SOFTWARE 1 PRACTICAL

DICTIONARIES

Week 5 – Practical 4a

All data in Python are objects, meaning they have some behaviour that can be called via a method. For example, we can call the method `upper()` for a string:

```
>>> word = "practical"
>>> word.upper()
"PRACTICAL"
```

To know which methods are available for a specific type we can use the function `help` with the type in parameter. For example, to know about string we can do the following:

```
>>> help(str)
Help on class str in module builtins:

class str(object)
 |  str(object='') -> str
…
```

The string type has many interesting methods, one in particular is the `split()` method. To get the documentation about that specific method you can call the method `help` again, with the type followed by the method's name.

```
>>> help(str.split)
Help on method_descriptor:

split(...)
    S.split(sep=None, maxsplit=-1) -> list of strings

    Return a list of the words in S, using sep as the
    delimiter string.  If maxsplit is given, at most
    maxsplit splits are done. If sep is not specified
    or is None, any whitespace string is a separator and
    empty strings are removed from the result.
>>>
```

We can also search for method specific to list. An interesting one is `count()`:

```
>>> help(list.count)
Help on method_descriptor:

count(...)
    L.count(value) -> integer -- return number of
    occurrences of value
```

To search for the dictionary ‘s methods, do the following:

```
>>> help(dict)
```

It will be very useful to look at the documentation while doing the exercises below.

There are several ways to traverse a dictionary. To do so we use different methods depending on the problem we try to solve.

```
>>> d = {'one':'un','two':'deux'}
>>> for key in d:
      print('the value is', d[key])

the value is un
the value is deux
>>> for val in d.values():
      print('the value is', val)

the value is un
the value is deux
>>> for key, val in d.items():
      print('key:', key)
      print('value:', val)

key: one
value: un
key: two
value: deux
```

The first method iterates through the keys one by one in no specific order. Note, another way to write the loop is `<for key in d.keys():>`. The second method iterates through the values, however you should note that in that loop, you will not be able to access the key associated with a given value. Finally, the third method iterates through the pairs contained in the dictionary.

Exercise 0:

In the interpreter, write assignment statements that create dictionaries for the following sets of data:

a) The months of the year, using numbers from 1 to 12 as keys and month names as values.

b) The roman numbers as keys (M, …, X, V, I) and their Arabic number equivalent (1000, …, 10, 5, 1).

c) The first 7 elements in the periodic table, where keys are chemical symbols ("H", "He", "Li", etc.) and values are the names of the elements.

d) Create an empty dictionary `roman`.

- Write a series of statements to add to the dictionary the following key-value pairing 100,000:T, 1000:M, 500:D, 100:K, L:50, 10:X, 5:V and 1:I.
- Write a statement to modify the value associated with the key 100 to C (instead of K)
- Write a statement to delete the pairing 100,00:T

Exercise 1:

Write a Python function display_dico(dico) that takes a dictionary as parameter and print the content of the dictionary, one paired element per line as follow:

```
Key --> Value
```

For example:

```
>>> display_dico({"un":1, "deux":2, "trois":3})
un --> 1
deux --> 2
trois --> 3
```

Note: if the order in which the mapped pairs of a dictionary appear differs from the one shown in the example, your solution is still valid.

Exercise 2 :

Write a function `concat_dico(dico1, dico2)` that takes two dictionaries as parameters and returns a single dictionary containing the pairs from both dictionaries. An important requirement is that **both** dictionaries are **NOT** modified by the function.

For example:

```
>>> concat_dico ({"one":1, "two":2, "three":3},
                 {"four":4, "five":5})
{"one":1, "two":2, "three":3, "four":4, "five":5}
```

The Advanced bit:

An issue may arise when both dictionaries share a least one common key. Rewrite the function so that the method store the values in a list if `dico1` and `dico2` share a common key. In the example below both dictionaries share the keys `"two"` and `"five"`.

```
>>> concatDico ({"one":1, "two":2, "five":5},
                {"two": "10", "five":"101"})
{"one":1, "two":[2, "10"], "five":[5,"101"]}
```

Exercise 3:

Write a function `map_list(keys, values)` that takes two list of the same length as parameters and returns a dictionary where the keys are the elements from the list `keys` and the values are the elements from the list `values`. The mapping follows the lists indices.

For example:

```
>>> map_list(['un', 'two'], [1,2])
{'un':1, 'two':2}
```

The Advanced bit:

An issue may arise if the list `keys` as duplicate elements as the keys must be unique. Rewrite the function so that the method returns **`None`** and print an error message if `keys` has duplicates. Note that having duplicate values in the `values` list is fine.

Note: This function could be used to map the list of English alphabet characters with the list of their frequencies in the English language.

Exercise 4:

Write a function `reverse_dictionary(dico)` that reverse the mapping between keys and values. The parameter `dico` is a dictionary where the keys and values are all immutable. The function should return a dictionary where the pair `key1:value1` in `dico` becomes the pair `value1:key1`. For example

```
>>> reverse_dictionary({"one":1, "two":2})
{1:"one", 2:"two"}
```

The Advanced bit:

An issue may arise again if the dictionary `dico` as duplicate elements in its values. Rewrite the function so that the method returns **`None`** and print an error message if that is the case.