Laboratory 11

Topics

1. Sets
2. Dictionaries
3. Search in tables

Discussion

A. Discuss common things and differences between:
   a. A list;
   b. A set;
   c. A dictionary.
B. Discuss if a dictionary can have:
   a. Two keys with the same value;
   b. Two values with the same key.
C. If Python didn't offer `set`, but you needed it in a program, what container could you use instead?

Exercises

Part 1 – Complex data structures

***Task**: For each of the following exercises, write a program in Python that meets the given requirements. Complete at least one exercises during the laboratory session, and the remaining ones at home.*

11.1.1 Counting words. Write a program counting all occurrences of a word in a text file whose name is obtained as an input. Assume the file contains only alphabetical characters or white spaces. The program shall output all the words in the file, with the number of occurrences near each word. [P8.2]

11.1.2 Most frequent words. Extend the program of exercise 11.1.1 so that it shows the `5` most frequent words in the file, not including "the", prepositions and conjunctions. [P8.3]

11.1.3 Two strings. Write a program taking as input two strings, then visualizing (with no repetitions):

I. Each character appearing in both strings.
II. Each character appearing in only one of the strings.
III. All the alphabetical letters not appearing in any string.

Hint: use `set()` to turn the string into a set of character. [P8.9]

11.1.4 Censor. Write a 'censor' program, reading a file (`bad_words.txt`) containing a list of bad words (come `'sex'`, `'drugs'`, `'Python'` and so on), a word for each line, reading them into a set. Then read another file (`raw_text.txt`), containing some of the bad words read before. The program shall generate a third file (`censored_text.txt`) containing the previous text, but with all

words and sub-words censored if they contain bad words, using as many asterisks (`'*'`) as the length of the bad word. [P8.14]

11.1.5 Sparse vectors. A sparse vector is a sequence of numbers, most of which are `0s`. A dictionary is an efficient structure to memorize a sparse vector; in such a dictionary, keys are positions in which there are only values different from zero and each value is the number itself. For instance, the sequence `0 0 0 0 0 4 0 0 0 2 9` can be represented by the dictionary `{5:4, 9:2, 10:9}`. Write a function, `sparse_array_sum(a, b)`, whose arguments are two sparse arrays represented as dictionaries, `a` and `b`. Without modifying the parameters, the function shall return their sum vector as a sparse vector, where each value in the i^{th} position is the sum of the values of `a` and `b` in each position `i`. [P8.22]

11.1.6 Sieve of Eratosthenes. The Sieve of Eratosthenes is an iterative algorithm, well known in Ancient Greece, computing all prime numbers before an integer number *n*. In each iteration, it deletes all values multiple of the lowest value present in the set, starting from 2 up to $\sqrt{n}$. After the last iteration, only prime numbers remain. Implement a function using the Sieve of Eratosthenes. First, insert all numbers from `2` to `n`. Then, eliminate all multiples of `2`, except for `2` (as is: `4, 6, 8, 10, 12,` and so on, up to `n`). As a second step, eliminate all multiples of `3`, except for `3` (as is, `6, 9, 12, 15,` and so on, up to `n`). Go on until only prime numbers remain. [P8.4]

Part 2 – Complex operations

***Task**: For each of the following exercises, write a program in Python that meets the given requirements. Complete at least one exercise during the laboratory session, and the remaining ones at home.*

11.2.1 Pro capite income. Write a program reading values from the file `rawdata_2004.txt` inserting them in a dictionary whose keys are the names of Countries, and the values are the annual income pro capite. Please, notice that each field is separated by a tabulation character `'\t'`. Then, the program shall ask the user to input names of Countries to show each value of pro capite yearly income. Program shall terminate when the string `'quit'` is written. Try also working with `rawdata_2021.csv` performing the same exercise. [P8.17]

11.2.2 Genes. In cells, DNA contains fundamental information so that the organism provides all the functions useful to live. To support the cell functions, instructions in DNA are first transcribed into RNA molecules. Later, RNA molecules get translated into proteins. Translation from RNA to messenger RNA (mRNA) to proteins is based on genetic code, which is the set of rules in which a sequence of nucleotides (`'A'`, `'C'`, `'U'` e `'G'`) gets translated in a sequence of amino acids for proteic synthesis. Each amino acid corresponds to one or more sequences of 3 nucleotides, called *codons*. The following table[1] shows the codon that in mRNA code the `20` ordinary amino acids.

1 Codice genetico (Wikipedia): https://it.wikipedia.org/wiki/Codice_genetico

Amino acid	Codons	Amino acid	Codons
Ala	GCU, GCC, GCA, GCG	Leu	UUA, UUG, CUU, CUC, CUA, CUG
Arg	CGU, CGC, CGA, CGG, AGA, AGG	Lys	AAA, AAG
Asn	AAU, AAC	Met	AUG
Asp	GAU, GAC	Phe	UUU, UUC
Cys	UGU, UGC	Pro	CCU, CCC, CCA, CCG
Gln	CAA, CAG	Ser	UCU, UCC, UCA, UCG, AGU, AGC
Glu	GAA, GAG	Thr	ACU, ACC, ACA, ACG
Gly	GGU, GGC, GGA, GGG	Trp	UGG
His	CAU, CAC	Tyr	UAU, UAC
Ile	AUU, AUC, AUA	Val	GUU, GUC, GUA, GUG

Following codons indicate the start and the end of the translation process. Note, start codon also codificates Methionine (`'Met'`).

Instruction	Codons
start	AUG, GUG
stop	UAG, UGA, UAA

Write a program elaborating a sequence of mRNA inserted as an input from the user, showing the sequence of amino acid translated. To simulate the translation problem, use a dictionary `genetic_code`, which, reading all info reported as a table in `codice_genetico.csv`, memorizes codons and amino acids codified by those codons, choosing opportune keys and values. Then, iterate through the sequence inserted by the user until you find a `'start'`. Iterate through the next part memorizing the result of the translation in a list called `protein`. Translation shall stop when the `'stop'` sequence is found. Example: starting from the sequence (`'start'` and `'stop'` are underlined), GUAUGCACGUGACUUUCCUCAUGAGCUGAU, the program shall output: MetHisValThrPheLeuMetSerstop. If no codon of `'start'` and `'stop'` is found, then output an error.

11.2.3 Labyrinth. Write a program reading a text file (`maze.txt`) containing an image of a labyrinth, where asterisks (`'*'`) are walls and spaces (`' '`) are paths.

Create a dictionary `corridor` which uses tuples `(row, column)` as keys, meaning corresponding positions to the corridors and which values are sets of positions corresponding to the corridor and near the positions specified by the key. Example: start key is `(1, 1)`, highlighted in blue, and its adjacent positions are `{(1, 2), (0, 1), (2, 1)}`. In the end, output the dictionary. [P8.20]

11.2.4 Thread of Ariadne. The thread of Ariadne helped Theseus finding the exit from Minosses' labyrinth, tracking the direction towards the exit. Extend exercise 11.2.3 so that, starting from any point, you can find the labyrinth exit, using the following algorithm:

> Starting from `corridors`, build a new dictionary `paths` whose keys are the tuples `(row, column)` of positions initially equal to `'?'`. Then loop through the keys of `paths` and, for

each key representing the edges of the labyrinth (exit), substitute `'?'` with a path leading to the exit: `'N'` (as 'North'), `'E'` (as 'East'), `'S'` (as 'South') o `'W'` (as 'West').
After the substitution, use the dictionary `corridors` to find the adjacent paths, loop through the keys of `paths` with value `'?'`. and, for each of those, verify if they have at least one adjacent position whose value is in `paths` different from `'?'`. If so, replace `'?'` with a string leading to the position of the adjacent cell.
Example: Key `(1, 1)` has, as adjacent cells, `{(1, 2), (0, 1), (2, 1)}`. If key `(0, 1)` contains `'N'`, in `paths`, if in the current iteration key `(1, 1)` is still `'?'`, since there is a value other from `'?'`, will get the value `'N'`, as the adjacent cell is in the 'North' direction.
If you can't substitute any keys in the current iteration, terminate the program.

In the end, show the obtained labyrinth, where each position of the corridor contains the fastest way to the exit. Example:

```
*N*******
*NWW*?*S*
*N*****S*
*N*S*EES*
*N*S***S*
*NWW*EES*
*****N*S*
*EEEEN*S*
*******S*
```

[P8.21]