Assignment 3 – Recursion and Dictionary

Required Files

- assignment_3_template.py
- sent_score.txt
- example1.txt / example2.txt

Restrictions

In this assignment, you are **not permitted** to use advanced Python features, such as map, filter, etc.. Moreover, you are also **not permitted** to import any python library.

You will receive 0 marks for the part (not just only the sub-part, but the entire Part) that violates any one of these restrictions. When in doubt, check with your tutor.

Part 1: Fast Food or Health Food? [15 marks]

Fabulous Hong wants to find out how many ways he can take *n* meals (*n* > 0), if he is only given the choice of fast-food meal or health-food meal. Being a very health-conscious person, he does **NOT** want to take two consecutive fast-food meals. For example, if he has to take 4 meals, he can have 8 different ways of doing it, which are shown below, where 'F' stands for a fast-food meal and 'H' a health-food meal:

FHFH, FHHF, FHHH, HFHF, HFHH, HHFH, HHHF, HHHH

You are to help Fabulous. Write a function **num_of_ways(n)** that asks for *n* (the number of meals, a positive integer), and calculates how many ways Fabulous can take these *n* meals.

Input:

- n (positive integer): The total number of meals

Output:

- Number of possible ways

Constraints:

- The function must be **RECURSIVE**
- You should **NOT** use any loop structures (*for*, *while* or *do-while* loop) in your program.

Examples:

```
>>> num_of_ways(3)
5
>>> num_of_ways(4)
8
>>> num_of_ways(7)
34
>>> num_of_ways(10)
144
```

Useful tips:

1. There are 2 base cases here, one is *n* = 1, and the other is *n* = 2. When *n* = 1, the function should return 2 since there are 2 ways ('F' or 'H') to take one meal.
2. For the general case, you need to find out how many ways to take *n* meals. You can take this approach. If the first meal is a fast-food meal, how many ways are there to take the rest of the meals? On the other hand, if the first meal is a health-food meal, how many ways are there to take the rest of the meals? You then combine the answers.

Coursemology Notes:

- You may assume that the input given is valid (n > 0) and need not consider invalid cases within your function
- You may assume that the input given will not exceed 30.

Part 2: SEND MORE MONEY! [60 marks]

You received the following cryptic message (which you have identified that it is not a scam) from your secret agent partner, requesting you to send over some amount indicated by the following sum:

SEND + MORE = MONEY

Here, it is understood that each letter represents a **unique** digit (0 to 9) and no two letters share the same digit.

For instance, consider the following mapping:

S	9
E	5
N	6
D	7
M	1
O	0
R	8
Y	2

Using this mapping, SEND represents 9567, MORE represents 1085, and MONEY represents 10652. This is correct because 9567 + 1085 = 10652.

In this task, your final goal is to determine what digit each letter represents, and hence the amount of MONEY that your partner is requesting. To help you out with this problem, it will be split into various parts, and you will get to consolidate these into a functional algorithm at the end.

Part 2A: Does it work? [15 marks]

First, we will try to find if a given mapping is valid. There are 2 criteria that determines if a mapping is valid.

1. The first digit of a word cannot be 0
2. The addition result must be correct

Write a function **check(puzzle, mapping)** that takes in the puzzle as a tuple and the mapping as a dictionary and returns the dictionary if it is a valid mapping, and False otherwise

Input:

- puzzle (of type tuple). Contains a tuple of at least 2 elements. For a n-sized tuple, the first (n-1) elements are words that are to be added together, and the n-th element is the result that it should add up to
- mapping (of type dictionary). Contains a dictionary where the key is the alphabet, and the value is the corresponding integer.

Output:

- If it is valid, return the dictionary (i.e., the mapping)

- Else, return False

Examples:

```
>>> check(("SEND", "MORE", "MONEY"), {'S': 9, 'E': 5, 'N': 6, 'D': 7, 'M': 1, 'O': 0, 'R': 8, 'Y': 2})
{'S': 9, 'E': 5, 'N': 6, 'D': 7, 'M': 1, 'O': 0, 'R': 8, 'Y': 2}
>>> check(("SEND", "MORE", "MONEY"), {'S': 9, 'E': 3, 'N': 5, 'D': 8, 'M': 1, 'O': 6, 'R': 2, 'Y': 4})
False
>>> check(("ANG", "KU", "KUI"), {'A': 2, 'N': 7, 'G': 9, 'K': 3, 'U': 1, 'I': 0})
{'A': 2, 'N': 7, 'G': 9, 'K': 3, 'U': 1, 'I': 0}
>>> check(("ADD", "EVEN", "MORE", "WORDS"), {'A': 5, 'D': 2, 'E': 9, 'V': 6, 'N': 3, 'M': 7, 'O': 8, 'R': 0, 'W': 1, 'S': 4})
{'A': 5, 'D': 2, 'E': 9, 'V': 6, 'N': 3, 'M': 7, 'O': 8, 'R': 0, 'W': 1, 'S': 4}
```

Coursemology Notes: You may assume that …

- The messages are always in upper case.
- The mapping given does not contain duplicate values (for instance, there will not be both “S”: 5 and “D”: 5 in the mapping). (Note for later: In our final part, we must ensure that we don’t generate such a mapping since the numbers are supposed to be unique)
- The mapping given will have the mapping of every letter required (there will not be a case where the puzzle has an alphabet “E” but the mapping provided does not have an entry for “E”)
- The given mapping will not contain letters that are not within the puzzle (if the puzzle is “A” + “B” = “C”, we won’t provide you a mapping where it contains “D”)
- Ordering of the output does not matter (note: dictionaries are unordered data structures)

Part 2B: Unique Letters [10 marks]

Now, let us try to tackle the puzzle. We will first need to find out what are the alphabets that are present within the puzzle.

Write a function **unique_letters(puzzle)** that takes in the puzzle as a tuple and returns a tuple of letters present within the puzzle. The output should not contain any duplicated alphabets.

Input:

- puzzle (of type tuple). Contains a tuple of at least 2 elements. For a n-sized tuple, the first (n-1) elements are words that are to be added together, and the n-th element is the result that it should add up to

Output:

- A tuple of the unique elements within the puzzle. The ordering of the output does not matter

Examples:

```
>>> unique_letters(("SEND", "MORE", "MONEY"))
('D', 'Y', 'S', 'N', 'R', 'E', 'M', 'O')
>>> unique_letters(("ANG", "KU", "KUI"))
('K', 'G', 'U', 'N', 'I', 'A')
>>> unique_letters(("ADD", "EVEN", "MORE", "WORDS"))
('D', 'S', 'V', 'N', 'W', 'O', 'R', 'E', 'M', 'A')
```

Coursemology Notes:

- Ordering of the output does not matter. We automatically sort your answers in Coursemology

Part 2C: Exploring the possibilities [25 marks]

Let us try to find a possible mapping for the puzzle. This may seem hard, but we will try to guide you through the thought process so don't worry!

Given a sequence of alphabets within the puzzle and a sequence of numbers which we could match the alphabets to, write a function **assign(letters, numbers_left, mapping, puzzle)** that will determine a possible mapping that satisfies the puzzle, if it exists.

To better illustrate what each parameter is, below are a few examples explained:

Example 1:

assign(("B", "C"), (5, 6, 8), {"A": 2}, ("A", "B", "C"))

This means that for the puzzle **A + B = C**, which I have decided that **A = 2**, I still need to assign some numbers to **"B" and "C"** to satisfy the puzzle, where they could either take up the values **5, 6 or 8**

In this case, the algorithm will determine correctly that B can be 6, C can be 8 and hence return {"A": 2, "B": 6, "C": 8}

Example 2:

assign(("B", "C"), (2, 4, 6, 8), {"A": 1}, ("A", "B", "C"))

This means that for the puzzle **A + B = C**, which I have decided that **A = 1**, I still need to allocate **"B" and "C"**, where they could either take up the values **2, 4, 6 or 8**

In this case, there is no possible way of doing it, and hence the algorithm will return False

Input:

- letters (tuple of characters) – the letters that have yet to be allocated a mapping
- numbers_left (tuple of integers) – the numbers that can be mapped to the letters
- mapping (dictionary) – the current mapping that has been achieved
- puzzle (tuple) – the addition puzzle as described in earlier parts

Output:

- If the puzzle is solvable, return the full mapping of all the letters in the puzzle
- If it is not solvable, return False

Code:

- Most of the code has been provided, you just need to understand the logic flow, and fill in the missing pieces of the code to get it work.
- This is a permutation-related problem. Given *n* number of letters and *m* number of digits, you can systematically try out assigning one letter to a specific number (as done in the code), and imagining that the remaining (*m-1*) numbers will be assigned to other (*n-1*) letters. In this way, the problem gets smaller (from *n* letters to *n-1* letters). If that solution doesn't work out, you will need to start out with a different number to the initial one letter instead, and try out again.
- Since this requires recursion, think about what the base cases are, and match your thoughts with the code provided.
- Tuple data structure is used instead of other structures, such as list, because tuples are immutable, and this can free the programmers from thinking about where and how to do – and undo – some of the updates of the data structures. The execution time for tuple version will be slightly longer, but the code will be easier to understand and verified truth.
- You are encouraged to try out other ways of solving the problems, and you are free to ignore the provided code and write you own version of the code. You are free not to write your solution using recursion, but make sure you attempt to solve it using recursion before venturing into other solutions.

Note:

- For this part, the correct implementation of "check" (Part 2A) has been provided. You may use it to help define this function

Examples:

```
>>> assign(("R", "Y"), (2, 3, 4, 8), {'S': 9, 'E': 5, 'N': 6, 'D': 7, 'M': 1, 'O': 0}, ("SEND", "MORE", "MONEY"))
{'S': 9, 'E': 5, 'N': 6, 'D': 7, 'M': 1, 'O': 0, 'R': 8, 'Y': 2}
>>> assign(("A"), (0, 1, 2, 4, 5, 6, 7, 8), {"B": 3, "C" : 9}, ("A", "B", "C"))
{'B': 3, 'C': 9, 'A': 6}
>>> assign(("A"), (0, 1, 2, 4, 5, 7, 8, 9), {"B": 3, "C": 6}, ("A", "B", "C"))
False
>>> assign(("A"), (), {"B": 3, "C": 6}, ("A", "B", "C"))
False
>>> assign((), (0, 1, 2, 4, 6, 7, 9), {"A": 5, "B": 3, "C": 8}, ("A", "B", "C"))
{'A': 5, 'B': 3, 'C': 8}
False
```

Coursemology Note:

- Ordering of the output does not matter (note: dictionaries are unordered data structures)

Part 2D: Putting it together [10 marks]

Nice! Now that we have all the functions required, we can solve the cryptic message we received.

Given the 3 functions defined earlier, write a function **solve(puzzle)** that returns the corresponding mapping. If the puzzle is not solvable, return False

Input:

- puzzle (tuple) – the addition puzzle as described in earlier parts

Output:

- If the puzzle is solvable, return the final mapping
- If it is not solvable, return False

Examples:

```
>>> solve(("SEND", "MORE", "MONEY"))
{'D': 7, 'S': 9, 'Y': 2, 'N': 6, 'M': 1, 'R': 8, 'E': 5, 'O': 0}
>>> solve((“ANG”, “KU”, “KUI”))                    # There are multiple correct answers
{'U': 1, 'A': 2, 'N': 7, 'I': 0, 'G': 9, 'K': 3}    # Any of them will be accepted
>>> solve(("YOU", "ARE", "COOL"))
{'A': 2, 'C': 1, 'O': 0, 'E': 5, 'U': 8, 'R': 9, 'L': 3, 'Y': 7}    # There are also multiple possibilities
>>> solve(("I", "CANT", "DO", "IT"))
False
```

Coursemology Notes:

- There may be multiple possibilities. Any of them will be accepted. The ans_check function is for us to verify your answer and to allow flexibility in your answer. You do not need to implement the function (nor do you have access to this function).

Useful tips:

1. Try out your own (simpler) examples which you will easily be able to verify manually

Part 3: Should I watch it? [25 marks]

Note: This is just a simplified method of doing textual analysis, but it is a good exposure to see how we could use code to help us determine certain sentiments.

Even though the movie has been released quite a while ago, your younger cousin has been pestering you to watch Frozen II with her. You do not really enjoy watching movies but maybe if the movie reviews are good enough…

In this part, you will be given a few movie reviews to analyse. A file, sent_scores.txt, has been loaded in for you where each word is assigned a certain score (>0 means it's a positive word while <0 means it's a negative word).

Your task is to write a function **score(file)** that takes in the file path of a movie review and output the corresponding sentiment score and most impactful word within the review.

Input:

- file – This is the .txt file that you want to analyse. Ensure that the txt file is in the same folder as your python file

Output:

- A 2-element tuple. The first element is the sum of the sentiment scores, rounded to 2 decimal places. The second element is the word that has the highest impact within the article.

To illustrate how these 2 elements are calculated, refer to the example below:

In the template, we have read in the file sent_scores.txt as a variable *word_scores*. This contains a list of lists in a format like this **[['vile', '-3.18'], ['tragic', '-3.15'], ['ugly', '-3.10'], ['pastel', '0.47']]**. This means "vile" is associated with a -3.18 score, "tragic" is -3.10, "pastel" is 0.47, etc.

We have also provided a function to read in the review and store each word as an element within a list. For instance, this review

> *The show was magnificent and I am so lucky to have got to see how multicultural our society could be. Although random at some points, this magnificent movie is a must-watch*

Is converted to **['The', 'show', 'was', 'magnificent', 'and', 'I', 'am', 'so', 'lucky', 'to', 'have', 'got', 'to', 'see', 'how', 'multicultural', 'our', 'society', 'could', 'be', 'Although', 'random', 'at', 'some', 'points', 'this', 'magnificent', 'movie', 'is', 'a', 'must-watch']**

To determine the sentiment score, we match each word to its corresponding score, if it exists. In this example, magnificent = 1.02, lucky = 1.11, multicultural = 0.94, random = -1.84. However, since magnificent appeared twice, the total score is **1.02 * 2** + 1.11 + 0.94 – 1.84 = **2.25**

The most impactful word is "magnificent", since it contributed **2.04** to this score, followed by "random", which contributed (-)1.84, then "lucky", which contributed 1.11. Note that we use the **highest absolute values** to compare which is the most impactful word.

Hence, the function should return (2.25, “magnificent”)

Examples:

```
>>> score("example1.txt")
(2.25, 'magnificent')
>>> score("example2.txt")
(-0.27, 'valid')                    # Note that even though the total score is negative,
                                    # “valid”, a positive word, had the highest absolute value
```

Coursemology Notes:

- Only copy-paste the “score” function into the answer box
- You may assume there will not be any tie among the most impactful words

Useful tips:

1. Look at the example1.txt / example2.txt and ensure you understand why the outputs are as such. Creating your own short examples can also help tremendously
2. Use the in-built “round” function to round numbers to 2 decimal places. You may search online to see its usage
3. It may be useful to convert the word_scores into a more accessible format so that you can retrieve the scores easily (think about what this assignment is about!)
4. You will eventually need some way to find out what the most impactful word is so consider how you can keep track of that

So, do you want to watch the show? 😉