SOFTWARE 1 PRACTICAL

Recursion

Week 7 – Practical 11

Write all your code for questions 1 to 6 in a file `recursion.py`. I have also provided 5 more files called:

- `test_exercise_1.py`,
- `test_exercise_2.py`,
- `test_exercise_3.py`,
- `test_exercise_5.py`,
- and `test_exercise_6.py`.

The files contain unit tests like the ones you will have during the formative and the summer exam. Unit tests are series of tests to validate individual function, or unit of code (hence the name). There are here to help you monitor your progress during the exam (in real life during code production). If you pass a test it means that you have successfully implemented that functionality. For your solution to an exercise to be correct, you must pass ALL the test for that exercise, not just one or two.

Note: Unit testing and creating test files like the one provided are not part of the learning objectives of this module. You do not need to understand how they are done; they are just here as a tool to help you during the examination. You can choose to ignore them completely and not use them during the exam, just be aware that students who used them last year find them very useful.

How to run the test?

First make sure that the test files are in the same directory as `recursion.py`. Then open the test file with Idle, and click run on the menu. If you would like to run the unit test within Visual Studio Code, there is a nice tutorial on the real python channel. When the code does not pass the test, you will still have to look at the output, which is the same as the one provided here for Idle.

If you have the following error message:

```
Traceback (most recent call last):
  File "D:/SOF1/test_exercise_1.py", line 8, in <module>
    from recursion import is_power
ModuleNotFoundError: No module named 'recursion'
```

It means the test file cannot find the `recursion.py` file. Two possibilities, you misspelled the name of the file, so rename it (you will lose marks during the exam), or the file is not in the same directory (again you will lose marks during an exam).

If you have the following error message when testing exercise 1:

```
Traceback (most recent call last):
  File "D:/SOF1/test_exercise_1.py", line 8, in <module>
    from recursion import is_power
ImportError: cannot import name 'is_power'
```

It means you misspelled the name of the function `is_power()` in your solution, rename the function. Guess what! You lose marks again. Read carefully the question during an exam. Why would I lose marks if the code is correct, but the name of the function is different? The reasoning behind the marking scheme is that we always assume you are working within a development team. The software designer gave you a well define brief and all other developers in the team expect you to follow it. Their part of the code may depend on yours and their code will be using your function with the requirements given by the designer. Once the team has finished its first sprint, they will integrate all the existing code, including yours. That is when trouble will come to light, as your function does not follow the brief you were given, and when the other developers' code call your function they will get an error message as it will not be defined. **You must always respect the brief and requiremenst given to you, even if it does not seem to be a sensible one!** If you do not agree with it, you should discuss this with the designer.

If you have the following output when testing exercise 1:

```
RESTART: D:/TPOP 2017-18/test_exercise_1.py
.....
----------------------------------------------------------
------------
Ran 5 tests in 0.047s

OK
```

Congratulations, you passed all the tests for that exercise. You can move on to the next one.

If you have the following output when testing exercise 1:

```
RESTART: D:/TPOP 2017-18/test_exercise_1.py
.F.F.
==========================================================
============
FAIL: testPowerOfItself (__main__.TestExercise1)
----------------------------------------------------------
------------
Traceback (most recent call last):
  File "D:/lilian/Google Drive/Teaching/Teaching 2017-
18/TPOP 2017-18/test_exercise_1.py", line 27, in
testPowerOfItself
    self.assertEqual(True, is_power(3,3), "is_power(3,3)
should be true")
AssertionError: True != False : is_power(3,3) should be
true

==========================================================
============
FAIL: testPowerTrue (__main__.TestExercise1)
----------------------------------------------------------
------------
Traceback (most recent call last):
  File "D:/lilian/Google Drive/Teaching/Teaching 2017-
18/TPOP 2017-18/test_exercise_1.py", line 13, in
testPowerTrue
    self.assertEqual(True, is_power(16,2), "is_power(16,2)
should be true")
AssertionError: True != False : is_power(16,2) should be
true

----------------------------------------------------------
------------
Ran 5 tests in 0.031s

FAILED (failures=2)
```

You are not done yet. You failed 2 out of 5 tests. The lines highlighted in yellow tell you how many tests have been ran, and how many have failed. The lines highlighted in green, tell you which test have failed. And more importantly, the lines highlighted in blue tell you what was expected, and in darker blue what your solution has returned. Change your solution and try again.

If you have the following output when testing exercise 1:

```
=================================================================
=============
ERROR: testPowerOfItself (__main__.TestExercise1)
-----------------------------------------------------------------
-------------
Traceback (most recent call last):
  File "D:/lilian/Google Drive/Teaching/Teaching 2017-
18/TPOP 2017-18/test_exercise_1.py", line 27, in
testPowerOfItself
    self.assertEqual(True, is_power(3,3), "is_power(3,3)
should be true")
TypeError: is_power() missing 1 required positional
argument: 'c'
```

When you had FAIL in the previous paragraph, it meant that the returned value was not what was expected. When you have an ERROR like above, it means you have an error in your code that makes the test crashed, not failed. In the example above, it is a TypeError due to too many parameters in my function declaration (used three instead of the two requested). You need to sort out the error before you can run the test again.

Good Luck!

Exercise 1:

A number, **a**, is a power of **b** if it is divisible by **b** and **a/b** is a power of **b**. Write a **recursive** function called `is_power(a,b)` that takes parameters **a** and **b** and returns True if **a** is a power of **b**, False otherwise.

For example:

- 27 is power of 3 if 27/3 = 9 is power of 3, that is 9/3 = 3 is power of 3, which is true.
- 18 is power of 3 if 18/3 = 6 is power of 3, that is if 6/3 = 2 is power of 3 which is false.

Could you also write an iterative version of this function?

Exercise 2: *(from practical 04)*

During the practical 04, we implemented the **function** `sum_digits(number)` to calculate and return the sum of digits of a given whole number (int) given as parameter. For example,

```
>>> print(sum_digits(1234))
10
```

At the time we used loops in our implementation. This time you **must** use **recursion**.

Exercise 3:

To compute the sum of all elements in a list, you can use the built-in function sum.

For example:

```
>>> sum([1,2,3,4])
10
>>> sum([])
0
```

Write a recursive function `rec_sum(numbers)` that take a list of integers as a parameter and returns the sum of all the elements in the list. The function should return 0 if the list is empty.

Exercise 4:

A word is considered **elfish** if it contains the letters: **e**, **l**, and **f** in it, in any order. For example, we would say that the following words are elfish: *whiteleaf*, *tasteful*, *unfriendly*, and *waffles*, because they each contain those letters.

a) Write a predicate function called iselfish(word) that, given a word, tells us if that word is elfish or not. The function must be **recursive**.

b) Write `something_ish(pattern, word)` a more generalized predicate function that, given two words, returns true if all the letters of `pattern` are contained in `word`.

I did not provide a unit test for this exercise, if you wish you could try to create a unit test for that exercise, and share it with someone else to test their code.

Exercise 5:

Write a recursive function `flatten(mlist)` where `mlist` is a multidimensional list that returns all the element from `mlist` into a one-dimensional list. Note, empty lists are ignored. For examples:

```
>>> flatten([1,[2,3],4])
[1,2,3,4]
>>> flatten([1,[2,[3,[4]]]])
[1,2,3,4]
>>> flatten([1,2,3,4])
[1,2,3,4]
>>> flatten([1,[]])
[1]
```

Exercise 6:

Write a recursive function `merge(sorted_listA, sorted_listB)` that merges the two lists into a single sorted list and returns it. The two parameters are list of comparable objects that are sorted in ascending order. For example, the lists contain only strings, or the lists contain only numbers.