Dictionaries (dict)

The official python documentation:
https://docs.python.org/3/tutorial/datastructures.html#dictionaries

Defining a dictionary

A dictionary consists of a collection of key-value pairs. Each key-value pair maps the key to the associated value.

A dictionary can be defined by using a comma-separated list of key-value pairs enclosed in braces ({}). A colon (:) separates each key from its associated value.

```
capitals_dict = { 'Bucharest': 'Romania', 'Budapest': 'Hungary',
'Chisinau': 'Moldova'}

print(capitals_dict)
```

Dictionaries have the following characteristics:

- A key can appear in a dictionary only once. Duplicate keys are not allowed.
- The key must be of an immutable type (i.e., it cannot be a list)

Accessing dictionary values

A value is accessed from a dictionary by specifying its corresponding key in square brackets ([]):

```
print(capitals_dict['Bucuresti'])
```

If you try to access a key that is not in the dictionary, Python throws an exception:

```
print(capitals_dict['Madrid'])
>> print(capitals_dict['Madrid'])
>> KeyError: 'Madrid'
```

To update a value assigned to a key, you can just assign a new value to an existing key:

```
print(capitals_dict)# {'Bucharest': 'Romania', 'Budapest':
'Hungary', 'Chisinau': 'Moldova'}

capitals_dict['Bucuresti'] = 'RO'
print(capital_dictionary) # {'Bucharest': 'RO', 'Budapest':
'Hungary', 'Chisinau': 'Moldova'}
```

When traversing dictionaries, the corresponding key and value can be retrieved at the same time using the items() method. This method transforms the dictionary to dict_items (don't confuse dict_items with lists).

```
print(dictionar_capitale.items()) # dict_items([('Bucharest',
'RO'), ('Budapest', 'Hungary'), ('Chisinau', 'Moldova')])
```

To get the list of all keys, use the keys() method. For values, use the values() method:

```
print(dictionar_capitale.keys()) # dict_keys(['Bucharest',
'Budapest', 'Chisinau'])
print(dictionar_capitale.values()) # dict_values(['RO',
'Hungary', 'Moldova'])
```

Traversing dictionaries using the reduce() function:

```
student_grade = { 'Alex': 10, 'Mihai': 9, 'Ioana': 10}

print(student_grade.items()) # dict_items([('Alex', 10),
('Mihai', 9), ('Ioana', 10)])


def function_sum(sum, student):
     name, grade = student # we unpack each tuple received as a
                           # parameter (example: ('Alex', 10))
     return sum + grade

import functools
def student_average(dictionary):
     sum_notes = functools.reduce(function_sum,
                 dictionary.items(), 0)
     return sum_notes / len(dictionary) # the length of a
               # dictionary (the number of elements) is obtained
               # with the help of the len function


print(student_average(student_grade))
```

Traversing dictionaries recursively

For traversing dictionaries recursively, we convert the dictionary received as a parameter into 'dict_items', then we convert 'dict_items' into a list that we will traverse recursively.

```
def recursive_sum(dict_list):
     if len(dict_list) > 0:
          name, note = dict_list[0]
          return note + recursive_sum(dict_list[1:])
     else:
          return 0


def medie_elevi_recursiva(dictionary):
     suma_notes = suma_recursiva(list(dictionar.items())) # the
dictionary received as a parameter is converted to dict_items,
then to list
     return suma_notes/len(dictionary)

print(recursive_student_average(student_grade))
```

Solved exercises with dictionaries

Write a function that takes a list of tuples (string, integer) and creates a dictionary where each string is associated with the sum of all the values it is associated with in the list.

Example:
Input: [("Ana",7), ("Alin",3), ("Ana",9)]
Output: {'Ana': 16, 'Alin': 3}

```
def transform(list, dictionary = {}):
     if (list == []):
          return dictionary
     if(list[0][0] in dictionary):
          dictionary[list[0][0]] = list[0][1] +
               dictionary[list[0][0]]
     else:
          dictionary[list[0][0]] = list[0][1]
     return transform(list[1:],dictionary)

lst = [("Ana",7), ("Alin",3), ("Ana",9)]
print(transform(lst))
```