Assignment #9: Dictionaries

Master in Informatics and Computing Engineering
Programming Fundamentals
Instance: 2018/2019

***Goals**: write programs using dictionaries*

***Pre-requirements (prior knowledge)**: See bibliography of Lecture #14*

***Rules**: you may work with colleagues, however, each student must write and submit in Moodle his or her this assignment separately. Be sure to indicate with whom you have worked. We may run tools to detect plagiarism (e.g.duplicate code submitted)*

***Deadline**: 8:00 Monday of the week after (3/12/2018)*

***Submission**: to submit, first pack your files in a folder RE09, then compress it with zip to a file with name `2018xxxxx.zip` (your_code.zip) and last (before the deadline) go to the Moodle activity (**you have only 2 attempts**)*

1. Hash Collisions

Consider a collection of documents, where each document is associated to a numerical *hash*. When two documents have the same hash, there is an *hash collision*; this needs to be detected and avoided.

Write a Python function called `collisions(list)` that receives a `list` of positive integers and returns a dictionary with the frequency of the hash associated to each element in the `list`. As the hash function, use the modulus-8 of the sum of the digits in the number. For example, if the number is 24 then the hash is `6` (because (2 + 4) % 8 = 6)

Save the program in the file `collisions.py`

For example:

- `collisions([24, 42])` returns the dictionary: `{6: 2}` (because the hash 6 was found twice)
- `collisions([1, 789, 100, 9807, 1208, 92, 101])` returns the dictionary: `{1: 2, 0: 2, 3: 2, 2: 1}`

Note that the order in a dictionary does not matter.

2. Sparse dot product

A sparse vector is a vector whose entries are almost all zero, e.g., [0, 1, 0, 0, 0, 0, 0, 0, 2, 0]. For memory efficiency, sparse vectors can be represented as dictionaries with keys representing the indices of non-zero values, and then the value of a key corresponds to the value of the vector at that index. For example, the vector [0, 1, 0, 0, 2] can be stored as the dictionary {1: 1, 4: 2}.

Write a Python function `sparse_dot_product(dict1, dict2)` that computes the inner product of two sparse vectors, `dict1` and `dict2`, represented as dictionaries.

Save the program in the file `sparse_dot_product.py`

For example:

- `sparse_dot_product({1: 3, 7: 1}, {1: 3, 7: 1})` returns the integer: `10`
- `sparse_dot_product({0: 1, 1: 1}, {2: 1, 3: 1})` returns the integer: `0`

3. Roman to integer

Write a Python function `roman_to_integer(astring)` that, given a *valid* roman numeral as a string, returns its corresponding decimal value. A number in Roman Numerals is a string of Roman Symbols (characters). The mapping between Roman symbols to decimal values is depicted in the following table:

Roman symbol	Decimal value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

Hint: a Roman Numeral is read from left to right and its conversion should be performed as follows: If the value of the current roman symbol is greater than or equal to the value of the next symbol, then add this value to the running total; otherwise subtract this value to the running total.

Save the program in the file `roman_to_integer.py`

For example:

- `roman_to_integer('LXXXIV')` returns the integer: `84`
- `roman_to_integer('XLIII')` returns the integer: `43`
- `roman_to_integer('MMMCMXCIX')` returns the integer: `3999`

4. Isomorphic strings

Given two strings of the same length, `astring1` and `astring2`, write a Python function `isomorphic(astring1, astring2)` that determines if they are isomorphic. Two strings are called isomorphic if the characters in one string can be remapped to get the second string. Remapping a character means replacing all occurrences of it with another character, while preserving the ordering of the characters. No two characters may map to the same character, but a character may map to itself. For example, the strings `"foo"` and `"app"` are isomorphic because it is possible to map `'f'` to `'a'` and `'o'` to `'p'`. The strings `"bar"` and `"foo"` are not isomorphic because it is not possible to map both `'a'` and `'r'` to `'o'`.

The function must return a properly formatted string:
`'<astring1>' and '<astring2>' are isomorphic because we can map: <alist>`
or
`'<astring1>' and '<astring2>' are not isomorphic`

if the strings are isomorphic or not, respectively. `<alist>` represents a list of tuples with all remapped characters between `astring1` and `astring2`.

Save the program in the file `isomorphic.py`

For example:

- `isomorphic('ab', 'aa')` returns the string: `'ab' and 'aa' are not isomorphic`
- `isomorphic('paper', 'title')` returns the string: `'paper' and 'title' are isomorphic because we can map: [('p', 't'), ('a', 'i'), ('e', 'l'), ('r', 'e')]`
- `isomorphic('foo', 'bar')` returns the string: `'bar' and 'foo' are not isomorphic`
- `isomorphic('turtle', 'tletur')` returns the string: `'turtle' and 'tletur' are isomorphic because we can map: [('t', 't'), ('u', 'l'), ('r', 'e'), ('l', 'u'), ('e', 'r')]`

5. Budgeting

Write a Python function **`budget(budget, products, wishlist)`** that, given an integer **`budget`**, **`products`** and purchases that a buyer intends to do (**`wishlist`**), checks if the budget is not exceeded, making the appropriate adjustments, if needed. The argument **`products`** is a dictionary where each key is the product name and the value is the price (per unit); the **`wishlist`** is a dictionary where each key is the product name and the value the quantity desired.

Due to lack of attention, the buyer might have wanted more than he could afford. The function should uncover these cases and calculate what products should have their desired quantity lowered (or even be removed) in order to reduce the total amount to fit the budget. Examine products with lower price first. The function should return the updated wishlist.

Save the program in the file **`budgeting.py`**

For example:

- `budgeting(1000, {'ps4': 350, 'smartphone': 400, 'jacket': 40, 'pc': 600, 'glasses': 75}, {'ps4': 1, 'smartphone': 1, 'pc': 1})` returns the dictionary: `{'smartphone': 1, 'pc': 1}`
- `budgeting(1500, {'xbox': 250, 'smartphone': 500, 'jeans': 50, 'pc': 600, 'glasses': 100}, {'glasses': 3, 'jeans': 2, 'pc': 1, 'xbox':1})` returns the dictionary: `{'glasses': 3, 'jeans': 2, 'pc': 1, 'xbox':1}`
- `budgeting(1200, {'xbox': 250, 'smartphone': 500, 'jeans': 50, 'pc': 600, 'glasses': 100}, {'glasses': 3, 'jeans': 2, 'pc': 1, 'xbox':1})` returns the dictionary: `{'glasses': 3, 'jeans': 1, 'pc': 1, 'xbox':1}`

The end.

FPRO, 2018/19