return `v0`.
- `d[k] = v` Set value for key `k` to `v`.
- `k in d` Is there an item with key `k`?
- `for k in d:` Iterate over all keys.

You can think of a map as a set of `(key, value)` pairs, with the restriction that any key can appear only one time.

Python dictionaries can be created like this:

```
{ "a" : 13, "b" : 17, "c" : 99 }
```

Transcribing mRNA to proteins

A strand of mRNA encodes a sequence of proteins.

https://en.wikipedia.org/wiki/Genetic_code#RNA_codon_table

```
codon = { "UUU" : "F",  "CUU" : "L",  "AUU" : "I",  "GUU" : "V",
          "UUC" : "F",  "CUC" : "L",  "AUC" : "I",  "GUC" : "V",
          "UUA" : "L",  "CUA" : "L",  "AUA" : "I",  "GUA" : "V",
          "UUG" : "L",  "CUG" : "L",  "AUG" : "M",  "GUG" : "V",
          "UCU" : "S",  "CCU" : "P",  "ACU" : "T",  "GCU" : "A",
          "UCC" : "S",  "CCC" : "P",  "ACC" : "T",  "GCC" : "A",
          "UCA" : "S",  "CCA" : "P",  "ACA" : "T",  "GCA" : "A",
          "UCG" : "S",  "CCG" : "P",  "ACG" : "T",  "GCG" : "A",
          "UAU" : "Y",  "CAU" : "H",  "AAU" : "N",  "GAU" : "D",
          "UAC" : "Y",  "CAC" : "H",  "AAC" : "N",  "GAC" : "D",
          "UAA" : "Stop",  "CAA" : "Q",  "AAA" : "K",  "GAA" : "E",
          "UAG" : "Stop",  "CAG" : "Q",  "AAG" : "K",  "GAG" : "E",
          "UGU" : "C",  "CGU" : "R",  "AGU" : "S",  "GGU" : "G",
          "UGC" : "C",  "CGC" : "R",  "AGC" : "S",  "GGC" : "G",
          "UGA" : "Stop",  "CGA" : "R",  "AGA" : "R",  "GGA" : "G",
          "UGG" : "W",  "CGG" : "R",  "AGG" : "R",  "GGG" : "G" }
```

Calculator with variables

Let’s add variables to our calculator. A variable has a name (an identifier) and a value (a number). The value can be changed.

We need a map from strings to numbers.

```
Welcome to SeoulTech Supercalculator v0.3
Enter an expression: a = 19
a = 19
Enter an expression: 7 * a / 2
==> 66.5
Enter an expression: x = 0.2
x = 0.2
Enter an expression: a * x^3 - 2 * x
==> -0.248
```

Concordance

A concordance lists all the words in a text with the line numbers where it appears.

1: Friends, Romans, countrymen, lend me your ears;
2: I come to bury Caesar, not to praise him.
3: The evil that men do lives after them;
4: The good is oft interred with their bones;
5: So let it be with Caesar. The noble Brutus
6: Hath told you Caesar was ambitious:
7: If it were so, it was a grievous fault,
8: And grievously hath Caesar answer'd it.
9: Here, under leave of Brutus and the rest–
10: For Brutus is an honourable man;
11: So are they all, all honourable men–
12: Come I to speak in Caesar's funeral.
13: He was my friend, faithful and just to me:
14: But Brutus says he was ambitious;
15: And Brutus is an honourable man.
16: He hath brought many captives home to Rome
17: Whose ransoms did the general coffers fill:
18: Did this in Caesar seem ambitious?

```
A        : 7,24
AFTER    : 3
ALL      : 11,11,23,30
AM       : 29
AMBITION     : 20,25
AMBITIOUS   : 6,14,18,21,26
AN       : 10,15,22,27
AND      : 8,9,13,15,22,27
ANSWER'D    : 8
ARE      : 11
....
WHOSE    : 17
WITH     : 4,5,33,34
WITHHOLDS   : 31
WITHOUT  : 30
YET      : 21,26
YOU      : 6,23,30,31
YOUR     : 1
```

Building a concordance

1. Create an empty map.
2. Scan the text word by word. For each word, look it up in the map.
   (a) If it does not yet appear, add it with the current line number.
   (b) If it already appears, add the current line number to its value.
3. Print out the map.

Concordance

```
concordance = dict()
lineNumber = 0

for s in fd.readlines():
  line = s.rstrip()
  lineNumber += 1
  print("%4d: %s" % (lineNumber, line))
  words = line.split()
  for w in words:
    word = w.rstrip(",:;.?!-").upper()
    lns = concordance.get(word, [])
    if lns == [] or lns[-1] != lineNumber:
      lns.append(lineNumber)
    concordance[word] = lns
```

Printing the dictionary

```
for w in concordance:
  lns = concordance[w]
  print("%-10s : %d" % (w, lns[0]), end=’’)
  for ln in lns[1:]:
    print(", %d" % ln, end="")
  print()
```

But keys appear in some “random” order.

Need to extract the keys to a list, sort the list, and then print the concordance:

```
words = list(concordance.keys())
words.sort()
for w in words:
  lns = concordance[w]
  # ...
```

Implementing a Map with a List

Again we implement the map ADT using a Python list to store the data.

```
def __getitem__(self, k):
  i = self._findkey(k)
  if i >= 0:
    return self._data[i][1]
  else:
    raise KeyError(k)

def _findkey(self, k):
  for i in range(len(self._data)):
    if k == self._data[i][0]:
      return i
  return -1
```

Implementing. . . (continued)

```
def __setitem__(self, k, value):
  i = self._findkey(k)
  if i >= 0:
    self._data[i] = (k, value)
  else:
    self._data.append((k, value))

def __contains__(self, k):
  return self._findkey(k) >= 0
```